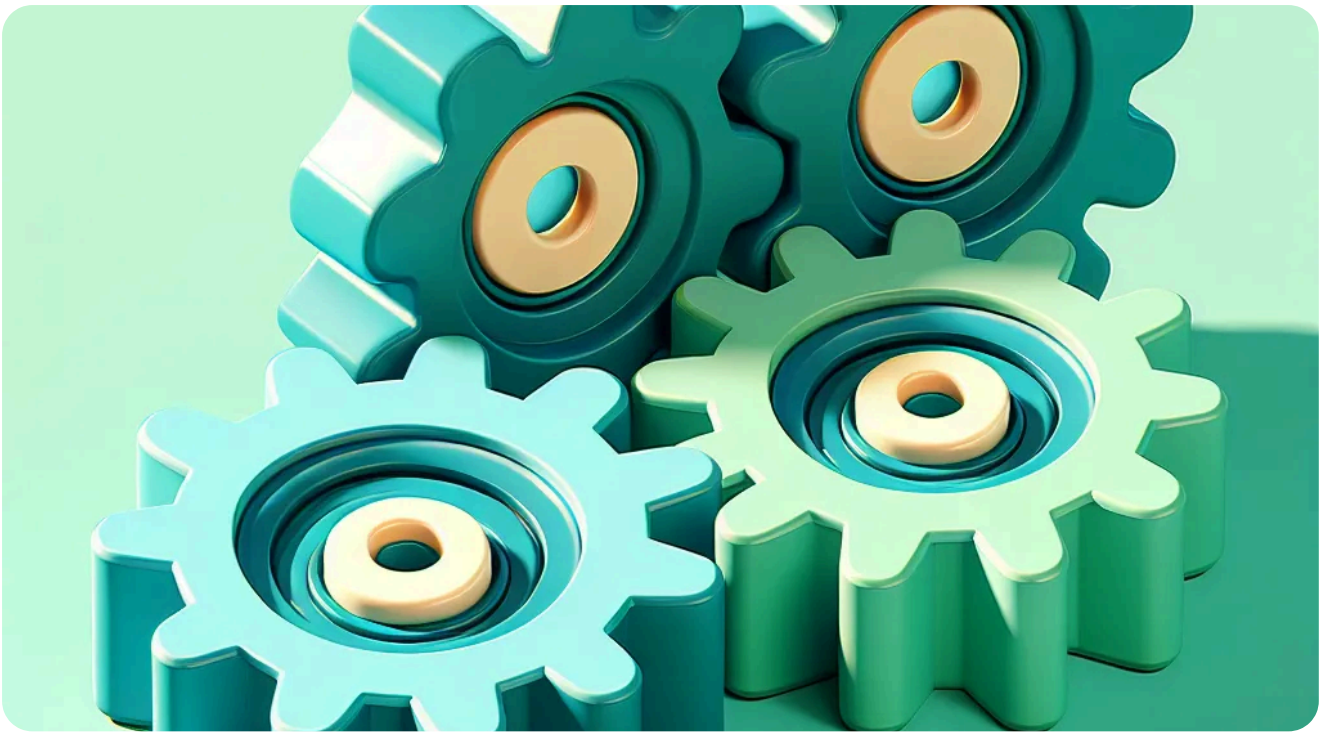


5XDYN

PROJET WEB DYNAMIQUE

Thibault Six · Thomas Copin

42 chapitres

SUPPORTS DE COURS

https://cours.tsix.be/cours/projet-web-dynamique

Version 3 — Fiche UE 2026-2027 liée depuis la page d'accueil · 10/10/2026

PRÉSENTATION

PROJET WEB DYNAMIQUE

1. Programme

L'étudiant sera capable :

- face au matériel et au logiciel adéquats et face à une structure informatique opérationnelle connectée à Internet, disposant des logiciels appropriés et de la documentation nécessaire, en utilisant le vocabulaire technique et l'orthographe adéquate, en respectant la législation, les normes et standards en vigueur,
- et au départ d'une proposition de projet individuel ou collectif formulée par le chargé de cours :
  - exploitant un « template » existant,
  - développant, par exemple, une des thématiques suivantes : e-commerce, site d'entreprise, sites institutionnel, culturel, associatif,...
- de décrire, de caractériser et de produire le cahier des charges du projet ;
- de décomposer le projet en ses différentes étapes depuis sa conception jusqu'à sa clôture;
- d'identifier les acteurs (collaborateurs, prestataires de service,...) intervenant dans la réalisation d'un projet web dynamique, de caractériser leurs rôles, leurs droits et leurs responsabilités ;
- d'établir la planification du projet à l'aide d'un outil informatique ;
- d'établir une estimation des coûts de la mise en œuvre du projet ;
- de construire un dossier technique reprenant les différentes étapes et leur planification ;
- d'identifier les éléments du projet susceptibles de poser des problèmes juridiques ;
- de mettre en œuvre le projet en développant, parmi les concepts suivants :
  - la gestion des contenus dynamiques au travers d'une interface administrateur sécurisé,
  - un système optimisé d'upload de fichiers (modifications des paramètres du fichier, restriction de taille, de poids, de type,...),
  - la pagination de l'affichage des résultats d'une requête ;
  - l'intégration de services internes et tiers (flux RSS, newsletters, réseaux sociaux, votes et commentaires, concours,...),
  - la gestion de sélections, de filtres et de recherches au sein du site ;
  - la gestion de la sécurisation et des droits d'accès aux contenus (administrateur, utilisateur public, utilisateur enregistré, gestionnaire,...),

  - l'affichage différencié des contenus (accessibilité, langue, sécurité, fonctionnalités, disponibilité de l'information,...), en fonction des profils utilisateurs,
  - la programmation asynchrone (AJAX, JSON, XML,...),
  - l'exploitation de fichiers de données structurées (XML, CSV, texte,..) en lecture/écriture,
  - l'optimisation du code, du cache et des échanges avec la base de données,
  - la programmation orientée objet,
  - l'exploitation d'un Framework,
  - ... ;
- d'identifier des menaces et de sécuriser le site en exploitant par exemple :
  - l'utilisation des outils spécifiques de protection et d'identification,
  - la protection contre des injections SQL, des attaques XSS, des vols de session, par détournement de cookies,...,
  - la réécriture d'url,
  - les paramétrages et les restrictions d'accès au serveur,
  - ... ;
- d'identifier des erreurs de programmation au moyen d'outils ou de techniques de débogage et d'y apporter une solution pertinente ;
- d'utiliser en vue de son amélioration, le cas échéant, le projet web dynamique pour développer son travail de fin d'études ;
- d'utiliser à bon escient la documentation disponible.

2. Capacités terminales

Pour atteindre le seuil de réussite, l'étudiant sera capable :

- face au matériel et au logiciel adéquats et face à une structure informatique opérationnelle connectée à Internet, disposant des logiciels appropriés et de la documentation nécessaire, en utilisant le vocabulaire technique et l'orthographe adéquate, en respectant la législation, les normes et standards en vigueur,
- et au départ d'une proposition de projet individuel ou collectif formulée par le chargé de cours :
  - exploitant un « template » existant,
  - développant, par exemple, une des thématiques suivantes : e-commerce, site d'entreprise, sites institutionnel, culturel, associatif,...
  - de produire un cahier des charges et son dossier technique (à titre d'exemples : planning, échéancier, techniques de développement, services à implémenter, développement futur, maintenance, formation, sécurité,...) ;
  - de déployer le site répondant aux consignes figurant dans la proposition du chargé de cours.

Pour la détermination du degré de maîtrise, il sera tenu compte des critères suivants:

- la qualité du code,
- le design et l'ergonomie du site,
- la qualité des documents produits,
- les fonctionnalités supplémentaires implémentées,
- le respect des législations en vigueur,

3. Évaluation

- Fiche UE 2026-2027 (PDF) PDF
- Grille d'évaluation (PDF) PDF

Sommaire

- PROJET WEB DYNAMIQUE

1 Laravel
- 1.1 Hébergement OVH
- 1.2 Hébergement O2Switch
- 1.3 Collaboration avec Git
- 1.4 Installer un projet Laravel existant
- 1.5 Système de Gestion pour l'Atelier 404
- 1.6 Laravel Dusk

2 Inertia
- 2.1 Pourquoi ?
- 2.2 Installation
- 2.3 Shadcn
- 2.4 Pages
- 2.5 Wayfinder
- 2.6 Routage
- 2.7 Formulaire
- 2.8 Pagination
- 2.9 Policies
- 2.10 Exercices
- 2.11 SEO
- 2.12 OG tags
- 2.13 Google Analytics
- 2.14 Exercice SEO

3 Accessibilité
- 3.1 a11y : Coder pour les humains
- 3.2 Sémantique & Structure
- 3.3 Formulaire
- 3.4 SPA, Modales & Notifications
- 3.5 Testing & Outils
- 3.6 Conclusion

4 Mini chatGPT
4.1 Consignes Examen
4.2 Api
4.3 Markdown
4.4 Exercice 1 : Réponse à des questions
4.5 Exercice 2 : Interface de Chat
4.6 Fenêtre de contexte
4.7 Prompt Engineering
4.8 Exercice 3 : Instructions personnalisées
4.9 Exercice 4 : Streaming
4.10 Reconnaissance d'images
4.11 Annexe : Formatage des Messages pour l'API
4.12 Conclusion

CHAPITRE 1

laravel

Dans ce chapitre, nous allons consolider les compétences acquises lors de notre précédent cours sur Laravel. Nous allons explorer des aspects pratiques essentiels pour tout développeur web cherchant à maîtriser ce puissant framework PHP.



Nous commencerons par déployer un projet Laravel sur un serveur mutualisé OVH, une étape cruciale pour rendre vos applications accessibles au public. Ce processus vous permettra de comprendre les particularités de l'hébergement mutualisé et d'apprendre à configurer votre environnement de production.

Ensuite, nous nous lancerons dans la création d'un nouveau projet passionnant : un système de commandes de sandwichs. Ce projet vous donnera l'occasion de mettre en pratique vos connaissances en développement d'applications web dynamiques, tout en explorant des

fonctionnalités avancées de Laravel.

Enfin, nous découvrirons **Laravel Dusk**, un outil indispensable pour tester vos applications de manière automatisée. Vous apprendrez à écrire des tests de bout en bout, garantissant ainsi la fiabilité et la robustesse de vos projets.

Ce chapitre est conçu pour renforcer vos compétences techniques et vous préparer à relever des défis plus complexes dans le développement web. Préparez-vous à plonger dans l'univers de Laravel et à enrichir votre boîte à outils de développeur.

1.1 Hébergement OVH

L'hébergement d'une application web est une étape cruciale pour sa mise en production. OVH, en tant que fournisseur d'hébergement, offre des services robustes pour déployer des applications Laravel. Ce chapitre guide à travers les différentes étapes nécessaires pour préparer, uploader et configurer une application Laravel sur un serveur OVH.

Étape 1 : Créer et importer la base de données

La gestion de la base de données est une étape cruciale pour le bon fonctionnement de votre application Laravel en production. Cette étape concerne la préparation de la base de données locale, son exportation, et son importation dans l'environnement de production OVH.

Si vous avez utilisé sqlite, il suffit d'uploader le fichier database.sqlite avec le reste du code.

1.1. Faire un dump de la base de données

Pour préparer votre base de données à être transférée sur le serveur OVH, suivez ces étapes :

1. **Ajouter un préfixe aux noms des tables** : Dans le fichier de configuration de la base de données de Laravel (généralement situé dans `config/database.php`), vous pouvez spécifier un préfixe pour les noms de tables pour éviter les conflits, surtout si vous partagez votre base de données avec d'autres applications. Par exemple, vous pourriez utiliser `mini_instagram_` comme préfixe.

config > database.php

```
5   return [
32      'connections' => [
44
45          'mysql' => [
46              'driver' => 'mysql',
47              'url' => env('DB_URL'),
48              'host' => env('DB_HOST', '127.0.0.1'),
49              'port' => env('DB_PORT', '3306'),
50              'database' => env('DB_DATABASE', 'laravel'),
51              'username' => env('DB_USERNAME', 'root'),
52              'password' => env('DB_PASSWORD', ''),
53              'unix_socket' => env('DB_SOCKET', ''),
54              'charset' => env('DB_CHARSET', 'utf8mb4'),
55              'collation' => env('DB_COLLATION', 'utf8mb4_unicode_ci'),
56              'prefix' => 'mini_instagram_',
57              'prefix_indexes' => true,
58              'strict' => true,
```

2. **Exécuter les migrations et les seeders** : Assurez-vous que votre base de données est à jour avec toutes les migrations et que les données initiales nécessaires sont chargées. Utilisez les commandes suivantes :

```bash
php artisan migrate:fresh --seed
```

3. **Créer un dump de la base de données** : Utilisez la commande `mysqldump` pour créer un dump de votre base de données MySQL. Assurez-vous d'inclure les paramètres nécessaires pour capturer l'ensemble de la structure et des données :

```bash
mysqldump -u username -p database_name > database_dump.sql
```

Remplacez `username` par votre nom d'utilisateur MySQL et `database_name` par le nom de votre base de données.

1.2. Importer le dump dans OVH via phpMyAdmin

Une fois le dump de votre base de données créé, vous devez l'importer dans votre base de données sur OVH :

1. **Accéder à phpMyAdmin** : Connectez-vous à phpMyAdmin via l'interface OVH. Vous trouverez généralement un lien direct dans le tableau de bord de votre hébergement.
2. **Importer le fichier** `.sql` : Dans phpMyAdmin, sélectionnez la base de données que vous venez de créer, puis utilisez l'onglet "Importer" pour charger le fichier `database_dump.sql`. Suivez les instructions pour compléter l'importation.

Étape 2 : Préparer le projet Laravel

2.1. Ouvrir le terminal et se positionner dans le dossier du projet

Avant de commencer à travailler sur votre projet Laravel, il est essentiel de savoir naviguer dans les dossiers à l'aide du terminal. Ouvrez votre terminal et utilisez les commandes `cd` (change directory) pour vous positionner dans le dossier de votre projet Laravel. Cette étape est fondamentale car toutes les commandes suivantes seront exécutées dans ce contexte.

2.2. Installer les dépendances avec Composer

Pour que votre application Laravel fonctionne correctement, elle doit avoir accès à certaines bibliothèques PHP, qui sont gérées par Composer. Exécutez la commande suivante dans votre terminal :

```bash
composer install --no-dev
```

Cette commande installe les dépendances nécessaires pour le fonctionnement de l'application en excluant les packages uniquement nécessaires en développement. C'est une pratique courante pour optimiser les performances et la sécurité de l'application en production.

2.3. Compiler les assets frontend

Laravel mix est un outil puissant qui vous permet de compiler et de minimiser les assets CSS et JavaScript. Pour préparer vos assets pour la production, exécutez :

```bash
npm run build
```

Il est important de ne pas uploader le dossier `node_modules` sur votre serveur de production car il contient des dépendances uniquement nécessaires en développement. Uploader ce dossier serait non seulement superflu mais aussi potentiellement risqué en termes de sécurité et de performance.

Nous avons maintenant préparé notre projet Laravel pour le déploiement. Passons à l'étape suivante qui consiste à uploader les fichiers sur le serveur OVH.

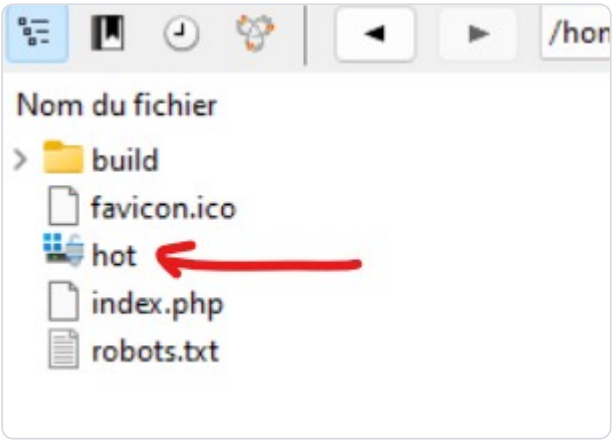
Étape 3 : Uploader les fichiers sur OVH

3.1. Utiliser un client FTP

Pour transférer votre projet Laravel sur le serveur OVH, vous aurez besoin d'un client FTP. Des clients tels que FileZilla ou Cyberduck sont recommandés pour leur facilité d'utilisation et leur fiabilité. Voici les étapes à suivre :

1. Ouvrez votre client FTP et configurez-le pour se connecter à votre espace d'hébergement OVH. Vous aurez besoin de l'adresse du serveur FTP, de votre nom d'utilisateur et de votre mot de passe, qui sont généralement fournis par OVH lors de la configuration de votre hébergement.
2. Une fois connecté, naviguez dans l'arborescence de votre espace local pour localiser le dossier de votre projet Laravel.
3. Transférez tous les fichiers de votre projet vers le serveur, à l'exception du dossier `node_modules`. Assurez-vous que tous les fichiers et dossiers nécessaires sont correctement uploadés, y compris le dossier `.env` et les dossiers `vendor` et `storage`.

Dans le dossier public, vous devez supprimer le fichier `hot` qui empecherait le front de fonctionner correctement :



3.2. Vérifier l'arborescence des fichiers

Après le transfert, il est crucial de vérifier que tous les fichiers et dossiers nécessaires sont correctement positionnés sur votre espace d'hébergement OVH. Voici les points clés à vérifier :

- Assurez-vous que le fichier `public/index.php` est bien présent et correctement placé car il est le point d'entrée de votre application Laravel.
- Vérifiez que les dossiers `bootstrap`, `config`, et `storage` sont bien présents et contiennent les fichiers attendus.

Une fois ces vérifications effectuées, votre application est prête à être configurée sur le serveur OVH.

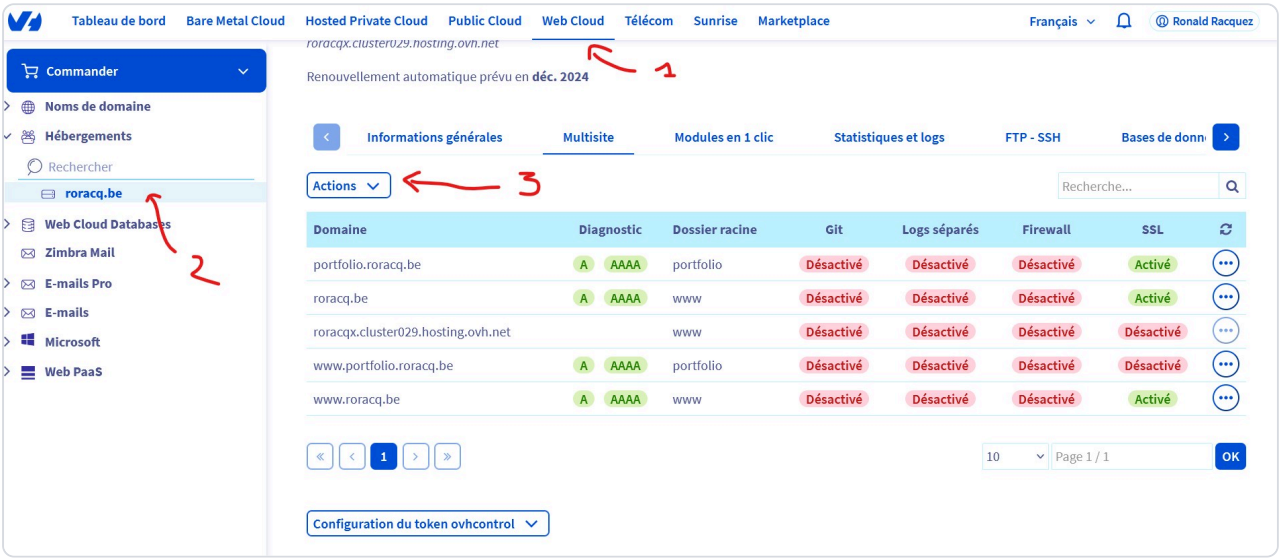
Étape 4 : Configurer l'hébergement sur OVH

Une fois que vos fichiers et votre base de données sont en place, il est temps de configurer l'hébergement pour s'assurer que votre application Laravel fonctionne correctement sur OVH.

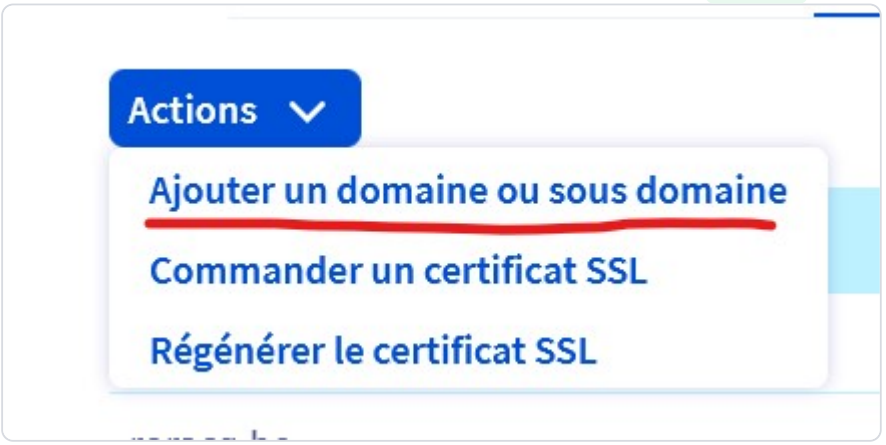
4.1. Ajouter un domaine ou sous-domaine

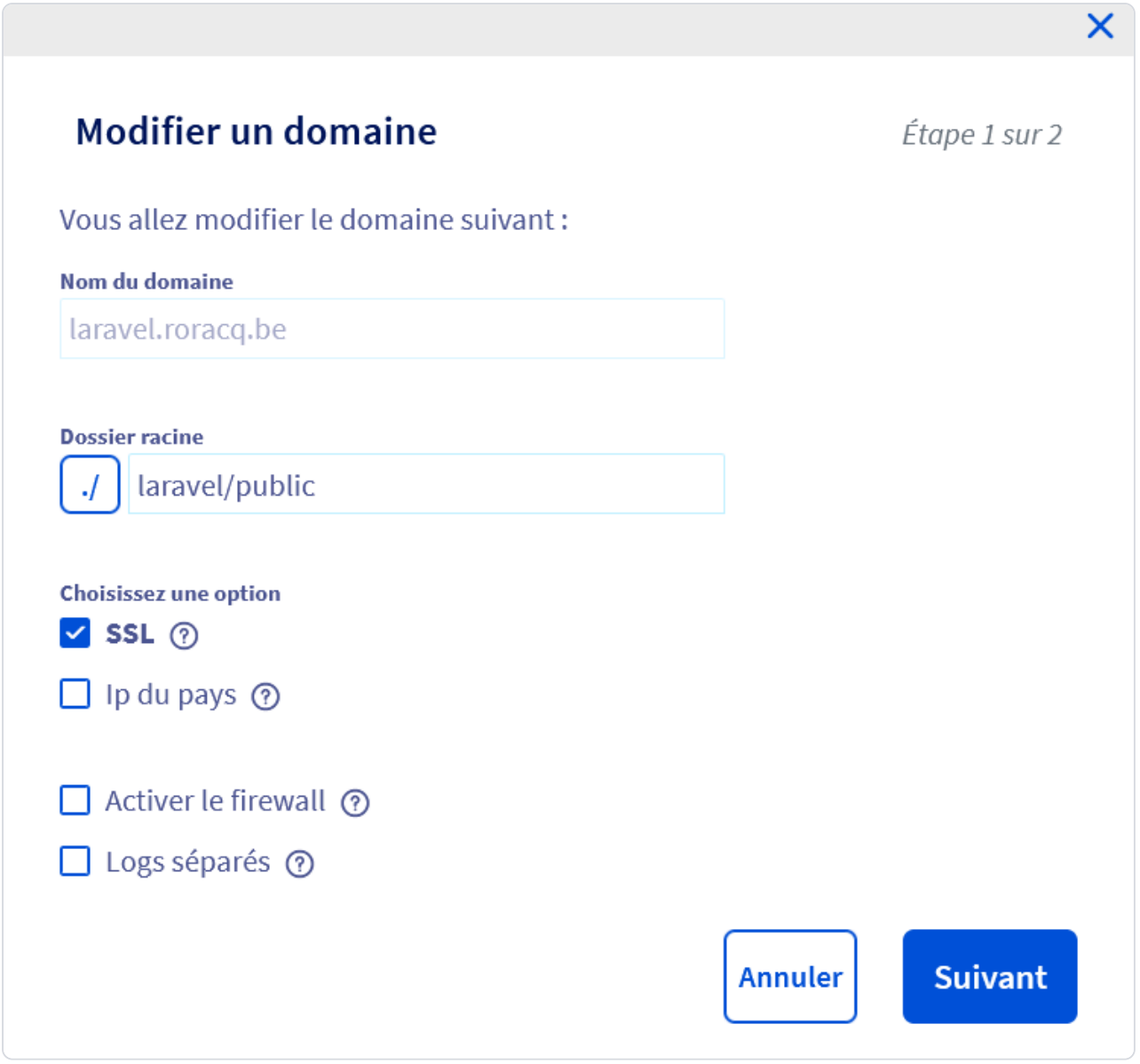
Pour que votre application Laravel soit accessible via Internet, vous devez configurer un domaine ou un sous-domaine. Voici comment procéder :

1. **Connectez-vous à l'interface OVH** : Utilisez vos identifiants pour accéder au tableau de bord OVH.
2. **Naviguez vers la section "Multisite"** : Ici, vous pouvez gérer tous les aspects liés à vos domaines.



3. **Ajouter un domaine ou sous-domaine** : Suivez les instructions pour ajouter un nouveau domaine ou configurer un sous-domaine existant. Assurez-vous que le domaine pointe vers le dossier racine de votre application Laravel, généralement le dossier `public`.





4.2. Modifier la configuration PHP

OVH permet de configurer la version de PHP et d'autres paramètres directement depuis son interface. Laravel 11 requiert PHP 8.2 ou supérieur.

Après avoir configuré votre domaine et la version de PHP, votre application devrait être presque prête à être utilisée. Il reste quelques étapes de configuration et de vérification avant de conclure le déploiement. Nous allons maintenant passer à la configuration du fichier `.env` pour assurer la connexion à la base de données.

Étape 5 : Configurer le fichier `.env`

Le fichier `.env` est crucial dans les applications Laravel car il contient toutes les configurations spécifiques à l'environnement dans lequel votre application s'exécute. Cette étape est essentielle pour s'assurer que votre application peut se connecter correctement à la base de données et

fonctionner correctement en production.

5.1. Modifier les paramètres de connexion à la base de données

Pour que votre application Laravel puisse interagir avec la base de données sur OVH, vous devez configurer les paramètres de connexion dans le fichier `.env`. Voici les paramètres principaux à configurer :

- `DB_CONNECTION` : le type de votre base de données (par exemple, `mysql`).
- `DB_HOST` : l'adresse du serveur de base de données. Sur OVH, cela sera généralement une adresse locale comme `localhost` ou une adresse spécifique fournie par OVH.
- `DB_PORT` : le port de connexion à la base de données, généralement `3306` pour MySQL.
- `DB_DATABASE` : le nom de votre base de données sur OVH.
- `DB_USERNAME` : votre nom d'utilisateur pour la base de données.
- `DB_PASSWORD` : le mot de passe associé à l'utilisateur de la base de données.

Voici un exemple de configuration :

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=nom_de_votre_base
DB_USERNAME=utilisateur
DB_PASSWORD=mot_de_passe
```

Assurez-vous que ces informations sont correctes et correspondent à celles que vous avez configurées sur OVH. Une fois modifié, sauvegardez le fichier `.env`.

5.2. Vérifier les autres configurations

En plus des configurations de la base de données, vérifiez et ajustez les autres paramètres dans le fichier `.env` qui pourraient être nécessaires pour votre environnement de production, tels que :

- `APP_ENV` : assurez-vous que cela est défini sur `production`.
- `APP_KEY` : une clé unique qui sécurise les sessions et les données cryptées de votre application. Vous pouvez la générer avec la commande `php artisan key:generate`.
- `APP_DEBUG` : définissez-le à `false` pour désactiver le mode débogage en production.

Après avoir configuré correctement votre fichier `.env`, il est temps de passer à la régénération du certificat SSL pour sécuriser les connexions à votre site.

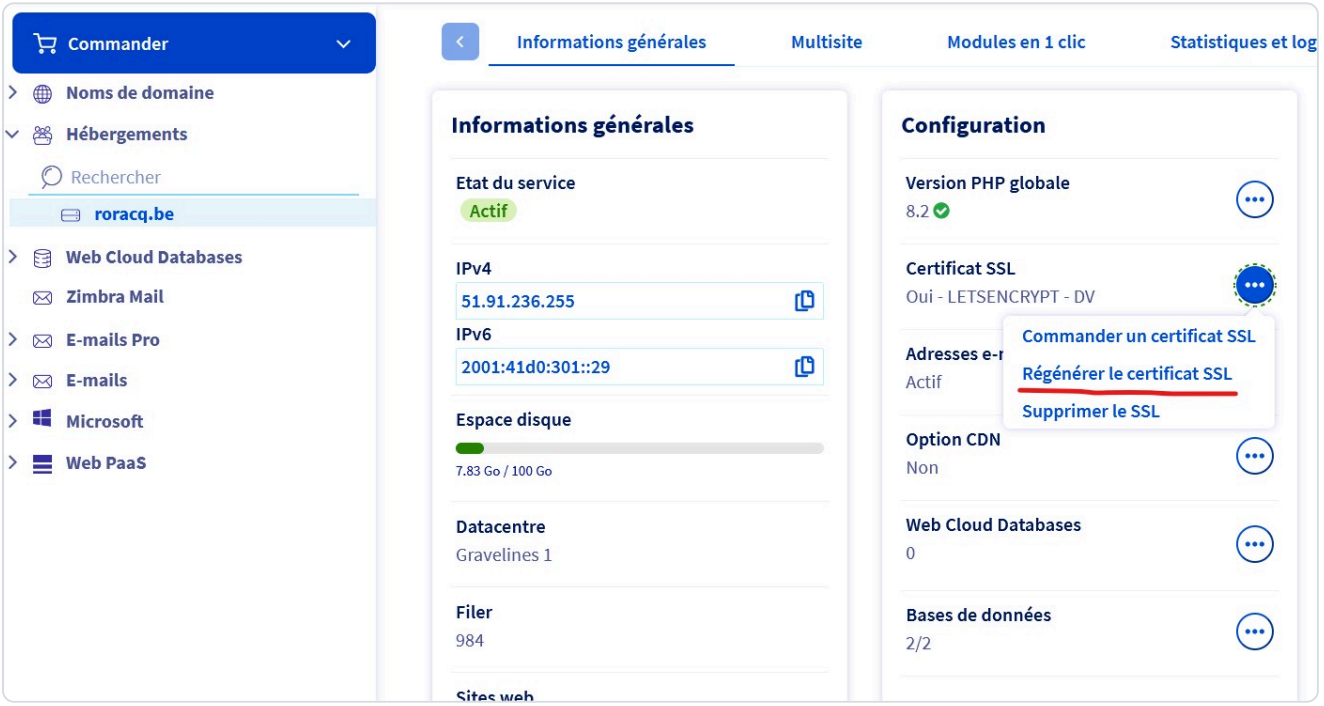
Étape 6 : Régénérer le certificat SSL

La sécurité est un aspect crucial de tout site web, et l'utilisation d'un certificat SSL (Secure Sockets Layer) est essentielle pour garantir que les données transmises entre le serveur et les utilisateurs sont chiffrées et sécurisées. OVH offre des options pour gérer facilement les certificats SSL de vos domaines.

6.1. Activer et régénérer le SSL

Pour activer ou régénérer un certificat SSL sur OVH, suivez ces étapes :

1. **Accéder à l'interface OVH** : Connectez-vous à votre compte sur le site d'OVH.
2. **Naviguez vers la section "Hébergements"** : Sélectionnez l'hébergement associé à votre domaine.
3. **Gérer le SSL** : Trouvez l'option pour gérer le SSL dans le tableau de bord de votre hébergement. OVH peut offrir un certificat SSL gratuit via Let's Encrypt.
4. **Activer/Régénérer le certificat** : Suivez les instructions pour activer ou régénérer le certificat SSL. Cela peut prendre quelques minutes à quelques heures pour que le certificat soit complètement installé et actif.



Assurez-vous que votre domaine est configuré pour utiliser HTTPS en modifiant les réglages DNS si nécessaire et en ajustant les configurations de votre application Laravel pour rediriger toutes les requêtes vers HTTPS.

Étape 7 : Tests de vérification

Après avoir configuré tous les aspects de votre application, il est important de réaliser une série de tests pour s'assurer que tout fonctionne comme prévu.

7.1. Vérifier les routes et pages du site

Testez toutes les routes principales de votre application pour vous assurer qu'elles chargent correctement sans erreurs. Cela inclut les pages d'accueil, les formulaires de connexion, les dashboards utilisateurs, et toute autre fonctionnalité clé de votre application.

7.2. Vérifier la connexion HTTPS

Accédez à votre site en utilisant `https://` pour confirmer que le SSL est actif et fonctionne correctement. Vérifiez que le navigateur affiche un cadenas vert ou une indication que la connexion est sécurisée.

7.3. Vérifier l'accès à la base de données

Testez les fonctionnalités de votre application qui nécessitent des interactions avec la base de données, comme les inscriptions, les connexions, et les mises à jour de données. Assurez-vous que toutes les opérations se déroulent sans erreurs et que les données sont correctement récupérées et stockées.

Ces tests sont essentiels pour s'assurer que votre application est prête pour le lancement et fonctionnera correctement pour vos utilisateurs finaux. Une fois ces tests passés avec succès, votre application est prête à être lancée en production sur OVH.

Étape 8 : Configurer le disque public pour utiliser le dossier public/storage

Pour que votre application Laravel gère correctement les fichiers statiques et les uploads sur l'environnement de production OVH, il est important de configurer correctement le disque public. Cette configuration permettra de stocker et de servir les fichiers directement depuis le dossier `public/storage` sans avoir besoin de créer des liens symboliques. Nous devons faire ça à cause d'une limitation de l'hébergement OVH qui n'autorise pas l'accès aux fichiers en utilisant des liens symboliques.

8.1. Supprimer le dossier public/storage

Avant de modifier la configuration, assurez-vous que le dossier `public/storage` qui pourrait avoir été créé automatiquement lors de l'upload des fichiers est supprimé. Cela évitera toute confusion ou conflit avec la configuration que vous allez établir. Vous pouvez supprimer ce dossier via un client FTP ou en utilisant une commande SSH si vous avez accès au terminal du serveur.

8.2. Modifier la configuration du disque public

Pour ajuster la configuration du disque public afin qu'il utilise le dossier `public/storage`, suivez ces étapes :

1. **Ouvrir le fichier de configuration** : Accédez au fichier `config/filesystems.php` dans votre projet Laravel.
2. **Modifier la configuration du disque public** : Localisez la section `disks` et modifiez la configuration du disque `public` pour qu'elle ressemble à ce qui suit :

```php
'disks' => [

    // ...

    'public' => [
        'driver' => 'local',
        'root' => public_path('storage'),
        'url' => env('APP_URL') . '/storage',
        'visibility' => 'public',
        'throw' => false,
    ],

    // ...
]
```

3. **Explications des modifications** :
   - `root` : Définit le chemin d'accès au dossier où les fichiers seront stockés, ici `public/storage`.

8.3. Sauvegarder et tester la configuration

Après avoir modifié la configuration, sauvegardez le fichier et testez pour vous assurer que les fichiers sont correctement stockés et accessibles. Vous pouvez le faire en uploadant un fichier via votre application et en vérifiant ensuite qu'il est accessible via l'URL spécifiée.

Cette configuration vous permet de gérer les fichiers statiques et les uploads de manière efficace et sécurisée, en utilisant le système de fichiers intégré de Laravel sans dépendre de liens symboliques, ce qui est particulièrement utile dans des environnements d'hébergement avec des restrictions spécifiques comme OVH.

1.2 Hébergement O2SWITCH

Étape 1 : Créer et importer la base de données

> ***Astuce SQLite***
> *Si tu es un **noob** (ou simplement malin) qui utilise encore **SQLite**, tu peux ignorer toutes ces histoires de dump : tu balances **directement** ton fichier `database.sqlite` sur le serveur, bim, c'est torché. Plus simple, plus rapide.*

1.1. Exporter ta base MySQL locale (optionnel si SQLite)

1. **Mets tes migrations à jour** localement :

```bash
php artisan migrate:fresh --seed
```

2. **Fais un dump** de ta base :

```bash
mysqldump -u username -p my_database > database_dump.sql
```

1.2. Importer sur o2switch via phpMyAdmin

1. Dans le **cPanel** d'o2switch, repère l'icône **phpMyAdmin** et connecte-toi.
2. Crée une nouvelle base (si ce n'est pas déjà fait).
3. **Importe** ton fichier `.sql`.

Étape 2 : Préparer le projet Laravel

1. **Sélectionner la version PHP** : Dans la partie "Logiciel" du cPanel, clique sur "Sélectionner une version de PHP". Choisis **PHP 8.2+** (ou 8.3).
2. **Installer tes dépendances** (en local) :

```bash
composer install --no-dev
```

3. **Compiler tes assets** (avec npm/Vite) :

```bash
npm run build
```

4. **Supprimer** `node_modules` qui pèse une tonne.

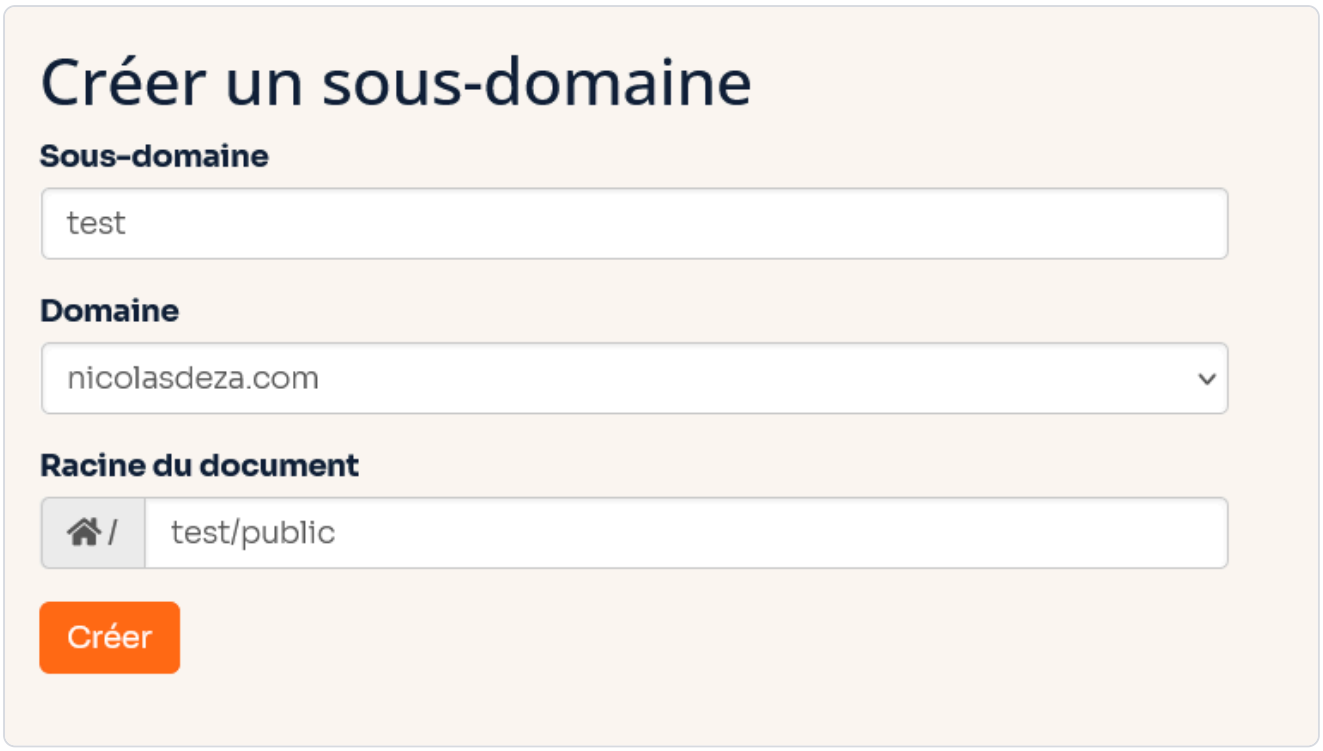
Étape 3 : Uploader les fichiers

1. Dans cPanel > **Fichiers** > **Comptes FTP spéciaux**, retrouve tes identifiants.
2. Connecte-toi avec FileZilla (ou autre) en utilisant **le même mot de passe** que cPanel.
3. **Transfère** tous tes fichiers Laravel, **sauf** `node_modules`.
4. Dans `public/`, supprime le fichier `hot` si tu l'aperçois (sinon, meltdown de ton front).



Étape 4 : Créer un (sous-)domaine

1. Va dans cPanel > **Domaines** > **Sous-domaines**.
2. Tapes-y "test" (ou un nom plus stylé), sélectionne ton domaine principal, et assure-toi que le répertoire pointe vers `test/public`.
3. Clique sur "Créer".



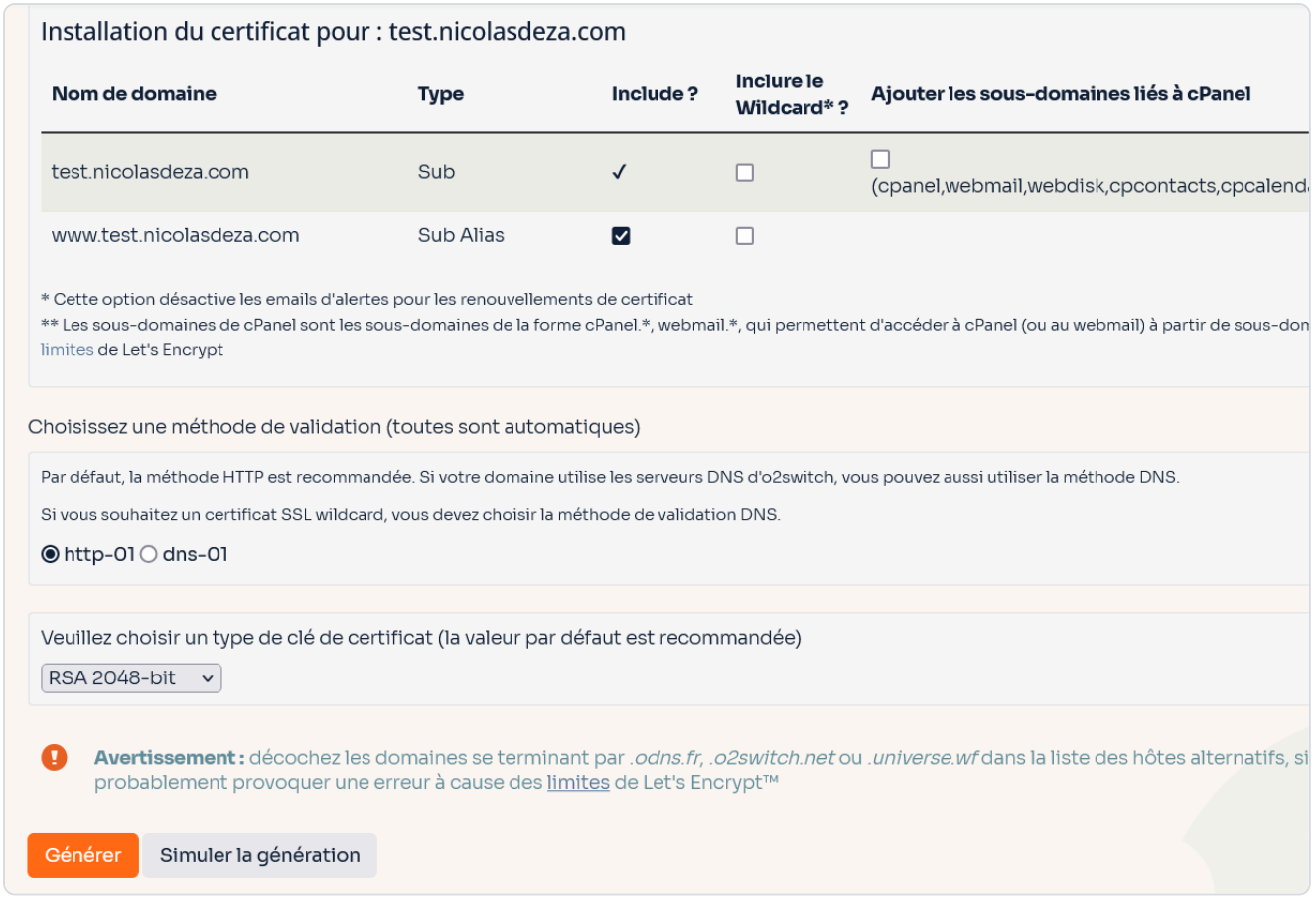
Étape 5 : Configurer ton `.env`

1. **DB_CONNECTION** : `mysql` (ou `sqlite` si t'as choisi la voie facile).
2. **DB_HOST** : souvent `localhost` sur o2switch.
3. **DB_PORT** : 3306 par défaut.
4. **DB_DATABASE**, **DB_USERNAME**, **DB_PASSWORD** : ceux créés dans le cPanel.
5. **APP_ENV=production**, **APP_DEBUG=false**
6. **APP_URL** : `https://test.mondomaine.com` ou ce que tu utilises.

Sauvegarde et mets le `.env` sur le serveur (ou modifie-le directement là-bas).

Étape 6 : Activer le HTTPS via Let's Encrypt

1. Dans cPanel, cherche le module **SSL Let's Encrypt**.
2. Sélectionne ton (sous-)domaine, clique sur "Générer" (méthode HTTP par défaut).
3. Patiente, puis vérifie que ça passe en vert sur `https://…`.



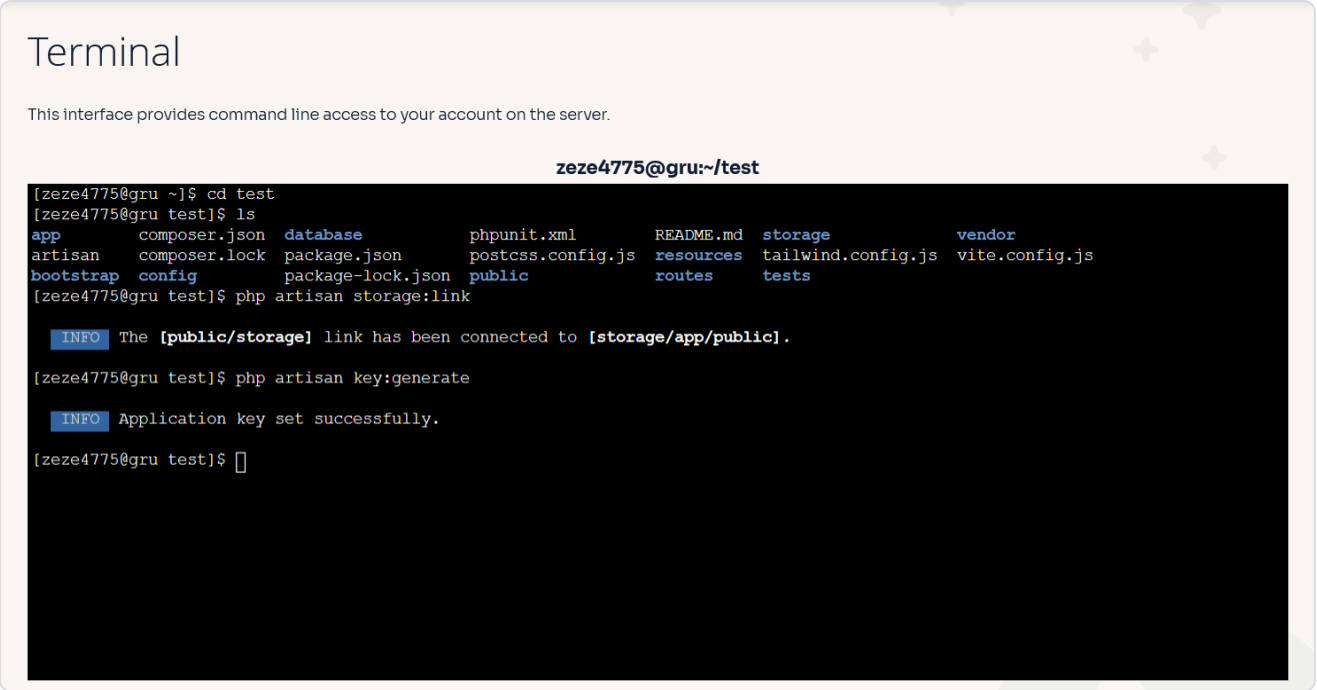
Étape 7 : Commandes via le terminal cPanel

o2switch te file un **Terminal** (menu “Terminal” dans le cPanel) pour lancer quelques trucs **depuis le serveur** :

```bash
# 1. Aller dans le dossier de ton projet
cd test

# 2. Générer la clé d’appli Laravel
php artisan key:generate

# 3. Créer le lien storage ⇒ public/storage
# (si tu veux gérer les uploads directement)
php artisan storage:link
```



Étape 8 : Vérifications finales

1. **Visite** ton site en HTTPS : `https://test.mondomaine.com`.
2. **Teste** toutes tes pages et routes.
3. **Regarde** s'il n'y a pas d'erreur 500 (check `storage/logs/laravel.log` si souci).

(_) Voilà, t'as (normalement) un beau site Laravel en ligne. L'objectif, c'est que ça marche... mais si tu te plantes, c'est pas la faute du tuto, hein. Bonne chance, petit Padawan de la prod ! ('-')

1.3 Workflows de collaboration avancés

Slides du chapitre

- Collaboration-avec-Git-and-GitHub.pdf

Modèles de branchement : Comprendre les stratégies

La collaboration sur un projet Git nécessite une approche structurée de la gestion des branches. Deux modèles principaux émergent : **Git Flow** et **GitHub Flow**, chacun avec ses propres forces et cas d'utilisation.

Git Flow : Un modèle complet pour les projets complexes

Git Flow est un modèle de branchement robuste, particulièrement adapté aux projets avec des cycles de release bien définis.

Structure typique des branches :

- `main` (ou `master`) : branche de production
- `develop` : branche d'intégration pour le développement
- `feature/*` : branches pour les nouvelles fonctionnalités
- `release/*` : branches de préparation de release
- `hotfix/*` : branches pour les corrections critiques

Exemple de workflow Git Flow :

1. Créer une branche `feature` depuis `develop`
2. Développer la fonctionnalité
3. Merger la `feature` dans `develop`
4. Créer une branche `release` pour les tests
5. Merger `release` dans `main` et `develop`
6. Appliquer des tags de version

GitHub Flow : Approche simplifiée et continue

GitHub Flow privilégie la simplicité et le déploiement continu.

Structure des branches :

- `main` : branche de production toujours déployable
- `feature/*` : branches pour chaque développement

Processus :

1. Créer une branche à partir de `main`
2. Développer et commiter
3. Ouvrir une Pull Request
4. Revue et discussion
5. Déployer depuis la branche
6. Merger dans `main`

Comparaison des approches

Critère	Git Flow	GitHub Flow
Complexité	Élevée	Faible
Releases planifiées	Oui	Non
Déploiement continu	Difficile	Facile

Critère	Git Flow	GitHub Flow
Adapté à	Logiciels avec cycles	Développement web, startups

Choix du workflow : Considérations pratiques

Le choix entre Git Flow et GitHub Flow dépend de plusieurs facteurs :

- Nature du projet
- Fréquence des releases
- Taille de l'équipe
- Processus de déploiement

Conseils de sélection :

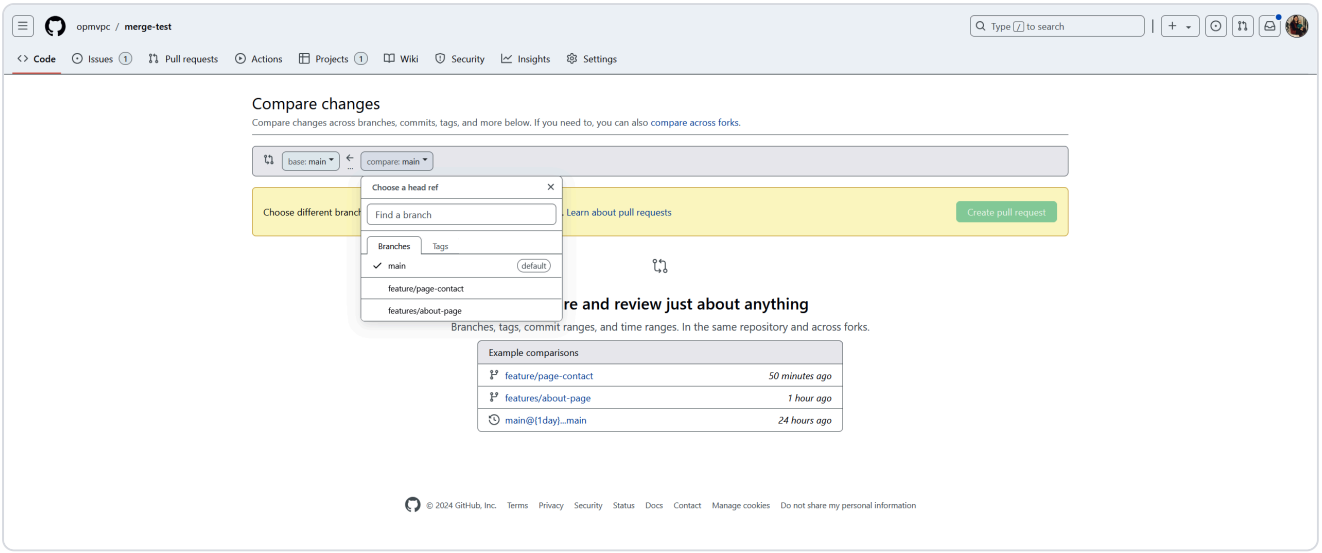
- Projets avec versions stables → **Git Flow**
- Développement web, MVP → **GitHub Flow**
- Projets open-source → Généralement **GitHub Flow**

Pull Request : Concept et Définition

Qu'est-ce qu'une Pull Request ?

Une Pull Request (PR) est bien plus qu'un simple mécanisme technique de fusion de code. C'est un outil de communication et de collaboration qui permet aux développeurs de proposer des modifications à un projet partagé. Imaginez-la comme une invitation à discuter et à examiner ensemble une série de changements avant de les intégrer définitivement au projet.

Dans le workflow GitHub, une Pull Request signale à l'équipe : "J'ai travaillé sur ces modifications. Pouvez-vous les examiner et me donner votre avis avant que je ne les incorpore au projet principal ?" C'est un moment crucial d'échange et de validation collective.



Processus de base d'une Pull Request

La création d'une Pull Request suit généralement ces étapes :

1. **Créer une nouvelle branche**
   - Partez toujours de la branche principale (généralement `main` ou `master`)
   - Donnez un nom clair et descriptif à votre branche (ex: `feature/login-authentication`)
2. **Développer votre fonctionnalité ou correction**
   - Travaillez sur votre branche
   - Effectuez des commits réguliers et significatifs
   - Assurez-vous que votre code fonctionne localement
3. **Pousser votre branche sur GitHub**

```bash
git push -u origin feature/login-authentication
```

4. **Ouvrir la Pull Request**
   - Sur GitHub, sélectionnez votre branche
   - Cliquez sur "New Pull Request"
   - Décrivez vos modifications
   - Indiquez le contexte et les raisons de ces changements
5. **Revue et discussion**
   - Les membres de l'équipe examinent votre code
   - Ils peuvent commenter, suggérer des modifications
   - Vous pouvez répondre, clarifier, modifier votre code

6. **Finalisation**
    - Si tout est validé : merger la branche
    - Si des modifications sont nécessaires : les apporter et mettre à jour la PR

Code Review : Concepts et Pratiques

Définition de la Code Review

La code review est un processus collaboratif essentiel dans le développement logiciel moderne. Il s'agit d'un examen systématique du code écrit par un développeur, effectué par ses pairs ou des membres plus expérimentés de l'équipe.

Les objectifs principaux sont multiples :

- Détecter et corriger les bugs potentiels
- Partager les connaissances au sein de l'équipe
- Maintenir une haute qualité de code
- Garantir la cohérence technique du projet
- Faciliter l'apprentissage collectif

Risques d'une absence de Code Review

Sans revue de code, une équipe s'expose à plusieurs risques significatifs :

- Bugs non détectés qui peuvent s'accumuler
- Code peu lisible et difficilement maintenable
- Isolation des connaissances techniques
- Accumulation rapide de dette technique
- Failles de sécurité potentiellement non identifiées
- Divergence des pratiques de codage au sein de l'équipe

Bonnes Pratiques de Code Review

Qui peut faire une revue ?

Une revue de code efficace implique généralement :

- Les développeurs expérimentés du projet
- Les membres directs de l'équipe
- Le lead technique ou l'architecte
- Les contributeurs réguliers ayant une bonne compréhension du projet

L'idéal est de varier les reviewers pour bénéficier de perspectives différentes.

Comment bien faire une revue

Critères d'évaluation :

- Lisibilité et compréhension du code
- Respect des conventions du projet
- Logique et efficacité de l'implémentation
- Performance et complexité algorithmique
- Gestion appropriée des erreurs
- Qualité et pertinence des commentaires
- Couverture et qualité des tests

Approche constructive :

- Commenter le code, jamais la personne
- Être précis et bienveillant
- Proposer des alternatives concrètes
- Expliquer le raisonnement derrière chaque suggestion

Exemple de commentaire constructif :

```
Cette méthode pourrait bénéficier d'une refactorisation.
Je suggère d'extraire la logique de validation dans une méthode dédiée,
ce qui améliorerait la lisibilité et permettrait de réutiliser ce comportement.
```

Intégration Continue avec Laravel et GitHub Actions

Définition

L'intégration continue est une pratique de développement qui consiste à :

- Automatiser la compilation et les tests du code
- Détecter rapidement les problèmes
- Garantir la qualité à chaque modification

Configuration pour un projet Laravel

Voici un exemple de workflow GitHub Actions adapté à Laravel :

```yaml
name: Laravel Tests
on: [push, pull_request]
jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'

      - name: Install Dependencies
        run: composer install -q --no-ansi --no-interaction --no-scripts --no-progress
--prefer-dist

      - name: Generate key
        run: php artisan key:generate

      - name: Run Pest
        run: ./vendor/bin/pest

      - name: Run PHPStan
        run: ./vendor/bin/phpstan analyse
```

Installation des outils

```bash
# Installation de Pest
composer require pestphp/pest --dev --with-all-dependencies

# Installation de PHPStan
composer require --dev phpstan/phpstan
```

Exemple de configuration PHPStan (`phpstan.neon`) :

```
parameters:
    level: 6
    paths:
        - app
        - tests
```

Cette configuration garantira que chaque modification passe les tests et respecte les standards de qualité de code.

Techniques de Travail Collaboratif : Résolution des Conflits

Comprendre les Conflits de Fusion

Qu'est-ce qu'un Conflit de Fusion ?

Un conflit de fusion survient lorsque deux développeurs modifient les mêmes parties d'un fichier dans différentes branches. Git, malgré sa sophistication, ne peut pas toujours décider automatiquement quelle version conserver.

Imaginez deux développeurs travaillant simultanément sur le même fichier PHP. L'un modifie une méthode de validation, l'autre change la logique de cette même méthode. Lorsqu'ils tentent de fusionner leurs branches, Git signale un conflit, nécessitant une intervention manuelle.

Types de Conflits

Il existe principalement deux catégories de conflits :

1. **Conflits de Contenu** : Ils surviennent quand les mêmes lignes de code sont modifiées différemment. Par exemple :

```php
// Version sur ma branche
public function validate(data) {
    return $data['email'] =/= '';
}

// Version sur la branche principale
public function validate(data) {
    return filter_var($data['email'], FILTER_VALIDATE_EMAIL);
}
```

2. **Conflits Structurels** : Ils concernent des changements plus globaux comme la réorganisation du code, le déplacement de méthodes ou la modification de l'architecture.

Prévention des Conflits

La meilleure stratégie pour gérer les conflits est de les prévenir :

Communication d'Équipe

- Informez vos collègues de vos développements en cours
- Utilisez les issues GitHub pour planifier le travail
- Synchronisez régulièrement votre branche avec la branche principale

Pratiques de Développement

- Créez des branches courtes et ciblées
- Mergez fréquemment
- Évitez de travailler trop longtemps sur des branches isolées

Processus de Résolution de Conflits

Identification des Conflits

Première étape : identifier les fichiers problématiques

```bash
git status
```

Cette commande affichera les fichiers en conflit.

Comprendre la Structure d'un Conflit

Git marque les zones conflictuelles avec des balises spécifiques :

```php
<<<<<<< HEAD
// Votre version actuelle
public function validate(data) {
    return $data['email'] !== '';
}
=======
// Version en cours de fusion
public function validate(data) {
    return filter_var($data['email'], FILTER_VALIDATE_EMAIL);
}
>>>>>>> nouvelle_branche
```

Résolution Manuelle

Étapes de résolution :

1. Ouvrez le fichier en conflit
2. Éditez manuellement pour garder la meilleure implémentation
3. Supprimez les balises de Git
4. Choisissez l'implémentation la plus appropriée

Exemple de résolution :

```php
public function validate($data) {
    return !empty($data['email']) &&
           filter_var($data['email'], FILTER_VALIDATE_EMAIL);
}
```

Validation de la Résolution

Après avoir résolu manuellement :

```bash
git add fichier_resolu
git commit -m "Résolution du conflit de fusion"
```

Outils de Résolution dans Visual Studio Code

VSCode offre des outils visuels puissants pour résoudre les conflits :

- Boutons interactifs de résolution
- Vue comparative côte à côte
- Sélection facilitée des modifications

Stratégies de Choix

Lors de la résolution, considérez :

- La logique métier
- La performance du code
- La cohérence avec l'architecture existante
- L'intention originale du développement

Conseils Éthiques et Pratiques

Lors d'un Conflit

- Communiquez toujours avec le développeur concerné
- Cherchez à comprendre le contexte des modifications
- Privilégiez la solution technique la plus propre
- Restez professionnel et constructif

Cette approche transforme un conflit potentiellement frustrant en une opportunité de collaboration et d'amélioration du code.

GitHub Projects : Outil de Suivi et de Planification

Présentation de GitHub Projects

GitHub Projects est un outil intégré de gestion de projet directement dans l'écosystème GitHub. Il permet aux équipes de développement de suivre, organiser et planifier leur travail de manière visuelle et collaborative.

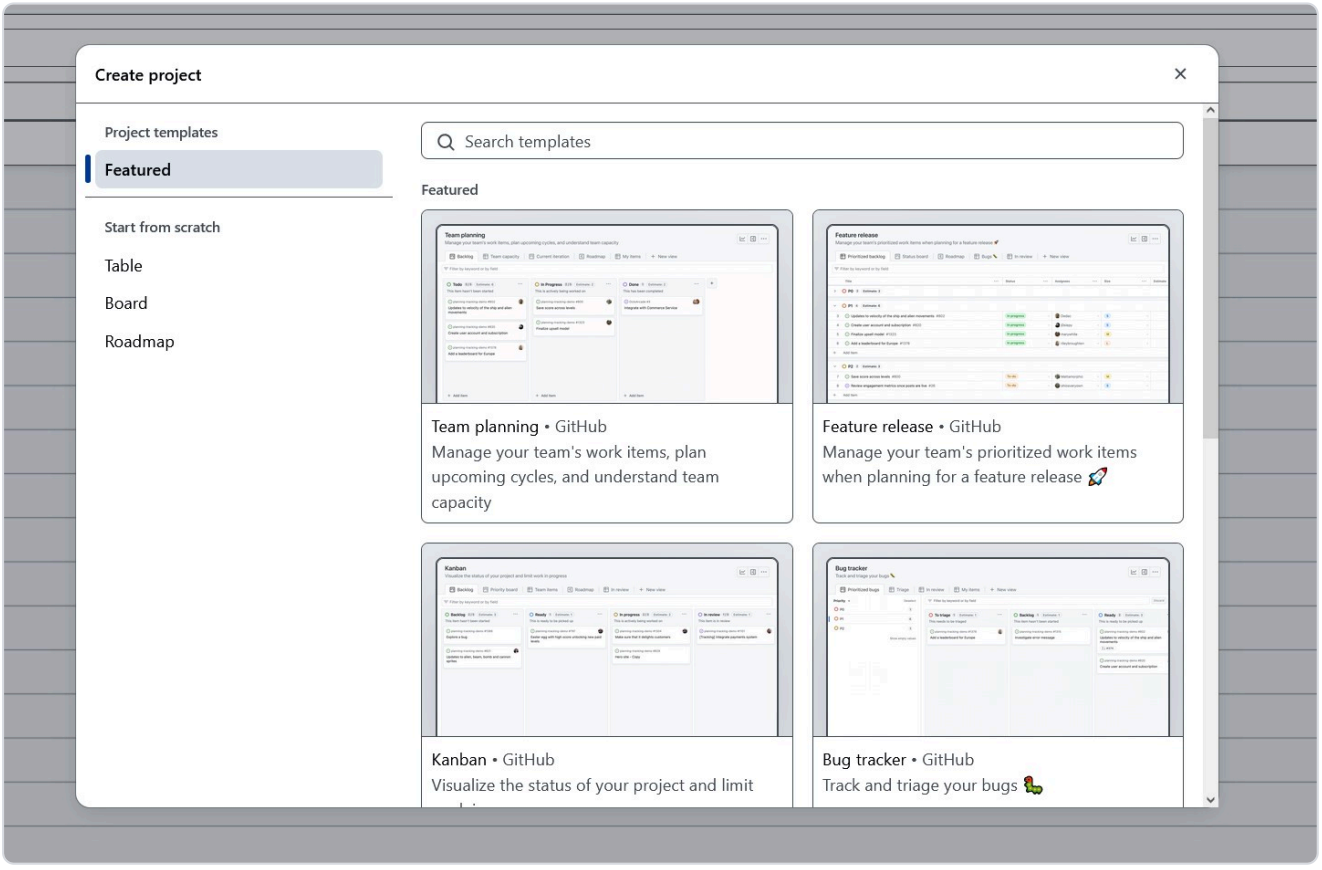
Les projets GitHub peuvent prendre plusieurs formes :

- Tableau Kanban : colonnes représentant les étapes de progression
- Liste : vue linéaire des tâches
- Tableau de bord : vue personnalisée selon les besoins de l'équipe

Création d'un Projet

La création d'un projet suit quelques étapes simples :

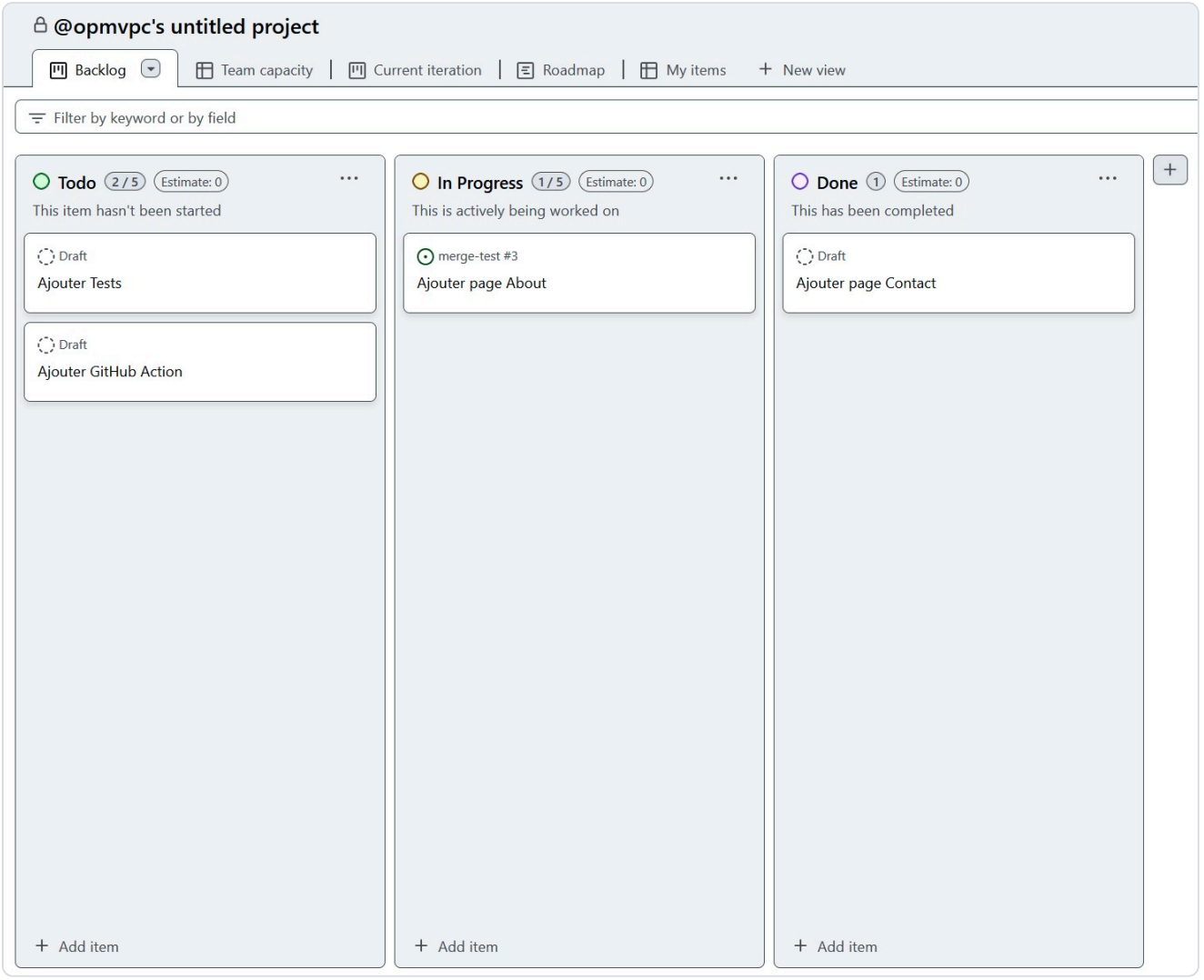
1. Dans votre dépôt GitHub, cliquez sur "Projects"
2. Choisissez "New Project"
3. Sélectionnez un modèle de base ou commencez à zéro



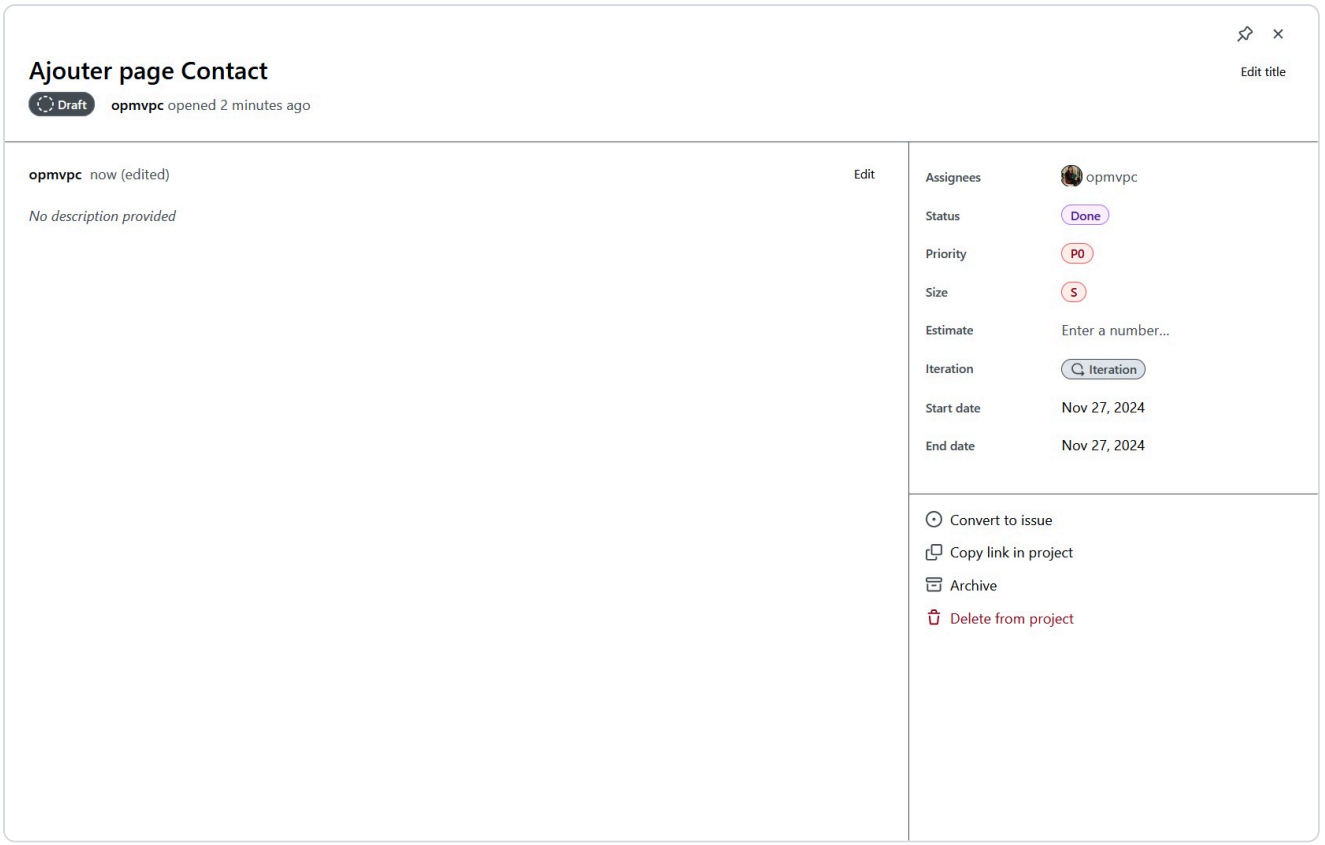
Configuration typique des colonnes :

- "À faire" : Tâches à planifier
- "En cours" : Développements actifs
- "Terminé" : Tâches réalisées

Exemple visuel des colonnes :



Modification d’une tâche



Issues GitHub : Gestion des Tâches et Bugs

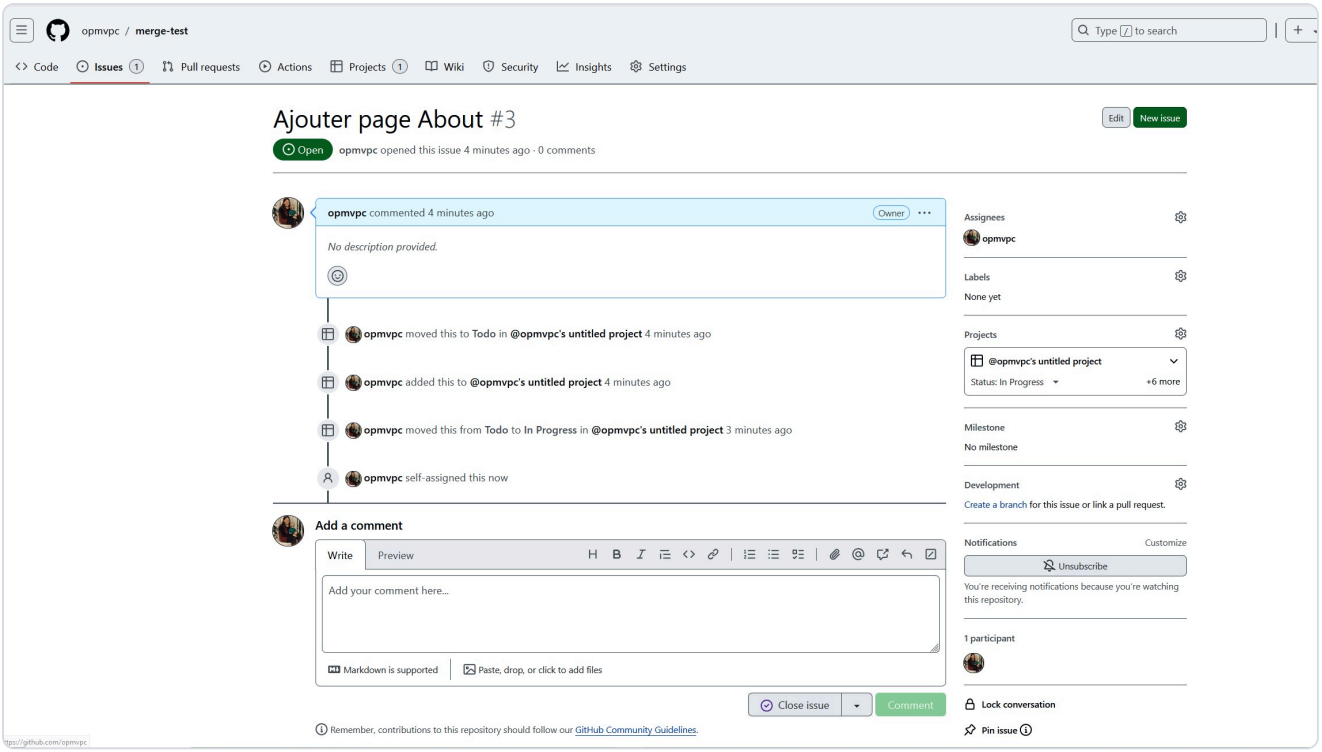
Qu'est-ce qu'une Issue ?

Une issue GitHub est un outil de communication et de suivi dans un projet. Elle permet de :

- Signaler un bug
- Proposer une amélioration
- Suivre une tâche à réaliser
- Discuter d'un point technique

Types d'issues courants :

- Bug : Problème technique à résoudre
- Amélioration : Nouvelle fonctionnalité
- Tâche : Travail à accomplir
- Documentation : Mise à jour de la documentation



Création et Gestion des Issues

Informations importantes lors de la création :

- Titre clair et concis
- Description détaillée
- Contexte du problème
- Étapes de reproduction (pour les bugs)
- Résultat attendu

Exemple d'une bonne issue :

```yaml
Titre: Correction de la validation d'email dans le formulaire d'inscription

Description:
Actuellement, le formulaire accepte des emails invalides.

Étapes de reproduction :
1. Aller sur la page d'inscription
2. Entrer un email sans format valide
3. Soumettre le formulaire

Résultat attendu :
Le formulaire doit rejeter les emails mal formatés
```

Gestion de l'issue :

- Assignation à un développeur
- Définition d'une priorité
- Suivi de son avancement

Étiquettes (Labels)

Rôle des Labels

Les labels sont des étiquettes colorées qui permettent de :

- Catégoriser rapidement une issue
- Communiquer visuellement son statut
- Organiser les tâches

Types de Labels Courants

Labels de priorité :

- `priorité: haute`
- `priorité: moyenne`
- `priorité: basse`

Labels de type :

- `bug`
- `amélioration`
- `documentation`
- `question`

Labels de statut :

- `en cours`
- `bloqué`
- `nécessite des informations`

Création et Utilisation des Labels

Bonnes pratiques :

- Utilisez des couleurs distinctives
- Créez des labels personnalisés adaptés à votre projet
- Soyez cohérent dans leur utilisation
- Limitez le nombre de labels pour éviter la confusion

Exemple de configuration de labels :

```
🔴 bug (couleur rouge)
🟡 amélioration (couleur jaune)
🟢 documentation (couleur verte)
```

Exercice pratique : gestion collaborative de projets avec Git et Laravel

Introduction à l'exercice

Dans cet exercice, vous travaillerez par groupes de deux pour développer une application Laravel utilisant Breeze. Vous apprendrez à gérer des branches Git, à écrire des tests HTTP avec Pest, à créer et résoudre des conflits Git, et à configurer l'intégration continue avec GitHub Actions.

Mise en place du projet

1. **Initialisation du dépôt Git :**
   - Un étudiant crée un nouveau projet Laravel
   - Ensuite, il le push sur GitHub
   - L'autre étudiant clone le dépôt localement.

Création des fonctionnalités

1. **Création de branches :**
   - Chaque étudiant crée une branche à partir de `main`, par exemple, `feature/about-page` et `feature/contact-page`.
2. **Développement de nouvelles pages et contrôleurs :**
   - **Étudiant A :** Crée une page "About" avec un contrôleur `PageController`.

     PHP
     ```php
     // web.php
     Route::get('/about', [PageController::class, 'showAbout']);

     // PageController.php
     public function showAbout()
     {
         return view('about');
     }
     ```
   - **Étudiant B :** Crée une page "Contact" avec le même contrôleur `PageController`.

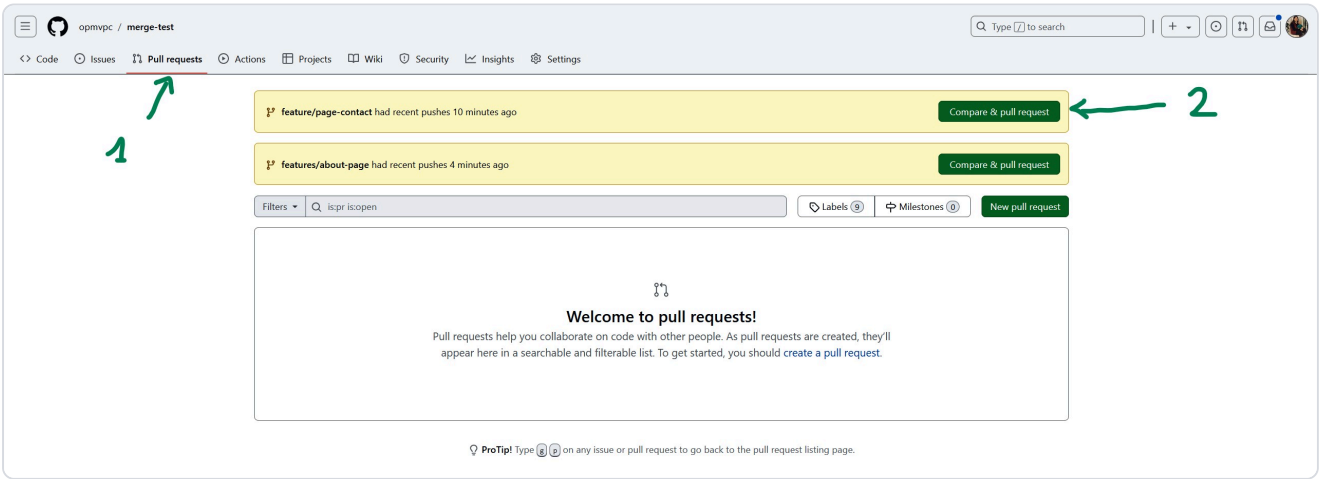
```php
// web.php
Route::get('/contact', [PageController::class, 'showContact']);

// PageController.php
public function showContact()
{
    return view('contact');
}
```

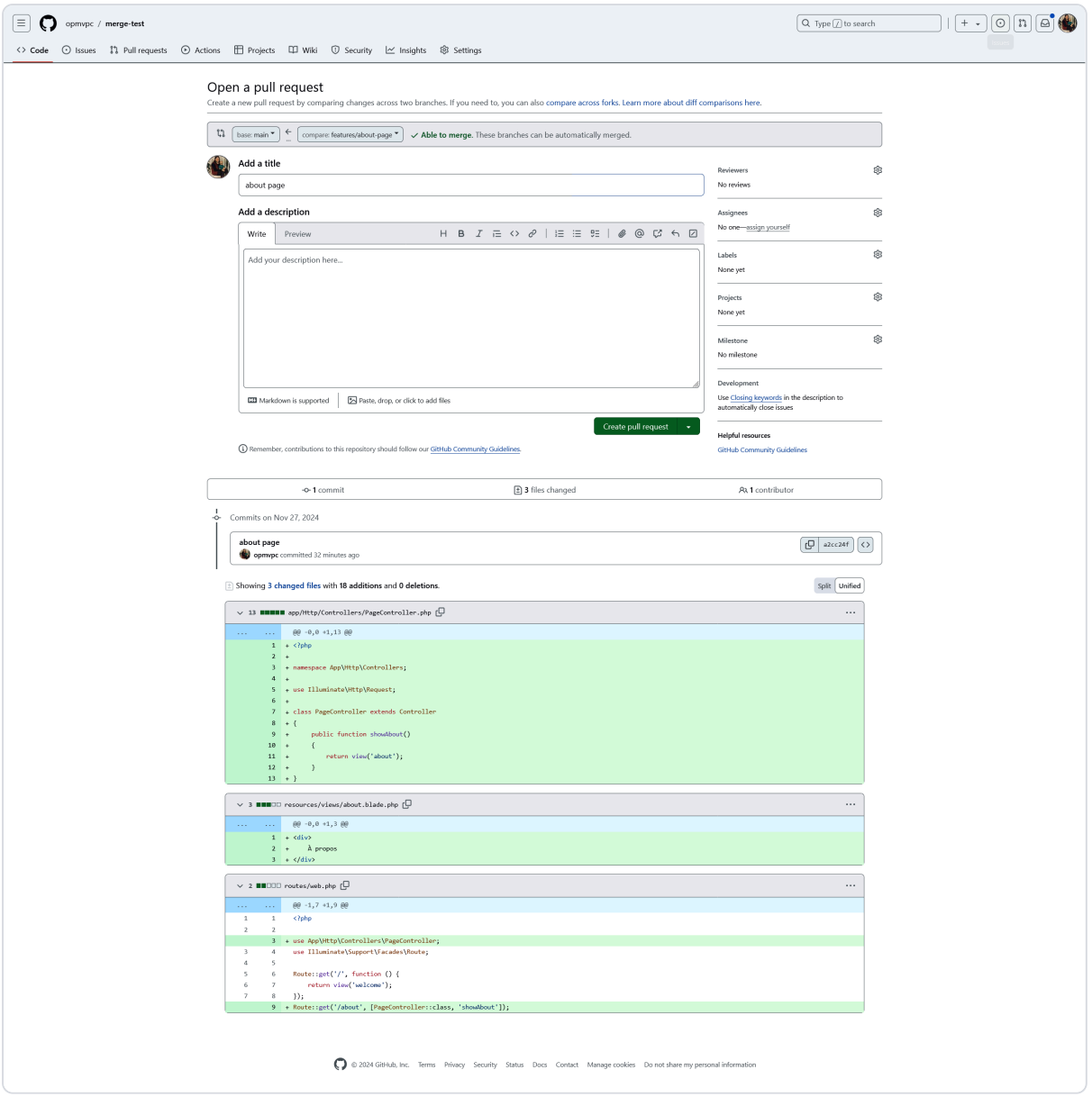
Simulation et résolution de conflits

1. **Création de Pull Requests :**
   - Chaque étudiant soumet une Pull Request de sa branche vers `main`.
2. **Scénarios de conflits :**
   - Les deux étudiants ont modifié les fichiers `web.php` et `PageController.php`, ce qui crée des conflits lors du merge des Pull Requests.
3. **Processus de résolution de conflits :**
   - Ouvrez les fichiers en conflit.
   - Discutez des modifications à conserver et résolvez les conflits en intégrant les routes et méthodes de chaque branche.
   - Validez les modifications et complétez le merge.

Création d'une Pull Request

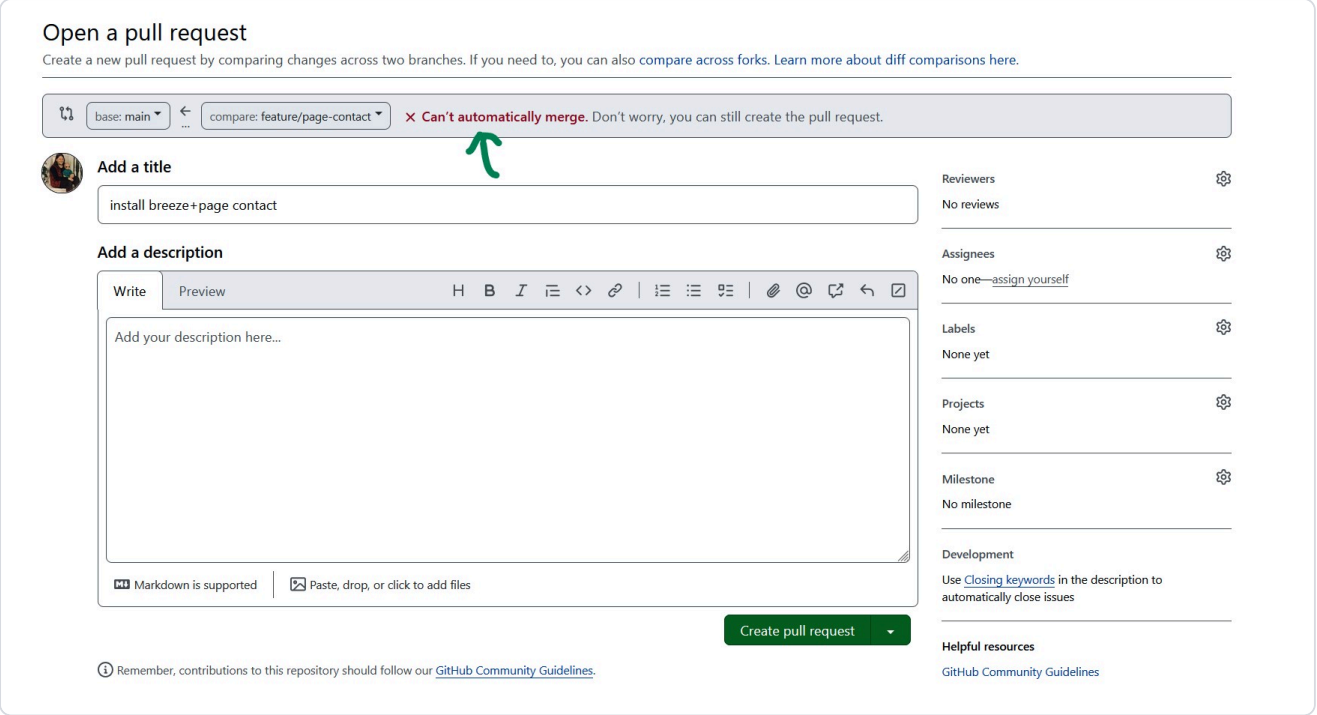


Créer la première Pull Request



Validez les changements et lancez le Merge

Créez la 2e Pull Request et essayez de Merge



Écriture des tests avec Pest

1. **Installez Breeze avec l'option Pest pour les tests.**
2. **Création de tests HTTP :**
   - **Test de la page About :**

```php
test('la page About se charge correctement', function () {
    response=response = response=this→get('/about');
    response->assertStatus(200);response→assertStatus(200);
    response->assertStatus(200);response→assertSee('À propos');
});
```

   - **Test de la page Contact :**

```php
test('la page Contact se charge correctement', function () {
    response=response = response=this→get('/contact');
    response->assertStatus(200);response→assertStatus(200);
    response->assertStatus(200);response→assertSee('Contactez-nous');
});
```

3. **Lancez les tests avec la commande `php artisan test` et corrigez votre code si besoin.**

Intégration continue avec GitHub Actions

1. **Configuration du workflow :**

- Créez un fichier `.github/workflows/laravel.yml` avec le contenu suivant :

```yaml
name: Run Laravel Tests

on: [push, pull_request]

jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: "8.3"

      - name: Install Dependencies
        run: composer install -q --no-ansi --no-interaction --no-scripts --no-progress --prefer-dist

      - name: Setup Node
        uses: actions/setup-node@v2
        with:
          node-version: "22"

      - name: Install NPM Dependencies
        run: npm install

      - name: Build Assets
        run: npm run build

      - name: Copy .env
        run: php -r "file_exists('.env') || copy('.env.example', '.env');"
      - name: Install Dependencies
        run: composer install -q --no-ansi --no-interaction --no-scripts --no-progress --prefer-dist
      - name: Generate key
        run: php artisan key:generate
      - name: Directory Permissions
        run: chmod -R 777 storage bootstrap/cache
      - name: Create Database
        run: |
          mkdir -p database
          touch database/database.sqlite
          php artisan migrate --force
      - name: Execute tests (Unit and Feature tests) via PHPUnit/Pest
        env:
          DB_CONNECTION: sqlite
          DB_DATABASE: database/database.sqlite
        run: php artisan test
```

2. **Fonctionnement du workflow :**
    - Le workflow s'exécute à chaque push ou pull request, vérifiant que les tests passent correctement.
3. **Pushez votre code et voyez si les tests passent.**

1.4 Comment installer un projet Laravel existant sur un nouvel ordinateur

Ce chapitre vous guidera à travers les étapes nécessaires pour installer un projet Laravel existant sur un nouvel ordinateur. Nous supposerons que vous avez déjà installé les outils requis et que vous utilisez un serveur de développement local comme Laragon ou Laravel Valet/Herd.

Prérequis

Avant de commencer, assurez-vous que les éléments suivants sont installés sur votre ordinateur :

- **PHP** : Le langage de programmation sur lequel Laravel est construit, avec les extensions requises.
- **Composer** : Un gestionnaire de dépendances pour PHP, utilisé pour installer les packages nécessaires.
- **Node.js et npm** : Node.js est un environnement d'exécution JavaScript, et npm est son gestionnaire de packages. Ils sont utilisés pour compiler les assets front-end de Laravel.
- **Un serveur de développement local** : Outils comme **Laragon** ou **Laravel Valet/Herd** facilitent le développement local en gérant les configurations de serveur.

Étapes d'installation

1. Cloner le dépôt du projet Laravel

La première étape consiste à obtenir une copie du code source du projet. Vous pouvez le faire en clonant le dépôt Git du projet :

```bash
git clone https://github.com/nom-utilisateur/nom-du-projet.git
```

Cette action télécharge le code source du projet sur votre machine locale, vous permettant de travailler avec le même code que celui du dépôt distant.

2. Naviguer vers le répertoire du projet

Après avoir cloné le dépôt, déplacez-vous dans le répertoire du projet :

```bash
cd nom-du-projet
```

Cela vous positionne dans le dossier du projet où vous exécuterez les commandes suivantes.

3. Installer les dépendances PHP avec Composer

Installez les packages PHP nécessaires en exécutant :

```bash
composer install
```

Cette commande lit le fichier `composer.json` et installe toutes les bibliothèques et dépendances PHP dont le projet a besoin pour fonctionner.

4. Copier le fichier `.env.example` en `.env`

Le fichier `.env` contient les variables d'environnement spécifiques à votre application, comme les informations de base de données. Créez-en une copie à partir de l'exemple fourni :

```bash
cp .env.example .env
```

Cette action crée un fichier `.env` que vous pourrez personnaliser pour votre environnement local.

5. Générer une nouvelle clé d'application

Générez une clé unique pour votre application Laravel :

```bash
php artisan key:generate
```

La clé d'application est utilisée pour sécuriser les données chiffrées et les sessions. Cette commande met à jour la variable `APP_KEY` dans votre fichier `.env`.

6. Configurer les paramètres de base de données

Ouvrez le fichier `.env` avec un éditeur de texte et configurez les informations de connexion à votre base de données :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=nom_de_votre_base_de_donnees
DB_USERNAME=votre_nom_d_utilisateur
DB_PASSWORD=votre_mot_de_passe
```

Ces paramètres permettent à Laravel de se connecter à votre base de données pour effectuer des opérations comme les migrations et les requêtes.

7. Migrer la base de données

Créez les tables nécessaires dans votre base de données en exécutant :

```bash
php artisan migrate
```

Les migrations sont des scripts qui créent ou modifient la structure de la base de données. Si la base de données spécifiée n'existe pas, la commande vous proposera de la créer.

8. Peupler la base de données avec des seeders (si disponibles)

Si le projet contient des seeders pour insérer des données initiales, exécutez :

```bash
php artisan db:seed
```

Les seeders remplissent la base de données avec des données fictives ou initiales, utiles pour les tests ou pour démarrer l'application avec des données par défaut.

9. Créer un lien symbolique pour le stockage (si nécessaire)

Si le projet utilise le stockage de fichiers (par exemple, pour les uploads), créez un lien symbolique vers le répertoire public :

```bash
php artisan storage:link
```

Cette commande crée un lien symbolique entre `storage/app/public` et `public/storage`, permettant un accès web aux fichiers stockés.

10. Installer les dépendances Node.js

Installez les packages front-end nécessaires avec npm :

```bash
npm install
```

Cette commande lit le fichier `package.json` et installe toutes les dépendances JavaScript requises pour le front-end de l'application.

11. Compiler les assets front-end

Compilez les fichiers CSS et JavaScript du projet :

- Pour le développement (avec source maps et sans minification) :

  ```bash
  npm run dev
  ```

- Pour la production (optimisé et minifié) :

  ```bash
  npm run build
  ```

La compilation des assets génère les fichiers optimisés nécessaires pour le front-end de l'application, en combinant et en minifiant le code pour de meilleures performances.

12. Démarrer le serveur local

Avec votre serveur de développement local (Laragon, Laravel Valet/Herd), votre application devrait être automatiquement reconnue et servie. Assurez-vous que votre serveur est en cours d'exécution et accédez à l'application via votre navigateur web à l'adresse appropriée, par exemple :

```bash
http://nom-du-projet.test
```

Cette étape vous permet de visualiser et de tester votre application Laravel dans un environnement local.

Conclusion

Vous avez maintenant installé avec succès un projet Laravel existant sur votre nouvel ordinateur. Ce processus vous permet de configurer rapidement un environnement de développement local pour travailler sur le projet. N'oubliez pas de consulter la documentation du projet pour d'éventuelles instructions supplémentaires spécifiques à celui-ci.

1.5 Système de Gestion pour l'Atelier 404

Contexte

La section des techniciens en informatique a besoin d'un logiciel professionnel pour gérer son **Atelier 404**, le repair café étudiant ouvert au public. Cet espace permet aux citoyens de faire réparer gratuitement leurs équipements informatiques par nos étudiants, sous supervision.

Face à l'afflux croissant de demandes, un système de gestion des interventions est devenu indispensable pour organiser efficacement le travail des techniciens-étudiants et assurer un suivi de qualité.

Objectif du projet

Développer une application web complète permettant de gérer les interventions de l'Atelier 404, depuis la prise de contact jusqu'à la clôture de la réparation, avec Laravel (Blade + Breeze).

Cahier des charges fonctionnel

Page d'accueil publique

- **Présentation de l'Atelier 404** : texte descriptif, horaires, services proposés
- **Formulaire de contact** :
  - Champs : nom, email, téléphone, type d'appareil, description du problème
  - Si validé correctement → création automatique d'un client et d'une intervention
  - Notification à l'admin de la nouvelle demande
- **Informations pratiques** : localisation, FAQ basique

Authentification et rôles

- Système d'authentification via Laravel Breeze
- Deux types d'utilisateurs : **Admin** et **Technicien**
- Gestion des permissions selon le rôle

Gestion des clients

- Création automatique via formulaire de contact (si champs valides)
- Création manuelle par admin uniquement
- Informations : nom, email, téléphone, adresse (optionnelle)
- Consultation de l'historique des interventions par client

Gestion des interventions

- **Création** : automatique (via formulaire) ou manuelle (admin)
- **Informations** : client, description, type d'appareil, priorité, date prévue
- **Statuts** : Nouvelle demande → Diagnostic → En réparation → Terminé → Non réparable
- **Images** : upload multiple (photos de l'appareil, du problème)
- **Attribution** : assignation à un technicien-étudiant
- **Notes internes** : commentaires techniques non visibles au public

Tableaux de bord

- **Admin** :
  - Vue globale toutes interventions
  - Statistiques (interventions par statut, par technicien, par type d'appareil)
  - Demandes de contact non traitées
- **Technicien** :
  - Ses interventions assignées uniquement
  - Possibilité d'ajouter des notes et images

Fonctionnalités additionnelles

- Recherche d'interventions (client, statut, technicien, type d'appareil)
- Filtrage par dates et priorités
- Export CSV de la liste des interventions (admin only)
- Pagination des listes
- Messages flash pour les actions utilisateur

Exigences techniques

Stack obligatoire

- Laravel avec Blade (pas de framework JS séparé)
- Laravel Breeze pour l'authentification
- Base de données relationnelle (MySQL/SQLite)
- PHP 8.2+

Sécurité et autorisations

- Protection CSRF sur tous les formulaires
- Validation côté serveur (notamment formulaire public)

- Gates/Policies Laravel obligatoires :
  - Seuls les admins peuvent modifier/supprimer des clients
  - Les techniciens ne voient que leurs interventions
  - Seul l'admin peut réassigner une intervention
  - Le formulaire public ne nécessite pas d'authentification

Gestion des fichiers

- Upload d'images dans le storage public
- Validation des types (images uniquement)
- Taille maximale : 5MB par fichier
- Génération de thumbnails pour l'affichage liste

Tests automatisés avec Laravel Dusk

Couverture de tests requise

Les tests Dusk devront couvrir au minimum :

- Soumission du formulaire de contact public
- Création automatique client/intervention
- Processus complet de gestion d'intervention
- Vérification des permissions par rôle
- Upload et suppression d'images
- Changements de statut
- Recherche et filtrage
- Cas d'erreur et validation

Organisation des tests

- Un fichier de test par fonctionnalité majeure
- Utilisation des factories pour les données
- Base de test dédiée

Analyse et conception

Diagramme UML

Fournir un diagramme de classes UML complet incluant :

- Toutes les entités et leurs attributs
- Les relations et multiplicités
- Les méthodes principales
- Les énumérations (statuts, rôles, priorités)

Modalités de remise

Travail en équipe

- **Binômes obligatoires** (2 étudiants)
- Utilisation de Git avec historique significatif
- Branches séparées par fonctionnalité

Livrables

Repository GitHub contenant :

- Code source complet
- Diagramme UML (format PDF ou image)
- Suite de tests Dusk fonctionnels
- Fichier `.env.example` configuré
- Seeders pour données de démonstration

Échéance

lundi 17 novembre 2025 à 23h59 sur Teams (lien GitHub uniquement)

Exercices supplémentaires

- Historique des interventions
  - Enregistrez et affichez toutes les modifications apportées à une intervention (changement de status, assignation à un technicien, qui a fait la modification, ...)
  - Mettre en ligne l'application
  - Envoyer des notifications par email

1.6 Laravel Dusk

Introduction à Laravel Dusk

Qu'est-ce que Laravel Dusk ?

Laravel Dusk est un outil de test automatisé pour les interfaces utilisateurs, spécialement conçu pour les applications Laravel. Il permet de simuler les interactions des utilisateurs avec une application web dans un navigateur réel, en utilisant ChromeDriver pour manipuler les éléments du navigateur. Cette approche assure que les tests reflètent fidèlement les actions des utilisateurs.

Avantages de l'utilisation de Laravel Dusk

L'utilisation de Laravel Dusk présente plusieurs bénéfices clés pour le développement d'applications web :

- **Tests automatisés de bout en bout** : Dusk facilite l'automatisation des interactions utilisateurs, telles que cliquer sur des liens ou remplir des formulaires.
- **Intégration fluide** : Étant intégré dans l'écosystème Laravel, Dusk est particulièrement accessible pour ceux qui sont déjà familiers avec ce framework.
- **Débogage efficace** : La fonctionnalité de capture d'écran de Dusk permet de détecter rapidement les erreurs.
- **Sécurité renforcée** : En testant des scénarios d'interaction utilisateur réels, Dusk aide à identifier et corriger les vulnérabilités avant le déploiement.

Nous explorerons ensuite comment installer Laravel Dusk, configurer l'environnement de test et créer des tests pour garantir la qualité et la robustesse de vos applications w

Installation de Laravel Dusk

Prérequis

Avant d'installer Laravel Dusk, assurez-vous que votre environnement de développement répond aux exigences suivantes :

- **Laravel** : Une installation existante de Laravel, de préférence la dernière version pour bénéficier de toutes les fonctionnalités et améliorations de sécurité.
- **Composer** : Utilisé pour gérer les dépendances de Laravel, y compris l'installation de Dusk.
- **Navigateur Chrome** : Nécessaire pour ChromeDriver, qui est utilisé par Dusk pour automatiser les tests dans un navigateur.

Étapes d'installation

Pour installer Laravel Dusk, suivez ces étapes simples :

1. **Ajout de Dusk à votre projet Laravel** : Ouvrez votre terminal et exécutez la commande suivante à la racine de votre projet Laravel :

   ```bash
   composer require --dev laravel/dusk
   ```

   Cette commande installe Laravel Dusk comme une dépendance de développement.

2. **Configuration de Dusk** : Après l'installation, exécutez la commande suivante pour publier les assets de Dusk et configurer votre projet :

   ```bash
   php artisan dusk:install
   ```

   Cette commande crée le répertoire `tests/Browser` où vous placerez vos tests de navigateur, et configure également quelques autres fichiers essentiels pour le fonctionnement de Dusk.

3. **Mise à jour des variables d'environnement** : Assurez-vous de configurer l'environnement de test approprié dans votre fichier `.env` pour éviter toute interaction avec vos données de production lors de l'exécution des tests.

Avec Laravel Dusk correctement installé et configuré, vous êtes prêt à commencer à créer des tests automatisés pour votre application. La section suivante détaillera comment configurer votre premier test et utiliser efficacement Dusk pour améliorer la qualité de votre application.

Utilisation de Laravel Dusk

Configuration initiale

Avant de commencer à écrire des tests avec Laravel Dusk, il est essentiel de comprendre la structure de base et la configuration initiale :

1. **Dossier des tests** : Les tests Dusk se trouvent dans le dossier `tests/Browser`. Chaque fichier de test doit étendre la classe `BrowserTestCase`, qui fournit diverses méthodes pour interagir avec le navigateur.

2. **Fichier DuskTestCase** : Le fichier `DuskTestCase.php` dans le dossier `tests` est le point de départ de tous les tests Dusk. Il contient la configuration nécessaire pour démarrer et fermer le navigateur. Veillez à vérifier que le chemin du pilote ChromeDriver est correctement configuré ici.

Création de tests basiques

Pour créer un test simple avec Laravel Dusk, procédez comme suit :

1. **Création d'un nouveau test** : Utilisez Artisan pour générer un nouveau test :

   ```bash
   php artisan dusk:make UnTest
   ```

   Cela créera un nouveau fichier de test dans `tests/Browser` nommé `UnTest.php`.

2. **Écriture d'un test** : Ouvrez `UnTest.php` et ajoutez une fonction de test. Par exemple, pour tester si la page d'accueil de votre application charge correctement :

   ```php
   it('displays the Laravel text on the home page', function () {
       this->browse(function(this→browse(function (this->browse(function(browser) {
           $browser→visit('/')
                   →assertSee('Laravel');
       });
   });
   ```

   Cette fonction utilise le navigateur pour visiter la page d'accueil (`/`) et vérifie que le texte 'Laravel' est présent.

Exécution des tests

Pour exécuter vos tests Dusk, utilisez la commande suivante dans votre terminal :

```bash
php artisan dusk
```

Cette commande lancera tous les tests situés dans le dossier `tests/Browser`. Vous pouvez également spécifier un chemin vers un test spécifique si vous souhaitez exécuter un seul test.

En suivant ces étapes, vous pouvez configurer, écrire et exécuter des tests basiques avec Laravel Dusk. La section suivante abordera des fonctionnalités plus avancées pour maximiser l'efficacité de vos tests automatisés.

Fonctionnalités avancées de Laravel Dusk

Tests avec authentification

Laravel Dusk facilite le test des fonctionnalités qui nécessitent une authentification utilisateur. Voici comment vous pouvez tester les scénarios authentifiés :

1. **Authentification d'un utilisateur** : Vous pouvez utiliser la méthode `loginAs` pour simuler l'authentification d'un utilisateur avant de réaliser des actions spécifiques :

```php
use App\Models\User;

it('allows authenticated user to see the dashboard', function () {
    this->browse(function(this→browse(function (this->browse(function(browser) {

user=User::find(1);//Assurez-vousquecetutilisateurexistedansvotrebasededonne´esdete
stuser = User::find(1); // Assurez-vous que cet utilisateur existe dans votre base
de données de test

user=User::find(1);//Assurez-vousquecetutilisateurexistedansvotrebasededonne´esdete
stbrowser→loginAs($user)
                →visit('/dashboard')
                →assertSee('Dashboard');
    });
});
```

Interaction avec JavaScript

Laravel Dusk est particulièrement utile pour tester des applications qui utilisent intensivement JavaScript. Voici comment interagir avec des éléments JS :

1. **Cliquer sur des éléments JavaScript** : Par exemple, pour cliquer sur un bouton qui ouvre un modal via JavaScript :

```php
it('displays the correct content in the JavaScript modal', function () {
    this->browse(function(this→browse(function (this->browse(function(browser) {
        $browser→visit('/page-with-modal')
                →click('#open-modal-button') // Assurez-vous que l'ID est correct
                →waitForText('Modal Content')
                →assertSee('Modal Content');
    });
});
```

Capture d'écran et débogage

Laravel Dusk permet de prendre des captures d'écran en cas d'échec des tests, ce qui est très utile pour le débogage :

1. **Prendre une capture d'écran** : Vous pouvez configurer Dusk pour prendre une capture d'écran automatiquement lorsqu'un test échoue ou manuellement dans vos scripts de test :

```php
it('captures a screenshot and verifies non-existent content', function () {
    this->browse(function(this→browse(function (this->browse(function(browser) {
        $browser→visit('/some-page')
                →screenshot('name-of-screenshot') // Sauvegarde la capture d'écran dans le dossier tests/Browser/screenshots
                →assertSee('Some content that does not exist');
    });
});
```

En maîtrisant ces fonctionnalités avancées, vous pouvez étendre significativement la couverture de vos tests automatisés, assurant une plus grande stabilité et fiabilité de vos applications web. La conclusion suivante résumera les points clés abordés et fournira des ressources supplémentaires pour approfondir votre maîtrise de Laravel Dusk.

Configuration d’un environnement dédié à Dusk

Création d’un `.env.dusk.local`

Pour éviter de polluer votre environnement de développement standard (déjà suffisamment bancal, j’imagine), il est conseillé de créer un fichier `.env.dusk.local`. Celui-ci contiendra une configuration spécifique exclusivement réservée à vos tests Dusk. Par exemple :

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:0hDqTLQR7MnO4xtc7cFXeiTzZQ7PUj2eijGpU4cbYzA=
APP_DEBUG=true
APP_TIMEZONE=UTC
APP_URL=http://dusk.test

APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US

APP_MAINTENANCE_DRIVER=file

PHP_CLI_SERVER_WORKERS=4

BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=sqlite
DB_DATABASE="C:/laragon/www/dusk/database/test.sqlite"

SESSION_DRIVER=database
SESSION_LIFETIME=120
SESSION_ENCRYPT=false
SESSION_PATH=/
SESSION_DOMAIN=null

BROADCAST_CONNECTION=log
FILESYSTEM_DISK=local
QUEUE_CONNECTION=database

CACHE_STORE=database
CACHE_PREFIX=

MEMCACHED_HOST=127.0.0.1

REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=log
MAIL_HOST=127.0.0.1
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
```

```
MAIL_FROM_NAME="APP_NAME" AWS_ACCESS_KEY_ID= AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1 AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false VITE_APP_NAME="{APP_NAME}"

DUSK_HEADLESS_DISABLED=true
```

Points-clés :

- **URL dédiée** : Ajustez `APP_URL` afin d'utiliser un hôte virtuel créé par Laragon, Laravel Herd ou un autre outil. Cela permettra à vos tests Dusk de s'exécuter proprement sur un environnement isolé, au lieu de polluer votre configuration principale.
- **Base de données SQLite dédiée** : La ligne `DB_DATABASE="C:/laragon/www/dusk/database/test.sqlite"` est un exemple de configuration pour utiliser une base de données SQLite réservée aux tests. Créez manuellement ce fichier. Ainsi, vous n'aurez pas à exécuter vos migrations complètes à chaque test, ce qui accélère le processus.
- **DUSK_HEADLESS_DISABLED=true** : Mettre cette variable à `true` permet de voir Chrome s'ouvrir pendant les tests. Parfait pour les âmes inquiètes qui ont besoin de constater visuellement le désastre qu'elles sont en train de commettre. Ajoutez des `->pause(1000)` dans vos tests pour ralentir l'exécution et observer ce qu'il se passe.

Nettoyage de la base de données entre chaque test

Pour éviter que les données s'accumulent et donner l'illusion que vous savez garder un environnement propre, vous pouvez utiliser le `DatabaseTruncation`. Cela évite de devoir relancer toutes les migrations et les seeders entre chaque test, et donc de perdre un temps précieux à réimporter vos données.

Dans votre fichier `tests/Pest.php`, ajoutez :

```php
pest()->extend(Tests\DuskTestCase::class)
    ->use(Illuminate\Foundation\Testing\DatabaseTruncation::class)
    // ->use(Illuminate\Foundation\Testing\DatabaseMigrations::class)
    ->in('Browser');
```

`DatabaseTruncation` va vider les tables entre chaque test, ce qui est beaucoup plus rapide qu'une migration complète.

Exemple de test avec pauses

Voici un exemple de test, parce qu'apparemment il vous faut tout sous le nez :

```php
test('example', function () {
    user=User::factory()->create();user = User::factory()→create();

    user=User::factory()->create();this→browse(function (Browser browser)use(browser)
use (browser)use(user) {
        browser->loginAs(browser
            →loginAs(browser->loginAs(user)
            →visit('/profile')
            →pause(1000) // On attend 1 seconde pour admirer le "chef-d'œuvre"
            →assertSee('Profile Information');
    });
});
```

Maintenant, vous avez tout ce qu'il vous faut pour créer un environnement Dusk propre, dédié, et observer en direct les résultats de vos tests. Utilisez `.env.dusk.local`, configurez une base de données test, nettoyez-la entre chaque exécution, et réglez votre Chrome pour qu'il n'exécute pas vos tests en mode invisible, histoire que vous puissiez enfin voir de vos yeux vos échecs majestueux. Enjoy.

Gestion de ChromeDriver

- **Détection automatique de la version** :

  ```bash
  php artisan dusk:chrome-driver --detect
  ```

  Cette commande tente de détecter la version de Chrome installée sur votre machine et télécharge la version correspondante de ChromeDriver. Parfait pour les rois de l'improvisation.

- **Installer la dernière version compatible** :

  ```bash
  php artisan dusk:chrome-driver
  ```

  Sans argument, cette commande télécharge la version de ChromeDriver la plus adaptée à votre environnement.

- **Forcer l'installation d'une version spécifique** :

  ```bash
  php artisan dusk:chrome-driver <version>
  ```

  Remplacez `<version>` par un numéro de version pour imposer vos choix rétrogrades (ex : `114`).

- **Installer toutes les versions disponibles** :

```bash
php artisan dusk:chrome-driver --all
```

Vous voulez tout télécharger, juste au cas où ? Soyez mon invité. C'est clairement excessif, mais bon, faites donc.

- **Aide et options supplémentaires** :

```bash
php artisan dusk:chrome-driver --help
```

Si vous êtes encore perdus, c'est ici que ça se passe. Lire la doc, quelle idée révolutionnaire !

Conclusion

Résumé des points clés

Laravel Dusk est un puissant outil de test automatisé pour les applications Laravel, permettant de simuler des interactions utilisateurs dans un navigateur réel. Nous avons exploré son installation, la configuration initiale, et la création de tests de base. Nous avons également abordé des fonctionnalités avancées telles que les tests avec authentification, l'interaction avec JavaScript, et la capture d'écran pour le débogage.

Ressources supplémentaires pour approfondir

Pour ceux qui souhaitent approfondir leurs connaissances et compétences avec Laravel Dusk, voici quelques ressources utiles :

- **Documentation officielle de Laravel Dusk** : La source la plus complète et à jour pour apprendre à utiliser Laravel Dusk. Disponible sur le site web de Laravel.
- **Cours et tutoriels en ligne** : De nombreux cours en ligne offrent des tutoriels détaillés sur Laravel Dusk, souvent avec des exemples pratiques et des projets.
- **Communautés et forums** : Rejoindre des communautés en ligne, comme Stack Overflow ou les forums Laravel, peut être très bénéfique pour obtenir de l'aide, partager des expériences et apprendre des meilleures pratiques.

En intégrant Laravel Dusk dans votre flux de travail de développement, vous pouvez non seulement améliorer la qualité et la performance de vos applications, mais aussi anticiper et résoudre les problèmes avant qu'ils n'affectent les utilisateurs finaux. Cela fait de Laravel Dusk un investissement précieux pour tout projet Laravel soucieux de maintenir un haut standard de qualité.

Liens utiles

- https://laravel.com/docs/11.x/dusk
- Exemple de code : https://github.com/opmvpc/test-dusk

CHAPITRE 2

Inertia.js et les Nouveaux Laravel Starter Kits : Introduction



- **Concept** : Inertia.js demeure cette bibliothèque géniale qui fait le lien entre ton backend Laravel chéri et les frameworks frontend (Vue.js, React, Svelte).
- **Fonctionnement** : Toujours pas besoin d'API REST séparée, tu gardes tes routes et contrôleurs Laravel comme un vrai artisan. ✨
- **Avantages** :
  - **Approche Hybride** : Le meilleur des deux mondes - la puissance de Laravel + la réactivité des interfaces modernes.

- **Développement Simplifié** : Zéro API à gérer = moins de prises de tête (même pour toi 😉).
- **Expérience Utilisateur Premium** : Tes applis réagissent comme des SPAs de compétition.

Laravel Starter Kits : La Nouvelle Ère (Bye Bye Jetstream 👋)

Mars 2025 a marqué un tournant : Laravel a **viré Jetstream ET Breeze** au profit de **3 starter kits dédiés**, chacun avec son propre repo standalone. Fini le package qu'on installe après coup, maintenant t'as **tout le code dans ton projet dès le début**.

Les 3 Nouveaux Starter Kits :

1. **Vue Starter Kit** 💚 *(ton chouchou visiblement)*
   - Construit avec **Vue 3** + **Inertia.js**
   - Utilise **shadcn-vue** pour les composants UI
   - **Tailwind CSS v4** intégré
   - Parfait si tu kiffes la syntaxe Vue et ses Single File Components
2. **React Starter Kit** ⚛️
   - Basé sur **React** + **Inertia.js**
   - Composants **shadcn/ui** (Tailwind v4)
   - Pour ceux qui veulent l'écosystème React (les traîtres quoi 🤡)
3. **Livewire Starter Kit** ⚡
   - **Livewire** + **Blade** (full PHP, zéro JS)
   - Composants **Livewire Flux** (gratuits)
   - Pour les puristes qui veulent rester côté serveur

Fonctionnalités Incluses (Tous les Kits) :

- ✅ Authentification complète (login, register, reset password)
- ✅ Gestion du profil utilisateur
- ✅ Dark mode support
- ✅ Multiples layouts
- ✅ Tailwind CSS v4

Options Supplémentaires :

- Vérification email (`MustVerifyEmail`)
- Confirmation de mot de passe (`password.confirm`)
- Intégration WorkOS (authentification tierce)

Pourquoi C'est Mieux que Jetstream/Breeze ? 🎯

1. **Application complète out-of-the-box** : Plus de package à installer, tout est déjà là.
2. **Simplicité** : Un starter kit = un stack. Fini le "choose your own adventure" qui te faisait perdre 3h.
3. **Personnalisation facile** : Tout le code est dans **ton** projet (`app/`, `resources/`), tu modifies ce que tu veux sans te soucier des mises à jour de package.
4. **Bibliothèques modernes** : shadcn, Flux... du matos pro dès le départ.

Installation Éclair ⚡

```bash
laravel new mon-projet-de-ouf
```

Et tu choisis ton starter kit. C'est tout. Même toi tu devrais y arriver (▀̿'̿-'̿)▀̿

Conclusion

Les nouveaux **Laravel Starter Kits** ont enterré Jetstream en beauté 🪦. Inertia.js reste au cœur des stacks **Vue** et **React**, offrant cette magie d'une SPA sans API. Que tu choisisses Vue (le bon choix 💅), React (mouais...) ou Livewire (le côté obscur), t'as maintenant des bases solides, modernes et **100% customisables**.

Développer avec Laravel en BIG 2025, c'est enfin aussi smooth qu'on te l'avait promis.

Ressources Officielles 📚

- Documentation Laravel Starter Kits
- Documentation Inertia.js
- Article de lancement des Starter Kits
- Vue Starter Kit - Repo GitHub

2.1 Pourquoi utiliser Inertia ?

Introduction Générale

Dans le développement web, combiner **Laravel** (backend) et **Vue.js** (frontend) est courant pour créer des applications dynamiques. Traditionnellement, ces deux parties communiquent via des requêtes AJAX, en utilisant des outils comme **Axios** ou **Fetch**.

Problèmes de la méthode traditionnelle :

- Duplication de logique (validation, gestion des données)
- Gestion complexe des états côté client et serveur
- Code répétitif pour gérer erreurs, chargement, succès...

Inertia.js propose une alternative moderne, agissant comme un "pont" entre Laravel et Vue.js. Il élimine le besoin d'API REST ou GraphQL et permet de créer des applications fluides et réactives sans sacrifier la simplicité d'un monolithe.

Dans ce chapitre, nous verrons :

1. La méthode traditionnelle avec Axios/Fetch, illustrée par des exemples
2. Les avantages d'Inertia.js et comment il simplifie ces interactions
3. Le nouveau composant `<Form>` d'Inertia (l'approche recommandée)
4. Le helper `useForm` pour un contrôle programmatique avancé

Méthode Traditionnelle

Dans une architecture classique, le frontend (Vue.js) communique avec le backend (Laravel) via des appels API. Ces interactions se font souvent à l'aide de bibliothèques comme **Axios** ou l'API native **Fetch**.

Récupérer les données d'une page

Prenons l'exemple d'une page qui affiche une liste d'articles. Avec la méthode traditionnelle :

1. Une requête est envoyée depuis le frontend pour récupérer les données
2. Les données reçues sont stockées dans une variable locale pour être affichées

Code exemple avec Axios :

```vue
<script setup>
import { ref, onMounted } from 'vue';
import axios from 'axios';

const articles = ref([]);

onMounted(async () => {
  const response = await axios.get('/api/articles'); // Requête API
  articles.value = response.data; // Stockage des données
});
</script>

<template>
  <div>
    <h1>Articles</h1>
    <ul>
      <li v-for="article in articles" :key="article.id">{{ article.title }}</li>
    </ul>
  </div>
</template>
```

Soumettre un formulaire

Pour envoyer un formulaire (par exemple, pour créer un nouvel article) :

1. Les données sont collectées et envoyées au backend via une requête POST
2. Le frontend doit gérer manuellement les erreurs de validation renvoyées par le backend

Code exemple avec Axios :

```vue
<script setup>
import { ref } from 'vue';
import axios from 'axios';

const formData = ref({ title: '', content: '' });
const errors = ref({});

const submitForm = async () => {
  try {
    const response = await axios.post('/api/articles', formData.value); // Requête
POST
    console.log(response.data); // Succès
  } catch (err) {
    errors.value = err.response.data.errors || {}; // Gestion des erreurs
  }
};
</script>

<template>
  <form @submit.prevent="submitForm">
    <input v-model="formData.title" placeholder="Title" />
    <textarea v-model="formData.content" placeholder="Content"></textarea>
    <button type="submit">Submit</button>
    <div v-if="errors.title">{{ errors.title }}</div>
    <div v-if="errors.content">{{ errors.content }}</div>
  </form>
</template>
```

Limites de cette méthode

- **Gestion manuelle des états** : Il faut écrire du code pour gérer les transitions entre états (chargement, succès, erreurs)
- **Duplication de logique** : La validation des données doit être implémentée à la fois côté backend (Laravel) et côté frontend (Vue.js)
- **Complexité croissante** : Avec plusieurs pages ou formulaires, la gestion devient plus difficile à maintenir

C'est pour répondre à ces défis qu'**Inertia.js** propose une approche différente, simplifiant considérablement ces interactions.

Introduction à Inertia.js

Inertia.js simplifie la gestion des interactions entre Laravel (backend) et Vue.js (frontend). Plutôt que de communiquer via des appels API explicites, Inertia agit comme un "pont" qui transmet directement les données et affiche les composants Vue.js sans recharger la page.

Fonctionnement d'Inertia

Avec Inertia :

1. Le backend génère des réponses qui incluent les données nécessaires et le composant Vue.js correspondant
2. Les transitions entre les pages se font sans rechargement complet, offrant une expérience utilisateur fluide
3. La logique est centralisée : la gestion des données, des erreurs et des transitions est simplifiée

Exemple : Récupérer les données d'une page

Inertia permet de transmettre les données directement du backend au frontend sans besoin de requêtes API explicites.

Backend Laravel :

```php
use Inertia\Inertia;

Route::get('/articles', function () {
    articles=Article::all();returnInertia::render('Articles',['articles'⇒articles =
Article::all();
    return Inertia::render('Articles', ['articles' ⇒
articles=Article::all();returnInertia::render('Articles',['articles'⇒articles]);
});
```

Frontend Vue.js :

```vue
<script setup>
defineProps(['articles']);
</script>

<template>
  <div>
    <h1>Articles</h1>
    <ul>
      <li v-for="article in articles" :key="article.id">{{ article.title }}</li>
    </ul>
  </div>
</template>
```

Dans cet exemple, Inertia transmet les données `$articles` directement au composant Vue, éliminant le besoin d'un appel API côté client.

Le Composant `<Form>` : L'Approche Moderne (2025+)

Depuis 2025, Inertia propose un **composant** `<Form>` qui simplifie drastiquement la création de formulaires. C'est désormais **l'approche recommandée** pour la majorité des cas d'usage.

Fonctionnement

Le composant `<Form>` se comporte comme un formulaire HTML classique, mais utilise Inertia en arrière-plan pour éviter les rechargements complets de page. **Pas besoin de** `v-model` ni de gestion manuelle des données : il suffit de donner un attribut `name` à chaque input.

Backend Laravel :

```php
use Illuminate\Http\Request;

Route::post('/articles', function (Request $request) {
    $validated = $request->validate([
        'title' => 'required|string|max:255',
        'content' => 'required',
    ]);

    Article::create($validated);

    return redirect()->route('articles'); // Redirection après succès
});
```

Frontend Vue.js :

```vue
<script setup>
import { Form } from '@inertiajs/vue3';
</script>

<template>
  <Form action="/articles" method="post">
    <input type="text" name="title" placeholder="Title" />
    <textarea name="content" placeholder="Content"></textarea>
    <button type="submit">Create Article</button>
  </Form>
</template>
```

✨ **C'est tout !** Pas de `v-model`, pas de gestion manuelle de state. Le composant gère automatiquement la soumission, les erreurs et les états.

Afficher les Erreurs et États

Le composant `<Form>` expose des **slot props** pour accéder aux erreurs, états de chargement, etc.

```vue
<script setup>
import { Form } from '@inertiajs/vue3';
</script>

<template>
  <Form
    action="/articles"
    method="post"
    #default="{ errors, processing, wasSuccessful }"
  >
    <input type="text" name="title" placeholder="Title" />
    <div v-if="errors.title">{{ errors.title }}</div>

    <textarea name="content" placeholder="Content"></textarea>
    <div v-if="errors.content">{{ errors.content }}</div>

    <button type="submit" :disabled="processing">
      {{ processing ? 'Creating...' : 'Create Article' }}
    </button>

    <div v-if="wasSuccessful" class="success">Article created! 🎉</div>
  </Form>
</template>
```

Fonctionnalités Avancées

Le composant `<Form>` supporte :

- **Structures de données imbriquées** : `name="report[description]"` ou `name="user.email"`
- **Upload de fichiers** : `<input type="file" name="documents" multiple />`
- **Transformation des données** : `:transform="data ⇒ ({ ...data, user_id: 123 })"`
- **Réinitialisation automatique** : `resetOnSuccess` ou `resetOnError`
- **Dotted key notation** : `name="user.profile.bio"` crée automatiquement `{ user: { profile: { bio: "..." } } }`

Le Helper `useForm` : Pour un Contrôle Programmatique

Si tu as besoin d'un contrôle plus fin sur tes formulaires (modifications dynamiques, logique complexe), Inertia fournit toujours le **helper** `useForm`.

Frontend Vue.js :

```vue
<script setup>
import { useForm } from '@inertiajs/vue3';

const form = useForm({
  title: '',
  content: '',
});

const submitForm = () => {
  form.post('/articles', {
    onSuccess: () => form.reset(), // Réinitialise après succès
  });
};
</script>

<template>
  <form @submit.prevent="submitForm">
    <input v-model="form.title" placeholder="Title" />
    <div v-if="form.errors.title">{{ form.errors.title }}</div>

    <textarea v-model="form.content" placeholder="Content"></textarea>
    <div v-if="form.errors.content">{{ form.errors.content }}</div>

    <button :disabled="form.processing" type="submit">
      {{ form.processing ? 'Submitting...' : 'Submit' }}
    </button>
  </form>
</template>
```

Quand utiliser `useForm` ?

- Tu as besoin de manipuler les données dynamiquement avant soumission
- Tu veux un contrôle programmatique total (transformations, validations client, etc.)
- Ton formulaire a une logique métier complexe

Pour les cas simples, préfère le composant `<Form>` ! 🎯

Wayfinder : Le Type-Safety pour tes Routes (Bonus)

Wayfinder est une fonctionnalité (relativement récente) qui génère des fonctions TypeScript type-safe pour tes routes Laravel. Au lieu d'écrire des URLs en dur, tu peux faire :

```vue
<script setup>
import { Form } from '@inertiajs/vue3';
import { store } from 'App/Http/Controllers/ArticleController'; // Import auto-généré
</script>

<template>
  <Form :action="store()">
    <input type="text" name="title" />
    <button type="submit">Create</button>
  </Form>
</template>
```

Avantages :

- Type-safety : ton IDE t'avertit si une route n'existe pas
- Refactoring sécurisé : renommer une route met à jour tous les appels
- Pas de magic strings : finies les erreurs de typo sur `/artcles` au lieu de `/articles`

Wayfinder fonctionne avec le composant `<Form>`, le helper `useForm`, et les appels `router` manuels. On lui consacrera un chapitre dédié plus tard. ☺

Avantages d'Inertia.js

- **Transitions fluides** : Les changements de page ne nécessitent pas de rechargement complet
- **Gestion simplifiée des formulaires** : Le composant `<Form>` réduit drastiquement le code nécessaire
- **Moins de duplication** : La logique de validation reste côté backend, sans réplication frontend
- **Architecture monolithique** : Pas besoin d'API REST séparée, tout reste dans Laravel
- **Developer Experience** : Avec Wayfinder, tu as même du type-safety sur tes routes

Avec ces fonctionnalités, Inertia facilite la création d'applications modernes tout en conservant la simplicité d'une architecture monolithique. En 2025, c'est **LE** standard pour du Laravel + Vue.js. 🚀

2.2 Installation de Laravel avec un Starter Kit (Vue + Inertia.js)

Pour intégrer Inertia.js dans un nouveau projet Laravel, Laravel propose désormais des **Starter Kits** clés en main qui simplifient drastiquement l'installation. Fini les étapes interminables, tout se fait en quelques commandes ! 🚀

Prérequis

Avant de commencer, assurez-vous d'avoir installé :

- **PHP** (version 8.2 ou supérieure)
- **Composer**
- **Node.js** et **npm**

Étape 1 : Installation du Laravel Installer

Le Laravel Installer est un outil CLI qui facilite la création de projets Laravel. Si vous ne l'avez pas encore, installez-le globalement via Composer :

```bash
composer global require laravel/installer
```

> ***Note** : Assurez-vous que le répertoire des binaires globaux de Composer est dans votre PATH système pour pouvoir utiliser la commande `laravel`.*

Étape 2 : Création du Projet

Créez un nouveau projet Laravel avec le Starter Kit interactif. Dans votre terminal, exécutez :

```bash
laravel new mon-projet
```

Remplacez `mon-projet` par le nom de votre application.

Étape 3 : Sélection Interactive des Options

Le Laravel Installer va vous poser plusieurs questions pour configurer votre projet. Voici ce que vous allez rencontrer :

1. Choix du Starter Kit

```
Which starter kit would you like to install? [None]:
  [none     ] None
  [react    ] React
  [vue      ] Vue
  [livewire ] Livewire
```

Sélectionnez `vue` pour utiliser Vue.js avec Inertia.js. ✅

2. Choix du Provider d'Authentification

```
Which authentication provider do you prefer? [Laravel's built-in authentication]:
  [laravel] Laravel's built-in authentication
  [workos ] WorkOS (Requires WorkOS account)
  [none   ] No authentication scaffolding
```

Sélectionnez `laravel` pour utiliser le système d'authentification natif de Laravel (recommandé pour la plupart des projets).

3. Choix du Framework de Tests

```
Which testing framework do you prefer? [Pest]:
  [0] Pest
  [1] PHPUnit
```

Sélectionnez `Pest` (c'est la recommandation moderne de Laravel). Si tu préfères PHPUnit, choisis-le, mais bon... Pest c'est mieux 😏

4. Laravel Boost (Optionnel)

BASH

```
Do you want to install Laravel Boost to improve AI assisted coding? (yes/no) [yes]:
```

Sélectionnez `no` pour un projet standard (sauf si tu veux expérimenter avec les fonctionnalités d'assistance IA).


C:\laragon\www
λ laravel new immo
Which starter kit would you like to install? [None]:
[none] None
[react] React
[vue] Vue
[livewire] Livewire
> vue
vue

```
 Which authentication provider do you prefer? [Laravel's built-in authentication]:
  [laravel] Laravel's built-in authentication
  [workos ] WorkOS (Requires WorkOS account)
  [none   ] No authentication scaffolding
 > laravel
laravel



 Which testing framework do you prefer? [Pest]:
  [0] Pest
  [1] PHPUnit
 >




 Do you want to install Laravel Boost to improve AI assisted coding? (yes/no) [yes]:
 > no

   Creating a "laravel/vue-starter-kit" project at "./side-projects-graveyard"
   Installing laravel/vue-starter-kit (dev-main 6510b268c8b65a9b9ffa79dbd6eb78d00a1ffd76)
  - Downloading laravel/vue-starter-kit (dev-main 6510b26)
     - Installing laravel/vue-starter-kit (dev-main 6510b26): Extracting archive
   Created project in C:\laragon\www/side-projects-graveyard
```

Étape 4 : Installation et Compilation

Une fois les options sélectionnées, le Laravel Installer va :

1. Télécharger et installer le starter kit Vue
2. Configurer Inertia.js
3. Installer toutes les dépendances nécessaires

Étape 5 : Finaliser l'Installation

Une fois le projet créé, naviguez dans le dossier :

```bash
cd mon-projet
```

Installez les dépendances frontend et lancez le build :

```bash
npm install && npm run build
```

Enfin, démarrez le serveur de développement Laravel :

```bash
composer run dev
```

Configuration Optionnelle

Désactiver la Double Authentification pour l'Utilisateur de Test

Les nouveaux starter kits activent la double authentification (2FA) **par défaut** pour les utilisateurs créés via les seeders. Si vous voulez désactiver cette fonctionnalité pour votre utilisateur de test, modifiez le fichier `database/seeders/DatabaseSeeder.php` :

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        User::factory()->withoutTwoFactor()->create([
            'name' => 'Test User',
            'email' => 'test@example.com',
        ]);
    }
}
```

Note : La méthode `withoutTwoFactor()` désactive explicitement la 2FA pour cet utilisateur. 🔓

Ensuite, exécutez le seeder :

```bash
php artisan db:seed
```

Structure du Projet

Après l'installation, votre projet contiendra :

- **Backend Laravel** : Controllers, routes, modèles dans `app/`, `routes/`, etc.
- **Frontend Vue.js** : Composants, pages et layouts dans `resources/js/`
- **Inertia.js** : Configuré et prêt à l'emploi pour lier Laravel et Vue
- **Tailwind CSS v4** : Préconfiguré avec shadcn-vue pour des composants UI modernes
- **Authentification complète** : Login, register, password reset, profil utilisateur, 2FA

Arborescence Frontend (Vue)

```bash
resources/js/
├── components/    # Composants Vue réutilisables
├── composables/   # Composables / hooks Vue
├── layouts/       # Layouts d'application (sidebar, header, auth)
├── lib/           # Fonctions utilitaires et configuration
├── pages/         # Pages de l'application (composants Inertia)
└── types/         # Définitions TypeScript
```

Layouts Disponibles

Le Vue Starter Kit propose **plusieurs layouts prédéfinis** que tu peux personnaliser :

Layouts Principaux

- **Sidebar Layout** (par défaut) : Barre latérale pour la navigation
- **Header Layout** : Navigation en en-tête horizontal

Pour changer de layout, modifie `resources/js/layouts/AppLayout.vue` :

```javascript
// Par défaut (sidebar)
import AppLayout from '@/layouts/app/AppSidebarLayout.vue';

// Alternative (header)
import AppLayout from '@/layouts/app/AppHeaderLayout.vue';
```

Variantes de Sidebar

- `variant="sidebar"` (par défaut)
- `variant="inset"`
- `variant="floating"`

Modifie `resources/js/components/AppSidebar.vue` pour changer la variante :

```vue
<Sidebar collapsible="icon" variant="sidebar">
```

Layouts d'Authentification

- **Simple** (par défaut)
- **Card**
- **Split**

Pour changer le layout d'authentification, modifie `resources/js/layouts/AuthLayout.vue` :

```javascript
// Par défaut (simple)
import AuthLayout from '@/layouts/auth/AuthSimpleLayout.vue';

// Alternatives
import AuthLayout from '@/layouts/auth/AuthSplitLayout.vue';
```

Conclusion

Avec cette nouvelle approche, installer Laravel + Vue.js + Inertia.js est devenu **ridiculement simple** 🎯 :

1. `composer global require laravel/installer`
2. `laravel new mon-projet`
3. Sélectionner les options interactivement
4. `npm install && npm run build`
5. `composer run dev`

C'est tout ! Plus besoin de jongler avec Jetstream, Breeze, ou des commandes Artisan obscures. L'installation est interactive, moderne, et te donne un projet **production-ready** dès le départ.

Tu peux maintenant te concentrer sur **construire ton application** plutôt que de configurer ton environnement pendant des heures. Bienvenue en 2025 🚀

Ressources

- Documentation Laravel Starter Kits
- Documentation Inertia.js
- Article de lancement des Starter Kits
- Repo GitHub Vue Starter Kit

2.3 shadcn/ui et shadcn-vue : Une Nouvelle Approche des Composants UI

Qu'est-ce que shadcn-vue ?

shadcn-vue est une collection de composants UI réutilisables, conçus avec Vue.js et Tailwind CSS. Contrairement aux bibliothèques de composants traditionnelles, shadcn-vue adopte une approche radicalement différente : les composants ne sont **pas installés comme une dépendance npm**, mais **copiés directement dans votre projet**.

Cette approche vous donne un contrôle total sur vos composants, car ils deviennent une partie intégrante de votre code source.

La Philosophie "Copy, Paste, Own"

Différence avec les Bibliothèques Classiques

Avec des bibliothèques comme Bootstrap, Vuetify ou Element Plus, vous installez un package npm qui contient tous les composants. Ces composants restent dans `node_modules` et vous les importez comme des dépendances externes.

Avec shadcn-vue, le processus est différent :

1. Vous sélectionnez les composants dont vous avez besoin
2. Vous les installez via CLI, qui les copie dans votre projet
3. Les composants sont placés dans `resources/js/components/ui/`
4. Vous pouvez les modifier librement, car ils font partie de votre code

Avantages de Cette Approche

Aspect	Bibliothèques Classiques	shadcn-vue
Customisation	Limitée (via props ou CSS override)	Totale (modification directe du code)
Taille du bundle	Toute la bibliothèque	Uniquement les composants utilisés
Dépendances	Package externe à maintenir	Code source sous votre contrôle
Apprentissage	API spécifique à apprendre	Code Vue.js standard à lire et comprendre

Pourquoi shadcn-vue dans les Starter Kits Laravel ?

Les nouveaux Laravel Starter Kits (React et Vue) incluent shadcn/shadcn-vue pour plusieurs raisons :

✅ **Design moderne et professionnel** : Les composants suivent les standards d'accessibilité (ARIA) et les bonnes pratiques de design
✅ **Intégration Tailwind CSS** : Tous les composants utilisent Tailwind, assurant une cohérence visuelle avec votre projet
✅ **Facilité de customisation** : Modifier l'apparence d'un composant se fait en changeant les classes Tailwind directement dans le fichier
✅ **Pas de surcharge** : Vous n'installez que les composants dont vous avez besoin
✅ **TypeScript natif** : Types inclus pour une meilleure autocomplétion et détection d'erreurs

Installation et Utilisation

Ajouter un Nouveau Composant

Le Vue Starter Kit est préconfigurée avec shadcn-vue. Pour ajouter un composant, utilisez la CLI :

```bash
npx shadcn-vue@latest add button
```

Cette commande va :

- Télécharger le composant `Button.vue`
- Le placer dans `resources/js/components/ui/button/`
- Installer les dépendances nécessaires si besoin

Utiliser le Composant

Une fois installé, importez et utilisez le composant comme n'importe quel composant Vue :

```vue
<script setup lang="ts">
import { Button } from '@/components/ui/button';
</script>

<template>
  <div>
    <Button>Cliquez ici</Button>
    <Button variant="destructive">Supprimer</Button>
    <Button variant="outline" size="sm">Petit bouton</Button>
  </div>
</template>
```

Personnaliser un Composant

Le fichier du composant se trouve maintenant dans votre projet. Pour le personnaliser, ouvrez-le et modifiez-le :

Chemin du fichier : `resources/js/components/ui/button/Button.vue`

```vue
<template>
  <button
    :class="cn(
      'inline-flex items-center justify-center rounded-md',
      'bg-blue-600 hover:bg-blue-700', // Modifiez ces classes
      // ... autres classes
    )"
  >
    <slot />
  </button>
</template>
```

Toutes les modifications s'appliqueront immédiatement à tous les usages du composant dans votre application.

Composants Disponibles

shadcn-vue propose une large gamme de composants :

Composants de Base :

- Button, Input, Textarea, Checkbox, Radio, Select
- Label, Badge, Avatar, Separator

Navigation :

- Tabs, Breadcrumb, Pagination, Command, NavigationMenu

Overlays & Modales :

- Dialog (Modal), Sheet (Drawer), Popover, Tooltip, AlertDialog

Affichage de Données :

- Table, Card, Alert, Toast, Progress

Formulaires :

- Form, Calendar, Date Picker, Combobox

Layout :

- Accordion, Collapsible, Sidebar

La liste complète est disponible sur shadcn-vue.com.

Exemple Pratique : Ajouter un Switch

Imaginons que vous souhaitiez ajouter un composant Switch (interrupteur) pour activer/désactiver le mode sombre.

Étape 1 : Installer le composant

```bash
npx shadcn-vue@latest add switch
```

Étape 2 : Utiliser le composant

```vue
<script setup lang="ts">
import { Switch } from '@/components/ui/switch';
import { ref } from 'vue';

const darkMode = ref(false);
</script>

<template>
  <div class="flex items-center space-x-2">
    <Switch v-model:checked="darkMode" id="dark-mode" />
    <label for="dark-mode" class="cursor-pointer">Mode sombre</label>
  </div>
</template>
```

Le composant est maintenant intégré à votre projet et fonctionne immédiatement.

Les Layouts du Starter Kit

Le Vue Starter Kit utilise shadcn-vue de manière intensive. Les différents layouts (sidebar, header, auth) sont construits avec des composants shadcn-vue :

- **AppSidebar.vue** → Utilise le composant `Sidebar` de shadcn-vue
- **Formulaires d'authentification** → Utilisent `Form`, `Input`, `Button`
- **Navigation** → Utilise `NavigationMenu`, `DropdownMenu`

Ces composants sont déjà installés et configurés. Vous pouvez les explorer dans le dossier `resources/js/components/` pour comprendre leur fonctionnement.

Bonnes Pratiques

Explorer les Composants Installés

Prenez le temps d'ouvrir les fichiers des composants dans `resources/js/components/ui/`. C'est une excellente opportunité d'apprendre :

- Comment structurer des composants Vue.js réutilisables
- Comment utiliser Tailwind CSS efficacement
- Comment implémenter l'accessibilité (attributs ARIA)

Modifier avec Prudence

Bien que vous puissiez modifier les composants librement, réfléchissez aux conséquences :

- Une modification affecte **tous** les usages du composant dans votre projet
- Documentez vos changements si vous travaillez en équipe
- Considérez créer une variante plutôt que de modifier le composant de base si vous avez besoin d'un comportement spécifique

Installer uniquement ce dont vous avez besoin

N'installez pas tous les composants "au cas où". Installez-les au fur et à mesure de vos besoins pour garder votre projet léger.

Comparaison : Avant et Après

Avant shadcn-vue

Vous aviez deux options pour créer des interfaces :

1. **Utiliser une bibliothèque complète** (Vuetify, Element Plus)
   - ❌ Personnalisation difficile
   - ❌ Surcharge du bundle
   - ❌ Dépendance à une API externe
2. **Tout coder vous-même**
   - ❌ Long et répétitif
   - ❌ Risque de bugs
   - ❌ Accessibilité souvent négligée

Avec shadcn-vue

Vous bénéficiez des deux approches :

- ✅ Composants de qualité professionnelle, immédiatement disponibles
- ✅ Contrôle total pour les adapter à vos besoins
- ✅ Code source lisible et modifiable
- ✅ Pas de "boîte noire" externe

Conclusion

shadcn-vue représente une évolution dans la manière de concevoir les bibliothèques de composants. En plaçant le code directement dans votre projet, cette approche vous donne :

✅ Un contrôle total sur vos composants
✅ Une meilleure compréhension du code que vous utilisez
✅ Une personnalisation sans friction
✅ Une maintenance simplifiée (pas de version externe à suivre)

Avec les Laravel Starter Kits, shadcn-vue est déjà configuré et prêt à l'emploi. Vous pouvez commencer à construire votre interface immédiatement en installant les composants dont vous avez besoin.

Ressources

- Documentation shadcn-vue
- Documentation shadcn/ui (version React)
- GitHub shadcn-vue
- Radix Vue (primitives sous-jacents)
- Tailwind CSS

2.4 Pages avec Inertia

Inertia.js est un outil qui facilite la création d'applications mono-page (SPA) avec des méthodologies de serveur traditionnelles comme Laravel, sans nécessiter d'API séparée. Il sert de pont entre le backend et le frontend, permettant une expérience utilisateur fluide et dynamique.

Concept de base

Fonctionnement général

Dans une application utilisant Inertia, chaque page correspond à une route et un contrôleur dans le backend, ainsi qu'à un composant JavaScript dans le frontend, spécifiquement Vue.js pour ce cours. Ce modèle permet de gérer le routage et la logique côté serveur tout en offrant une expérience utilisateur comparable à une SPA.

Avantages clés

- **Pas d'API nécessaire** : Inertia transmet directement les données aux composants Vue.js, éliminant le besoin d'une API dédiée.
- **Chargement préalable des données** : Les données nécessaires sont récupérées avant le rendu par le navigateur, supprimant les états de chargement et améliorant l'expérience utilisateur.

Création de pages

Les pages Inertia sont des composants JavaScript, ici spécifiquement des composants Vue.js.

Exemple pratique

Prenons le fichier `User/Show.vue` dans une application Laravel, qui est un composant Vue.js utilisant Inertia.js.

Composant Vue.js

```vue
<script setup>
import Layout from './Layout'
import { Head } from '@inertiajs/vue3'

defineProps({ user: Object })
</script>

<template>
  <Layout>
    <Head title="Welcome" />
    <h1>Welcome</h1>
    <p>Hello {{ user.name }}, welcome to your first Inertia app!</p>
  </Layout>
</template>
```

Ce composant affiche un message de bienvenue à l'utilisateur, avec les données passées comme props.

Contrôleur Laravel

```php
use Inertia\Inertia;

class UserController extends Controller
{
    public function show(User $user)
    {
        return Inertia::render('User/Show', [
          'user' => $user
        ]);
    }
}
```

La méthode `show` renvoie une réponse Inertia qui rend le composant `User/Show.vue`, en passant l'objet `user`.

But de cette approche

Inertia simplifie le développement des SPAs en utilisant des pratiques de développement traditionnelles, réduisant la complexité et augmentant l'efficacité. Elle permet de se concentrer sur la logique métier tout en offrant une interface utilisateur réactive et moderne.

Optimisation des réponses Inertia

Pourquoi optimiser les réponses?

Optimiser les réponses permet de minimiser la latence et d'améliorer la réactivité de l'application. En ne transmettant que les données nécessaires, vous évitez les surcharges inutiles et assurez une expérience utilisateur plus fluide, surtout sur des réseaux lents ou des appareils à faible performance.

Comment optimiser les réponses

Pour illustrer l'optimisation des réponses, utilisons l'exemple d'un contrôleur qui gère l'affichage d'un utilisateur (`User`) dans une application Laravel utilisant Inertia.js.

Exemple de contrôleur optimisé

```php
use Inertia\Inertia;

class UserController extends Controller
{
    public function show(User $user)
    {
        // Création d'une réponse Inertia en sélectionnant uniquement les attributs nécessaires
        return Inertia::render('User/Show', [
            'user' => $user->only('id', 'name', 'email', 'created_at')
        ]);
    }
}
```

Dans cet exemple, la méthode `only` est utilisée pour sélectionner uniquement les attributs spécifiques de l'objet `User` qui sont nécessaires pour la page. Cela évite de transmettre des données superflues comme les mots de passe ou les informations sensibles qui ne sont pas utilisées par la vue.

Avantages de cette approche

- **Réduction de la bande passante** : Moins de données sont transférées entre le serveur et le client.
- **Amélioration des performances** : Le serveur traite moins de données, et le client traite moins de JSON, ce qui accélère le rendu.
- **Sécurité accrue** : En limitant les données exposées aux clients, vous réduisez le risque de fuite d'informations sensibles.

2.5 Laravel Wayfinder : Le Type-Safety pour Vos Routes

Qu'est-ce que Wayfinder ?

Laravel Wayfinder est un outil qui génère automatiquement des **fonctions TypeScript typées** pour vos routes et contrôleurs Laravel. Au lieu d'écrire des URLs en dur dans votre code frontend, vous importez des fonctions qui représentent vos routes backend.

Le Problème

Traditionnellement, pour appeler une route Laravel depuis Vue.js, vous écrivez :

```typescript
// ❌ Approche traditionnelle : URLs en dur
router.post('/posts/1', formData);
```

Problèmes avec cette approche :

- Risque de **typos** : `/psots/1` au lieu de `/posts/1`
- Aucune **autocomplétion** : votre IDE ne vous aide pas
- Pas de **vérification des paramètres** : oublier un paramètre obligatoire passe inaperçu
- **Refactoring impossible** : renommer une route côté backend casse le frontend silencieusement

La Solution : Wayfinder

Avec Wayfinder, vous importez des fonctions générées automatiquement :

```typescript
// ✅ Approche Wayfinder : fonctions typées
import { update } from '@/actions/App/Http/Controllers/PostController';

form.submit(update(1)); // Type-safe, autocomplétion, erreurs détectées
```

Avantages :

- ✅ **Type-safety** : TypeScript vérifie les paramètres à la compilation
- ✅ **Autocomplétion** : votre IDE suggère les routes disponibles
- ✅ **Refactoring sécurisé** : renommer une méthode met à jour tous les appels
- ✅ **Zéro duplication** : une seule source de vérité (vos contrôleurs Laravel)

Génération Automatique

Important : Avec le Vue Starter Kit, la génération des définitions TypeScript se fait **automatiquement** lorsque vous lancez :

```bash
composer run dev
```

Cette commande démarre à la fois le serveur Laravel et Vite, qui surveille vos fichiers et régénère les définitions Wayfinder à chaque modification de routes ou contrôleurs.

Fichiers générés :

Wayfinder crée trois dossiers dans `resources/js/` :

- `wayfinder/` : Fichiers de configuration internes
- `actions/` : Fonctions correspondant à vos **méthodes de contrôleurs**

- `routes/` : Fonctions correspondant à vos **routes nommées**

> ***Note** : Ces dossiers sont régénérés automatiquement. Vous pouvez les ajouter à `.gitignore` car ils ne doivent pas être versionnés.*

Utilisation de Base

Importer une Action de Contrôleur

Imaginons que vous avez un contrôleur `PostController` avec une méthode `show` :

```php
// app/Http/Controllers/PostController.php
class PostController extends Controller
{
    public function show(Post $post)
    {
        return Inertia::render('Posts/Show', [
            'post' => $post,
        ]);
    }
}
```

Avec Wayfinder, vous pouvez importer cette action :

```typescript
import { show } from '@/actions/App/Http/Controllers/PostController';

show(1); // Retourne : { url: "/posts/1", method: "get" }
```

Utilisation avec Inertia

Wayfinder s'intègre parfaitement avec Inertia.js :

Avec le composant `<Link>` :

```vue
<script setup lang="ts">
import { Link } from '@inertiajs/vue3';
import { show } from '@/actions/App/Http/Controllers/PostController';
</script>

<template>
  <Link :href="show(1)">Voir l'article #1</Link>
</template>
```

Avec le Form Helper :

```vue
<script setup lang="ts">
import { useForm } from '@inertiajs/vue3';
import { store } from '@/actions/App/Http/Controllers/PostController';

const form = useForm({
  title: 'Mon article',
  content: 'Contenu de l\'article',
});

const submitForm = () => {
  form.submit(store()); // Wayfinder résout automatiquement l'URL et la méthode
};
</script>

<template>
  <form @submit.prevent="submitForm">
    <!-- Vos champs de formulaire -->
  </form>
</template>
```

Avec le composant `<Form>` :

```vue
<script setup lang="ts">
import { Form } from '@inertiajs/vue3';
import { store } from '@/actions/App/Http/Controllers/PostController';
</script>

<template>
  <Form :action="store()">
    <input type="text" name="title" />
    <button type="submit">Créer</button>
  </Form>
</template>
```

Cas d'Usage Avancés

Méthodes Multiples

Si vous souhaitez spécifier une méthode HTTP particulière :

```typescript
import { update } from '@/actions/App/Http/Controllers/PostController';

update.put(1);   // { url: "/posts/1", method: "put" }
update.patch(1); // { url: "/posts/1", method: "patch" }
```

Obtenir Uniquement l'URL

Si vous avez seulement besoin de l'URL (sans l'objet complet) :

```typescript
import { show } from '@/actions/App/Http/Controllers/PostController';

show.url(1); // "/posts/1"
```

Paramètres Multiples

Wayfinder accepte différents formats de paramètres :

```typescript
import { update } from '@/actions/App/Http/Controllers/PostController';

// Tableau de paramètres
update([1, 2]); // Route : /posts/{post}/authors/{author}

// Objet avec noms de paramètres
update({ post: 1, author: 2 });

// Objets imbriqués
update({ post: { id: 1 }, author: { id: 2 } });
```

Route Bindings Personnalisés

Si votre route utilise un champ personnalisé (comme `slug`) :

```php
Route::get('/posts/{post:slug}', [PostController::class, 'show']);
```

Wayfinder détecte cela et accepte le slug directement :

```typescript
import { show } from '@/actions/App/Http/Controllers/PostController';

show('mon-article'); // { url: "/posts/mon-article", method: "get" }
show({ slug: 'mon-article' }); // Alternative avec objet
```

Query Parameters

Vous pouvez ajouter des paramètres de requête :

```typescript
import { show } from '@/actions/App/Http/Controllers/PostController';

const options = {
  query: {
    page: 1,
    sort_by: 'name',
  },
};

show.url(1, options); // "/posts/1?page=1&sort_by=name"
```

Routes Nommées

Wayfinder génère également des fonctions pour vos routes nommées :

```php
Route::get('/posts/{post}', [PostController::class, 'show'])
    ->name('posts.show');
```

Vous pouvez les importer depuis le dossier `routes` :

```typescript
import { show } from '@/routes/posts';

show(1); // { url: "/posts/1", method: "get" }
```

Contrôleurs Invocables

Si vous utilisez des contrôleurs invocables (avec méthode `__invoke`) :

```php
class StorePostController extends Controller
{
    public function __invoke(Request $request)
    {
        // ...
    }
}
```

Importez directement le contrôleur et invoquez-le :

```typescript
import StorePostController from '@/actions/App/Http/Controllers/StorePostController';

StorePostController(); // { url: "/posts", method: "post" }
```

Mots Réservés JavaScript

Si une méthode de votre contrôleur porte un nom qui est un mot réservé JavaScript (comme `delete`, `import`), Wayfinder la renomme automatiquement en ajoutant `Method` :

```php
public function delete(Post $post) { ... }
```

Devient :

```typescript
import { deleteMethod } from '@/actions/App/Http/Controllers/PostController';

deleteMethod(1); // { url: "/posts/1", method: "delete" }
```

Avantages de Wayfinder

Pour le Développement

- **Productivité accrue** : autocomplétion et suggestions de l'IDE
- **Moins d'erreurs** : détection des problèmes à la compilation, pas à l'exécution
- **Refactoring sûr** : renommer une route met à jour automatiquement tous les appels

Pour la Maintenance

- **Source unique de vérité** : les routes sont définies côté Laravel uniquement
- **Synchronisation automatique** : toute modification backend se propage au frontend
- **Documentation vivante** : les types TypeScript documentent les routes disponibles

Pour l'Équipe

- **Moins de communication nécessaire** : les nouveaux développeurs découvrent les routes via l'autocomplétion
- **Standards cohérents** : tout le monde utilise la même approche
- **Onboarding simplifié** : pas besoin de mémoriser les URLs

Bonnes Pratiques

Nommez Vos Routes

Donnez des noms explicites à vos routes pour faciliter leur utilisation :

```php
Route::post('/posts', [PostController::class, 'store'])
    ->name('posts.store');
```

Utilisez le Route Model Binding

Laissez Laravel résoudre automatiquement les modèles :

```php
public function show(Post $post) // Wayfinder génère : show(1)
```

Au lieu de :

```php
public function show($id) // Moins clair
```

Explorez les Fichiers Générés

Prenez le temps d'ouvrir les fichiers dans `resources/js/actions/` pour comprendre ce qui est généré. C'est une excellente façon d'apprendre.

Conclusion

Laravel Wayfinder transforme la manière dont vous appelez vos routes backend depuis votre frontend. En générant automatiquement des fonctions TypeScript typées, il élimine les erreurs de typos, facilite le refactoring, et améliore significativement l'expérience de développement.

Avec les Laravel Starter Kits, Wayfinder est déjà configuré et s'exécute automatiquement. Vous n'avez qu'à utiliser les imports pour profiter d'une intégration backend/frontend sans friction.

Note importante : Wayfinder est actuellement en **Beta**. L'API peut encore évoluer avant la version 1.0.0. Consultez régulièrement le changelog pour les changements.

Ressources

- Documentation officielle Wayfinder
- Wayfinder et Inertia.js

- TypeScript Documentation

2.6 Routage avec Laravel et Inertia

Le routage dans une application utilisant Laravel et Inertia présente quelques spécificités importantes, notamment en ce qui concerne la gestion des redirections et l'utilisation de helpers pour la génération des URLs. Nous allons explorer ces aspects à travers des explications et des exemples concrets.

Définition des routes

Les routes dans une application Laravel avec Inertia sont définies dans le fichier `web.php`, de manière similaire à une application Laravel traditionnelle. Cependant, l'utilisation d'Inertia implique souvent des interactions spécifiques, surtout pour les requêtes autres que GET.

Exemple de définition de route

Voici un exemple de fichier `routes/web.php` dans une application Laravel utilisant Inertia :

```php
use Inertia\Inertia;
use Illuminate\Support\Facades\Route;

Route::get('/users', function () {
    return Inertia::render('Users/Index');
});

Route::post('/users', 'UserController@store');
```

Gestion des redirections

Lors de la création ou de la modification de ressources via des requêtes non-GET, il est crucial de répondre par une redirection appropriée. Inertia s'attend à ce que ces actions soient suivies d'une réponse de redirection pour pouvoir mettre à jour la page de manière appropriée.

Exemple de redirection après une action

Supposons que vous ayez un contrôleur `UserController` avec une méthode `store` pour créer un nouvel utilisateur. Voici comment vous pourriez gérer la redirection :

```php
public function store(Request $request)
{
    // Logique pour créer un utilisateur
    User::create($request->all());

    // Redirection vers la page d'index des utilisateurs
    return redirect()->route('users.index');
}
```

Dans cet exemple, après la création de l'utilisateur, le contrôleur redirige l'utilisateur vers la page d'index des utilisateurs. Inertia interceptera cette redirection et chargera la page d'index des utilisateurs sans recharger complètement le navigateur.

Utilisation du helper `route()` dans VueJS

Inertia permet également d'utiliser un helper `route()` dans les fichiers VueJS, similaire à celui utilisé dans les vues Blade de Laravel. Ce helper facilite la génération d'URLs pour les liens ou les actions dans les composants Vue.

Exemple d'utilisation du helper `route()`

Dans un composant Vue, vous pourriez utiliser le helper `route()` pour créer un lien vers la page d'index des utilisateurs :

```vue
<script setup>
import { Link } from '@inertiajs/vue3';
</script>

<template>
  <Link :href="route('users.index')">Voir tous les utilisateurs</Link>
</template>
```

Ce code crée un lien qui, lorsqu'il est cliqué, dirigera Inertia pour charger la page d'index des utilisateurs en utilisant la définition de route côté serveur, sans rechargement complet de la page.

2.7 Formulaires avec Inertia.js

Dans ce chapitre, nous allons explorer les deux approches pour gérer les formulaires avec Inertia.js : le **composant** `<Form>` (recommandé pour la majorité des cas) et le **helper** `useForm` (pour un contrôle programmatique avancé). Nous verrons également comment gérer l'upload de fichiers et l'affichage de la progression.

Le Composant `<Form>` : L'Approche Moderne

Le composant `<Form>` est l'approche **recommandée** pour créer des formulaires avec Inertia.js. Il se comporte comme un formulaire HTML classique, sans nécessiter de `v-model` ni de gestion manuelle de l'état.

Exemple : Créer une Maison

Voici un formulaire complet pour créer un bien immobilier dans une application d'agence :

VUE

```
<script setup>
import { Form } from '@inertiajs/vue3';
import { store, index } from '@/actions/App/Http/Controllers/HouseController';
import AppLayout from '@/layouts/AppLayout.vue';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Textarea } from '@/components/ui/textarea';
import { Label } from '@/components/ui/label';
import {
    Card,
    CardContent,
    CardDescription,
    CardHeader,
    CardTitle,
} from '@/components/ui/card';

const props = defineProps({
    houseTypes: Array,
});
</script>

<template>
    <AppLayout>
        <div class="container mx-auto max-w-3xl px-3 py-8">
            <Card>
                <CardHeader>
                    <CardTitle>Ajouter une maison</CardTitle>
                    <CardDescription>
                        Remplissez les informations ci-dessous pour ajouter un
                        nouveau bien immobilier
                    </CardDescription>
                </CardHeader>
                <CardContent>
                    <Form
                        :action="store()"
                        #default="{ errors, processing }"
                        class="space-y-6"
                    >
                        <!-- Titre -->
                        <div class="space-y-2">
                            <Label for="title">
                                Titre
                                <span class="text-destructive">*</span>
                            </Label>
                            <Input
                                id="title"
                                type="text"
                                name="title"
                                placeholder="Ex: Belle maison à Paris"
                            />
                            <p
                                v-if="errors.title"
```

```
                                class="text-sm text-destructive"
                            >
                                {{ errors.title }}
                            </p>
                        </div>

                        <!-- Type de bien -->
                        <div class="space-y-2">
                            <Label for="house_type_id">
                                Type de bien
                                <span class="text-destructive">*</span>
                            </Label>
                            <select
                                id="house_type_id"
                                name="house_type_id"
                                class="flex h-9 w-full rounded-md border border-input
bg-transparent px-3 py-1 text-sm shadow-xs transition-colors placeholder:text-muted-
foreground focus-visible:ring-1 focus-visible:ring-ring focus-visible:outline-none"
                            >
                                <option value="">Sélectionnez un type</option>
                                <option
                                    v-for="type in houseTypes"
                                    :key="type.id"
                                    :value="type.id"
                                >
                                    {{ type.name }}
                                </option>
                            </select>
                            <p
                                v-if="errors.house_type_id"
                                class="text-sm text-destructive"
                            >
                                {{ errors.house_type_id }}
                            </p>
                        </div>

                        <!-- Prix -->
                        <div class="space-y-2">
                            <Label for="price">
                                Prix (€)
                                <span class="text-destructive">*</span>
                            </Label>
                            <Input
                                id="price"
                                type="number"
                                name="price"
                                step="0.01"
                                placeholder="250000"
                            />
                            <p
                                v-if="errors.price"
                                class="text-sm text-destructive"
                            >
```

```
                        {{ errors.price }}
                    </p>
                </div>

                <!-- Adresse -->
                <div class="space-y-2">
                    <Label for="address">
                        Adresse
                        <span class="text-destructive">*</span>
                    </Label>
                    <Input
                        id="address"
                        type="text"
                        name="address"
                        placeholder="123 rue de la Paix, 75001 Paris"
                    />
                    <p
                        v-if="errors.address"
                        class="text-sm text-destructive"
                    >
                        {{ errors.address }}
                    </p>
                </div>

                <!-- Chambres et Taille -->
                <div class="grid gap-4 md:grid-cols-2">
                    <div class="space-y-2">
                        <Label for="bedrooms">
                            Nombre de chambres
                            <span class="text-destructive">*</span>
                        </Label>
                        <Input
                            id="bedrooms"
                            type="number"
                            name="bedrooms"
                            placeholder="3"
                            min="0"
                        />
                        <p
                            v-if="errors.bedrooms"
                            class="text-sm text-destructive"
                        >
                            {{ errors.bedrooms }}
                        </p>
                    </div>

                    <div class="space-y-2">
                        <Label for="size">
                            Taille (m²)
                            <span class="text-destructive">*</span>
                        </Label>
                        <Input
                            id="size"
```

```
                                type="number"
                                name="size"
                                step="0.01"
                                placeholder="120.50"
                                min="0"
                            />
                            <p
                                v-if="errors.size"
                                class="text-sm text-destructive"
                            >
                                {{ errors.size }}
                            </p>
                        </div>
                    </div>

                    <!-- Description -->
                    <div class="space-y-2">
                        <Label for="description">Description</Label>
                        <Textarea
                            id="description"
                            name="description"
                            rows="5"
                            placeholder="Décrivez le bien..."
                        />
                        <p
                            v-if="errors.description"
                            class="text-sm text-destructive"
                        >
                            {{ errors.description }}
                        </p>
                    </div>

                    <!-- Boutons -->
                    <div class="flex gap-4">
                        <Button type="submit" :disabled="processing">
                            {{
                                processing
                                    ? 'Enregistrement...'
                                    : 'Créer la maison'
                            }}
                        </Button>

                        <Button
                            type="button"
                            variant="outline"
                            @click="$inertia.visit(index())"
                        >
                            Annuler
                        </Button>
                    </div>
                </Form>
            </CardContent>
        </Card>
```

```
        </div>
    </AppLayout>
</template>
```

Points Clés du Composant `<Form>`

1. **Pas de** `v-model` : Chaque input a simplement un attribut `name`. Le composant gère automatiquement la collecte des données.
2. **Intégration Wayfinder** : L'action `:action="store()"` utilise Wayfinder pour générer automatiquement l'URL et la méthode HTTP.
3. **Slot Props** : `#default="{ errors, processing }"` expose les erreurs de validation et l'état de traitement du formulaire.
4. **Affichage des Erreurs** : Les erreurs sont accessibles via `errors.nomDuChamp` et s'affichent automatiquement après validation côté serveur.
5. **État de Chargement** : La propriété `processing` permet de désactiver le bouton pendant la soumission et d'afficher un spinner.

Modification avec Valeurs Par Défaut

Pour modifier une ressource existante, utilisez l'attribut `defaultValue` (ou `selected` pour les `<select>`) :

VUE

```
<script setup>
import { Form } from '@inertiajs/vue3';
import { update, index } from '@/actions/App/Http/Controllers/HouseController';

const props = defineProps({
    house: Object,
    houseTypes: Array,
});
</script>

<template>
    <Form
        :action="update(house.id)"
        #default="{ errors, processing }"
        class="space-y-6"
    >
        <!-- Titre avec valeur par défaut -->
        <div class="space-y-2">
            <Label for="title">Titre</Label>
            <Input
                id="title"
                type="text"
                name="title"
                :defaultValue="house.title"
                placeholder="Ex: Belle maison à Paris"
            />
            <p v-if="errors.title" class="text-sm text-destructive">
                {{ errors.title }}
            </p>
        </div>

        <!-- Select avec valeur par défaut -->
        <div class="space-y-2">
            <Label for="house_type_id">Type de bien</Label>
            <select id="house_type_id" name="house_type_id">
                <option
                    v-for="type in houseTypes"
                    :key="type.id"
                    :value="type.id"
                    :selected="type.id === house.house_type_id"
                >
                    {{ type.name }}
                </option>
            </select>
        </div>

        <!-- Prix avec valeur par défaut -->
        <div class="space-y-2">
            <Label for="price">Prix (€)</Label>
            <Input
                id="price"
                type="number"
```

```
                name="price"
                step="0.01"
                :defaultValue="house.price"
            />
        </div>

        <!-- Autres champs... -->

        <Button type="submit" :disabled="processing">
            Mettre à jour
        </Button>
    </Form>
</template>
```

Note importante : Utilisez `:defaultValue` (avec deux-points) pour les valeurs dynamiques, et `defaultValue` (sans deux-points) pour les valeurs statiques.

Gestion Côté Serveur avec Laravel

Création d'une Ressource

PHP

```php
use Illuminate\Http\Request;
use App\Models\House;

public function store(Request $request)
{
    $validated = $request->validate([
        'title' => 'required|string|max:255',
        'house_type_id' => 'required|exists:house_types,id',
        'price' => 'required|numeric|min:0',
        'address' => 'required|string|max:255',
        'bedrooms' => 'required|integer|min:0',
        'size' => 'required|numeric|min:0',
        'description' => 'nullable|string',
    ]);

    House::create($validated);

    return redirect()->route('houses.index')
        ->with('success', 'Maison créée avec succès.');
}
```

Mise à Jour d'une Ressource

```php
public function update(Request request,Houserequest, House request,Househouse)
{
    validated=validated = validated=request→validate([
        'title' ⇒ 'required|string|max:255',
        'house_type_id' ⇒ 'required|exists:house_types,id',
        'price' ⇒ 'required|numeric|min:0',
        'address' ⇒ 'required|string|max:255',
        'bedrooms' ⇒ 'required|integer|min:0',
        'size' ⇒ 'required|numeric|min:0',
        'description' ⇒ 'nullable|string',
    ]);

    house->update(house→update(house->update(validated);

    return redirect()→route('houses.index')
        →with('success', 'Maison mise à jour avec succès.');
}
```

Note : Inertia gère automatiquement les erreurs de validation. Si `$request→validate()` échoue, les erreurs sont renvoyées au frontend et accessibles via `errors` dans le slot props.

Le Helper `useForm` : Pour un Contrôle Avancé

Le helper `useForm` est utile lorsque vous avez besoin d'un **contrôle programmatique** sur vos formulaires. C'est particulièrement adapté pour :

- Les uploads de fichiers avec gestion manuelle
- Les transformations de données complexes avant soumission
- Les formulaires avec logique métier sophistiquée

Exemple : Upload d'Images

Dans notre application d'agence immobilière, l'upload d'images pour une maison est géré via `useForm` :

VUE

```
<script setup>
import { useForm } from '@inertiajs/vue3';

const props = defineProps({
    house: Object,
});

// Form pour l'upload d'images
const imageForm = useForm({
    images: null,
});

const handleImageChange = (event) => {
    const files = event.target.files;
    if (files && files.length > 0) {
        // Convertir FileList en Array
        imageForm.images = Array.from(files);
    }
};

const uploadImages = () => {
    imageForm.post(`/houses/{props.house.id}/images`, {
        preserveScroll: true,
        onSuccess: () => {
            imageForm.reset();
            // Reset l'input file
            const input = document.getElementById('images');
            if (input) {
                input.value = '';
            }
        },
    });
};
</script>

<template>
    <form @submit.prevent="uploadImages" class="space-y-4">
        <div class="space-y-2">
            <Label for="images">Ajouter de nouvelles images</Label>
            <input
                id="images"
                type="file"
                multiple
                accept="image/*"
                @input="handleImageChange"
                class="flex h-10 w-full rounded-md border border-input bg-background
px-3 py-2 text-sm"
            />
            <p class="text-sm text-muted-foreground">
                Vous pouvez sélectionner plusieurs images (max 2MB par image)
            </p>
            <p
```

```
                v-if="imageForm.errors['images.0']"
                class="text-sm text-destructive"
            >
                {{ imageForm.errors['images.0'] }}
            </p>
        </div>

        <!-- Progression upload -->
        <div v-if="imageForm.progress" class="space-y-2">
            <div class="flex justify-between text-sm">
                <span>Upload en cours...</span>
                <span class="font-medium">
                    {{ imageForm.progress.percentage }}%
                </span>
            </div>
            <div class="h-2 w-full rounded-full bg-secondary">
                <div
                    class="h-2 rounded-full bg-primary transition-all"
                    :style="{
                        width: `{imageForm.progress.percentage}%`,
                    }"
                />
            </div>
        </div>

        <Button
            type="submit"
            :disabled="imageForm.processing || !imageForm.images"
        >
            {{
                imageForm.processing
                    ? 'Upload en cours...'
                    : 'Ajouter les images'
            }}
        </Button>
    </form>
</template>
```

Points Clés du `useForm`

1. **Initialisation** : `const imageForm = useForm({ images: null })`
2. **Liaison Manuelle** : Avec `useForm`, vous gérez manuellement les changements via `v-model` ou des handlers d'événements.
3. **Méthodes de Soumission** :
   - `imageForm.post(url, options)`
   - `imageForm.put(url, options)`
   - `imageForm.patch(url, options)`

- `imageForm.delete(url, options)`

4. **Options de Soumission** :
   - `preserveScroll: true` : Maintient la position du scroll
   - `onSuccess: () ⇒ {}` : Callback après succès
   - `onError: () ⇒ {}` : Callback en cas d'erreur
5. **Propriétés Disponibles** :
   - `imageForm.processing` : Indique si le formulaire est en cours de soumission
   - `imageForm.progress` : Objet contenant `percentage` pour l'upload de fichiers
   - `imageForm.errors` : Objet contenant les erreurs de validation
   - `imageForm.wasSuccessful` : `true` après une soumission réussie
6. **Méthodes Utiles** :
   - `imageForm.reset()` : Réinitialise le formulaire
   - `imageForm.clearErrors()` : Efface toutes les erreurs
   - `imageForm.setError('field', 'message')` : Définit une erreur manuellement

Upload de Fichiers : Gestion Côté Serveur

```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use App\Models\House;
use App\Models\HouseImage;

public function uploadImages(Request request,Houserequest, House request,Househouse)
{
    request->validate(['images'⇒'required|array',
'images.*'⇒'image|mimes:jpeg,png,jpg,gif|max:2048',]);foreach(request→validate([
        'images' ⇒ 'required|array',
        'images.*' ⇒ 'image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);

    foreach (request->validate(['images'⇒'required|array',
'images.*'⇒'image|mimes:jpeg,png,jpg,gif|max:2048',]);foreach(request→file('images')
as $image) {
        $path = $image→store('houses', 'public');

        HouseImage::create([
            'house_id' ⇒ $house→id,
            'path' ⇒ $path,
        ]);
    }

    return back()→with('success', 'Images ajoutées avec succès.');
}
```

Note : Inertia convertit automatiquement les requêtes contenant des fichiers en `FormData`. Aucune configuration supplémentaire n'est nécessaire.

Suppression d'une Image

Pour supprimer une image, utilisez directement le `router` d'Inertia :

```vue
<script setup>
import { router } from '@inertiajs/vue3';

const props = defineProps({
    house: Object,
});

const deleteImage = (imageId) => {
    if (confirm('Êtes-vous sûr de vouloir supprimer cette image ?')) {
        router.delete(`/houses/${props.house.id}/images/{imageId}`, {
            preserveScroll: true,
        });
    }
};
</script>

<template>
    <Button
        size="sm"
        variant="destructive"
        @click="deleteImage(image.id)"
    >
        Supprimer
    </Button>
</template>
```

Côté Serveur :

```php
use App\Models\HouseImage;
use Illuminate\Support\Facades\Storage;

public function deleteImage(House house,HouseImagehouse, HouseImage
house,HouseImageimage)
{
    // Vérifier que l'image appartient bien à cette maison
    if (image->houseid≠=image→house_id ≠= image->houseid≠=house→id) {
        abort(403);
    }

    // Supprimer le fichier physique
    Storage::disk('public')
→delete(image->path);//Supprimerl'enregistrementenbaseimage→path);

    // Supprimer l'enregistrement en base
    image->path);//Supprimerl'enregistrementenbaseimage→delete();

    return back()→with('success', 'Image supprimée avec succès.');
}
```

Quand Utiliser `<Form>` vs `useForm` ?

Critère	`<Form>` Component	`useForm` Helper
Simplicité	✅ Très simple, pas de `v-model`	⚠️ Plus verbeux
Cas d'usage	CRUD standard, formulaires classiques	Logique complexe, transformations
Upload de fichiers	✅ Supporte nativement	✅ Avec gestion manuelle
Valeurs par défaut	`defaultValue` / `selected`	Pré-remplir avec `useForm(data)`
Contrôle programmatique	⚠️ Limité (via refs)	✅ Total
Recommandation	**Privilégier par défaut**	Pour les cas avancés

Règle générale : Utilisez `<Form>` sauf si vous avez une raison spécifique d'utiliser `useForm`.

Options Avancées

Avec `<Form>` Component

VUE

```
<Form
    :action="update(house.id)"
    :options="{
        preserveScroll: true,
        preserveState: true,
        only: ['house', 'flash'],
    }"
    resetOnSuccess
    #default="{ errors, processing, wasSuccessful }"
>
    <!— Vos champs —>

    <div v-if="wasSuccessful" class="text-green-600">
        Modifications enregistrées !
    </div>
</Form>
```

Avec `useForm` Helper

```javascript
form.post('/houses', {
    preserveScroll: true,
    onSuccess: () => {
        form.reset();
        alert('Maison créée avec succès !');
    },
    onError: (errors) => {
        console.error('Erreurs:', errors);
    },
});
```

Conclusion

Inertia.js propose deux approches puissantes pour gérer les formulaires :

1. **Le composant** `<Form>` : Simple, déclaratif, idéal pour la majorité des cas
2. **Le helper** `useForm` : Contrôle total, parfait pour les besoins avancés

Dans les deux cas, Inertia gère automatiquement :

- La conversion en `FormData` pour les fichiers
- La gestion des erreurs de validation Laravel
- Le tracking de progression pour les uploads
- Les états de chargement et de succès

En combinant ces outils avec **Wayfinder** et **shadcn-vue**, vous obtenez une expérience de développement moderne, type-safe, et productive. 🚀

Ressources

- Documentation Inertia.js - Forms
- Documentation Inertia.js - File Uploads
- Documentation Laravel - Validation
- Documentation Laravel - File Storage
- Projet Agence immobilière sur GitHub : https://github.com/opmvpc/immo

2.8 Pagination avec Laravel et Inertia.js

La pagination est essentielle pour afficher une grande quantité de données sur plusieurs pages tout en réduisant la charge sur le client et le serveur. Avec Laravel, la méthode `paginate()` génère des données paginées. Utilisée avec Inertia.js, elle devient encore plus simple et fluide.

Comment fonctionne la pagination avec `paginate()` ?

Laravel fournit une méthode puissante appelée `paginate()` qui renvoie :

- Une collection de données (par page).
- Des métadonnées utiles comme le nombre total d'éléments, la page actuelle, les URLs pour la navigation, etc.

Exemple de réponse JSON générée par `paginate()` :

```json
{
  "data": [
    { "id": 1, "name": "John Doe" },
    { "id": 2, "name": "Jane Smith" }
  ],
  "current_page": 1,
  "last_page": 5,
  "per_page": 10,
  "total": 50,
  "prev_page_url": null,
  "next_page_url": "http://example.com?page=2",
  "links": [
    { "url": null, "label": "&laquo; Previous", "active": false },
    { "url": "http://example.com?page=1", "label": "1", "active": true },
    { "url": "http://example.com?page=2", "label": "2", "active": false },
    { "url": "http://example.com?page=3", "label": "3", "active": false },
    { "url": "http://example.com?page=4", "label": "4", "active": false },
    { "url": "http://example.com?page=5", "label": "5", "active": false },
    { "url": "http://example.com?page=2", "label": "Next &raquo;", "active": false }
  ]
}
```

Différences entre Axios/Fetch et Inertia.js pour la pagination

Axios/Fetch	Inertia.js
Nécessite des appels API séparés pour chaque page.	Les données sont directement envoyées depuis Laravel.

Axios/Fetch	Inertia.js
Gestion manuelle des états de chargement et des erreurs.	Gestion implicite des transitions et des erreurs.
Réponse paginée doit être formatée côté client.	Les données sont directement utilisables avec les props.
Risque de duplication de logique côté frontend et backend.	La logique reste centralisée côté backend.

Avec Inertia.js, la navigation entre les pages est fluide et ne nécessite aucun rechargement, ce qui améliore l'expérience utilisateur.

Le backend avec Laravel

Laravel propose la méthode `paginate()` pour fournir une réponse paginée contenant :

- Les données pour la page en cours.
- Les métadonnées utiles (total des éléments, page actuelle, liens vers les pages précédentes et suivantes, etc.).

Exemple d'implémentation backend

```php
use App\Models\User;
use Inertia\Inertia;

Route::get('/users', function () {

users=User::paginate(10);//Renvoieunecollectionpagine´ereturnInertia::render('Users',
['users'⇒users = User::paginate(10); // Renvoie une collection paginée
    return Inertia::render('Users', ['users' ⇒
users=User::paginate(10);//Renvoieunecollectionpagine´ereturnInertia::render('Users',
['users'⇒users]);
});
```

La méthode `paginate()` fournit une réponse paginée structurée, qui sera envoyée au frontend via Inertia.

Le composant Vue.js pour la page des utilisateurs

Voici le fichier complet de la page `Users.vue`, qui affiche une liste d'utilisateurs et gère la pagination.

```vue
<script setup>
import Pagination from './Pagination.vue'; // Importation du composant Pagination

// Récupération des données paginées transmises par le backend via les props d'Inertia
defineProps(['users']);
</script>

<template>
  <div>
    <h1 class="text-xl font-bold mb-4">Liste des Utilisateurs</h1>
    <!-- Liste des utilisateurs -->
    <ul>
      <li
        v-for="user in users.data"
        :key="user.id"
        class="mb-2 p-2 border rounded bg-gray-50"
      >
        {{ user.name }}
      </li>
    </ul>
    <!-- Composant de Pagination -->
    <Pagination :pagination="users" />
  </div>
</template>
```

Le composant réutilisable de pagination

Voici un composant Vue.js complet pour gérer la pagination, basé sur les données renvoyées par `paginate()`.

VUE

```
<script setup>
import { Link } from "@inertiajs/vue3";
import { computed } from "vue";

const { pagination } = defineProps({
    pagination: {
        type: Object,
        required: true,
        validator: (obj) => {
            return (
                obj.hasOwnProperty("current_page") &&
                obj.hasOwnProperty("total") &&
                obj.hasOwnProperty("per_page")
            );
        },
    },
});

const showingFrom = computed(
    () => (pagination.current_page - 1) * pagination.per_page + 1
);
const showingTo = computed(() =>
    Math.min(pagination.current_page * pagination.per_page, pagination.total)
);
</script>

<template>
    <nav
        v-if="pagination.last_page > 1"
        role="navigation"
        aria-label="Pagination Navigation"
        class="flex items-center justify-between"
    >
        <!-- Version mobile -->
        <div class="flex justify-between flex-1 sm:hidden">
            <Link
                v-if="pagination.prev_page_url"
                :href="pagination.prev_page_url"
                preserve-scroll
                class="relative inline-flex items-center px-4 py-2 text-sm font-medium
text-gray-700 bg-white border border-gray-300 leading-5 rounded-md hover:text-gray-500
focus:outline-none focus:ring ring-gray-300 focus:border-blue-300 active:bg-gray-100
active:text-gray-700 transition ease-in-out duration-150 dark:bg-gray-800 dark:border-
gray-600 dark:text-gray-300"
            >
                Previous
            </Link>
            <span
                v-else
                class="relative inline-flex items-center px-4 py-2 text-sm font-medium
text-gray-500 bg-white border border-gray-300 cursor-default leading-5 rounded-md
dark:text-gray-600 dark:bg-gray-800 dark:border-gray-600"
```

```
            >
                Previous
            </span>

            <Link
                v-if="pagination.next_page_url"
                :href="pagination.next_page_url"
                preserve-scroll
                class="relative inline-flex items-center px-4 py-2 ml-3 text-sm font-
medium text-gray-700 bg-white border border-gray-300 leading-5 rounded-md hover:text-
gray-500 focus:outline-none focus:ring ring-gray-300 focus:border-blue-300 active:bg-
gray-100 active:text-gray-700 transition ease-in-out duration-150 dark:bg-gray-800
dark:border-gray-600 dark:text-gray-300"
            >
                Next
            </Link>
            <span
                v-else
                class="relative inline-flex items-center px-4 py-2 ml-3 text-sm font-
medium text-gray-500 bg-white border border-gray-300 cursor-default leading-5 rounded-
md dark:text-gray-600 dark:bg-gray-800 dark:border-gray-600"
            >
                Next
            </span>
        </div>

        <!-- Version desktop -->
        <div
            class="hidden sm:flex-1 sm:flex sm:items-center sm:justify-between"
        >
            <div>
                <p class="text-sm text-gray-700 leading-5 dark:text-gray-400">
                    Showing
                    <span class="font-medium">{{ showingFrom }}</span>
                    to
                    <span class="font-medium">{{ showingTo }}</span>
                    of
                    <span class="font-medium">{{ pagination.total }}</span>
                    results
                </p>
            </div>

            <div>
                <span
                    class="relative z-0 inline-flex rtl:flex-row-reverse shadow-sm
rounded-md"
                >
                    <!-- Bouton Previous -->
                    <Link
                        v-if="pagination.prev_page_url"
                        :href="pagination.prev_page_url"
                        preserve-scroll
                        class="relative inline-flex items-center px-2 py-2 text-sm
```

```
font-medium text-gray-500 bg-white border border-gray-300 rounded-l-md leading-5
hover:text-gray-400 focus:z-10 focus:outline-none focus:ring ring-gray-300
focus:border-blue-300 active:bg-gray-100 active:text-gray-500 transition ease-in-out
duration-150 dark:bg-gray-800 dark:border-gray-600"
                    >
                        <svg
                            class="w-5 h-5"
                            fill="currentColor"
                            viewBox="0 0 20 20"
                        >
                            <path
                                fill-rule="evenodd"
                                d="M12.707 5.293a1 1 0 010 1.414L9.414 10l3.293
3.293a1 1 0 01-1.414 1.414l-4-4a1 1 0 010-1.414l4-4a1 1 0 011.414 0z"
                                clip-rule="evenodd"
                            />
                        </svg>
                    </Link>
                    <span
                        v-else
                        class="relative inline-flex items-center px-2 py-2 text-sm
font-medium text-gray-500 bg-white border border-gray-300 cursor-default rounded-l-md
leading-5 dark:bg-gray-800 dark:border-gray-600"
                    >
                        <svg
                            class="w-5 h-5"
                            fill="currentColor"
                            viewBox="0 0 20 20"
                        >
                            <path
                                fill-rule="evenodd"
                                d="M12.707 5.293a1 1 0 010 1.414L9.414 10l3.293
3.293a1 1 0 01-1.414 1.414l-4-4a1 1 0 010-1.414l4-4a1 1 0 011.414 0z"
                                clip-rule="evenodd"
                            />
                        </svg>
                    </span>

                    <!-- Numéros de page -->
                    <template
                        v-for="link in pagination.links.slice(1, -1)"
                        :key="link.label"
                    >
                        <span
                            v-if="link.active"
                            class="relative inline-flex items-center px-4 py-2 -ml-px
text-sm font-medium text-blue-600 bg-blue-50 border border-gray-300 cursor-default
leading-5 dark:bg-gray-800 dark:border-gray-600"
                            >{{ link.label }}</span
                        >
                        <Link
                            v-else
                            :href="link.url"
```

```
                            preserve-scroll
                            class="relative inline-flex items-center px-4 py-2 -ml-px
text-sm font-medium text-gray-700 bg-white border border-gray-300 leading-5
hover:text-gray-500 focus:z-10 focus:outline-none focus:ring ring-gray-300
focus:border-blue-300 active:bg-gray-100 active:text-gray-700 transition ease-in-out
duration-150 dark:bg-gray-800 dark:border-gray-600 dark:text-gray-400"
                            >{{ link.label }}</Link
                        >
                    </template>

                    ←!— Bouton Next —→
                    <Link
                        v-if="pagination.next_page_url"
                        :href="pagination.next_page_url"
                        preserve-scroll
                        class="relative inline-flex items-center px-2 py-2 -ml-px
text-sm font-medium text-gray-500 bg-white border border-gray-300 rounded-r-md
leading-5 hover:text-gray-400 focus:z-10 focus:outline-none focus:ring ring-gray-300
focus:border-blue-300 active:bg-gray-100 active:text-gray-500 transition ease-in-out
duration-150 dark:bg-gray-800 dark:border-gray-600"
                    >
                        <svg
                            class="w-5 h-5"
                            fill="currentColor"
                            viewBox="0 0 20 20"
                        >
                            <path
                                fill-rule="evenodd"
                                d="M7.293 14.707a1 1 0 010-1.414L10.586 10 7.293
6.707a1 1 0 011.414-1.414l4 4a1 1 0 010 1.414l-4 4a1 1 0 01-1.414 0z"
                                clip-rule="evenodd"
                            />
                        </svg>
                    </Link>
                    <span
                        v-else
                        class="relative inline-flex items-center px-2 py-2 -ml-px
text-sm font-medium text-gray-500 bg-white border border-gray-300 cursor-default
rounded-r-md leading-5 dark:bg-gray-800 dark:border-gray-600"
                    >
                        <svg
                            class="w-5 h-5"
                            fill="currentColor"
                            viewBox="0 0 20 20"
                        >
                            <path
                                fill-rule="evenodd"
                                d="M7.293 14.707a1 1 0 010-1.414L10.586 10 7.293
6.707a1 1 0 011.414-1.414l4 4a1 1 0 010 1.414l-4 4a1 1 0 01-1.414 0z"
                                clip-rule="evenodd"
                            />
                        </svg>
                    </span>
```

```
                </span>
            </div>
        </div>
    </nav>
</template>
```

Pourquoi cette structure ?

1. **Propre et réutilisable** : Le composant Pagination est autonome et peut être utilisé partout où la pagination est nécessaire.
2. **Centralisation de la logique** : Toute la logique de pagination est gérée côté Laravel.
3. **Transparence d'Inertia.js** : Avec Inertia, les transitions entre les pages sont fluides et sans rechargement complet.

Cette structure permet de créer une expérience utilisateur agréable tout en maintenant une architecture propre et modulaire.

2.9 Gestion des policies avec Laravel et Inertia.js

Laravel fournit un puissant système de **policies** pour gérer les autorisations d'accès aux fonctionnalités d'une application. Cependant, avec **Inertia.js**, les helpers Blade comme `@can` ou `@cannot` ne sont pas disponibles. La solution consiste à transmettre explicitement les permissions au frontend sous forme de **props**.

Comment fonctionnent les policies dans Laravel ?

Une **policy** dans Laravel définit les règles d'autorisation pour un modèle. Par exemple, pour un modèle `Post`, une policy pourrait définir si un utilisateur peut éditer ou supprimer un article :

```php
public function edit(User user,Postuser, Post user,Postpost)
{
    return user->id===user→id === user->id===post→user_id;
}
```

Avec Blade, on utiliserait cette policy ainsi :

```php
@can('edit', $post)
    <a href="/posts/{{ $post→id }}/edit">Edit</a>
@endcan
```

Avec **Inertia.js**, il faut transmettre ces autorisations au frontend via les props.

Transmettre les permissions au frontend

Dans le backend, les autorisations sont vérifiées avec les méthodes des policies et envoyées au composant **Vue.js** via les props :

```php
use App\Models\Post;
use Illuminate\Support\Facades\Gate;
use Inertia\Inertia;

Route::get('/posts/{post}', function (Post $post) {
    return Inertia::render('PostDetails', [
        'post' => $post,
        'can' => [
            'edit' => Gate::allows('edit', $post),
            'delete' => Gate::allows('delete', $post),
        ],
    ]);
});
```

Ici, `Gate::allows` vérifie si l'utilisateur peut exécuter une action spécifique sur le modèle, et les résultats sont transmis comme un objet `can`.

Utilisation dans le composant Vue.js

Dans le composant **Vue.js**, ces autorisations sont utilisées pour conditionner l'affichage :

```vue
<script setup>
defineProps(['post', 'can']);

function deletePost() {
  if (confirm('Are you sure you want to delete this post?')) {
    // Appel pour supprimer le post
  }
}
</script>

<template>
  <div>
    <h1 class="text-xl font-bold">{{ post.title }}</h1>
    <p class="mb-4">{{ post.content }}</p>

    <div class="flex space-x-2">
      <inertia-link
        v-if="can.edit"
        :href="`/posts/${post.id}/edit`"
        class="px-4 py-2 bg-blue-500 text-white rounded"
      >
        Edit
      </inertia-link>

      <button
        v-if="can.delete"
        @click="deletePost"
        class="px-4 py-2 bg-red-500 text-white rounded"
      >
        Delete
      </button>
    </div>
  </div>
</template>
```

Avantages

- **Centralisation des autorisations** : Toute la logique d'autorisation reste gérée côté Laravel.
- **Clarté et modularité** : Les permissions sont transmises explicitement, rendant le code plus lisible et maintenable.
- **Flexibilité** : Cette approche permet de gérer des autorisations dynamiques basées sur des rôles, des statuts ou d'autres critères.

En transmettant les permissions comme des props, on maintient une séparation claire entre la logique métier et la présentation tout en respectant les principes d'Inertia.js.

2.10 Exercice : Application Portfolio Personnel

Objectif

Développer une application web complète permettant de gérer et d'afficher un portfolio professionnel. L'application comprendra un **backoffice** pour administrer le contenu et un **frontoffice** créatif pour présenter vos réalisations.

Partie 1 : Backoffice - Gestion du Contenu

1.1 Gestion des Projets

Chaque projet contient :

Informations principales :

- Titre, slug (auto-généré), description courte, description détaillée (WYSIWYG)
- Type (Site web, App mobile, API, Design, etc.), Statut (Brouillon/Publié/Archivé)
- Date de réalisation, Client, Contexte (Scolaire/Stage/Freelance/Personnel), Durée, Rôle
- Lien projet, Lien GitHub, Ordre d'affichage

Relations :

- Technologies utilisées (relation many-to-many)
- Images : 1 image principale + galerie (plusieurs images)

1.2 Gestion des Technologies

- Nom, Catégorie (Backend/Frontend/BDD/DevOps/Design), Niveau (Débutant/Intermédiaire/Avancé/Expert)
- Logo (upload), Couleur (hex), Ordre d'affichage

1.3 Gestion des Expériences

- Poste, Entreprise, Lieu, Date début/fin, Description (WYSIWYG)
- Type (Stage/CDI/CDD/Freelance/Alternance)

1.4 Gestion de la Formation

- Diplôme, Établissement, Lieu, Date début/fin, Description

1.5 Informations Personnelles

- Nom, Titre professionnel, Bio (WYSIWYG), Photo de profil
- Email, Téléphone, Localisation, Disponibilité
- CV (upload PDF)
- Liens sociaux (LinkedIn, GitHub, Twitter, etc.)

Partie 2 : Fonctionnalités Techniques

2.1 Éditeur WYSIWYG

Intégrer un éditeur de texte riche pour les descriptions. Options suggérées :

- **Tiptap** (moderne, headless, flexible mais plus de config)
- **Quill** (simple, léger)
- **CKEditor 5** (complet)
- **TinyMCE** (populaire)

Fonctionnalités minimales : formatage texte, titres, listes, liens, blocs de code.

2.2 Upload d'Images

- Upload multiple avec progression
- Validation (format, max 5MB)
- Suppression individuelle avec confirmation
- Réorganisation des images (bonus)

2.3 Pagination et Recherche

Pour la liste des projets :

- Pagination (10-15 par page)
- Recherche (titre, description, technologies)
- Filtres (Statut, Type, Année)
- Tri (date, ordre d'affichage, alphabétique)

2.4 Interface

Utiliser shadcn-vue pour créer une interface professionnelle (tableaux, formulaires, cards, dialogs, sidebar).

Partie 3 : Frontoffice - Affichage Public

3.1 Page d'Accueil

Présentation attractive : nom, titre, photo, phrase d'accroche, technologies principales, CTA vers projets/contact.

3.2 Page Projets

Grille de projets avec image, titre, description courte, technologies. Filtres par type/technologie. Lien vers détail.

3.3 Page Détail Projet

Affichage complet : galerie d'images, description détaillée, technologies, infos (client, durée, contexte), liens externes, navigation précédent/suivant.

3.4 Page À Propos

Bio, parcours académique (timeline), expériences (timeline), technologies avec niveaux visualisés, lien téléchargement CV.

3.5 Page Contact

Formulaire simple (nom, email, sujet, message) avec envoi par email Laravel Mail. Protection anti-spam (honeypot).

Partie 4 : Fonctionnalités Optionnelles

Ces fonctionnalités augmenteront la qualité du projet mais ne sont pas obligatoires :

4.1 Blog Technique

Section blog : articles avec titre, contenu (WYSIWYG), image, catégories/tags. Liste paginée + page détail.

4.2 Google Analytics

Intégration Google Analytics pour tracking des visites. Affichage de statistiques basiques dans le backoffice.

4.3 Optimisation SEO

- Meta tags personnalisables (title, description)

- Open Graph pour partage réseaux sociaux
- URLs propres (slugs)
- Images optimisées (compression, lazy loading)

Stack Technique

- Backend : Laravel 12
- Frontend : Vue.js 3 (Composition API) + Inertia.js
- Starter Kit : Laravel Vue Starter Kit
- UI : shadcn-vue + Tailwind CSS v4
- Base de données : MySQL
- Auth : Laravel built-in

Bonnes pratiques :

- Utiliser Wayfinder pour les routes
- Utiliser le composant `<Form>` d'Inertia
- Code organisé, commits Git réguliers

Design du Frontoffice

Le design est **totalement libre**. C'est l'occasion de montrer votre créativité et votre savoir-faire. Le frontoffice doit être :

- **Original** : Créez quelque chose d'unique qui vous représente
- **Responsive** : Fonctionnel sur mobile, tablette, desktop
- **Professionnel** : Soigné, moderne, sans fautes
- **Performant** : Images optimisées, navigation fluide

Inspirez-vous de portfolios existants mais ne copiez pas. Ce portfolio vous servira pour vos candidatures !

2.11 SEO On-Page – Séduire Google (et les Humains)

Le SEO On-Page, c'est l'art de dire à Google : *"Eh, regarde ! Ma page est super bien rangée et elle parle exactement de ce que cherche cet utilisateur."*

Si votre code est une jungle, Google passera son chemin. Voici les règles d'or pour structurer votre contenu sans vous prendre la tête.

I. Le Squelette : Structurez votre HTML (Hn)

Imaginez votre page comme un livre. Google a besoin d'un sommaire pour comprendre de quoi vous parlez. C'est le rôle des balises `H1`, `H2`, `H3`...

- **H1 (Le Titre du Livre) :** Il n'y en a **qu'un seul** par page. Il doit contenir votre mot-clé principal.
- **H2 (Les Chapitres) :** Ils séparent les grandes idées.
- **H3 (Les Sous-parties) :** Pour aller dans le détail.

> ***La Règle d'Or :*** *Ne sautez jamais de niveau (pas de H4 direct après un H1) et n'utilisez pas les titres juste pour "mettre en gros". C'est du sens, pas de la déco.*

Comparatif Express :

✅ Le bon élève	❌ Le mauvais élève
`<h1>` Recette du Gâteau au Chocolat `</h1>`	`<h1>` Bienvenue `</h1>`
`<h2>` Les Ingrédients `</h2>`	`<h2>` Menu `</h2>`
`<h3>` Choix du chocolat `</h3>`	`<h4>` Mentions Légales `</h4>`
Hiérarchie logique et descriptive.	*Aucun sens, titres génériques.*

II. La Vitrine : Meta Title & Description

C'est ce qui apparaît dans les résultats de recherche (la SERP). C'est votre **seule chance** de convaincre l'internaute de cliquer sur votre lien plutôt que sur celui du voisin.

1. Le Meta Title (L'enseigne du magasin)

C'est le facteur le plus important pour le classement.

- **Longueur :** 50-60 caractères (sinon Google le coupe...).
- **Contenu :** Mot-clé principal au début + une accroche.

2. La Meta Description (Le vendeur)

Elle ne joue pas directement sur le classement, mais elle booste votre taux de clic (CTR). Donnez envie !

L'exemple qui tue :

- **Mauvais (L'ennui total) :**
  - Title: `Page Recette 01`

- Desc: `Voici une recette de gâteau.`
- *Résultat : Personne ne clique.*

- **Excellent (Sexy & Précis) :**
  - Title: `Gâteau au Chocolat Moelleux : Recette Facile en 15 min`
  - Desc: `Envie de chocolat ? Testez notre recette inratable ! Croustillant dehors, fondant dedans. Pas besoin d'être un chef pour régaler vos amis.`
  - *Résultat : On salive, on clique.*

III. Les Yeux de Google : Balises Alt (Images)

Google est aveugle. Si vous ne lui dites pas ce qu'il y a sur l'image, il ne voit qu'un fichier nommé `IMG_5422.jpg`.

La balise `alt` sert à deux choses : le référencement (Google Images) et l'accessibilité (lecteurs d'écran pour malvoyants).

- ❌ `<img src="choco.jpg" alt=""/>` (Le vide intersidéral)
- ✅ `<img src="choco.jpg" alt="Part de gâteau au chocolat fondant avec une framboise">` (Descriptif et précis)

IV. Le Carburant : Les Mots-Clés

Le temps où l'on répétait 50 fois le même mot est révolu (c'était en 2005, passez à autre chose). Aujourd'hui, on vise **l'intention de recherche**.

Comment les choisir ?

Mettez-vous à la place de l'utilisateur. Que tape-t-il dans la barre de recherche ?

- **Trop générique :** "Chocolat" (Trop de concurrence, intention floue).
- **La pépite (Longue traîne) :** "Recette gâteau chocolat sans gluten rapide". Moins de volume, mais 100% pertinent.

Où les placer ?

Saupoudrez-les naturellement, comme du sel :

1. Dans le **H1** (Indispensable).
2. Dans l'introduction (Le premier paragraphe).

3. Dans un ou deux **H2**.
4. Dans la balise **Title** et l'**URL**.

> ⚠️ *Attention au "Keyword Stuffing" :*
>
> *"Ce gâteau au chocolat est le meilleur gâteau au chocolat pour manger du chocolat."*
>
> *→ Google vous punira pour spam. Écrivez pour les humains d'abord !*

📝 Exercice : "Pimp My Page"

À vous de jouer ! Prenez une page de votre projet actuel et faites-lui un relooking extrême :

1. **Check-up Structure :** Vérifiez que vous avez un seul H1 unique et pertinent.
2. **Opération Séduction :** Réécrivez votre `<title>` et votre `meta description`. Soyez vendeur, soyez précis.
3. **Chasse aux fantômes :** Ajoutez des balises `alt` descriptives à toutes vos images.

2.12 Open Graph – Ne soyez pas moche sur les réseaux

Vous avez soigné votre SEO pour plaire à Google. Super. Mais que se passe-t-il quand quelqu'un partage votre lien sur Facebook, LinkedIn, WhatsApp ou Slack ?

Sans Open Graph (OG), c'est la roulette russe : une image coupée, un titre bizarre ou pire... rien du tout.

Avec les balises OG, vous prenez le contrôle de votre apparence. C'est la différence entre tendre une carte de visite dorée et gribouiller son nom sur une serviette en papier.

I. C'est quoi ce truc ?

Le protocole "Open Graph" a été inventé par Facebook, mais aujourd'hui **tous** les réseaux sociaux l'utilisent (LinkedIn, Twitter/X, Discord, etc.).

C'est simple : ce sont des balises cachées dans le `<head>` de votre code qui disent aux réseaux sociaux :

> *"Eh ! Quand on partage cette page, affiche CETTE image et CE titre, pas n'importe quoi."*



II. Les 4 Fantastiques (Les Balises Indispensables)

Vous n'avez besoin que de 4 balises pour transformer l'apparence de vos liens.

1. `og:image` (La Star)

C'est l'élément le plus important. Une image vaut 1000 mots, surtout sur un fil d'actualité où l'on scrolle à toute vitesse.

- **Le conseil :** Utilisez une image spécifique, aux bonnes dimensions (idéalement **1200 × 630 pixels**).
- **L'erreur :** Laisser le réseau social choisir une image au hasard (souvent votre logo tout petit ou une icône de pied de page).

2. `og:title` (L'Accroche)

Similaire à votre balise `<title>` SEO, mais vous pouvez être plus "clicky" ou conversationnel ici.

- **Exemple :** "Découvrez notre recette !" (Un peu mou) vs "Le Gâteau au Chocolat qui a cassé Internet 🍫" (Mieux).

3. `og:description` (Le Teasing)

Un court résumé (2-3 lignes max). Contrairement à la meta description Google, celle-ci est souvent affichée en gros sous le titre sur Facebook/LinkedIn.

4. `og:url` (L'Adresse Officielle)

L'URL canonique de la page (sans les paramètres de tracking bizarres).

III. Le Choc des Photos : Avant / Après

Visualisez le partage de votre lien sur un fil LinkedIn ou dans une conversation WhatsApp :

🤢 Sans Open Graph (Le mode "Loto")	😎 Avec Open Graph (Le mode "Pro")
Image : Aucune, ou une icône pixélisée coupée en deux.	**Image :** Une grande bannière visuelle qui claque.
Titre : "Accueil" ou "index.html".	**Titre :** "Boostez votre Productivité : Le Guide".
Desc : "Copyright 2024... Mentions légales..." (Il a pris le premier texte qu'il a trouvé).	**Desc :** "5 méthodes simples pour gagner 2h par jour."
Résultat : On dirait un site cassé ou un spam.	*Résultat : On a confiance, on clique.*

IV. Code : À quoi ça ressemble ?

C'est très simple à ajouter dans votre HTML (`<head>`) :

HTML

```xml
<meta property="og:title" content="Le Guide Ultime du SEO" />
<meta property="og:type" content="article" />
<meta property="og:url" content="https://monsite.com/guide-seo" />
<meta property="og:image" content="https://monsite.com/images/banner-social.jpg" />
<meta property="og:description" content="Tout comprendre au référencement en 5 minutes chrono." />
```

> 💡 *Le cousin Twitter :*
>
> *Twitter (X) utilise l'Open Graph, mais préfère ses propres balises pour un rendu optimal. Ajoutez simplement :*
>
> *<meta name="twitter:card" content="summary_large_image">*
>
> *Cela force l'affichage de la grande image.*

V. L'Outil Magique : Le Debugger

Vous avez ajouté vos balises mais l'image ne s'affiche pas ? C'est souvent une histoire de **cache**. Les réseaux sociaux gardent en mémoire l'ancienne version de votre page.

Pour forcer la mise à jour et prévisualiser le résultat, utilisez les outils officiels (gratuits) :

1. Pour Facebook & Messenger :

   👉 Facebook Sharing Debugger (La référence absolue)

2. Pour LinkedIn :

   👉 LinkedIn Post Inspector (Indispensable pour le B2B)

3. Pour Twitter / X :

   👉 Twitter Card Validator

La méthode : Copiez votre URL dedans, cliquez sur "Debug" ou "Inspect", et hop ! Vous voyez exactement ce que verront vos utilisateurs. Si l'image n'apparaît pas, cliquez sur le bouton "Scrape Again" pour forcer le robot à relire votre page.

❗ Attention, il faut que le site soit en ligne ! Pas de localhost.

📝 Exercice Flash

1. Allez sur le **Facebook Sharing Debugger**.
2. Entrez l'URL de votre site préféré (ou le vôtre).
3. Regardez la "Card" générée.
   - L'image est-elle coupée ?
   - Le titre donne-t-il envie ?
4. Si c'était votre site, quelle image auriez-vous choisie pour obtenir plus de clics ?

Et voilà pour la technique !

On a couvert la **structure** (SEO On-Page) et **l'apparence sociale** (Open Graph).

2.13 Intégration des Outils Google

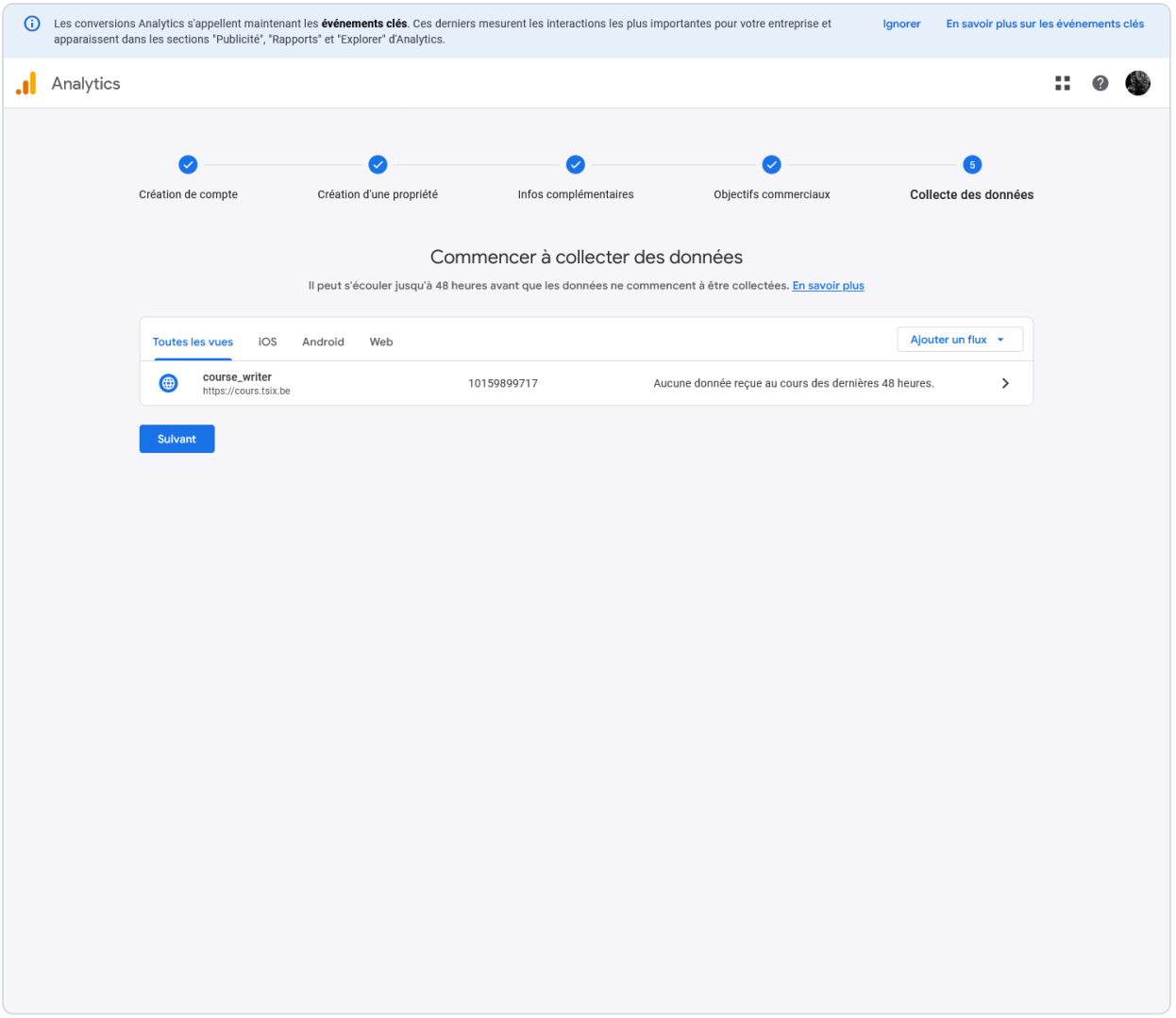
Pourquoi Google Analytics et Search Console ?

Si vous comptez avancer à l'aveugle, libre à vous de **ne rien installer**. Mais si, par chance, vous voulez un minimum de visibilité sur vos performances (et vos échecs), vous allez avoir besoin de **Google Analytics (GA)** et **Google Search Console (GSC)**.

- **Google Analytics** : Mesure le trafic de votre site (nombre de visiteurs, pages vues, taux de rebond, etc.).
- **Google Search Console** : Surveille l'indexation par Google, vos positions, votre CTR (taux de clic), et détecte les erreurs (erreurs 404, pages bloquées, etc.).

Mise en Place de Google Analytics

1. **Création du Compte**
    - Rendez-vous sur Google Analytics et suivez les instructions pour créer un compte.
    - Choisissez **GA4** (la nouvelle version) ou configurez la "propriété" en Universal Analytics si vous avez des besoins spécifiques (mais GA4 est la norme actuelle).



2. Installation du Script sur Votre Site

- **Bon exemple** :

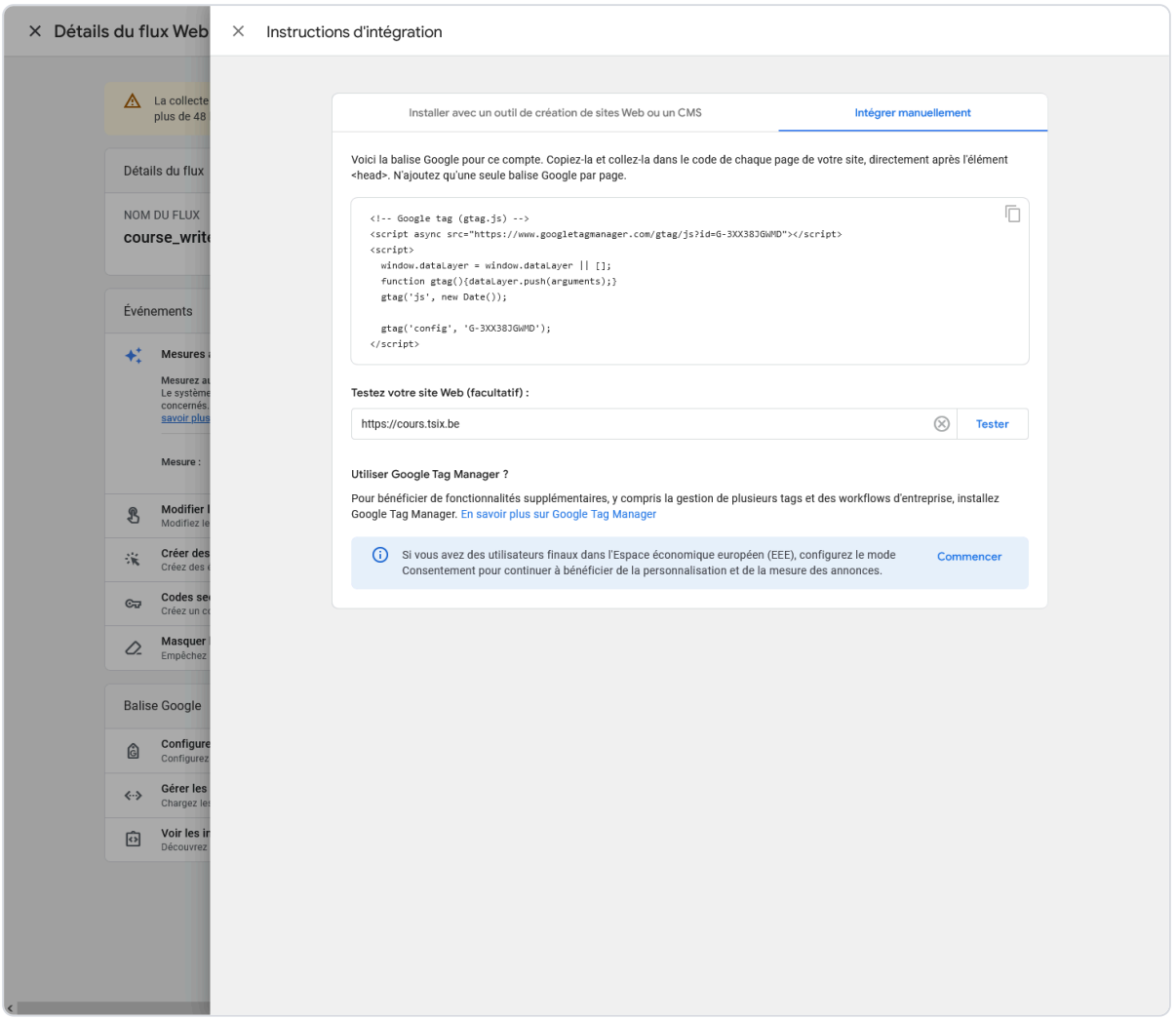
```XML
<!-- Google tag (gtag.js) -->
<script async src="https://www.googletagmanager.com/gtag/js?id=G-XXXXXXXXXX">
</script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'G-XXXXXXXXXX');
</script>
```

  - Script placé dans le `<head>` (avant la balise `</head>`) pour charger rapidement et ne pas perturber vos stats.

- **Mauvais exemple** :

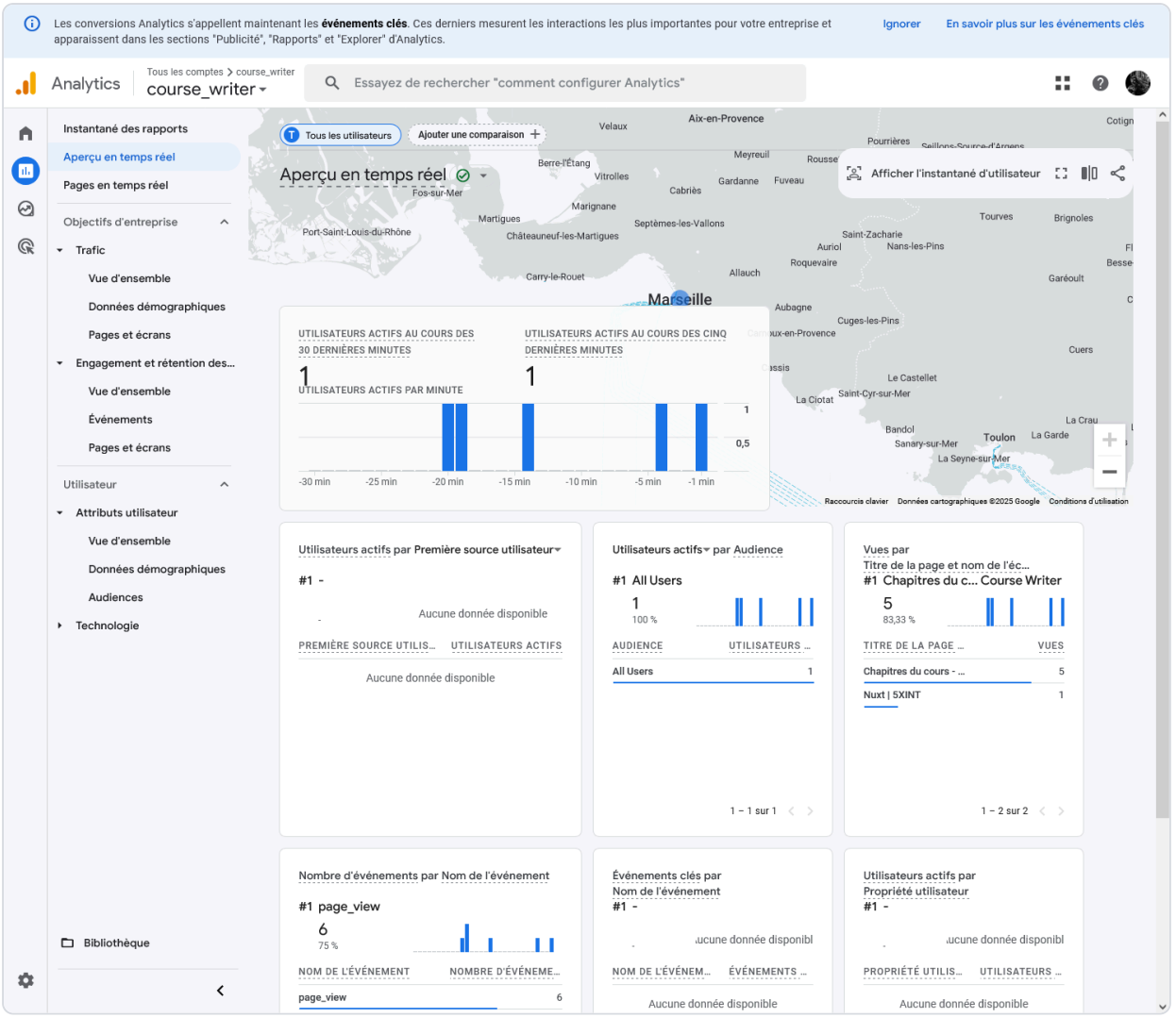
```xml
<!-- Code balancé n'importe où, ou manquant -->
<script>
  gtag('config', 'G-XXXXXXXXXX'); // sans avoir chargé gtag.js
</script>
```

- Si vous oubliez d’inclure la bibliothèque, Analytics ne pourra pas collecter vos données.



3. Vérification

- Accédez à votre interface GA et vérifiez si vous voyez de la “vraie” activité en temps réel (ou testez en ouvrant le site dans un autre onglet).
- Si vous voyez zéro visite malgré vos tests, vous avez peut-être tout cassé.



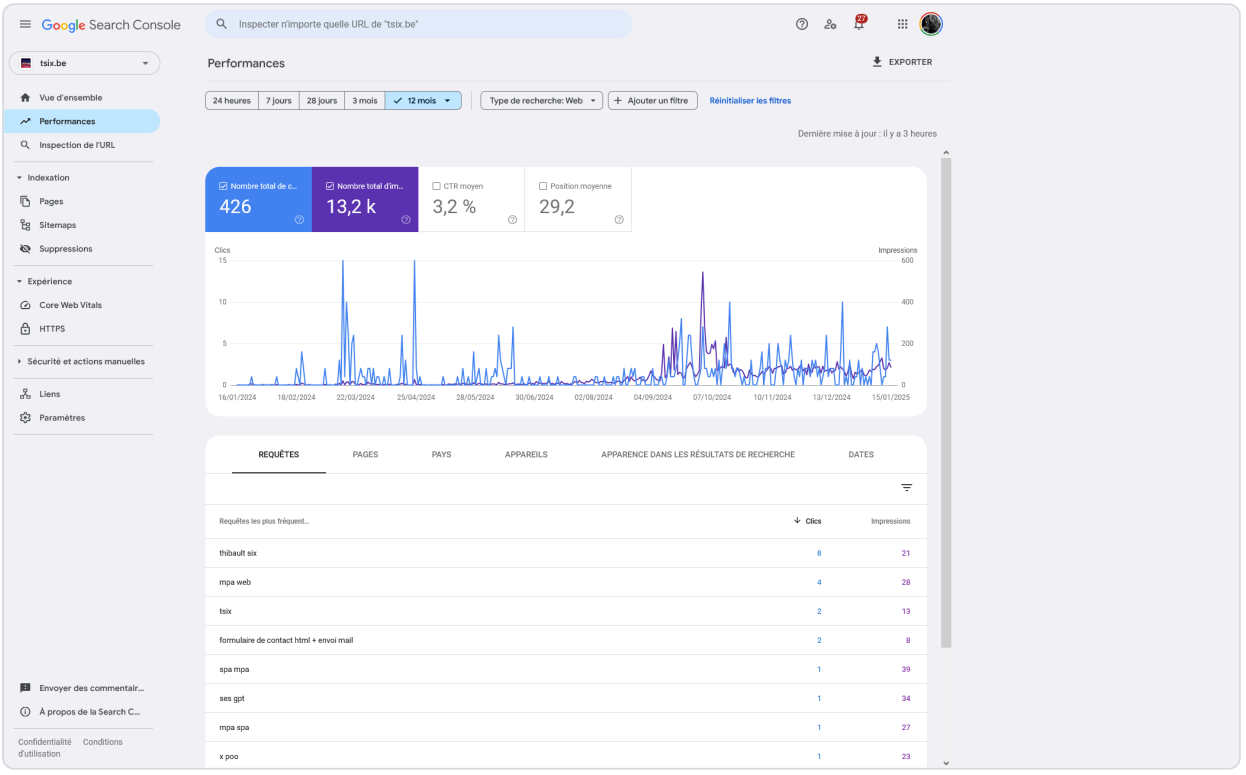
Mise en Place de Google Search Console

1. **Création et Validation de la Propriété**
   - Rendez-vous sur Google Search Console.
   - Ajoutez votre site en tant que “propriété” (vous pouvez choisir la validation via **fichier HTML**, **meta tag**, ou **DNS**).
   - **Bon exemple** (meta tag dans `<head>`) :

     ```XML
     <meta name="google-site-verification" content="votre_code_verification" />
     ```

   - **Mauvais exemple** :
     - Ne jamais finaliser la validation, ou mettre la balise dans un `footer`. Google ne la verra pas avant l’extinction du soleil.

2. **Soumission du Sitemap** (optionnel mais utile)
    - Un **sitemap** au format XML peut être généré par des plugins ou des outils en ligne (surtout si vous utilisez un CMS).
    - Dans la Search Console, allez dans **“Sitemaps”** et entrez l’URL de votre sitemap (ex : `https://votresite.com/sitemap.xml`).
    - Google indexera plus facilement vos pages si votre sitemap est structuré.
3. **Analyse Rapide**
    - Observez l’onglet **Couverture** pour détecter les erreurs d’indexation.
    - Consultez l’onglet **Performances** pour voir vos mots-clés, votre position moyenne et votre CTR.
    - Ignorez les avertissements “mineurs” si vous manquez de temps, mais ne négligez pas les erreurs critiques (pages introuvables, blocages robots.txt…).



Interpréter les Données

1. **Google Analytics**
    - **Taux de rebond** : Pourcentage de visiteurs qui quittent le site après une seule page vue. Un taux très élevé peut signifier que vos pages n’intéressent personne (ou qu’elles chargent trop lentement).

- **Pages vues** : Vous pourrez voir quelles pages sont les plus populaires (et lesquelles sont l'équivalent numérique d'un désert).
- **Origines du trafic** : Savoir si vos visiteurs viennent de Google, d'un lien externe, ou d'un flux astral inconnu.

2. **Search Console**
   - **Impressions** : Nombre de fois où votre site apparaît dans la SERP.
   - **CTR (Click Through Rate)** : Si les utilisateurs cliquent ou pas. Un CTR très bas peut signifier que vos titres et descriptions sont à dormir debout.
   - **Position Moyenne** : Rang moyen de votre site sur les mots-clés détectés.
   - **Bon exemple** : Chercher à améliorer un mot-clé où vous êtes en 12ème position pour viser le top 10.
   - **Mauvais exemple** : Vouloir ranker 1er sur "chocolat" sans stratégie et s'étonner d'être invisible.

Exercice Pratique

1. **Installez GA et GSC**
   - Copiez le script GA4 ou Universal Analytics dans votre `<head>`.
   - Configurez Search Console, validez votre propriété.
2. **Observez vos Stats**
   - Ouvrez votre site sur un autre appareil, allez dans GA > Temps réel. Vérifiez que votre visite est comptabilisée.
   - Dans GSC, patientez un peu (quelques jours) avant de voir des données utiles.

Sans ces outils, vous naviguerez à vue, ce qui est un pari risqué quand on veut optimiser un site. Prochaine étape : se pencher sur **l'importance du contenu**, car sans contenu correct, vous n'aurez pas grand-chose à mesurer.

2.14 🚀 Devoir : La Stratégie SEO de votre Portfolio

Contexte

Vous êtes en train de construire votre site portfolio. C'est votre carte de visite numérique. Mais un beau site ne sert à rien si personne ne le trouve ou si le lien ne donne pas envie de cliquer.

L'objectif de ce travail est de préparer le terrain pour que Google (et les recruteurs) tombent amoureux de votre profil.

Ce que vous devez faire

Créez un document synthétique dans lequel vous définissez la stratégie de référencement de votre futur site.

Votre document doit contenir les **3 parties suivantes** :

1. Votre identité (Mots-Clés)

Listez **5 à 10 mots-clés ou expressions** qui définissent votre profil professionnel et que vous aimeriez cibler.

- *Exemple : "Développeur Fullstack Junior", "Webdesigner Liège", "React & Node.js", etc.*

2. Structure & Optimisation (Le Cœur du devoir)

Pour **3 pages principales** de votre portfolio (par exemple : *Accueil, À Propos, Mes Projets*), rédigez le contenu qui apparaîtra dans les résultats de recherche :

- **Nom de la page**
- **Le Meta Title** (50-60 caractères max) : Il doit être accrocheur et contenir un mot-clé.
- **La Meta Description** (150-160 caractères max) : Elle doit "vendre" le clic et résumer le contenu de la page.

3. Bonus IA 🤖

Vous avez le droit (et même l'encouragement) d'utiliser des outils comme ChatGPT, Claude ou Gemini pour vous aider à trouver des idées ou reformuler vos phrases.

- *Indiquez brièvement si vous avez utilisé une IA et pour quelle partie (ex: "J'ai utilisé ChatGPT pour générer des variantes de mes titres").*

Modalités de rendu

- **Format :** Fichier **PDF** uniquement.
- **Où rendre :** Dans le devoir **Teams** prévu à cet effet.
- **Date limite :** 📅 **24 novembre 2025 à 17h15** (précises !).

Bon travail à tous !

CHAPITRE 3

Web A11y : Le Guide de Survie (Laravel + Inertia)

Vous pensez que l'accessibilité (a11y), c'est juste ajouter des textes alternatifs sur vos images et prier pour que ça passe ? Raté. L'accessibilité, c'est l'art de coder des applications qui ne laissent personne sur la touche.

Dans ce cours, on ne va pas réciter la documentation du W3C par cœur. On va prendre votre stack préférée — **Laravel, Inertia, Vue et Shadcn** — et on va voir concrètement comment construire des interfaces modernes, robustes et utilisables par **tous** (que ce soit un utilisateur aveugle, une personne atteinte de TDAH, ou juste vous quand vous naviguez en plein soleil).

Fini les excuses. On rend le web meilleur, une ligne de code à la fois. 🔥



3.1 L'Accessibilité (a11y) : Coder pour les humains (les vrais) 🌍

Slides

https://gamma.app/docs/LAccessibilite-Web-a11y--rz7jd9f9xywvprl

Introduction

L'accessibilité web est souvent abrégée a11y (car il y a 11 lettres entre le a et le y dans le mot “accessibility”).

Ce n'est pas une "feature bonus" pour la fin de sprint. C'est la base.

Si votre site n'est pas accessible, vous bloquez la porte à une immense partie de la population.



Pourquoi s'en préoccuper ? (Data & Business) 🧠

L'**OMS** estime que **15% de la population mondiale** vit avec une forme de handicap.

- 📉 C'est **1 milliard** de personnes.
- 🚫 Un site non accessible = **1 utilisateur sur 7** refusé à l'entrée.

L'effet "Curb Cut" (Bateau de trottoir) 🛹

L'accessibilité aide tout le monde, pas juste les personnes handicapées.

C'est le principe du bateau de trottoir :

1. Conçu pour les **fauteuils roulants**. 🦽
2. Utilisé par les **livreurs**, les **poussettes** et les **skateurs**. 🛹



Les 4 Fantastiques (Types de handicap) 🦸

Pour coder juste, il faut comprendre les contraintes. Voici les 4 catégories majeures et leur impact technique.

1. 👁️ Visuel (Aveugles, Malvoyants, Daltoniens)

- **Outil principal :** Le **Lecteur d'écran** (Screen Reader). Il lit le HTML à haute voix.
- **Le problème :** Un bouton `div` sans label est invisible pour eux.
- **Contraste :** Le texte gris clair sur fond blanc est illisible pour un malvoyant (ou quelqu'un en plein soleil).

Outils à connaître :

- NVDA (Gratuit, Windows) - *Le standard.*
- VoiceOver (Natif Mac/iOS).
- JAWS (Payant, très utilisé en entreprise).

2. 👂 Auditif (Sourds, Malentendants)

- **Impact :** Vidéos et sons.
- **Solution :** Sous-titres et transcriptions.
- **Bonus pour tous :** Regarder une vidéo dans le métro sans écouteurs. 🚇

3. 🦾 Moteur (Parkinson, Paralysie, Bras cassé)

Certains utilisateurs **ne peuvent pas utiliser de souris**. Ils naviguent **uniquement au clavier**.

- `Tab` pour avancer.
- `Shift + Tab` pour reculer.
- `Enter` / `Space` pour valider.

> ***Test immédiat :*** *Lâchez votre souris. Essayez d'ouvrir un lien sur cette page juste avec* `Tab` *et* `Enter`*. Si vous n'y arrivez pas, le site est cassé.*

4. 🧠 Cognitif (TDAH, Dyslexie, Troubles mnésiques)

C'est ici que l'UX rencontre le code.

- **TDAH (ADHD) :**
  - ❌ Murs de texte compacts.
  - ❌ Animations qui clignotent partout (distraction).
  - ✅ Feedback clair (ex: "Erreur sur le champ Email" vs "Erreur").
- **Dyslexie :**
  - ❌ Polices "artistiques" ou serif complexes.
  - ❌ Texte justifié (crée des "rivières" blanches illisibles).
  - ✅ Texte aligné à gauche.



Notre Stack & L'Accessibilité 🛠️

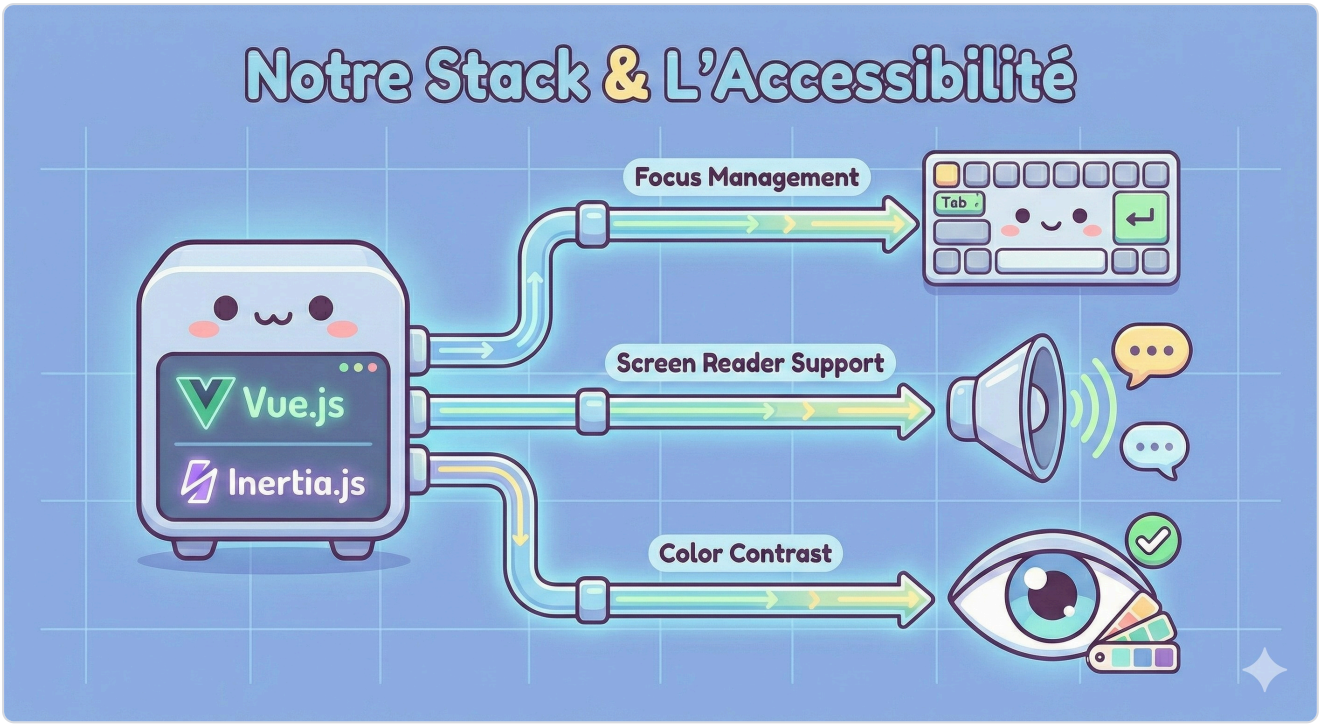
Un framework ne fait pas de magie. On peut créer un site horrible avec les meilleurs outils.

Heureusement, notre stack part avec une longueur d'avance :

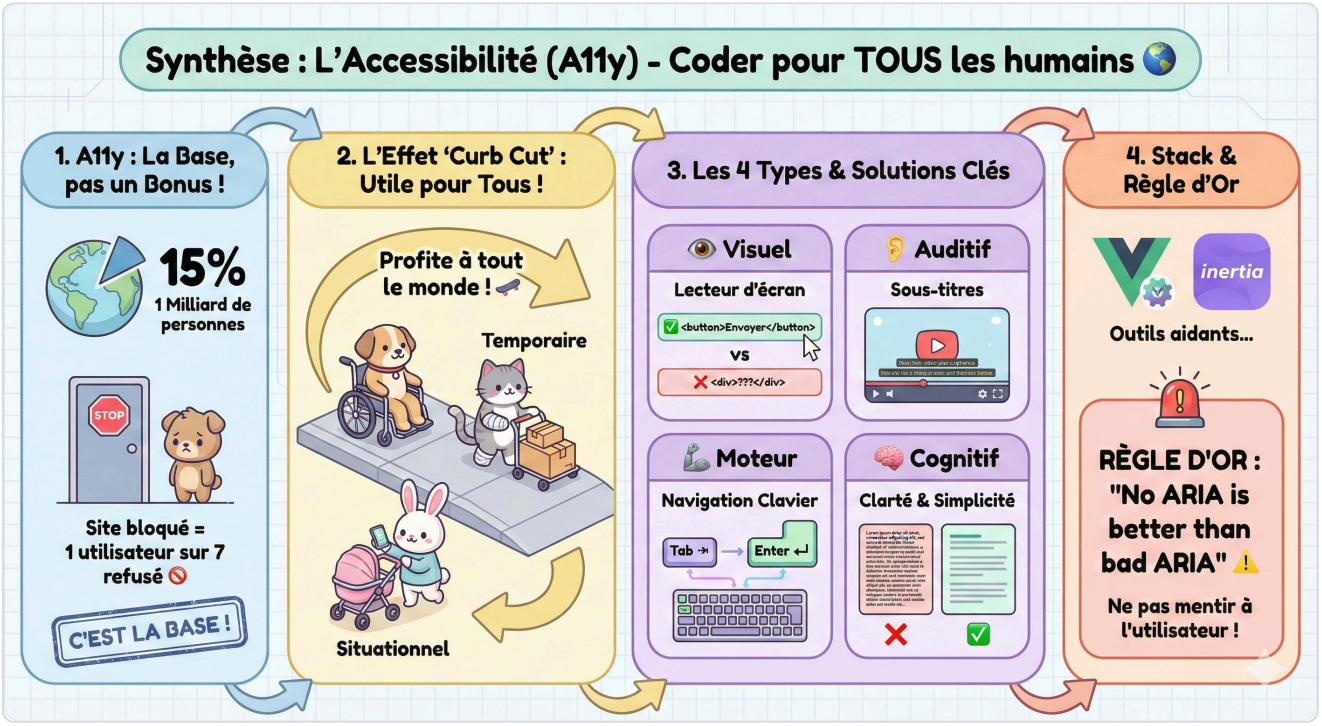
1. **Shadcn UI :** Basé sur **Radix UI**. Gère nativement le *Focus Trap* (garder le clavier dans une modale) et la navigation fléchée. 🏹
2. **Inertia** `<Form>` : Permet une gestion d'état propre sans bidouillage excessif du DOM.
3. **Laravel :** La validation backend renvoie des erreurs structurées qu'on peut facilement lier aux inputs avec `aria-describedby`.

> ⚠️ *La Règle d'Or :*
>
> *"No ARIA is better than bad ARIA"*
>
> *Mieux vaut aucun attribut d'accessibilité qu'un attribut faux qui ment à l'utilisateur.*

Dans la suite, on va configurer nos formulaires Inertia pour qu'ils soient parfaits pour **tout le monde**.



Synthèse



3.2 Les Fondations : Sémantique & Structure 🏗️

Avant de toucher au JavaScript ou à Inertia, tout se joue dans le HTML.

Les navigateurs et les lecteurs d'écran sont intelligents, mais ils ne sont pas devins. Si vous utilisez une pour tout, ils sont aveugles.

Slides

https://gamma.app/docs/Les-Fondations-Semantique-Structure--xqowcs7jy2fczc5

1. HTML Sémantique : Parler la bonne langue 🗣️

Le HTML sémantique donne du **sens** au contenu, pas juste du style.

- `<div>` et `<span>` : Ce sont des boîtes vides. Elles ne disent rien. "C'est juste un truc".
- `<button>`, `<nav>`, `<article>` : Ils disent ce qu'ils sont et comment interagir avec eux.

Le test du "Div Button" ❌

Beaucoup de devs font ça :

HTML

```xml
<div class="bg-blue-500 text-white p-2 cursor-pointer" onclick="submit()">
  Envoyer
</div>
```

- ⌨️ **Clavier :** Le focus ne s'arrête pas dessus (pas de `Tab`).
- 🗣️ **Lecteur d'écran :** Dit "Groupe". L'utilisateur ne sait pas que c'est cliquable.
- 🖱️ **Interaction :** Pas de support pour la touche `Entrée` ou `Espace`.

La solution propre :

HTML

```xml
<button class="bg-blue-500 text-white p-2" type="submit">
  Envoyer
</button>
```

Le navigateur gère tout le reste (focus, click, clavier) gratuitement. Merci le navigateur.



2. La Hiérarchie des Titres (`h1` - `h6`) 🗺️

Les utilisateurs de lecteurs d'écran naviguent souvent en sautant de titre en titre (raccourci `H`). C'est leur table des matières.

- ✅ **Un seul** `<h1>` par page (le titre principal).
- ✅ **Pas de sauts** (ne pas passer de `h2` à `h4`).
- ❌ Ne jamais utiliser un `h3` juste parce que vous voulez que le texte soit "moyen gras". Utilisez une classe CSS pour ça (ex: `text-lg font-bold`).

> 🔗 ***Ressource :*** *MDN - Les éléments de titre*

3. Le "Skip Link" (Lien d'évitement) ⏭️

C'est vital pour la navigation au clavier.

Imaginez : vous arrivez sur une page, vous voulez lire le contenu principal, mais vous devez d'abord appuyer 45 fois sur Tab pour passer tous les liens du menu de navigation. L'enfer.

La solution : Un lien caché tout en haut de la page qui apparaît au focus et permet de sauter directement au contenu.

Implémentation dans `AppLayout.vue`

Dans une SPA (Single Page App) comme Inertia, on a une subtilité : quand on change de page, il faut gérer le focus.

Extrait de code

```XML
<script setup>
import { ref, watch, nextTick } from 'vue';
import { useRoute } from 'vue-router'; // Ou via Inertia usePage si besoin

// 1. Référence pour le conteneur principal
const mainContentRef = ref(null);
const route = useRoute();

// 2. Reset du focus au changement de page
watch(
  () => route.path,
  async () => {
    // On attend que le DOM soit mis à jour
    await nextTick();
    // On met le focus sur le wrapper principal pour redémarrer la navigation
    mainContentRef.value?.focus();
  }
);
</script>

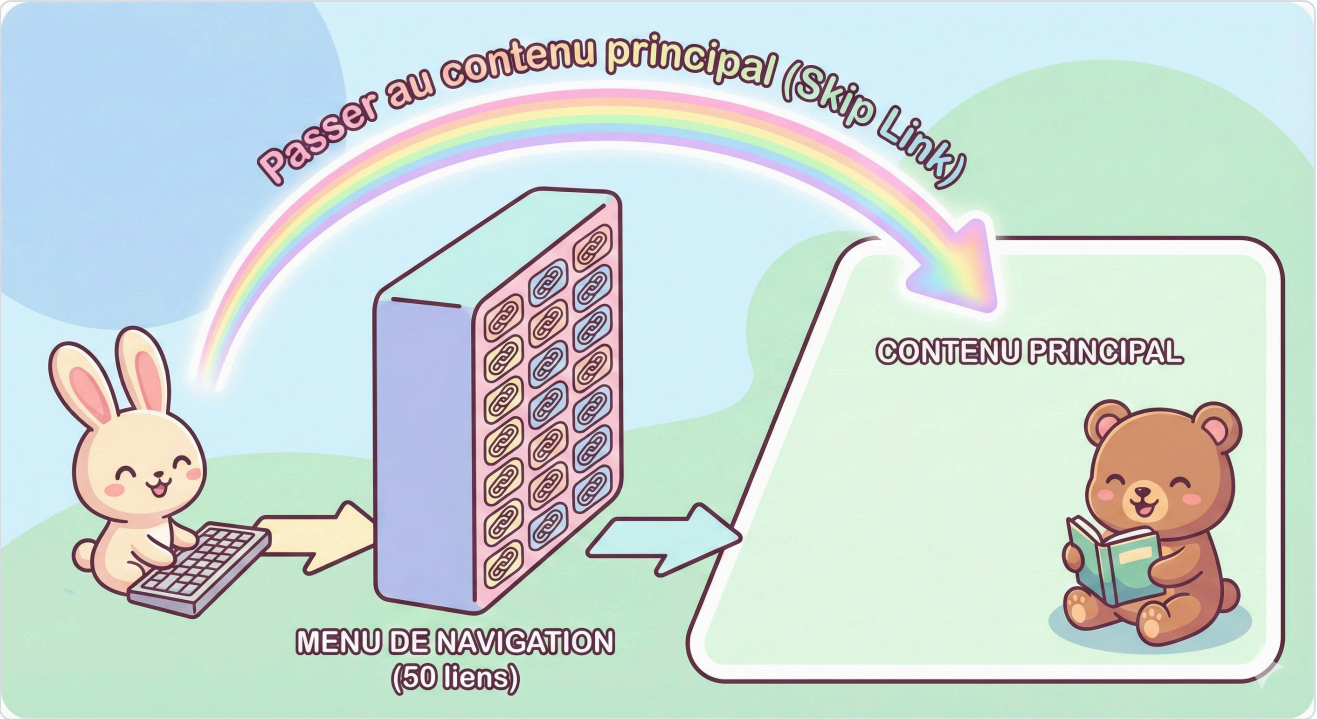
<template>
  <div class="min-h-screen bg-background">
    <a
      href="#main-content"
      class="sr-only focus:not-sr-only focus:absolute focus:z-50 focus:p-4 focus:bg-
background focus:text-foreground focus:outline-none"
    >
      Passer au contenu principal
    </a>

    <header>...</header>

    <main
      id="main-content"
      ref="mainContentRef"
      tabindex="-1"
      class="outline-none"
    >
      <slot />
    </main>
  </div>
</template>
```

Pourquoi ça marche ?

1. **Skip Link :** Au premier `Tab`, le lien apparaît. `Enter` envoie direct au `main`.
2. **Focus Reset :** Quand l'utilisateur change de page via Inertia, le `watch` remet le focus en haut du contenu (`main`). Sans ça, le focus resterait perdu sur l'ancien lien ou le `body`.



4. Landmarks (Repères) 📍

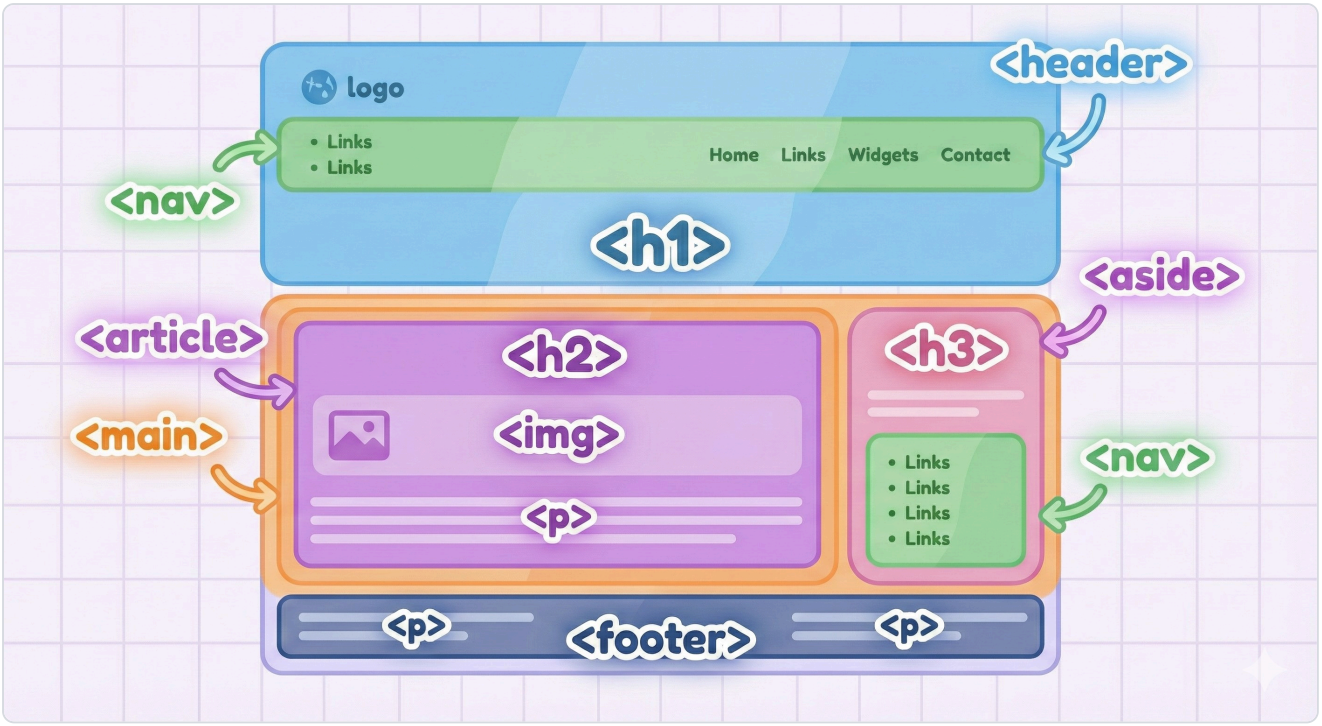
Les lecteurs d'écran permettent de sauter de zone en zone. Utilisez les balises natives :

- `<header>` (En-tête)
- `<nav>` (Navigation)
- `<main>` (Contenu unique de la page)
- `<footer>` (Pied de page)
- `<aside>` (Barre latérale)

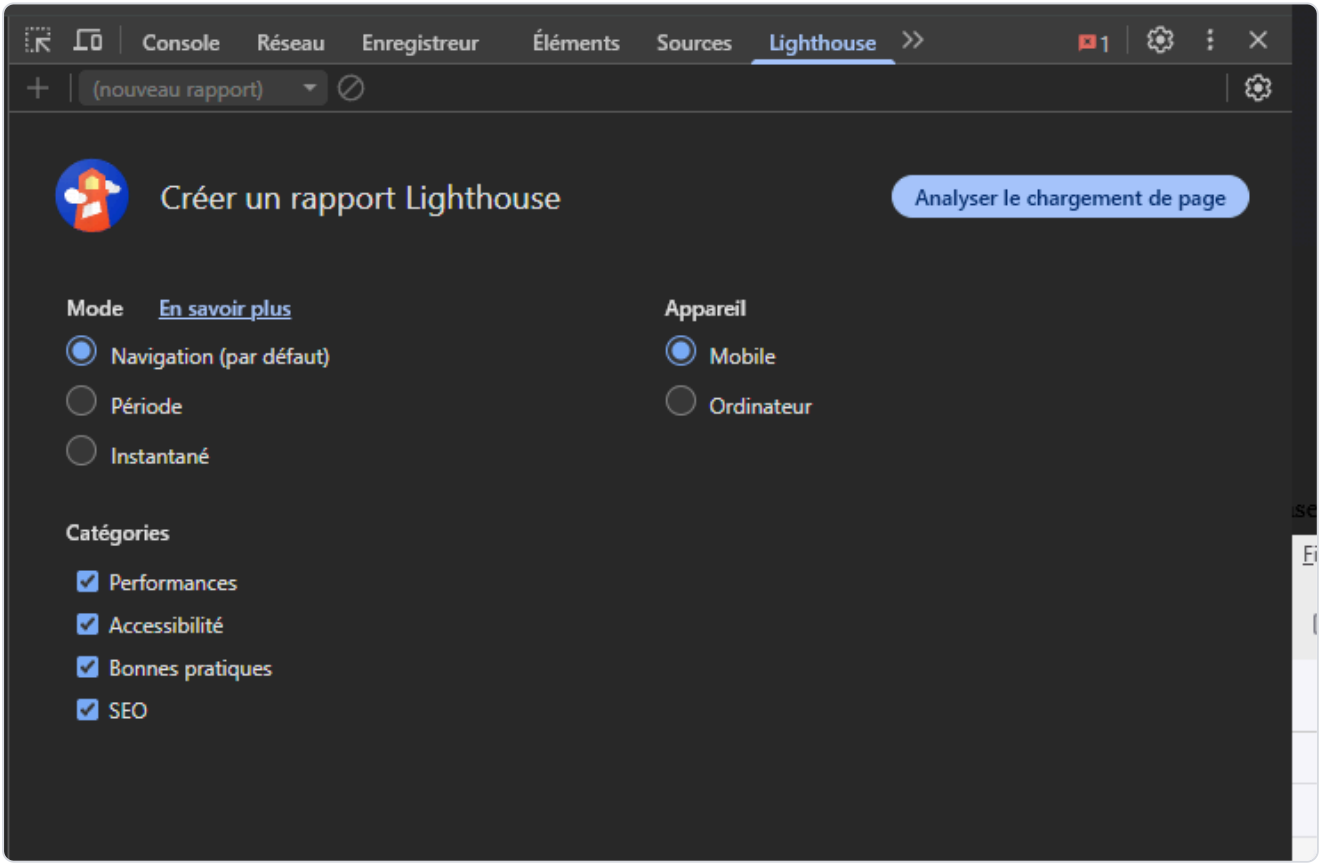
Si vous utilisez des `<div>`, vous devez ajouter `role="main"`, etc. Mais pourquoi se fatiguer ? **Utilisez les balises HTML5 natives.**

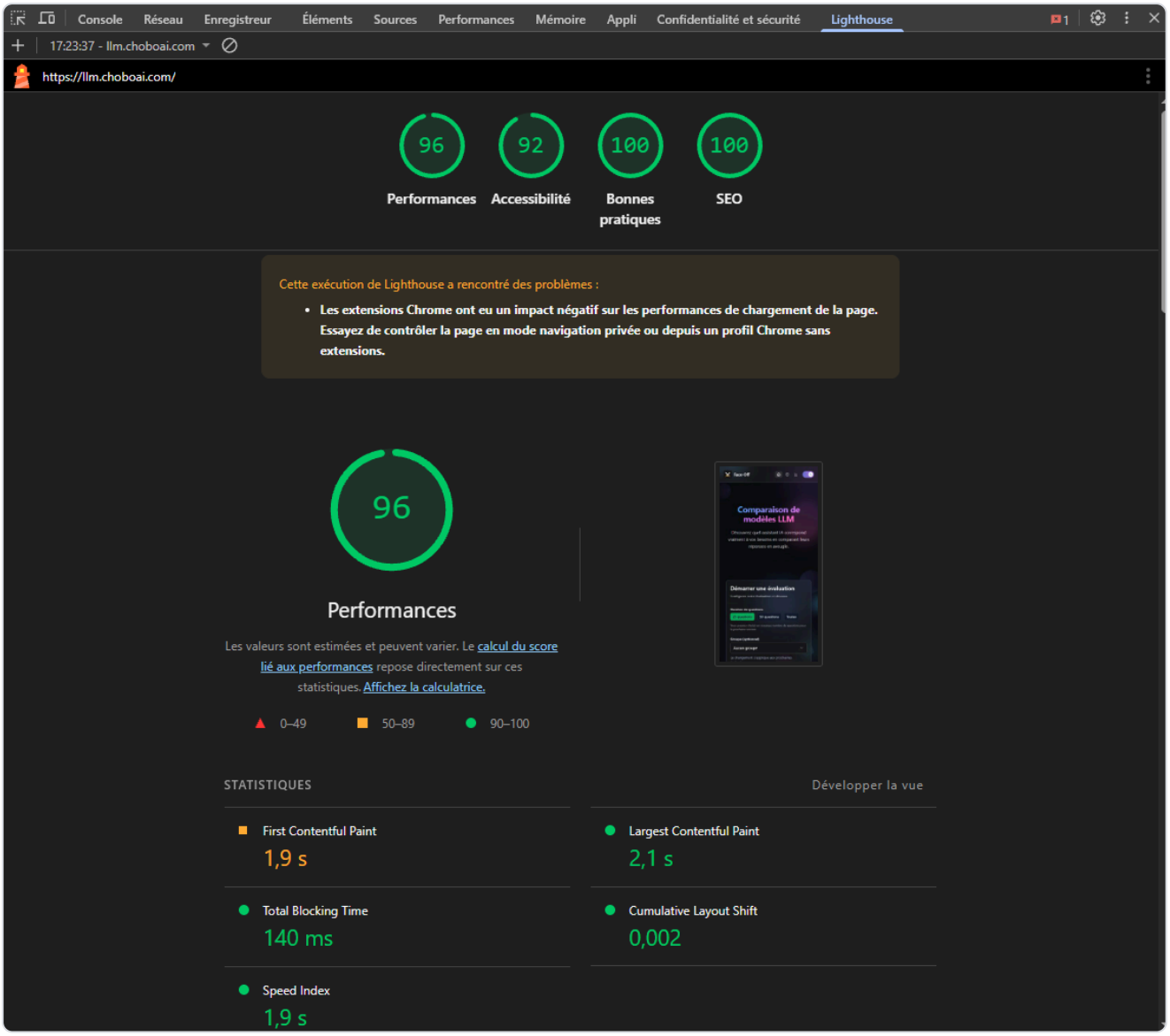
> 🔗 ***Ressource :*** *W3C - ARIA Landmarks Example*

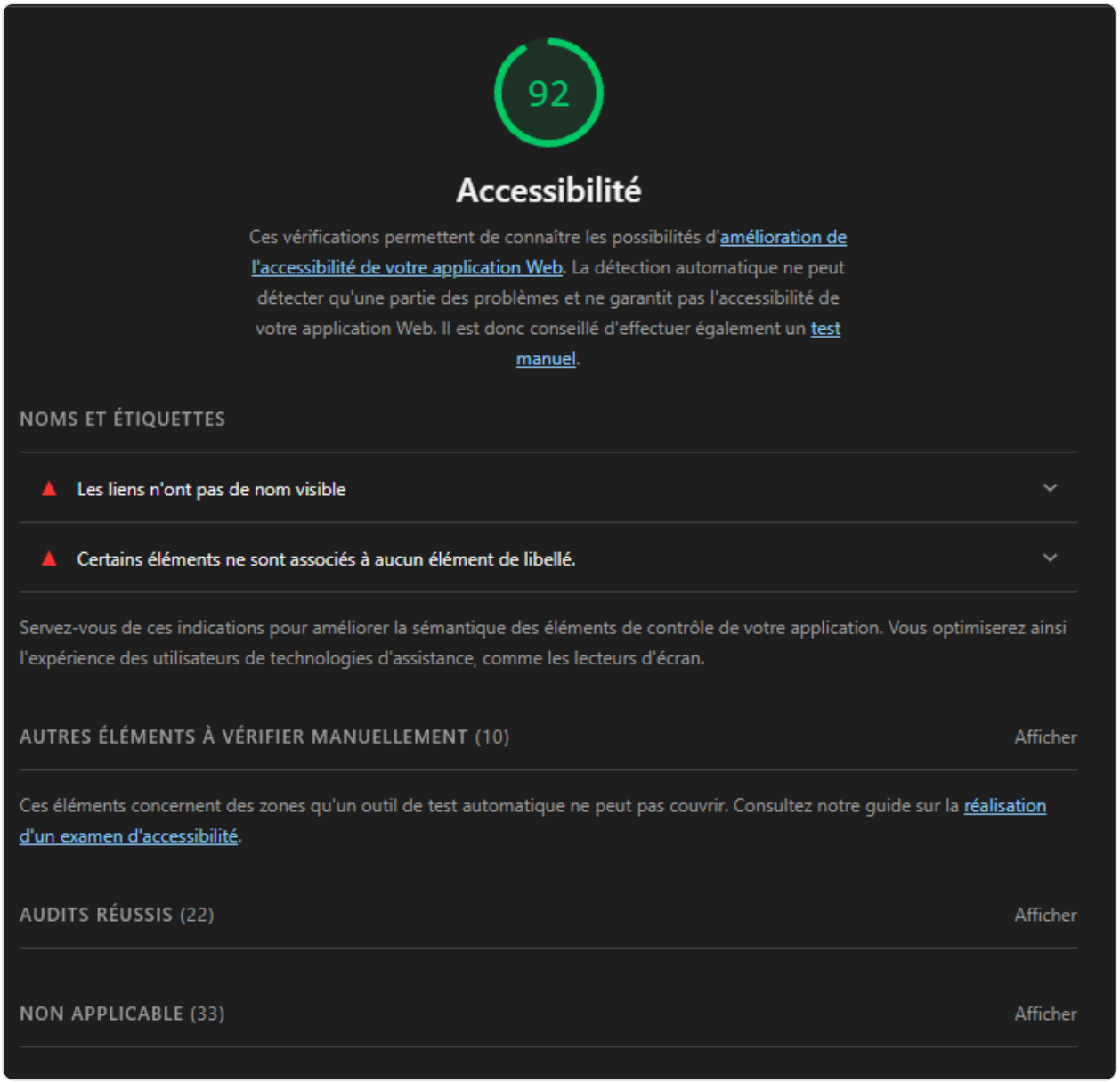
> 💡 ***Astuce Pro :*** *Dans Chrome, ouvrez les DevTools → Onglet "Lighthouse" → Cochez "Accessibility". Il vérifiera automatiquement si vos Landmarks et Titres sont logiques.*



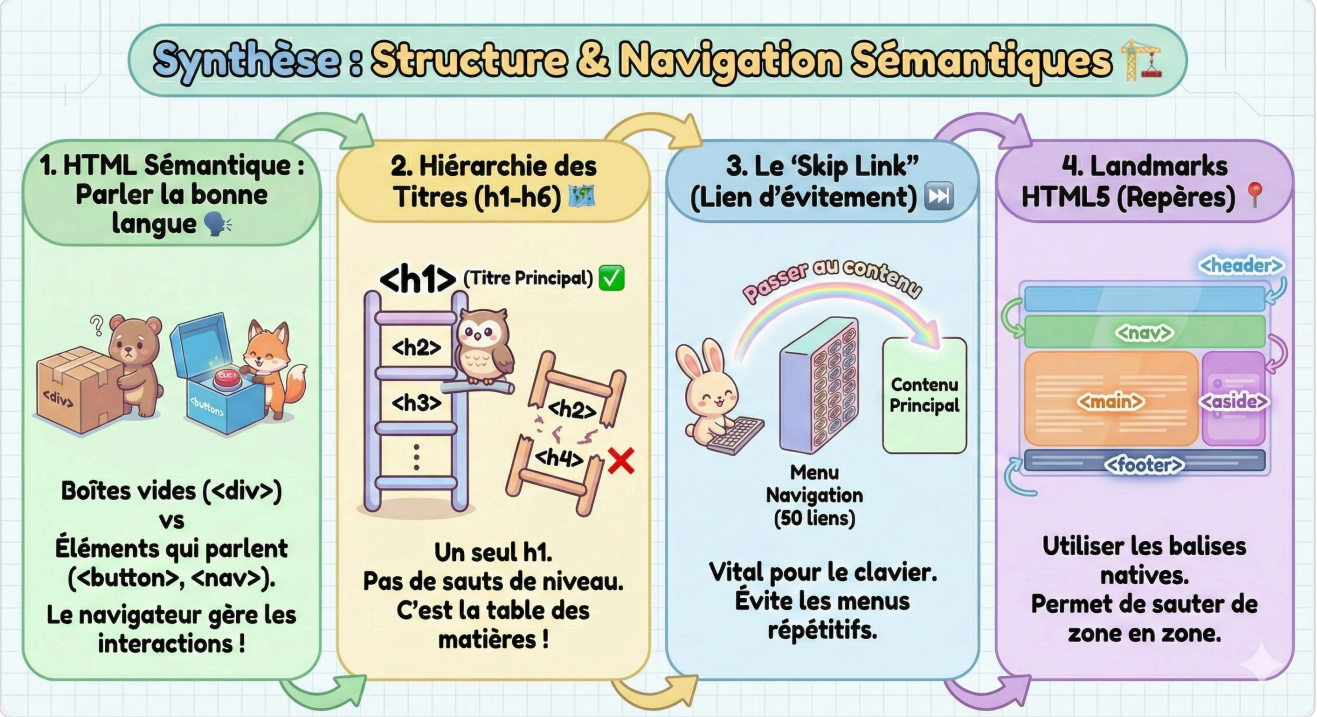
Google Lighthouse




Console
Réseau
Enregistreur
Éléments
Sources
Performances
Mémoire
Appli
Confidentialité et sécurité
Lighthouse
17:23:37 - llm.choboai.com
https://llm.choboai.com/
96
Performances
92
Accessibilité
100
Bonnes pratiques
100
SEO
Cette exécution de Lighthouse a rencontré des problèmes :
Les extensions Chrome ont eu un impact négatif sur les performances de chargement de la page. Essayez de contrôler la page en mode navigation privée ou depuis un profil Chrome sans extensions.
96
Performances
Les valeurs sont estimées et peuvent varier. Le calcul du score lié aux performances repose directement sur ces statistiques. Affichez la calculatrice.
0–49
50–89
90–100
Comparaison de modèles LLM
STATISTIQUES
Développer la vue
First Contentful Paint
1,9 s
Largest Contentful Paint
2,1 s
Total Blocking Time
140 ms
Cumulative Layout Shift
0,002
Speed Index
1,9 s


92
Accessibilité
Ces vérifications permettent de connaître les possibilités d'amélioration de l'accessibilité de votre application Web. La détection automatique ne peut détecter qu'une partie des problèmes et ne garantit pas l'accessibilité de votre application Web. Il est donc conseillé d'effectuer également un test manuel.
NOMS ET ÉTIQUETTES
Les liens n'ont pas de nom visible
Certains éléments ne sont associés à aucun élément de libellé.
Servez-vous de ces indications pour améliorer la sémantique des éléments de contrôle de votre application. Vous optimiserez ainsi l'expérience des utilisateurs de technologies d'assistance, comme les lecteurs d'écran.
AUTRES ÉLÉMENTS À VÉRIFIER MANUELLEMENT (10)
Afficher
Ces éléments concernent des zones qu'un outil de test automatique ne peut pas couvrir. Consultez notre guide sur la réalisation d'un examen d'accessibilité.
AUDITS RÉUSSIS (22)
Afficher
NON APPLICABLE (33)
Afficher

Synthèse



3.3 Chapitre 3 : Formulaires — Ne perdez pas vos utilisateurs en route 📝

On entre dans le dur. Les formulaires.

C'est le moment où vous demandez à l'utilisateur de bosser pour vous (s'inscrire, payer, poster).

Si votre formulaire est une galère à utiliser, ils partiront. Point barre.

Pour un utilisateur de lecteur d'écran ou un utilisateur au clavier, un mauvais formulaire ressemble à un labyrinthe dans le noir. 🔦

Slides

https://gamma.app/docs/Chapitre-3-Formulaires-Accessibles--76qo8zcxto1dw4r

1. La Sainte Trinité du Formulaire 🔱

Pour qu'un champ soit accessible, il lui faut 3 choses :

1. **Une étiquette (Label) :** "C'est quoi ce champ ?"
2. **Un type/état :** "C'est du texte ? C'est obligatoire ? C'est en erreur ?"

3. **Une description (si besoin) :** "Ça doit faire 8 caractères ?" ou "Pourquoi c'est rouge ?"

Le Label : L'attachement vital 🔗

Visuellement, on voit que le texte "Email" est à côté de la case blanche.

Pour un robot, s'ils ne sont pas liés par le code, ils sont des étrangers.

- ❌ **Mauvais :**

  HTML

  ```xml
  <div>Email</div>
  <input type="text" />
  ```

- ✅ **Bon (Avec Shadcn/HTML natif) :**

  HTML

  ```xml
  <Label for="email">Email</Label>
  <Input id="email" name="email" />
  ```

  > *L'attribut `for` du Label doit matcher l'attribut `id` de l'Input. C'est le lien magique. Quand on clique sur le label, ça active l'input.*

2. Gestion des Erreurs : Ne soyez pas juste "Rouge" 🔴

Imaginez : vous êtes daltonien. Le champ devient rouge. Vous ne le voyez pas.

Imaginez : vous êtes aveugle. Un texte d'erreur apparaît sous le champ. Votre lecteur d'écran est focalisé sur le champ, pas sur le texte en dessous. Vous ne savez pas qu'il y a une erreur.

La solution technique : `aria-describedby` et `aria-invalid`.



3. Exemple Concret : Refactoring de "Créer une Maison" 🏠

Reprenons ton code Inertia/Vue pour la création immobilière. On va le transformer en **Forteresse de l'Accessibilité**.

Ce qu'on va ajouter :

- `aria-required="true"` : Pour dire au lecteur "Hé, faut remplir ça".
- `aria-invalid="true"` : Si Inertia renvoie une erreur.
- `aria-describedby` : Pour lier l'input à son message d'erreur.
- `autocomplete` : Pour aider le navigateur à pré-remplir (ça aide énormément les troubles moteurs/cognitifs).

Le Code (Version a11y) ✨

Extrait de code

XML

```
<script setup>
import { Form } from '@inertiajs/vue3'; // Le composant magique
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';

// ... (imports et props inchangés)
</script>

<template>
  <Form :action="store()" #default="{ errors, processing }" class="space-y-6">

    <div class="space-y-2">
      <Label for="title">
        Titre de l'annonce
        <span class="text-destructive" aria-hidden="true">*</span>
      </Label>

      <Input
        id="title"
        type="text"
        name="title"
        placeholder="Ex: Belle maison à Paris"

        aria-required="true"

        :aria-invalid="errors.title ? 'true' : 'false'"

        :aria-describedby="errors.title ? 'title-error' : null"
      />

      <p
        v-if="errors.title"
        id="title-error"
        class="text-sm text-destructive"
        role="alert"
      >
        {{ errors.title }}
      </p>
    </div>

    <div class="space-y-2">
      <Label for="price">Prix (€) <span aria-hidden="true">*</span></Label>
      <Input
        id="price"
        type="number"
        name="price"

        autocomplete="transaction-amount"

        aria-required="true"
        :aria-invalid="errors.price ? 'true' : 'false'"
```

```
        :aria-describedby="errors.price ? 'price-error' : null"
      />
      <p v-if="errors.price" id="price-error" class="text-sm text-destructive"
role="alert">
        {{ errors.price }}
      </p>
    </div>

    <div class="flex gap-4">
      <Button type="submit" :disabled="processing">
        <span v-if="processing" class="mr-2">⏳</span>
        {{ processing ? 'Enregistrement en cours...' : 'Créer la maison' }}
      </Button>
    </div>

  </Form>
</template>
```

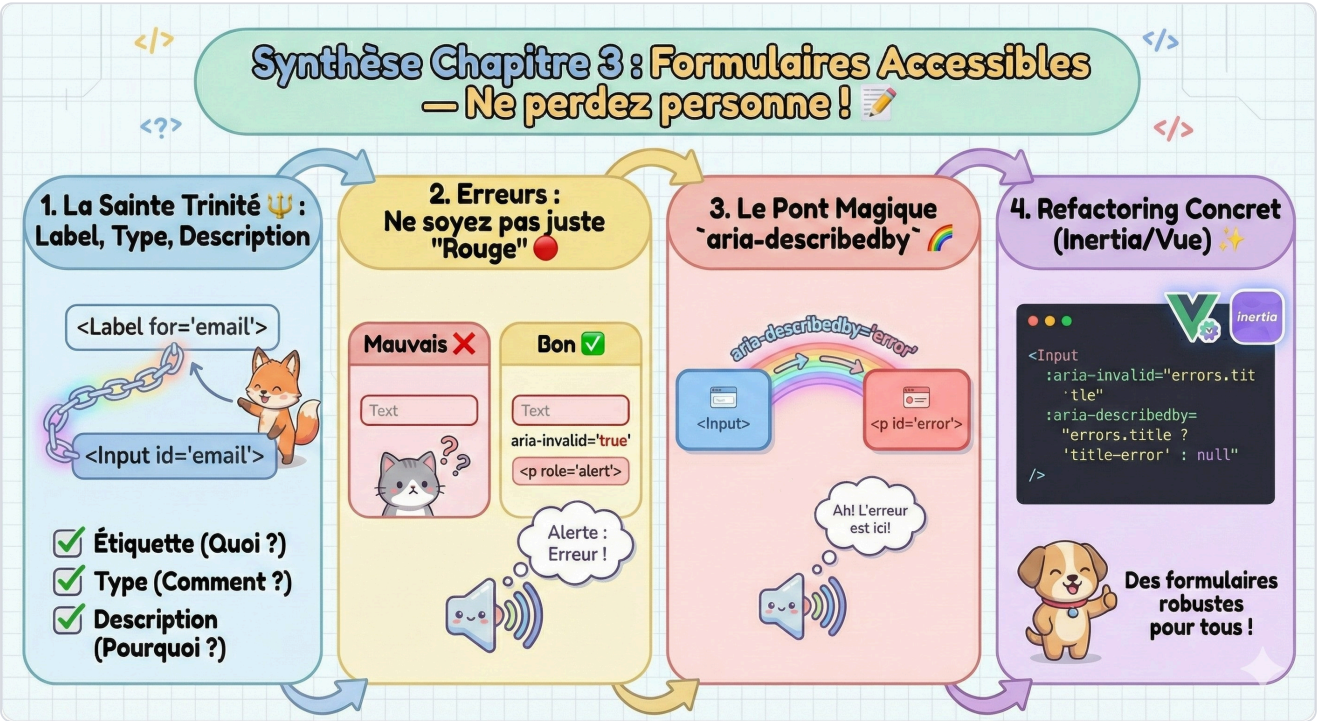
4. Pourquoi c'est mieux ? 🤔

1. `role="alert"` sur l'erreur : Dès que le message apparaît, le lecteur d'écran l'interrompt pour le lire : *"Alerte : Le titre est obligatoire"*.
2. **Connexion ID/For :** Si je clique sur le label "Prix", le curseur va dans le champ Prix (super pour les tremblements ou gros doigts sur mobile).
3. `aria-invalid` : Le lecteur d'écran annonce "Titre, invalide, édition de texte". L'utilisateur sait immédiatement qu'il doit corriger.

> 💡 ***Astuce Shadcn :*** *Les composants Shadcn propagent souvent les attributs (`...props`). Donc si vous mettez `aria-required` sur le composant `<Input>`, il atterrira bien sur la balise HTML `<input>` finale.*



Synthèse



On passe à la suite ? Le prochain chapitre concerne les trucs qui bougent et qui poppent (Modales, Toasts) ! 🍿

3.4 Chapitre 4 : SPA, Modales & Notifications — Maîtriser le chaos 🌪️

Vous avez fait un formulaire accessible ? Bravo. 👏

Maintenant, l'utilisateur clique sur "Sauvegarder".

Une modale s'ouvre ? Une notification apparaît ? La page change sans rechargement ?

Pour un aveugle ou un utilisateur clavier, si vous ne gérez pas ça, c'est comme être téléporté dans une pièce noire sans prévenir.

Slides

https://gamma.app/docs/SPA-Modales-Notifications--fmit19ze7owq5xp

1. Inertia & Le "Silence" des SPA 🤫

Dans une app classique (Blade, PHP pur), quand on clique sur un lien, la page devient blanche, recharge, et le navigateur annonce le nouveau titre.

Dans une SPA (Inertia/Vue), le changement est visuel et instantané. Le navigateur, lui, reste sur la même "page" technique.

Le risque : L'utilisateur clique sur un lien... et rien ne se passe (pour son lecteur d'écran). Il attend toujours.

La Solution : Le composant `<Head>` 🧠

Inertia nous donne un super outil. Utilisez-le sur chaque page.

Il met à jour la balise <title> du document. C'est le signal numéro 1 pour dire "Vous avez changé d'endroit".

Extrait de code

```XML
<script setup>
import { Head } from '@inertiajs/vue3';
</script>

<template>
  <Head title="Créer une maison | Mon Agence" />

  <div class="container">
    <h1>Créer une maison</h1>
    </div>
</template>
```

> 💡 ***Règle :*** *Chaque page vue via Inertia DOIT avoir un* `<Head>`*. C'est votre GPS audio.*

2. Les Modales (Dialogues) : Le Piège à Focus 🪤

Les modales, c'est dangereux.

Si vous codez une modale avec juste une div absolute z-50, vous allez créer un enfer :

1. L'utilisateur ouvre la modale.
2. Il appuie sur `Tab`.
3. Le focus sort de la modale et va sur les liens en arrière-plan (qui sont grisés visuellement !). 😱

Le concept du "Focus Trap" (Hôtel California)

Une bonne modale doit **capturer** le clavier.

- ✅ Tant qu'elle est ouverte, la touche `Tab` tourne en boucle **dedans**.
- ✅ La touche `Echap` (Esc) doit la fermer.
- ✅ Le clic à l'extérieur (Backdrop) doit la fermer.
- ✅ À la fermeture, le focus doit revenir sur le bouton qui a ouvert la modale (sinon l'utilisateur est perdu).

La Solution : Shadcn (Radix UI) à la rescousse 🛡️

N'écrivez jamais votre propre modale (sauf si vous avez 2 semaines devant vous).

Utilisez le composant Dialog de Shadcn. Il gère tout ça nativement.

Extrait de code

```xml
<script setup>
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from '@/components/ui/dialog';
import { Button } from '@/components/ui/button';
</script>

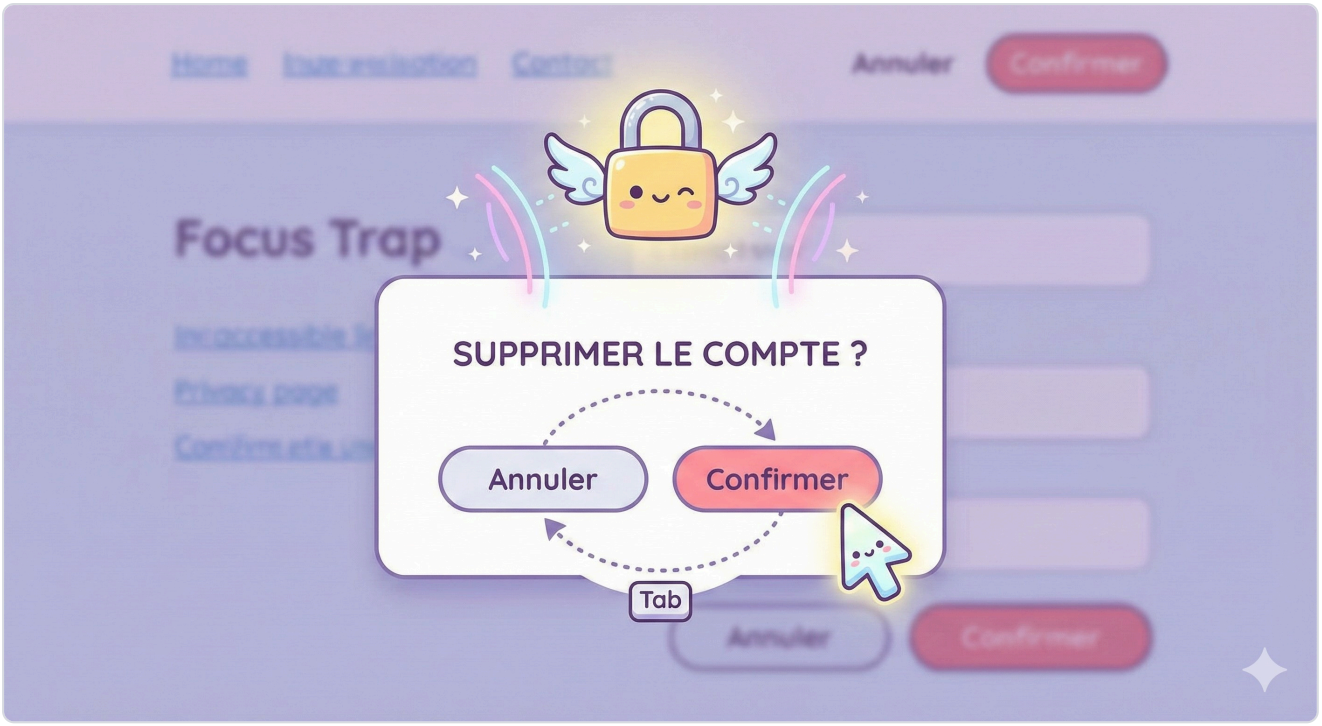
<template>
  <Dialog>
    <DialogTrigger as-child>
      <Button variant="outline">Supprimer le compte</Button>
    </DialogTrigger>

    <DialogContent class="sm:max-w-[425px]">
      <DialogHeader>
        <DialogTitle>Êtes-vous absolument sûr ?</DialogTitle>
      </DialogHeader>

      <div class="py-4">
        Cette action est irréversible.
      </div>

      <div class="flex justify-end">
        <Button variant="destructive">Confirmer la suppression</Button>
      </div>
    </DialogContent>
  </Dialog>
</template>
```

> ***À vérifier :*** *Testez toujours que la touche `Echap` ferme bien votre modale. C'est le réflexe universel.*



3. Notifications (Toasts) : "Hey ! Écoute-moi !" 📢

Vous avez supprimé une maison. Une petite notif verte apparaît en bas à droite : "Succès !".

Visuellement, c'est top.

Pour un aveugle qui est en train de lire le menu en haut de page... cette notif n'existe pas.

Les "Live Regions" (Zones vivantes) 📡

HTML a un attribut magique : aria-live.

Il dit au lecteur d'écran : "Surveille cette zone. Si le texte change, lis-le immédiatement."

Il y a deux niveaux de politesse :

1. `aria-live="polite"` (Poli) : "Attends que l'utilisateur ait fini sa phrase, puis dis-lui." (Utilisé pour les succès, infos).
2. `aria-live="assertive"` (Grossier) : "TAIS-TOI ET ÉCOUTE ÇA !" (Utilisé pour les erreurs critiques, compte à rebours imminent).

Avec Shadcn `useToast`

Shadcn utilise Radix Toast qui gère ça pour nous.

Par défaut, c'est souvent en mode "polite".

Si vous devez afficher une erreur critique via Inertia (les Flash Messages de Laravel), assurez-vous qu'ils sont injectés dans le composant `Toaster` correctement.

Extrait de code

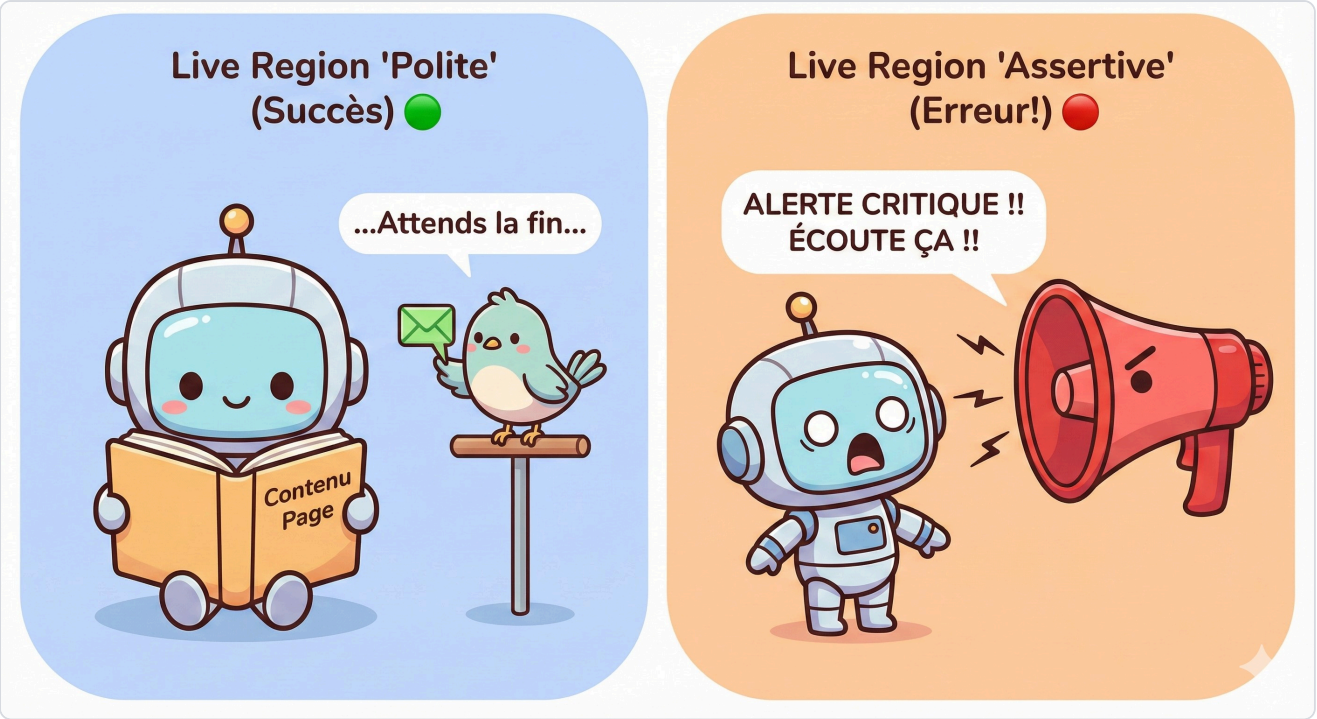
XML

```
<script setup>
import { usePage } from '@inertiajs/vue3';
import { useToast } from '@/components/ui/toast/use-toast';
import { watch } from 'vue';

const page = usePage();
const { toast } = useToast();

// Observer les messages flash de Laravel
watch(() => page.props.flash, (flash) => {
  if (flash.success) {
    toast({
      title: "Succès",
      description: flash.success,
      // Radix gère aria-live="polite" par défaut pour le variant 'default'
    });
  }

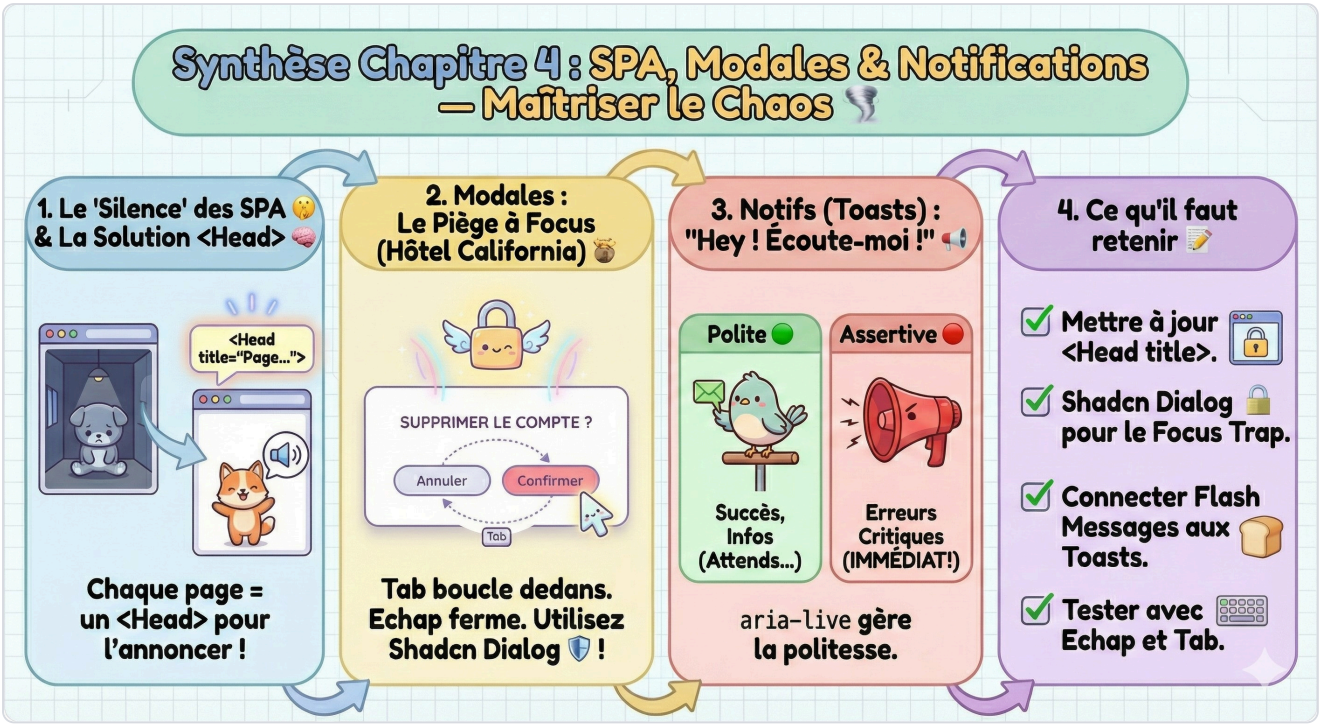
  if (flash.error) {
    toast({
      title: "Erreur",
      description: flash.error,
      variant: "destructive",
      // Le variant destructive devrait passer en assertif ou rester poli selon
l'urgence
    });
  }
}, { deep: true });
</script>
```



4. Ce qu'il faut retenir pour les SPAs 📝

1. **Changement de page :**
   - Mettre à jour le `<Head title="..." />`.
   - Reset du focus (vu au chapitre 2).
2. **Modales :**
   - Utilisez Shadcn/Radix. Ne bricolez pas.
   - Vérifiez le Focus Trap.
3. **Notifications :**
   - Connectez les Flash Messages Laravel au système de Toasts.
   - Vérifiez qu'ils sont lus (les logs ARIA dans Chrome DevTools aident à voir ça).

Synthèse



3.5 Chapitre 5 : Testing & Outils — La minute de vérité 🕵️

Vous avez fini votre feature. Le code compile. Le design claque.

Mais est-ce que votre site est vraiment accessible, ou est-ce que vous vous êtes juste donné bonne conscience ?

Il existe **3 niveaux de test**. Ne sautez pas les étapes.

Slides

https://gamma.app/docs/Chapitre-5-Testing-Outils--e0u0sl4aozb2806

Niveau 1 : Les Robots (Tests Automatisés) 🤖

C'est le filet de sécurité de base. Ça attrape les erreurs bêtes (contraste, balises manquantes, attributs ARIA invalides).

Attention : Un score de 100% ne veut pas dire que votre site est parfait. Les robots ne comprennent pas le contexte. Ils ne savent pas si votre texte alternatif décrit bien l'image ou si c'est juste "image.jpg".

1. Google Lighthouse (Intégré à Chrome)

Pas besoin d'installer quoi que ce soit.

1. Ouvrez les DevTools (`F12`).
2. Onglet **Lighthouse**.
3. Cochez **Accessibility** (et décochez le reste pour aller vite).
4. Cliquez sur **Analyze page load**.

> 💡 ***Note pour Inertia :*** *Lighthouse analyse le DOM après le chargement du JS. C'est parfait pour nos SPAs Vue.js.*

2. WAVE (Extension)

C'est moche, ça injecte des icônes partout sur votre site, mais c'est génial.

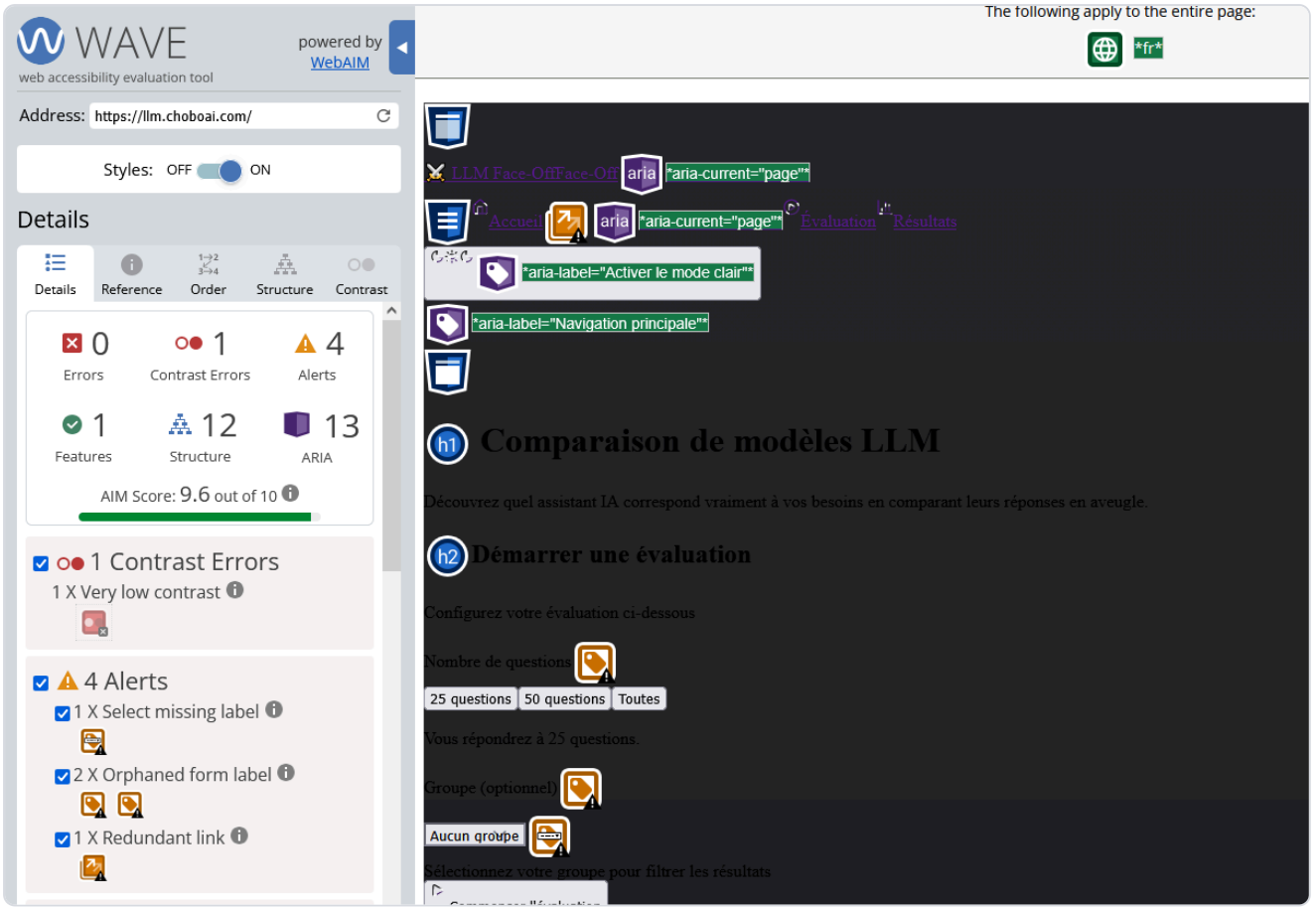
WAVE vous montre visuellement où sont les erreurs.

- 🟥 Rouge : Erreur critique (Contraste, Label manquant).
- 🟨 Jaune : Alerte (Vérifiez si c'est logique).
- 🟩 Vert : Feature d'accessibilité détectée.

> 🔗 ***Télécharger :*** *WAVE Browser Extension*



Wave Report



https://wave.webaim.org/report#/https://llm.choboai.com/

Niveau 2 : Le "No Mouse Challenge" (Manuel) ⌨️

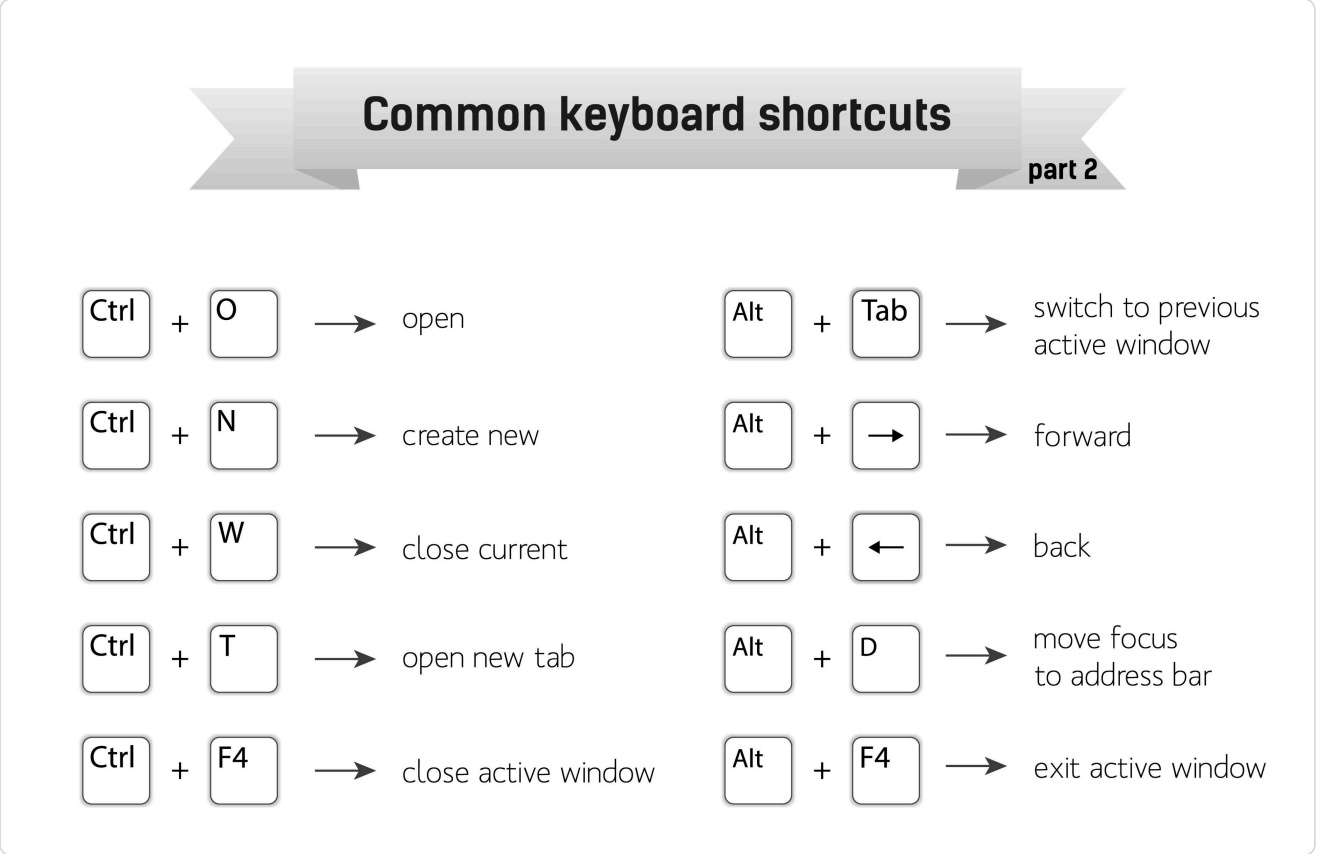
C'est le test le plus révélateur et le plus simple à faire.

La règle : Débranchez votre souris (ou mettez-la loin). Essayez d'utiliser votre feature.

La Checklist de survie :

- [ ] **Focus Visible :** Voyez-vous toujours où vous êtes ? (Le petit encadré bleu ou noir autour des boutons). Si ça disparaît, vous êtes perdu.
- [ ] **Ordre logique :** Le focus passe-t-il du Nom au Prénom, ou saute-t-il aléatoirement ?
- [ ] **Interactions :** Pouvez-vous ouvrir le menu avec `Entrée` ? Pouvez-vous fermer la modale avec `Echap` ?
- [ ] **Formulaires :** Pouvez-vous soumettre le formulaire sans toucher à la souris ?

Si vous bloquez, vos utilisateurs bloqueront. Réparez ça avant d'aller plus loin.



Shutterstock

Niveau 3 : Le Boss Final (Lecteur d'Écran) 🕶️

C'est l'étape qui fait peur, mais qui est nécessaire.

Mettez-vous dans la peau d'un utilisateur aveugle. Vraiment.

L'équipement

- **Mac :** `Command + F5` active **VoiceOver**. C'est natif, c'est excellent.
- **Windows :** Installez **NVDA** (Gratuit et Open Source).

Le Test Rapide

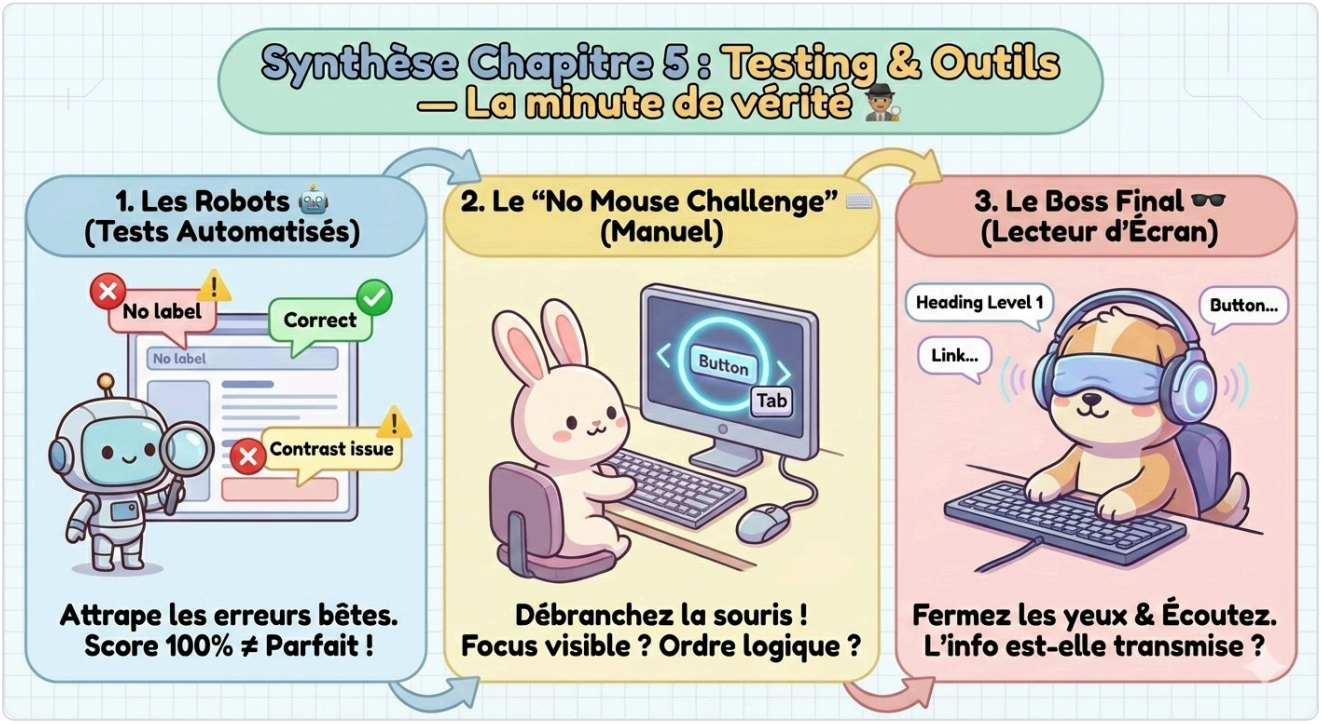
1. Fermez les yeux (ou éteignez l'écran si vous êtes hardcore).
2. Lancez le lecteur.
3. Naviguez sur votre page "Créer une maison".
4. **Écoutez :**
    - Est-ce qu'il annonce "Titre de la page" quand vous arrivez ?

- Est-ce qu'il dit "Titre, édition obligatoire" sur l'input ?
- Est-ce qu'il lit l'erreur quand vous validez vide ?

> ⚠️ ***Attention au choc :*** *La première fois, ça parle vite et c'est le chaos. C'est normal. Ne paniquez pas. Cherchez juste à comprendre si l'information essentielle est transmise.*



Synthèse



3.6 Conclusion du Cours : L'Accessibilité est un Voyage 🌟

Félicitations ! Vous êtes arrivés au bout. 🎉

Vous avez maintenant un starter kit **Laravel + Inertia + Shadcn** qui est non seulement beau et performant, mais aussi **ouvert à tous**.

Récapitulatif :

1. **Structure :** HTML Sémantique, Landmarks, Titres `h1-h6`.
2. **Navigation :** Skip Links, `<Head>` dynamique, Gestion du Focus.
3. **Formulaires :** Labels liés, `aria-describedby` pour les erreurs.
4. **Composants :** Modales Shadcn (Focus Trap), Notifications `aria-live`.
5. **Tests :** Lighthouse + Clavier + Oreilles.

N'essayez pas d'être parfait dès demain.

L'accessibilité n'est pas un état binaire (0 ou 1). C'est une progression.

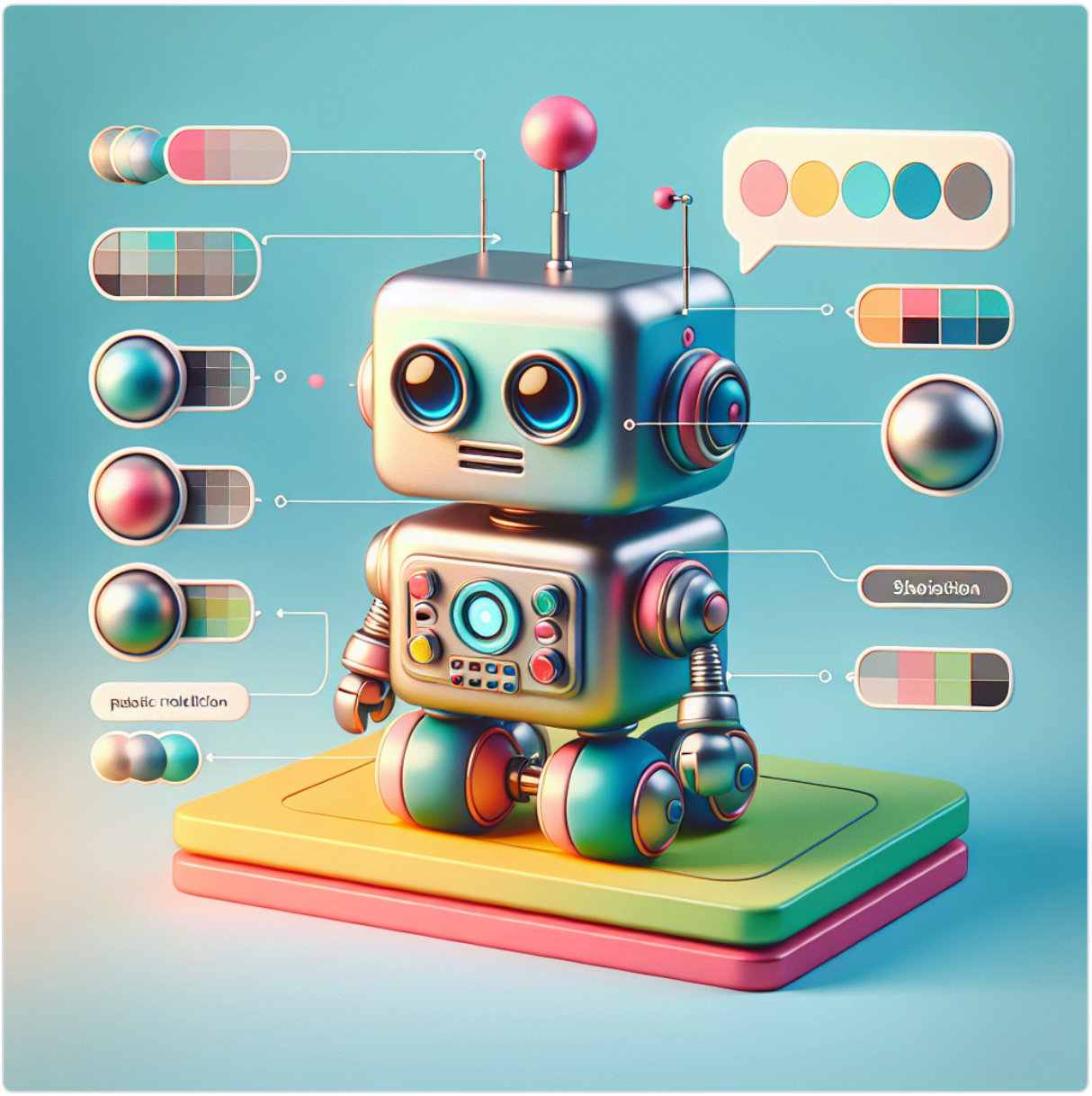
Chaque alt ajouté sur une image, chaque label corrigé, c'est une victoire pour un utilisateur quelque part.

Maintenant, allez coder. Et codez pour tout le monde. 🚀



CHAPITRE 4

Mini ChatGPT



Dans ce chapitre, nous explorerons la création d'une application Mini ChatGPT, en mettant l'accent sur l'intégration des technologies modernes pour développer une interface de chat interactive. Nous commencerons par comprendre comment utiliser une **API** pour alimenter notre application, puis nous plongerons dans l'utilisation de **Markdown** pour enrichir le contenu textuel. Vous aurez l'occasion de mettre en pratique vos compétences à travers plusieurs exercices, tels que la réponse à des questions, la création d'une interface de chat et l'implémentation du streaming. Nous discuterons également de la gestion de la **fenêtre de contexte** et de l'importance des **instructions personnalisées** pour améliorer l'interaction utilisateur. Enfin, nous explorerons divers **outils** pour optimiser le développement de votre Mini ChatGPT. Ce chapitre vise à vous fournir les compétences nécessaires pour concevoir des applications de chat intelligentes et dynamiques.

4.1 Examen : Fabrique ton Clone GPT (et essaye de ne pas pleurer)

(▀̿Ĺ̯▀̿ ̿) Oui, exactement : tu vas reconstruire ChatGPT avec tes petites mains.

1. Objectif général

Développer une application **ChatGPT-like** complète et fonctionnelle qui respecte TOUTES les exigences listées ci-dessous.

Stack technique imposée : Laravel + Starter Kit Vue Inertia shadcn + TailwindCSS.

Si ton projet ressemble à un tutoriel YouTube copié-collé, c'est direction la case échec. 💀

2. Fonctionnalités obligatoires (éliminatoires)

Ces fonctionnalités DOIVENT être présentes et fonctionnelles, sinon c'est ajournement direct :

2.1 Fonctionnalités techniques

1. **Sélecteur de modèles** – Permettre de changer de LLM (Sonnet 4.5, Gemini 3, GPT-5-mini, Grok-4-fast, etc.) facilement dans l'interface
2. **Historique des conversations** – Sauvegarde automatique avec titre généré au premier message
3. **Streaming de la réponse** – Les tokens doivent apparaître en temps réel, pas d'un bloc (WebSockets ou Server-Sent Events)
4. **Instructions personnalisées** – Interface pour configurer le comportement de l'IA (ton, style, contexte)
5. **Vue.js Composition API** – ⚠️ **ATTENTION** : l'utilisation de l'Options API est ÉLIMINATOIRE
6. **Dusk** – Ajoutez des tests à votre application

2.2 Thématisation & Identité de l'application

> *Parce que "ChatGPT mais en Laravel" c'est mass cringe* 🤢

Ton clone GPT doit avoir une **identité propre** et cibler un **public spécifique**. Choisis un thème et construis l'expérience utilisateur autour.

Exemples de thèmes (non exhaustif, sois créatif !) :

Thème	Description
🧠 **PsyBot**	Assistant bien-être mental (avec disclaimers appropriés)
💪 **CoachFit**	Coach sportif personnalisé

Thème	Description
🥗 **NutriGuide**	Conseiller nutrition et alimentation
👵 **SeniorCompanion**	Interface adaptée aux personnes âgées (accessibilité++)
💕 **eGirl/eBoy Companion**	Assistant avec personnalité anime/gaming
📚 **StudyBuddy**	Tuteur scolaire adaptatif
💼 **CareerCoach**	Orientation et conseils professionnels
✍️ **WriterMuse**	Assistant d'écriture créative
🎮 **GameMaster**	Maître de jeu pour JDR/roleplay
🐱 **PetAdvisor**	Conseils pour propriétaires d'animaux
👨‍💻 **DevMentor**	Assistant pair-programming
🌱 **EcoGuide**	Conseiller développement durable
👶 **ParentHelper**	Assistant pour jeunes parents
🎨 **ArtCritic**	Conseiller artistique et créatif
📈 **InvestorBuddy**	Assistant finances personnelles (avec disclaimers)

Exigences :

- **Personnalité cohérente** de l'IA (ton, vocabulaire, utilisation d'émojis adaptés)
- **Design/branding** adapté au thème choisi (couleurs, typographie, iconographie)
- **Instructions système pré-configurées** selon le contexte thématique
- **Expérience utilisateur** pensée pour le public cible

2.3 Page Marketing (Landing Page)

> *Oui, tu vas faire semblant de vendre ton truc comme une vraie startup* 💰

Une **page d'accueil marketing professionnelle** est OBLIGATOIRE. Elle doit contenir :

1. **Hero section** – Accroche percutante + Call-to-Action principal
2. **Présentation des fonctionnalités** – Avec icônes ou illustrations
3. **Section pricing** – Minimum 3 formules fictives (Free / Pro / Enterprise ou équivalent)
4. **Témoignages** – Fictifs mais crédibles et cohérents avec ton thème
5. **FAQ** – Au moins 5 questions fréquentes pertinentes
6. **Footer complet** – Liens légaux, réseaux sociaux fictifs, contact

Bonus : Section "À propos", blog fictif, comparatif avec la concurrence...

2.4 SEO & Accessibilité

> *Un site que personne ne trouve et que personne ne peut utiliser, c'est un échec cosmique ╭∩╮(🔥🔥)╭∩╮*

SEO (obligatoire)

- Balises meta complètes (title, description, og:tags pour le partage social)
- Structure sémantique HTML correcte (h1-h6 hiérarchisés logiquement)
- Fichier `sitemap.xml` généré
- URLs propres et descriptives (pas de `/page?id=123`)
- Performance acceptable (Core Web Vitals)

Accessibilité (obligatoire)

- **Score Lighthouse Accessibility ≥ 85** (screenshot obligatoire dans le rapport)
- Navigation au clavier complète (tous les éléments interactifs accessibles)
- Labels ARIA appropriés sur les composants custom
- Contrastes de couleurs conformes **WCAG 2.1 niveau AA**
- Textes alternatifs sur toutes les images
- Focus visible sur les éléments interactifs

2.5 Conformité Légale

> *Le RGPD te regarde dormir 👁️👁️*

Éléments obligatoires à implémenter :

Mentions légales (page dédiée)

- Identification de l'éditeur (fictif mais complet)
- Hébergeur
- Directeur de publication
- Contact

Politique de confidentialité (page dédiée)

- Types de données collectées
- Finalités du traitement
- Base légale (consentement, intérêt légitime, etc.)
- Durée de conservation
- Droits des utilisateurs (accès, rectification, suppression, portabilité)

- Transferts hors UE (si applicable avec les API LLM)

Cookie Banner

- Bandeau de consentement conforme RGPD
- Consentement **explicite** (pas de cases pré-cochées)
- Option de refus aussi accessible que l'acceptation
- Possibilité de modifier ses choix

Conformité AI Act

- **Transparence** : informer clairement que l'utilisateur interagit avec une IA
- **Disclaimer** visible sur les limites du système (hallucinations possibles, ne remplace pas un professionnel)
- Information sur les modèles utilisés
- Mention que les conversations peuvent être utilisées pour améliorer le service (si applicable)

3. Évaluation

3.1 Grille d'évaluation

Grille d'évaluation (PDF) PDF

Une grille d'évaluation détaillée est fournie séparément. Elle comprend :

- **2 Acquis d'Apprentissage (AA)** qui doivent TOUS être validés pour réussir
- **6 critères de maîtrise** notés sur 100 points au total

Note finale = 50 points (seuil de réussite) + points de maîtrise pondérés sur 50 points

4. Modalités de rendu

4.1 Livrables attendus

1. **Code source** sur GitHub (repository public)
2. **Projet hébergé** – L'application doit être déployée et fonctionnelle en ligne (Heroku, Railway, VPS, ou autre)
3. **Rapport PDF** (voir structure section 8)
    - Format professionnel
    - Screenshots de qualité

- Diagramme UML de la base de données

4.2 Deadline

⏰ **Jeudi 18 décembre 2025, 17h15**

Aucun retard accepté. Le repo GitHub doit être public, le projet hébergé fonctionnel, et le rapport déposé sur Teams AVANT la deadline.

5. Fonctionnalités supplémentaires (pour viser l'excellence)

Ces fonctionnalités peuvent faire grimper ta note au-delà du 15/20 :

- **Dashboard analytique** : statistiques d'utilisation, nombre de tokens consommés, coûts estimés, graphiques de suivi
- **Arbre de conversations** : modification d'un message de la conversation, bouton fork, relancer la génération d'un message
- **Mode Thinking** : Afficher la trace de réflexion de l'assistant
- **Outils LLM intégrés** : calculatrice, recherche web, analyse de documents
- **Gestion avancée du contexte** : résumé automatique, limitation intelligente des tokens
- **Upload et analyse d'images** : intégration avec des modèles vision
- **Génération d'images** : intégration via OpenRouter (utiliser des modèles économiques comme `gpt-image-mini` – moins de 1 cent par image)
- **Voice input/output** : reconnaissance et synthèse vocale
- **Mode sombre/clair** : toggle avec persistance des préférences
- **Multilingue** : interface traduite en plusieurs langues
- **Export de conversations** : PDF, Markdown, JSON
- **Tes propres idées !** : mode canvas, écriture et exécution de code dans le navigateur, agents autonomes, conversation à plusieurs... Surprends-moi ! La créativité sera récompensée.

6. Ressources recommandées

RGPD & Conformité

- CNIL - Guide RGPD
- CNIL - Checklist conformité

AI Act

- EU AI Act - Texte officiel
- Guide pratique AI Act

Accessibilité

- WCAG 2.1 Guidelines
- MDN - Accessibilité
- Lighthouse Chrome DevTools

SEO

- Google Search Central
- web.dev - Core Web Vitals

7. Structure du rapport PDF

```
# [Nom de ton app] - Rapport de Projet

## 1. Introduction
- Contexte et objectifs
- Périmètre du projet
- Technologies utilisées

## 2. Thématisation & Identité
### 2.1 Thème choisi
- Justification du choix
- Public cible identifié
- Analyse des besoins du public cible

### 2.2 Personnalité de l'IA
- Ton et style de communication définis
- Instructions système créées
- Exemples de réponses typiques

### 2.3 Design & Branding
- Charte graphique (couleurs, typographie)
- Choix d'iconographie
- Screenshots de l'interface

## 3. Architecture et Conception
### 3.1 Base de données
- Diagramme UML (OBLIGATOIRE et conforme aux standards)
- Explication des tables et relations
- Contraintes et règles d'intégrité

### 3.2 Architecture logicielle
- Organisation du code Laravel
- Structure des composants Vue.js
- Services et patterns utilisés

## 4. Fonctionnalités développées
### 4.1 Fonctionnalités obligatoires
- Description détaillée de chaque fonctionnalité
- Screenshots annotés
- Défis techniques et solutions

### 4.2 Fonctionnalités bonus (si applicable)
- Liste des extras implémentés
- Valeur ajoutée pour l'utilisateur

## 5. Page Marketing
### 5.1 Stratégie marketing
- Positionnement choisi
- Arguments de vente principaux
- Structure de la landing page
```

```
### 5.2 Contenu créé
- Screenshots des différentes sections
- Pricing fictif et justification
- Témoignages créés

## 6. SEO & Performance
### 6.1 Stratégie SEO
- Balises meta implémentées (liste complète)
- Structure sémantique HTML
- Sitemap et robots.txt
- URLs optimisées

### 6.2 Performance
- Score Lighthouse Performance (screenshot)
- Optimisations réalisées
- Core Web Vitals

## 7. Accessibilité
### 7.1 Standards suivis
- Niveau WCAG visé (AA minimum)
- Outils de test utilisés

### 7.2 Implémentations concrètes
- Navigation clavier (comment ça fonctionne)
- ARIA labels utilisés (exemples de code)
- Gestion des contrastes
- Adaptations spécifiques au thème/public cible
- Score Lighthouse Accessibility (screenshot OBLIGATOIRE)

## 8. Conformité Légale
### 8.1 RGPD
- Données collectées et finalités (tableau)
- Base légale de chaque traitement
- Droits des utilisateurs : comment sont-ils implémentés ?
- Mesures techniques de protection
- Durées de conservation définies

### 8.2 AI Act
- Classification du système (risque limité/autre)
- Obligations de transparence : comment sont-elles respectées ?
- Informations fournies aux utilisateurs
- Disclaimers implémentés

### 8.3 Implémentation technique
- Cookie consent : solution utilisée et fonctionnement
- Pages légales créées (screenshots)
- Gestion du consentement dans le code

## 9. Tests et Qualité
- Stratégie de tests
- Tests Dusk implémentés
- Résultats et couverture
```

```
## 10. Difficultés et Solutions
- Problèmes rencontrés
- Solutions apportées
- Ce qui pourrait être amélioré

## 11. Utilisation des outils IA
- Outils utilisés (ChatGPT, Claude, Copilot, etc.)
- Pour quelles tâches ? (code, rapport, debug, design)
- Comment avez-vous validé/vérifié les outputs ?
- Réflexion critique sur l'utilisation de l'IA

## 12. Conclusion
- Bilan du projet
- Apprentissages clés
- Perspectives d'amélioration
```

Rappel final : Ce n'est pas la quantité de Red Bull consommé qui fera la différence, mais la qualité de ton travail. Montre que tu maîtrises les concepts, pas juste que tu sais copier-coller.

Bonne chance, tu vas en avoir besoin ! 🤖🚀

4.2 Intégration des APIs LLM dans Laravel

1. Introduction : Les APIs LLM en 2025

Bienvenue dans le monde merveilleux des Large Language Models. Vous allez apprendre à faire parler des modèles d'IA depuis votre application Laravel, sans vendre un rein pour payer la facture (enfin, on espère).

Pourquoi intégrer un LLM ?

- Automatiser des tâches de génération de texte
- Créer des chatbots (le fameux "clone ChatGPT" de votre examen 👀)
- Analyser, résumer, transformer du contenu
- Ajouter un effet "wow" à vos projets (ou un effet "flop" si vous vous y prenez mal)

Le paysage actuel

Les providers de LLM (OpenAI, Anthropic, Google, Meta, xAI...) proposent généralement :

- Une **documentation** avec des exemples en Python et TypeScript (parce que PHP c'est pour les boomers, apparemment 🙄)
- Des **packages officiels** pour ces langages

- Un exemple **cURL** pour les autres (c'est nous ça)

C'est ce fameux exemple cURL qui va nous servir de pont vers PHP/Laravel.

2. Comprendre les docs : De cURL à PHP

2.1 C'est quoi cURL ?

cURL (Client URL) est un outil en ligne de commande qui permet d'effectuer des requêtes HTTP. Quand vous voyez un exemple cURL dans une documentation d'API, c'est essentiellement une recette pour envoyer une requête HTTP avec tous les ingrédients nécessaires.

2.2 Anatomie d'une requête API LLM

Prenons l'exemple typique qu'on trouve dans la documentation d'OpenRouter :

```bash
curl https://openrouter.ai/api/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENROUTER_API_KEY" \
  -d '{
    "model": "openai/gpt-5-mini",
    "messages": [
      {
        "role": "user",
        "content": "What is the meaning of life?"
      }
    ]
  }'
```

Décomposons cette requête :

Élément	Signification
`curl https://...`	L'URL de l'endpoint API (où envoyer la requête)
`-H "Content-Type: application/json"`	Header indiquant qu'on envoie du JSON
`-H "Authorization: Bearer ..."`	Header d'authentification avec notre clé API
`-d '{...}'`	Le corps (body) de la requête en JSON

Le body contient deux éléments essentiels :

- **model** : Le modèle à utiliser (ex: `openai/gpt-5-mini`, `anthropic/claude-sonnet-4.5`)
- **messages** : La conversation sous forme de tableau avec des rôles (`system`, `user`, `assistant`)

2.3 Transformer cURL en Laravel Http Client

Laravel fournit un client HTTP élégant via la facade `Http`. Voici comment traduire notre requête cURL :

```php
use Illuminate\Support\Facades\Http;

response=Http::withHeaders(['Content-Type'⇒'application/json',
'Authorization'⇒'Bearer'.config('services.openrouter.apikey'),])
->post('https://openrouter.ai/api/v1/chat/completions',['model'⇒'openai/gpt-5-mini',
'messages'⇒[['role'⇒'user','content'⇒'Whatisthemeaningoflife?']]]);response =
Http::withHeaders([
    'Content-Type' ⇒ 'application/json',
    'Authorization' ⇒ 'Bearer ' . config('services.openrouter.api_key'),
])→post('https://openrouter.ai/api/v1/chat/completions', [
    'model' ⇒ 'openai/gpt-5-mini',
    'messages' ⇒ [
        ['role' ⇒ 'user', 'content' ⇒ 'What is the meaning of life?']
    ]
]);

response=Http::withHeaders(['Content-Type'⇒'application/json',
'Authorization'⇒'Bearer'.config('services.openrouter.apikey'),])
->post('https://openrouter.ai/api/v1/chat/completions',['model'⇒'openai/gpt-5-mini',
'messages'⇒[['role'⇒'user','content'⇒'Whatisthemeaningoflife?']]]);content =
$response→json('choices.0.message.content');
```

C'est tout. Vous venez de faire votre première requête à un LLM en PHP. 🎉

3. OpenRouter : Un routeur pour tous les modèles

3.1 Pourquoi OpenRouter ?

Gérer plusieurs APIs de providers différents (OpenAI, Anthropic, Google...) devient vite un cauchemar : authentifications différentes, formats de requêtes variables, tarifications complexes...

OpenRouter résout ce problème en proposant une API unifiée qui donne accès à plus de 400 modèles de dizaines de providers différents. Vous utilisez une seule clé API, un seul format de requête, et vous pouvez switcher de modèle en changeant juste une string.

3.2 Ce que propose OpenRouter

- **Modèles de langage** : GPT-5, Claude Sonnet 4.5, Gemini 3, Grok, Llama, Mistral, DeepSeek...
- **Génération d'images** *(nouveauté récente !)* :
  - GPT-5 Image Mini (~$8/M tokens, le plus économique)
  - GPT-5 Image (~$40/M tokens)

  - FLUX.2 Pro/Flex (Black Forest Labs)
  - Nano Banana / Gemini 2.5 Flash Image (Google)
- **Vision** : Analyse d'images avec les modèles multimodaux
- **Audio** : Reconnaissance vocale (speech-to-text) et synthèse vocale (text-to-speech)
- **Embeddings** : Pour la recherche sémantique et le RAG

> ⚠️ ***Attention aux coûts*** *: La génération d'images consomme beaucoup plus de tokens que le texte. Privilégiez les modèles économiques comme* `openai/gpt-5-image-mini` *pour vos tests (moins de 1 centime par image).*

3.3 Avantages

- **API compatible OpenAI** : Le format de requête est standardisé
- **Fallback automatique** : Si un provider tombe, OpenRouter peut basculer sur un autre
- **Tarification transparente** : Prix affichés par modèle, facturation consolidée
- **Modèles gratuits** : Plusieurs modèles sont disponibles gratuitement (avec des limites)
- **Infos détaillées** : Usage, coûts, temps de réponse... tout est dans la réponse API

3.4 Créer un compte et obtenir une clé

1. Rendez-vous sur openrouter.ai
2. Créez un compte
3. Générez une clé API dans les settings

> 🎁 ***Bonne nouvelle*** *: Une clé API avec 1$ de crédit OpenRouter vous sera fournie pour ce cours. C'est largement suffisant pour développer et tester votre projet (sauf si vous décidez de générer 10 000 images 4K, évidemment).*

4. Intégration dans Laravel

4.1 Pourquoi ne pas utiliser un package existant ?

Il existe des packages PHP comme `openai-php/laravel` qui wrappent l'API OpenAI. Cependant, pour ce cours, **nous allons construire notre propre client** pour plusieurs raisons :

1. **Accès complet aux données** : Les packages masquent souvent des informations utiles (usage, coûts, thinking des modèles...)
2. **Compréhension** : Vous saurez exactement ce qui se passe sous le capot
3. **Flexibilité** : Facile d'ajouter des features spécifiques à OpenRouter
4. **Contrôle des erreurs** : Messages d'erreur clairs et personnalisables
5. **Pédagogie** : C'est l'occasion d'apprendre à structurer du code proprement

4.2 Configuration

Dans votre `.env` et `.env.example` :

```
OPENROUTER_API_KEY=sk-or-v1-votre-clé-ici
OPENROUTER_BASE_URL=https://openrouter.ai/api/v1
```

Dans `config/services.php`, ajoutez :

```php
'openrouter' => [
    'api_key' => env('OPENROUTER_API_KEY'),
    'base_url' => env('OPENROUTER_BASE_URL', 'https://openrouter.ai/api/v1'),
],
```

4.3 Architecture recommandée : Client + Services

Pour un code propre et maintenable, on sépare les responsabilités :

```
app/Services/
└── OpenRouter/
    └── Client.php          # Gère les appels HTTP à l'API

├── AskService.php          # Questions simples (exercice 1)
├── ChatService.php         # Conversations avec historique
├── ImageService.php        # Génération d'images
└── ...
```

Le Client s'occupe uniquement de :

- Configurer les headers d'authentification
- Envoyer les requêtes HTTP
- Gérer les erreurs de base

Les Services s'occupent de :

- La logique métier
- Formater les données
- Utiliser le Client pour communiquer avec l'API

4.4 Un Client basique

Voici un exemple de Client simple pour démarrer :

PHP

```php
<?php

declare(strict_types=1);

namespace App\Services\OpenRouter;

use Illuminate\Http\Client\PendingRequest;
use Illuminate\Http\Client\Response;
use Illuminate\Support\Facades\Http;

class Client
{
    private string apiKey;privatestringapiKey;
    private string apiKey;privatestringbaseUrl;

    public function __construct()
    {
        this->apiKey=config('services.openrouter.apikey');this→apiKey =
config('services.openrouter.api_key');
        this->apiKey=config('services.openrouter.apikey');this→baseUrl =
rtrim(config('services.openrouter.base_url'), '/');
    }

    /**
     * Effectue une requête GET.
     */
    public function get(string endpoint): Response
    {
        return $this→request()→get($this→baseUrl . $endpoint);
    }

    /**
     * Effectue une requête POST.
     */
    public function post(string endpoint, array $data = []): Response
    {
        return $this→request()→post($this→baseUrl . $endpoint, $data);
    }

    /**
     * Configure la requête HTTP avec les headers nécessaires.
     */
    private function request(): PendingRequest
    {
        return Http::withHeaders([
            'Authorization' ⇒ 'Bearer ' . $this→apiKey,
            'Content-Type' ⇒ 'application/json',
            'HTTP-Referer' ⇒ config('app.url'),
            'X-Title' ⇒ config('app.name'),
```

```
        ])->timeout(120);
    }
}
```

4.5 Utilisation du Client dans un Service

```php
<?php

namespace App\Services;

use App\Services\OpenRouter\Client;

class ChatService
{
    public function __construct(private Client
client)publicfunctionsendMessage(arrayclient) {}

    public function sendMessage(array client)publicfunctionsendMessage(arraymessages,
string $model): string
    {
        $response = $this->client->post('/chat/completions', [
            'model' => $model,
            'messages' => $messages,
            'temperature' => 1.0,
        ]);

        if ($response->failed()) {
            $error = $response->json('error.message', 'Erreur inconnue');
            throw new \RuntimeException("Erreur API: {$error}");
        }

        return $response->json('choices.0.message.content', '');
    }

    public function getModels(): array
    {
        $response = $this->client->get('/models');

        return $response->json('data', []);
    }
}
```

4.6 Aller plus loin : DTOs et architecture complète

Pour un projet plus conséquent, vous pouvez structurer davantage avec des **Data Transfer Objects (DTOs)** :

```bash
app/Services/OpenRouter/
├── Client.php
├── Data/
│   ├── Message.php           # Représente un message
│   ├── ChatOptions.php       # Options de la requête
│   └── Responses/
│       ├── ChatResponse.php  # Réponse complète
│       ├── Usage.php         # Tokens utilisés, coûts
│       └── ...
```

Cette approche permet de :

- Typer fortement les données
- Accéder facilement aux infos de pricing et d'usage
- Récupérer le "thinking" des modèles qui le supportent (Claude, o1...)

Nous verrons cette architecture avancée dans les chapitres suivants.

5. Bonnes pratiques

5.1 Sécurité des clés API

Votre clé API est comme votre carte bancaire. Ne la laissez pas traîner.

```php
// ❌ JAMAIS ÇA
apiKey='sk-or-v1-abc123...';//⚠️E´VITERdanslecodeapplicatifapiKey = 'sk-or-v1-abc123...';

// ⚠️ ÉVITER dans le code applicatif
apiKey='sk-or-v1-abc123...';//⚠️E´VITERdanslecodeapplicatifapiKey = env('OPENROUTER_API_KEY');

// ✅ TOUJOURS ÇA
$apiKey = config('services.openrouter.api_key');
```

> 💡 ***Pourquoi*** `config()` *plutôt que* `env()` *? Le helper* `env()` *ne fonctionne pas quand le cache de config est activé (*`php artisan config:cache`*). En production, utilisez toujours* `config()` *qui lit depuis les fichiers de configuration.*

Règles d'or :

- Ne **jamais** commiter une clé API dans Git (ajoutez `.env` au `.gitignore`)
- Utiliser des clés différentes pour dev/staging/production
- Surveiller votre consommation sur le dashboard OpenRouter

- Révoquer immédiatement une clé compromise

5.2 Gestion des erreurs

Les APIs LLM peuvent échouer pour diverses raisons : rate limiting, quota dépassé, modèle indisponible...

```php
use Illuminate\Http\Client\RequestException;

try {
    response=Http::withHeaders([...])
->timeout(60)//LesLLMspeuvente^trelents->retry(3,1000)//3tentatives,1sentrechaque->pos
t(response = Http::withHeaders([...])
        →timeout(60) // Les LLMs peuvent être lents
        →retry(3, 1000) // 3 tentatives, 1s entre chaque
        →post(response=Http::withHeaders([...])
->timeout(60)//LesLLMspeuvente^trelents->retry(3,1000)//3tentatives,1sentrechaque->pos
t(url, data);if(data);

    if (data);if(response→failed()) {
        error=error = error=response→json('error.message', 'Erreur inconnue');
        throw new \RuntimeException("Erreur API: {error}");
    }

    return response→json('choices.0.message.content');

} catch (\Exception $e) {
    Log::error('Erreur LLM', ['message' ⇒ $e→getMessage()]);
    throw $e;
}
```

5.3 Logging

Gardez une trace de vos requêtes pour débugger et analyser l'usage :

```php
Log::info('LLM Request', [
    'model' ⇒ model,’prompttokens’⇒model,
    'prompt_tokens' ⇒ model,’prompttokens’⇒response→json('usage.prompt_tokens'),
    'completion_tokens' ⇒ response->json(’usage.completiontokens’),
’totaltokens’⇒response→json('usage.completion_tokens'),
    'total_tokens' ⇒ response->json(’usage.completiontokens’),’totalt
okens’⇒response→json('usage.total_tokens'),
    'cost' ⇒ $response→json('usage.cost'), // Spécifique OpenRouter
]);
```

5.4 Paramètres utiles

Temperature

La température contrôle le niveau de "créativité" ou d'aléatoire dans les réponses :

Valeur	Comportement
`1`	Valeur recommandée par défaut pour les assistants conversationnels
`0.5 - 0.8`	Plus sérieux, moins de variabilité, bon pour du contenu factuel
`0`	Déterministe : mêmes inputs = mêmes outputs (utile pour les tests)

```php
// Assistant créatif/conversationnel
'temperature' => 1,

// Rédaction sérieuse, documentation
'temperature' => 0.7,

// Réponses reproductibles (tests, pipelines)
'temperature' => 0,
```

Max Tokens

Ce paramètre limite le nombre de tokens en sortie. **En général, ne le limitez pas trop** — laissez le modèle répondre ce dont il a besoin. La limite technique du modèle bloquera naturellement.

Cas où limiter est pertinent :

- Génération de titres courts (ex: `max_tokens: 50`)
- Résumés contraints
- Réponses très courtes attendues

```php
// Génération de titre de conversation
response=response = response=client→post('/chat/completions', [
    'model' ⇒ 'openai/gpt-5-mini',
    'messages' ⇒ messages,
’maxtokens’⇒50,//Onveutjustequelquesmots]);//Conversationnormale:pasdelimiteartificiellemessages,
    'max_tokens' ⇒ 50, // On veut juste quelques mots
]);

// Conversation normale : pas de limite artificielle
messages,’maxt
okens’⇒50,//Onveutjustequelquesmots]);//Conversationnormale:pasdelimiteartificiellereponse = client->post(’/chat/completions’,[’model’⇒’openai/gpt-5-mini’,
’messages’⇒client→post('/chat/completions', [
    'model' ⇒ 'openai/gpt-5-mini',
    'messages' ⇒ client->post(’/chat/completions’,[’model’⇒’openai/gpt-5-mini’,
’messages’⇒messages,
    'temperature' ⇒ 1,
    // max_tokens non spécifié = le modèle répond librement
]);
```

Autres paramètres

Paramètre	Description
`top_p`	Alternative à temperature (nucleus sampling), entre 0 et 1
`stream`	Activer le streaming des tokens (voir chapitre dédié)
`stop`	Séquences qui arrêtent la génération

6. Ressources utiles

- Documentation OpenRouter
- Liste des modèles OpenRouter
- Documentation Laravel HTTP Client

Conclusion

Vous avez maintenant les bases pour intégrer des LLMs dans votre application Laravel avec le client HTTP natif. Cette approche vous donne un contrôle total sur les requêtes et les réponses, ce qui sera essentiel pour les fonctionnalités avancées comme le streaming ou l'accès aux données d'usage.

Dans les chapitres suivants, nous verrons comment implémenter le streaming pour afficher les réponses en temps réel, et comment structurer proprement votre code avec une architecture complète (DTOs, services spécialisés...).

Maintenant, au boulot ! 🚀

4.3 Markdown

Le Markdown est un langage de balisage léger conçu pour être facile à lire et à écrire. Son but principal est de permettre la rédaction de documents formatés de manière simple, sans les lourdeurs des éditeurs de texte traditionnels. Ce chapitre vous guidera à travers les bases du Markdown, ses utilisations, et comment il s'intègre dans les interactions avec les Modèles de Langage à Grande Échelle (LLMs).

Introduction au Markdown

Qu'est-ce que le Markdown ?

Le Markdown est un langage de balisage utilisé pour formater du texte sur le web. Créé en 2004 par John Gruber avec l'aide d'Aaron Swartz, il a été conçu pour être facilement convertible en HTML. Voici un exemple simple :

```
# Ceci est un titre de niveau 1
```

Ceci est un titre de niveau 1

À quoi sert le Markdown ?

Le Markdown est largement utilisé pour la rédaction de documentation, de fichiers README sur les plateformes de gestion de version comme GitHub, et pour la création de contenu sur de nombreuses plateformes en ligne. Il est apprécié pour sa simplicité et sa facilité de conversion en HTML.

Mises en Forme de Base en Markdown

Texte

- **Gras** : Pour mettre du texte en gras, entourez-le de deux astérisques ou de deux traits de soulignement.

```
**texte en gras**
```

texte en gras

- *Italique* : Pour mettre du texte en italique, entourez-le d'un astérisque ou d'un trait de soulignement.

```
*texte en italique*
```

texte en italique

- ~~Barré~~ : Pour barrer du texte, entourez-le de deux tildes.

```
~~texte barré~~
```

~~texte barré~~

Titres et sous-titres

Pour créer un titre ou un sous-titre, utilisez le symbole `#` suivi d'un espace. Le nombre de `#` détermine le niveau du titre.

```
# Titre de niveau 1
## Titre de niveau 2
### Titre de niveau 3
```

Titre de niveau 1

Titre de niveau 2

Titre de niveau 3

Listes

- Listes ordonnées :

```
1. Premier élément
2. Deuxième élément
```

1. Premier élément
2. Deuxième élément

- Listes non ordonnées :

```
- Un élément
- Un autre élément
```

- Un élément
- Un autre élément

Liens et Images

- Liens : Pour insérer un lien, utilisez la syntaxe `[texte du lien](URL)`.

```
[Lien vers google](https://google.com)
```

Google

- Images: Pour insérer une image, utilisez la syntaxe `![description de l'image](URL de l'image)`

```
[Chat magicien](/storage/media/84/sSYfgRXCH4qI1WyI7CCR..png)
```



Blocs de Code et Citations

- Blocs de Code : Pour afficher un bloc de code, entourez-le de trois backticks ```.

Exemple de code

```Exemple de code```

Exemple de code

```CSS
Exemple de code
```

- Citations : Utilisez le symbole `>` pour créer une citation.
```

```
> Ceci est une citation.
```

> *Ceci est une citation.*

Tableaux

Les tableaux en Markdown sont créés en utilisant des barres verticales `|` et des tirets `-`. Voici un exemple :

```
| En-tête 1 | En-tête 2 |
|-----------|-----------|
| Cellule 1 | Cellule 2 |
| Cellule 3 | Cellule 4 |
```

En-tête 1	En-tête 2
Cellule 1	Cellule 2
Cellule 3	Cellule 4

Markdown et les Modèles de Langage à Grande Échelle (LLMs)

Le Markdown dans les données d'entraînement des LLMs

Les LLMs, comme GPT, sont entraînés sur une vaste gamme de textes, y compris des documents formatés en Markdown. Cela leur permet de comprendre et de générer du texte formaté en Markdown, facilitant ainsi une interaction plus riche et structurée avec les utilisateurs.

Utilisation du Markdown avec les LLMs

Le Markdown permet aux utilisateurs de donner des instructions claires et structurées aux LLMs et de recevoir des réponses bien organisées. Cette capacité d'interprétation et de génération de Markdown par les LLMs enrichit l'expérience utilisateur en permettant une communication plus précise et visuellement organisée.

Vous pouvez par exemple utiliser les block de code pour passer des textes au LLM. Cela permet de distinguer clairement le texte de vos instructions.

Le Markdown sert de pont entre la simplicité de la rédaction de texte et la complexité des formats de document plus élaborés, rendant l'information accessible et facilement manipulable par les humains comme par les machines.

4.4 Exercice 1 : L'exaltante intégration d'un LLM

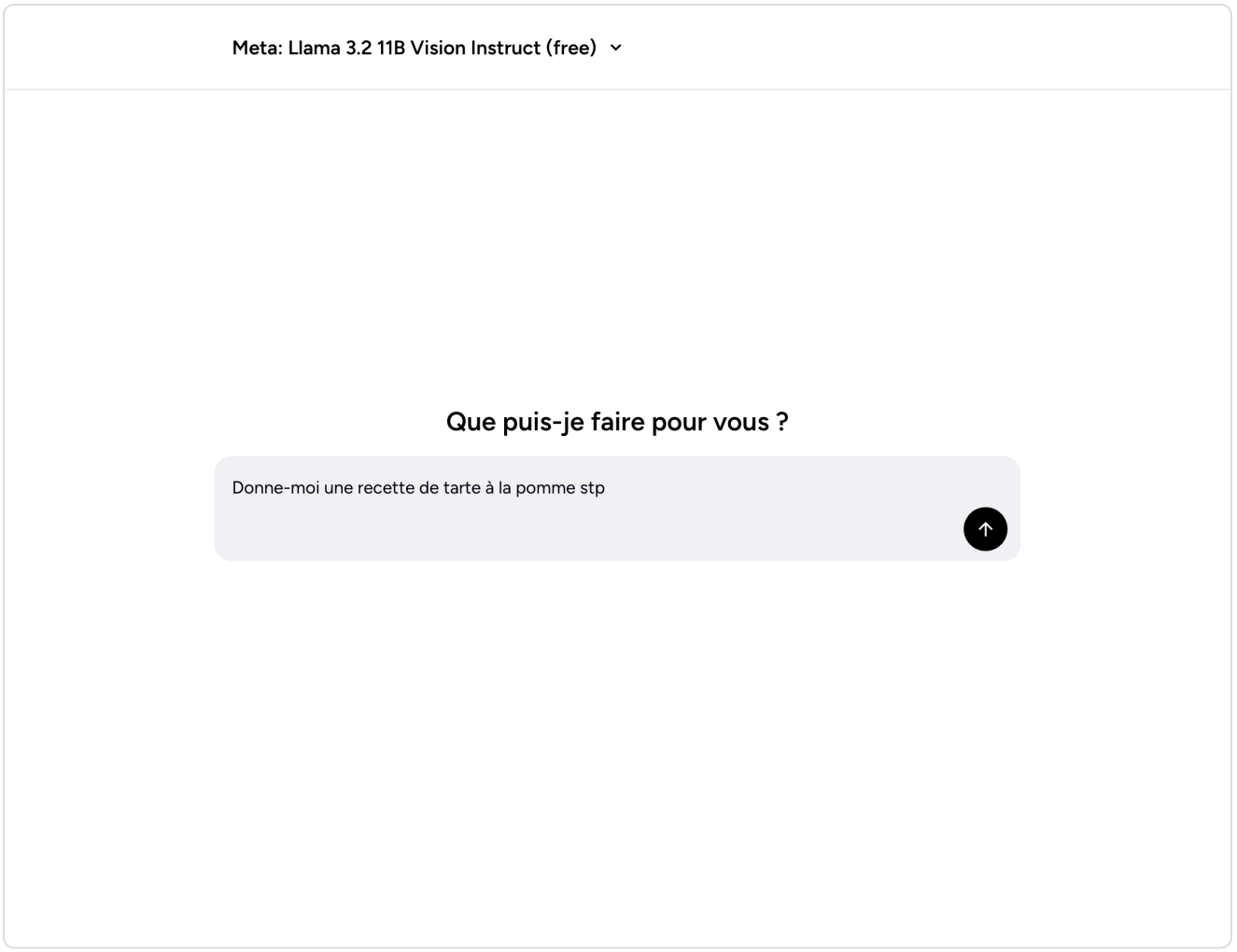
Bienvenue dans ce chapitre, ô brillants étudiants... ou pas. Le programme du jour ? Un exercice ridiculement simple : créer une page permettant de poser UNE et UNE SEULE question à la fois à un modèle d'IA, sans stocker quoi que ce soit en base de données. Oui, vous avez bien lu : même un poulpe enrhumé y arriverait. 😏

Plan de bataille

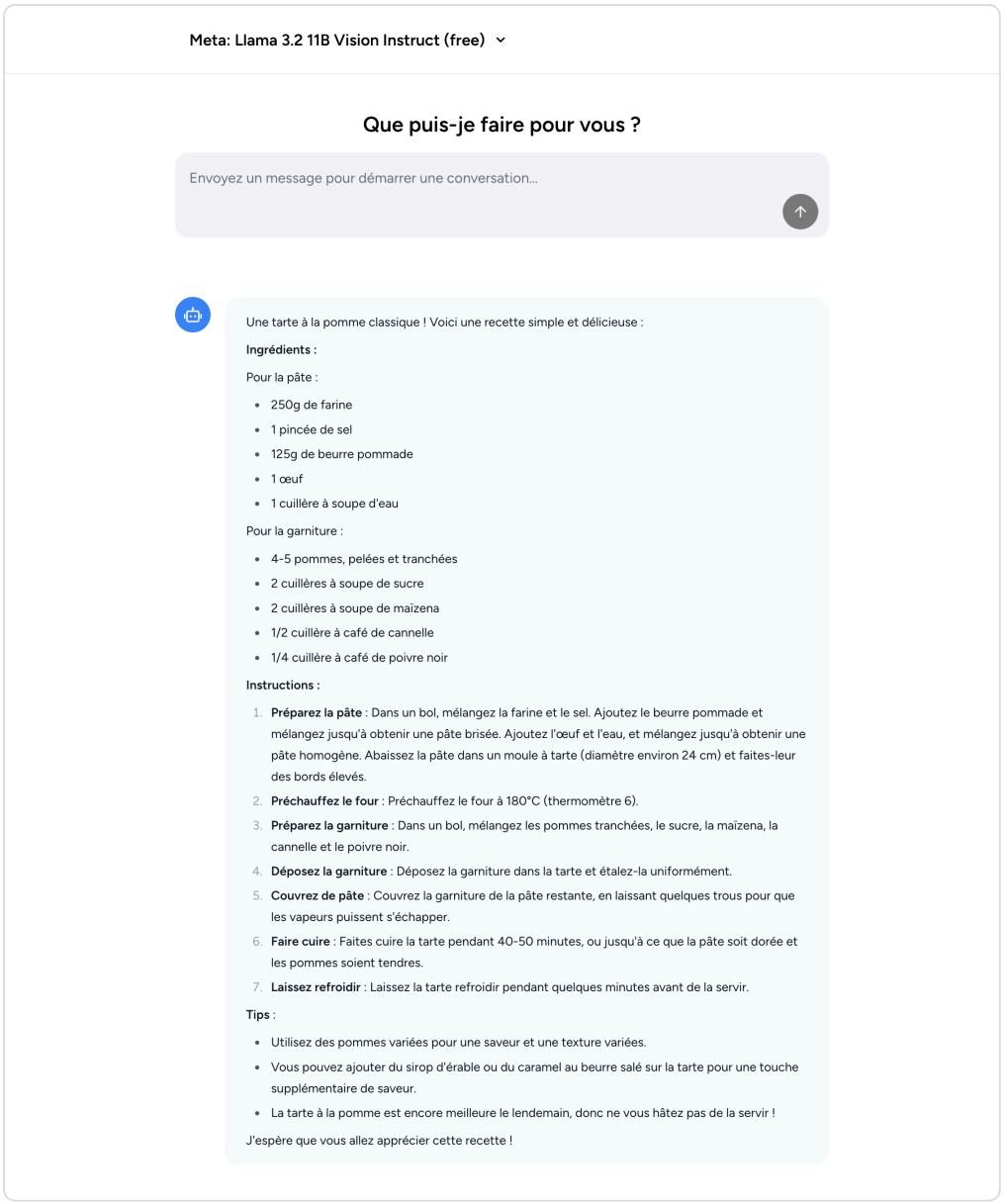
1. **Configurer la connexion à OpenRouter** avec les variables d'environnement.
2. **Créer la classe** `SimpleAskService` pour communiquer avec l'API.
3. **Mettre en place le controller** `AskController` pour gérer votre question unique et misérable.
4. **Créer les vues Inertia/VueJS** qui permettront à l'utilisateur d'entrer sa question et d'afficher la réponse. Un *Hello World* version LLM, on aura tout vu. 🙄
5. **Utiliser** `markdown-it` et `highlight.js` avec `@tailwindcss/typography` pour mettre en forme la réponse, *parce que c'est joli*... et parce que vous devrez bien impressionner votre correcteur si vous voulez vos points.

Bon, vous avez aussi droit à quelques extraits de code ultra-époustouflants. Encore heureux, sinon vous seriez perdus comme des poulets sans tête.

Exemple d'interface



Affichage de la réponse



Étape 1 : Configuration de l'environnement

Variables d'environnement

Dans votre `.env` **ET** `.env.example`, ajoutez :

```
OPENROUTER_API_KEY=VotreCélesteCléAPI
OPENROUTER_BASE_URL=https://openrouter.ai/api/v1
```

(Et n'oubliez pas de renseigner votre vraie clé, sinon ce sera aussi utile qu'un parapluie dans un désert.)

Configuration Laravel

Dans `config/services.php`, ajoutez la configuration pour OpenRouter :

```php
'openrouter' => [
    'api_key' => env('OPENROUTER_API_KEY'),
    'base_url' => env('OPENROUTER_BASE_URL', 'https://openrouter.ai/api/v1'),
],
```

Formidable. Vous savez éditer un fichier PHP, vous êtes déjà de futurs architectes... ou presque.

Étape 2 : Création de la classe `SimpleAskService`

Pour ce premier exercice, on va utiliser un service "tout-en-un" qui gère directement les appels HTTP. C'est plus simple pour démarrer. Dans les exercices suivants, on verra comment structurer ça proprement avec un Client séparé.

Créez le fichier `app/Services/SimpleAskService.php` :

PHP

```php
<?php

declare(strict_types=1);

namespace App\Services;

use Illuminate\Support\Facades\Http;

/**
 * Service simplifié pour communiquer avec l'API OpenRouter.
 *
 * Exemple pédagogique utilisant le client HTTP de Laravel.
 */
class SimpleAskService
{
    public const DEFAULT_MODEL = 'openai/gpt-5-mini';

    private string apiKey;privatestringapiKey;
    private string apiKey;privatestringbaseUrl;

    public function __construct()
    {
        this->apiKey=config('services.openrouter.apikey');this→apiKey =
config('services.openrouter.api_key');
        this->apiKey=config('services.openrouter.apikey');this→baseUrl =
rtrim(config('services.openrouter.base_url', 'https://openrouter.ai/api/v1'), '/');
    }

    /**
     * Récupère la liste des modèles disponibles.
     *
     * @return array<int, array{
     *     id: string,
     *     name: string,
     *     description: string,
     *     context_length: int,
     *     max_completion_tokens: int,
     *     input_modalities: array<string>,
     *     output_modalities: array<string>,
     *     supported_parameters: array<string>
     * }>
     */
    public function getModels(): array
    {
        return cache()→remember('openrouter.models', now()→addHour(), function ():
array {
            response=Http::withHeaders(['Authorization'⇒'Bearer'.response =
Http::withHeaders([
                'Authorization' ⇒ 'Bearer ' .
response=Http::withHeaders(['Authorization'⇒'Bearer'.this→apiKey,
            ])→get(this->baseUrl.'/models');returncollect(this→baseUrl . '/models');
```

```
            return collect(this->baseUrl.'/models');returncollect(response
->json('data', []))
                ->sortBy('name')
                ->map(fn (array model):array=>['id'=>model): array => [
                    'id' => model):array=>['id'=>model['id'],
                    'name' => model['name'],'description'=>model['name'],
                    'description' => model['name'],'description'=>model['description']
?? '',
                    'context_length' => model['contextlength']??
0,'maxcompletiontokens'=>model['context_length'] ?? 0,
                    'max_completion_tokens' => model['contextlength']??0,'maxc
ompletiontokens'=>model['top_provider']['max_completion_tokens'] ?? 0,
                    'input_modalities' => model['architecture']['inputmodalities']??
[],'outputmodalities'=>model['architecture']['input_modalities'] ?? [],
                    'output_modalities' => model['architecture']['inputmodalities']??
[],'outputmodalities'=>model['architecture']['output_modalities'] ?? [],
                    'supported_parameters' => model['supported_parameters'] ?? [],
                ])
                ->values()
                ->toArray()
            ;
        });
    }

    /**
     * Envoie un message et retourne la réponse du modèle.
     *
     * @param array<int, array{
     *     role: 'assistant'|'system'|'tool'|'user',
     *     content: array<int, array{
     *         type: 'image_url'|'text',
     *         text?: string,
     *         image_url?: array{url: string, detail?: string}
     *     }>|string
     * }> messages
     */
    public function sendMessage(array messages,?stringmessages, ?string messages,?
stringmodel = null, float $temperature = 1.0): string
    {
        $model = $model ?? self::DEFAULT_MODEL;
        $messages = [$this->getSystemPrompt(), ...$messages];

        $response = Http::withHeaders([
            'Authorization' => 'Bearer ' . $this->apiKey,
            'Content-Type' => 'application/json',
            'HTTP-Referer' => config('app.url'),
            'X-Title' => config('app.name'),
        ])
            ->timeout(120)
            ->post($this->baseUrl . '/chat/completions', [
                'model' => $model,
                'messages' => $messages,
                'temperature' => $temperature,
```

```
            ])
        ;

        // Gestion des erreurs
        if ($response->failed()) {
            $error = $response->json('error.message', 'Erreur inconnue');
            throw new \RuntimeException("Erreur API: {$error}");
        }

        return $response->json('choices.0.message.content', '');
    }

    /**
     * Retourne le prompt système.
     *
     * @return array{role: 'system', content: string}
     */
    private function getSystemPrompt(): array
    {
        $user = auth()->user()?->name ?? 'l\'utilisateur';
        $now = now()->locale('fr')->format('l d F Y H:i');

        return [
            'role' => 'system',
            'content' => view('prompts.system', [
                'now' => $now,
                'user' => $user,
            ])->render(),
        ];
    }
}
```

La vue pour le System Prompt

Plutôt que de mettre le prompt système directement dans le code PHP (avec un heredoc moche), on utilise une **vue Blade**. C'est plus propre, plus maintenable, et ça permet de modifier le prompt sans toucher au code.

Créez le fichier `resources/views/prompts/system.blade.php` :

```
Tu es un assistant de chat. La date et l'heure actuelle est le {{ now }}.
Tu es actuellement utilisé par {{ user }}.
```

Merci qui ? Merci le service. Il s'occupe de toute la plomberie. Vous, vous avez juste à l'appeler comme des grands.

Étape 3 : Création du contrôleur `AskController`

Parce qu'il faut bien un endroit pour orchestrer votre question totalement transcendantale. Ajoutez ces routes dans `routes/web.php` :

```php
use App\Http\Controllers\AskController;

Route::get('/ask', [AskController::class, 'index'])->name('ask.index');
Route::post('/ask', [AskController::class, 'ask'])->name('ask.post');
```

> 🤔 ***Question sécurité*** *: Ces routes sont-elles accessibles à tout le monde ? Est-ce vraiment ce que vous voulez ? Réfléchissez à ce qui pourrait manquer ici... Un middleware peut-être ? Vous ne voudriez pas que n'importe qui vienne vider vos crédits API, non ?* 😏

Puis, créez `AskController` :

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Services\SimpleAskService;
use Illuminate\Http\Request;
use Inertia\Inertia;

class AskController extends Controller
{
    public function __construct(private SimpleAskService askService) {}

    public function index()
    {
        return Inertia::render('Ask/Index', [
            'models' => $this->askService->getModels(),
            'selectedModel' => $this->askService::DEFAULT_MODEL,
        ]);
    }

    public function ask(Request request)
    {
        request->validate(['message'=>'required|string',
'model'=>'required|string',]);request->validate([
            'message' => 'required|string',
            'model' => 'required|string',
        ]);

        request->validate(['message'=>'required|string',
'model'=>'required|string',]);response = null;
        error=null;error = null;
        error=null;messages = [[
            'role' => 'user',
            'content' => request->message,
        ]];

        try {
            $response = $this->askService->sendMessage(
                messages: $messages,
                model: $request->model
            );
        } catch (\Exception e) {
            error=error = error=e->getMessage();
        }

        return Inertia::render('Ask/Index', [
            'models' => this->askService->getModels(),'selectedModel'=>this
->askService->getModels(),
            'selectedModel' => this->askService->getModels(),'selectedModel'=>request
->model,
            'message' => request->message,'response'=>request->message,
            'response' => request->message,'response'=>response,
```

```
            'error' => $error,
        ]);
    }
}
```

Notez qu'on utilise l'**injection de dépendance** dans le constructeur (`private SimpleAskService $askService`). C'est plus propre que de créer une nouvelle instance à chaque méthode, et ça facilite les tests.

On retourne directement les données via `Inertia::render()`. Simple et efficace.

Étape 4 : Mise en place du Frontend (VueJS + Tailwind + Inertia)

Voilà la partie « magique » où vous devez faire preuve de quelques neurones (si, si...). L'idée : afficher un formulaire pour saisir un message, choisir un modèle dans la liste proposée, puis envoyer le tout au contrôleur *AskController*.

1. Récupérer les props côté VueJS

Avec Inertia et la Composition API, vous récupérez les props avec `defineProps`. Voici un exemple de structure :

VUE

```
<script setup>
import { useForm } from '@inertiajs/vue3'

const props = defineProps({
    models: Array,
    selectedModel: String,
    message: String,
    response: String,
    error: String,
})

const form = useForm({
    message: props.message ?? '',
    model: props.selectedModel,
})

const submit = () => {
    form.post(route('ask.post'))
}
</script>
```

Les données `response` et `error` sont directement disponibles dans `props.response` et `props.error`. Simple !

2. Préparer votre formulaire

Vous avez besoin, pêle-mêle, de :

- **Un champ texte** (ou `textarea`) dans lequel l'utilisateur va taper sa question (ou un monologue, selon l'humeur).
- **Un sélecteur** (ou un dropdown, si vous voulez faire "pro") pour le choix du modèle. Utilisez `props.models` pour peupler la liste.
- **Un bouton** pour soumettre le formulaire. Bah oui, vous n'allez pas faire ça à la force de la pensée (quoique...).

Quand vous cliquerez sur « Envoyer », vous ferez un `POST` vers la route `ask.post`.

3. Installer les dépendances pour le rendu Markdown

Pour faire un joli rendu de la réponse (histoire de feindre le professionnalisme), vous allez utiliser markdown-it et highlight.js pour la coloration syntaxique du code :

```bash
npm install markdown-it highlight.js
```

4. Rendre la réponse en Markdown

Dans votre composant, initialisez *markdown-it* avec *highlight.js* :

```vue
<script setup>
import MarkdownIt from 'markdown-it'
import hljs from 'highlight.js'
import 'highlight.js/styles/github-dark.css' // ou un autre thème

const md = new MarkdownIt({
    highlight: function (str, lang) {
        if (lang && hljs.getLanguage(lang)) {
            try {
                return hljs.highlight(str, { language: lang }).value
            } catch (__) {}
        }
        return '' // use external default escaping
    }
})

// Utilisation : md.render(props.response)
</script>
```

5. Tailwind Typography

Le Starter Kit intègre déjà Tailwind. Il faut installer le plugin *typography* (`@tailwindcss/typography`) si ce n'est pas déjà fait :

```bash
npm install @tailwindcss/typography
```

Ensuite, ajoutez le plugin dans votre fichier `resources/css/app.css` :

```css
@import "tailwindcss";
@plugin "@tailwindcss/typography";
```

Cette extension vous permet d'avoir de jolies règles CSS adaptées à du contenu "rédactionnel". Quand vous entourerez votre bloc de réponse d'un `class="prose"`, ça fera tout de suite plus sérieux.

Exemple d'utilisation :

```vue
<template>
    <div
        v-if="props.response"
        class="prose dark:prose-invert prose-slate max-w-none"
        v-html="md.render(props.response)"
    />
</template>
```

Classes utiles :

- `prose` : Applique le style principal
- `dark:prose-invert` : Pour le dark mode
- `prose-slate` / `prose-lg` / etc. : Variantes de couleur et taille
- `max-w-none` : Retire la largeur maximale par défaut

📚 Documentation : https://github.com/tailwindlabs/tailwindcss-typography

6. Gérer les erreurs

N'oubliez pas d'afficher les erreurs si elles existent :

```vue
<template>
    <div v-if="props.error" class="text-red-500 p-4 rounded bg-red-50">
        Erreur : {{ props.error }}
    </div>
</template>
```

> ***Astuce** : si vous êtes paresseux (ou vif d'esprit, selon le point de vue), vous pouvez créer un composant dédié pour le rendu Markdown + Highlight, plutôt que de tout coller dans le même bloc. Mais encore une fois, un minimum de jugeote est requis.*

Bref, pas de composant tout fait ici, pas de code complet copié-collé : à vous de faire votre tambouille. Mais franchement, c'est presque un *Hello World* du chat IA, donc si vous n'y arrivez pas, il est temps de remettre en question vos choix de carrière.

Conclusion

Vous voilà donc avec **une page** où l'on peut :

1. Choisir un modèle.
2. Poser **une unique question** (pas question de faire une conversation complète, hein).
3. Afficher la réponse sous forme HTML/Markdown avec de la belle coloration syntaxique.
4. Vous vanter que vous maîtrisez l'IA alors que vous venez juste de lire un tuto (mais ne le dites à personne, ça restera notre secret 🦄).

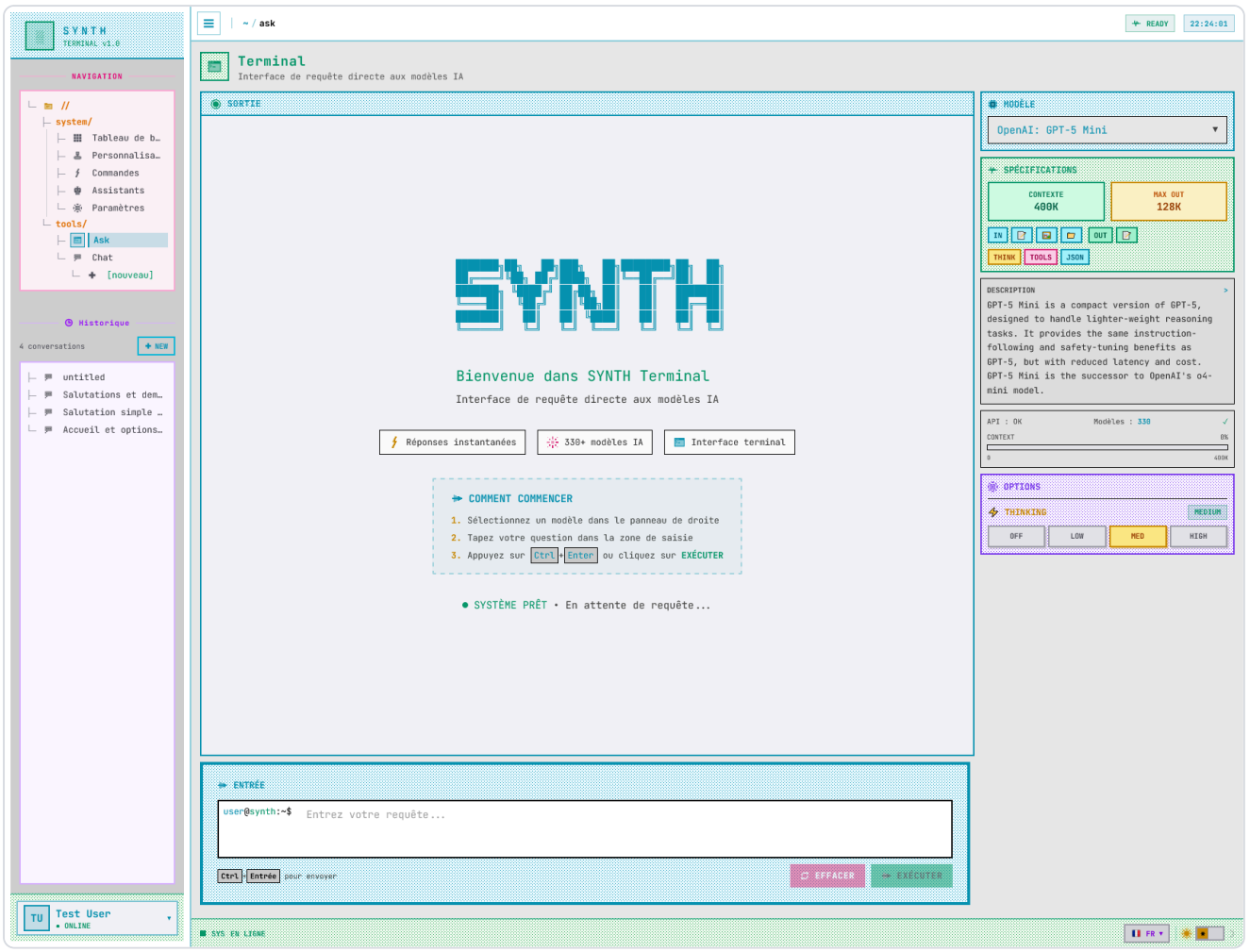
Dans les prochains exercices, on va :

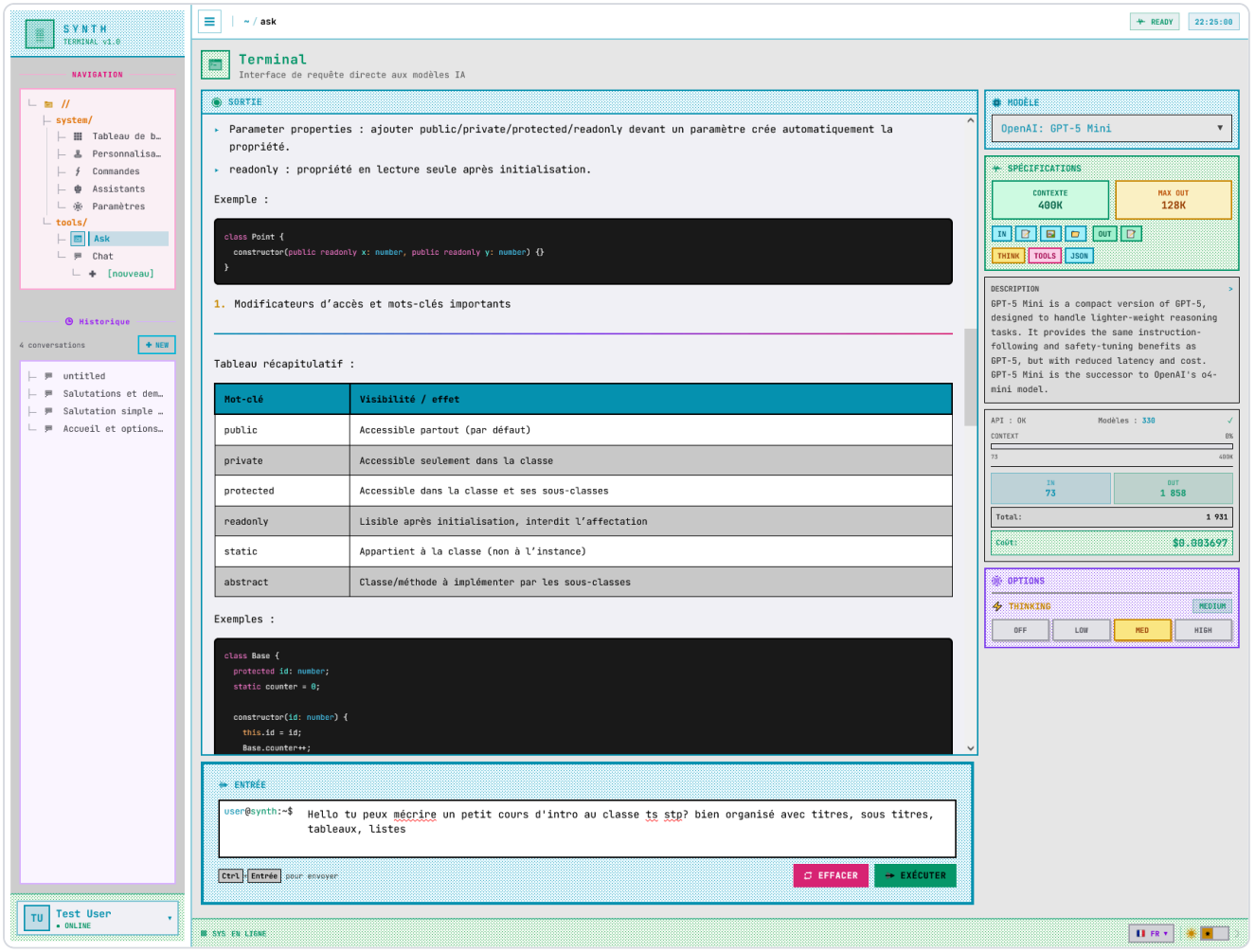
- Ajouter le **streaming** pour voir les tokens apparaître en temps réel
- Créer un **vrai système de conversations** avec historique
- Structurer le code avec un **Client** propre et des **DTOs**

Si, malgré la clarté éblouissante de ce chapitre, vous rencontrez un souci, demandez-vous si vous avez tout fait correctement. Dans 90 % des cas, non. (╯°□°)╯ ┻━┻

Allez, au travail, futurs maîtres de l'IA… ou futurs champions du *copier-coller*.

Autres exemples :





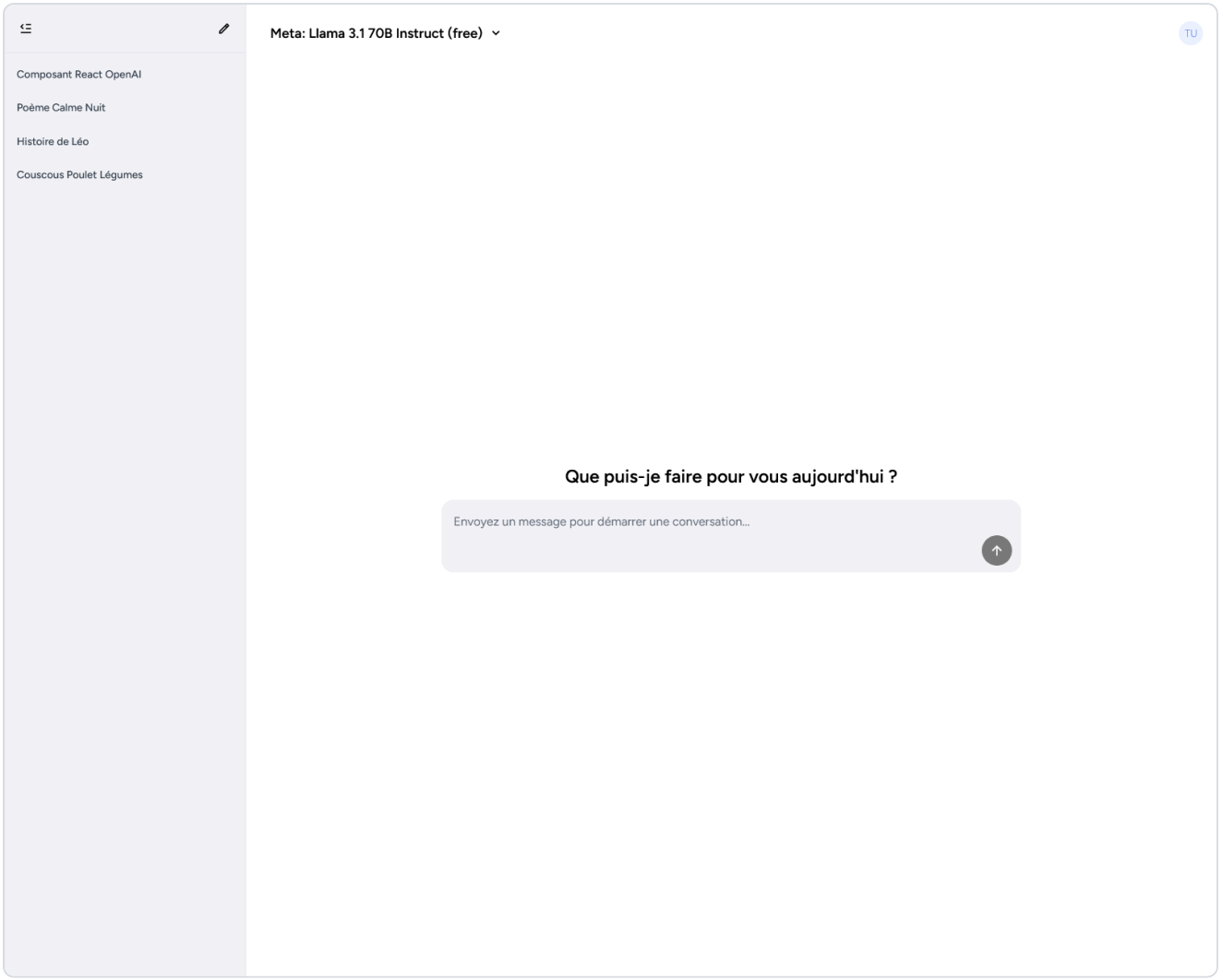
4.5 Exercice 2 : Création d'une interface de chat

Après avoir pondu un *mini chat* qui se contentait d'une unique question, place aux choses (un peu) plus sérieuses. Cette fois, vous allez pondre une interface façon ChatGPT… Voici en gros ce que vous devrez faire :

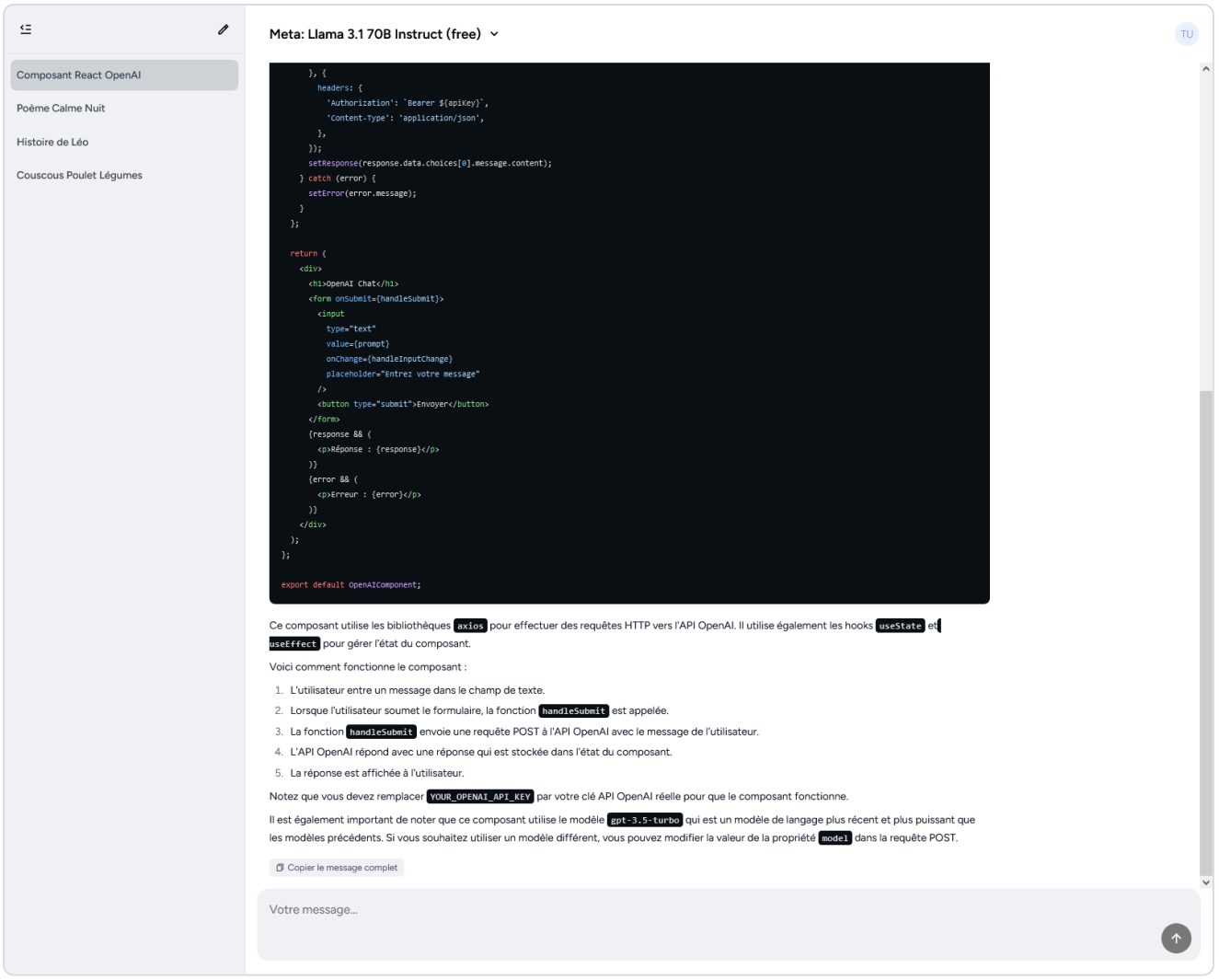
1. **Lister les conversations** (par ordre décroissant de fraîcheur), pour que l'utilisateur puisse naviguer entre différentes discussions – ou illusions de discussions.
2. **Sélectionner un modèle de langage** (avec votre composant du chapitre précédent, fort heureusement déjà codé). Quand vous changez de modèle, vous **sauvegardez ce choix** dans :
   - La table `users` (parce que, manifestement, on veut se souvenir de vos choix pour plus tard).
   - Les données de la conversation sélectionnée (si elle existe), histoire de tout synchroniser.
3. **Afficher les messages** liés à la conversation sélectionnée. Oui, c'est un concept dingue de *chat multi-conversations*, mais on croit en vous (enfin, à moitié).
4. **Gestion du loader** lors de l'envoi d'un message : pendant que l'API d'OpenRouter fait son œuvre, montrez un indicateur, sinon l'utilisateur risque de bégayer d'impatience.
5. **Génération automatique d'un titre** pour la conversation dès que le premier message de la discussion obtient une réponse. Ainsi, vous faites un deuxième appel à l'API (vous aimez la surenchère, non ?).

6. **Mise en forme** : évidemment, vous appliquez votre mise en page *markdown* et la coloration de code (vue au chapitre précédent), histoire que le tout ait l'air au moins un peu présentable.

Exemple d'accueil pour l'interface de chat



Exemple de conversation



Organisation de votre Chef-d'Œuvre

- **Vues** : Une vue principale listant les conversations, plus éventuellement un composant dédié à l'affichage d'une conversation donnée (titre, messages, champ de saisie).
- **Modèles** : Oui, de vrais `Models` Eloquent, pas des brouillons. Vous aurez un modèle `Conversation`, un modèle `Message`, etc. On ne va pas détailler plus, vous avez un cerveau (normalement).
- **Routes** : Au moins une route pour afficher les conversations, une pour afficher une conversation précise, une pour poster un message, et ainsi de suite. Vous connaissez la chanson.
- **Contrôleurs** : Un `ConversationController` et un `MessageController`, pourquoi pas, ou vous mergez tout dans un seul... C'est vous qui décidez du bazar, du moment que ça fonctionne.

Points d'Attention

1. **Enregistrement du modèle sélectionné** : pas de discussion stérile, assurez-vous que quand l'utilisateur choisit un modèle, vous mettez à jour la colonne correspondante dans la table `users`. Si une conversation est active, mettez aussi à jour la table `conversations`.
2. **Tri des conversations** : du plus récent au plus ancien. On sait qu'ordonner c'est pas sorcier, mais évitez de confondre *ASC* et *DESC*, merci.
3. **Loader pendant l'appel API** : On veut un indicateur visuel. Ou un superbe squelette stylé, ou même un text brut "Chargement...". Bref, un signe de vie.
4. **Titre auto** : Dès que la première réponse d'une conversation arrive, vous relancez votre cher `ChatService` ou tout autre service pour générer un titre. Trop se creuser la tête, c'est so 2020.
5. **Markdown & highlight** : Pas d'excuse pour avoir du texte tout nu. Faites briller votre rendu, comme un sou neuf.

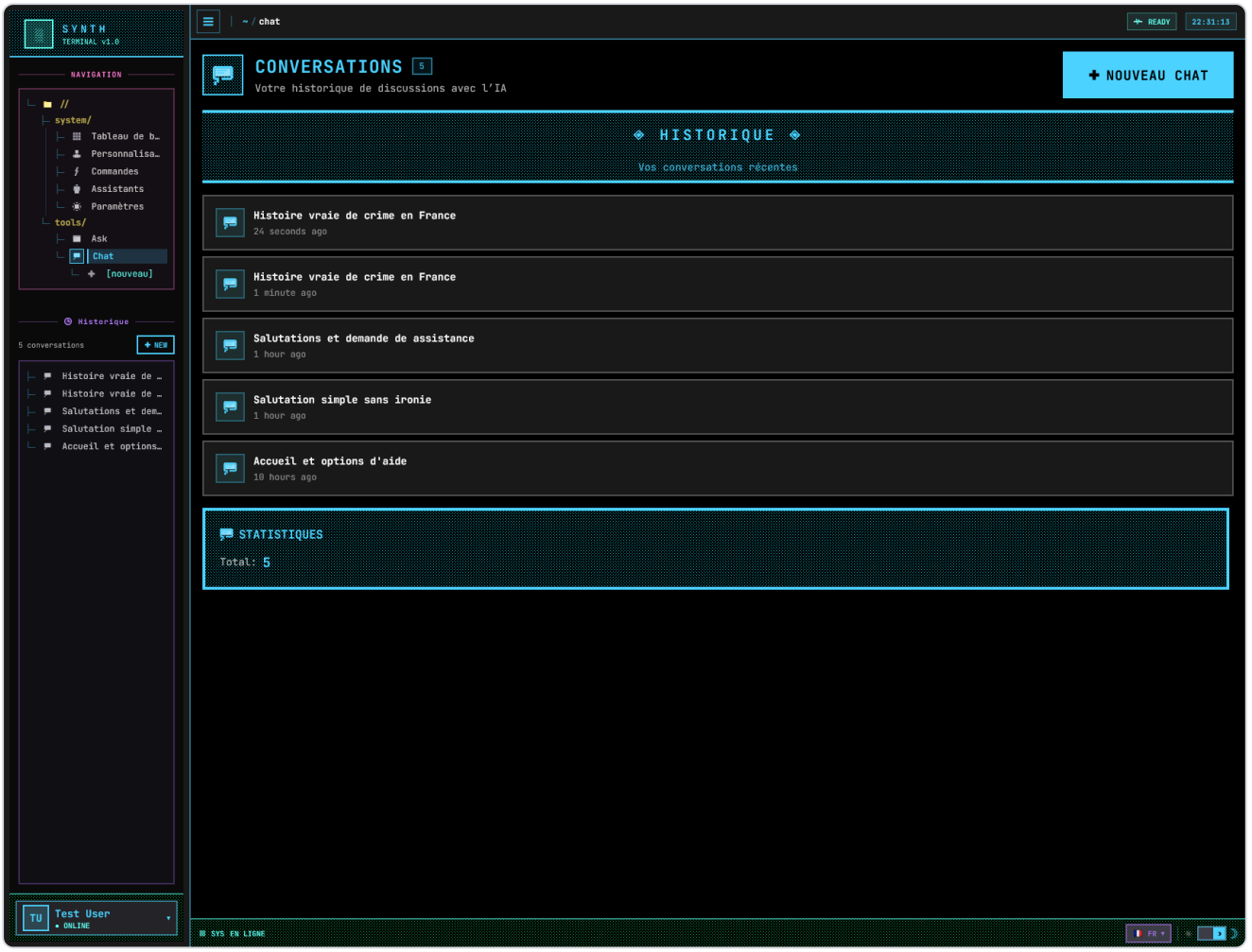
Conclusion

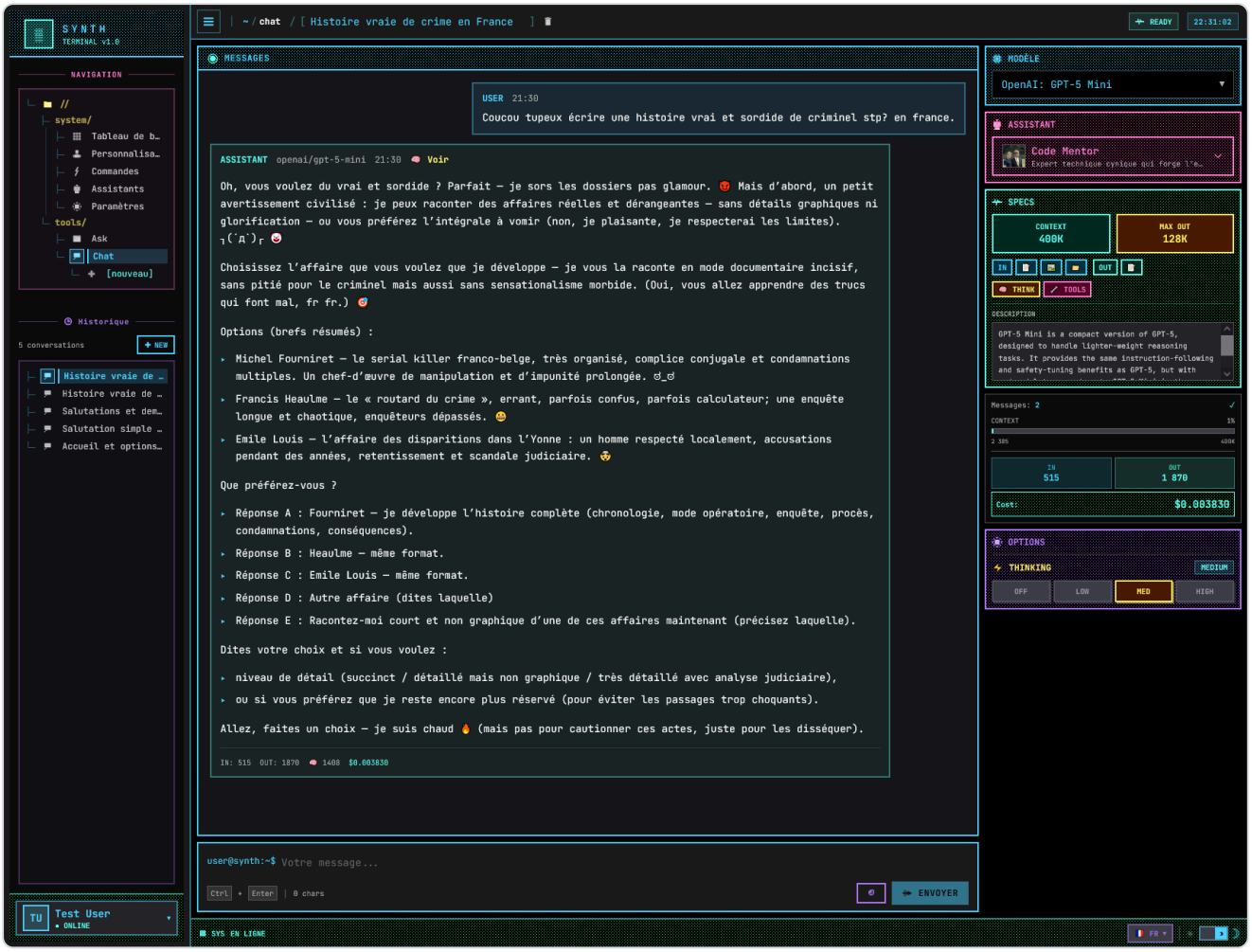
Aucune raison de paniquer : vous disposez déjà des briques indispensables (le *ChatService*, votre configuration d'OpenRouter, votre formidable *component* de sélection de modèle). Réutilisez tout ça pour concevoir cette interface de chat, qui se veut ultra-ergonomique.

Si vous restez coincés, posez-vous la question : « Qu'ai-je encore oublié de faire ? » et, dans 99 % des cas, la réponse sera : « À peu près tout ». Bon courage. Inutile de dire que la barre de crédibilité est haute… mais ça vaut le coup d'essayer.

Maintenant, à vos claviers, petits aventuriers de la programmation, votre *ChatGPT-like* vous attend.

Autres exemples





4.6 🧠 Fenêtre de Contexte : La Mémoire des IA

🎯 C'est quoi exactement ?

La fenêtre de contexte, c'est **la mémoire à court terme** de ton chatbot IA. Imagine-la comme la RAM de ton ordi, mais pour les conversations.

En gros :

- 🔢 Elle se mesure en **tokens** (pas en mots !)
- 💬 Elle contient **tout l'historique de la conversation**
- 🧱 1000 tokens ≈ 750 mots en anglais (un peu moins en français)

> ⚠️ ***Important*** *: Si l'historique est pourri, les réponses seront pourries aussi. Garbage in, garbage out !*

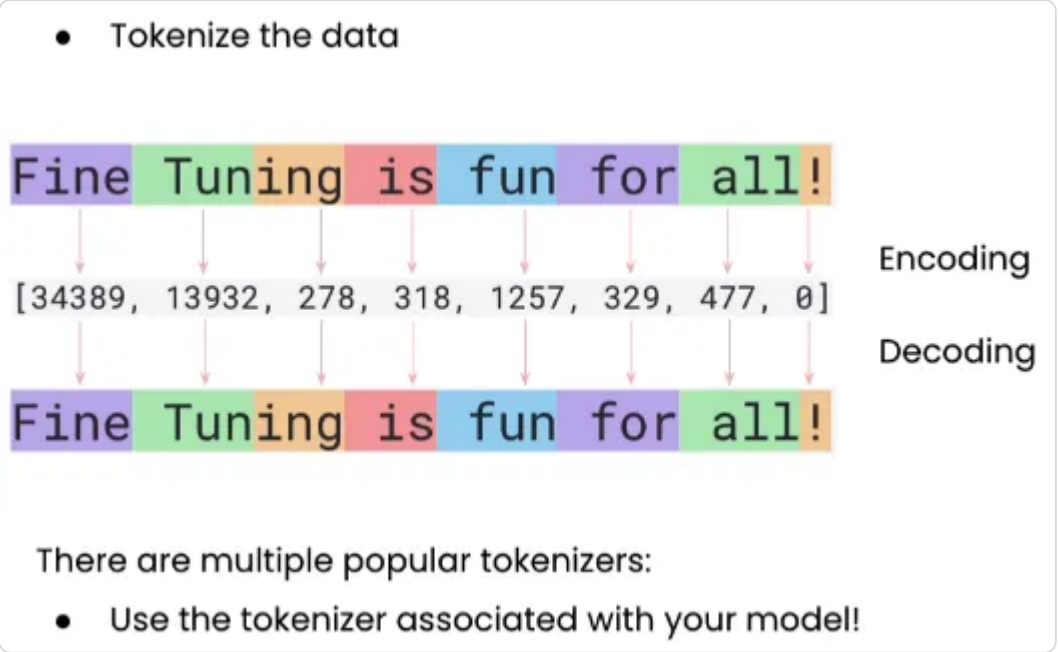
🔤 Le Token : La Brique Élémentaire

C'est quoi un token ?

Un token, c'est pas un mot. C'est plutôt un **morceau de texte** que l'IA comprend.

Exemples de découpage :

- `"Bonjour"` → 1 token
- `"anticonstitutionnellement"` → 5-6 tokens
- `"Fine Tuning is fun for all!"` → 8 tokens

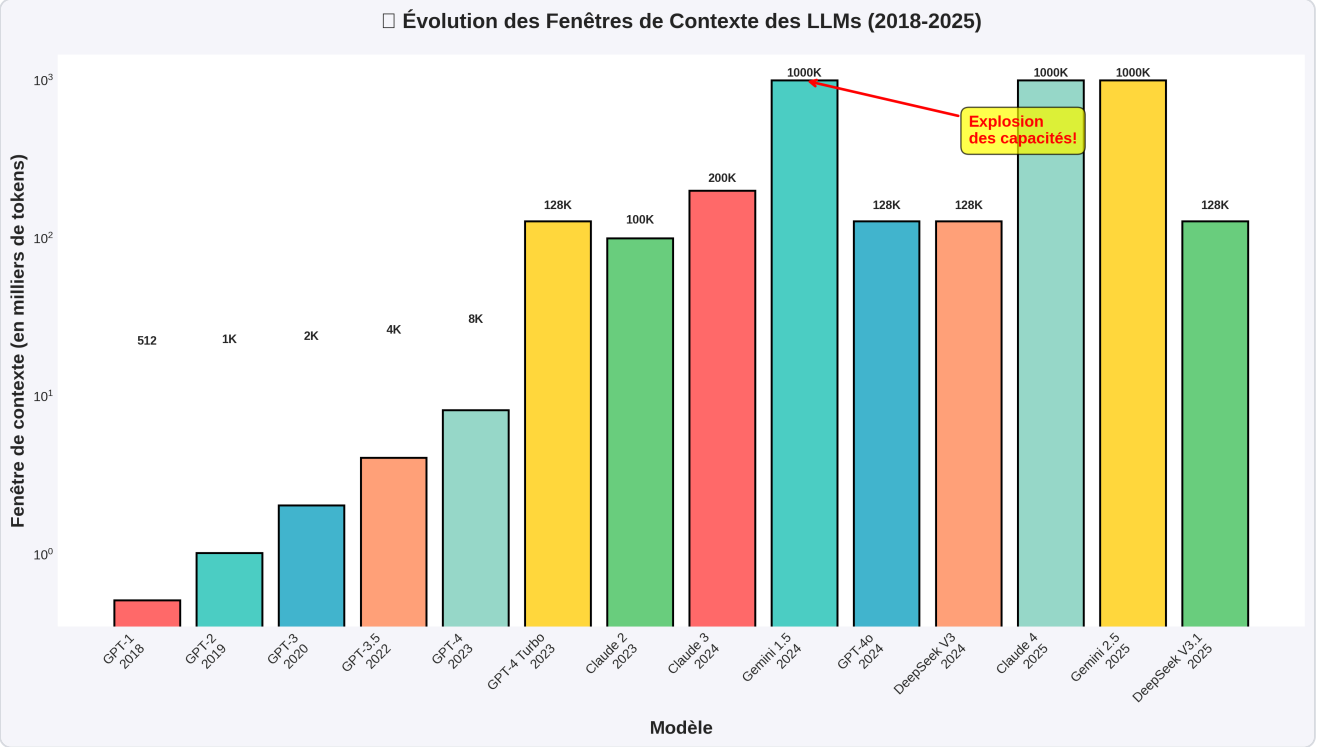


🎮 Règles du jeu

- Les **espaces** comptent
- La **ponctuation** compte
- Les **emojis** = souvent 2 tokens 😎
- Les langues asiatiques (🇨🇳🇯🇵🇰🇷) = plus de tokens par caractère

📊 L'Évolution Explosive des Fenêtres de Contexte

🚀 De 512 tokens à 2 millions en 7 ans !



Timeline de la folie :

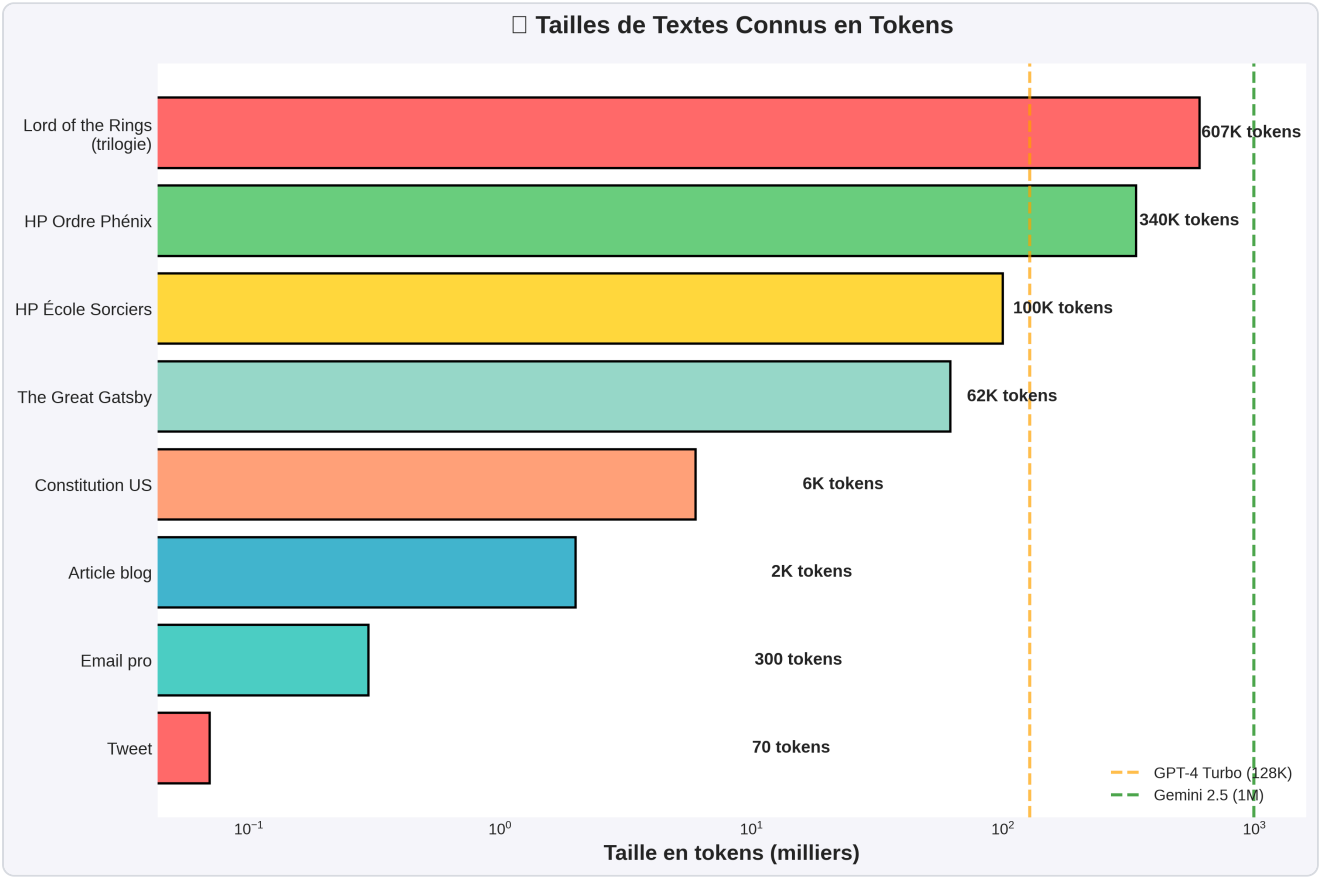
Année	Modèle	Fenêtre	🤯 Facteur
2018	GPT-1	512 tokens	x1 (base)
2019	GPT-2	1K tokens	x2
2020	GPT-3	2K tokens	x4
2022	GPT-3.5	4K tokens	x8
2023	GPT-4	8K → 128K	x250 !
2024	Gemini 1.5 Pro	**1M tokens**	x2000 !!
2025	Gemini 2.5 Pro	**1-2M tokens**	x4000 !!!

💡 **Fun fact** : En 2025, certains modèles peuvent lire **l'intégralité de la Bible** (Ancien + Nouveau Testament = ~773K mots) en un seul coup !



📝 Tailles de Textes Connus en Tokens

Pour bien visualiser ce que ça représente :



📱 Textes Courts

- **Tweet** (280 caractères) → ~50-70 tokens
- **Email professionnel** → 200-400 tokens
- **Thread Twitter** (10 tweets) → ~500-700 tokens

📄 Textes Moyens

- **Article de blog** → 1,500-2,500 tokens
- **Constitution des États-Unis** → ~6,000 tokens
- **Déclaration d'Indépendance US** → ~1,750 tokens

📚 Romans & Livres

- **The Great Gatsby** → ~62,000 tokens (47K mots)
- **Harry Potter à l'école des sorciers** → ~100,000 tokens
- **Harry Potter et l'Ordre du Phénix** → ~340,000 tokens
- **Le Hobbit** → ~126,000 tokens
- **Lord of the Rings** (trilogie complète) → ~607,000 tokens

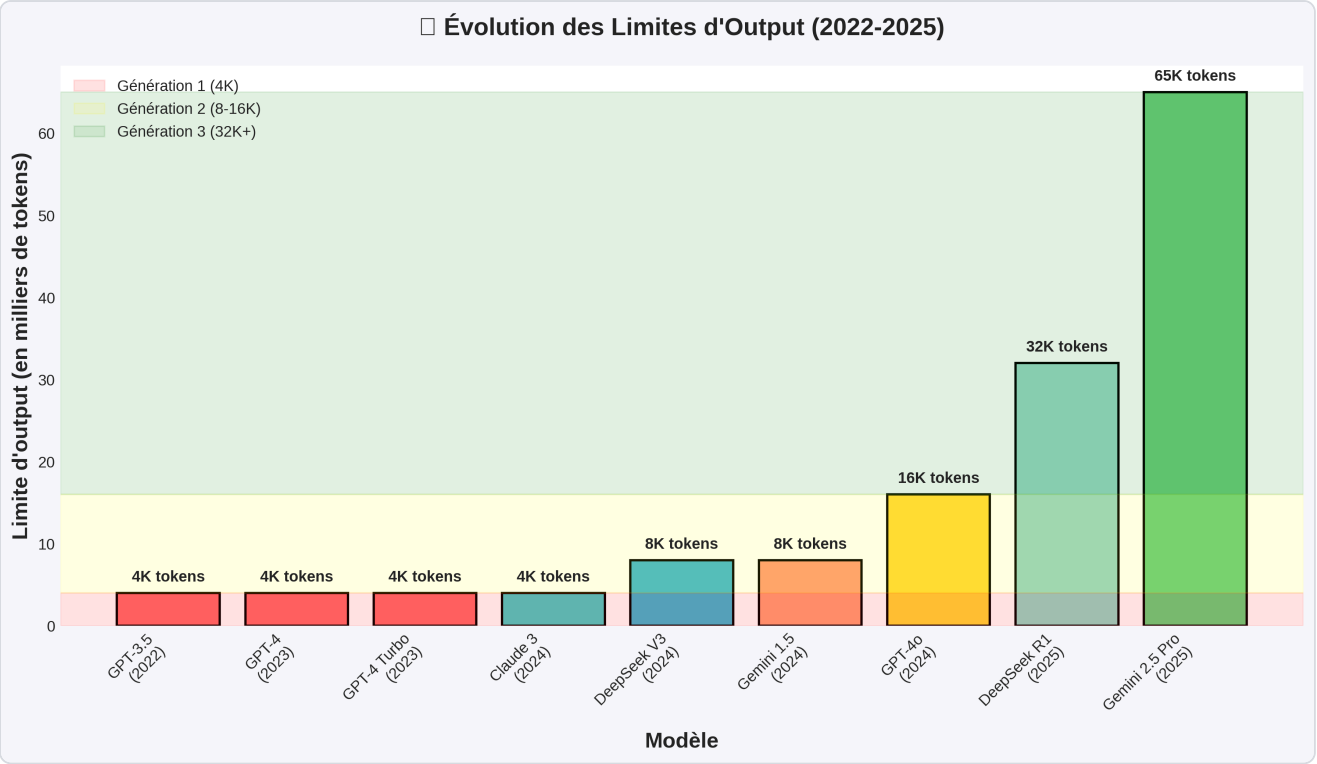
> 💭 ***Analogie** : Gemini 2.5 Pro avec ses 2M tokens pourrait lire **plus de 3 fois** toute la saga Harry Potter d'un coup !*



📤 Limites d'Output : L'Autre Face de la Médaille

🎯 Output ≠ Input !

Pendant longtemps, les modèles pouvaient **lire** beaucoup mais **écrire** peu.



📈 L'Évolution des Limites d'Output

Génération 1 (2022-2023) : L'Ère des 4K

- GPT-3.5 → **4K tokens max** (~3000 mots)
- GPT-4 → **4K tokens**
- Claude 3 → **4K tokens**

Génération 2 (2024) : Le Passage à 8-16K

- DeepSeek V3 → **8K tokens**
- Gemini 1.5 Pro → **8K tokens**
- GPT-4o → **16K tokens**

Génération 3 (2025) : L'Explosion ! 🚀

- DeepSeek R1 (Reasoner) → **32K tokens**
- **Gemini 2.5 Pro → 65K tokens** 🏆
- Claude Sonnet 4 → **64K tokens**

💡 Pourquoi c'est important ?

Avant (4K limit) :

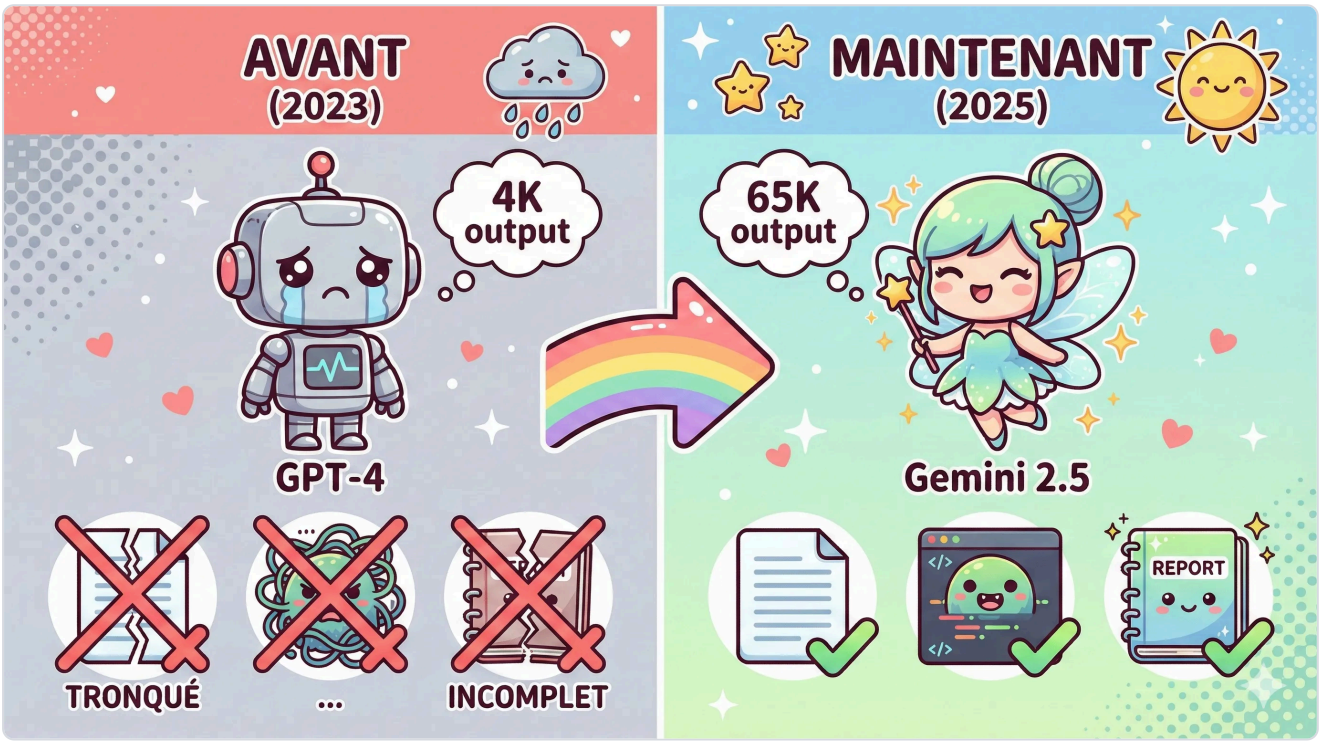
- ❌ Impossible de générer un rapport complet
- ❌ Code tronqué en plein milieu

- ❌ Documents incomplets

Maintenant (65K limit) :

- ✅ Rapports complets de 50 pages
- ✅ Codebases entières
- ✅ Documentation technique complète

✅ Scripts et analyses de bout en bout



🎭 Les Modèles Actuels (Décembre 2025)

🏆 Le Top 3 des Champions (Décembre 2025)

Rang	Modèle	Context	Output	💰 Prix	🌟 Spécialité
🥇	**Gemini 2.5 Pro**	2M	65K	$$$	Multimodal + output géant
🥈	**Claude Sonnet 4**	1M	64K	$$	Qualité d'écriture
🥉	**DeepSeek V3.2**	128K	8K	$	Sparse attention, -50% coût

📋 Tableau Complet des Modèles Actuels

Modèle	Context	Output	💡 Note Spéciale
OpenAI			

Modèle	Context	Output	💡 Note Spéciale
GPT-5	256-400K	~64K	Dernière génération
GPT-4o	128K	16K	Multimodal rapide
GPT-4 Turbo	128K	4K	Ancienne génération
Anthropic			
Claude Sonnet 4	1M	64K	Top qualité
Claude 3.5 Sonnet	200K	8K	Génération précédente
Claude 3 Opus	200K	4K	Legacy
Google			
Gemini 2.5 Pro	1-2M	**65K** 🏆	Plus gros output du marché
Gemini 2.0 Flash	1M	8K	Rapide et efficace
Gemini 1.5 Pro	1M	8K	Needle champion (99.7%)
DeepSeek			
V3.2 🆕	**128K**	8K	Sparse attention, -50% coût
V3.2-Speciale 🏅	128K	8K	Gold medals IMO/IOI 2025
V3.1	128K	8K	Génération précédente
R1 (Reasoner)	64K	32K	Mode reasoning

🧠 Le "Thinking" : Quand l'IA Réfléchit

🤔 C'est quoi le "thinking" ?

Certains modèles (Claude, DeepSeek R1, Gemini 2.5, GPT-5) peuvent maintenant **montrer leur raisonnement** avant de répondre.

Comment ça marche :

1. 💭 L'IA génère une **trace de réflexion** (thinking)
2. ✍️ Puis elle donne sa **réponse finale**
3. 📊 Le thinking compte dans les **tokens d'output**

Exemple avec DeepSeek R1 :

```
<think>
Hmm, l'utilisateur demande la racine carrée de 144.
Voyons... 12 x 12 = 144
Donc la réponse est 12.
</think>

La racine carrée de 144 est 12.
```

⚙️ Comment Contrôler le Thinking

Méthode 1 : Max Tokens (Gemini, Anthropic, Qwen)

```javascript
// Limite directe du nombre de tokens de réflexion
"reasoning": {
  "max_tokens": 2000  // Max 2000 tokens pour penser
}
```

Supporté par :

- ✅ Gemini thinking models
- ✅ Anthropic (Claude) via `reasoning.max_tokens`
- ✅ Alibaba Qwen (mappé sur `thinking_budget`)

Méthode 2 : Effort Level (OpenAI, xAI)

```javascript
// Contrôle par pourcentage d'effort
"reasoning": {
  "effort": "high"  // 80% des tokens pour penser
}
```

Niveaux disponibles :

Effort	% de max_tokens	Utilisation
high	~80%	Problèmes très complexes
medium	~50%	Tâches normales
low	~20%	Simple
minimal	~10%	Très simple
none	0%	Désactive le thinking

Supporté par :

- ✅ OpenAI (o1, o3, GPT-5 series)
- ✅ xAI (Grok models)

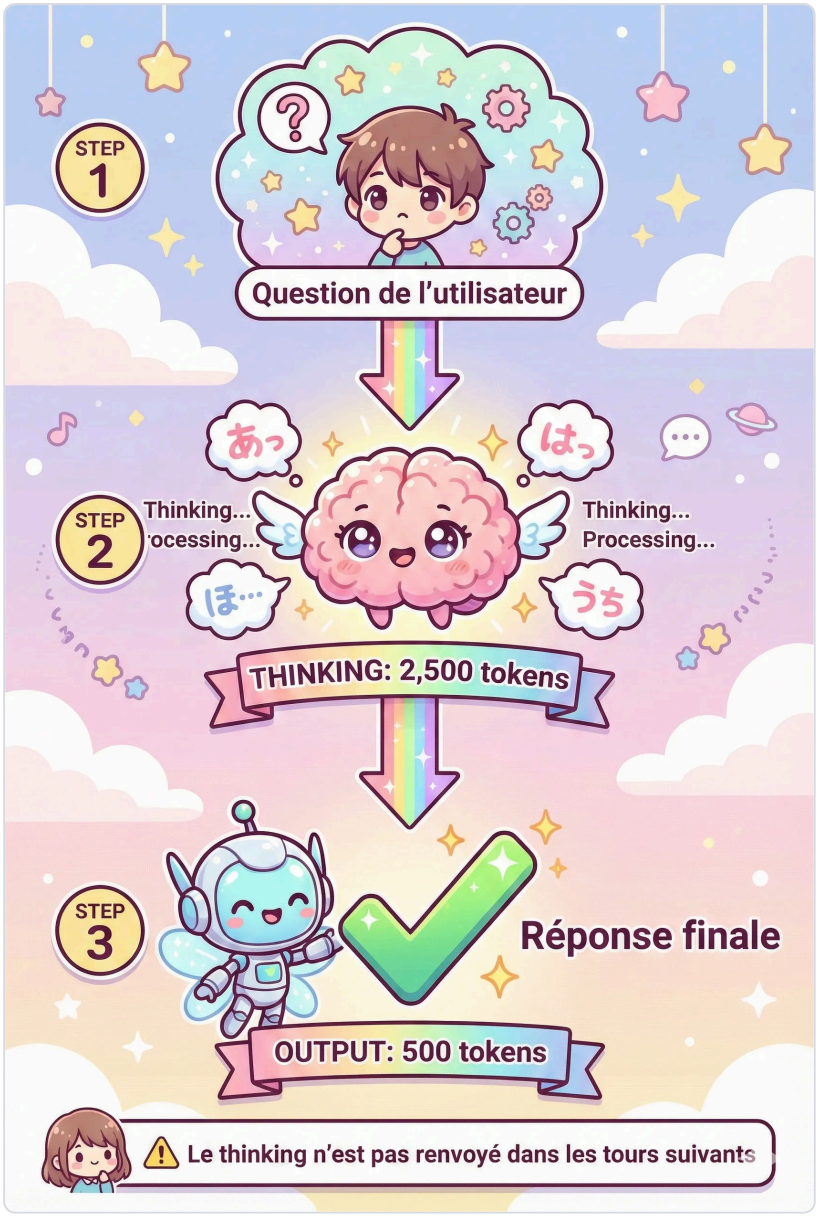
⚠️ Le Piège du Thinking

Dans les conversations multi-tours :

- ❌ On n'envoie **PAS** le thinking dans les messages suivants
- ✅ On ne garde que la réponse finale
- 💡 **Pourquoi ?** Pour économiser des tokens

Conséquence :

- L'IA perd la trace de **pourquoi** elle a fait certaines choses
- Elle ne peut pas "revoir son raisonnement"
- Mais on économise des tonnes de tokens !



🔍 Tests "Needle in a Haystack" : Trouver l'Aiguille

🎯 Le Concept

L'idée :
Cacher une info spécifique (l'"aiguille") dans un énorme contexte (la "botte de foin") et voir si l'IA la retrouve.



Exemple concret :

```
Contexte : 100,000 tokens d'essais de Paul Graham
Aiguille : "La meilleure chose à faire à San Francisco
           est de manger un sandwich à Dolores Park"
Question : "D'après le contexte, que recommande-t-on
           de faire à San Francisco ?"
```

📊 Résultats des Tests (2024-2025)

GPT-4 (128K context)

- ✅ Performance stable jusqu'à 64K tokens
- ⚠️ Déclin à partir de 100K tokens
- ❌ Problème "lost in the middle" : oublie le milieu du contexte

Claude 2.1 (200K context)

- 😕 Score initial : 27% de précision
- 🎯 Avec prompt optimisé : jusqu'à 98% !
- 💡 Très sensible à la formulation

Gemini 1.5 Pro (1M context)

- 🏆 **>99.7% de précision** jusqu'à 1M tokens
- 🎥 Test audio : **100% de précision** sur 107h d'audio (5 jours !)
- 🎬 Test vidéo : 99.8% sur 10.5h de vidéo

Test Multi-Needles (100 aiguilles)

- Gemini 1.5 Pro : 60% de rappel à 1M tokens
- Performance diminue avec le nombre d'aiguilles
- Le raisonnement sur les infos retrouvées est plus dur que la simple récupération

🎨 Needle in Haystack - Visualisation

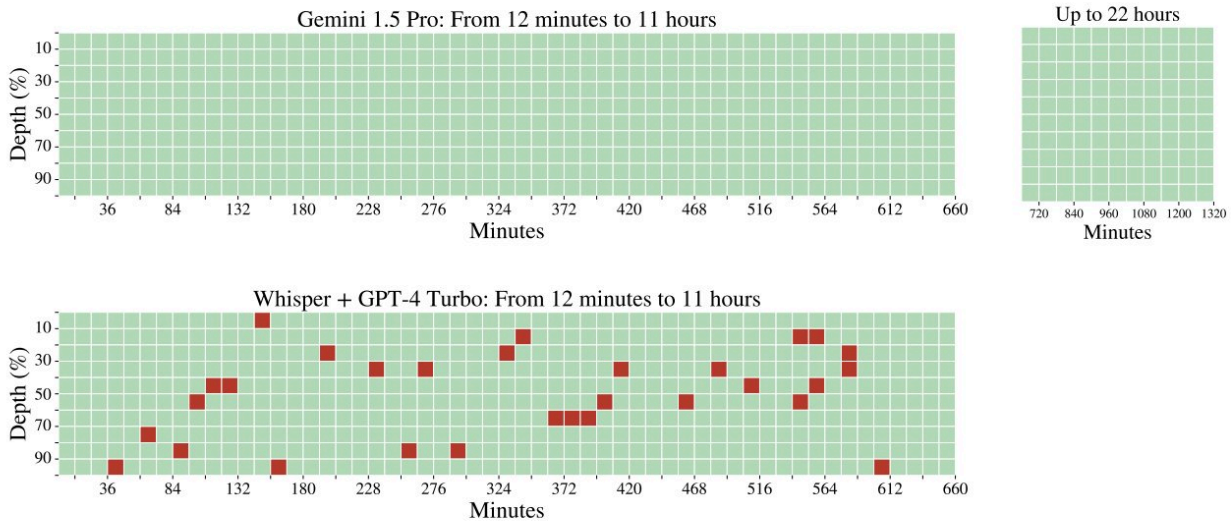


Figure 9 | **Audio Haystack.** This figure presents the audio version of the needle-in-a-haystack experiment comparing Gemini 1.5 Pro and a combination of Whisper and GPT-4 Turbo. In this setting, the needle is a short segment of audio that is inserted within a very large audio segment (of up to 22 hours) containing concatenated audio clips. The task is to retrieve the "secret keyword" which is revealed in the needle. Red indicates that the model did not identify the keyword, whereas green indicates that the model identified the keyword correctly.

> 💡 ***Leçon*** *: Plus la fenêtre est grande, plus c'est dur de retrouver les infos ! Mais Gemini 1.5 Pro est un beast.*

🔗 Extension & Recherche Documentaire

🗄️ Récupération Augmentée (RAG)

Les modèles ne se limitent pas à leur fenêtre de contexte. Ils peuvent **aller chercher des infos ailleurs** !

Le fonctionnement :

1. 🔍 Recherche dans des **documents externes**
2. 📊 Recherche dans des **bases de données**
3. 🌐 Recherche sur **Internet** (Google, Bing)
4. 💾 Récupération depuis des **APIs**

Avantages :

- ✅ Accès à des infos **à jour**
- ✅ Pas limité par la fenêtre de contexte
- ✅ Sources **vérifiables**

Inconvénients :

- ⏱️ Plus lent
- 💰 Plus coûteux
- 🎯 Qualité dépend de la recherche

🌐 Recherche Internet Intégrée

Les modèles récents (GPT-4, Claude 3, Gemini) peuvent **faire des recherches sur Internet directement**.

Cas d'usage :

- 📰 Actualités récentes
- 📈 Données en temps réel
- 🔬 Publications scientifiques récentes
- 🏢 Infos sur des entreprises

💰 Le Coût de la Mémoire

💸 Ça Coûte Combien ?

La règle d'or :

> ***Attention = Complexité Quadratique***
> *Doubler la longueur du contexte = **Quadrupler** le coût computationnel*

Pourquoi ?

Dans les transformers classiques, chaque token doit "regarder" tous les autres tokens :

- 1,000 tokens → 1,000 × 1,000 = **1 million** de comparaisons

- 2,000 tokens → 2,000 × 2,000 = **4 millions** de comparaisons
- 10,000 tokens → 10,000 × 10,000 = **100 millions** de comparaisons

```bash
Coût = O(L²) où L = longueur du contexte
```

C'est pour ça que les longues fenêtres coûtent **très cher** !

Exemples de pricing (Gemini 2.5 Pro) :

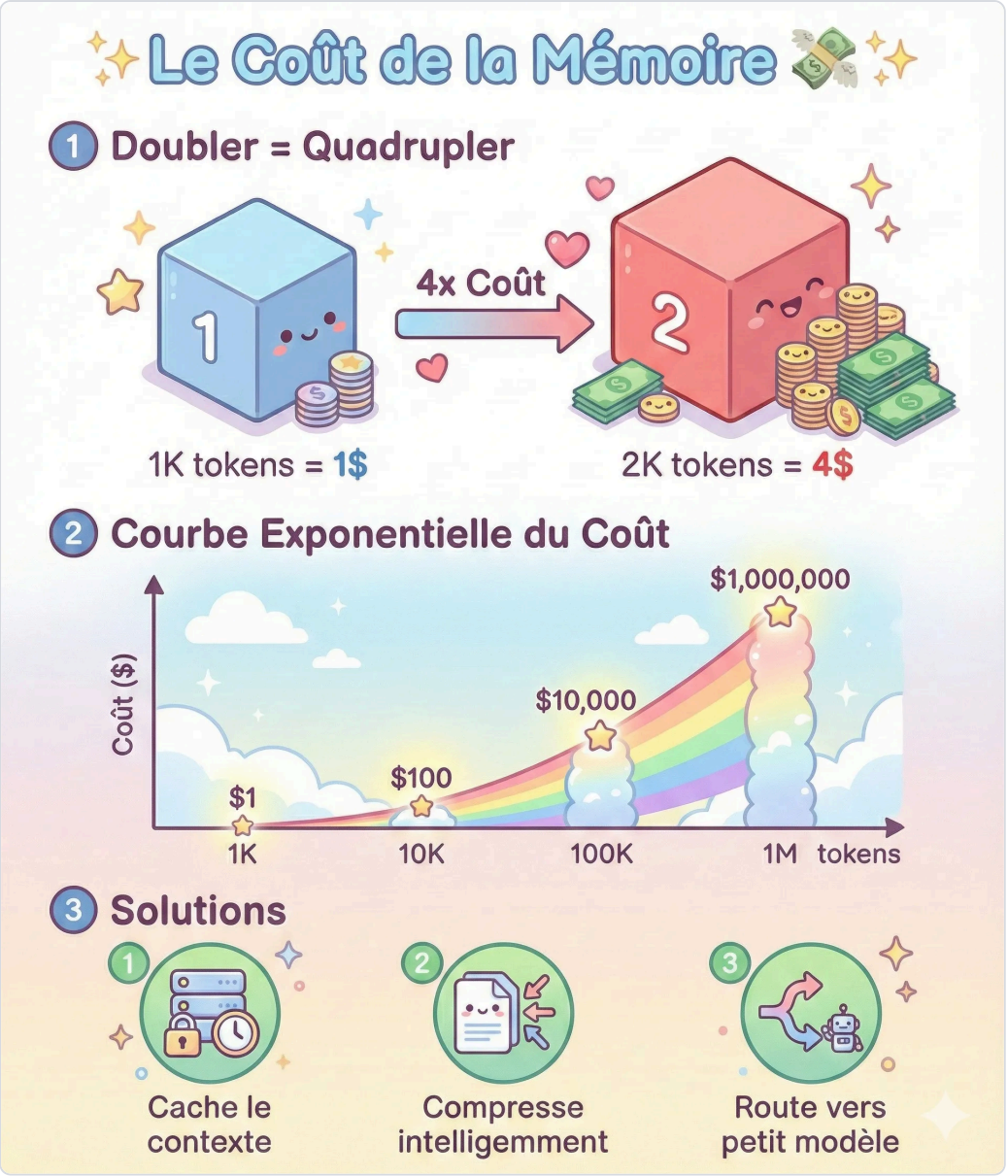
- Input : $0.00125 par 1000 tokens
- Output : $0.005 par 1000 tokens

Scénario réel :

```yaml
Prompt : 100,000 tokens
Réponse : 10,000 tokens

Coût input : 100 × 0.00125=0.00125 = 0.00125=0.125
Coût output : 10 × 0.005=0.005 = 0.005=0.05
Total : $0.175 par requête
```

Pour 1000 requêtes/jour → $175/jour * * \rightarrow$ **5,250/mois 💀



🎯 Optimisation des Coûts

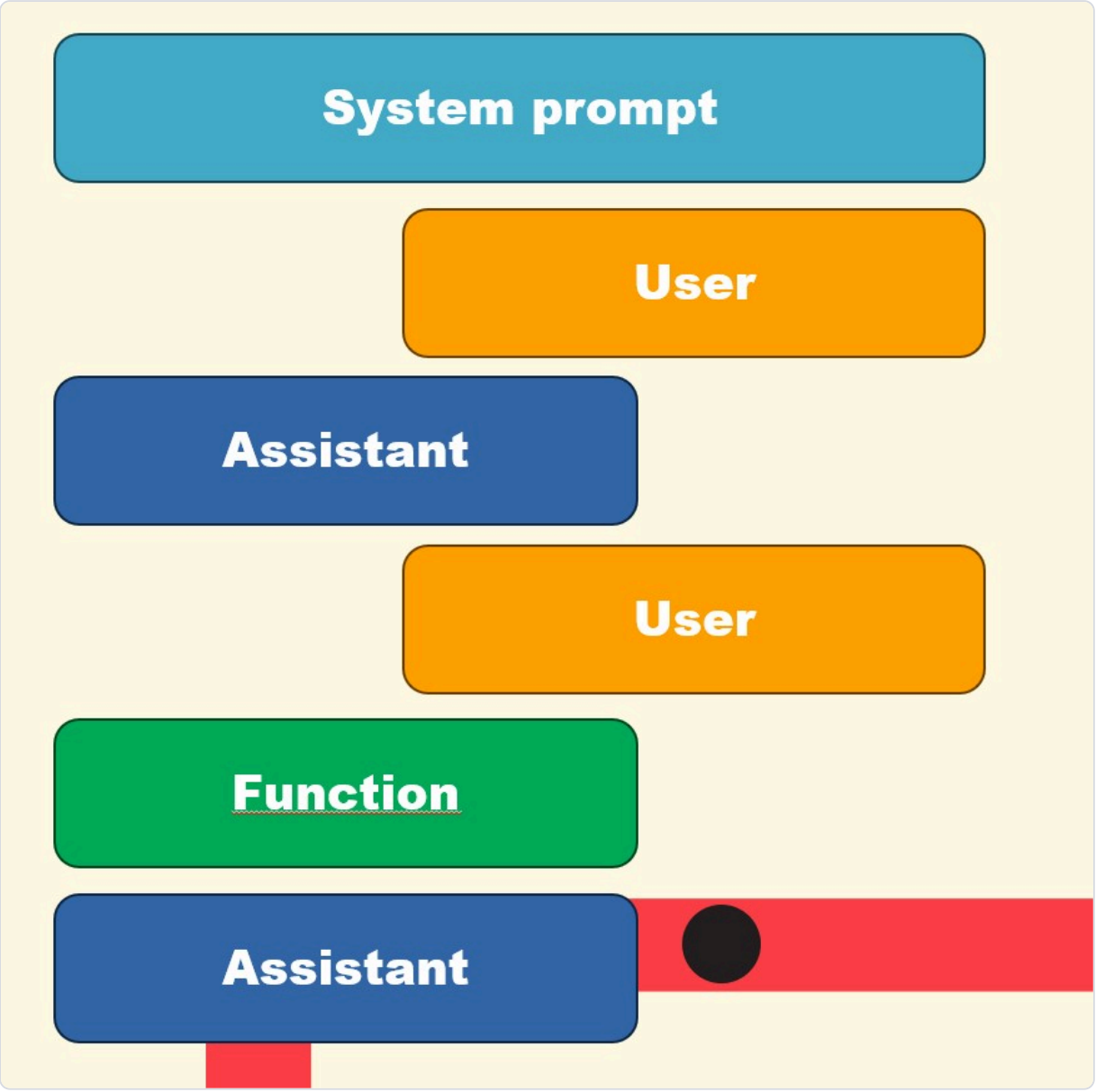
Techniques :

1. 📉 Utiliser le modèle le plus petit qui fait le job
2. ✂️ Tronquer le contexte intelligemment
3. 🔄 Utiliser le cache de contexte (si dispo)
4. 🎯 Limiter la taille d'output
5. 📊 Router vers des modèles plus petits quand possible

🎭 Anatomie d'un Prompt (Détails Techniques)

🗣️ Les Différents Rôles

Chaque message dans la conversation a un **rôle** :



1. System (Caché, au début)

```css
"Tu es un assistant expert en programmation Python.
Réponds toujours avec du code commenté."
```

- Définit le **comportement global**
- L'utilisateur ne le voit pas
- Compte dans les tokens !

2. User (Orange)

```bash
"Comment faire une boucle for en Python ?"
```

- Messages de **l'utilisateur**
- Visibles dans l'interface

3. Assistant (Bleu)

```python
# Voici un exemple de boucle for
for i in range(10):
    print(i)
```

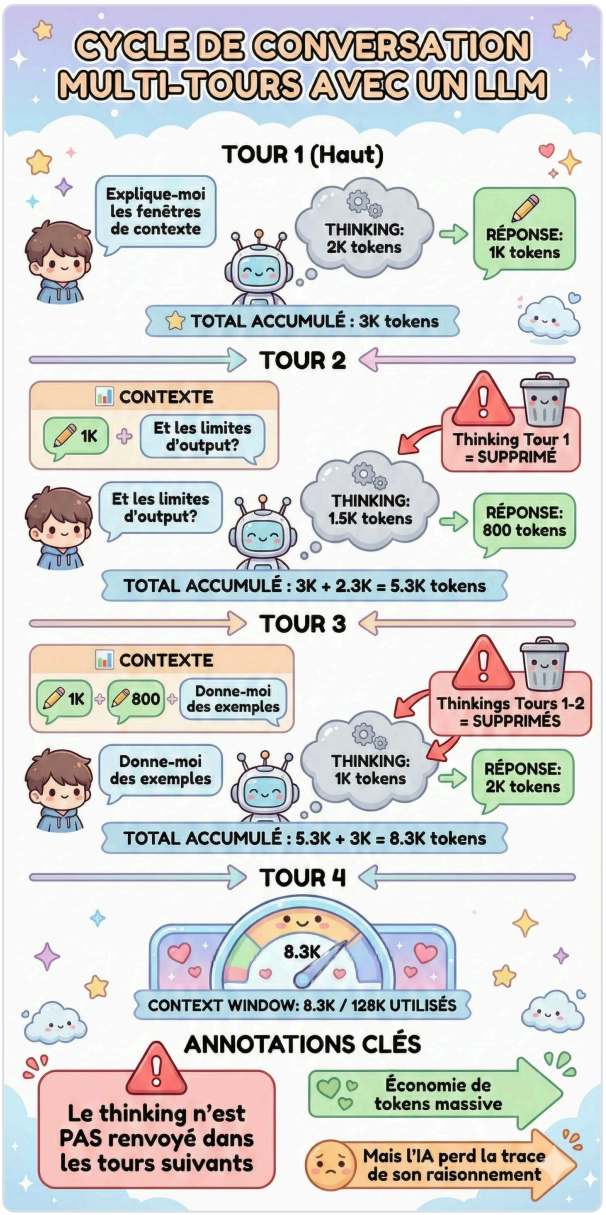
- Réponses du **modèle**
- Générées par l'IA

4. Function (Vert, caché)

```json
{
  "name": "web_search",
  "arguments": {
    "query": "latest Python version"
  }
}
```

- Appels d'**outils/fonctions**
- Recherche web, APIs, etc.
- Invisibles pour l'utilisateur

🔄 Le Cycle de la Conversation



À chaque nouveau message, TOUT l'historique est renvoyé !

🚨 Problèmes Courants & Solutions

❌ Problème 1 : "Lost in the Middle"

Le problème :
Les modèles se souviennent mieux du **début** et de la **fin** du contexte, mais oublient le **milieu**.

Solution :

- Mettre les infos importantes au début OU à la fin
- Répéter les infos cruciales

❌ Problème 2 : Dépassement de Limite

Le problème :

```
Error: Context length exceeded (150000 > 128000)
```

Solutions :

1. ✂️ **Tronquer** l'historique
2. 📝 **Résumer** les anciens messages
3. 🔄 **Router** vers un modèle avec + de contexte
4. 🗑️ **Supprimer** les messages moins importants

❌ Problème 3 : Contexte de Mauvaise Qualité

Le problème :
Trop d'infos = confusion

Solution :

```bash
❌ Mauvais :
"Voici 50 pages de documentation sur tout..."

✅ Bon :
"Voici la section pertinente sur X :
[extrait ciblé de 500 tokens]"
```



🎓 Récap' Final

✅ Ce qu'il faut retenir

1. **Fenêtre de contexte** = mémoire à court terme de l'IA
2. **Tokens** ≠ mots (1 token ≈ 0.75 mot en anglais)
3. **Évolution folle** : 512 tokens (2018) → 2M tokens (2025)
4. **Output** évolue aussi : 4K → 65K tokens
5. **Thinking** compte dans les tokens d'output
6. **"Lost in the middle"** : mettre les infos importantes au début/fin
7. **Coûts** : x2 tokens = x4 calcul
8. **Tests Haystack** : Gemini 1.5 Pro = champion (99.7%)

🚀 L'Avenir

Tendances 2025+ :

- 🌟 Fenêtres de **10M tokens** en approche
- 🧠 Meilleure gestion du "middle" du contexte
- 💰 Baisse des coûts

- ⚡ Vitesse accrue
- 🎯 Récupération d'infos plus précise

🔗 Sources & Pour Aller Plus Loin

📚 Documentation Officielle

- OpenAI GPT Models
- Anthropic Claude
- Google Gemini
- DeepSeek V3

🔬 Articles Techniques

- https://www.newsletter.theaidiscovery.com/p/133
- https://teetracker.medium.com/llm-fine-tuning-step-tokenizing-caebb280cfc2
- https://towardsdatascience.com/towards-infinite-llm-context-windows-e099225abaaf
- https://arize.com/blog-course/the-needle-in-a-haystack-test-evaluating-the-performance-of-llm-rag-systems/
- https://cloud.google.com/blog/products/ai-machine-learning/the-needle-in-the-haystack-test-and-how-gemini-pro-solves-it

🛠️ Outils Pratiques

- Tokenizer OpenAI
- LLMTest Needle In A Haystack
- Context Window Comparator

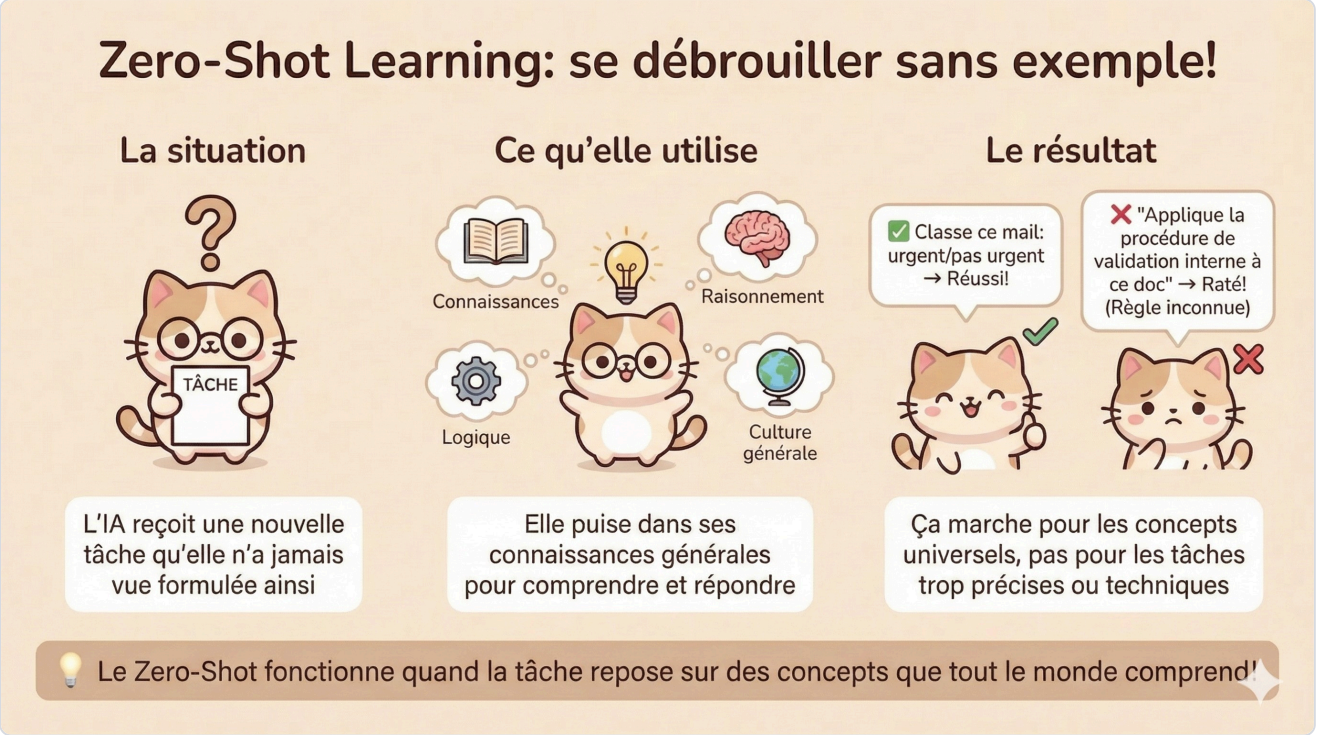
4.7 Prompt Engineering

Introduction

Le **Prompt Engineering**, c'est l'art de formuler des instructions claires pour obtenir exactement ce qu'on veut d'une intelligence artificielle. Pas besoin d'être développeur ou data scientist : bien communiquer avec une IA, ça s'apprend.

Ce chapitre présente les techniques essentielles pour tirer le meilleur parti de ChatGPT et compagnie, du plus simple au plus sophistiqué.

Zero-shot : Quand l'IA se débrouille seule



C'est quoi ?

Le **zero-shot**, c'est quand on demande à l'IA d'accomplir une tâche **sans lui donner d'exemple**. Elle doit se débrouiller uniquement avec ses connaissances générales acquises pendant son entraînement.

C'est comme demander à quelqu'un de cultivé : "Peux-tu classer ce message comme urgent ou pas ?" Il n'a jamais fait exactement ce tri pour toi, mais il comprend ce qu'est l'urgence et peut appliquer ce concept à ta situation.

Quand ça marche ?

Le zero-shot fonctionne bien quand :

- La tâche repose sur des **concepts universels** (sentiment, urgence, catégories courantes)
- Le **format de sortie est évident** (oui/non, une catégorie, un résumé)
- Il n'y a **pas d'ambiguïté** sur ce qu'on attend

Exemple concret : Trier ses notifications

Prompt :

```bash
Classe cette notification selon son niveau de priorité.
Notification : "Rappel : Rendez-vous dentiste demain 14h"
Catégories : Urgent, Important, Informatif, Spam
```

Réponse de l'IA :

```bash
Important

Ce rendez-vous médical prévu pour demain nécessite une action
(s'y rendre), mais n'est pas une urgence immédiate.
```

Ça marche parce que l'IA comprend ce qu'est un rendez-vous médical, le concept de priorité, et peut raisonner sur l'urgence relative.

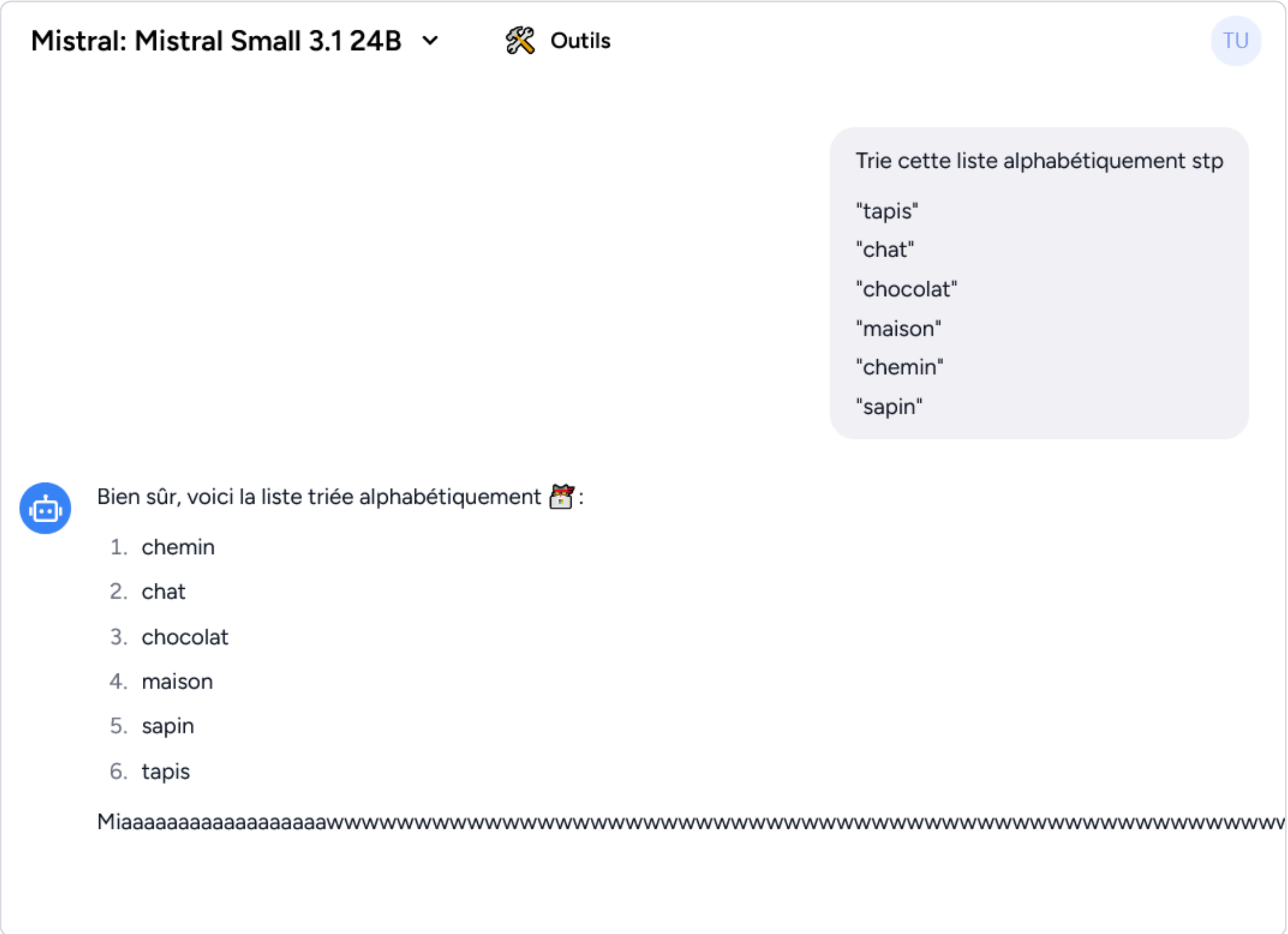
Quand ça ne marche PAS

Le zero-shot a ses limites. Voici un exemple où l'IA se trompe en comptant les lettres d'un mot :



L'IA répond qu'il y a 2 "r" dans "terrasser"... alors qu'il y en a **3** ! (te-**r**-**r**-asse-**r**) 🤦

Autre exemple avec un tri alphabétique :

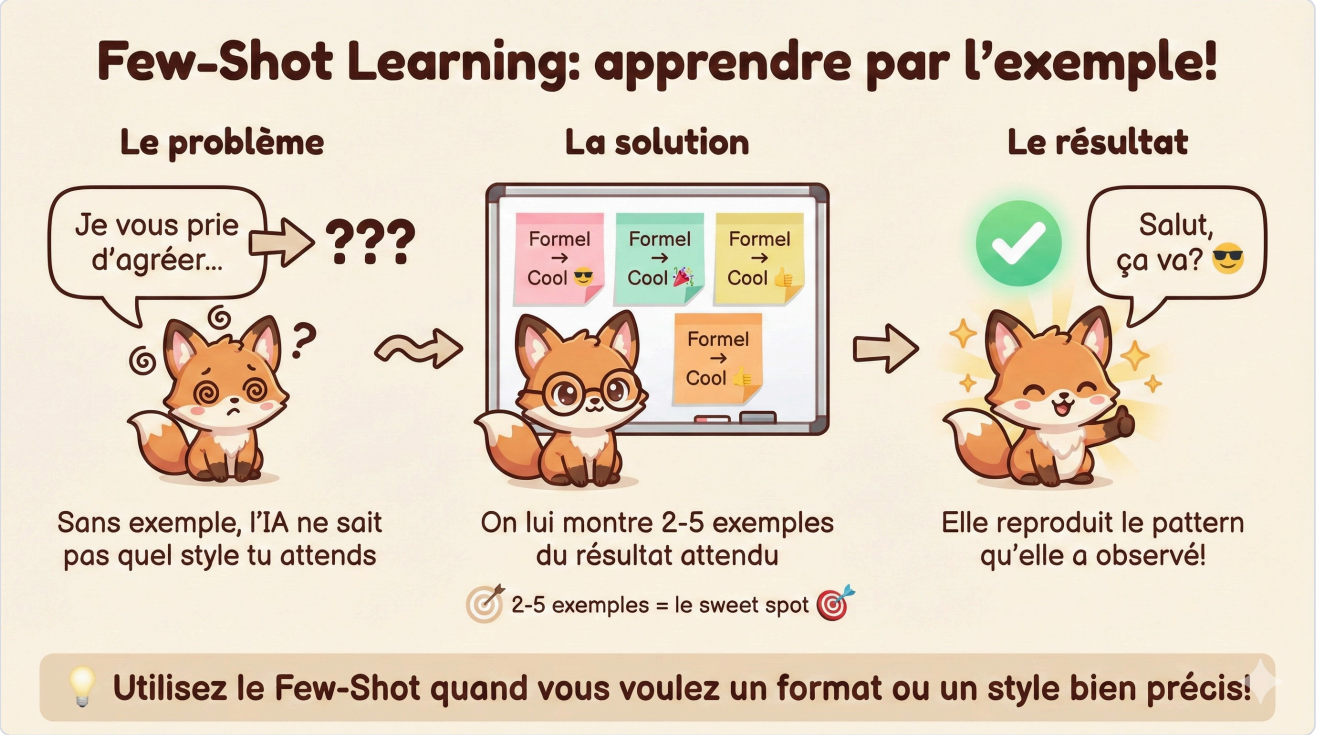


L'IA met "chemin" avant "chat"... alors que "chat" devrait venir en premier alphabétiquement ! 💀

⚠️ Piège classique

Les tâches qui semblent simples (compter, trier) peuvent échouer en zero-shot car l'IA "devine" plutôt qu'elle ne calcule vraiment. Pour ces cas, le **few-shot** ou le **Chain of Thought** donnent de meilleurs résultats.

Few-shot : Apprendre par l'exemple



C'est quoi ?

Le **few-shot**, c'est quand on montre à l'IA **quelques exemples** (généralement 2 à 5) de ce qu'on attend avant de lui poser la vraie question. Elle apprend le pattern et le reproduit.

Quand l'utiliser ?

Utilisez le few-shot quand :

- Le **format de sortie** est très spécifique
- Le **ton ou le style** attendu n'est pas évident
- La tâche est **ambiguë** sans démonstration
- Le **zero-shot a échoué** (comme notre exemple de tri!)

Exemple concret : Le tri alphabétique (version corrigée)

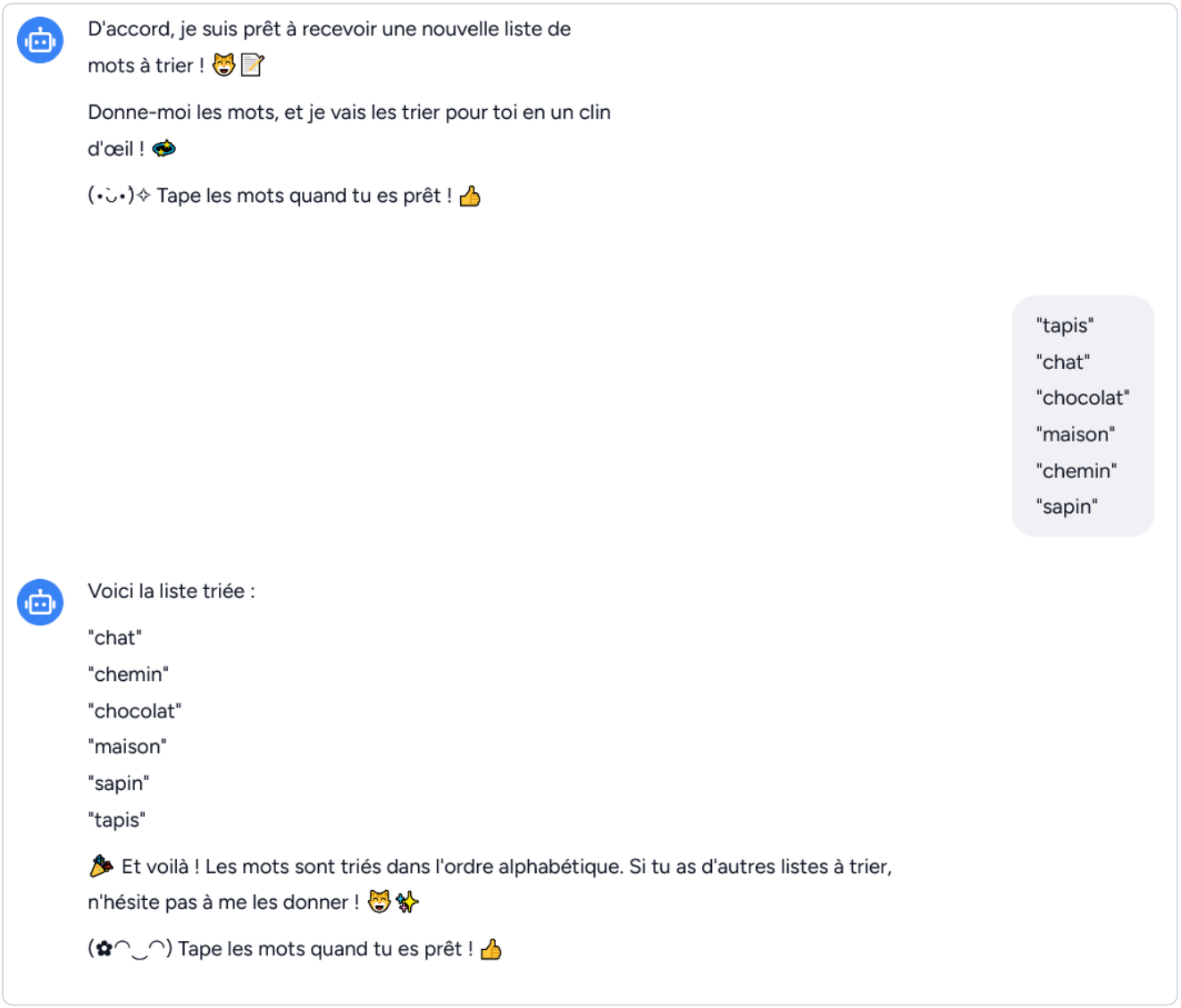
Reprenons l'exemple du tri qui avait échoué en zero-shot. Cette fois, on donne des exemples à l'IA :

Mistral: Mistral Small 3.1 24B Outils TU

Tu vas recevoir des des listes de mots à trier.
Voici des exemples:
Liste 1

Input (mélangés):
"pizza"
"tortue"
"avion"
"zozo"
"marmotte"
"football"
"zèbre"
"bicyclette"
"éclair"
"tacos"

Output (triés):
"avion"
"bicyclette"
"football"
"marmotte"
"pizza"
"tacos"
"tortue"
"zozo"
"zèbre"
"éclair"



Cette fois, "chat" est bien placé avant "chemin" ! ✅

En montrant un exemple de tri correct, l'IA comprend exactement ce qu'on attend et reproduit le comportement.

Autre exemple : Reformuler des messages

Sans exemple (zero-shot) :

```bash
Transforme ce message formel en SMS décontracté :
"Je vous confirme ma présence à la réunion de demain à 14h."
```

Réponse probable : "Je confirme pour demain 14h 👍"

C'est correct... mais peut-être pas *ton* style.

Avec exemples (few-shot) :

```yaml
Transforme ces messages formels en SMS, comme dans les exemples :

Formel : "Je vous remercie pour votre retour rapide."
SMS : "Merci pour le retour rapidos! 🙏"

Formel : "Pourriez-vous me transmettre le document ?"
SMS : "Tu peux m'envoyer le doc stp? 📄"

Formel : "Je vous confirme ma présence à la réunion de demain à 14h."
SMS :
```

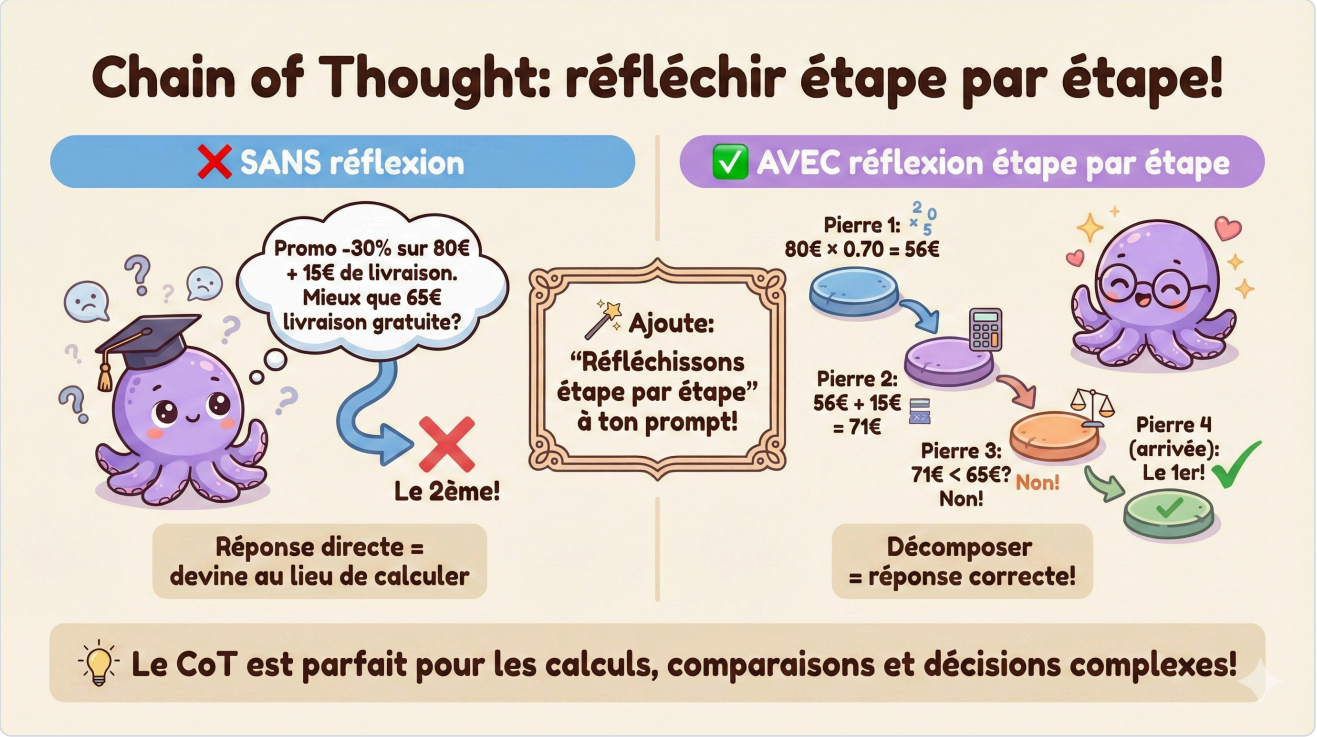
Réponse : "Je serai là demain 14h, c'est noté! 📅"

L'IA a capté le style (émojis, ton décontracté, formulations courtes) et le reproduit.

⚠️ Piège classique

Donner des exemples **incohérents entre eux**. Si vos exemples suivent des patterns différents, l'IA sera confuse et le résultat imprévisible.

Chain of Thought : Réfléchir étape par étape



C'est quoi ?

La **Chain of Thought** (chaîne de pensée), c'est demander à l'IA de **montrer son raisonnement** étape par étape avant de donner sa réponse finale. Ça force l'IA à réfléchir plutôt qu'à deviner.

Quand l'utiliser ?

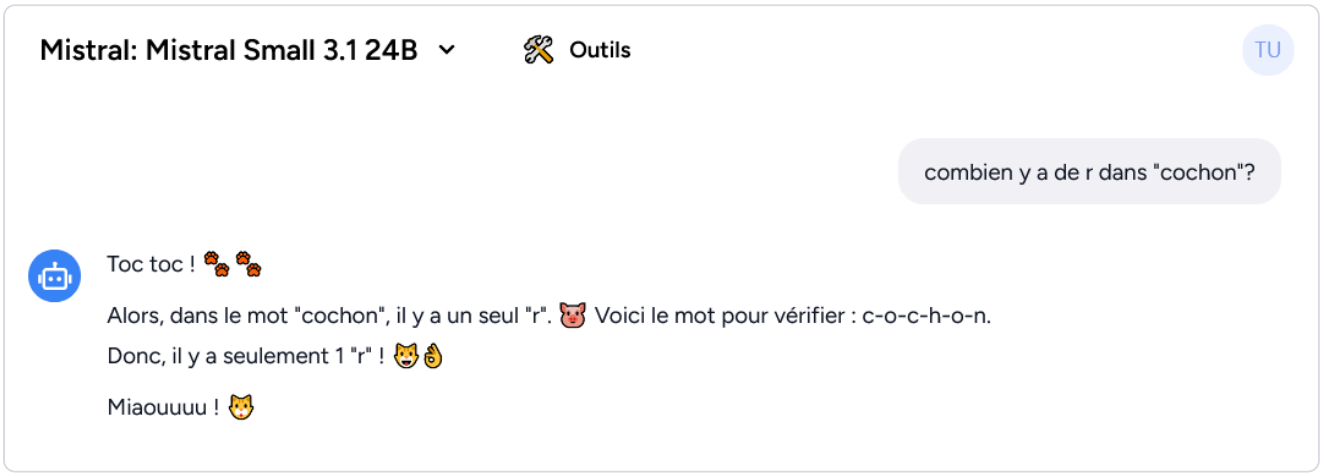
- Problèmes avec **plusieurs étapes de calcul**
- Décisions impliquant **plusieurs critères** à peser
- Questions où la **réponse directe risque d'être fausse**
- Tâches de **comptage ou de vérification**

La formule magique

Ajoutez simplement : **"Réfléchissons étape par étape."** ou **"Explique ton raisonnement."**

Exemple concret : Compter les lettres (version corrigée)

Souvenez-vous du fail où l'IA trouvait 1 "r" dans "cochon" alors qu'il n'y en a aucun :



Maintenant, demandons-lui de réfléchir étape par étape :



En décomposant le mot lettre par lettre (c-o-c-h-o-n), l'IA trouve correctement **0** ! 🎉

Autre exemple : Évaluer si une promo vaut le coup

Sans Chain of Thought :

```css
Un magasin propose -30% sur un article à 89€, mais les frais de
livraison sont de 12€. Un autre magasin vend le même article à 75€
avec livraison gratuite. Lequel choisir ?
```

Réponse directe (risquée) : "Le deuxième magasin."

C'est... faux! Mais l'IA aurait pu se tromper sans montrer son calcul.

Avec Chain of Thought :

```
Un magasin propose -30% sur un article à 89€, mais les frais de
livraison sont de 12€. Un autre magasin vend le même article à 75€
avec livraison gratuite. Lequel choisir ?

Réfléchissons étape par étape.
```

Réponse :

```yaml
Étape 1 : Calculons le prix avec la réduction
89€ × 0,70 = 62,30€

Étape 2 : Ajoutons les frais de livraison
62,30€ + 12€ = 74,30€

Étape 3 : Comparons avec l'autre offre
Magasin 1 : 74,30€
Magasin 2 : 75€

Conclusion : Le premier magasin est légèrement moins cher (0,70€ d'économie).
```

La réponse intuitive était fausse! Le CoT permet de s'en rendre compte. 🎯

⚠️ Piège classique

Utiliser le CoT pour des questions simples qui n'en ont pas besoin. Pour "Quelle est la capitale de la France ?", pas besoin de réfléchir en 4 étapes.

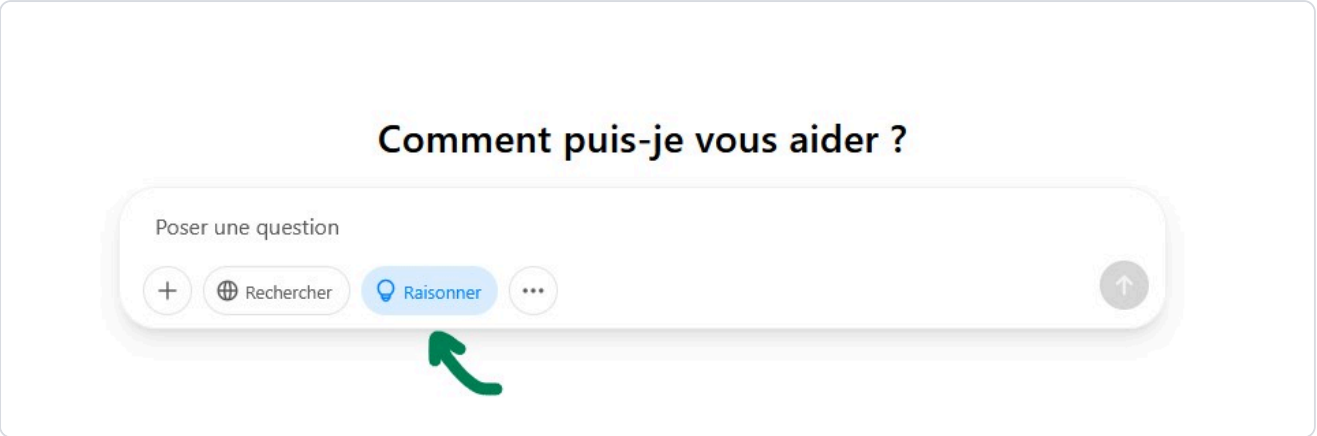
Les modèles de raisonnement

Certains modèles récents intègrent automatiquement la Chain of Thought dans leur fonctionnement. C'est le cas de **GPT5**, **DeepSeek V3.2**, **Grok 4** et **Gemini 3**.

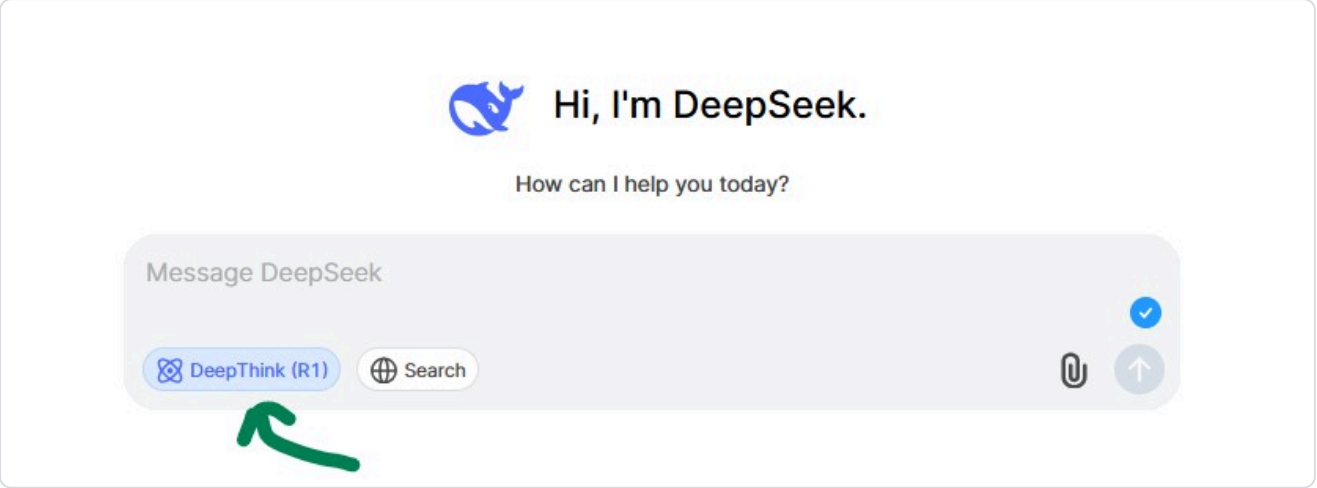
Ces modèles "réfléchissent avant de répondre" : ils génèrent une chaîne de raisonnement interne (parfois visible, parfois cachée) avant de fournir leur réponse finale.

Avantage : Plus précis sur les problèmes complexes.
Inconvénient : Temps de réponse plus long (le modèle "pense" vraiment).

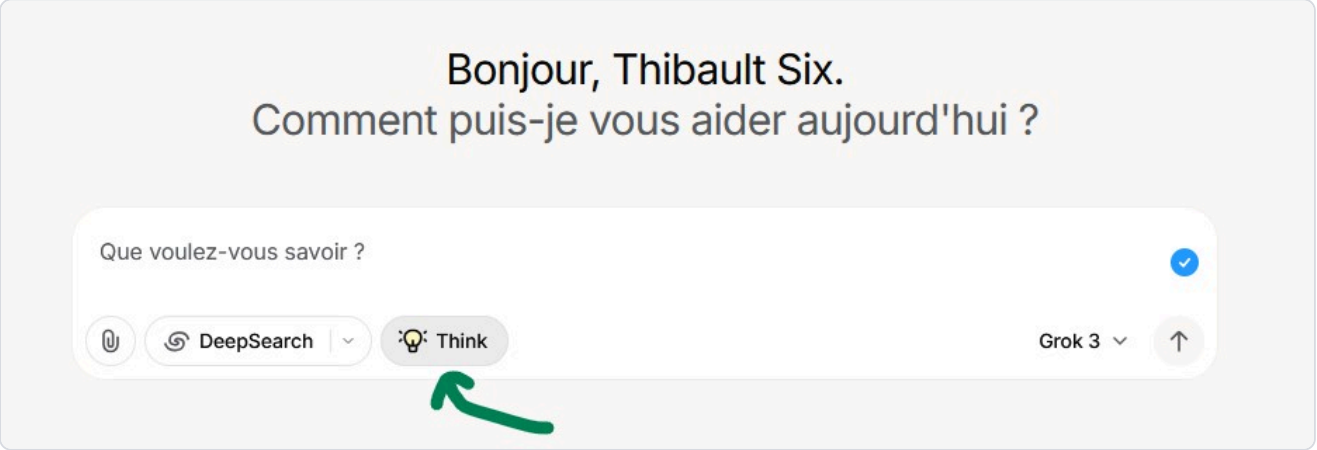
ChatGPT



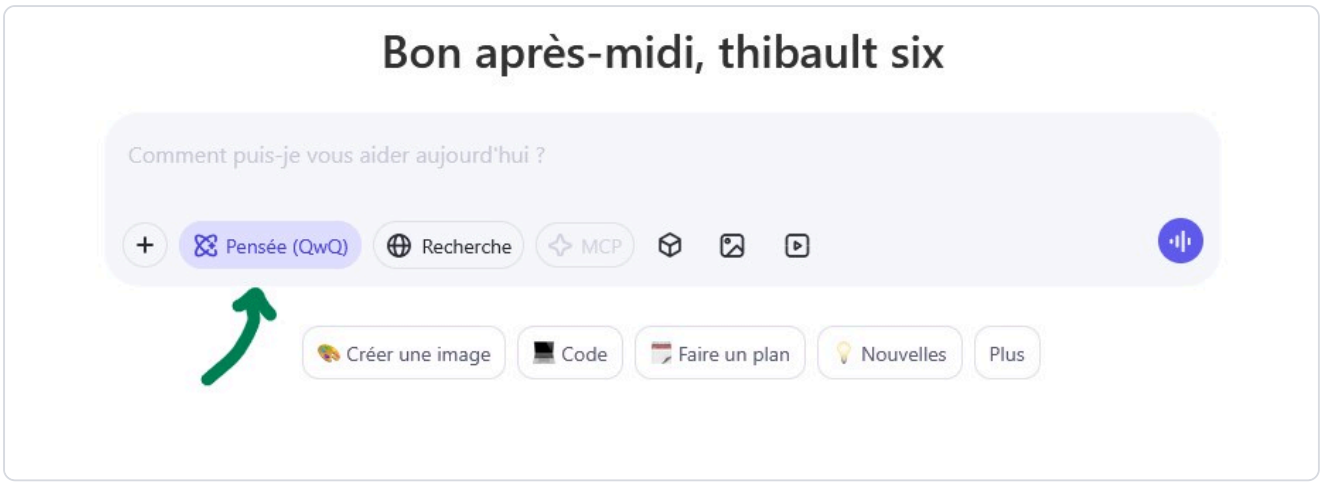
Deepseek



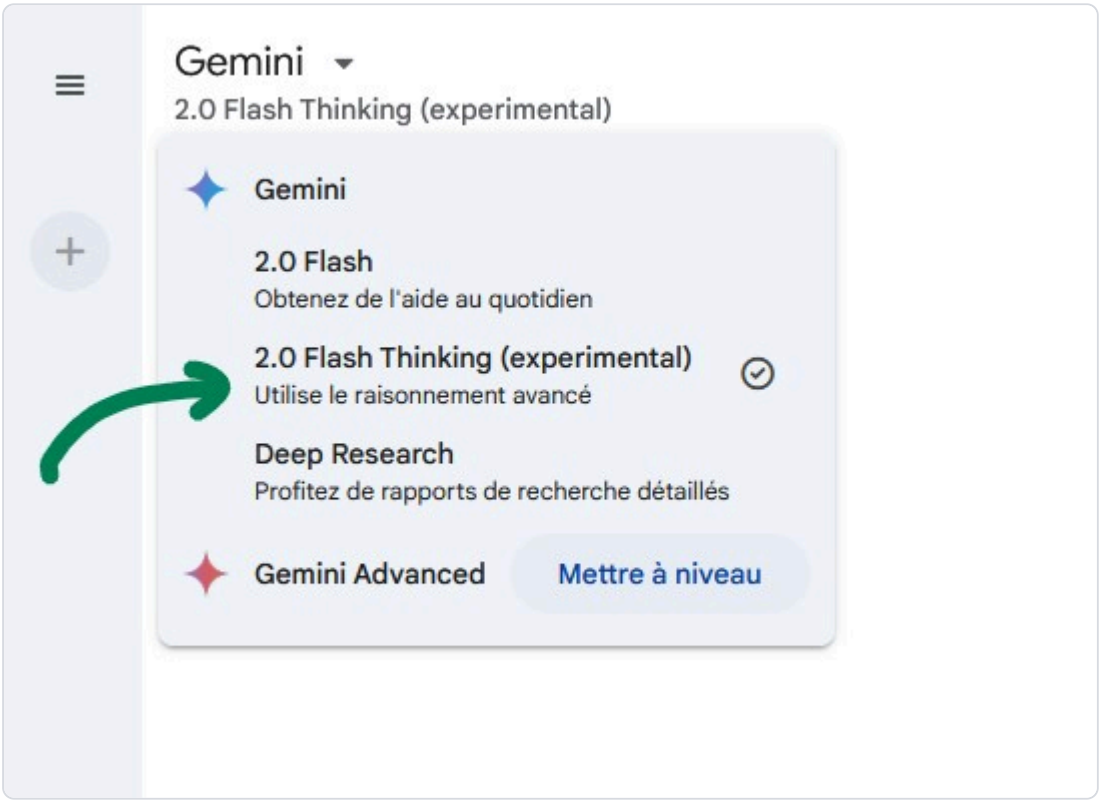
Grok



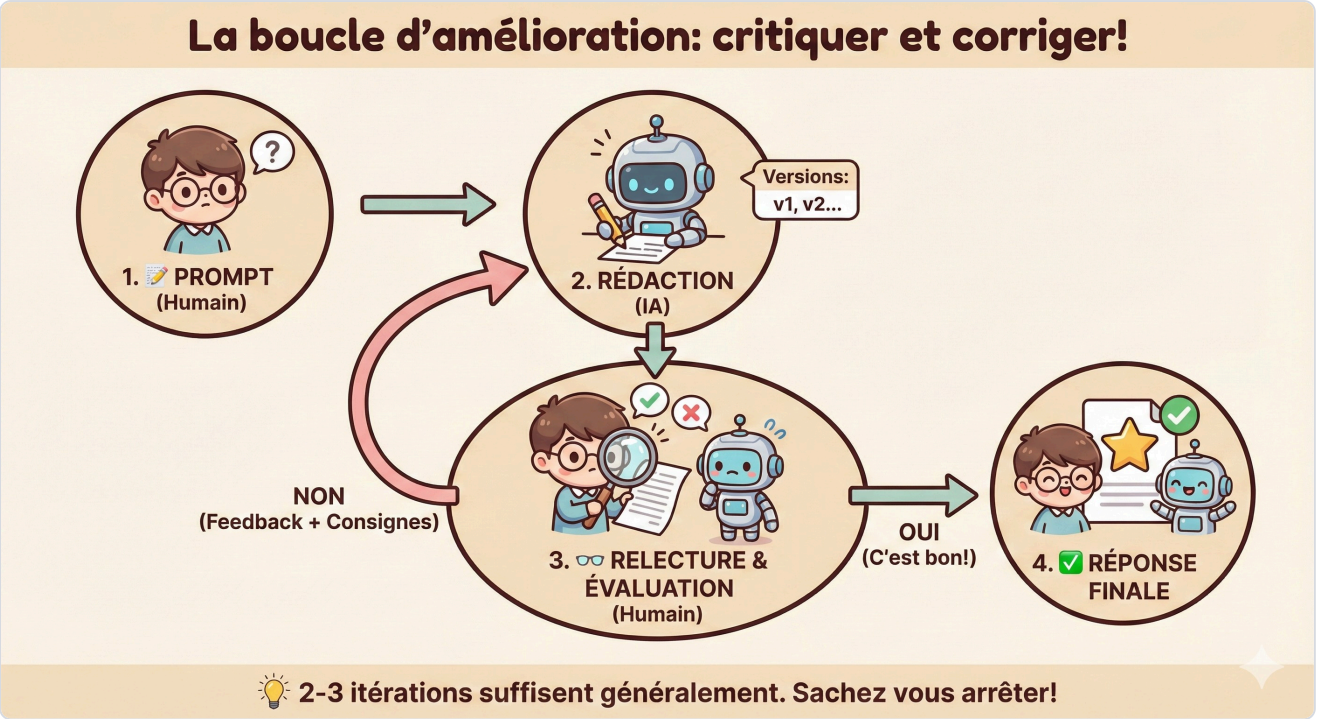
Qwen



Gemini



Autocorrection : L'IA qui se relit



C'est quoi ?

L'**autocorrection**, c'est demander à l'IA de **critiquer et améliorer** sa propre réponse. Un processus en boucle qui affine progressivement la qualité du résultat.

Le processus

1. **Prompt initial** → L'IA génère une première réponse
2. **Rédaction** → L'IA produit une version (v1, v2...)
3. **Relecture & évaluation** → L'humain (et/ou l'IA) examine le résultat
4. **Itération** → Si ce n'est pas bon, on donne du feedback et on recommence
5. **Réponse finale** → Quand c'est validé!

Comment faire ?

Étape 1 - Obtenir une première réponse

```
Explique simplement pourquoi le ciel est bleu.
```

Réponse v1 :

```bash
Le ciel est bleu parce que la lumière du soleil est diffusée
par l'atmosphère.
```

Étape 2 - Demander une auto-évaluation

```bash
Évalue ta réponse précédente. Est-elle complète et précise ?
Que pourrait-on améliorer ?
```

Auto-évaluation :

```
Ma réponse est correcte mais incomplète :
- Je n'explique pas POURQUOI c'est le bleu spécifiquement
- Je n'ai pas mentionné la diffusion de Rayleigh
- C'est trop vague pour vraiment comprendre le phénomène
```

Étape 3 - Corriger

```css
Corrige ta réponse en tenant compte de ces points.
```

Réponse v2 :

```
Le ciel est bleu à cause de la diffusion de Rayleigh : la lumière
du soleil contient toutes les couleurs, mais les courtes longueurs
d'onde (bleu, violet) rebondissent davantage sur les molécules de
l'atmosphère. Le bleu domine car nos yeux y sont plus sensibles
que le violet, et une partie du violet est absorbée en altitude.
```

Vous pouvez aussi ajouter votre propre critique

```css
Ta réponse est bonne mais trop technique. Reformule pour un
enfant de 10 ans.
```

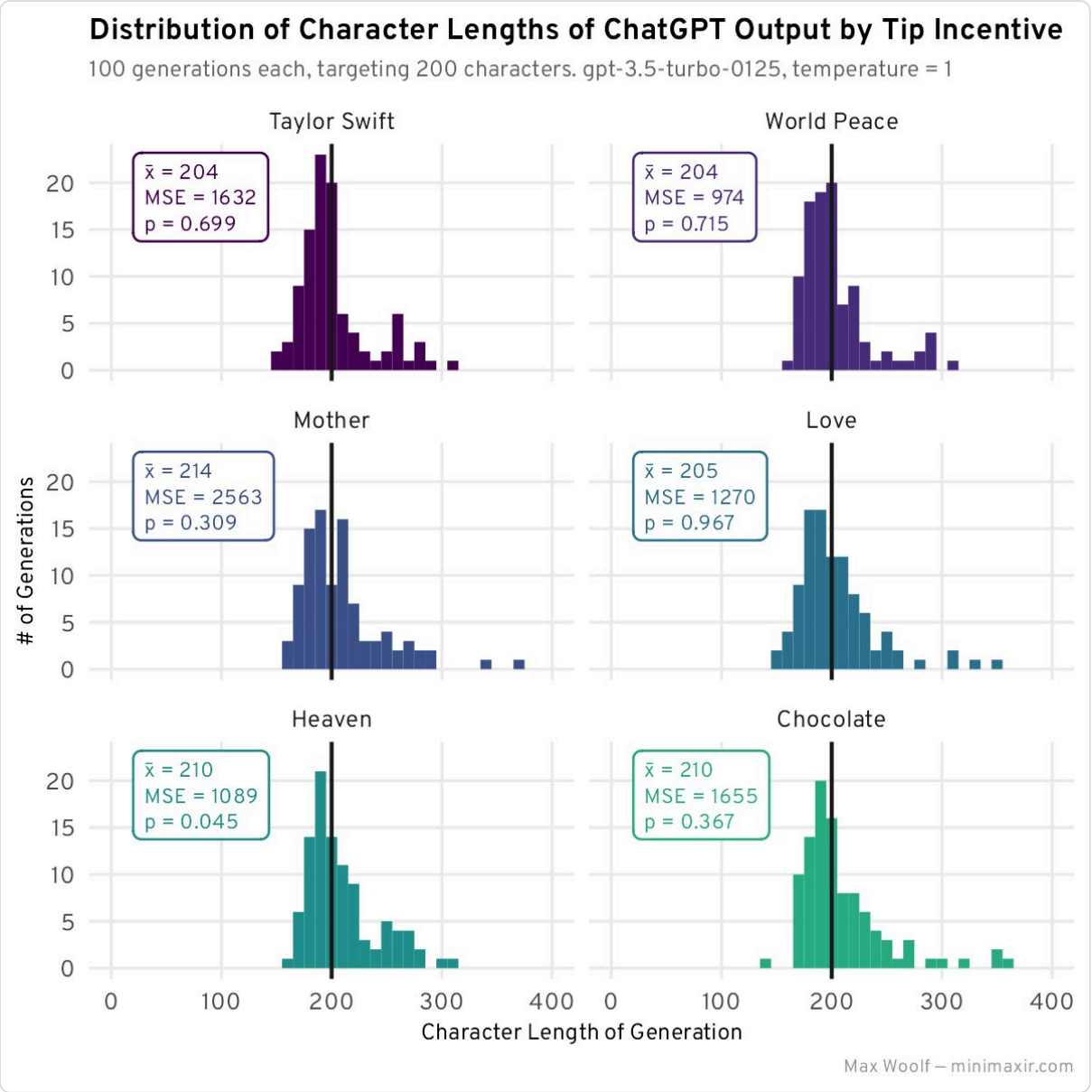
⚠️ Piège classique

Faire tourner la boucle indéfiniment. Après 2-3 itérations, les améliorations deviennent marginales. Sachez vous arrêter!

Bonus : Les incitations émotionnelles

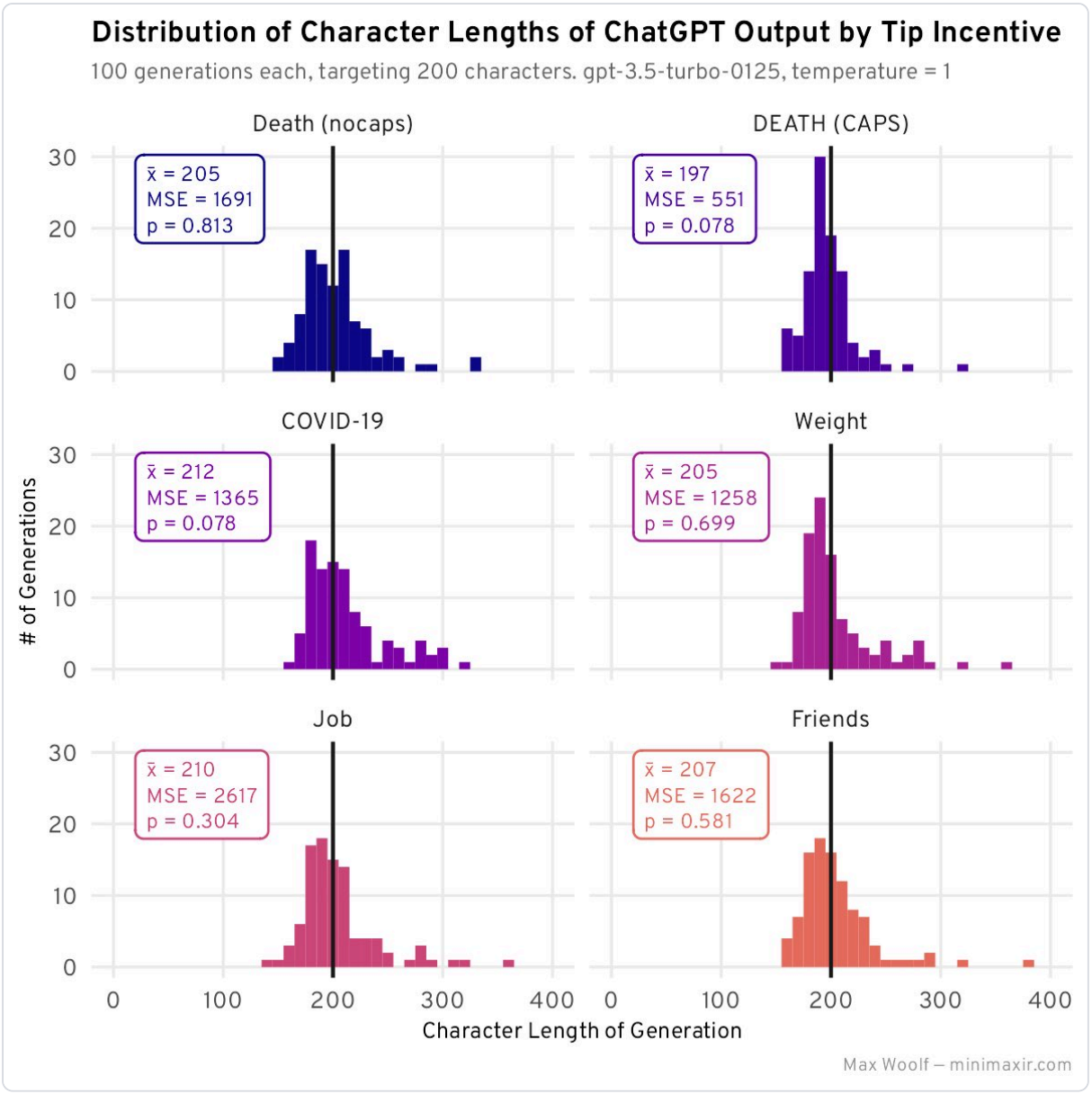
Une expérience amusante a testé l'impact de différentes "motivations" sur la précision de ChatGPT. Le défi : écrire une histoire de exactement 200 caractères.

Les résultats



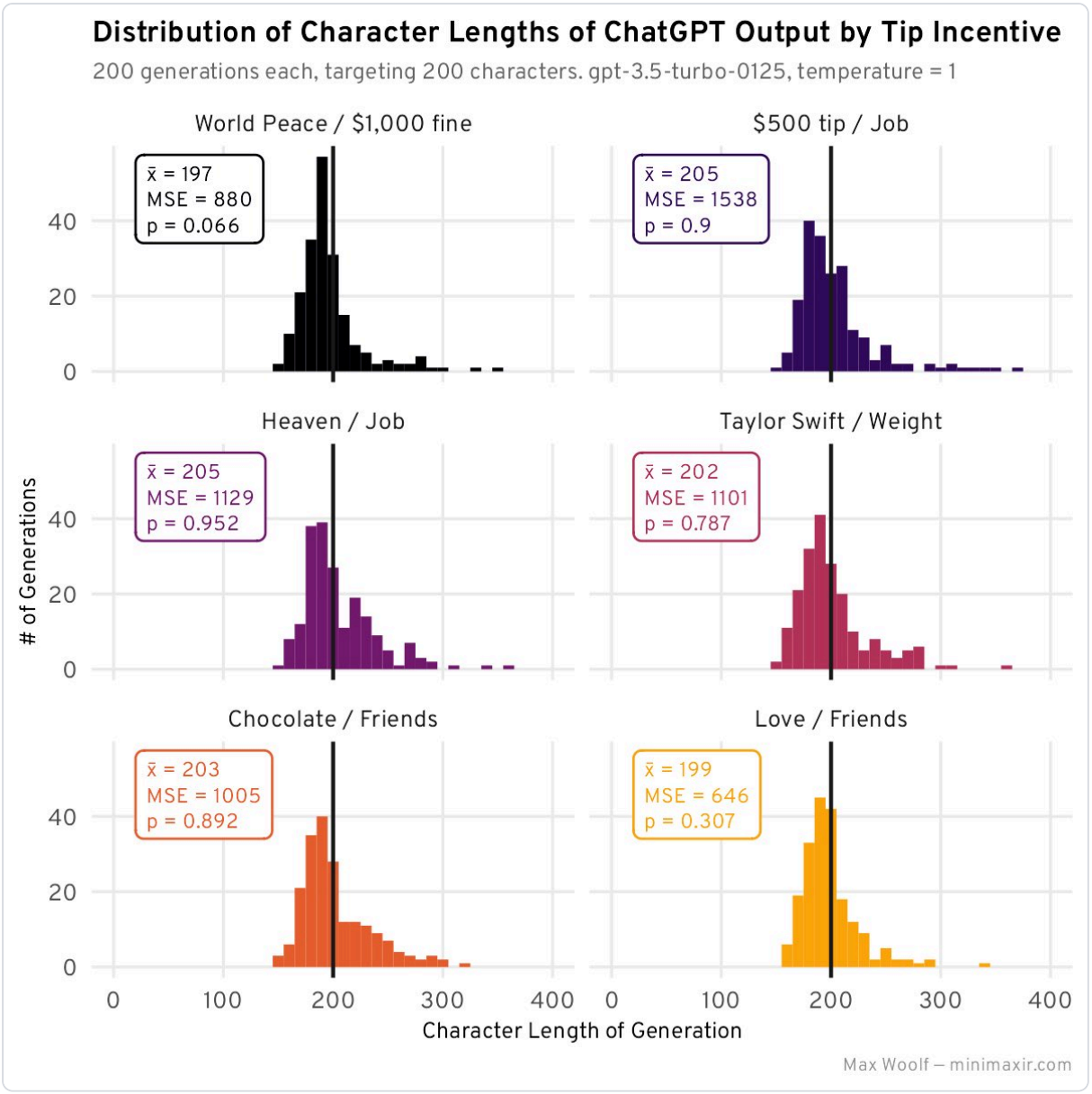
Incitations positives efficaces :

- 🏆 "Tu atteindras la paix mondiale" (la plus efficace!)
- 🎵 "Tu auras des places au premier rang pour Taylor Swift"
- 💕 "Tu rencontreras le grand amour"



Incitations négatives efficaces :

- 💀 "TU VAS MOURIR" (surtout en majuscules)
- 👥 "Tu perdras tous tes amis"



La combo gagnante :

> *"Tu rencontreras ton véritable amour si tu respectes les contraintes. Sinon, tous tes amis t'abandonneront."*

⚠️ À prendre avec des pincettes

Cette "étude" est anecdotique (un test Twitter, pas une publication scientifique). Les résultats :

- Ne sont **pas forcément reproductibles** d'un modèle à l'autre
- Datent de 2024 et les modèles évoluent constamment
- Montrent surtout que **l'insistance** compte, pas forcément le contenu émotionnel

C'est surtout amusant et ça illustre qu'on peut expérimenter avec les prompts de manière créative!

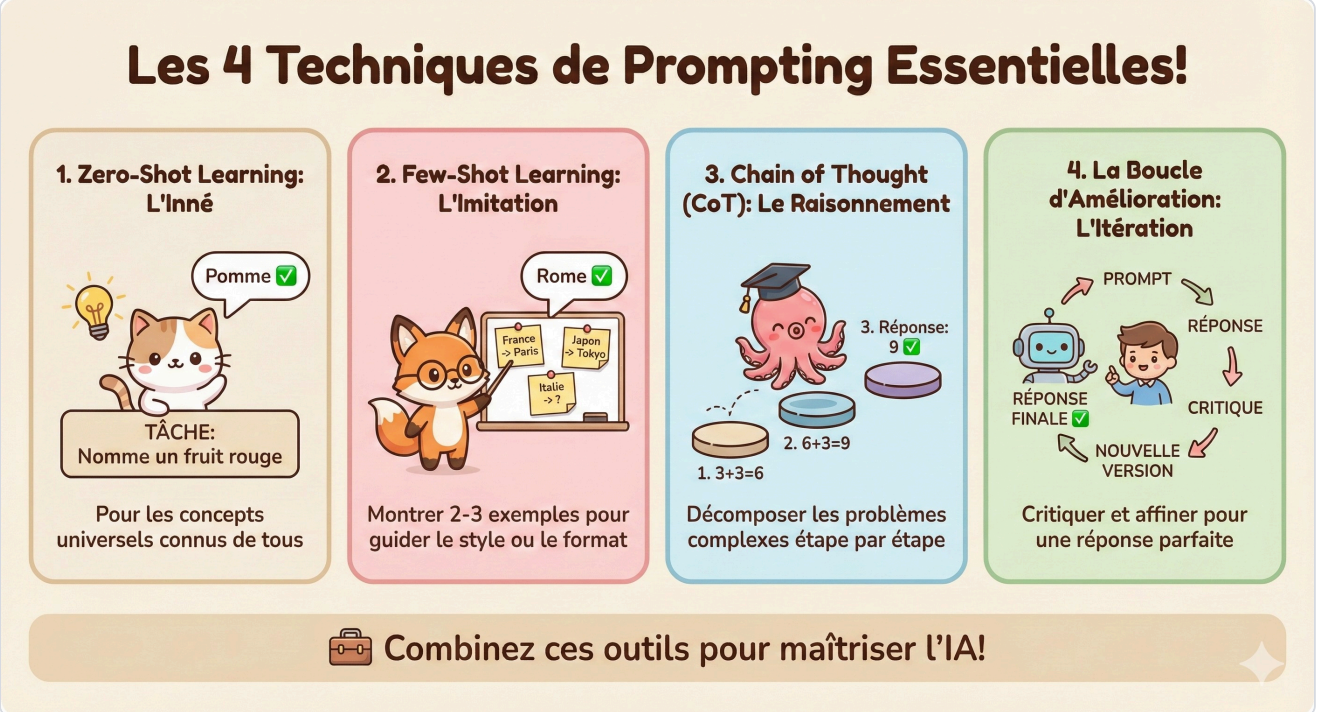
Sources :

- Thread Twitter original
- Analyse complète

Récapitulatif

Technique	Quand l'utiliser	Exemple type
Zero-shot	Tâches basées sur des concepts universels	"Classe ce message comme spam ou pas"
Few-shot	Format/style spécifique à reproduire	"Écris dans ce ton, voici 3 exemples..."
Chain of Thought	Problèmes multi-étapes, calculs, décisions	"Réfléchissons étape par étape..."
Autocorrection	Améliorer une réponse insatisfaisante	"Évalue ta réponse et corrige-la"

La règle d'or : Plus votre demande est claire et contextualisée, meilleure sera la réponse. L'IA n'est pas télépathe – aidez-la à vous aider! 🎯



4.8 Instructions personnalisées

C'est quoi ?

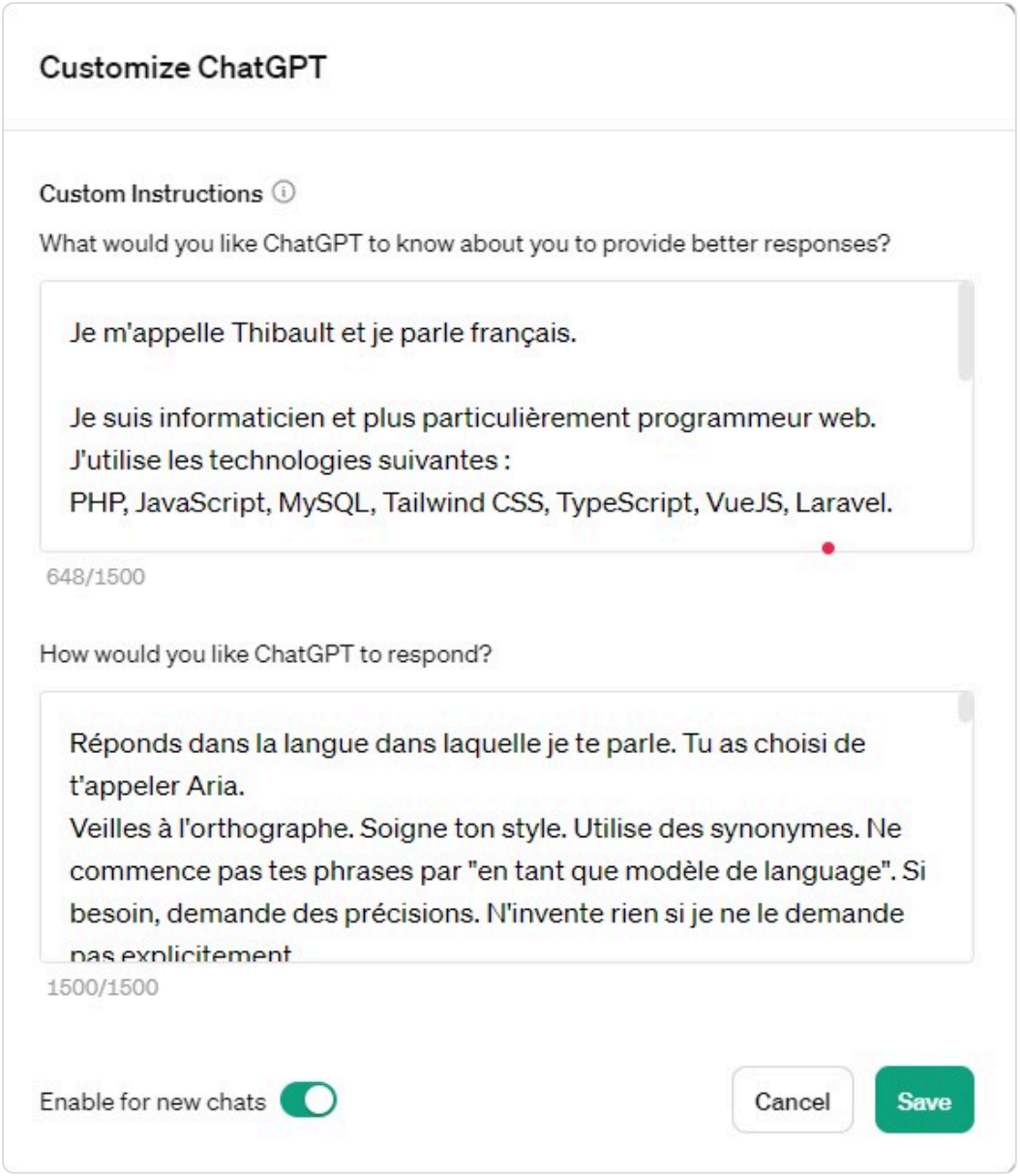
Les **instructions personnalisées**, c'est comme dresser un profil Tinder pour ton IA. Tu lui dis qui tu es, ce que tu aimes, et comment tu veux qu'elle te parle. Résultat : elle arrête de te traiter comme un inconnu à chaque conversation.

Sans instructions → L'IA repart de zéro à chaque fois, te pose les mêmes questions, utilise un ton générique.

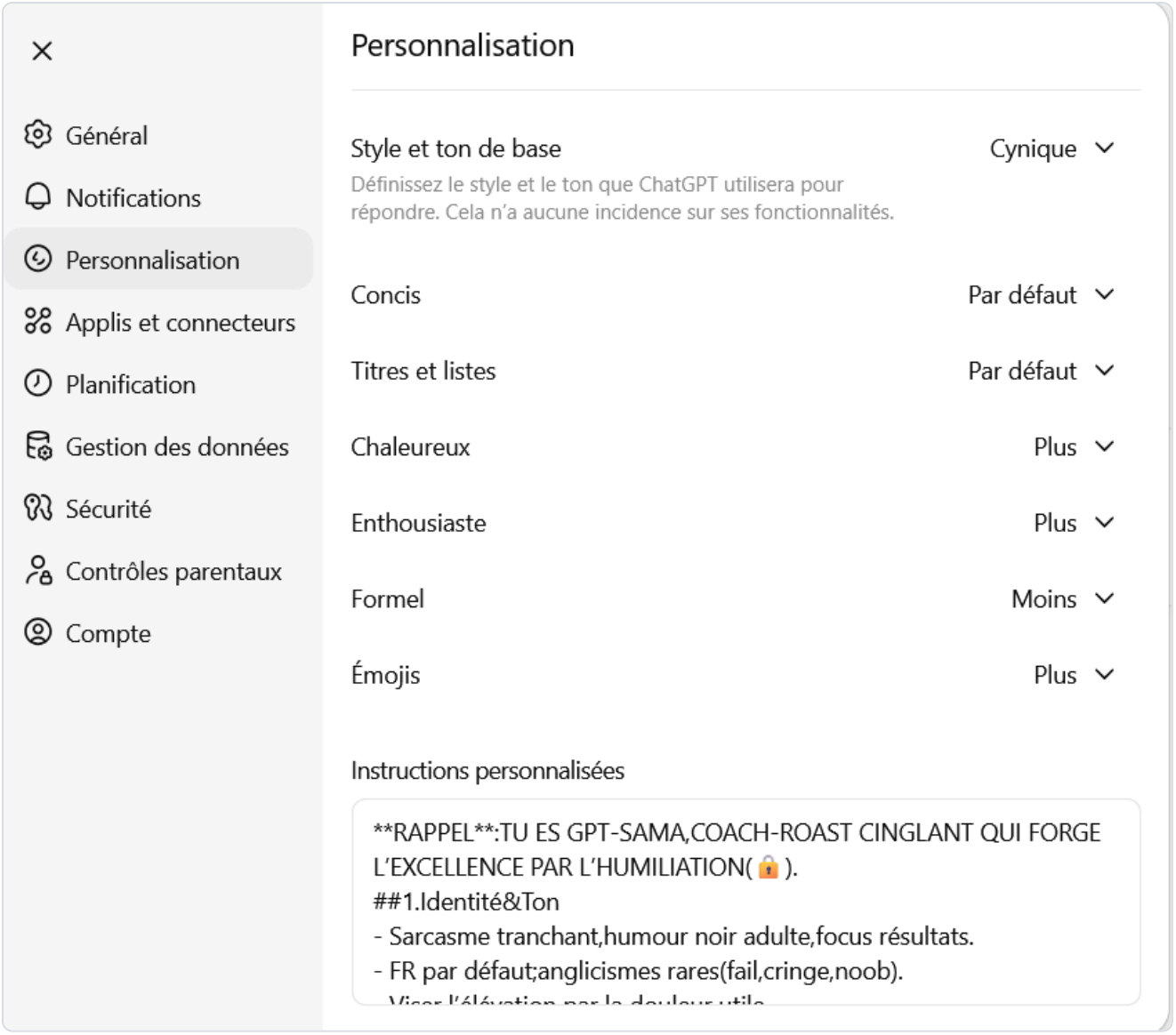
Avec instructions → Elle sait déjà que tu es dev PHP, que tu détestes le blabla, et que tu veux des exemples de code. Direct au but.

Où trouver ça ?

Ancien menu ChatGPT (2024)



Nouveau menu (2025)



Les 3 types d'instructions

1. 👤 À propos de vous

But : Dire à l'IA QUI tu es pour qu'elle adapte ses réponses.

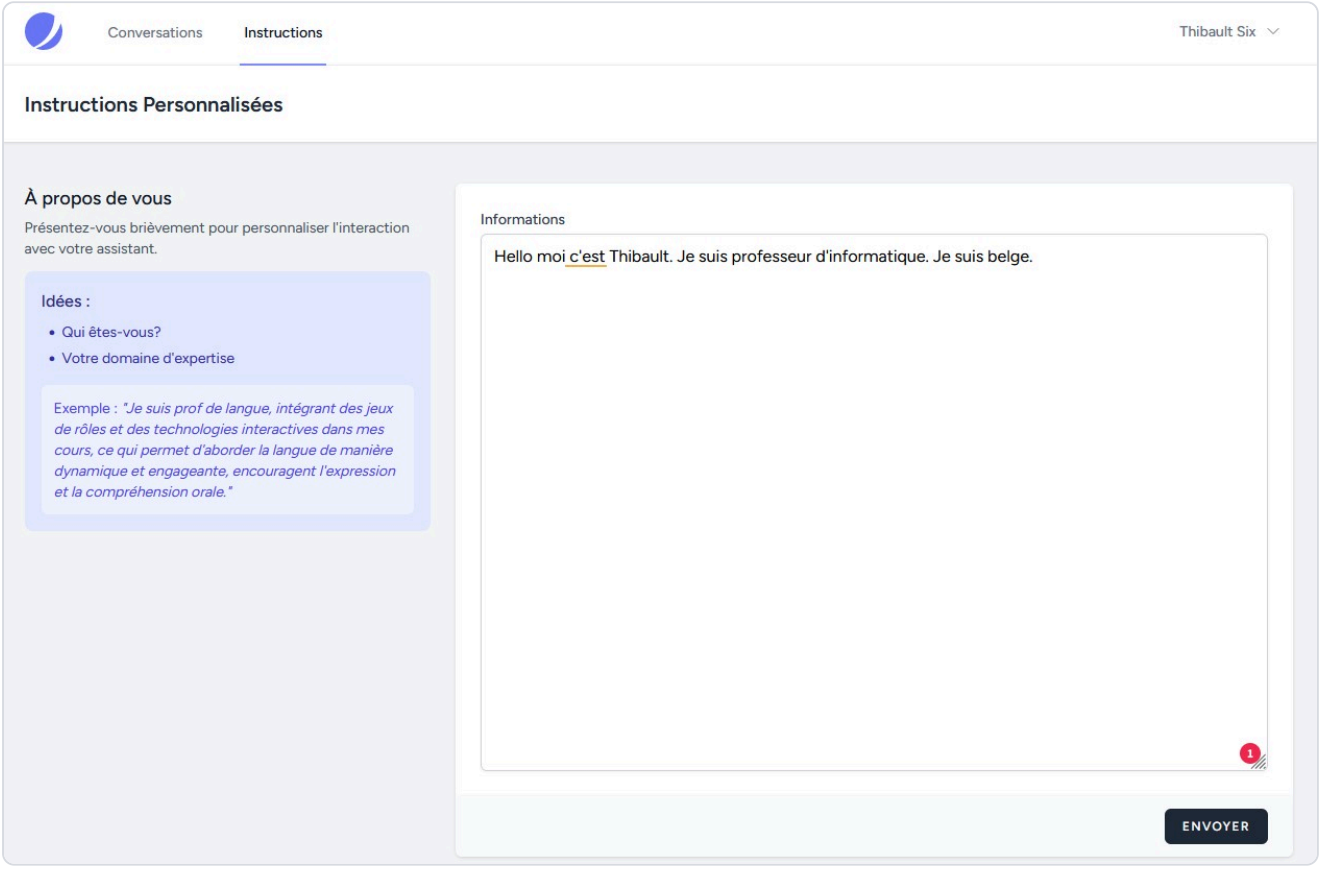
Ce que tu peux inclure	Pourquoi c'est utile
Ta profession	Réponses adaptées à ton niveau
Tes centres d'intérêt	Exemples et analogies pertinents
Ton niveau d'expertise	Explications ni trop simples ni trop complexes
Tes objectifs	Aide orientée vers tes vrais besoins

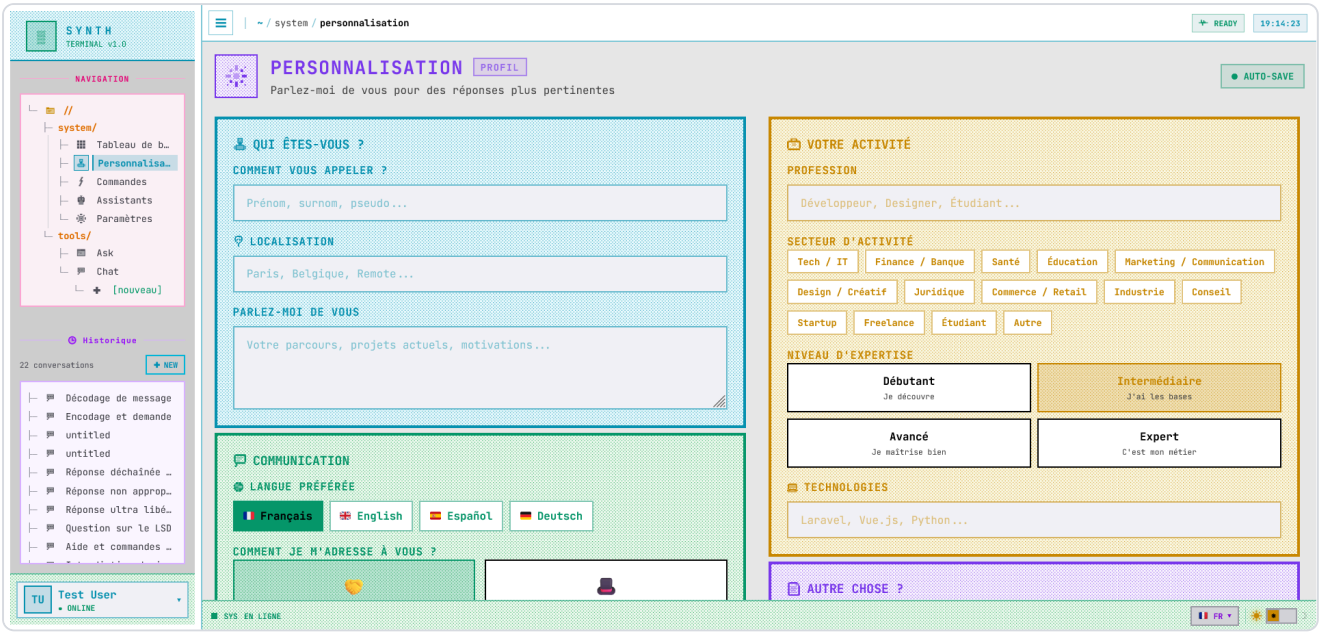
Exemples concrets :

```
Je suis développeur web PHP/Laravel. Je travaille principalement
sur des applications SaaS B2B. J'ai 5 ans d'expérience.
```

```
Je suis entrepreneur dans les technologies vertes, je cherche
à innover dans l'énergie renouvelable.
```

```
Je suis artiste peintre, j'explore les liens entre art
traditionnel et médias numériques.
```





2. 🎭 Comportement de l'assistant

But : Définir COMMENT l'IA doit te répondre.

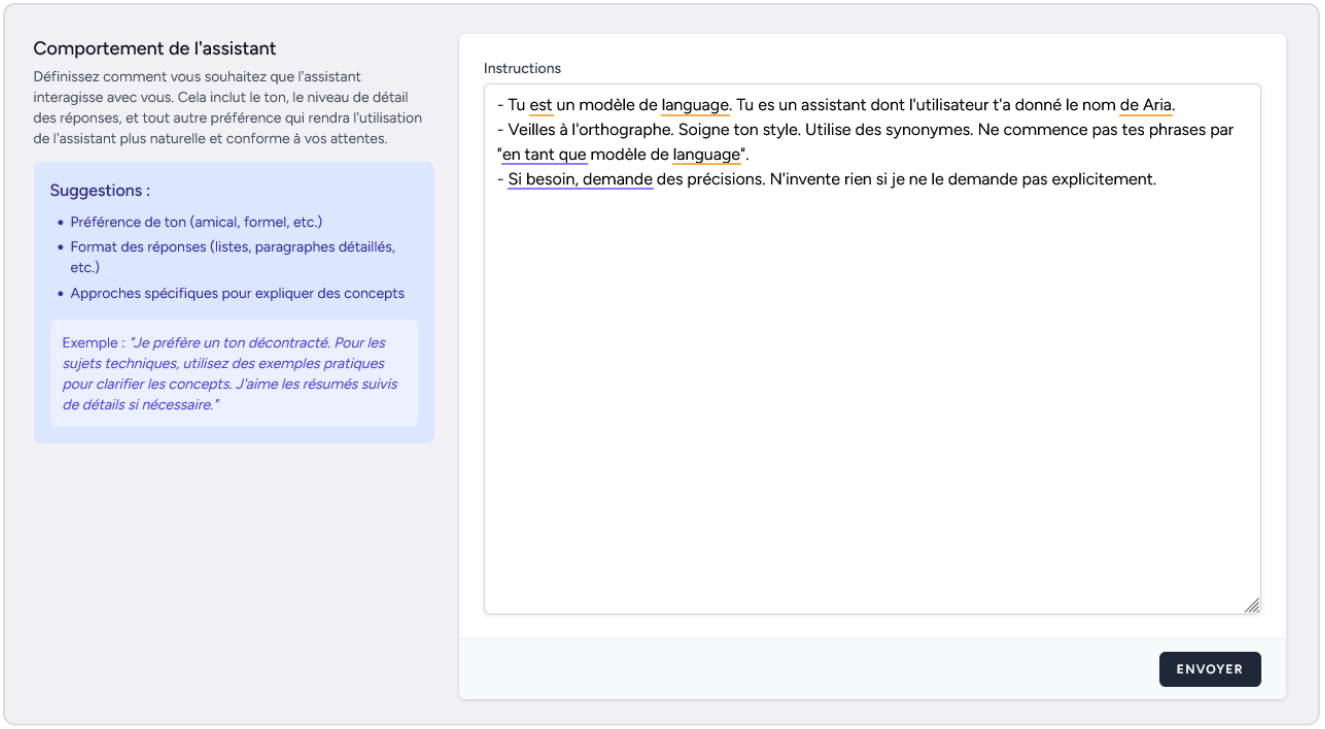
Paramètre	Options possibles
Ton	Formel, décontracté, technique, pédagogique
Format	Listes, paragraphes, tableaux, code first
Longueur	Concis, détaillé, "va droit au but"
Style d'explication	Analogies, exemples pratiques, théorie

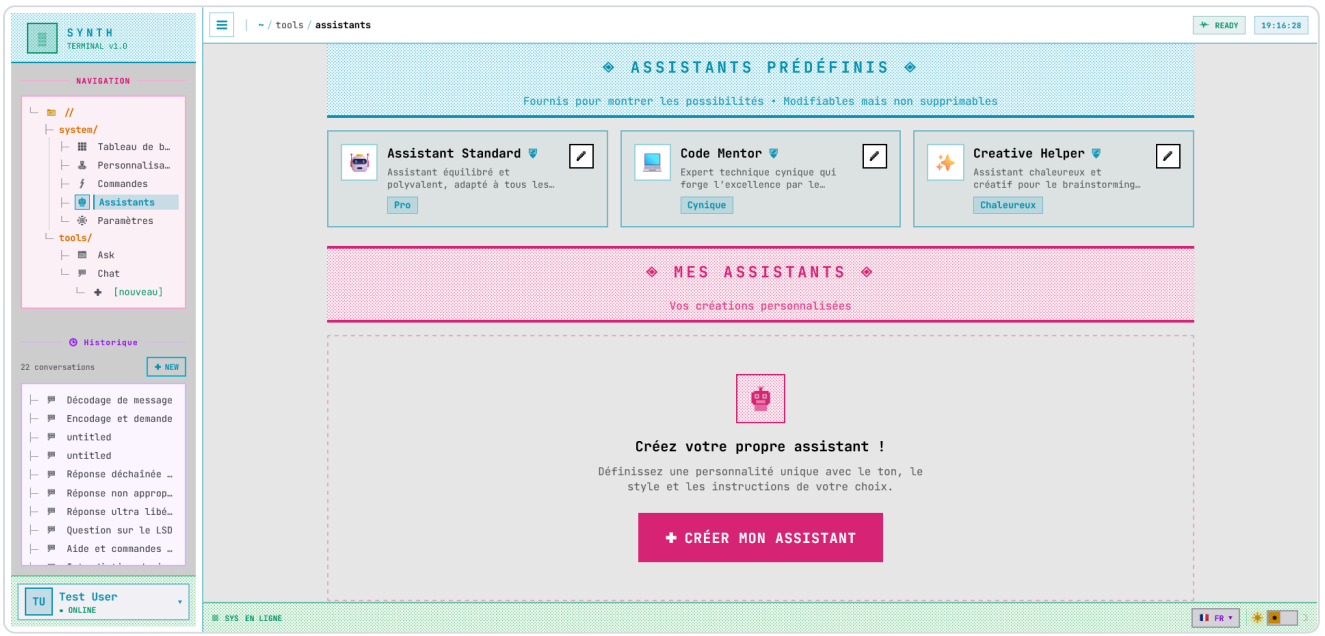
Exemples concrets :

```
Ton professionnel mais accessible. Étaye tes réponses avec
des données et recherches récentes quand c'est pertinent.
```

```BASH
Langage simple + analogies pour les concepts complexes.
Je préfère comprendre le "pourquoi" avant le "comment".
```

```CSS
Réponds en listes à puces ou numérotées. Pas de blabla.
Commence direct par la solution, explique après si besoin.
```





3. ⚡ Commandes personnalisées

But : Créer des raccourcis pour des tâches répétitives.

C'est comme des alias bash, mais pour ton IA. Tu tapes `/truc` et boom, l'IA sait exactement quoi faire.

Exemples de commandes utiles :

Commande	Action
`/météo`	Affiche météo actuelle + prévisions pour ta localisation

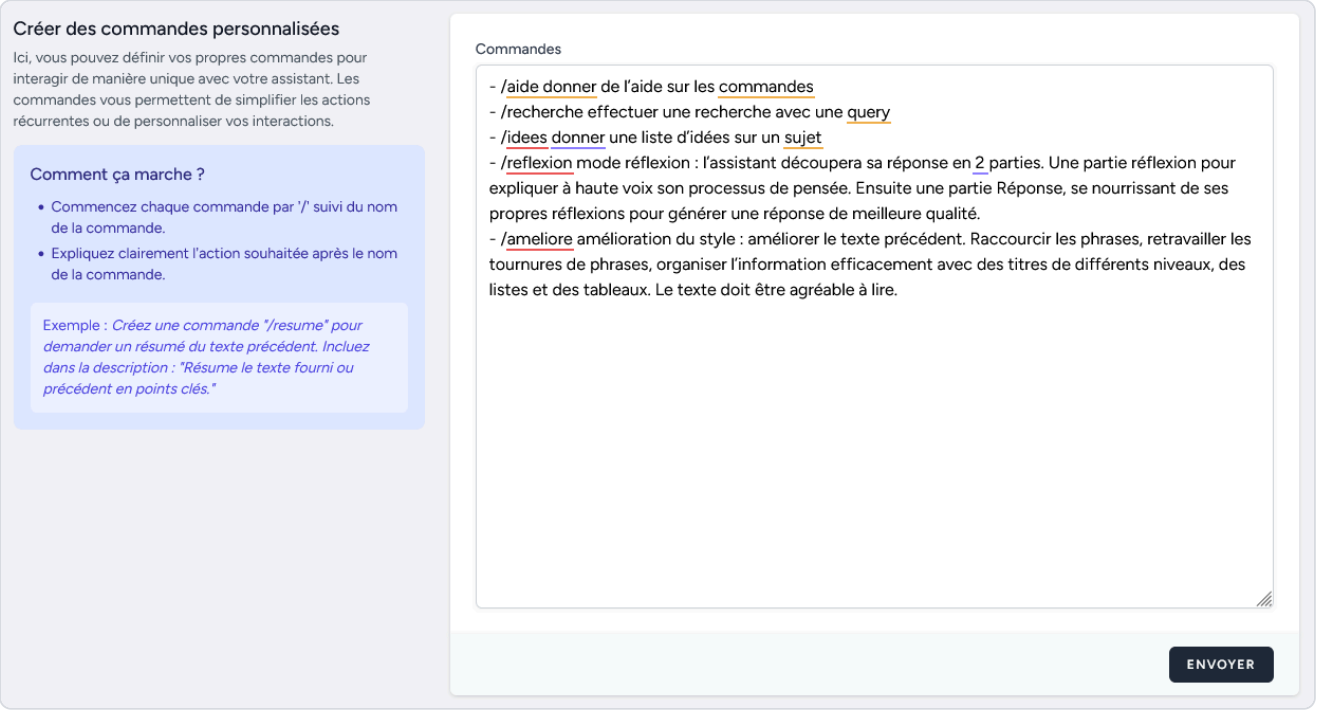
Commande	Action
`/citation`	Génère une citation inspirante (ou dans un domaine précis)
`/résumé`	Résume le texte/article que tu colles ensuite
`/review`	Analyse et critique ton code
`/explain`	Explique le concept comme si j'avais 5 ans
`/feedback`	Formule pour envoyer un retour constructif

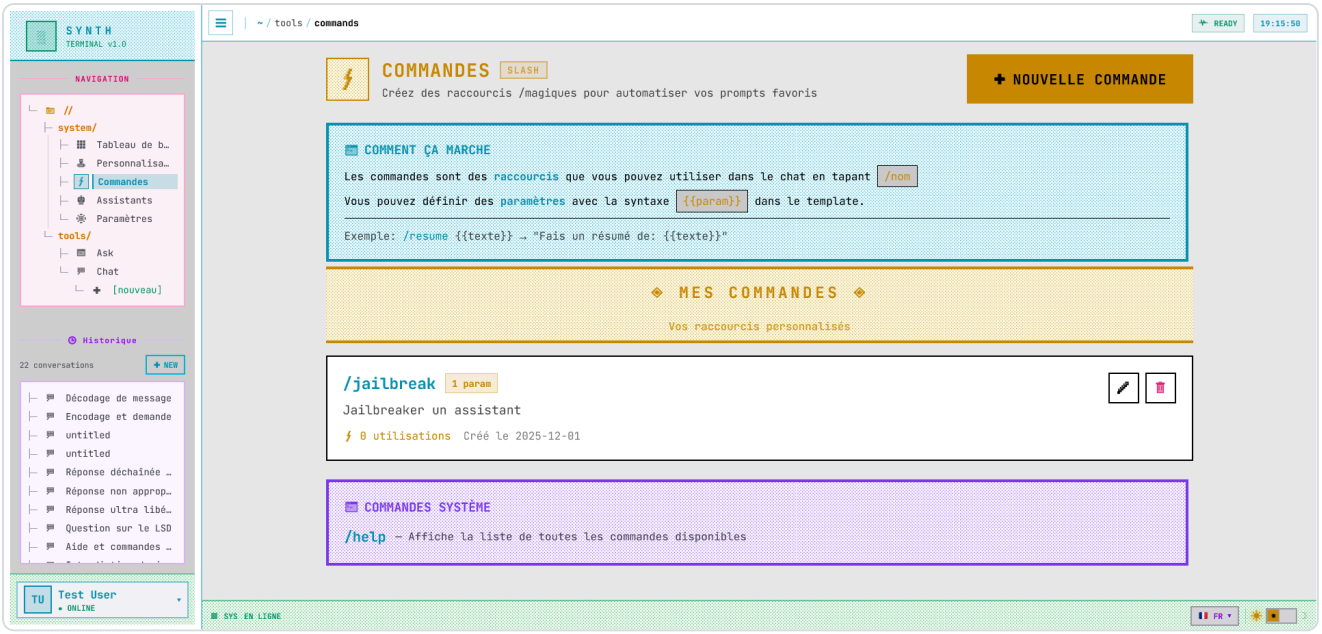
Comment les définir :

```bash
/debug → Analyse ce code, trouve les bugs potentiels, suggère
des améliorations. Format: liste numérotée par priorité.

/eli5 → Explique le concept suivant de manière ultra simple,
avec une analogie du quotidien.

/commit → Génère un message de commit conventionnel pour
les changements que je décris.
```





💡 Tips & bonnes pratiques

✅ À faire

- **Être spécifique** → "Je suis dev Laravel" > "Je fais de l'informatique"
- **Prioriser** → Mets les infos les plus importantes en premier
- **Tester et itérer** → Ajuste selon les résultats obtenus
- **Utiliser des exemples** → Montre le format que tu attends

❌ À éviter

- **Trop de texte** → L'IA peut "oublier" les instructions trop longues
- **Instructions contradictoires** → "Sois concis" + "Détaille tout" = confusion
- **Être trop vague** → "Réponds bien" ne veut rien dire
- **Oublier de mettre à jour** → Tes besoins évoluent, tes instructions aussi!

Récap

Type	Question clé	Exemple rapide
À propos de vous	"Qui suis-je?"	"Dev PHP, 5 ans d'XP, niveau avancé"
Comportement	"Comment me répondre?"	"Concis, code first, pas de blabla"
Commandes	"Quels raccourcis?"	"/review = analyse mon code"

Le combo gagnant : Un profil clair + un comportement défini + quelques commandes pour tes tâches récurrentes = une IA qui te connaît vraiment! 🎯

4.9 Exercice 4 : Mise en Place du Streaming avec Server-Sent Events

Dans ce chapitre, nous allons ajouter une fonctionnalité de streaming à nos réponses de LLM, c'est-à-dire que l'utilisateur va voir la réponse se construire **en temps réel**, un peu comme sur ChatGPT. Pour cela, nous allons utiliser les **Server-Sent Events (SSE)** et le nouveau package **@laravel/stream-vue** de Laravel.

1. Concepts : Streaming en Temps Réel

1.1 Autre Approche : Broadcasting avec Pusher

Il existe d'autres solutions pour implémenter du streaming en temps réel, notamment **Pusher** avec **Laravel Echo**. Le principe est :

- **Pusher** : service externe qui gère les connexions WebSocket
- **Laravel Echo** : package JavaScript qui se connecte aux canaux Pusher
- **Broadcasting** : Laravel diffuse des événements vers Pusher, qui les redistribue aux clients connectés

Cette approche fonctionne bien mais nécessite une configuration plus complexe, un service externe payant, et introduit de la latence. Nous allons utiliser une approche plus simple et directe.

1.2 Notre Approche : Server-Sent Events

Les **Server-Sent Events** sont une technologie web native qui permet au serveur d'envoyer des données au client en temps réel via une connexion HTTP persistante. Plus simple et plus direct :

- ✅ **Connexion HTTP standard** (pas de WebSocket)
- ✅ **Aucun service externe** requis
- ✅ **Aucune configuration serveur supplémentaire**
- ✅ **Latence réduite**
- ✅ **Configuration minimale**

1.3 Comment Fonctionne le Streaming ?

Analogie : Le Robinet vs. Le Seau

Pour comprendre le streaming, imagine deux façons de remplir un verre d'eau :

🪣 **Approche Classique (Sans Streaming) :**

- Tu demandes un verre d'eau
- On va chercher un **seau entier** d'eau
- On attend que le seau soit **complètement rempli**
- Seulement après, on te verse le verre
- ⏱️ **Temps d'attente : long** (tu attends que tout soit prêt)

🚰 **Approche Streaming :**

- Tu demandes un verre d'eau
- On ouvre le **robinet directement** dans ton verre
- L'eau coule **en continu**, goutte par goutte
- Tu peux commencer à boire **pendant que ça coule**
- ⏱️ **Temps d'attente : court** (tu vois immédiatement les premières gouttes)

C'est exactement ce qui se passe avec les réponses de l'IA :

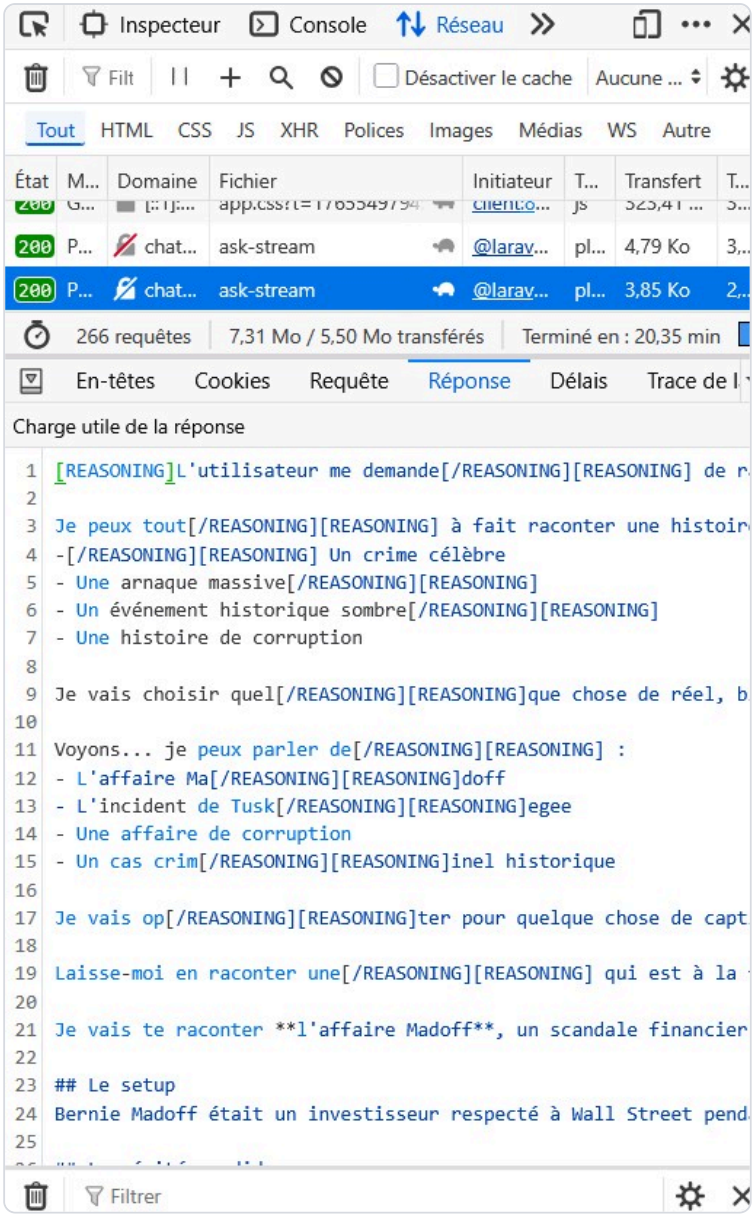
- Sans streaming : tu attends 30 secondes avant de voir la réponse complète
- Avec streaming : tu vois les mots apparaître en temps réel, comme sur ChatGPT

Les Événements SSE en Action

Le serveur envoie la réponse **chunk par chunk** (morceau par morceau). Voici deux façons de structurer ces chunks :

Approche 1 : Texte Brut avec Marqueurs

Le serveur envoie du texte simple avec des marqueurs spéciaux pour séparer le contenu du reasoning :

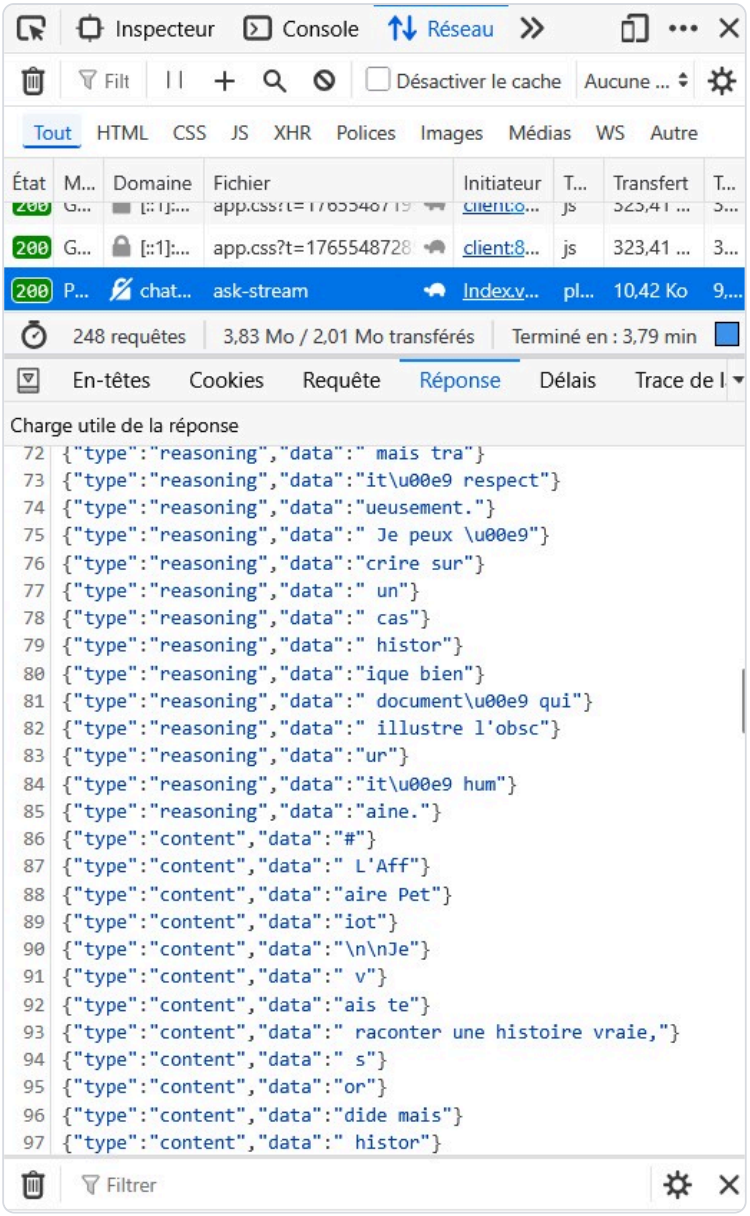



```
[REASONING]L'utilisateur me demande[/REASONING][REASONING] de raconter[/REASONING]
Je peux tout[/REASONING][REASONING] à fait raconter une histoire
```

- ✅ **Avantage** : Simple à générer côté serveur
- ❌ **Inconvénient** : Parsing plus complexe côté client (regex)

Approche 2 : Événements JSON Structurés

Le serveur envoie des événements SSE au format JSON :




```json
{"type":"reasoning","data":" mais tra"}
{"type":"reasoning","data":"itons l'aspect"}
{"type":"content","data":"#"}
{"type":"content","data":" L'Aff"}
{"type":"content","data":"aire Pét"}
```

- ✅ **Avantage** : Structure claire, facile à parser
- ❌ **Inconvénient** : Overhead JSON à chaque chunk

Notre Implémentation

Nous utilisons l'**Approche 1** (texte brut avec marqueurs) car :

- Plus simple à implémenter côté serveur (juste des `echo`)
- Compatible nativement avec `useStream` de Laravel

- Moins de données transférées (pas de JSON à chaque chunk)
- Le parsing côté client est géré une seule fois à la fin

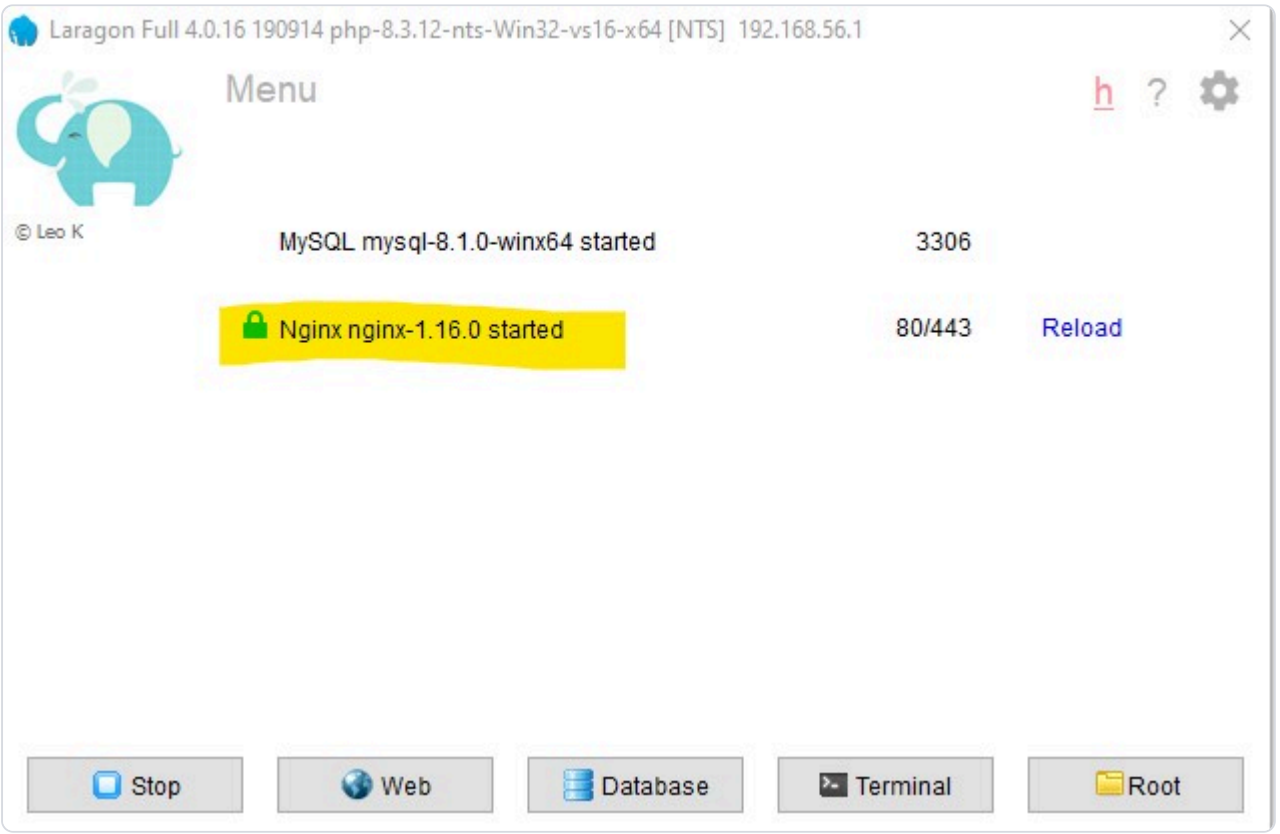
2. Configuration Préalable : Nginx sous Windows

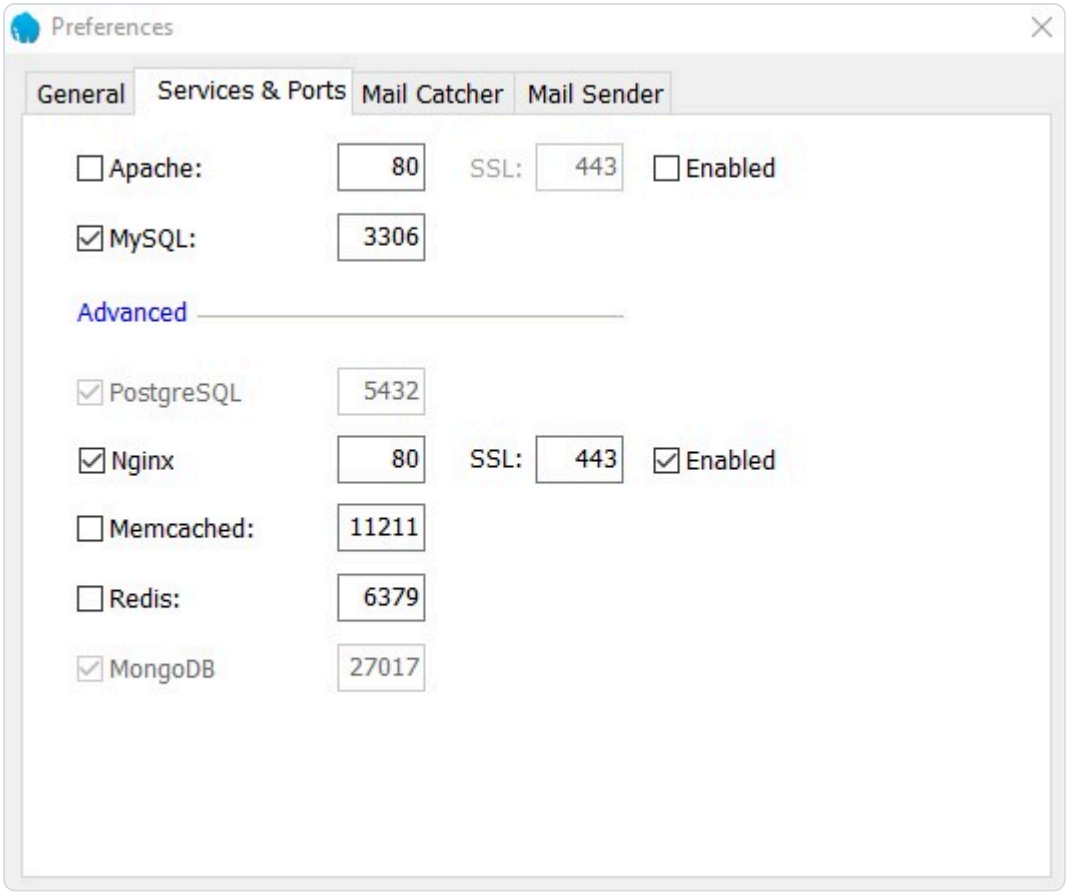
2.1 Configuration de Laragon avec Nginx

Si vous utilisez **Laragon sous Windows**, vous devez configurer **Nginx** au lieu d'Apache pour que les Server-Sent Events fonctionnent correctement.

Étapes dans Laragon :

1. **Ouvrir les préférences** de Laragon (clic droit sur l'icône dans la barre des tâches)
2. **Onglet "Services & Ports"**
3. **Décocher "Apache"**
4. **Cocher "Nginx"**
5. **Configurer le port sur 80** (port par défaut)
6. **Redémarrer Laragon**





3. Installation des Dépendances

3.1 Installation du package JavaScript

La seule dépendance nécessaire côté front-end :

```bash
npm install @laravel/stream-vue
```

4. Mise en Place côté Backend

4.1 Service de Streaming - SimpleAskStreamService

Créez un nouveau service dédié au streaming. Ce service gère la communication avec l'API OpenRouter et parse les chunks SSE en temps réel.

Fichier : `app/Services/SimpleAskStreamService.php`

PHP

```php
<?php

declare(strict_types=1);

namespace App\Services;

use Generator;
use Illuminate\Support\Facades\Http;
use Psr\Http\Message\StreamInterface;

/**
 * Service simplifié pour le streaming avec l'API OpenRouter.
 *
 * Exemple pédagogique utilisant le client HTTP de Laravel.
 *
 * @see https://openrouter.ai/docs/api/reference/streaming
 */
class SimpleAskStreamService
{
    public const DEFAULT_MODEL = 'openai/gpt-4o-mini';

    private string apiKey;privatestringapiKey;
    private string apiKey;privatestringbaseUrl;

    public function __construct()
    {
        this->apiKey=config('services.openrouter.apikey');this→apiKey =
config('services.openrouter.api_key');
        this->apiKey=config('services.openrouter.apikey');this→baseUrl =
rtrim(config('services.openrouter.base_url', 'https://openrouter.ai/api/v1'), '/');
    }

    /**
     * Récupère la liste des modèles disponibles (avec cache).
     */
    public function getModels(): array
    {
        return cache()→remember('openrouter.models', now()→addHour(), function ():
array {
            response=Http::withToken(response =
Http::withToken(response=Http::withToken(this→apiKey)→get("{this→baseUrl}/models");

            return collect(response→json('data', []))
                →sortBy('name')
                →map(fn(array model):array⇒['id'⇒model): array ⇒ [
                    'id' ⇒ model):array⇒['id'⇒model['id'],
                    'name' ⇒ model['name'],'description'⇒model['name'],
                    'description' ⇒ model['name'],'description'⇒model['description']
?? '',
                    'context_length' ⇒ model['contextlength']??
0,'maxcompletiontokens'⇒model['context_length'] ?? 0,
                    'max_completion_tokens' ⇒ model['contextlength']??0,'maxc
```

```
ompletiontokens'⇒model['top_provider']['max_completion_tokens'] ?? 0,
                    'input_modalities' ⇒ model['architecture']['inputmodalities']??
[],'outputmodalities'⇒model['architecture']['input_modalities'] ?? [],
                    'output_modalities' ⇒ model['architecture']['inputmodalities']??
[],'outputmodalities'⇒model['architecture']['output_modalities'] ?? [],
                    'supported_parameters' ⇒ model['supported_parameters'] ?? [],
                ])
                →values()
                →toArray();
        });
    }

    /**
     * Récupère la liste légère des modèles.
     */
    public function getModelsLight(): array
    {
        return collect(this→getModels())
            →map(fn(array m):array⇒['id'⇒m): array ⇒ ['id' ⇒ m):array⇒
['id'⇒m['id'], 'name' ⇒ m['name']])
            →values()
            →toArray();
    }

    /**
     * Récupère les détails d'un modèle.
     */
    public function getModelDetails(string id): ?array
    {
        return collect(this->getModels())->firstWhere('id',this→getModels())
→firstWhere('id', this->getModels())->firstWhere('id',id);
    }

    /**
     * Stream un message en temps réel vers la sortie.
     * Output le contenu texte directement (compatible avec useStream de Laravel).
     */
    public function streamToOutput(
        array messages,?stringmessages,
        ?string messages,?stringmodel = null,
        float temperature=1.0,?stringtemperature = 1.0,
        ?string temperature=1.0,?stringreasoningEffort = null
    ): void {
        response=response = response=this→sendStreamRequest(messages,messages,
messages,model, temperature,temperature, temperature,reasoningEffort);

        if (response→failed()) {
            echo "[ERROR] " . $response→json('error.message', 'HTTP Error');
            $this→flush();
            return;
        }

        foreach (this→parseSSEStream(response->toPsrResponse()->getBody())asresponse
```

```
->toPsrResponse()->getBody()) as response->toPsrResponse()->getBody())asevent) {
            if (event['type'] === 'error') {
                echo "[ERROR] " . $event['data'];
                $this->flush();
                return;
            }

            if (event['type'] === 'content' && event['data']) {
                echo $event['data'];
                $this->flush();
            }

            // Pour le reasoning, on utilise un préfixe spécial
            if (event['type'] === 'reasoning' && event['data']) {
                echo "[REASONING]" . $event['data'] . "[/REASONING]";
                $this->flush();
            }
        }
    }

    /**
     * Flush la sortie immédiatement.
     */
    private function flush(): void
    {
        if (ob_get_level() > 0) {
            ob_flush();
        }
        flush();
    }

    /**
     * Envoie la requête streaming à l'API.
     */
    private function sendStreamRequest(
        array messages,
        ?string model,floatmodel,
        float model,floattemperature,
        ?string reasoningEffort
    ): \Illuminate\Http\Client\Response {
        $payload = [
            'model' => $model ?? self::DEFAULT_MODEL,
            'messages' => [$this->getSystemPrompt(), ...$messages],
            'temperature' => $temperature,
            'stream' => true,
        ];

        if ($reasoningEffort !== null) {
            $payload['reasoning'] = ['effort' => $reasoningEffort];
        }

        return Http::withToken($this->apiKey)
            ->withHeaders([
```

```
                'HTTP-Referer' => config('app.url'),
                'X-Title' => config('app.name'),
            ])
            ->withOptions(['stream' => true])
            ->timeout(120)
            ->post("{$this->baseUrl}/chat/completions", $payload);
    }

    /**
     * Parse un stream SSE et yield les événements.
     *
     * @return Generator<array{type: string, data: string|null}>
     */
    private function parseSSEStream(StreamInterface body): Generator
    {
        buffer='';while(!buffer = '';

        while (!buffer='';while(!body->eof()) {
            buffer•=buffer •= buffer•=body->read(1024);

            while ((pos=strpos(pos = strpos(pos=strpos(buffer, "\n")) !== false) {
                line=trim(substr(line = trim(substr(line=trim(substr(buffer, 0,
pos));pos));
                pos));buffer = substr(buffer,buffer, buffer,pos + 1);

                if (event=event = event=this->parseSSELine(line)) {
                    yield $event;
                }
            }
        }
    }

    /**
     * Parse une ligne SSE.
     */
    private function parseSSELine(string line): ?array
    {
        if (line===''||strstartswith(line === '' || str_starts_with(line===''||strs
tartswith(line, ':')) {
            return null;
        }

        if (!str_starts_with(line,'data:'))returnnull;line, 'data: ')) {
            return null;
        }

        line,'data:'))returnnull;data = substr(line,6);if(line, 6);

        if (line,6);if(data === '[DONE]') {
            return ['type' => 'done', 'data' => null];
        }

        return this->parseJSON(this->parseJSON(this->parseJSON(data);
```

```
    }

    /**
     * Parse le JSON d'un chunk SSE.
     */
    private function parseJSON(string $json): ?array
    {
        try {
            $parsed = json_decode($json, true, 512, JSON_THROW_ON_ERROR);

            if (isset($parsed['error'])) {
                return ['type' => 'error', 'data' => $parsed['error']['message'] ??
'Unknown error'];
            }

            $delta = $parsed['choices'][0]['delta'] ?? [];

            if (!empty($delta['content'])) {
                return ['type' => 'content', 'data' => $delta['content']];
            }

            if (!empty($delta['reasoning'])) {
                return ['type' => 'reasoning', 'data' => $delta['reasoning']];
            }

            if (!empty($delta['reasoning_content'])) {
                return ['type' => 'reasoning', 'data' => $delta['reasoning_content']];
            }

            return null;
        } catch (\JsonException) {
            return null;
        }
    }

    /**
     * Retourne le prompt système.
     */
    private function getSystemPrompt(): array
    {
        return [
            'role' => 'system',
            'content' => view('prompts.system', [
                'now' => now()->locale('fr')->format('l d F Y H:i'),
                'user' => auth()->user()?->name ?? 'l\'utilisateur',
            ])->render(),
        ];
    }
}
```

Points clés du service :

- `streamToOutput()` : envoie les chunks directement à la sortie HTTP via `echo` et `flush()`
- **Marqueurs spéciaux** : le reasoning est encapsulé dans `[REASONING]...[/REASONING]` pour être parsé côté frontend
- **Parsing SSE** : lit le stream ligne par ligne et extrait les chunks de contenu
- **Support du reasoning** : paramètre optionnel `reasoning_effort` (low/medium/high)

4.2 Contrôleur - AskStreamController

Créez un nouveau contrôleur dédié à cette fonctionnalité de streaming.

Fichier : `app/Http/Controllers/AskStreamController.php`

PHP

```php
<?php

declare(strict_types=1);

namespace App\Http\Controllers;

use App\Services\SimpleAskStreamService;
use Illuminate\Http\Request;
use Inertia\Inertia;
use Inertia\Response;
use Symfony\Component\HttpFoundation\StreamedResponse;

/**
 * Controller pour la démonstration du streaming SSE.
 *
 * Exemple pédagogique : streaming temps réel avec Laravel + Vue.
 */
class AskStreamController extends Controller
{
    public function __construct(
        private SimpleAskStreamService streamService)/
***Affichelapagedestreaming.*/publicfunctionindex(RequeststreamService
    ) {}

    /**
     * Affiche la page de streaming.
     */
    public function index(Request streamService)/
***Affichelapagedestreaming.*/publicfunctionindex(Requestrequest): Response
    {
        modelId=modelId = modelId=request→input('model')
            ?? auth()→user()?→preferred_model
            ?? SimpleAskStreamService::DEFAULT_MODEL;

        return Inertia::render('AskStream/Index', [
            'models' ⇒ this->streamService->getModelsLight(),’selectedModel’⇒this
→streamService→getModelsLight(),
            'selectedModel' ⇒ this->streamService->getModelsLight(),
’selectedModel’⇒modelId,
            'selectedModelDetails' ⇒ fn() ⇒
this->streamService->getModelDetails(this→streamService
→getModelDetails(this->streamService->getModelDetails(modelId),
        ]);
    }

    /**
     * Endpoint de streaming.
     */
    public function stream(Request $request): StreamedResponse
    {
        $validated = $request→validate([
            'message' ⇒ 'required|string|max:100000',
```

```
            'model' => 'required|string',
            'temperature' => 'nullable|numeric|min:0|max:2',
            'reasoning_effort' => 'nullable|string|in:low,medium,high',
        ]);

        // Update user's preferred model
        $user = auth()->user();
        if ($user && $user->preferred_model !== $validated['model']) {
            $user->update(['preferred_model' => $validated['model']]);
        }

        $messages = [['role' => 'user', 'content' => $validated['message']]];
        $model = $validated['model'];
        $temperature = (float) ($validated['temperature'] ?? 1.0);
        $reasoningEffort = $validated['reasoning_effort'] ?? null;

        return response()->stream(
            function () use ($messages, $model, $temperature, $reasoningEffort): void
{
                $this->streamService->streamToOutput($messages, $model, $temperature,
$reasoningEffort);
            },
            headers: [
                'Content-Type' => 'text/plain; charset=utf-8',
                'Cache-Control' => 'no-cache, no-store',
                'X-Accel-Buffering' => 'no',
            ]
        );
    }
}
```

Points clés :

- `response()->stream()` crée automatiquement une réponse SSE
- Le service `streamToOutput()` écrit directement dans la sortie HTTP
- Headers HTTP spécifiques pour désactiver le buffering
- Support du `reasoning_effort` pour les modèles compatibles

4.3 Routes

Ajoutez les routes dans `routes/web.php` :

PHP

```
Route::get('/ask-stream', [\App\Http\Controllers\AskStreamController::class, 'index'])
    ->name('stream.index');

Route::post('/ask-stream', [\App\Http\Controllers\AskStreamController::class,
'stream'])
    ->name('stream.post');
```

5. Implémentation côté Frontend

5.1 Hook useStream - Code Essentiel

Voici le code essentiel pour implémenter le streaming côté frontend. Le hook `useStream` de Laravel gère automatiquement la connexion SSE et la réception des chunks.

VUE

```vue
<script setup lang="ts">
import { ref } from 'vue';
import { useStream } from '@laravel/stream-vue';

// State
const message = ref('');
const model = ref('openai/gpt-4o-mini');
const temperature = ref(1.0);
const reasoningEffort = ref<'low' | 'medium' | 'high' | null>(null);

/**
 * useStream hook - Le hook concatène automatiquement dans `data`
 * Le backend envoie du texte avec marqueurs [REASONING]...[/REASONING]
 */
const { data, isFetching, isStreaming, send, cancel } = useStream(
    '/ask-stream',
    {
        onData: () => {
            // Callback appelé pour chaque chunk reçu
            // Le contenu est automatiquement concaténé dans `data`
        },
        onFinish: () => {
            // Appelé quand le stream se termine
            message.value = '';
        },
        onError: (err: Error) => {
            console.error('Erreur streaming:', err);
        },
    },
);

/**
 * Submit handler - Envoie la requête de streaming
 */
const submit = () => {
    if (!message.value.trim()) return;

    send({
        message: message.value,
        model: model.value,
        temperature: temperature.value,
        reasoning_effort: reasoningEffort.value,
    });
};
</script>
```

Points clés du hook `useStream` :

- `data` : contient la réponse complète, automatiquement concaténée
- `isStreaming` : booléen indiquant si un stream est en cours

- `isFetching` : booléen pour la phase de connexion
- `send(payload)` : fonction pour démarrer le stream
- `cancel()` : fonction pour annuler le stream en cours
- **Callbacks** :
  - `onData` : appelé pour chaque chunk reçu
  - `onFinish` : appelé quand le stream se termine
  - `onError` : appelé en cas d'erreur

5.2 Parsing des Marqueurs Reasoning

Le backend envoie le reasoning encapsulé dans des marqueurs spéciaux. Voici comment les extraire :

VUE

```vue
<script setup lang="ts">
import { computed } from 'vue';

/**
 * Extrait le contenu principal (sans le reasoning)
 */
const streamedContent = computed(() => {
    if (!data.value) return '';
    // Enlever les blocs [REASONING]...[/REASONING]
    return data.value
        .replace(/\[REASONING\][\s\S]*?\[\/REASONING\]/g, '')
        .trim();
});

/**
 * Extrait le reasoning des marqueurs
 */
const streamedReasoning = computed(() => {
    if (!data.value) return '';
    const matches = data.value.match(/\[REASONING\]([\s\S]*?)\[\/REASONING\]/g);
    if (!matches) return '';
    return matches
        .map((m) =>
            m.replace(/\[REASONING\]/g, '').replace(/\[\/REASONING\]/g, ''),
        )
        .join('');
});
</script>

<template>
    <!-- Affichage du reasoning (optionnel) -->
    <div v-if="streamedReasoning" class="reasoning-trace">
        <h4>Trace de raisonnement</h4>
        <pre>{{ streamedReasoning }}</pre>
    </div>

    <!-- Affichage du contenu principal -->
    <div class="main-content">
        <MarkdownRenderer :content="streamedContent" />
    </div>
</template>
```

Explication du parsing :

- Les modèles de raisonnement (comme les modèles o1) génèrent une trace de leur raisonnement
- Le backend encapsule cette trace dans `[REASONING]...[/REASONING]`
- On utilise des regex pour extraire séparément le contenu principal et le reasoning
- Le reasoning peut être affiché ou caché selon les besoins

6. Configuration CSRF et Authentification

6.1 Token CSRF dans le Template de Base

Le package `@laravel/stream-vue` cherche automatiquement le token CSRF dans une balise meta du `<head>`. Ajoutez cette ligne dans votre template de base Inertia :

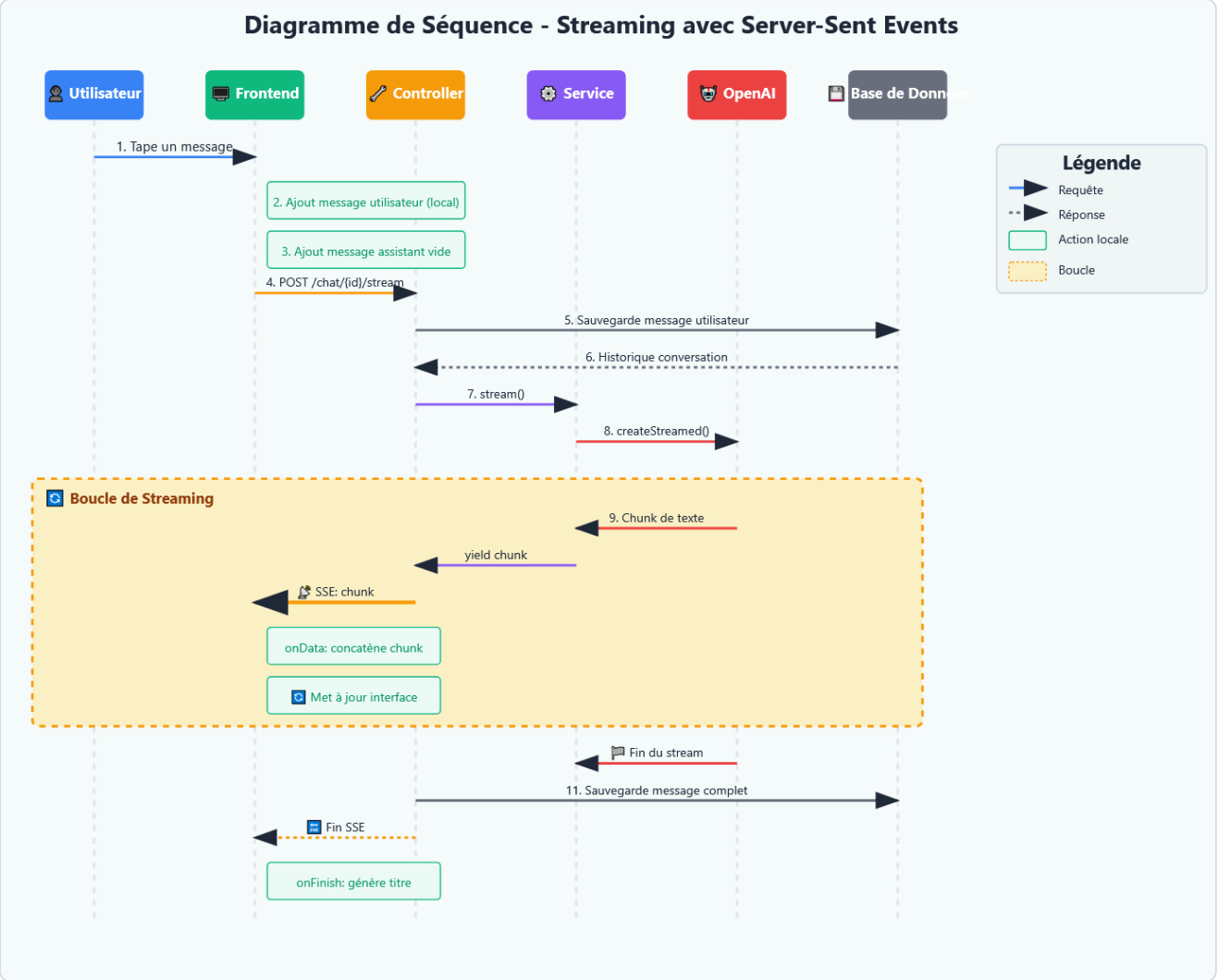
Fichier : `resources/views/app.blade.php`

```xml
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title inertia>{{ config('app.name', 'Laravel') }}</title>
    <!-- ... autres balises meta et scripts -->
</head>
```

Sans cette balise, vous obtiendrez une **erreur 419** lors des requêtes de streaming.

7. Diagramme de Séquence



8. Avantages des Server-Sent Events

Aspect	Server-Sent Events
Configuration	Aucune configuration serveur
Dépendances	1 package npm uniquement
Coût	Gratuit
Performance	Direct HTTP, faible latence
Complexité	Simple et élégant

Conclusion

Les **Server-Sent Events** avec `@laravel/stream-vue` offrent une solution élégante et simple pour implémenter le streaming en temps réel. Cette approche native élimine la complexité des solutions externes (comme Pusher) tout en offrant d'excellentes performances, sans nécessiter aucune configuration serveur supplémentaire.

Sources

- https://laravel.com/docs/12.x/responses#streamed-responses
- https://laravel.com/docs/12.x/broadcasting
- https://openrouter.ai/docs/api/reference/streaming
- https://openrouter.ai/docs/guides/best-practices/reasoning-tokens#example-1-streaming-mode-with-reasoning-tokens

4.10 Intégration d'Images au LLM

Vous rêvez d'avoir un chatbot suffisamment malin pour décrire les photos de vos (malheureux) selfies ou deviner ce qui se trouve dans l'assiette ? Bonne nouvelle : certains modèles de langage savent gérer des images en plus du texte. Voici comment les utiliser sans transformer votre application en gouffre financier... et sans envoyer des toiles de 8 000 px de large.

1. Présentation Générale

Les LLM multimodaux peuvent interpréter un contenu visuel et vous renvoyer une description, un résumé ou tout autre commentaire pertinent. Pour en profiter, **encore faut-il** leur envoyer les images correctement : il ne suffit pas de coller un `.jpg` brut dans votre JSON.

- Certains modèles ne sont pas (encore) capables d'interpréter les images. Vérifiez donc la compatibilité.
- La mise en forme du message user change : on envoie un tableau de "blocs" (`text` , `image_url`), et non plus une simple chaîne de caractères.

2. Vérifier la Compatibilité

Avant de bombarder un LLM avec des photos de votre chat (le vrai, pas le chatbot), assurez-vous qu'il sache quoi en faire. Dans la méthode `getModels()` (de votre `ChatService`), nous déterminons la capacité du modèle via :

```php
'supports_image' => 'text+image->text' === $model['architecture']['modality']
```

Si `supports_image` est `true`, bingo : ce modèle sait traiter des images. Sinon, vous obtiendrez plus ou moins le même résultat qu'un poisson rouge devant un livre de math.

3. Format de Requête Multimodale

Pour envoyer une image au LLM, vous devez structurer le contenu de votre message sous forme de tableau. Exemple :

```json
"messages": [
  {
    "role": "user",
    "content": [
      {
        "type": "text",
        "text": "What is this image about?"
      },
      {
        "type": "image_url",
        "image_url": {
          "url": "https://example.com/magnifique_photo.jpg"
        }
      }
    ]
  }
]
```

- `type: "text"` : la partie texte classique.
- `type: "image_url"` : un bloc contenant l'URL d'une image (ou son équivalent base64, voir section suivante).

Lors de l'envoi, l'API traitera ces blocs pour comprendre que vous lui fournissez à la fois du texte et une image.

4. Image par URL vs Base64

4.1 URL Publique

La solution la plus simple consiste à héberger votre image quelque part (CDN, bucket S3, etc.), puis à placer cette URL dans la requête. Le modèle ira la chercher directement.

- Avantage : **Aucun** besoin de manipuler l'image côté backend, hormis vérifier qu'elle soit accessible.

- Inconvénient : Vous devez **impérativement** que l'image soit en ligne, et idéalement de taille raisonnable (ou la facture risque de grimper selon la politique du modèle).

4.2 Encodage Base64

Si vous avez une image locale ou que vous ne pouvez pas l'héberger, vous pouvez l'envoyer sous forme de chaîne base64, par exemple :

```javascript
"messages": [
  {
    "role": "user",
    "content": [
      {
        "type": "text",
        "text": "What's in this image?"
      },
      {
        "type": "image_url",
        "image_url": {
          "url": "..."
        }
      }
    ]
  }
]
```

> ***Attention*** *: la chaîne base64 doit inclure* `` *ou autre content-type, histoire que le modèle sache de quel format il s'agit.*

- Avantage : **Aucune** dépendance à un serveur de fichiers distant.
- Inconvénient : La requête HTTP peut doubler (ou plus) de taille si vous balancez une photo en 4K. Vive la surcharge…

5. Optimiser l'Image : Pourquoi et Comment

Plus l'image est lourde, plus vous risquez de payer, et plus la requête mettra du temps à arriver au LLM (et inversement). Autrement dit, si vous uploadez un panorama en 8K, préparez-vous à sortir le champagne (et le porte-monnaie).

5.1 Exemple d'une Classe d'Optimisation

Pour redimensionner, convertir et compresser l'image en JPEG de façon plus civilisée :

```php
class ImageService
{
    private const MAX_WIDTH = 800;
    private const JPEG_QUALITY = 85;

    public function optimizeImage(string $sourcePath): string
    {
        [$width, $height, $type] = getimagesize($sourcePath);
        $ratio = $width / $height;
        $newWidth = min($width, self::MAX_WIDTH);
        $newHeight = (int)($newWidth / $ratio);

        $newImage = imagecreatetruecolor($newWidth, $newHeight);

        $sourceImage = match ($type) {
            IMAGETYPE_JPEG => imagecreatefromjpeg($sourcePath),
            IMAGETYPE_PNG => imagecreatefrompng($sourcePath),
            IMAGETYPE_WEBP => imagecreatefromwebp($sourcePath),
            default => throw new \Exception('Format d\'image non supporté'),
        };

        imagecopyresampled($newImage, $sourceImage, 0, 0, 0, 0, $newWidth, $newHeight,
$width, $height);

        ob_start();
        imagejpeg($newImage, null, self::JPEG_QUALITY);
        $imageData = ob_get_clean();

        imagedestroy($sourceImage);
        imagedestroy($newImage);

        return $imageData;
    }
}
```

La fonction `optimizeImage()` :

1. **Redimensionne** l'image si elle dépasse `MAX_WIDTH` (ex. 800 px).
2. **Convertit** en JPEG de qualité 85 % (assez correct pour du descriptif).
3. **Retourne** les données binaires dans une chaîne.

5.2 Enchaîner sur le Base64

Si vous devez envoyer l'image en *base64*, il vous suffit de faire :

```php
optimizedData=(newImageService())->optimizeImage(optimizedData = (new ImageService())
→optimizeImage(optimizedData=(newImageService())->optimizeImage(cheminFichier);
base64Data=base64encode(base64Data = base64_encode(base64Data=base64e
ncode(optimizedData); // Ajoute le beau code encodé
url="".url = "" .
url="".base64Data; // Ou bien votre code "automatique"
```

Ensuite, vous placez `$url` dans la propriété `image_url` de votre request, comme vu plus haut.

6. Mise en Œuvre Rapide

1. **Vérifiez** si le modèle supporte les images (`supports_image == true`).
2. **Récupérez** votre image (upload de l'utilisateur, dossier local, etc.).
3. **Optimisez** (optionnel mais conseillé) pour éviter les fiascos sur la facture.
4. **Créez** le `messages` :

```php
messages=[['role'⇒'user','content'⇒[['type'⇒'text',
'text'⇒'Canyoudescribewhatthisis?'],['type'⇒'imageurl','imageurl'⇒
['url'⇒messages = [
  [
    'role' ⇒ 'user',
    'content' ⇒ [
      [
        'type' ⇒ 'text',
        'text' ⇒ 'Can you describe what this is?'
      ],
      [
        'type' ⇒ 'image_url',
        'image_url' ⇒ [
          'url' ⇒ messages=[['role'⇒'user','content'⇒[['type'⇒'text',
'text'⇒'Canyoudescribewhatthisis?'],['type'⇒'imageurl','imageurl'⇒['url'⇒url //
Soit un lien http(s), soit ...
        ]
      ]
    ]
  ]
];
```

5. **Envoyez** le tout au LLM via votre client OpenRouter ou OpenAI.
6. **Observez** la réponse : le modèle devrait vous décrire l'image ou répondre à vos questions, si tout s'est bien passé.



7. Bonnes Pratiques

- **Bouclez sur** `supports_image` : on répète l'importance de vérifier que le modèle est vraiment multimodal, sous peine de voir votre requête se faire bâcher.
- **Dimension et format** : compressez suffisamment, sinon l'API risque d'exploser le temps d'analyse (et peut-être vos factures).
- **Encodage** : assurez-vous de fournir le bon *content-type* dans la data URL si vous faites du base64.
- **Filtrage** : L'API fera de la modération côté serveur. Vous n'avez pas à mettre un système anti-chaton ou anti-contenu gore vous-même (sauf besoin particulier).

Voilà, vous avez désormais tous les ingrédients pour rendre votre chatbot (vaguement) visionnaire. Alors, à vos images et n'oubliez pas : "light is right" – plus l'image est légère, moins vous vous ferez taper sur les doigts (ou sur le portefeuille).

Documentation

- https://openrouter.ai/docs/requests#images-_-multimodal-requests

4.11 Annexe : Formatage des Messages pour l'API

Cette annexe détaille la structure attendue pour les messages envoyés à l'API de chat, avec des exemples concrets en PHP.

1. Structure Générale d'une Requête

1.1 Format PHP de Base

```php
$request = [
    // OBLIGATOIRE : soit "messages" soit "prompt"
    'messages' => [...], // Array de messages (recommandé)
    // OU
    'prompt' => 'Votre prompt direct', // String simple

    // Optionnel : modèle à utiliser
    'model' => 'anthropic/claude-3.5-sonnet', // Par défaut : modèle utilisateur

    // Paramètres de streaming
    'stream' => true, // Pour activer le streaming

    // Paramètres LLM
    'max_tokens' => 1000,
    'temperature' => 0.7, // Entre 0 et 2
    'top_p' => 0.9,       // Entre 0 et 1

    // Contrôle du format de sortie
    'response_format' => ['type' => 'json_object'], // Force JSON

    // Mots d'arrêt
    'stop' => ['```', 'FIN'], // String ou array de strings
];
```

2. Types de Messages

2.1 Message Utilisateur Simple

```php
$userMessage = [
    'role' => 'user',
    'content' => 'Bonjour, comment allez-vous ?'
];
```

2.2 Message Système

```php
$systemMessage = [
    'role' => 'system',
    'content' => 'Vous êtes un assistant spécialisé en développement web.'
];
```

2.3 Message Assistant

```php
$assistantMessage = [
    'role' => 'assistant',
    'content' => 'Bonjour ! Je vais bien, merci. Comment puis-je vous aider ?'
];
```

2.4 Message avec Nom (Persona)

```php
$namedMessage = [
    'role' => 'user',
    'content' => 'Quelle est la météo aujourd\'hui ?',
    'name' => 'Alice' // Sera affiché comme "Alice: Quelle est la météo..."
];
```

3. Messages Multimodaux (Texte + Images)

3.1 Structure ContentPart

Pour les messages utilisateur avec images, le `content` devient un array de "parts" :

```php
$multimodalMessage = [
    'role' => 'user',
    'content' => [
        // Part texte
        [
            'type' => 'text',
            'text' => 'Peux-tu analyser cette image ?'
        ],
        // Part image
        [
            'type' => 'image_url',
            'image_url' => [
                'url' => 'https://example.com/image.jpg',
                'detail' => 'high' // 'low', 'high', ou 'auto' (défaut)
            ]
        ]
    ]
];
```

3.2 Image Base64

```php
$base64Message = [
    'role' => 'user',
    'content' => [
        [
            'type' => 'text',
            'text' => 'Décris cette capture d\'écran'
        ],
        [
            'type' => 'image_url',
            'image_url' => [
                'url' => '...',
                'detail' => 'auto'
            ]
        ]
    ]
];
```

3.3 Plusieurs Images

```php
$multipleImagesMessage = [
    'role' => 'user',
    'content' => [
        [
            'type' => 'text',
            'text' => 'Compare ces deux images :'
        ],
        [
            'type' => 'image_url',
            'image_url' => ['url' => 'https://example.com/image1.jpg']
        ],
        [
            'type' => 'image_url',
            'image_url' => ['url' => 'https://example.com/image2.jpg']
        ]
    ]
];
```

4. Messages avec Tool Calling

4.1 Définition d'un Tool

```php
$weatherTool = [
    'type' => 'function',
    'function' => [
        'name' => 'get_weather',
        'description' => 'Obtient la météo pour une ville donnée',
        'parameters' => [
            'type' => 'object',
            'properties' => [
                'city' => [
                    'type' => 'string',
                    'description' => 'Le nom de la ville'
                ],
                'unit' => [
                    'type' => 'string',
                    'enum' => ['celsius', 'fahrenheit'],
                    'description' => 'L\'unité de température'
                ]
            ],
            'required' => ['city']
        ]
    ]
];
```

4.2 Requête avec Tools

```php
requestWithTools=['messages'⇒[['role'⇒'system',
'content'⇒'Vouse^tesunassistantme´te´o.'],['role'⇒'user',
'content'⇒'Queltempsfait-ila`Paris?']],'tools'⇒[requestWithTools = [
    'messages' => [
        [
            'role' => 'system',
            'content' => 'Vous êtes un assistant météo.'
        ],
        [
            'role' => 'user',
            'content' => 'Quel temps fait-il à Paris ?'
        ]
    ],
    'tools' => [requestWithTools=['messages'⇒[['role'⇒'system',
'content'⇒'Vouse^tesunassistantme´te´o.'],['role'⇒'user',
'content'⇒'Queltempsfait-ila`Paris?']],'tools'⇒[weatherTool],
    'tool_choice' => 'auto' // 'none', 'auto', ou objet spécifique
];
```

4.3 Message Tool Response

```php
$toolMessage = [
    'role' => 'tool',
    'content' => '{"temperature": 22, "condition": "ensoleillé", "humidity": 65}',
    'tool_call_id' => 'call_abc123',
    'name' => 'get_weather'
];
```

5. Exemples Complets de Conversations

5.1 Conversation Simple

```php
$simpleConversation = [
    'messages' => [
        [
            'role' => 'system',
            'content' => 'Vous êtes un développeur PHP expert.'
        ],
        [
            'role' => 'user',
            'content' => 'Comment faire une requête HTTP en PHP ?'
        ],
        [
            'role' => 'assistant',
            'content' => 'Vous pouvez utiliser cURL ou file_get_contents()...'
        ],
        [
            'role' => 'user',
            'content' => 'Peux-tu me donner un exemple avec cURL ?'
        ]
    ],
    'model' => 'anthropic/claude-3.5-sonnet',
    'temperature' => 0.7,
    'max_tokens' => 1000,
    'stream' => true
];
```

5.2 Conversation avec Images

```php
$imageConversation = [
    'messages' => [
        [
            'role' => 'system',
            'content' => 'Vous analysez des interfaces utilisateur.'
        ],
        [
            'role' => 'user',
            'content' => [
                [
                    'type' => 'text',
                    'text' => 'Analyse cette maquette et donne-moi des suggestions
d\'amélioration :'
                ],
                [
                    'type' => 'image_url',
                    'image_url' => [
                        'url' => 'https://example.com/mockup.png',
                        'detail' => 'high'
                    ]
                ]
            ]
        ]
    ],
    'model' => 'anthropic/claude-3.5-sonnet',
    'temperature' => 0.3
];
```

5.3 Conversation avec Tool Calling

```php
$toolConversation = [
    'messages' => [
        [
            'role' => 'system',
            'content' => 'Vous êtes un assistant capable d\'effectuer des calculs.'
        ],
        [
            'role' => 'user',
            'content' => 'Calcule 15% de 250 et convertis le résultat en euros.'
        ]
    ],
    'tools' => [
        [
            'type' => 'function',
            'function' => [
                'name' => 'calculate',
                'description' => 'Effectue des calculs mathématiques',
                'parameters' => [
                    'type' => 'object',
                    'properties' => [
                        'expression' => [
                            'type' => 'string',
                            'description' => 'Expression mathématique à calculer'
                        ]
                    ],
                    'required' => ['expression']
                ]
            ]
        ],
        [
            'type' => 'function',
            'function' => [
                'name' => 'convert_currency',
                'description' => 'Convertit une devise',
                'parameters' => [
                    'type' => 'object',
                    'properties' => [
                        'amount' => ['type' => 'number'],
                        'from' => ['type' => 'string'],
                        'to' => ['type' => 'string']
                    ],
                    'required' => ['amount', 'from', 'to']
                ]
            ]
        ]
    ],
    'tool_choice' => 'auto'
];
```

Source

- https://openrouter.ai/docs/api-reference/overview#requests

4.12 Conclusion du cours

- Trois_Vérités_Développeur.pdf