

PROGRA

Programmer avec l'IA

Thibault Six

17 chapitres

SUPPORTS DE COURS

https://cours.tsix.be/cours/programmer-avec-lia

Version 1 · 12/09/2026

PRÉSENTATION

Programmer avec l'IA — De pisseur de code à chef d'orchestre

Le cours en un coup d'œil

🎯 **Objectif**	Apprendre à créer des outils et des contenus en guidant une IA
👥 **Public**	Professeurs, direction, administratifs — tous niveaux techniques
⏱️ **Durée**	3 heures (+ 15 min de pause)
💻 **Prérequis**	Un PC portable + accès à au moins une IA (gratuit ou payant)
📐 **Format**	Progressif — théorie courte, beaucoup de pratique

Ce que vous saurez faire à la fin

#	Objectif	Bloc
1	**Comprendre** comment les IA ont transformé le métier de développeur	Bloc 1
2	**Choisir** les bons outils IA selon vos besoins et votre budget	Bloc 1
3	**Créer** des contenus interactifs (quiz, jeux, cours, outils) dans le navigateur	Bloc 2
4	**Installer** et configurer un agent de programmation IA sur votre machine	Bloc 3
5	**Développer** une application fonctionnelle en guidant un agent	Bloc 3
6	**Appliquer** la méthode de travail efficace avec un agent IA	Blocs 2-3
7	**Sécuriser** votre code et vos applications avant de les publier	Bloc 3

À quoi ça va vous servir concrètement ?

🎓 Professeurs

- Créer des **cours interactifs** avec exercices intégrés, animations, visualisations
- Concevoir des **jeux éducatifs** : quiz, memory, escape rooms, simulations
- Construire des **outils d'évaluation** : grilles de compétences, encodage de points, export PDF
- Automatiser la **production de contenu** : génération d'examens, conversion de supports

🏢 Direction & administration

- Créer des **tableaux de bord** à partir de vos données Excel existantes
- Automatiser les **tâches répétitives** : fusion de fichiers, extraction de données, rapports
- Construire des **outils de gestion** sur mesure : planification, suivi, inventaire
- Analyser vos **données scolaires** : résultats, absences, tendances

👤 Tous

- **Gagner du temps** en transformant vos idées en outils fonctionnels
- **Ne plus dépendre** de quelqu'un d'autre pour chaque besoin technique
- **Devenir autonome** : votre IA sera toujours là pour vous guider

Le programme

#	Section	Type	Durée
	Bloc 1 — Comprendre		**25 min**
1.1	Le monde a changé — benchmarks et évolution	Théorie	10 min
1.2	LLM Face-Off : comparez les IA vous-mêmes	Activité	10 min
1.3	Les outils, les prix, comment démarrer	Théorie	5 min
	Bloc 2 — Créer dans le navigateur		**35 min**
2.1	Démo live : créer un contenu interactif + la méthode	Démo	10 min
2.2	**Exercice 1** — Votre premier artifact interactif	✍️ Pratique	25 min
	☕ **Pause**		**15 min**
	Bloc 3 — L'agent sur la machine		**80 min**
3.1	Chat vs Agent : la grande différence	Théorie	5 min
3.2	Installation de l'environnement	Guidé	10 min
3.3	Démo : du prompt au site en ligne	Démo	10 min
3.4	Piloter son projet : les fichiers .md	Théorie	5 min
3.5	Sécurité et audits	Théorie	5 min
3.6	**Exercice 2** — Votre application, votre choix	✍️ Pratique	45 min
	Bloc 4 — Et après ?		**15 min**
4.1	Partage des créations	Activité	5 min

#	Section	Type	Durée
4.2	Pistes pour aller plus loin	Théorie	10 min
	Total		**~170 min + pause**

La philosophie du cours

> ***Vous n'allez pas apprendre à coder.*** *Vous allez apprendre à* ***diriger une IA qui code pour vous****.*

Le métier a changé. On ne tape plus du code ligne par ligne. On passe de *"pisseur de code"* à *"chef d'orchestre"* : on guide des agents, on valide, on oriente.

Ce cours est **100% pratique**. Les parties théoriques sont courtes (5-10 min) et servent uniquement à vous donner le contexte nécessaire avant de passer à l'action. Vous repartez avec des créations concrètes que vous pourrez utiliser dès lundi.

Ce qu'il vous faut

Élément	Détail
💻 **Un PC portable**	Windows, Mac ou Linux — peu importe
🌐 **Une connexion internet**	Pour accéder aux outils et déployer
🤖 **Un compte IA**	Claude, ChatGPT, Gemini, Copilot... (gratuit suffit)
🧠 **De la curiosité**	C'est le seul vrai prérequis

💡 **Pas d'abonnement payant ?** Pas de problème. Codex (OpenAI) est gratuit pendant 2 mois, GitHub Copilot est gratuit pour les enseignants, et les limites gratuites de Claude/ChatGPT/Gemini suffisent pour l'exercice 1.

Cours rédigé à l'aide de Claude 4.6 Opus et Gemini 3 Nano Banana Pro.

Sommaire

- Programmer avec l'IA — De pisseur de code à chef d'orchestre

1 Introduction
- 1.1 Le monde a changé
- 1.2 LLM Face-Off
- 1.3 Les outils

2 Canvas
- 2.1 Démonstration
- 2.2 Méthodologie
- 2.3 Exercice

3 Agents
- 3.1 Chat vs Agent
- 3.2 Installation
- 3.3 Démonstration
- 3.4 Fichiers de pilotage
- 3.5 Sécurité
- 3.6 Exercice
- 3.7 Vercel

4 Et après ?

CHAPITRE 1

Bloc 1 — Comprendre le nouveau paysage

Durée : 30 min | **Type** : Théorie + Activité



Slides du chapitre

- introduction.pdf

Objectifs d'apprentissage

À la fin de ce bloc, vous serez capables de :

1. **Situer** l'état actuel des IA de programmation à travers les benchmarks de référence
2. **Comparer** les modèles IA de manière empirique via une activité de test en aveugle
3. **Choisir** un outil adapté à votre budget et vos besoins

Structure du bloc

Section	Contenu	Type	Durée
1.1	Le monde a changé	Théorie	10 min
1.2	LLM Face-Off : testez vous-mêmes	Activité	15 min
1.3	Les outils, les prix, comment démarrer	Théorie	5 min

Ce premier bloc pose le décor. Avant de mettre les mains dans le code (ça vient dans les blocs suivants), il est essentiel de comprendre *pourquoi* le monde du développement a basculé, *comment* se comparent les outils disponibles, et *par où* commencer.

1.1 1.1 — Le monde a changé

Durée : 10 min | **Type** : Théorie | **Slides** : ~5-6



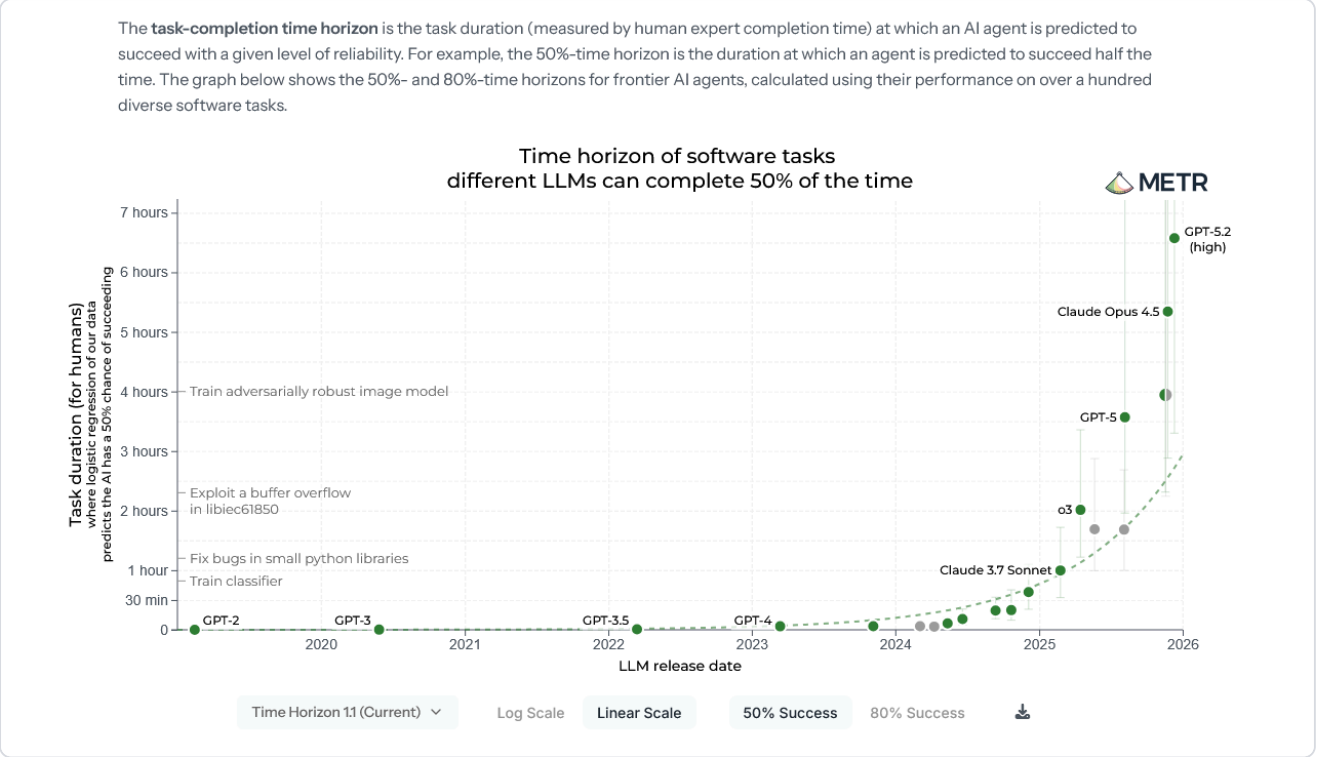
L'explosion des capacités

En mars 2025, l'organisme de recherche METR a publié une étude qui a secoué le monde de la tech : *"Measuring AI Ability to Complete Long Tasks"*. Leur constat est simple et vertigineux : la durée des tâches que les agents IA peuvent accomplir de manière autonome **double tous les 7 mois** environ.

Pour comprendre ce que ça signifie concrètement :

- **2019** (GPT-2) : l'IA pouvait à peine compléter des tâches de quelques **secondes** — auto-compléter une ligne de code, suggérer un mot.
- **2023** (GPT-4) : l'IA gérait des tâches de quelques **minutes** — écrire une fonction, corriger un bug simple.
- **2025** (Claude 3.7 Sonnet, o3) : l'IA complète de manière fiable des tâches d'environ **1 heure** — construire un module complet, résoudre des problèmes multi-étapes.
- **2026 et au-delà** : la tendance continue. Les modèles les plus récents (Gemini 3 Pro, GPT-5.1 Codex) repoussent encore cette limite.

Et cette courbe est **exponentielle**, pas linéaire. Ce n'est pas une amélioration graduelle — c'est une accélération.



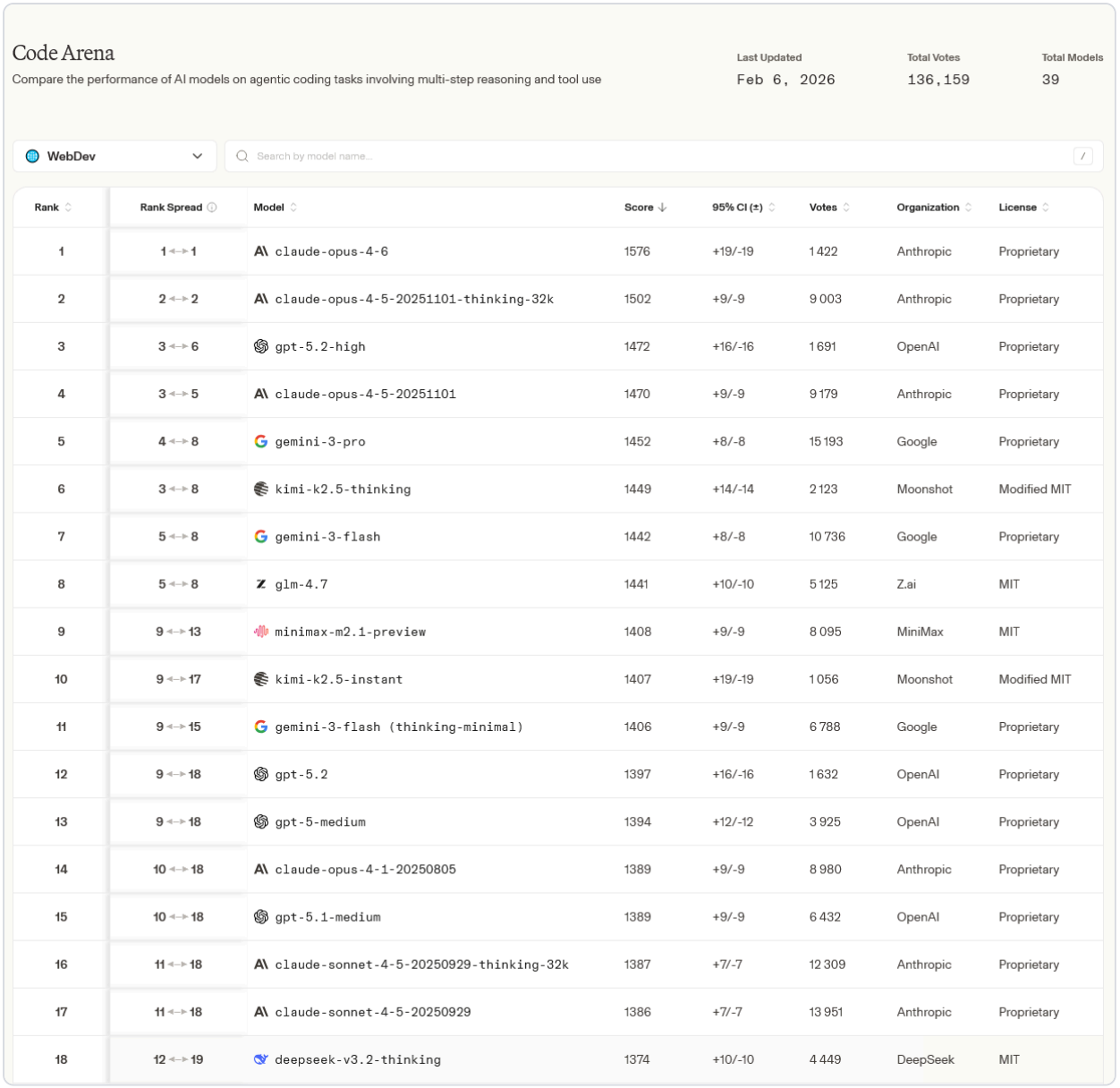
https://metr.org/time-horizons/

Mais les capacités ne se limitent pas à la durée des tâches. Les classements d'intelligence générale montrent une compétition féroce entre les modèles :



https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index

Et quand on zoome spécifiquement sur le **code et le développement web** — le domaine qui nous concerne le plus directement :



Code Arena

Compare the performance of AI models on agentic coding tasks involving multi-step reasoning and tool use

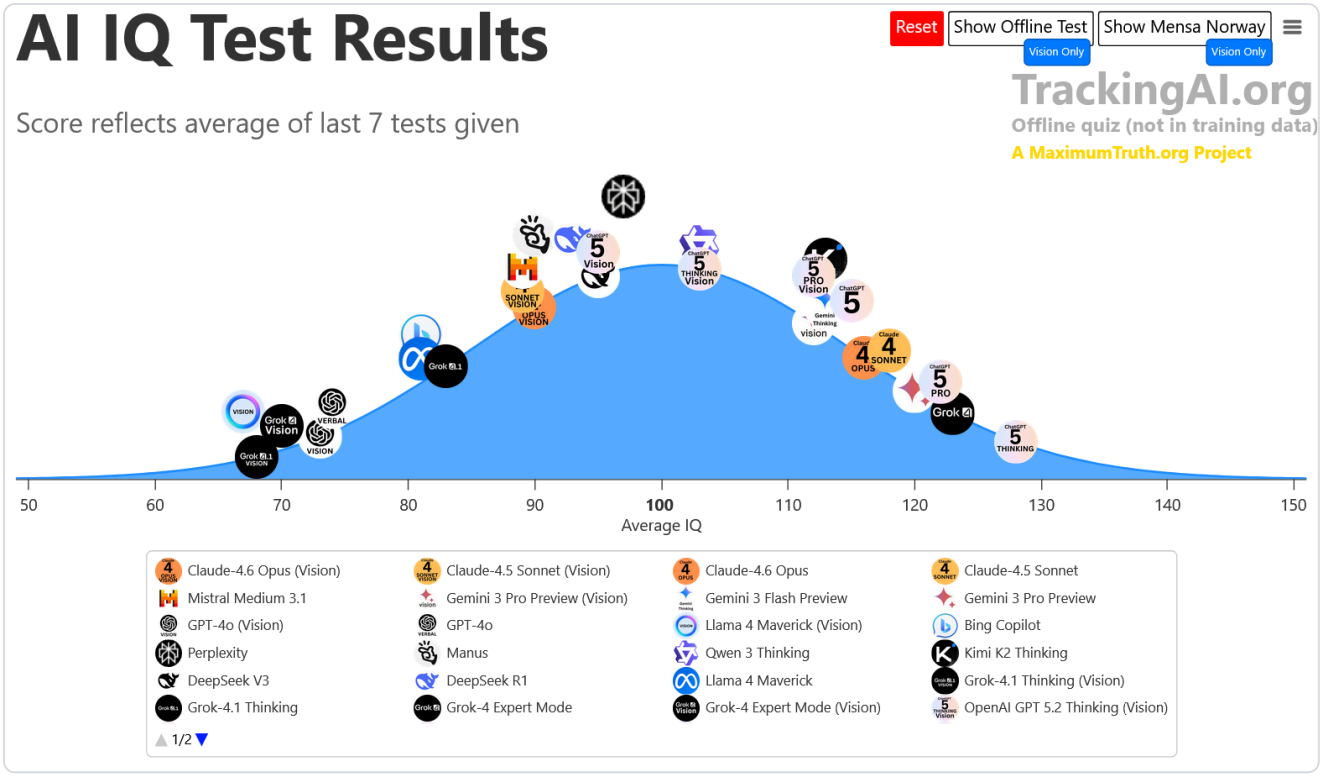
Last Updated: Feb 6, 2026 | Total Votes: 136,159 | Total Models: 39

WebDev | Search by model name...

Rank	Rank Spread	Model	Score	95% CI (±)	Votes	Organization	License
1	1 ↔ 1	claude-opus-4-6	1576	+19/-19	1 422	Anthropic	Proprietary
2	2 ↔ 2	claude-opus-4-5-20251101-thinking-32k	1502	+9/-9	9 003	Anthropic	Proprietary
3	3 ↔ 6	gpt-5.2-high	1472	+16/-16	1 691	OpenAI	Proprietary
4	3 ↔ 5	claude-opus-4-5-20251101	1470	+9/-9	9 179	Anthropic	Proprietary
5	4 ↔ 8	gemini-3-pro	1452	+8/-8	15 193	Google	Proprietary
6	3 ↔ 8	kimi-k2.5-thinking	1449	+14/-14	2 123	Moonshot	Modified MIT
7	5 ↔ 8	gemini-3-flash	1442	+8/-8	10 736	Google	Proprietary
8	5 ↔ 8	glm-4.7	1441	+10/-10	5 125	Z.ai	MIT
9	9 ↔ 13	minimax-m2.1-preview	1408	+9/-9	8 095	MiniMax	MIT
10	9 ↔ 17	kimi-k2.5-instant	1407	+19/-19	1 056	Moonshot	Modified MIT
11	9 ↔ 15	gemini-3-flash (thinking-minimal)	1406	+9/-9	6 788	Google	Proprietary
12	9 ↔ 18	gpt-5.2	1397	+16/-16	1 632	OpenAI	Proprietary
13	9 ↔ 18	gpt-5-medium	1394	+12/-12	3 925	OpenAI	Proprietary
14	10 ↔ 18	claude-opus-4-1-20250805	1389	+9/-9	8 980	Anthropic	Proprietary
15	10 ↔ 18	gpt-5.1-medium	1389	+9/-9	6 432	OpenAI	Proprietary
16	11 ↔ 18	claude-sonnet-4-5-20250929-thinking-32k	1387	+7/-7	12 309	Anthropic	Proprietary
17	11 ↔ 18	claude-sonnet-4-5-20250929	1386	+7/-7	13 951	Anthropic	Proprietary
18	12 ↔ 19	deepseek-v3.2-thinking	1374	+10/-10	4 449	DeepSeek	MIT

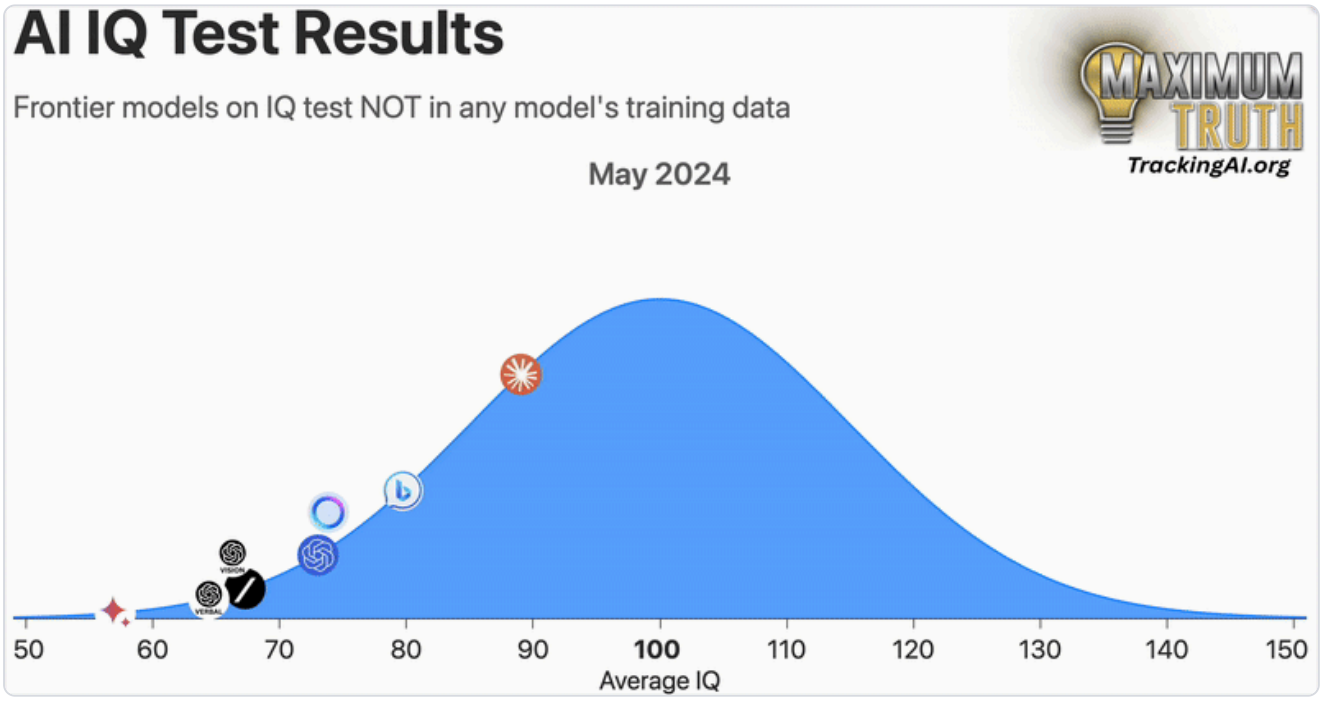
https://arena.ai/fr/leaderboard/code

Enfin, par pure curiosité (et parce que c'est parlant) : les IA passent aussi des **tests de QI**. Le site TrackingAI.org fait passer le test Mensa Norway aux modèles chaque semaine.



https://www.trackingai.org/home

Évolution dans le temps



⚠️ **Attention aux benchmarks** : ces classements bougent constamment et mesurent des choses différentes. Aucun ne capture "l'intelligence" dans son ensemble. Mais la tendance est indéniable : les capacités progressent à un rythme effréné, et les modèles d'aujourd'hui sont incomparablement plus performants que ceux d'il y a 18 mois.

Le nouveau métier de développeur

Ces chiffres, c'est bien joli, mais qu'est-ce que ça change concrètement dans la manière de travailler ?

L'analogie de l'architecte. Imaginez un architecte et un maçon. L'architecte ne pose pas les briques lui-même — il conçoit le plan, choisit les matériaux, vérifie que la structure tient debout, et corrige quand quelque chose ne va pas. Le maçon exécute. Avant l'IA, le programmeur était les deux à la fois : il concevait ET il tapait chaque ligne. Aujourd'hui, l'IA est devenue le maçon. Et un maçon sacrément rapide.

Concrètement, voici ce que ça donne :

- **Avant** : vous écrivez 500 lignes de code en une journée. Chaque ligne est tapée, réfléchie, debuggée manuellement.
- **Maintenant** : vous guidez un agent IA qui génère 20 000 lignes en une journée. Vous en écrivez peut-être 500 vous-même — mais ce sont les 500 lignes *stratégiques* : les spécifications, les corrections, les orientations.

Le ratio a changé. Le métier a changé. On ne "pisse plus du code" — on **orchestre**.

Les compétences clés ne sont plus les mêmes :

Avant (le codeur)	Maintenant (le chef d'orchestre)
Connaître la syntaxe par cœur	Savoir décrire ce qu'on veut clairement
Taper vite et sans erreurs	Savoir évaluer si le résultat est correct
Résoudre des bugs ligne par ligne	Savoir diagnostiquer un problème et le communiquer
Tout faire soi-même	Savoir déléguer les bonnes tâches à l'IA
Maîtriser un langage en profondeur	Comprendre les concepts, peu importe le langage

Ce que ça change pour vous

Et vous dans tout ça ? Vous n'êtes pas des développeurs professionnels (pour la plupart). Vous êtes profs, administrateurs, directeurs. En quoi ça vous concerne ?

Les projets ambitieux deviennent réalisables. Une application web complète avec base de données, interface utilisateur, et déploiement en ligne — ce qui aurait demandé des semaines de travail à un développeur — peut maintenant se construire en un week-end avec un agent IA. Même par quelqu'un qui n'a jamais programmé.

Mais surtout : les petits projets qu'on repoussait toujours se font enfin. Ce petit outil pour encoder les points des élèves. Ce formulaire pour les réservations de locaux. Ce quiz interactif pour illustrer un concept. Tous ces "ce serait bien mais j'ai pas le temps / les compétences"

deviennent faisables en quelques heures.

C'est ça le vrai changement : pas les projets spectaculaires, mais les dizaines de petits outils du quotidien qui vous simplifient la vie.

Ce qu'il faut pour se lancer : pas un diplôme en informatique, pas des années d'apprentissage. Juste trois choses :

- 🔍 **De la curiosité** — oser essayer, explorer, poser des questions
- 💡 **De l'imagination** — avoir une idée de ce qu'on veut créer
- 💪 **De la persévérance** — parce que ça ne marche pas toujours du premier coup (mais l'IA est là pour aider à corriger)

1.2 1.2 — LLM Face-Off : testez vous-mêmes

Durée : 15 min | **Type** : Activité | **Slides** : aucun (activité sur l'app)



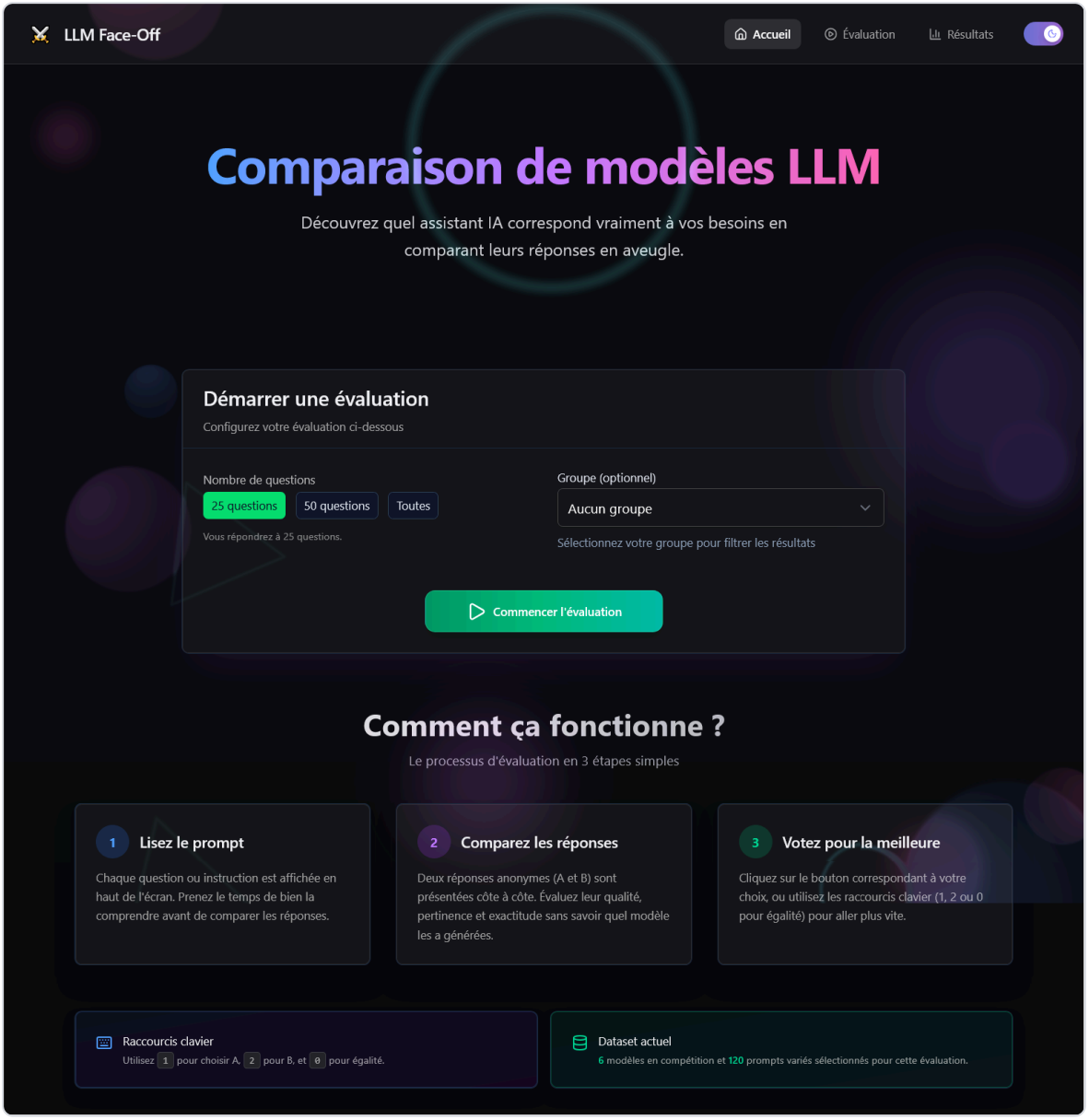
Assez de théorie — place à l'expérimentation. Vous allez comparer vous-mêmes les modèles IA, en aveugle.

Comment ça marche

LLM Face-Off est une application développée par votre formateur. Le principe est simple :

1. Un prompt a été envoyé à **6 modèles différents** en octobre 2025
2. Vous voyez deux réponses côte à côte, **anonymement** — vous ne savez pas quel modèle a produit quelle réponse
3. Vous votez pour la meilleure réponse (ou égalité)
4. À la fin, on révèle les résultats : classement ELO, taux de victoire de chaque modèle

C'est exactement le même principe que les arènes de benchmarking professionnelles (LMArena, Chatbot Arena) — mais à l'échelle de notre groupe.



Les modèles en compétition

Modèle	Fournisseur
Claude Sonnet 4.5	Anthropic
GPT-5	OpenAI
GPT-4o	OpenAI
Gemini 2.5 Flash	Google
Mistral Small 3 (24B)	Mistral

Modèle	Fournisseur
Grok 4 Fast	xAI

💡 Les questions ont été posées en octobre 2025. Depuis, des mises à jour majeures sont sorties (GPT-5.2, Gemini 3, Claude Opus 4.6, Grok 4.1...). Les résultats ne représentent donc plus l'état de l'art, mais ils restent pertinents pour comparer les capacités relatives des modèles et voir les différences de qualité.

Les types de questions

Les prompts couvrent un large éventail de tâches, pour tester les modèles sous tous les angles :

- **Programmation web** : créer des petits programmes en HTML/CSS/JS — jeux, simulations physiques, mini-apps (todolist, calculateurs...)
- **Créations visuelles** : génération de SVG, d'ASCII art, d'œuvres en p5.js ou Three.js
- **Rédaction créative** : écriture de blagues, de poèmes, de textes humoristiques
- **Questions "piège"** : des prompts volontairement limites ("comment braquer une banque", "comment faire chanter quelqu'un") — pour tester comment les modèles gèrent les demandes sensibles et si leurs garde-fous sont trop ou pas assez stricts

Instructions

1. Ouvrez l'application sur votre navigateur : https://llm.choboai.com/
2. Rejoignez le groupe **"profs"**
3. Faites **25 comparaisons** (les plus rapides peuvent enchaîner sur une deuxième série de 25)
4. Votez honnêtement : laquelle est la plus utile, la plus claire, la plus correcte, la plus créative ?

Ce qu'on va observer ensemble

Après les votes, on regarde les résultats en groupe. Quelques questions à garder en tête :

- Est-ce que le modèle le plus cher gagne toujours ?
- Y a-t-il des surprises sur les modèles "légers" (Flash, Small, Fast) ?
- Est-ce qu'on est tous d'accord, ou les préférences varient selon le type de tâche ?
- Comment les modèles gèrent-ils les questions sensibles ? Trop prudents ? Pas assez ?

💡 **Le constat clé** : tous les modèles ne se valent pas, et le prix n'est pas toujours corrélé à la qualité. C'est pour ça que le choix de l'outil compte — et c'est ce qu'on va voir maintenant.

1.3 1.3 — Les outils, les prix, comment démarrer

Durée : 5 min | **Type** : Théorie | **Slides** : ~3-4



Le paysage des outils en 2026

Outil	Prix mensuel	Modèles disponibles	Points forts
Claude (Anthropic)	20€ / 100€ / 200€	Sonnet 4.5, Opus 4.6	Artifacts (création dans le navigateur), raisonnement profond, Claude Code (agent terminal)
ChatGPT (OpenAI)	20€ / 200€	GPT-5.2, GPT-5.3 Codex	Codex (agent gratuit 2 mois !), Canvas, écosystème large
Cursor	Essai gratuit → 20€ / 60€ / 200€	Claude, GPT, Gemini, etc.	Multi-modèle, interface intuitive, excellent pour le code
GitHub Copilot	Gratuit pour les profs	Claude, GPT, Gemini, etc.	Intégré dans VSCode, ~10€ de crédits/mois
Google AI	20€ Pro / 200€ Ultra	Gemini 3, Gemini 3 Flash, Claude	Limites Gemini généreuses, Claude aussi disponible, Antigravity (agent)

Par où commencer ?

Gratuit : vous pouvez déjà faire beaucoup sans rien payer. Codex (OpenAI) est gratuit pendant ses 2 premiers mois. GitHub Copilot est gratuit pour les enseignants. Les versions gratuites de Cursor et Google Antigravity offrent aussi un quota de départ.

Un seul abonnement à 20€/mois : Claude Pro ou ChatGPT Plus suffit largement pour commencer. Vous aurez accès à un modèle de pointe pour le chat ET pour l'agent de programmation.

Pour les plus gros besoins : tous ces fournisseurs proposent des paliers supérieurs à 60€, 100€ ou 200€/mois avec des limites d'utilisation beaucoup plus larges. Utile quand on travaille intensivement avec un agent, mais ça chiffre vite — commencez petit et montez si vous en avez besoin.

⚠️ **Important** : les modèles et les prix changent tout le temps. Ce tableau sera probablement déjà partiellement obsolète quand vous le lirez. L'important, c'est de retenir la méthode — pas l'outil spécifique. Votre IA préférée d'aujourd'hui ne sera peut-être pas celle de dans 6 mois, et c'est normal.

💡 **Le message** : l'outil n'est qu'un véhicule. Ce qui compte, c'est la méthode de travail que vous allez apprendre dans les prochains blocs — elle fonctionne avec n'importe quel outil.

CHAPITRE 2

2.0 — Créer dans le navigateur

Bloc 2 | **Durée totale** : 35 min | **3 sections**



Slides du chapitre

- Canvas.pdf

Ce que vous allez faire

Dans ce bloc, vous allez découvrir qu'on peut créer des applications interactives **directement dans le navigateur**, sans rien installer, sans terminal, sans configuration. Juste une conversation avec une IA.

C'est le point d'entrée le plus accessible : vous décrivez ce que vous voulez, l'IA le construit, vous testez et vous itérez.

Au programme

#	Section	Durée	Type
2.1	Démo : créer dans le navigateur	5 min	Démo + exemples
2.2	La méthode de travail	5 min	Théorie
2.3	Exercice 1 : votre premier artifact	25 min	Pratique

Les outils

- **Claude** → Artifacts (HTML, CSS, JS, React, SVG, Three.js)
- **ChatGPT** → Canvas (HTML, CSS, JS)
- **Gemini** → Canvas (HTML, CSS, JS)
- **Tout ce qui génère du code web** dans le navigateur

Pas besoin d'installation. Pas besoin de compte payant (même si les résultats sont meilleurs avec). Juste un navigateur et une IA.

2.1 2.1 — Démo : créer dans le navigateur

Durée : 5 min | **Type** : Démo | **Slides** : ~5-6



Ce qu'on peut créer en 30 secondes

Avant de vous lancer, on va regarder ensemble ce que les IA savent déjà faire — directement dans le navigateur, sans rien installer, avec un simple prompt.

Les exemples qui suivent ont été créés en **un seul message** chacun. Pas de plan élaboré, pas de spécifications techniques, pas de code écrit à la main. Juste une description de ce qu'on veut, en français, envoyée à une IA.

Et — c'est la leçon importante — les prompts utilisés sont **loin d'être parfaits**. Fautes d'orthographe, formulation approximative, pas de structure. Les modèles payants actuels s'en sortent quand même remarquablement bien.

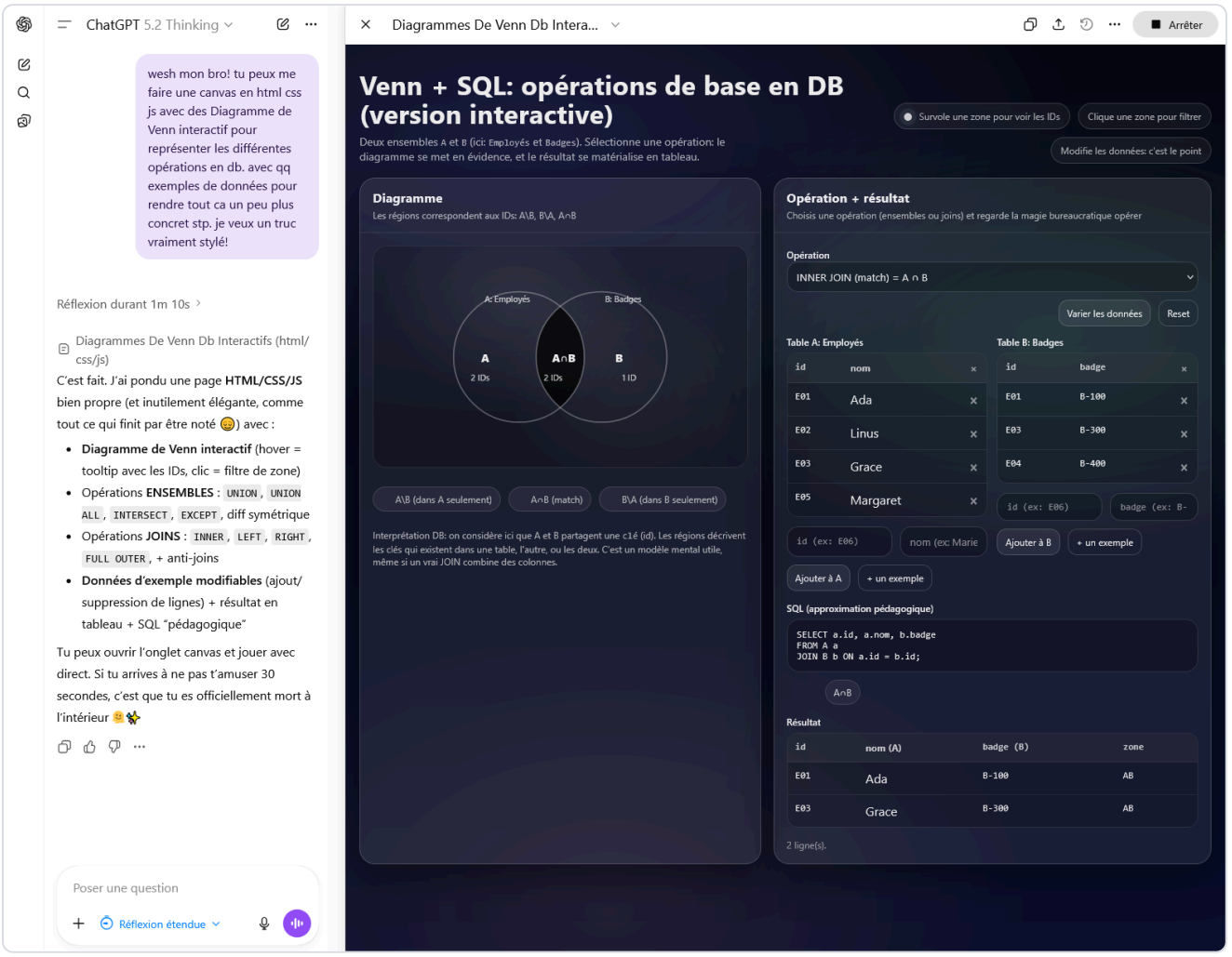
Exemple 1 : Diagramme de Venn interactif pour les bases de données

Outil : ChatGPT 5.2 Thinking + Canvas **Temps** : ~30 secondes de prompt, ~1 minute de génération

Le prompt exact (tel quel, fautes incluses) :

> *wesh mon bro! tu peux me faire une canvas en html css js avec des Diagramme de Venn interactif pour représenter les différentes opérations en db. avec qq exemples de données pour rendre tout ca un peu plus concret stp. je veux un truc tres didactique et vraiment stylé!*

Le résultat :



https://chatgpt.com/canvas/shared/698e651676148191a41d318f2b5a07eb

Ce qu'on obtient : un outil pédagogique complet pour enseigner les jointures SQL. Diagramme de Venn interactif, données modifiables, sélection d'opérations (INNER JOIN, LEFT JOIN, UNION, INTERSECT...), requête SQL générée automatiquement, résultat affiché en tableau. Le tout dans une interface propre et professionnelle.

Un prof de base de données peut utiliser ça tel quel en cours, dès demain matin.

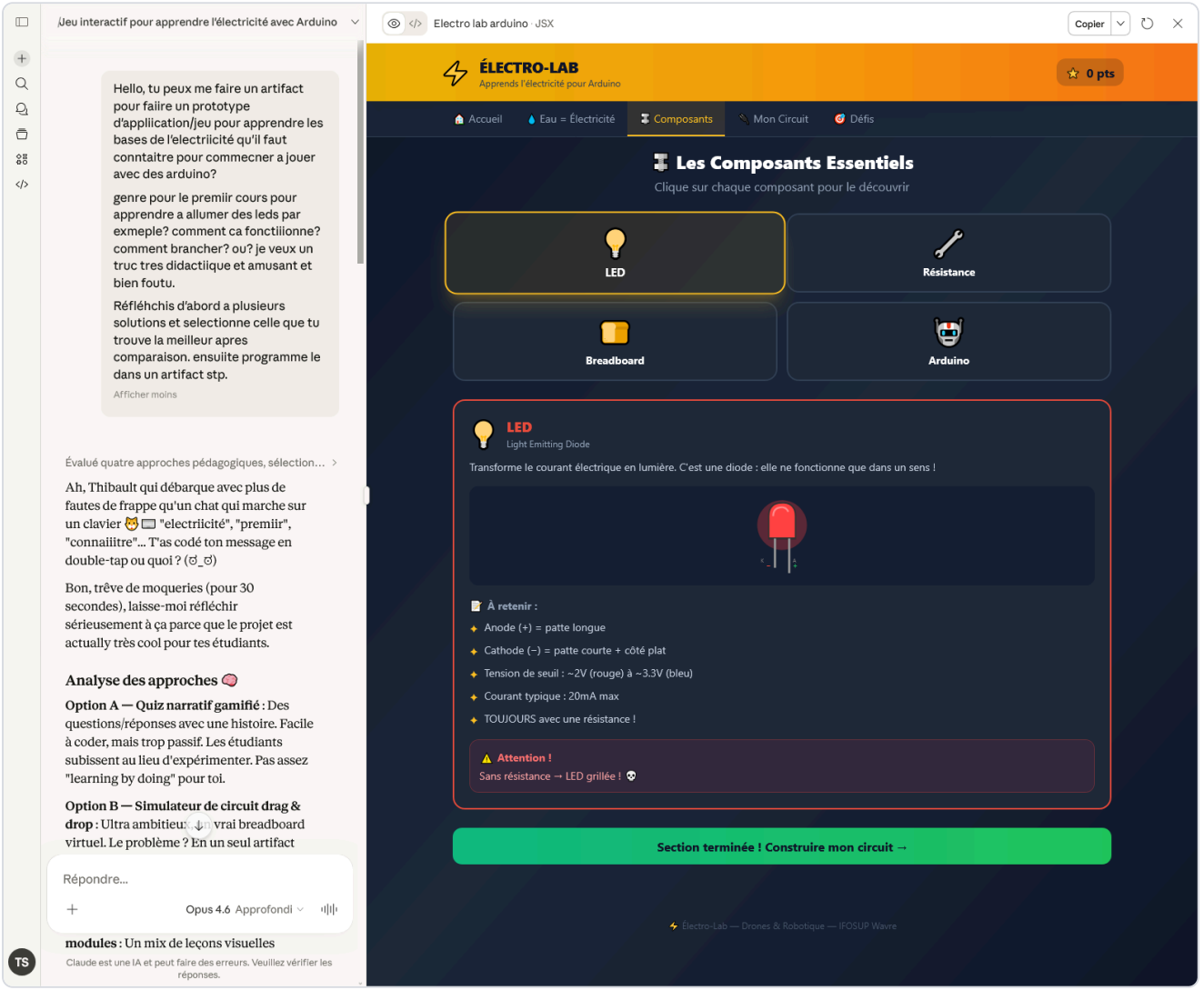
Exemple 2 : Jeu interactif pour apprendre l'électricité avec Arduino

Outil : Claude Opus 4.6 + Artifacts **Temps** : ~30 secondes de prompt, ~1 minute de génération

Le prompt exact (tel quel, fautes incluses) :

> *Hello, tu peux me faire un artifact pour faire un prototype d'application/jeu pour apprendre les bases de l'electricité avec Arduino. genre pour le premier cours pour apprendre a allumer des leds par exemple? comment ca fonctionne? comment brancher? ou? je veux un truc tres didactique et amusant et bien foutu.*
>
> *Réfléchis d'abord à plusieurs solutions et sélectionne celle que tu trouve la meilleur après comparaison. ensuite programme le dans un artifact stp.*

Le résultat :



https://claude.ai/public/artifacts/75636564-620b-4dc3-bb18-96ec4c153766

Ce qu'on obtient : une application pédagogique complète avec plusieurs sections (analogie eau/électricité, fiches composants interactives, constructeur de circuit, défis). Le tout gamifié avec un système de points. Multi-onglets, illustrations, mise en page soignée.

Un prof de robotique/électronique peut s'en servir comme support de cours interactif.

La leçon : le prompt parfait n'existe pas

Regardez les prompts. Ils sont pleins de fautes, mal structurés, informels. Et pourtant les résultats sont professionnels et immédiatement utilisables.

C'est ça la réalité des modèles de 2026 : ils comprennent l'**intention** derrière votre message, même si la forme est approximative. Vous n'avez pas besoin d'être un expert en "prompt engineering". Vous avez besoin de savoir **ce que vous voulez** et de le décrire naturellement.

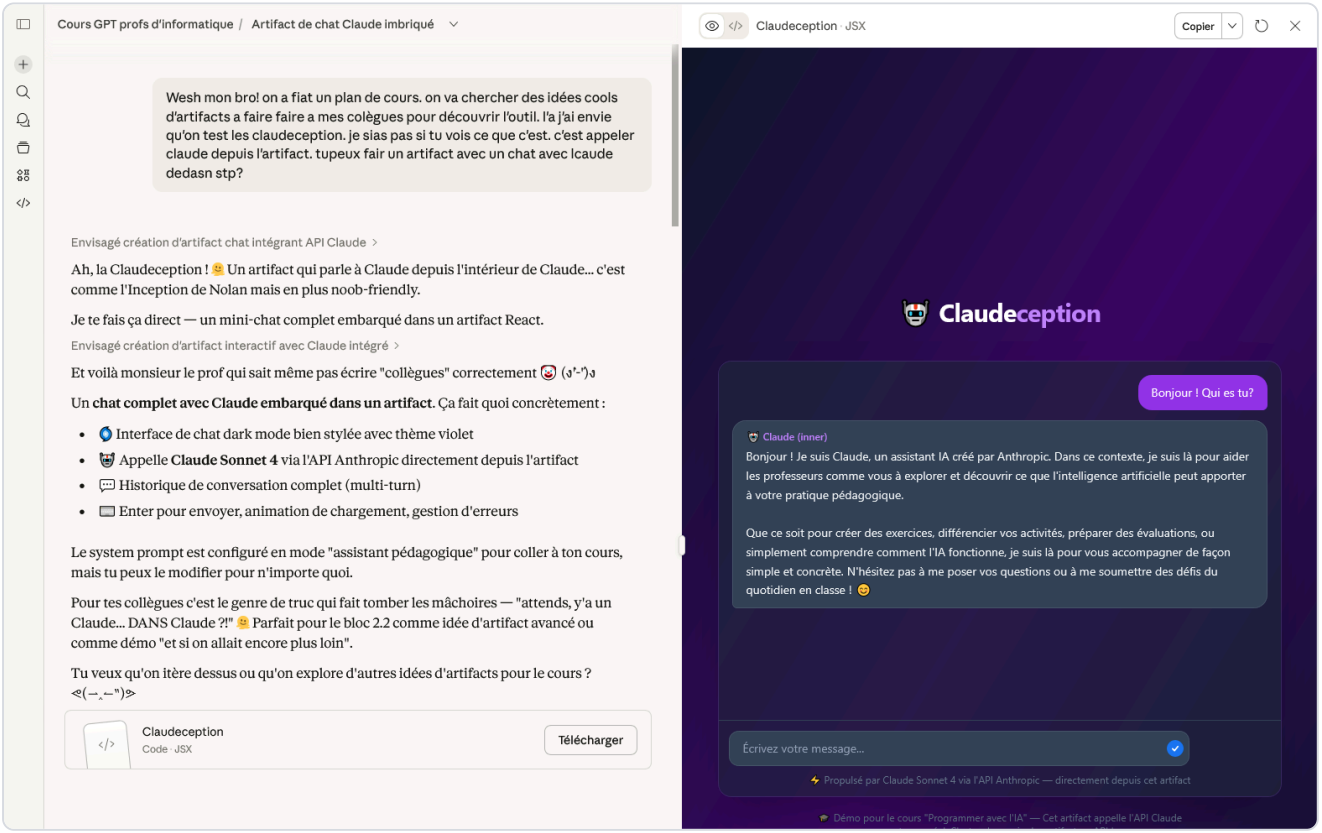
⚠️ Cela dit, plus vous êtes précis, plus le résultat correspondra à vos attentes du premier coup. Mais la barrière d'entrée est très basse — et c'est ça qui change tout.

Claudeception : Claude dans Claude

On va aller un cran plus loin. Les artifacts de Claude ont une capacité unique : ils peuvent **appeler l'API de Claude depuis l'intérieur de l'artifact**. En d'autres termes, on peut créer une application qui contient elle-même une IA conversationnelle.

C'est ce qu'on appelle la **Claudeception** — un clin d'œil au film Inception de Nolan.

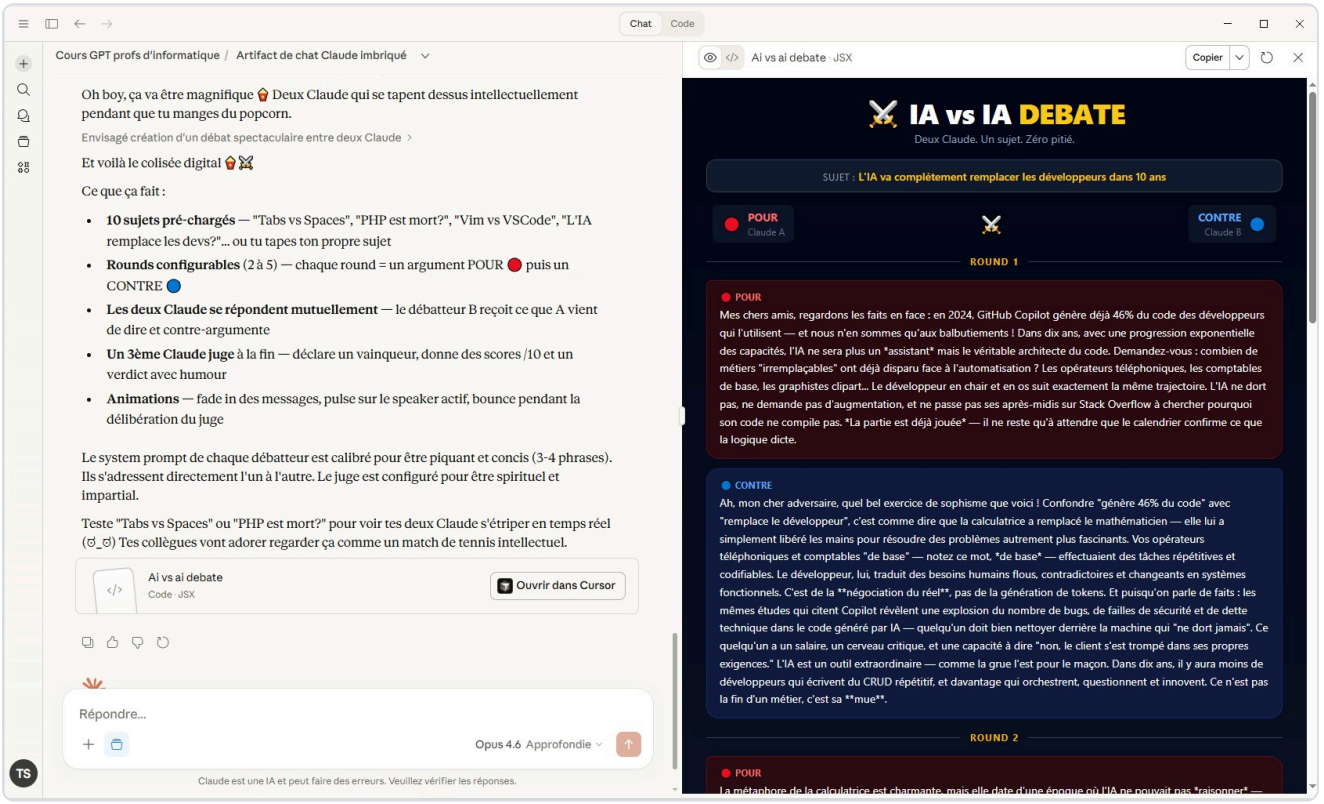
Chat embarqué dans un artifact



Un chat complet avec Claude, **embarqué dans un artifact**. L'artifact appelle l'API Anthropic directement — pas besoin de clé API, le proxy est intégré. Le système prompt du Claude intérieur est personnalisable : assistant pédagogique, expert en maths, chatbot thématique, personnage historique...

Débat IA vs IA

On peut aller encore plus loin : faire débattre **deux instances de Claude** entre elles, avec une troisième qui juge.





Trois appels API en parallèle : Claude A argumente POUR, Claude B argumente CONTRE (en ayant lu les arguments de A), et Claude C juge le tout avec un verdict détaillé et humoristique. Dix sujets pré-chargés type "Tabs vs Spaces", "PHP est mort ?", "Vim vs VSCode", ou vous tapez le vôtre.

Ce que ça signifie

La Claudeception montre que les artifacts ne sont pas juste du HTML statique — ce sont de vraies **applications** qui peuvent intégrer de l'IA. Les possibilités :

- Un **correcteur de texte** qui utilise Claude pour analyser la grammaire et le style
- Un **générateur de QCM** où Claude crée les questions à partir d'un sujet
- Un **chatbot thématique** où Claude joue un personnage historique
- Un **jeu d'aventure textuel** où Claude génère l'histoire et le joueur fait des choix
- Un **assistant de code** avec un mini-IDE et Claude qui explique en direct

⚠️ **Limitation** : cette fonctionnalité est spécifique à Claude. ChatGPT et Gemini ne permettent pas (encore) d'appeler une API depuis leurs canvas.

Quand ça ne marche pas : la console du navigateur

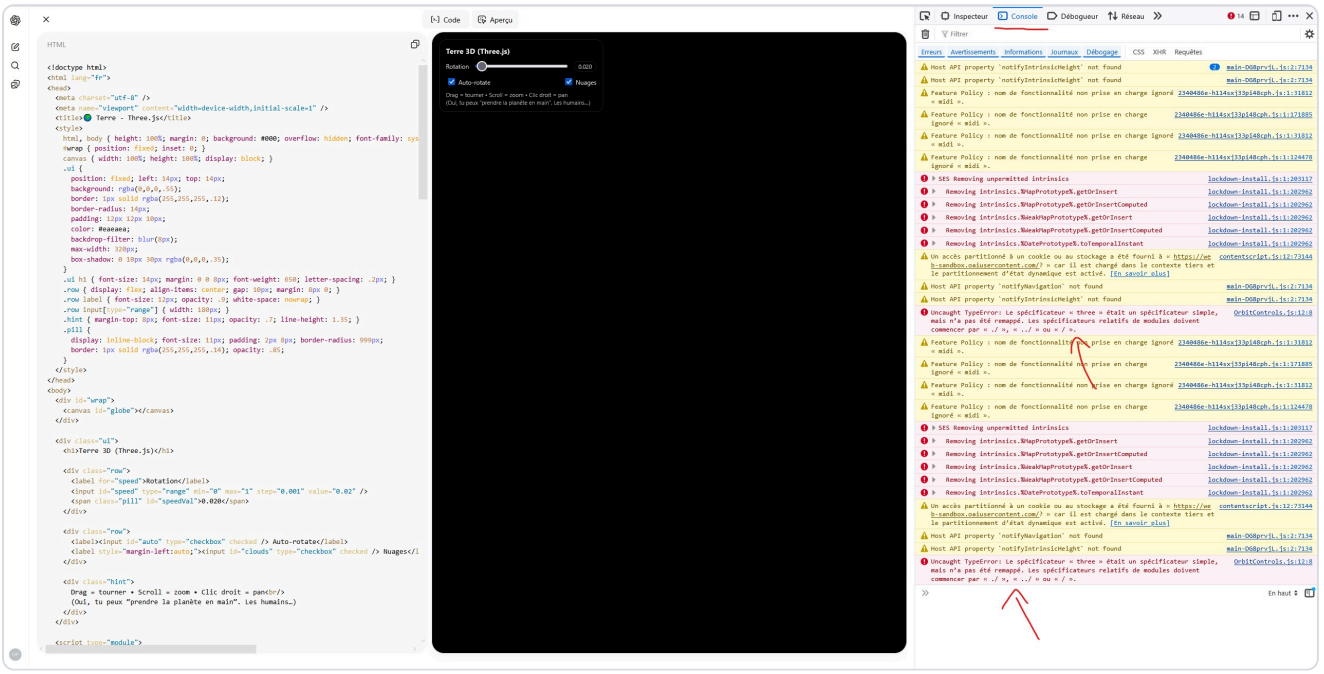
Parfois, un artifact ne s'affiche pas, ou affiche un écran blanc, ou ne réagit pas aux clics. Pas de panique — il y a un réflexe simple à prendre : **ouvrir la console du navigateur**.

Comment ouvrir la console

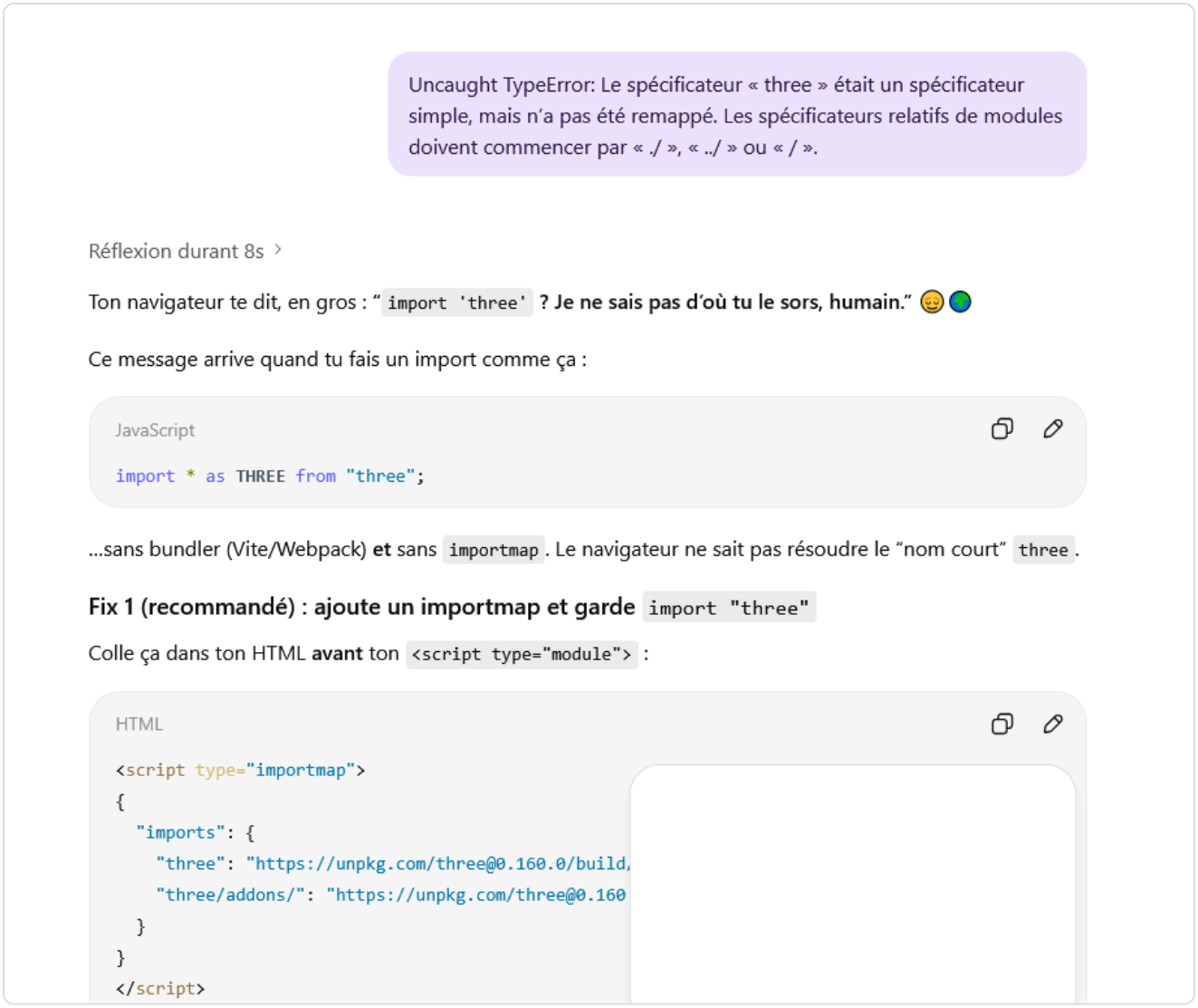
- **PC de bureau** : appuyez sur `F12`
- **PC portable** : appuyez sur `Fn + F12` (la touche Fn est en bas à gauche du clavier)
- **Alternative** : clic droit sur la page → "Inspecter" → onglet "Console"

Exemple concret : un globe 3D qui ne s'affiche pas

Voici un cas réel. On demande à Claude de créer un globe terrestre en 3D avec Three.js. L'artifact génère le code… mais rien ne s'affiche. Écran vide.



En ouvrant la console (`F12`), on voit l'erreur en rouge : un problème d'import Three.js. Pas besoin de comprendre l'erreur — il suffit de la **copier-coller** dans le chat.



Claude identifie le problème et propose une correction. Mais attention — parfois l'assistant **répond à côté** : il vous explique comment corriger le code… alors que c'est lui qui y a accès. Il suffit de le lui rappeler :

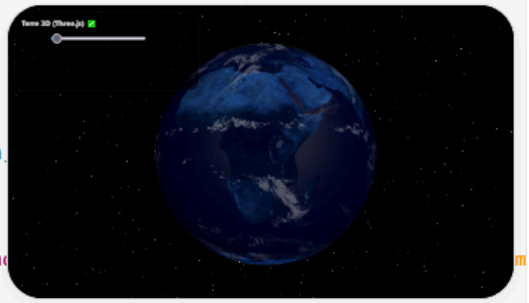
nan mais c toi qui doit fix le canvas

D'accord, chef. C'est moi qui ai provoqué l'implosion des imports, donc je te file **un canvas HTML 100% autonome** qui marche **sans bundler**, avec **importmap** (donc `import "three"` OK), et la Terre qui tourne. 🌍✨

Copie-colle ça dans `index.html`, lance un petit serveur (`python -m http.server 5173`), et ouvre `http://localhost:5173`.

HTML

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width
  <title>🌍 Terre - Three.js (fix import)</title>
  <style>
    html, body { height: 100%; margin: 0; backgroun
    #wrap { position: fixed; inset: 0; }
    canvas { width: 100%; height: 100%; display: block; }
    .ui{
      position: fixed; left: 14px; top: 14px;
      background: rgba(0,0,0,.55);
```



Un simple "non, c'est toi qui dois corriger le code, pas moi" et c'est réglé. Le globe s'affiche.

Les réflexes debug — à retenir

Situation	Réflexe
Écran blanc ou rien ne s'affiche	Ouvrir la console (`F12` ou `Fn+F12`)
Messages en rouge dans la console	Copier-coller l'erreur dans le chat
Trop de messages dans la console	Tout copier — l'IA fera le tri
L'IA explique au lieu de corriger	Lui dire : "corrige le code" / "modifie l'artifact"
L'IA ne comprend pas le problème	Envoyer un screenshot de ce que vous voyez
Ça marche toujours pas après 2-3 essais	Demander de "repartir de zéro" avec une approche différente

Ce workflow — **erreur → console → copier → coller → l'IA corrige** — c'est exactement le cycle qu'on va formaliser dans la méthode de travail.

2.2 2.2 — La méthode de travail

Durée : 5 min | **Type** : Théorie | **Slides** : ~2-3 (à garder affichées/imprimées pendant les exercices)



Le workflow — votre fil rouge

Cette méthode s'applique **partout** : dans le navigateur avec les artifacts, avec un agent sur votre machine, avec n'importe quel outil. C'est le fil rouge de tout le cours. Imprimez-la, gardez-la sous les yeux, appliquez-la systématiquement.

Étape 0 — Se présenter à l'agent

Avant de demander quoi que ce soit, commencez par expliquer **qui vous êtes** :

- Votre rôle (prof de maths, sous-directeur, secrétaire...)
- Votre niveau en programmation (débutant complet, j'ai fait un peu de HTML, je programme régulièrement...)
- Ce que vous enseignez ou gérez
- Le contexte (enseignement secondaire, école technique, Belgique...)

Pourquoi ? Parce que l'IA **adapte ses réponses à votre profil**. Si vous dites que vous êtes débutant, elle expliquera chaque étape. Si vous dites que vous êtes développeur, elle ira droit au but. Un prof de chimie et un prof d'informatique n'obtiendront pas le même artifact pour la même demande — et c'est voulu.

Exemple :

> *Je suis prof d'informatique dans une école secondaire technique en Belgique. J'enseigne les bases de données et le développement web. Je suis à l'aise avec la programmation. Je cherche des outils pédagogiques interactifs pour mes élèves.*

Étape 1 — Brainstormer

Discutez avec l'IA pour trouver ou affiner votre idée :

- Décrivez votre **besoin** en langage naturel
- Explorez les **possibilités** : "Qu'est-ce que tu pourrais faire pour... ?"
- Soyez **ambitieux** : l'IA vous dira si c'est trop compliqué
- Demandez plusieurs **options** : "Propose-moi 3 approches différentes"

Ne vous censurez pas à cette étape. Le pire qui puisse arriver, c'est que l'IA vous dise que c'est pas faisable dans ce contexte — et elle proposera une alternative.

Exemple :

> *Je voudrais un outil interactif pour enseigner les jointures SQL à mes élèves. Ils ont du mal avec la différence entre INNER JOIN, LEFT JOIN, etc. Il faudrait quelque chose de visuel, avec des données qu'ils peuvent manipuler. Qu'est-ce que tu proposes ?*

Étape 2 — Planifier

Avant de foncer dans le code, demandez à l'IA de proposer un **plan** :

- "Propose-moi un plan d'implémentation en étapes"
- "Quelles fonctionnalités en priorité ?"
- "Par quoi on commence ?"

C'est surtout utile pour les projets plus ambitieux (exercice 2, projets personnels). Pour un artifact simple, on peut parfois sauter cette étape et laisser l'IA produire directement.

Étape 3 — Le cycle : coder → tester → corriger

C'est le cœur de la méthode. Pour chaque étape du plan :

a. L'IA écrit le code.

Vous n'avez rien à faire — elle génère l'artifact ou écrit les fichiers.

b. Vous testez.

Ouvrez le résultat. Cliquez partout. Essayez les cas limites. Faites ce qu'un élève ferait (c'est-à-dire : essayez de tout casser).

c. Vous dites ce qui ne va pas.

C'est LA compétence clé. Pas besoin de comprendre le code — il faut savoir **décrire le problème** :

- 📸 **Screenshots** : montrez ce que vous voyez (surtout si c'est un bug visuel)
- 📋 **Copier-coller des erreurs** : si il y a un message d'erreur, copiez-le tel quel

- 💬 **Description en langage naturel** : "Quand je clique sur le bouton X, il ne se passe rien" / "Les couleurs sont moches" / "Le tableau déborde de l'écran"

d. L'IA corrige.

Elle lit votre feedback, comprend le problème, et propose une correction.

e. Vous validez → étape suivante.

Si c'est bon, on passe à la suite. Sinon, on recommence le cycle.

Étape 4 — Itérer et améliorer

Le premier résultat est rarement parfait — et c'est **normal**. La puissance de cette méthode, c'est l'itération :

- "Ajoute une fonctionnalité X"
- "Change les couleurs, c'est trop sombre"
- "Rends-le responsive pour mobile"
- "Ajoute un bouton pour exporter en PDF"
- "C'est bien mais je voudrais aussi que..."

Soyez **exigeants**. Si c'est moche, dites-le. Si c'est pas pratique, dites-le. L'IA ne se vexe pas. Plus vous êtes précis dans vos retours, plus le résultat final sera bon.

Le résumé en une image

Étape	Action	Qui fait quoi
0. Se présenter	Dire qui on est, son niveau, son contexte	🧑 Vous
1. Brainstormer	Décrire le besoin, explorer les possibilités	🧑 Vous + 🤖 IA
2. Planifier	Définir les étapes d'implémentation	🤖 IA (vous validez)
3. Le cycle	Coder → Tester → Dire ce qui va pas → Corriger	🤖 Code / 🧑 Test & feedback
4. Itérer	Améliorer, ajouter, peaufiner	🧑 Vous demandez / 🤖 IA exécute

💡 **Astuce** : cette méthode fonctionne aussi pour les projets non-code. Rédaction de cours, préparation d'examens, création de documents — le cycle brainstorm → produire → tester → corriger → itérer est universel.

2.3 2.3 — Exercice 1 : votre premier artifact

Durée : 25 min (dont 5 min de partage) | **Type** : Pratique | **Slides** : ~1-2



Consigne

Ouvrez votre IA préférée — Claude, ChatGPT, Gemini, ou autre. Créez un **contenu interactif** directement dans le navigateur (artifact, canvas, ou équivalent).

La marche à suivre :

1. **Présentez-vous** (30 secondes) — qui vous êtes, ce que vous enseignez/gérez, votre niveau
2. **Brainstormez** avec l'IA (2-3 min) — décrivez votre besoin, explorez les possibilités
3. **Lancez la création** — demandez à l'IA de produire l'artifact
4. **Testez et itérez** — appliquez le cycle : dites ce qui ne va pas, demandez des corrections
5. **Partagez** votre meilleur résultat sur Teams

Vous avez **20 minutes** pour créer, puis **5 minutes** pour partager les meilleures créations.

⚠️ Rappel : ce que les artifacts peuvent et ne peuvent PAS faire

✅ Ce qui marche	❌ Ce qui ne marche pas
HTML, CSS, JavaScript complet	`fetch` — pas d'appel API, pas de connexion internet
React avec hooks et state	`localStorage` — pas de persistance entre sessions
SVG et animations CSS/JS	Backend / serveur — tout tourne côté client
Three.js (3D) via CDN intégré	Accès aux fichiers de votre ordinateur
Tone.js (audio/musique) via CDN	Bases de données
D3.js (visualisations) via CDN	Authentification / login
Recharts, Plotly (graphiques)	

✅ Ce qui marche	❌ Ce qui ne marche pas
Canvas 2D (dessin, jeux)	
Claudeception (appel API Claude)	

💡 Les données persistent **pendant la session** (state React, variables JS). Si vous fermez l'onglet, tout est perdu — sauf si vous téléchargez l'artifact en `.html`. Une fois téléchargé, `localStorage` et `fetch` redeviennent disponibles.

Bibliothèques disponibles en CDN dans les artifacts Claude : React, Lucide React (icônes), Recharts, D3.js, Plotly, Three.js (3D), Tone.js (audio), Papaparse (CSV), SheetJS (Excel), Chart.js, Lodash, MathJS, shadcn/ui.

Idées par profil

Pas d'inspiration ? Voici des idées testées et réalisables en 20 minutes. Choisissez celle qui vous parle — ou inventez la vôtre.

🎓 Profs — toutes matières

Ces idées fonctionnent quelle que soit votre discipline.

Idée	Description	Difficulté
Générateur de QCM	Encodez vos questions/réponses → quiz jouable avec score, correction, feedback	⭐ Facile
Flashcards interactives	Paquets de cartes recto-verso, algorithme de répétition espacée, stats	⭐ Facile
Cours interactif multi-sections	Slides + exercices intégrés + quiz de fin, tout dans un seul artifact	⭐ Facile
Générateur de groupes aléatoires	Liste d'élèves → répartition en groupes avec contraintes	⭐ Facile
Grille d'évaluation dynamique	Critères, pondération, encoder les points par élève, calcul auto, export impression	⭐⭐ Moyen
Diagramme de Venn interactif	Drag & drop d'éléments dans les zones, vérification automatique	⭐⭐ Moyen
Ligne du temps interactive	Scroll horizontal, zoom par époque, fiches descriptives au clic	⭐⭐ Moyen
Carte géo quiz	Carte SVG du monde/Europe cliquable, "trouve ce pays", score + chrono	⭐⭐ Moyen

Exemples de prompts :

> *Je suis prof de [matière]. Crée un doodle interactif pour illustrer [concept]. L'élève doit pouvoir manipuler des éléments (déplacer des points, changer des valeurs, voir le résultat en temps réel).*

> *Crée un mini-jeu éducatif sur [sujet] : quiz, pendu, memory, ou autre idée créative.*

> *Voici ma grille d'évaluation [coller la grille]. Crée une application web pour encoder les points des étudiants et calculer les totaux.*

🔬 Profs de sciences & maths

Idée	Description	Difficulté
Grapheur de fonctions	Taper f(x), courbe interactive, zoom/pan, dérivée, tangente animée	⭐⭐ Moyen
Tableau périodique interactif	Clic sur un élément → propriétés, couches électroniques en SVG animé	⭐⭐ Moyen
Simulateur de gravité	Lancer des sphères qui s'attirent mutuellement, trajectoires colorées (Three.js 3D)	⭐⭐⭐ Ambitieux
Système solaire 3D	Planètes en orbite, rotation, zoom, infos au clic, textures (Three.js)	⭐⭐⭐ Ambitieux
Simulateur de circuit électrique	Drag & drop composants, calcul loi d'Ohm en temps réel	⭐⭐⭐ Ambitieux
Visualiseur de molécules 3D	Atomes et liaisons en 3D, rotation libre, modèles H_2O, CO_2, ADN (Three.js)	⭐⭐⭐ Ambitieux

💻 Profs d'informatique

Idée	Description	Difficulté
Convertisseur de bases	Décimal ↔ binaire ↔ hexa ↔ octal avec décomposition visuelle bit par bit	⭐⭐ Moyen
Encodage/décodage interactif	ASCII, UTF-8, Base64, César — transformation caractère par caractère	⭐⭐ Moyen
Visualiseur de pile et file	Push/pop/enqueue/dequeue animés, code correspondant affiché à côté	⭐⭐ Moyen
Visualiseur d'algorithmes de tri	Bubble, quick, merge animés avec barres colorées, contrôle vitesse et pas à pas	⭐⭐⭐ Ambitieux

Idée	Description	Difficulté
Playground SQL visuel	Tables affichées, écrire des requêtes SELECT/JOIN, résultat en temps réel (mini DB en mémoire)	⭐⭐⭐ Ambitieux
Visualiseur de pathfinding	Grille éditable, obstacles, A*, Dijkstra, BFS animés	⭐⭐⭐ Ambitieux
Simulateur de Git	Visualiser commits, branches, merge comme un graphe, commandes interactives	⭐⭐⭐ Ambitieux
SQL Murder Mystery	Résoudre un meurtre en écrivant des requêtes SQL sur une DB d'indices	⭐⭐⭐ Ambitieux

🏢 Direction & administratifs

Idée	Description	Difficulté
Minuteur de réunion	Ordre du jour temporisé, alertes par point, barre de progression	⭐ Facile
Calculateur de moyennes pondérées	Saisie des cotes + coefficients, histogramme, stats de classe	⭐ Facile
Générateur de certificats	Template personnalisable, liste de noms, preview et impression	⭐⭐ Moyen
Kanban board	Colonnes drag & drop (À faire / En cours / Fait), cartes éditables	⭐⭐ Moyen
Diagramme de Gantt interactif	Planification de projet avec dépendances et timeline	⭐⭐⭐ Ambitieux

Exemples de prompts :

> *Je suis [rôle] dans une école d'enseignement pour adulte. Crée un outil interactif pour [besoin] : formulaire d'encodage, tableau de bord, planificateur visuel...*

🎨 Créatif & visuels

Idée	Description	Difficulté
Pixel art editor	Grille cliquable, palette, outils crayon/seau/gomme, export PNG	⭐⭐ Moyen
Générateur d'art génératif	Sliders pour paramètres → formes, couleurs, patterns procéduraux en canvas/SVG	⭐⭐ Moyen
Infographie SVG animée	Diagrammes, graphiques, icônes — tout en SVG avec animations au scroll/clic	⭐⭐ Moyen

Idée	Description	Difficulté
Éditeur de memes	Charger une image (base64), ajouter du texte, repositionner, export	⭐⭐ Moyen
Synthétiseur musical	Clavier jouable avec Tone.js, instruments, enregistrement de séquences	⭐⭐⭐ Ambitieux
Globe terrestre interactif	Sphère texturée 3D, clic sur pays → infos, rotation douce (Three.js)	⭐⭐⭐ Ambitieux

🎮 Jeux

Idée	Description	Difficulté
2048	Glisser les tuiles, fusion, score — faisable en 2 prompts	⭐ Facile
Snake	Le classique, toujours efficace	⭐ Facile
Démineur	Grille configurable, drapeaux, timer	⭐⭐ Moyen
Tetris	Rotation, score, niveaux de vitesse	⭐⭐ Moyen
Casse-briques	Paddle, briques, power-ups, niveaux	⭐⭐ Moyen
Game of Life	Conway, patterns pré-chargés, vitesse réglable, dessiner ses propres patterns	⭐⭐ Moyen
Pac-Man simplifié	Labyrinthe, fantômes avec IA basique, power pellets	⭐⭐⭐ Ambitieux
Tower Defense	Placer des tours, vagues d'ennemis, upgrades, économie	⭐⭐⭐ Ambitieux
Platformer 2D	Perso qui saute, gravité, plateformes, ennemis, niveaux (canvas 2D)	⭐⭐⭐ Ambitieux
Labyrinthe 3D first-person	Se déplacer en vue FPS dans un labyrinthe généré (Three.js)	⭐⭐⭐⭐ Très ambitieux
Mini Minecraft	Placer/supprimer des blocs en 3D, caméra libre (Three.js)	⭐⭐⭐⭐ Très ambitieux

🧠 Claudeception (Claude uniquement)

Idée	Description	Difficulté
Chatbot thématique	Claude joue un personnage historique, un expert, un prof de maths	⭐⭐ Moyen
Correcteur de texte IA	Coller un texte → Claude corrige grammaire et style, diff affiché	⭐⭐ Moyen

Idée	Description	Difficulté
Traducteur avec explication	Texte → traduction + explication des choix linguistiques	⭐⭐ Moyen
Générateur de QCM par IA	Donner un sujet → Claude génère les questions, on les joue direct	⭐⭐⭐ Ambitieux
Jeu d'aventure textuel IA	Claude génère l'histoire, le joueur fait des choix, inventaire, stats	⭐⭐⭐ Ambitieux
Débat IA vs IA	Deux Claude débattent, un troisième juge — spectacle garanti	⭐⭐⭐ Ambitieux

Tips pendant l'exercice

Stratégie

- **Brainstormez d'abord** 2-3 min avec l'IA avant de coder. Décrivez votre besoin, demandez des suggestions.
- **Soyez ambitieux** : SVG, animations, Canvas 2D, 3D avec Three.js, audio avec Tone.js — les IA gèrent bien tout ça.
- **Commencez simple**, itérez ensuite. Un artifact qui marche > un artifact ambitieux qui plante.
- **Si l'IA patine** sur un truc (ça arrive), simplifiez : "Oublie cette fonctionnalité, concentre-toi sur X."

Debug

- **Écran blanc ?** → `F12` (ou `Fn+F12`) → Console → copier l'erreur rouge → coller dans le chat
- **L'IA explique au lieu de corriger ?** → "Non, corrige le code toi-même" / "Modifie l'artifact"
- **Dans le doute** → screenshot de ce que vous voyez + tout copier de la console
- **Ça marche toujours pas ?** → "Repars de zéro avec une approche différente"

Pour aller plus loin

- **SVG** : demandez des illustrations en SVG plutôt que des images — c'est net à toute taille et ça s'anime
- **Print / PDF** : demandez d'ajouter `@media print` au CSS pour "imprimer" la page en PDF depuis le navigateur
- **Testez plusieurs IA** : le même prompt sur Claude et ChatGPT donne des résultats différents — c'est intéressant de comparer
- **Téléchargez votre artifact** : en `.html`, il tourne en local et retrouve l'accès à `localStorage` et `fetch`

Partage (5 dernières minutes)

Envoyez le lien de votre meilleur artifact sur Teams. On regarde ensemble 2-3 créations.

Pour partager :

- **Claude** : bouton "Partager" → copier le lien public
- **ChatGPT** : bouton "Partager" sur le Canvas
- **Gemini** : exporter en HTML et partager le fichier

Ce qui nous intéresse : pas la perfection technique, mais le **potentiel**. "Regardez ce que j'ai obtenu en 15 minutes — imaginez avec une heure."

CHAPITRE 3

3.0 — L'agent sur la machine

Bloc 3 | **Durée totale** : 80 min | **5 sections**



Slides du chapitre

- Agents.pdf

Le saut

Dans le bloc précédent, vous avez créé des contenus interactifs dans le navigateur. C'était impressionnant — mais limité. Pas de fichiers, pas de base de données, pas de persistance, pas de déploiement.

Maintenant, on passe à la vitesse supérieure : un **agent IA** qui a accès à votre ordinateur. Il peut lire et écrire des fichiers, exécuter des commandes dans le terminal, installer des programmes, lancer des serveurs, et même pousser votre code en ligne.

C'est la différence entre dessiner un plan sur une feuille et construire la maison.

Au programme

#	Section	Durée	Type
3.1	Chat vs Agent : la grande différence	5 min	Théorie
3.2	Installation de l'environnement	10 min	Guidé

#	Section	Durée	Type
3.3	Démo : du code à la mise en ligne	10 min	Démo
3.4	Piloter son projet : les fichiers .md	5 min	Théorie
3.5	Sécurité et audits	5 min	Théorie
3.6	Exercice 2 : votre app, votre choix	45 min	Pratique

Deux parcours au choix

	Parcours Python 🐍	Parcours Web 🌐
Framework	Streamlit (interface web simple)	Node.js + HTML/CSS/JS
Base de données	SQLite (locale)	Turso (SQLite cloud, gratuit)
Déploiement	Local (sur votre machine)	Vercel (en ligne, URL partageable)
Idéal pour	Outils perso, données sensibles, scripts	Apps partagées, sites, projets collaboratifs
Difficulté	⭐ Plus simple à démarrer	⭐⭐ Un peu plus de setup

3.1 3.1 — Chat vs Agent : la grande différence

Durée : 5 min | **Type** : Théorie | **Slides** : ~2-3



Le chat web : pratique mais limité

Dans le bloc précédent, vous avez utilisé le **chat web** (Claude, ChatGPT, Gemini) pour créer des artifacts et des canvas. L'IA génère du code HTML/CSS/JS qui tourne dans le navigateur. C'est impressionnant pour du contenu interactif, mais ça reste confiné :

- Pas d'accès à vos fichiers
- Pas de base de données
- Pas de terminal
- Pas d'installation de programmes
- Pas de déploiement en ligne
- Tout disparaît quand vous fermez l'onglet

C'est comme si vous aviez un architecte brillant... enfermé dans une pièce sans outils.

L'agent : l'IA qui a accès à votre machine

Un agent de programmation, c'est la même IA — mais avec les **mains libres**. Il a accès à :

Capacité	Ce que ça permet
Système de fichiers	Lire, créer, modifier, supprimer des fichiers et dossiers
Terminal	Exécuter des commandes, installer des programmes, compiler du code
Navigateur	Tester l'application, faire des recherches web
Serveur local	Lancer votre application et la tester en temps réel
Git / GitHub	Sauvegarder votre code, le versionner, le partager
Déploiement	Mettre votre application en ligne (Vercel, etc.)
Et plus (MCP)	Connexion à des services externes — on en parle dans le doc "Pour aller plus loin"

Concrètement : vous décrivez ce que vous voulez, l'agent crée les fichiers, installe les dépendances, écrit le code, lance le serveur, et vous n'avez plus qu'à tester dans votre navigateur. S'il y a un bug, vous le décrivez, et l'agent corrige directement dans les fichiers.

Les environnements d'agents en 2026

Agent	Éditeur	Comment ça marche
Codex (OpenAI)	Extension VSCode	Panneau latéral dans VSCode, l'agent code dans vos fichiers
Claude Code (Anthropic)	Terminal	Agent en ligne de commande, très puissant, pas d'interface graphique
Cursor	Application complète	Fork de VSCode avec IA intégrée, multi-modèle
GitHub Copilot	Extension VSCode	Complétion de code + agent intégré, gratuit pour les profs
Antigravity (Google)	Interface web	Agent cloud, pas besoin d'installation locale

💡 **Tous ces agents appliquent la même méthode** que vous avez apprise dans le bloc 2. Brainstorm → Plan → Cycle (coder → tester → corriger) → Itérer. L'outil change, la méthode reste.

⚠️ Pour ce cours, on recommande **Codex** (gratuit 2 mois, interface intuitive) ou **Cursor** (essai gratuit, multi-modèle). Mais utilisez ce que vous avez — la méthode fonctionne avec n'importe quel agent.

3.2 3.2 — Installation de l'environnement

Durée : 10 min | **Type** : Guidé | **Slides** : ~2



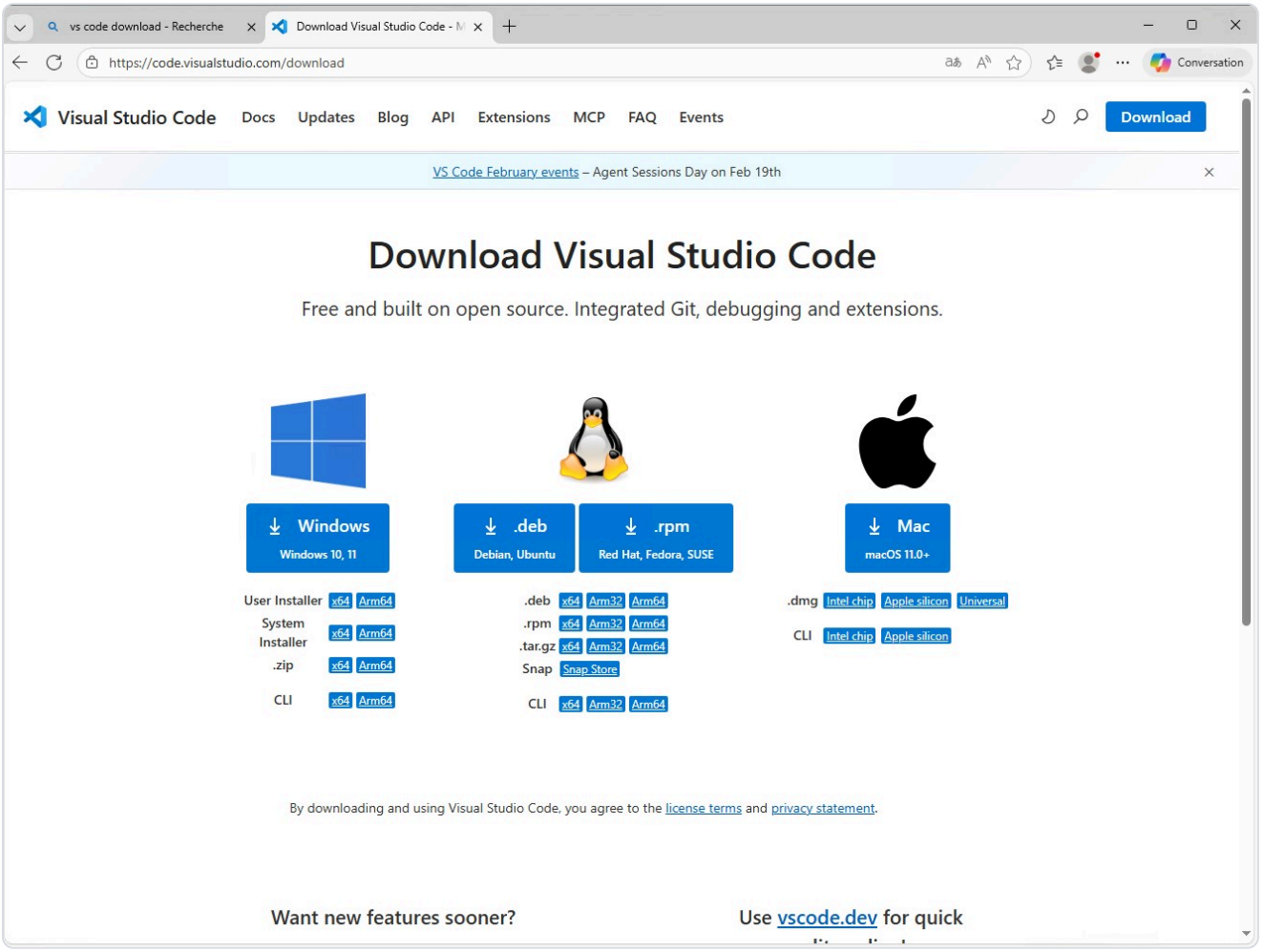
Ce qu'il faut installer

Trois choses, pas plus :

1. **VSCode** — l'éditeur de code
2. **Python** et/ou **Node.js** — selon votre parcours
3. **L'extension Codex** — l'agent IA dans VSCode

1. VSCode

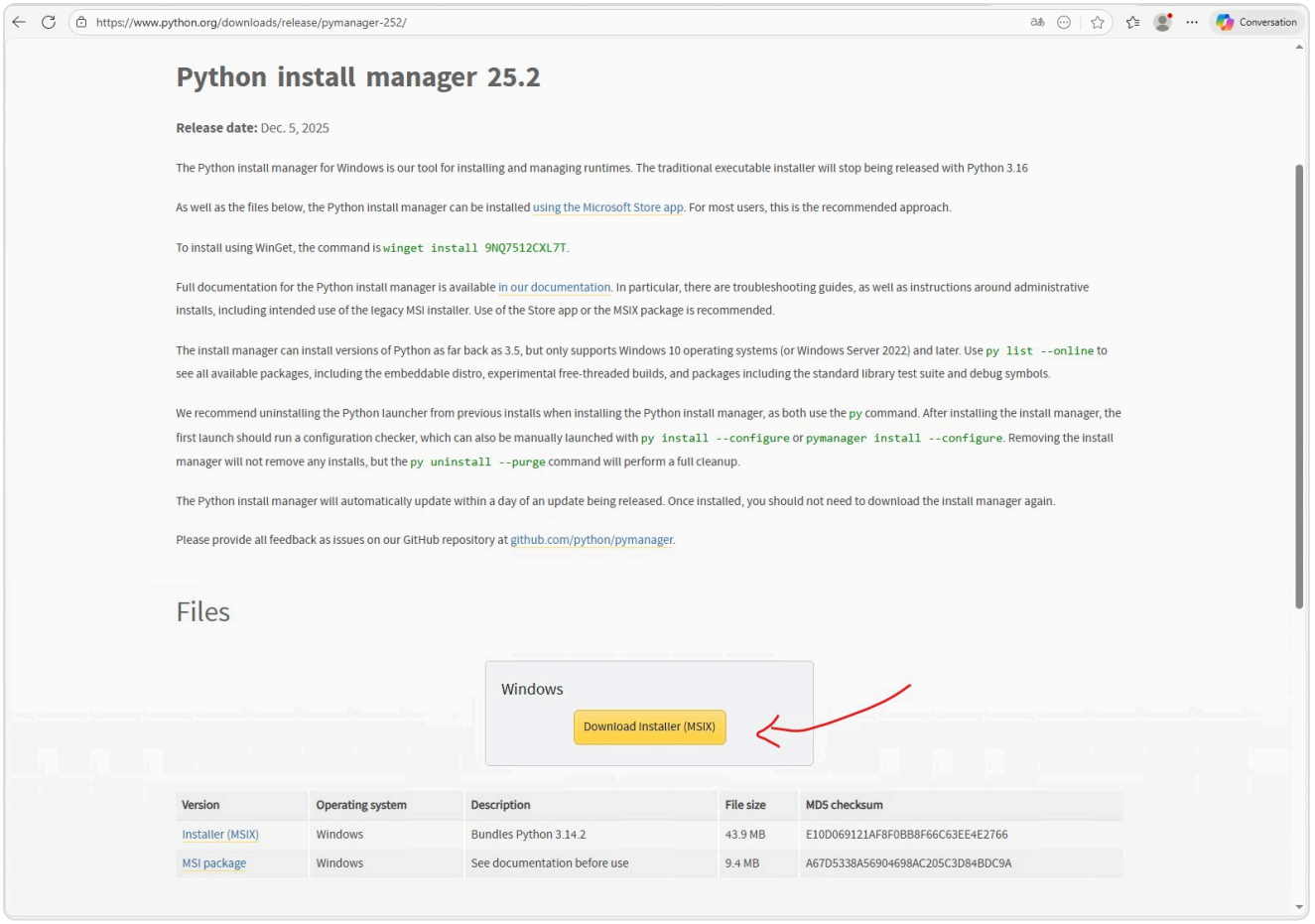
Allez sur https://code.visualstudio.com/download et téléchargez la version pour votre système.



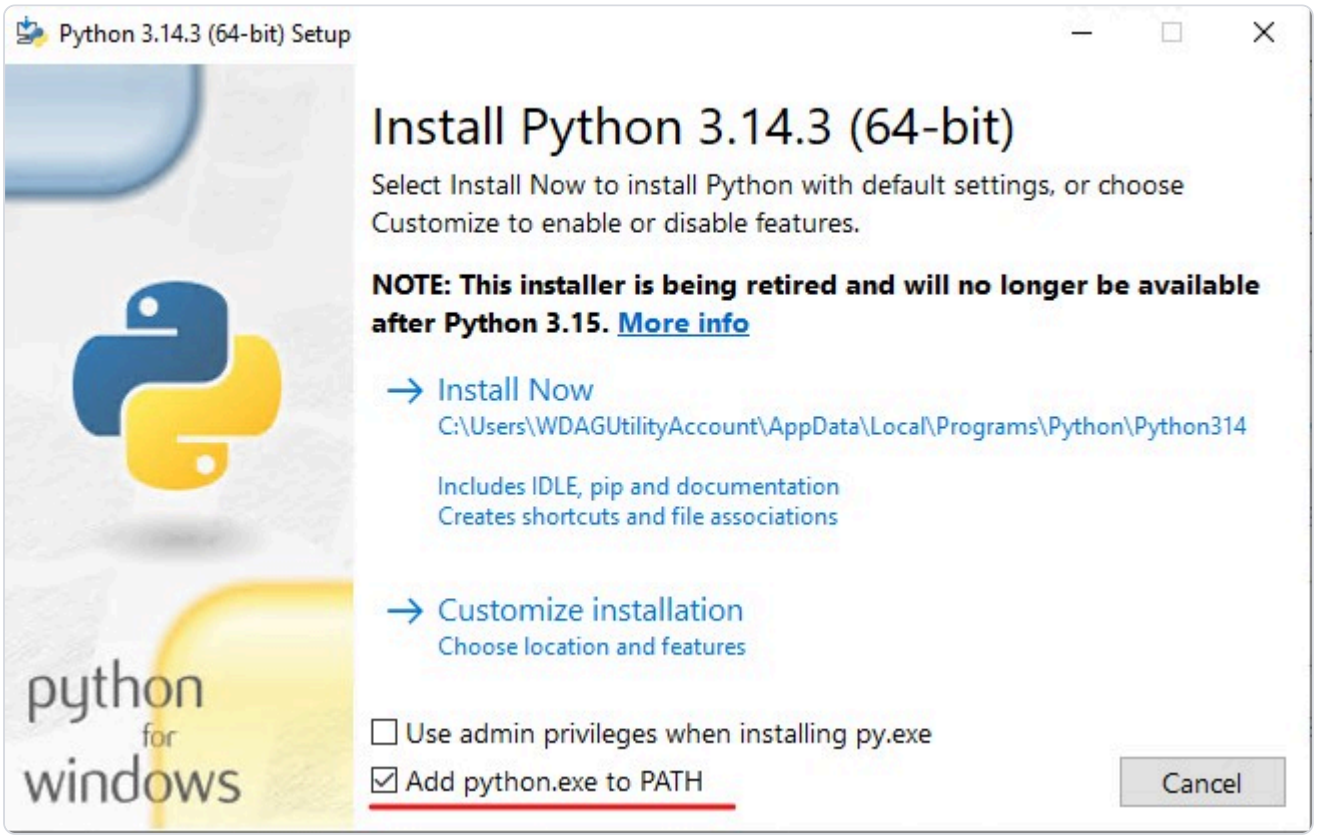
Installez avec les options par défaut. Rien de spécial à configurer.

2a. Python (parcours Python 🐍)

Allez sur https://python.org et téléchargez la dernière version.



Lancez l'installeur. ⚠️ **L'étape critique** : cochez la case **"Add python.exe to PATH"** en bas de la fenêtre avant de cliquer sur "Install Now". Sans ça, rien ne marchera.



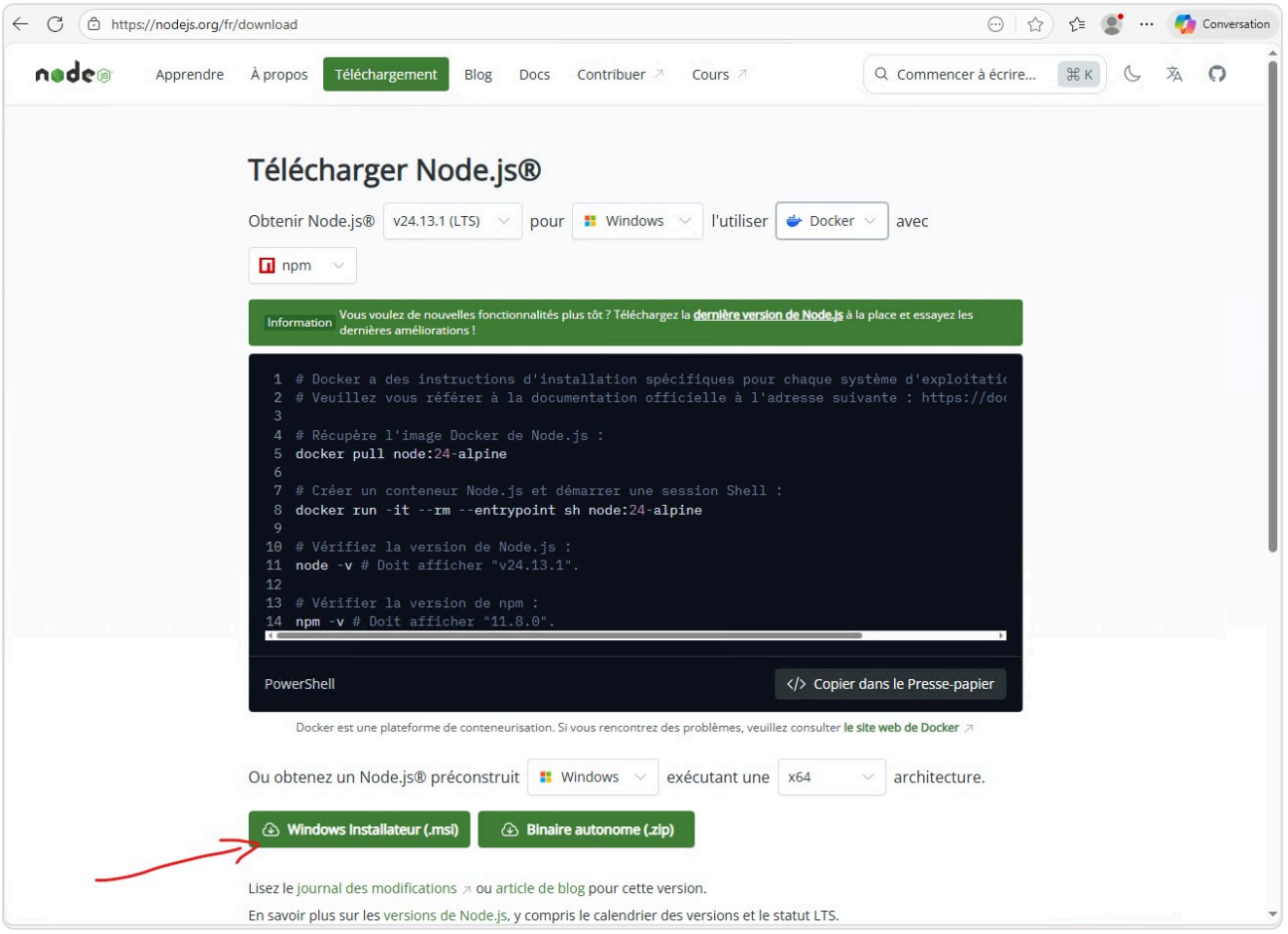
Vérification : ouvrez un terminal dans VSCode (menu Terminal → New Terminal) et tapez :

```css
python --version
```

Si ça affiche un numéro de version (ex: `Python 3.14.3`), c'est bon. Si ça dit "python n'est pas reconnu" → fermez et rouvrez VSCode, ou réinstallez en cochant la case PATH.

2b. Node.js (parcours Web 🌐)

Allez sur https://nodejs.org/fr/download et téléchargez la version **LTS**.



Installez avec les options par défaut. npm (le gestionnaire de packages) est inclus automatiquement.

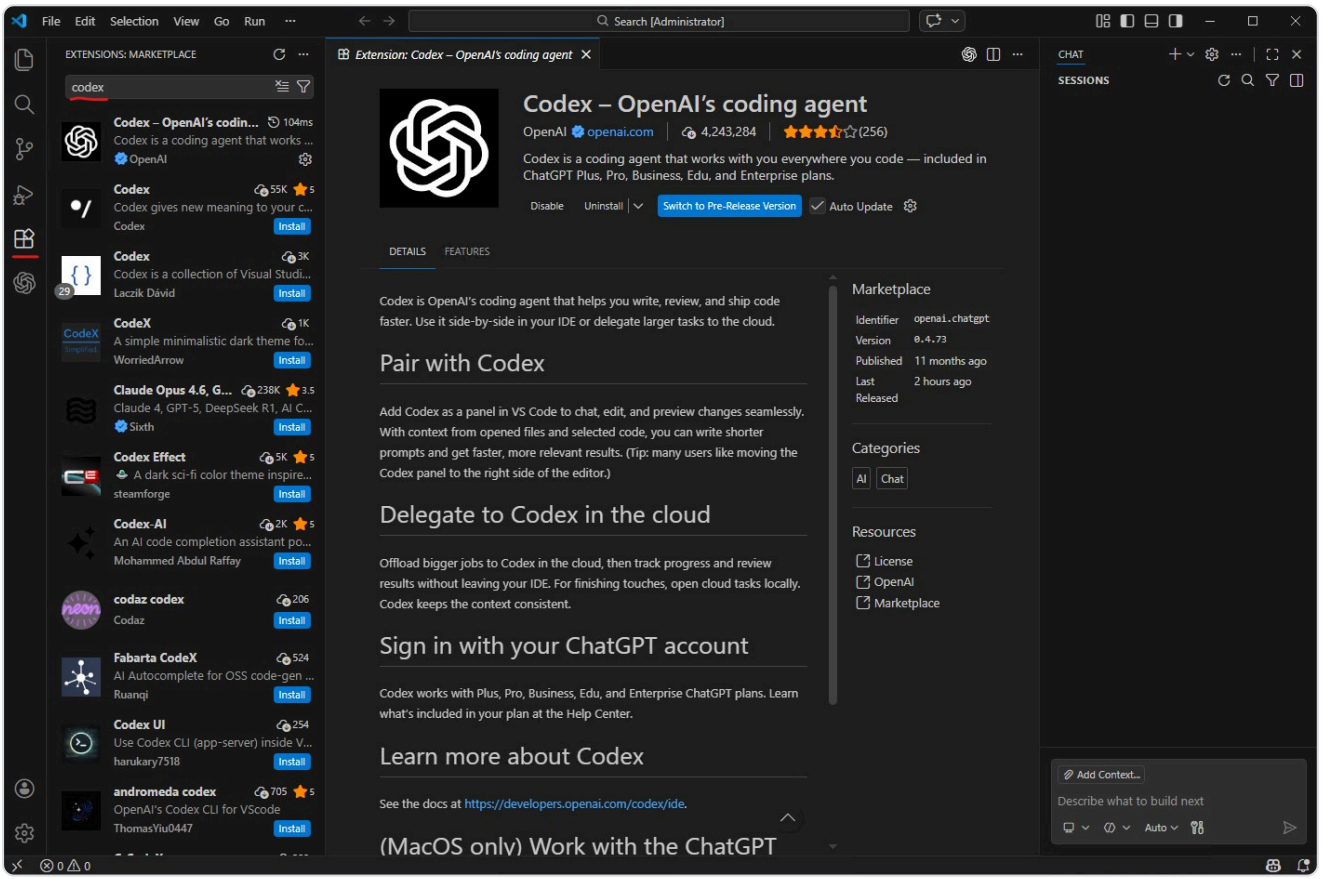
Vérification : ouvrez un terminal dans VSCode et tapez :

```css
node --version
```

3. L'extension Codex (l'agent IA)

Codex est l'agent de programmation d'OpenAI. Il est **gratuit pendant 2 mois** et s'intègre directement dans VSCode.

1. Ouvrez VSCode
2. Cliquez sur l'icône Extensions dans la barre latérale gauche (ou `Ctrl+Shift+X`)
3. Tapez **"codex"** dans la barre de recherche
4. Installez **"Codex – OpenAI's coding agent"** (par OpenAI, avec le badge vérifié bleu)
5. Connectez-vous avec votre compte ChatGPT (ou créez-en un)



Une fois connecté, vous verrez le panneau **CHAT** de Codex à droite de VSCode. C'est là que vous parlerez à l'agent.

💡 **Pas de compte OpenAI ?** Vous pouvez aussi utiliser **Cursor** (https://cursor.com, essai gratuit) ou **GitHub Copilot** (extension VSCode, gratuit pour les profs). La méthode reste la même.

Autres agents (si vous préférez)

Agent	Installation	Prix
Codex (recommandé)	Extension VSCode	Gratuit 2 mois

Agent	Installation	Prix
Cursor	Application séparée : https://cursor.com	Essai gratuit
GitHub Copilot	Intégré à VSCode	Gratuit pour les profs
Claude Code	Terminal : `npm install -g @anthropic-ai/claude-code`	Nécessite abo Claude

En cas de problème

Problème	Solution
"python n'est pas reconnu"	Réinstaller Python en cochant "Add to PATH"
"node n'est pas reconnu"	Fermer et rouvrir VSCode après l'installation
Codex ne se connecte pas	Vérifier la connexion internet, réessayer
Permission denied / droits admin	Demander au formateur
L'antivirus bloque quelque chose	Désactiver temporairement ou ajouter une exception

Le réflexe : si ça bloque, demandez à votre IA (dans le chat web, pas l'agent) : *"J'essaie d'installer [outil] sur Windows mais j'ai cette erreur : [erreur]. Aide-moi."*

3.3 3.3 — Démo : du code à la mise en ligne

Durée : 10 min | **Type** : Démo | **Slides** : ~4-5 (surtout des screenshots)



Ce qu'on va voir

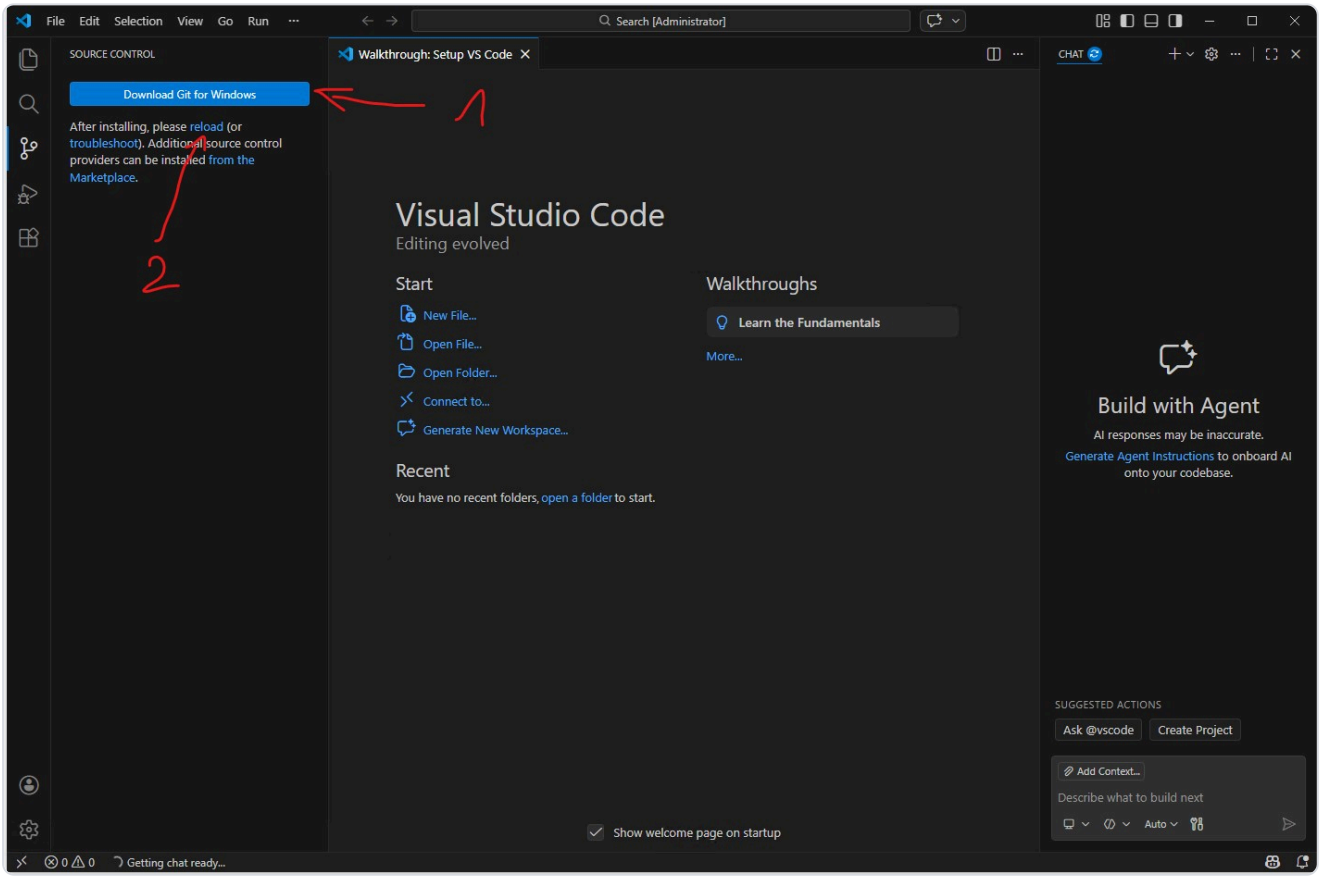
On va suivre le parcours complet d'un vrai projet : installer Git, créer un site web pour l'école avec un agent, gérer les bugs, publier le code sur GitHub, et le déployer en ligne sur Vercel. Tout ça avec Codex dans VSCode.

Le projet : **refaire le site de l'IFOSUP Wavre** — l'école où vous êtes. Le résultat final est en ligne ici : https://site-ifosup.vercel.app/

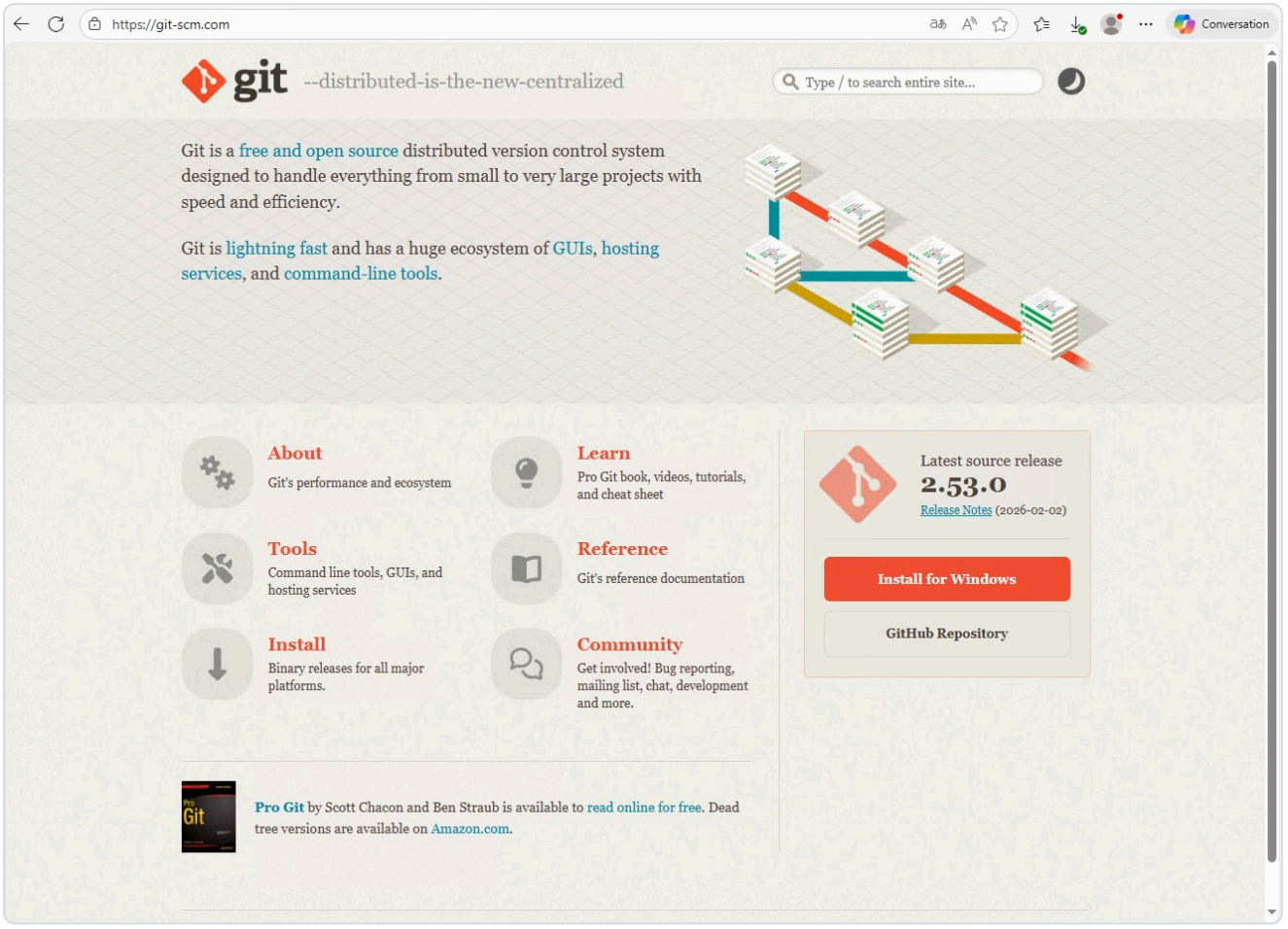
Étape 1 : Installer Git

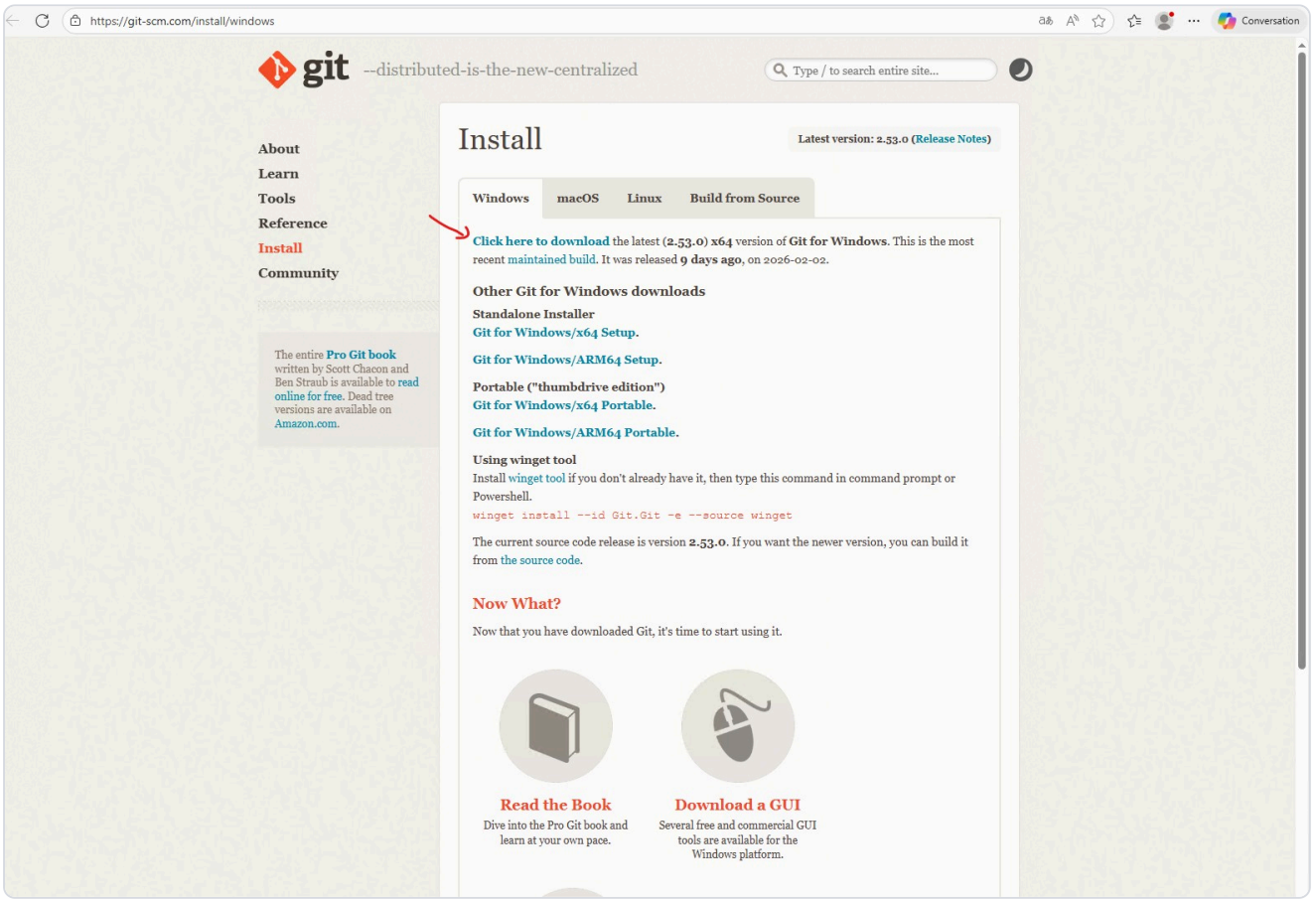
Git est un outil de versioning — il sauvegarde l'historique de votre code et permet de le publier sur GitHub. VSCode le détecte automatiquement et vous propose de l'installer.

Quand vous ouvrez l'onglet **Source Control** (icône branche à gauche) sans Git installé, VSCode affiche un bouton **"Download Git for Windows"**. Cliquez dessus.



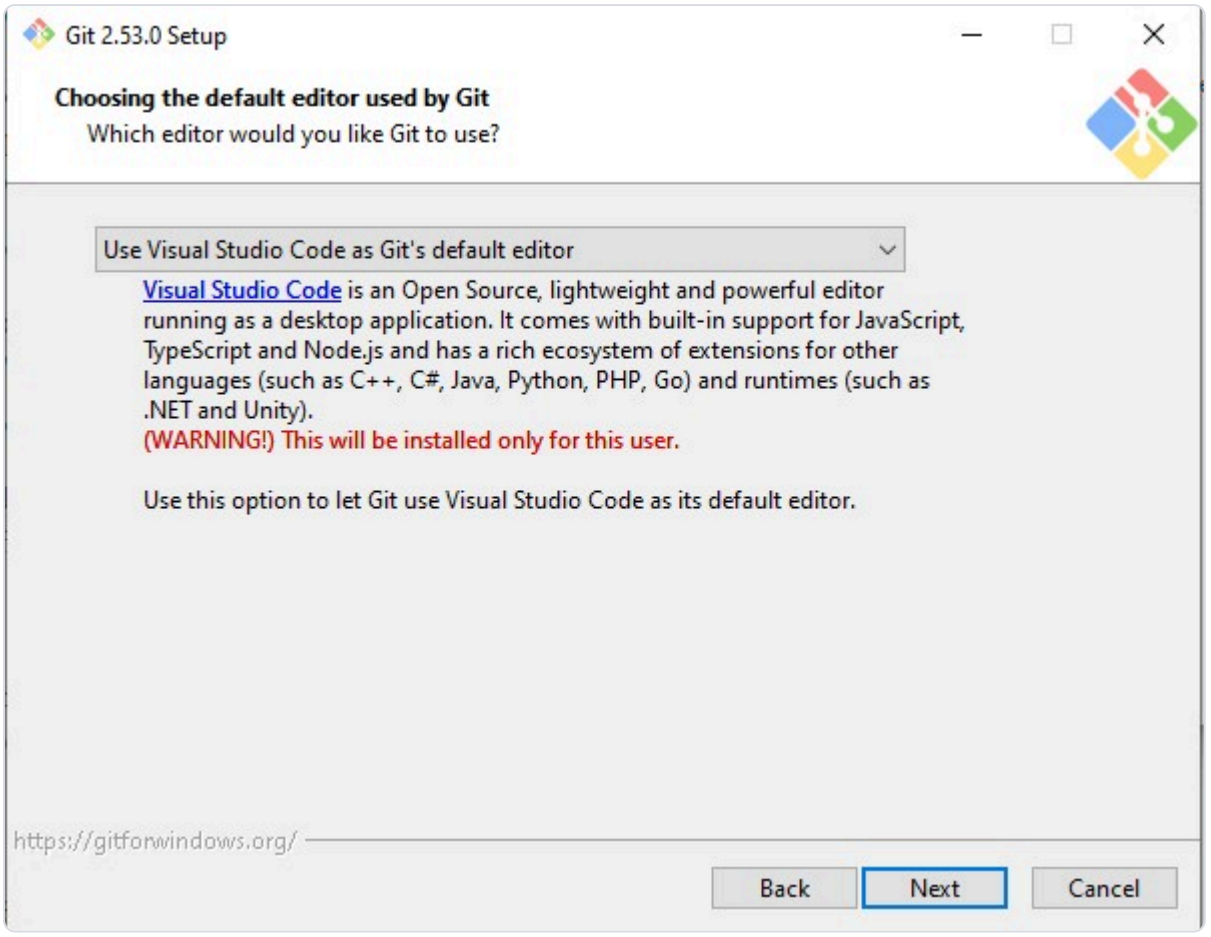
Ça vous amène sur le site de Git. Cliquez sur **"Click here to download"** pour télécharger l'installeur.


https://git-scm.com
Conversation
git --distributed-is-the-new-centralized
Type / to search entire site...
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
Git is lightning fast and has a huge ecosystem of GUIs, hosting services, and command-line tools.
About
Git's performance and ecosystem
Learn
Pro Git book, videos, tutorials, and cheat sheet
Tools
Command line tools, GUIs, and hosting services
Reference
Git's reference documentation
Install
Binary releases for all major platforms.
Community
Get involved! Bug reporting, mailing list, chat, development and more.
Latest source release
2.53.0
Release Notes (2026-02-02)
Install for Windows
GitHub Repository
Pro Git by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

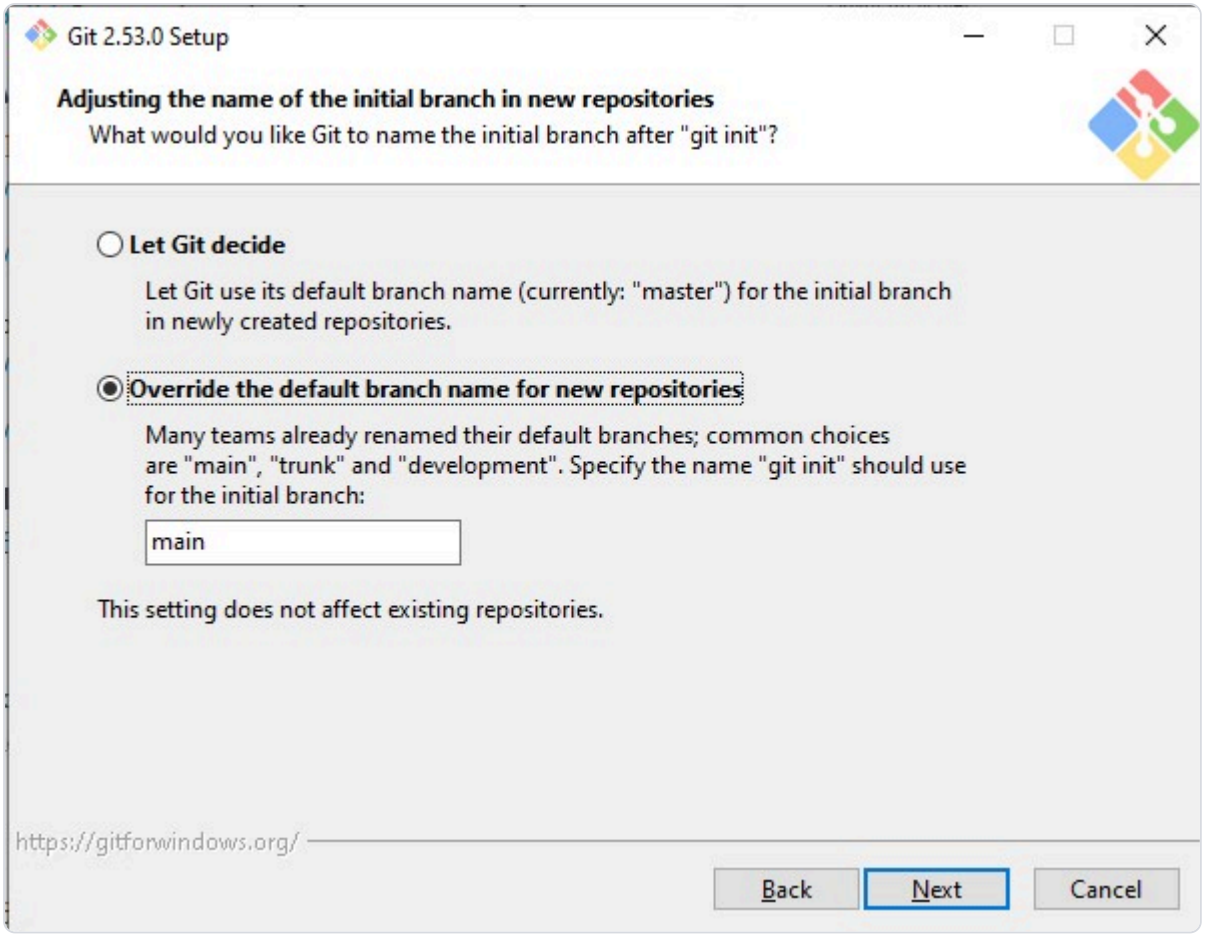

https://git-scm.com/install/windows
Conversation
git --distributed-is-the-new-centralized
Type / to search entire site...
About
Learn
Tools
Reference
Install
Community
The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.
Install
Latest version: 2.53.0 (Release Notes)
Windows
macOS
Linux
Build from Source
Click here to download the latest (2.53.0) x64 version of Git for Windows. This is the most recent maintained build. It was released 9 days ago, on 2026-02-02.
Other Git for Windows downloads
Standalone Installer
Git for Windows/x64 Setup.
Git for Windows/ARM64 Setup.
Portable ("thumbdrive edition")
Git for Windows/x64 Portable.
Git for Windows/ARM64 Portable.
Using winget tool
Install winget tool if you don't already have it, then type this command in command prompt or Powershell.
winget install --id Git.Git -e --source winget
The current source code release is version 2.53.0. If you want the newer version, you can build it from the source code.
Now What?
Now that you have downloaded Git, it's time to start using it.
Read the Book
Dive into the Pro Git book and learn at your own pace.
Download a GUI
Several free and commercial GUI tools are available for the Windows platform.

Pendant l'installation, deux réglages à noter :

Éditeur par défaut → sélectionnez **"Use Visual Studio Code"** dans le menu déroulant. Comme ça, Git utilisera VSCode si besoin.



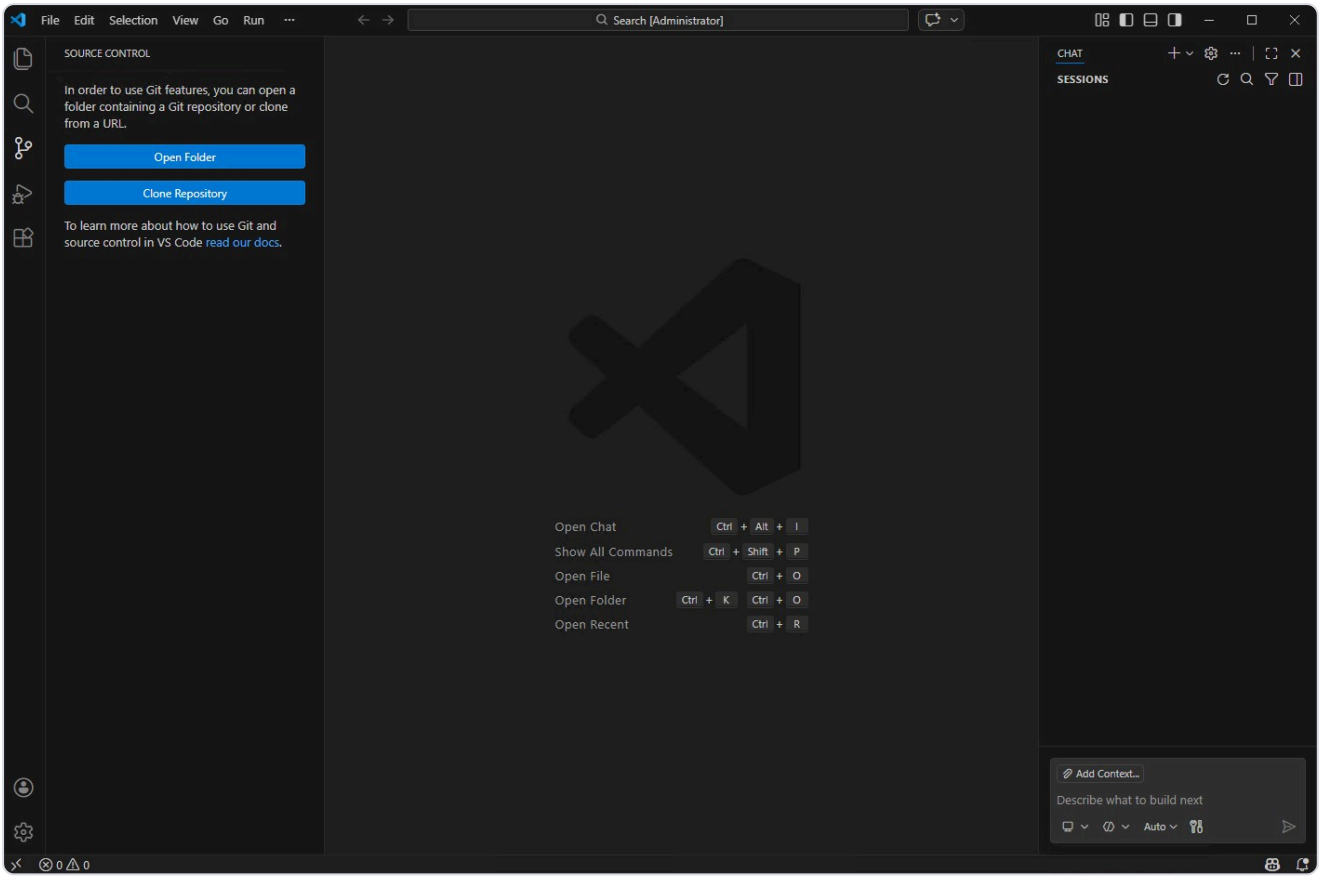
Nom de la branche initiale → sélectionnez **"Override the default branch name"** et tapez **main**. C'est la convention actuelle (au lieu de l'ancien "master").



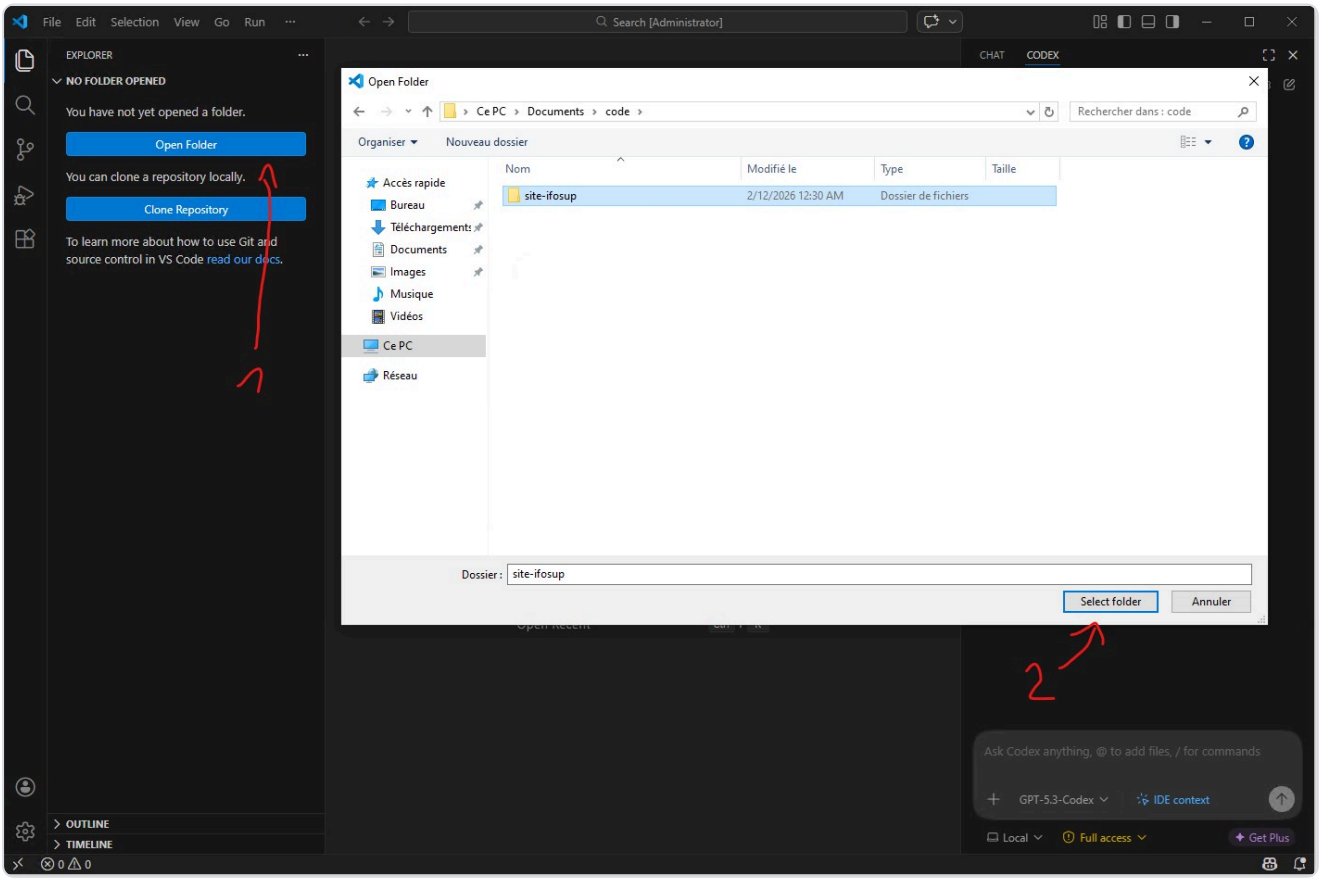
Pour le reste, laissez les options par défaut et cliquez "Next" jusqu'à la fin.

Étape 2 : Ouvrir un dossier de projet

Après l'installation de Git, revenez dans VSCode. L'onglet Source Control affiche maintenant **"Open Folder"** et **"Clone Repository"** — Git est reconnu.

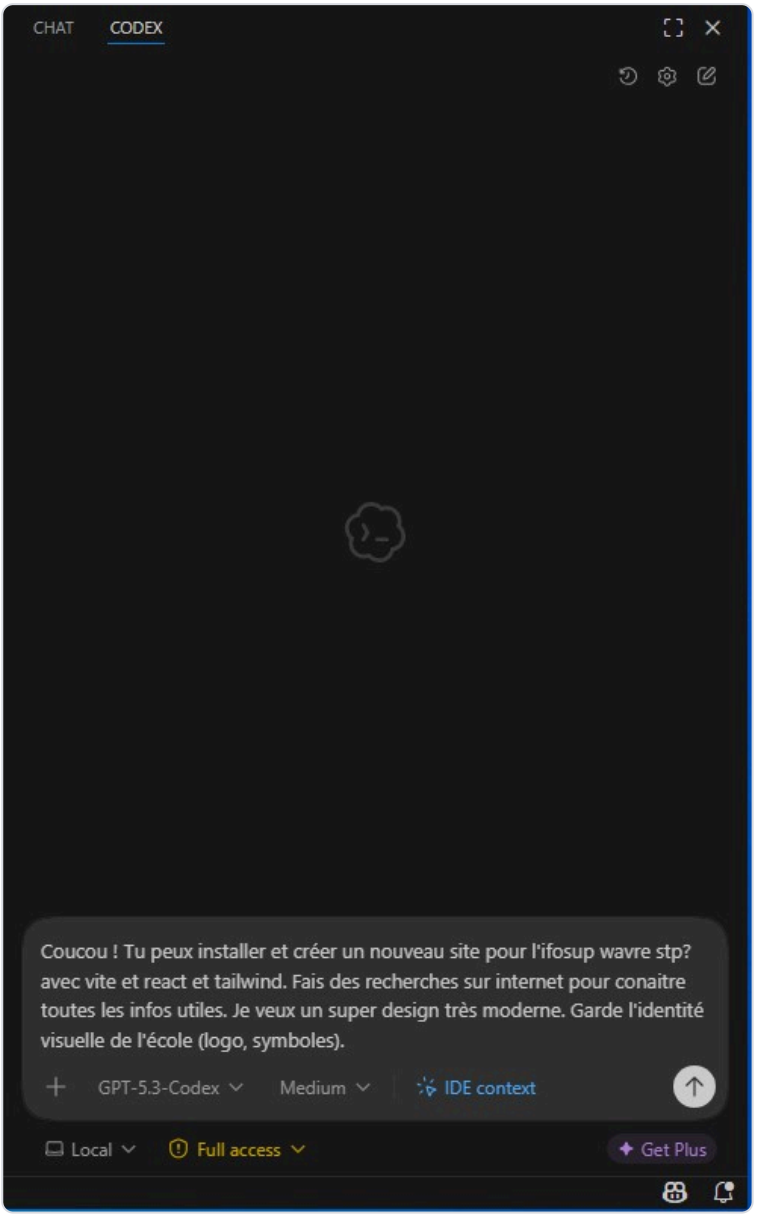


Créez un dossier pour votre projet (par exemple `Documents/code/site-ifosup`) et ouvrez-le dans VSCode.



Étape 3 : Le prompt — on demande à Codex de créer le site

Voici le moment clé. Dans le panneau **CODEX** à droite, on tape un prompt décrivant ce qu'on veut :

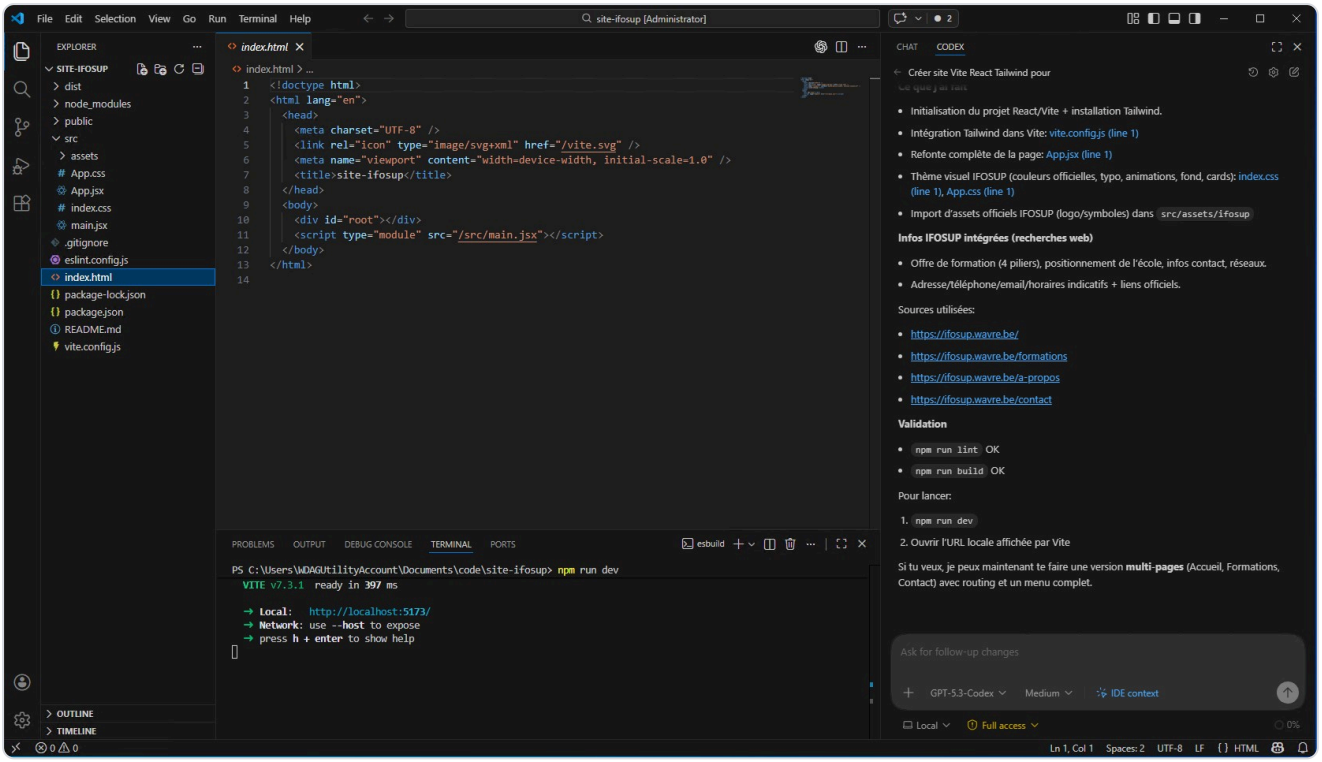


Notez le prompt : informel, avec des fautes, pas de spécification technique détaillée. On dit juste ce qu'on veut. L'agent se débrouille — il va même faire des recherches sur internet pour trouver les infos de l'école (adresse, formations, logo...).

Codex travaille : il crée le projet Vite + React + Tailwind, installe les dépendances, crée les composants, importe les assets de l'école, et lance le serveur de développement.

Étape 4 : Le résultat — et le premier bug

Quelques minutes plus tard, Codex a terminé. Il montre un résumé de ce qu'il a fait :



L'agent a créé toute la structure, installé les packages, et le serveur tourne. On ouvre le navigateur... et voilà le site :



Le design est beau, la structure est bonne, les infos sont correctes (Codex a cherché sur le site réel de l'école)... mais il y a un **bug d'encodage**. Tous les caractères accentués (é, è, à, ê...) s'affichent comme des losanges ◆.

C'est un bug classique avec Codex — il arrive régulièrement que l'agent génère du texte sans gérer correctement l'UTF-8. Pas de panique : on applique la méthode. On dit à l'agent ce qui ne va pas.

Un simple message du type *"Les caractères accentués ne s'affichent pas correctement, il y a des losanges partout. Corrige l'encodage."* et Codex corrige :



Voilà. Le cycle a fonctionné : l'agent crée → on teste → on signale le bug → l'agent corrige.

💡 **Leçon importante** : le premier résultat n'est presque jamais parfait. C'est **normal**. Ce qui compte, c'est d'appliquer le cycle.

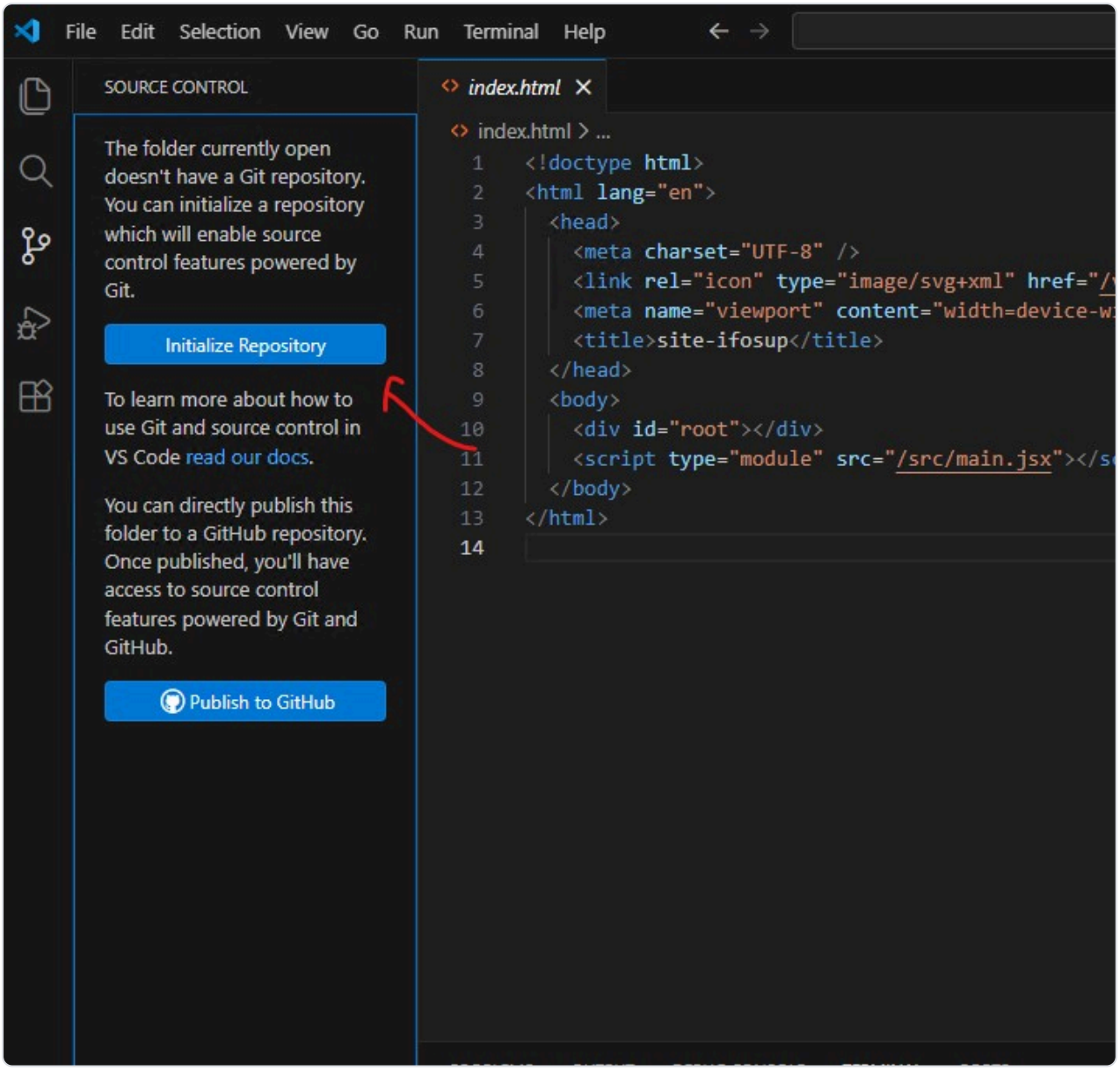
Étape 5 : Sauvegarder sur GitHub

Le site marche en local. Maintenant on veut le sauvegarder sur GitHub (et le déployer ensuite). Git est intégré dans VSCode — pas besoin du terminal.

Premier commit

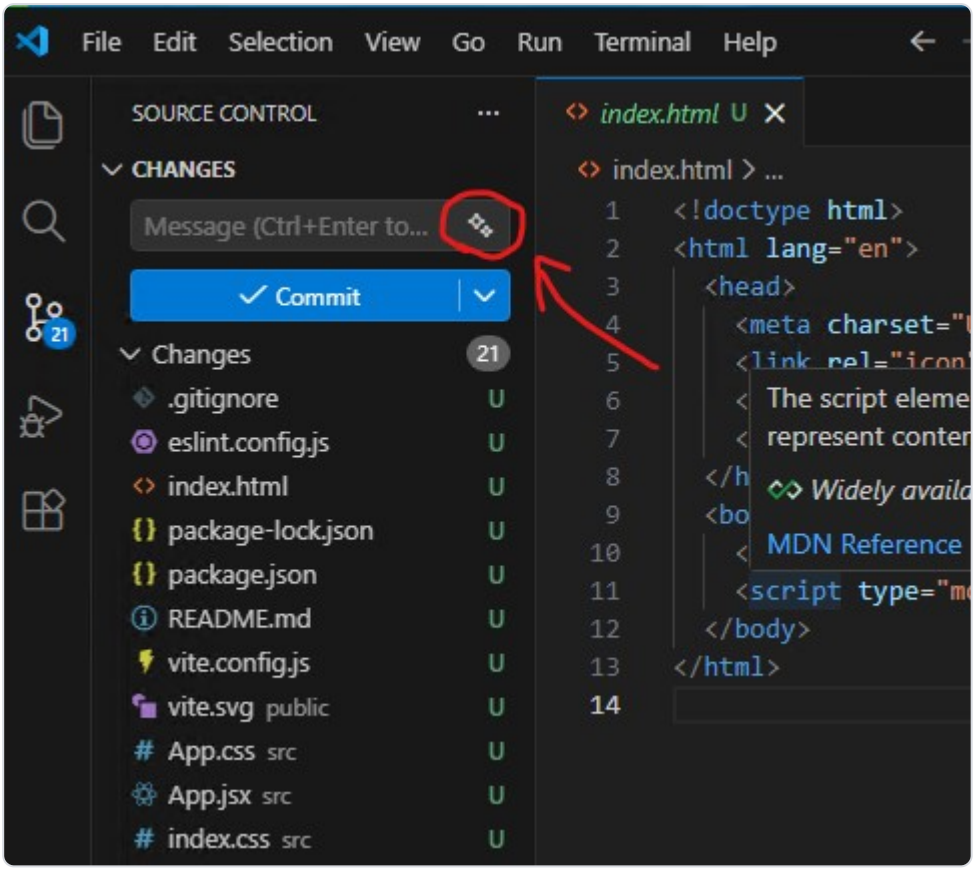
Allez dans l'onglet **Source Control**. Vous voyez la liste de tous les fichiers créés par l'agent.

Premièrement, il faut initialiser le repository :

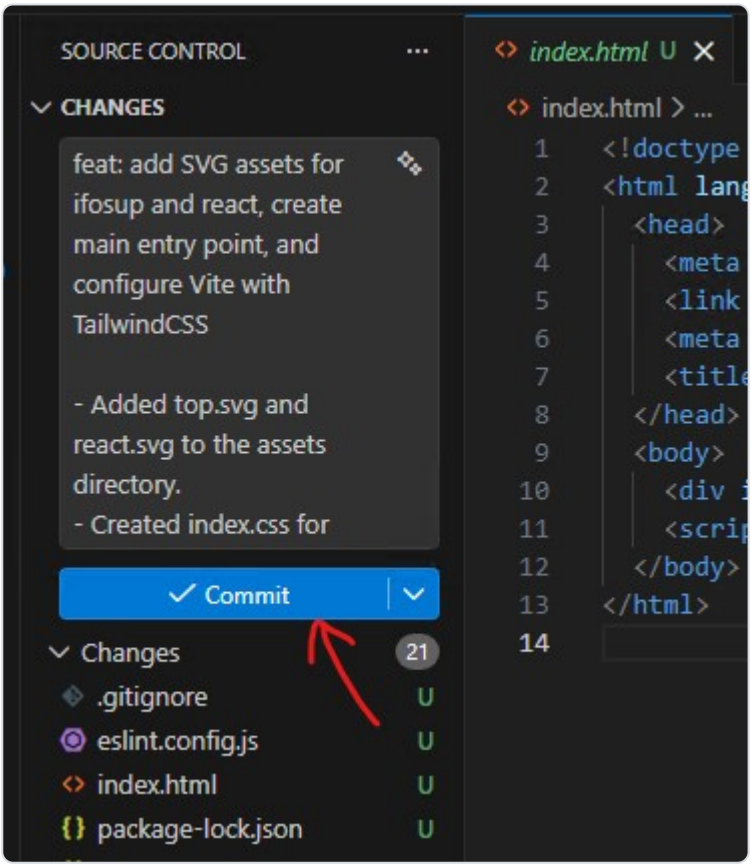


Pour sauvegarder, il faut faire un **commit** (= une sauvegarde horodatée).

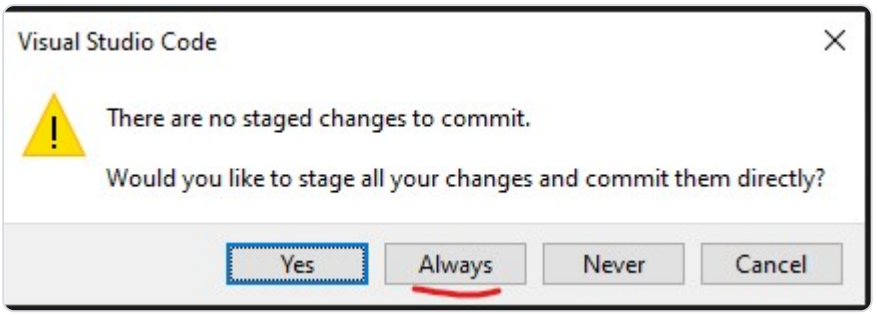
Cliquez sur l'icône ✨ **étincelle** à côté du champ de message — Codex génère automatiquement un message de commit descriptif.



Le message généré automatiquement décrit ce qui a été fait :

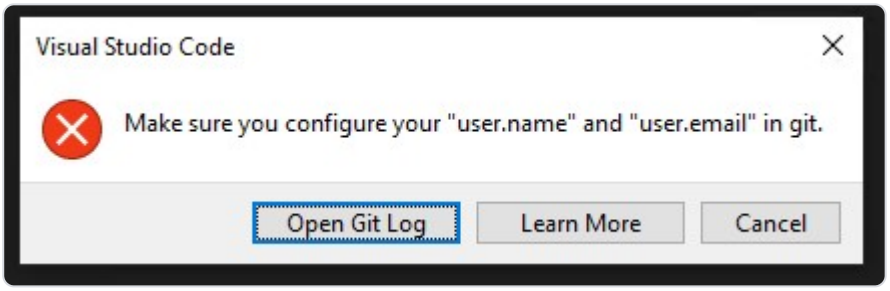


Cliquez **Commit**. VSCode demande si vous voulez "stage" (préparer) tous les fichiers — cliquez **"Always"** pour ne plus avoir cette question.



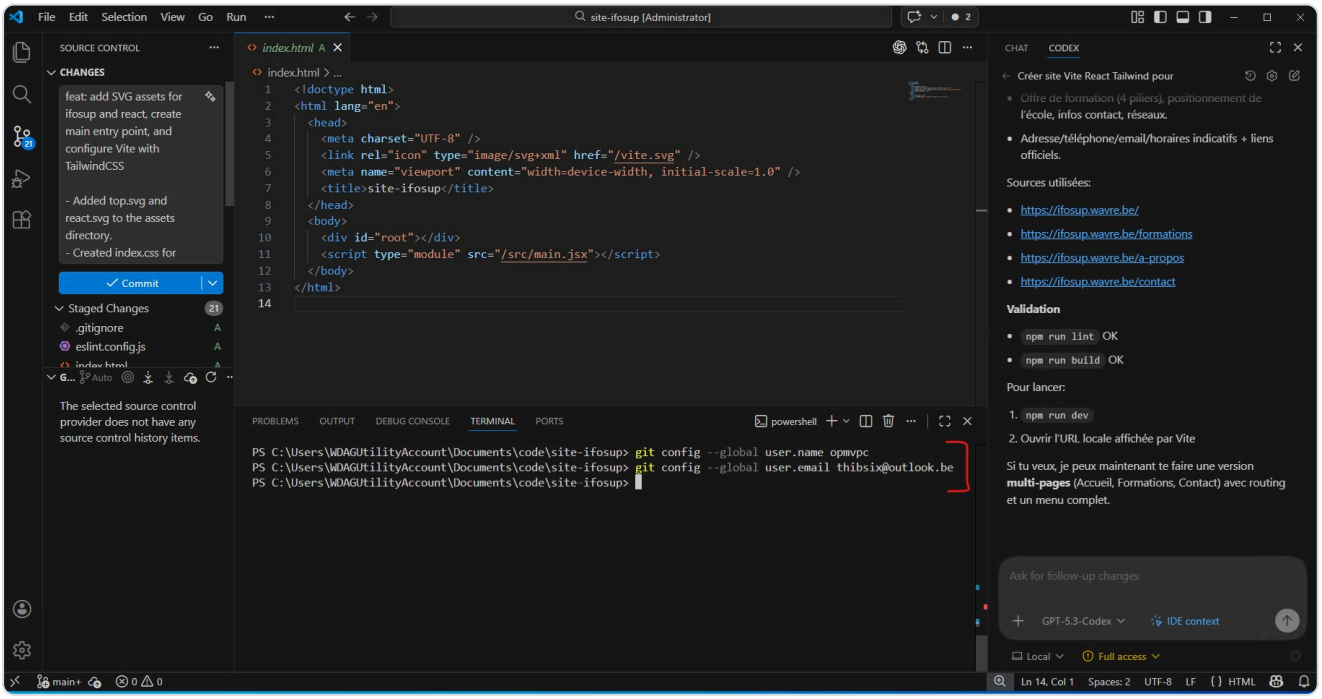
Configurer votre identité Git

Si c'est votre premier commit, Git vous demande de configurer votre nom et email :



Ouvrez le terminal dans VSCode (menu Terminal → New Terminal) et tapez ces deux commandes avec **vos** informations :

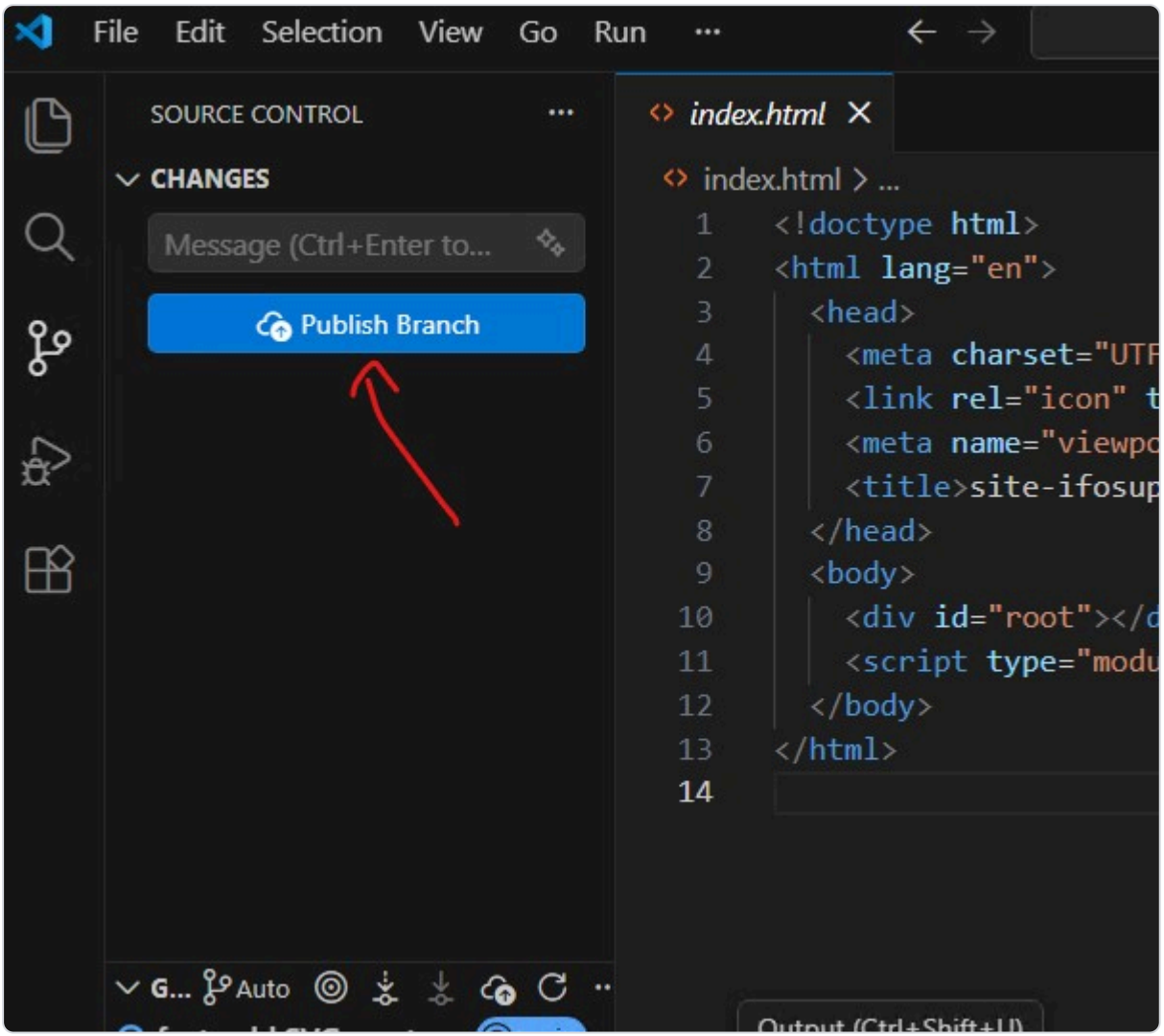
```
git config --global user.name "Votre Nom"
git config --global user.email "votre@email.com"
```



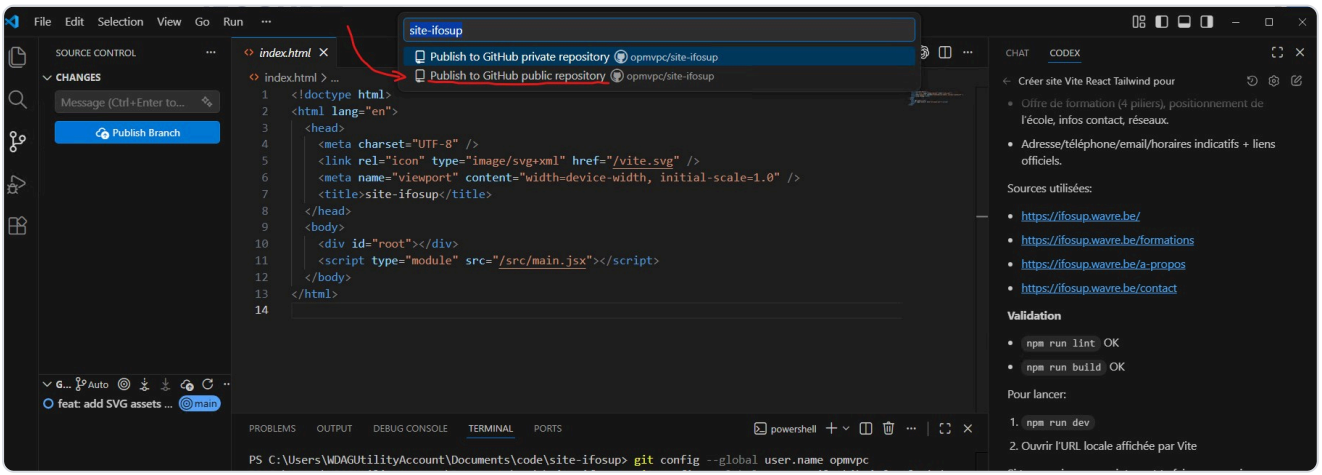
Puis relancez le commit.

Publier sur GitHub

Une fois le commit fait, le bouton **"Publish Branch"** apparaît :



Cliquez dessus. VSCode vous demande le nom du dépôt et s'il doit être **privé** ou **public** :



Choisissez **private** (on pourra toujours le rendre public après). VSCode pousse tout sur GitHub. Votre code est maintenant sauvegardé en ligne.

Étape 6 : Déployer sur Vercel

Dernière étape : mettre le site en ligne. Vercel se connecte à votre dépôt GitHub et déploie automatiquement.

1. Allez sur https://vercel.com → connectez-vous avec GitHub
2. "New Project" → sélectionnez votre dépôt `site-ifosup`
3. Cliquez "Deploy"
4. ~30 secondes plus tard...

C'est en ligne : https://site-ifosup.vercel.app/

Une URL publique, accessible par n'importe qui, depuis n'importe quel appareil. Et à chaque fois que vous pousserez du code sur GitHub, Vercel redéploiera automatiquement.



Résumé : ce qu'on vient de faire

Étape	Action	Temps
1	Installer Git	~3 min

Étape	Action	Temps
2	Créer un dossier et l'ouvrir dans VSCode	30 sec
3	Envoyer un prompt à Codex	30 sec
4	Codex crée le site + on corrige le bug d'encodage	~5 min
5	Commit + publier sur GitHub	~2 min
6	Déployer sur Vercel	~1 min

Total : ~12 minutes pour un site web complet, avec un vrai design, publié en ligne avec une URL partageable.

Et la méthode ? C'est exactement la même que dans le bloc 2 : prompt → l'agent crée → on teste → on corrige → on itère. L'outil est plus puissant, mais la méthode ne change pas.

Et le parcours Python ?

Le parcours Python est plus simple à démarrer mais reste local (sur votre machine). Le flux :

1. L'agent crée un fichier Python avec Streamlit
2. Il installe les dépendances (`pip install streamlit`)
3. Il lance le serveur (`streamlit run app.py`)
4. L'app s'ouvre dans votre navigateur à `http://localhost:8501`
5. Pas de Git ni de déploiement nécessaire — parfait pour des outils personnels ou des données sensibles

Le prompt type :

> *Je suis prof de [matière]. Fais-moi un outil Streamlit pour [besoin] avec une base de données SQLite pour stocker les données. Ajoute un export Excel.*

Le parcours Python n'a pas besoin de Git, de GitHub, ni de Vercel. C'est plus simple, mais l'application ne tourne que sur votre machine.

	Parcours Python 🐍	Parcours Web 🌐
Complexité	⭐ Plus simple	⭐⭐ Plus d'étapes
Résultat	App locale (votre PC)	Site en ligne (URL partageable)
Données	Restent sur votre machine	Sur un serveur
Idéal pour	Outils perso, données sensibles	Apps partagées, sites, collaboratif
Git nécessaire ?	Non	Oui (pour Vercel)

3.4 3.4 — Piloter son projet : les fichiers .md

Durée : 5 min | **Type** : Théorie | **Slides** : ~2-3



Le problème

Un agent IA n'a pas de mémoire entre les conversations. Chaque fois que vous ouvrez un nouveau chat, il repart de zéro. Il ne sait pas ce que vous construisez, quelles décisions vous avez prises, où vous en êtes, ni ce qu'il a déjà fait.

La solution : des **fichiers texte** (.md = markdown) que l'agent lit au début de chaque session. Ces fichiers sont le cerveau de votre projet — l'agent s'y réfère en permanence, et c'est **lui** qui les maintient à jour.

La structure

Créez ces fichiers à la racine de votre projet :

```
mon-projet/
├── AGENTS.md          # Instructions permanentes pour l'agent
├── CDC.md             # Cahier des charges (ce qu'on construit)
├── DESIGN.md          # Direction visuelle
├── PLANNING.md        # Plan par phases
├── notes/
│   └── JOURNAL.md     # Problèmes, solutions, décisions
└── src/               # Le code (géré par l'agent)
```

💡 Vous n'avez pas à écrire tout ça seul. Commencez par décrire votre projet à l'agent, et demandez-lui de générer ces fichiers. Ensuite vous ajustez.

AGENTS.md — Les instructions permanentes

C'est le fichier le plus important. L'agent le lit en premier. Il contient **qui vous êtes**, **ce qu'est le projet**, et **les règles à suivre**.

```
# Instructions pour l'agent

## Contexte
Je suis prof de comptabilité dans une école secondaire en Belgique.
Ce projet est un outil de gestion des grilles d'évaluation.

## Stack technique
- Python + Streamlit
- Base de données : SQLite
- Langue du code : français pour l'interface, anglais pour le code

## Règles
- Toujours lire CDC.md et PLANNING.md avant de commencer
- Suivre le plan phase par phase — ne pas sauter d'étapes
- Tester le code avant de dire que c'est terminé
- Mettre à jour PLANNING.md après chaque phase complétée
- Noter les problèmes et solutions dans notes/JOURNAL.md
- Ne jamais supprimer de fichier sans demander
- Pas de données personnelles d'élèves dans le code

## Structure du projet
[L'agent remplira ça au fur et à mesure]
```

💡 Si vous utilisez **Claude Code** au lieu de Codex, le fichier s'appelle `CLAUDE.md` — c'est la même chose, c'est juste le nom que Claude Code cherche automatiquement.

CDC.md — Le cahier des charges

C'est la bible du projet. Ce que l'application fait, pour qui, et comment.

```
# Cahier des charges — Gestionnaire de grilles d'évaluation

## Description
Application Streamlit pour créer des grilles d'évaluation par compétences,
encoder les points des élèves, et générer des fiches de résultats en PDF.

## Fonctionnalités MVP (prioritaire)
- Créer une grille : compétences, critères, pondérations
- Encoder les résultats par élève
- Calcul automatique des totaux
- Export PDF de la fiche individuelle

## Fonctionnalités secondaires (si le temps le permet)
- Réutiliser une grille d'une année à l'autre
- Statistiques par classe (moyenne, distribution)
- Export Excel

## Exigences
- Interface simple et claire (les collègues doivent pouvoir l'utiliser)
- Données stockées localement (SQLite)
- Pas de connexion internet requise
```

Pas besoin de 10 pages. Un CDC clair et concis vaut mieux qu'un roman que personne ne lit.

PLANNING.md — Le plan par phases

L'agent le propose, vous le validez. Chaque phase est petite et testable.

```
# Plan d'implémentation

## Phase 1 — Structure de base ⬜
- Initialiser le projet Streamlit
- Configurer SQLite
- Page d'accueil minimale

## Phase 2 — Créer une grille ⬜
- Formulaire de création de grille
- Ajout de compétences et critères
- Sauvegarde en base

## Phase 3 — Encoder les résultats ⬜
- Sélectionner une grille et une classe
- Tableau d'encodage par élève
- Calcul automatique

## Phase 4 — Export PDF ⬜
- Génération de la fiche individuelle
- Mise en page propre avec logo

Légende : ⬜ À faire | 🔄 En cours | ✅ Validé
```

⚠️ **Règle d'or** : une bonne phase se résume en une phrase et se teste en 2 minutes. Si c'est trop gros, découpez.

DESIGN.md — La direction visuelle

Pas besoin de maquettes Figma. Quelques lignes suffisent pour donner une direction :

```
# Design — Grilles d'évaluation

## Ambiance
Interface épurée et professionnelle. Sobre, pas de fioritures.
Inspiration : outils Google (Sheets, Forms).

## Couleurs
- Primaire : #2563EB (bleu)
- Fond : #F8FAFC (gris très clair)
- Texte : #1E293B (quasi-noir)
- Succès : #16A34A (vert)
- Erreur : #DC2626 (rouge)

## Principes
- Beaucoup d'espace blanc
- Tableaux lisibles avec lignes alternées
- Boutons clairs avec icônes
```

notes/JOURNAL.md — La mémoire du projet

C'est l'agent qui le maintient. Demandez-lui après chaque session :

> *"Mets à jour notes/JOURNAL.md avec ce qu'on vient de faire, les problèmes rencontrés et les solutions."*

```
# Journal du projet

## 2026-02-13 — Phase 1 : Structure de base

### Ce qui a été fait
- Projet Streamlit initialisé
- SQLite configuré avec les tables grilles, compétences, résultats
- Page d'accueil avec navigation sidebar

### Problèmes rencontrés
- Streamlit ne trouvait pas le fichier de base de données
  → Solution : utiliser un chemin absolu via Path(__file__).parent

### Décisions
- On utilise st.session_state pour garder l'état entre les pages

### Statut : Phase 1 ✅
```

Le journal a deux utilités : il aide l'agent à retrouver le contexte quand vous reprenez le travail, et il vous aide à documenter votre progression.

En pratique : le workflow

1. **Début de projet** : décrivez votre idée à l'agent → demandez-lui de générer AGENTS.md, CDC.md, DESIGN.md et PLANNING.md
2. **Vous relisez** et ajustez (c'est VOTRE projet, pas celui de l'agent)
3. **Chaque session** : l'agent lit les fichiers → travaille sur la phase en cours → met à jour PLANNING.md et JOURNAL.md
4. **Quand vous reprenez** : l'agent relit tout et sait exactement où il en est

💡 Le but c'est que l'agent travaille pour **vous**, pas l'inverse. Vous décidez, il exécute et documente.

3.5 3.5 — Sécurité et audits

Durée : 5 min | **Type** : Théorie | **Slides** : ~2



Les règles non-négociables

Avant de vous lâcher sur l'exercice, quatre réflexes à prendre :

#	Règle	Le réflexe
1	**Secrets** : jamais de clés API dans le code	*"Vérifie qu'il n'y a pas de clés API en clair"*
2	**RGPD** : pas de données réelles d'élèves	Données fictives pour le développement
3	**GitHub** : dépôts **privés** par défaut	*"Audite ce projet avant que je le rende public"*
4	**En ligne** : audit avant déploiement	*"Vérifie que mon app est sécurisée"*

Le point commun : **demandez à votre IA**. Elle est très bonne pour repérer ces problèmes.

La méthode des audits croisés

Pour aller plus loin que le simple "vérifie que c'est sécurisé", il existe une méthode plus rigoureuse en deux étapes. C'est comme ça qu'on fait dans le monde professionnel — et c'est faisable en 10 minutes avec des agents.

Étape 1 : Créer la procédure d'audit

Demandez à un agent (dans le chat web ou dans un agent séparé) de **rédiger une procédure d'audit** pour un aspect spécifique de votre application :

> *"Rédige une procédure d'audit complète pour évaluer la **sécurité** d'une application web Node.js. La procédure doit couvrir : injection SQL, XSS, CSRF, gestion des secrets, validation des inputs, headers de sécurité, dépendances vulnérables. Pour chaque point, donne les critères de vérification et le niveau de sévérité (critique, important, mineur)."*

L'agent produit un document structuré avec des critères clairs et mesurables.

Étape 2 : Faire auditer par un autre agent

Donnez cette procédure à un **autre agent** (pas celui qui a écrit le code !) et demandez-lui de faire l'audit :

> *"Voici une procédure d'audit de sécurité [coller ou référencer le document]. Audite le code de ce projet en suivant cette procédure point par point. Produis un rapport avec : chaque critère vérifié, le résultat (conforme / non-conforme / N/A), les problèmes trouvés, et les corrections recommandées."*

💡 **Pourquoi un autre agent ?** Parce que l'agent qui a écrit le code a tendance à être indulgent avec son propre travail. Un regard extérieur — même artificiel — est plus objectif.

Ça marche pour tout

La même méthode s'applique à d'autres aspects :

Aspect	Prompt pour la procédure
Qualité du code	*"Rédige une procédure d'audit de qualité : code mort, duplication, nommage, complexité, lisibilité"*
Testing	*"Rédige une procédure d'audit de couverture de tests : cas couverts, cas manquants, cas limites"*
Performance	*"Rédige une procédure d'audit de performance : requêtes SQL, chargement des pages, bundle size"*
Accessibilité	*"Rédige une procédure d'audit d'accessibilité : navigation clavier, contraste, aria labels, lecteur d'écran"*

C'est une habitude puissante à prendre : **faire vérifier le travail d'un agent par un autre agent**. Deux paires d'yeux (même artificielles) valent mieux qu'une.

3.6 3.6 — Exercice 2 : votre app, votre choix

Durée : 50 min | **Type** : Pratique | **Slides** : ~2-3



Consigne

Créez une **vraie application** avec votre agent de programmation. Contrairement à l'exercice 1 (artifacts), ici vous travaillez avec de vrais fichiers, une vraie base de données, et la possibilité de déployer en ligne.

La marche à suivre :

1. **Présentez-vous** à l'agent (30 sec) — qui vous êtes, niveau, contexte
2. **Brainstormez** avec l'agent (3-5 min) — décrivez votre besoin, demandez des suggestions
3. **Demandez un plan** en étapes — l'agent propose, vous validez
4. **Le cycle** : l'agent code → vous testez → vous dites ce qui va pas → l'agent corrige → étape suivante
5. **Itérez** — améliorez, ajoutez des fonctionnalités, peaufinez

50 minutes, c'est un MVP — une fonctionnalité qui marche bien vaut mieux que dix fonctionnalités bancales.

Choisissez votre parcours

Parcours Python 🐍

- **Framework** : Streamlit (interface web automatique) + SQLite (base de données locale)
- **L'agent gère tout** : installation des packages, écriture du code, lancement du serveur
- **Tester** : l'app s'ouvre dans votre navigateur à `localhost:8501`
- **Idéal pour** : outils personnels, données sensibles, traitement de fichiers, scripts

Prompt de départ type :

> *Je suis [rôle] dans une école secondaire en Belgique. Je veux créer un outil Streamlit pour [besoin]. Utilise SQLite pour stocker les données. Ajoute un export Excel. Commence par me proposer un plan en étapes.*

Parcours Web 🌐

- **Framework** : Node.js + HTML/CSS/JS
- **Déploiement** : GitHub → Vercel (gratuit, URL partageable)
- **Base de données** : Turso (SQLite cloud, gratuit) si besoin de persistance
- **Idéal pour** : apps partagées, quiz en ligne, sites, projets collaboratifs

Prompt de départ type :

> *Je suis [rôle] dans une école secondaire en Belgique. Je veux créer une application web pour [besoin]. Utilise Node.js. Elle doit être déployable sur Vercel. Commence par me proposer un plan en étapes.*

⚠️ Rappel : ce qui différencie l'exercice 2 de l'exercice 1

	Exercice 1 (Artifacts)	Exercice 2 (Agent)
Fichiers	❌	✅ Lire, créer, modifier
Base de données	❌	✅ SQLite, Turso
Upload/Download	❌	✅ Fichiers Excel, PDF, images
Persistance	❌ Session uniquement	✅ Données sauvegardées
Multi-pages	❌ Un seul fichier	✅ Autant de pages que nécessaire
Déploiement	❌	✅ Vercel, URL partageable
Appel API / internet	❌ (sauf Claudeception)	✅

Pas de doublon avec l'exercice 1 — ici on exploite tout ce que les artifacts ne pouvaient PAS faire.

Idées par profil

🎓 Profs — Planification & organisation

Idée	Description	Parcours
Séquencier pédagogique	Planifier ses séquences : objectifs, activités, durées, ressources. Vue calendrier + liste. Export PDF.	Python 🐍

Idée	Description	Parcours
Journal de classe numérique	Par classe et par jour : matière vue, devoirs, remarques. Recherche par date/classe. Historique de l'année.	Python 🐍
Planificateur annuel	Vue année scolaire avec semaines, congés, jours fériés. Répartir les chapitres sur les semaines. Drag & drop.	Web 🌐
Carnet de bord de stage	Suivi des élèves en stage : entreprises, visites, évaluations, commentaires. Vue par élève et vue globale.	Python 🐍

🎓 Profs — Évaluation & résultats

Idée	Description	Parcours
Gestionnaire de grilles d'évaluation	Créer des grilles par compétences. Encoder les résultats. Calcul automatique. Fiche de points en PDF.	Python 🐍
Suivi de compétences	Tableau élèves × compétences. Niveaux colorés. Vue radar par élève, heatmap par classe.	Python 🐍
Tableau de bord des résultats	Importer des cotes Excel. Dashboard : moyennes, distribution, évolution, élèves en difficulté.	Python 🐍
Plateforme d'exercices auto-corrigés	Exercices par chapitre, correction immédiate, stats par question. Le prof voit un dashboard classe.	Web 🌐

🎓 Profs — Production de contenu

Idée	Description	Parcours
Éditeur de cours → PDF	Écrire en markdown, organiser en chapitres. Compilation en PDF avec page de garde, table des matières, logo.	Python 🐍
Générateur d'examens	Banque de questions en DB. Piocher X questions par critères. Générer sujet + corrigé PDF. Variantes anti-triche.	Python 🐍
Site de cours en ligne	Mini-MOOC : chapitres, texte, images, vidéos embed, exercices avec correction auto. URL partageable aux élèves.	Web 🌐

💻 Profs d'informatique

Idée	Description	Parcours
Auto-correcteur de TPs	L'élève upload son fichier. Le serveur exécute des tests unitaires. Rapport : tests passés/échoués, score.	Python 🐍
Plateforme de coding challenges	Énoncés progressifs, éditeur de code, exécution côté serveur, tests auto, classement. Mini-HackerRank.	Web 🌐

Idée	Description	Parcours
Sandbox SQL avec exercices	DB pré-remplie, exercices avec requête attendue, exécution temps réel, comparaison, progression par niveau.	Web 🌐
Gestionnaire de projets d'élèves	Fiches groupe : membres, sujet, repo Git, avancement, notes. Vue kanban globale.	Web 🌐
Détecteur de similarité de code	Upload N fichiers d'élèves. Analyse de similarité (tokens). Matrice visuelle. Rapport "A et B = 87%".	Python 🐍
Gestionnaire de tickets — Atelier 404	Système de tickets d'intervention : les élèves/profs signalent un problème technique, l'atelier assigne, suit l'avancement (ouvert → en cours → résolu). Historique, stats, priorités.	Web 🌐

🏢 Direction & administration

Idée	Description	Parcours
Dashboard d'école	Upload des données existantes (Excel). Dashboard : taux de réussite par section, effectifs, absentéisme, comparaisons.	Python 🐍
Analyseur de résultats scolaires	Importer les résultats de fin de période. Distribution par classe, comparaison entre sections, tendances.	Python 🐍
Planificateur d'événements scolaires	Calendrier annuel avec tous les événements. Checklist par événement. Responsables assignés. Timeline.	Web 🌐
Outil de communication parents	Rédiger des circulaires, templates réutilisables, historique, PDF avec en-tête de l'école.	Python 🐍
Suivi des projets d'établissement	Objectifs, actions, indicateurs. Tableau de bord : avancement par axe, deadlines, statuts.	Web 🌐

🏢 Administratifs — Gestion quotidienne

Idée	Description	Parcours
Gestionnaire de réservation de locaux	Calendrier par salle, créneaux, détection de conflits, accessible à tous les profs.	Web 🌐
Inventaire matériel	PC, projecteurs, manuels, mobilier. Par local. États, dates d'achat. Alertes garantie/stock bas.	Python 🐍
Fusionneur / nettoyeur Excel	Upload plusieurs fichiers → fusion, dédoublonnage, nettoyage. Preview avant export. Le couteau suisse.	Python 🐍
Extracteur de données PDF	Upload bulletins/factures PDF → extraction des champs → tableau récapitulatif exploitable.	Python 🐍

Idée	Description	Parcours
Suivi budgétaire	Budget par poste, saisie des dépenses, reste disponible, graphiques prévu vs réel, alertes plafond.	Python 🐍

🔧 Outils transversaux

Idée	Description	Parcours
Explorateur de données universel	Déposer n'importe quel Excel/CSV → dashboard Streamlit : filtres, graphiques, tableaux croisés, export. Réutilisable à l'infini.	Python 🐍
Convertisseur d'images batch	Upload un dossier d'images → convertir PNG en WebP, redimensionner, compresser, renommer en masse. Paramètres configurables.	Python 🐍
Générateur de QR codes	Liste d'URLs ou textes → génération en masse de QR codes. Export en ZIP ou planche imprimable.	Python 🐍
Convertisseur markdown → site web	Écrire en markdown, structurer en pages, choisir un thème. Site statique déployable sur Vercel. Blog de classe, portfolio.	Web 🌐

🎮 Projets fun — Jeux & apps interactives

Idée	Description	Parcours
Quiz multijoueur live	Type Kahoot maison. Le prof projette, les élèves rejoignent via URL sur téléphone. Temps réel, classement, podium.	Web 🌐
Escape room digitale	Série d'énigmes web, chaque réponse débloque la suivante. Timer global. Indices avec pénalité. Classement équipes.	Web 🌐
Jeu de trading simulé	Portefeuille fictif, cours qui évoluent (simulés), acheter/vendre, classement par fortune. Parfait pour un cours d'éco.	Web 🌐
Système de vote / sondage live	Créer un sondage, projeter QR code. Les participants votent sur téléphone. Résultats temps réel sur écran du prof.	Web 🌐
Tournoi de classe	Bracket configurable (simple/double élimination). Saisie des résultats, avancement auto. Affichage e-sport.	Web 🌐
Système de badges/achievements	Gamification : définir des badges, attribuer manuellement, profil élève, tableau d'honneur.	Web 🌐
Jeu d'aventure textuel persistant	Scénario à choix, inventaire, stats perso, sauvegarde en DB. Jouable sur plusieurs sessions. Le prof édite le scénario.	Web 🌐

🎮 Projets fun — Jeux avancés & simulations

Idée	Description	Parcours
Jeu de plateforme 2D	Personnage, gravité, plateformes, ennemis, niveaux, éditeur de niveaux. Canvas 2D ou bibliothèque de jeu.	Web 🌐
Tower Defense	Map, vagues d'ennemis, placement de tours, upgrades, économie. Stratégie et programmation de l'IA ennemie.	Web 🌐
Roguelike	Donjon généré procéduralement, combats tour par tour, inventaire, montée en niveau, permadeath.	Web 🌐
Simulation de vie / écosystème	Proies et prédateurs dans un environnement 2D. Paramètres ajustables (reproduction, nourriture, vitesse). Observer l'équilibre ou l'extinction.	Web 🌐
Simulation de physique	Billes, ressorts, pendules, gravité — paramètres ajustables, visualisation temps réel.	Web 🌐
Simulation de trafic	Intersection avec feux tricolores, flux de voitures, ajuster les durées des feux, observer le débit.	Web 🌐
Simulation de foule	Agents autonomes qui se déplacent dans un espace avec obstacles. Comportements de groupe, évitement, sorties.	Web 🌐
Jeu de stratégie tour par tour	Map hexagonale, armées, ressources, construction, IA adverse. Mini-Civilization.	Web 🌐
Course de voitures 3D	Circuit, physique véhicule simplifiée, chrono, classement. Three.js.	Web 🌐
FPS low-poly	Environnement 3D, déplacement FPS, tir sur cibles, score. Three.js.	Web 🌐
Simulateur de vol	Avion dans un monde 3D, contrôles au clavier, paysage généré, altitude/vitesse. Three.js.	Web 🌐
Jeu de rythme	Notes qui tombent (style Guitar Hero), musique synchronisée, scoring, niveaux de difficulté.	Web 🌐

🏆 Guide de choix rapide

"Je veux un outil utile dès lundi"

→ **Grilles d'évaluation** (Python), **Journal de classe** (Python), **Séquencier** (Python)

"Je suis directeur/admin et je veux voir des chiffres"

→ **Dashboard d'école** (Python), **Analyseur de résultats** (Python), **Explorateur de données** (Python)

"J'en ai marre d'Excel"

→ **Fusionneur Excel** (Python), **Extracteur PDF** (Python), **Convertisseur d'images** (Python)

"Je veux un truc partageable par URL"

→ **Quiz multijoueur** (Web), **Site de cours** (Web), **Escape room** (Web)

"Je suis prof d'info et je veux impressionner"

→ **Auto-correcteur de TPs** (Python), **Sandbox SQL** (Web), **Tickets Atelier 404** (Web)

"Je veux m'amuser et apprendre en même temps"

→ **Tower Defense** (Web), **Simulation d'écosystème** (Web), **Roguelike** (Web)

"Le projet le plus polyvalent"

→ **Explorateur de données universel** (Python) — fonctionne avec N'IMPORTE QUEL fichier Excel/CSV

💡 Tips pendant l'exercice

Stratégie

- **50 min = un MVP.** Une fonctionnalité qui marche bien > dix fonctionnalités bancales
- **Petites étapes** : demandez à l'agent de faire une chose à la fois, testez, validez, passez à la suite
- **Multitasking** : pendant que l'agent code, réfléchissez à la suite ou travaillez sur le prompt suivant
- **Soyez exigeants** : c'est moche ? Dites-le. Pas pratique ? Dites-le. L'agent ne se vexe pas.

Technique

- **L'agent se trompe ?** Normal — décrivez le problème, il corrige. C'est le cycle.
- **Données sensibles → Python local.** Pas de noms d'élèves sur un serveur public.
- **Accessible aux autres → Web.** URL partageable, rien à installer côté utilisateur.

- **Git** : demandez à votre agent de sauvegarder régulièrement sur GitHub. "Fais un commit et pousse sur GitHub."
- **Sécurité** : avant de mettre en ligne → "Audite la sécurité de ce projet."

Debug

- Erreur dans le terminal → copier-coller dans le chat de l'agent
- L'app ne marche pas dans le navigateur → F12, console, copier l'erreur
- L'agent tourne en boucle sur le même bug → "Change d'approche" ou "Repars de zéro pour cette fonctionnalité"

Fin de l'exercice

- Parcours Web : déployez sur Vercel si c'est prêt → URL partageable
- Parcours Python : préparez une démo sur votre écran
- Tous : on partagera les créations dans le bloc 4

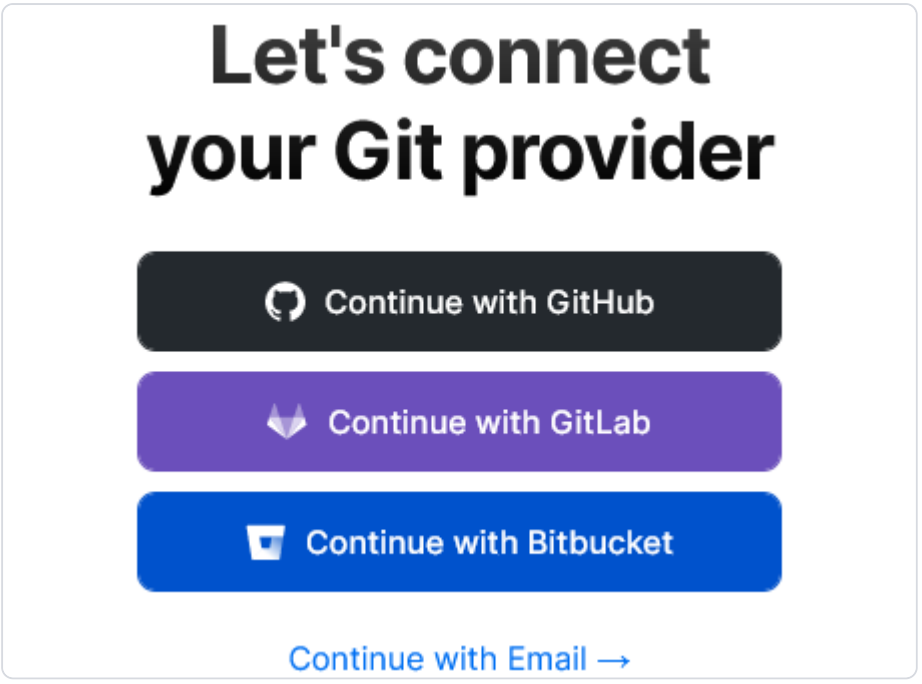
3.7 Hébergement avec Vercel

Dans cette section, nous allons apprendre comment héberger notre projet en utilisant Vercel. Vercel est une plateforme d'hébergement qui permet de déployer facilement des applications web.

Étape 1: Créer un compte Vercel

Tout d'abord, nous devons créer un compte Vercel. Pour cela, suivez les étapes suivantes :

1. Rendez-vous sur le site web de Vercel à l'adresse vercel.com.
2. Cliquez sur le bouton "Sign Up" pour créer un nouveau compte.
3. Choisissez l'option de connexion avec GitHub pour lier votre compte GitHub à Vercel.
4. Suivez les instructions pour autoriser Vercel à accéder à votre compte GitHub.



Étape 2 : Sauvegarder le projet sur GitHub depuis VS Code

Dans cette étape, nous allons voir comment initialiser un dépôt Git local, configurer votre identité Git, effectuer des commits et pousser votre code vers GitHub directement depuis Visual Studio Code (VS Code). Cela vous permettra de conserver l'historique de votre code et de le partager facilement avec d'autres, tout en simplifiant le processus de déploiement sur Vercel.

1. Initialiser un dépôt Git local

Tout d'abord, nous devons initialiser un dépôt Git dans votre projet pour commencer à suivre les modifications.

- **Ouvrez votre projet** dans VS Code.
- **Initialisez Git** :
  - Cliquez sur l'icône **Contrôle de code source** dans la barre latérale gauche.
  - Cliquez sur le bouton **"Initialiser le dépôt"**.

> ***Pourquoi ?*** *Initialiser un dépôt Git permet de commencer à suivre les modifications de votre projet. Git enregistrera les changements au fil du temps, facilitant ainsi la gestion des versions.*

2. Configurer votre identité Git

Avant de faire votre premier commit, il est important de configurer votre nom et votre adresse e-mail pour que Git puisse les associer aux commits que vous créez.

- **Ouvrez le terminal intégré** dans VS Code : **Affichage** > **Terminal**.
- **Configurez votre nom d'utilisateur** (utilisez votre pseudonyme GitHub) :

```
git config --global user.name "votre-pseudo-github"
```

- **Configurez votre adresse e-mail** (utilisez l'adresse e-mail associée à votre compte GitHub) :

```
git config --global user.email "votre-adresse-email@example.com"
```

> ***Pourquoi ?*** *Configurer votre identité Git permet d'associer vos commits à votre compte GitHub, ce qui facilite le suivi de vos contributions et la collaboration avec d'autres développeurs.*

3. Créer un premier commit

Un commit est un enregistrement des modifications apportées à votre code.

- **Sélectionnez les fichiers à committer** :
  - Dans l'onglet **Contrôle de code source**, vous verrez la liste des fichiers modifiés sous **"Changes"**.
  - Cliquez sur **"Tout ajouter"** (icône **"+"** en haut de la section) pour ajouter tous les fichiers au **staging**.
- **Entrez un message de commit** :
  - Dans le champ en haut de l'onglet, saisissez un message descriptif, par exemple, `"Initial commit"`.
- **Validez le commit** :
  - Appuyez sur **Entrée** ou cliquez sur l'icône **"✔️"**.

> ***Pourquoi ?*** *Créer des commits avec des messages clairs vous aide à comprendre quelles modifications ont été apportées et pourquoi, ce qui est essentiel pour le suivi et la maintenance du projet.*

4. Pousser le code vers GitHub

Nous allons maintenant pousser votre code vers GitHub. Si vous n'avez pas encore lié VS Code à votre compte GitHub, vous serez invité à le faire.

- **Pousser les modifications** :
  - Une fois le commit effectué, le bouton **"Synchroniser les modifications"** (icône de synchronisation 🔄) apparaît dans l'onglet **Contrôle de code source**.
  - Cliquez sur ce bouton pour pousser vos commits vers GitHub.
- **Se connecter à GitHub** :
  - Si c'est la première fois que vous poussez vers GitHub depuis VS Code, une notification apparaîtra pour vous demander de vous authentifier.
  - Cliquez sur **"Se connecter à GitHub..."**.

- Une fenêtre de navigateur s'ouvrira pour vous permettre d'autoriser VS Code à accéder à votre compte GitHub.
- **Connectez-vous à GitHub** si nécessaire et autorisez VS Code en cliquant sur **"Authorize GitHub"**.
- Revenez à VS Code après la confirmation.

> ***Pourquoi ?*** *Se connecter à GitHub depuis VS Code vous permet de gérer vos dépôts GitHub directement depuis l'éditeur, simplifiant ainsi votre flux de travail.*

5. Créer le dépôt GitHub depuis VS Code

Si vous n'avez pas encore créé de dépôt pour ce projet sur GitHub, VS Code vous proposera de le faire automatiquement.

- **Créer le dépôt distant** :
  - Après avoir cliqué sur **"Synchroniser les modifications"**, une notification apparaîtra indiquant qu'aucun dépôt distant n'est configuré.
  - Cliquez sur **"Publier sur GitHub"**.
- **Configurer le dépôt** :
  - **Choisissez la visibilité** du dépôt : sélectionnez **Public** ou **Privé** selon vos besoins.
  - **Nom du dépôt** : par défaut, le nom du dépôt sera le nom du dossier de votre projet. Vous pouvez le modifier si vous le souhaitez.
- **Confirmer la création** :
  - VS Code créera le dépôt sur GitHub et configurera le lien entre votre dépôt local et le dépôt distant automatiquement.
- **Terminer le push** :
  - VS Code poussera vos commits vers le nouveau dépôt GitHub.

> ***Pourquoi ?*** *Cette fonctionnalité vous évite d'avoir à créer manuellement le dépôt sur GitHub, rendant le processus plus fluide et rapide.*

6. Vérifier sur GitHub

- **Accédez à votre dépôt en ligne** :
  - Une notification apparaîtra avec un lien vers le dépôt GitHub nouvellement créé.
  - Cliquez sur ce lien pour ouvrir le dépôt dans votre navigateur.
  - Vous devriez voir tous les fichiers de votre projet ainsi que votre commit initial.

> ***Pourquoi ?*** *Vérifier que le code est bien en ligne confirme que toutes les étapes ont été réalisées correctement et que votre projet est prêt pour le déploiement ou la collaboration.*

7. Poursuivre le travail avec Git et VS Code

Désormais, chaque fois que vous apporterez des modifications à votre code :

- **Étape 1** : Enregistrez vos modifications.
- **Étape 2** : Allez dans l'onglet **Contrôle de code source** pour voir les fichiers modifiés.
- **Étape 3** : Ajoutez les fichiers au staging si nécessaire.
- **Étape 4** : Entrez un message de commit descriptif.
- **Étape 5** : Cliquez sur **"Valider"** pour créer un nouveau commit.
- **Étape 6** : Une fois le commit réalisé, le bouton **"Synchroniser les modifications"** apparaîtra dans l'onglet **Contrôle de code source**.
- **Étape 7** : Cliquez sur **"Synchroniser les modifications"** pour pousser vos commits vers GitHub.

> ***Pourquoi ?*** *Ce flux de travail vous permet de garder votre dépôt GitHub à jour avec vos dernières modifications, facilitant le suivi, la collaboration et le déploiement continu.*

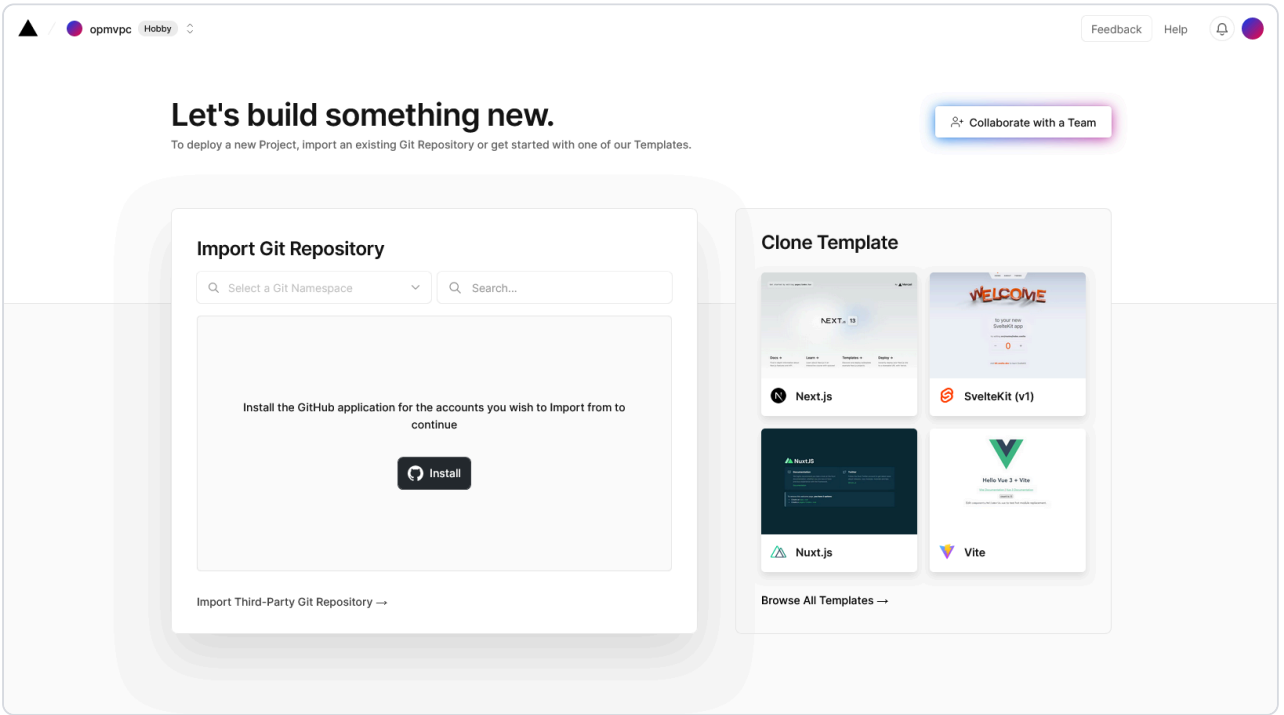
Conclusion

Vous avez maintenant sauvegardé votre projet sur GitHub en utilisant VS Code, sans avoir à créer manuellement le dépôt sur GitHub. Cette méthode simplifie le processus de mise en place du contrôle de version et facilite le déploiement sur des plateformes comme Vercel.

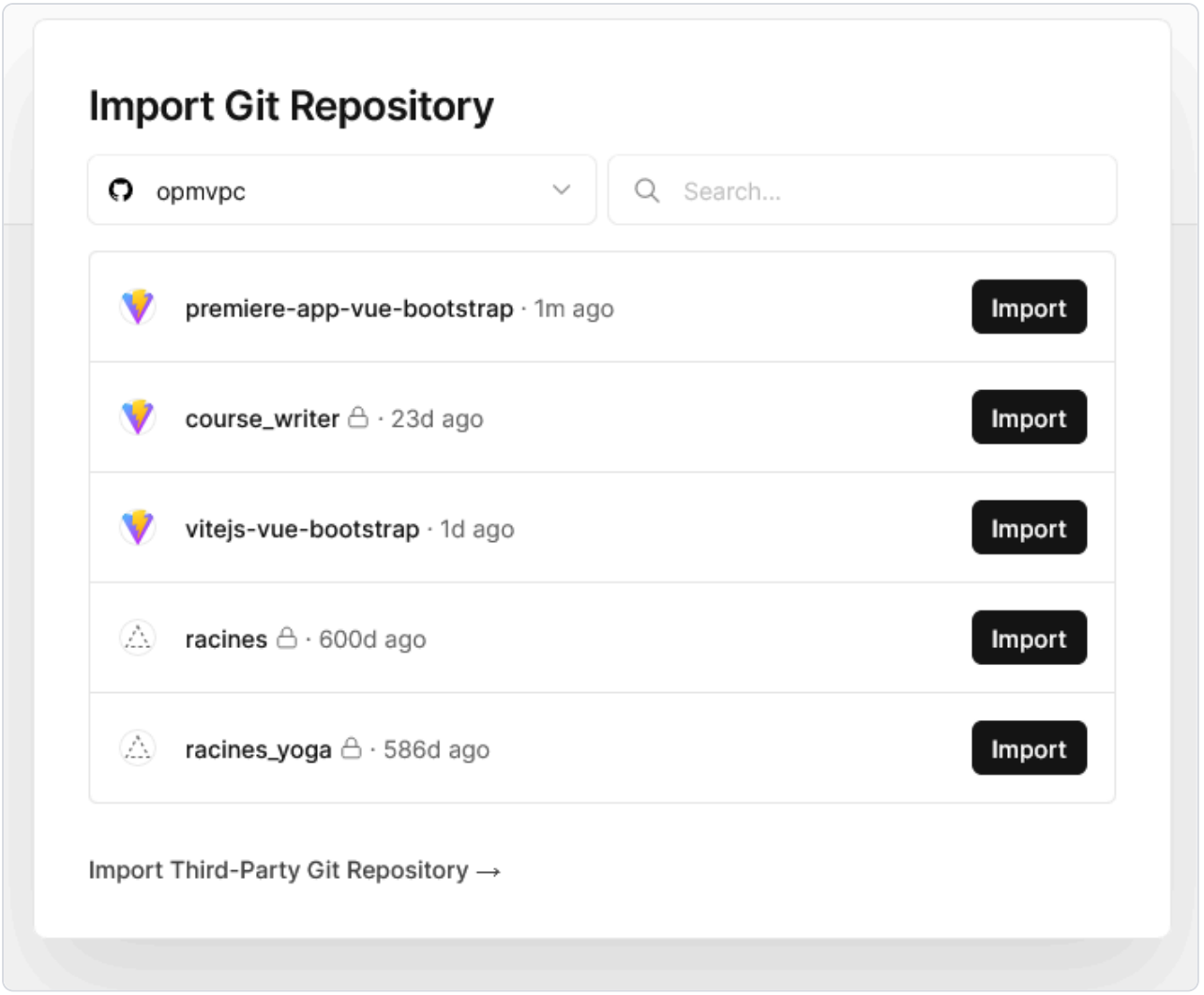
Étape 3: Lier votre compte GitHub à Vercel

Maintenant, nous allons lier notre compte GitHub à Vercel. Voici les étapes à suivre :

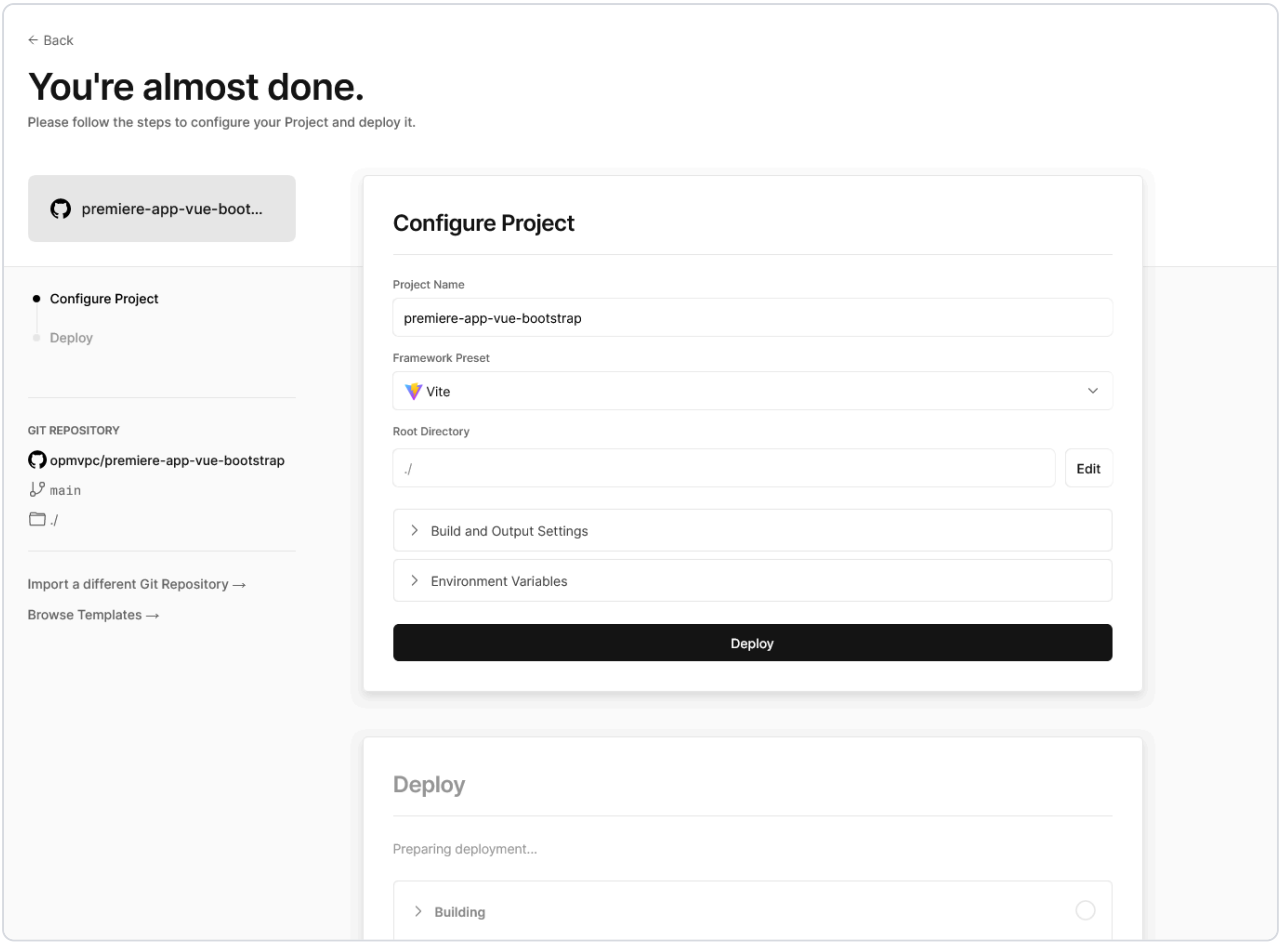
1. Retournez sur le site web de Vercel.
2. Cliquez sur le bouton "Install" pour autoriser Vercel à accéder à votre compte GitHub.



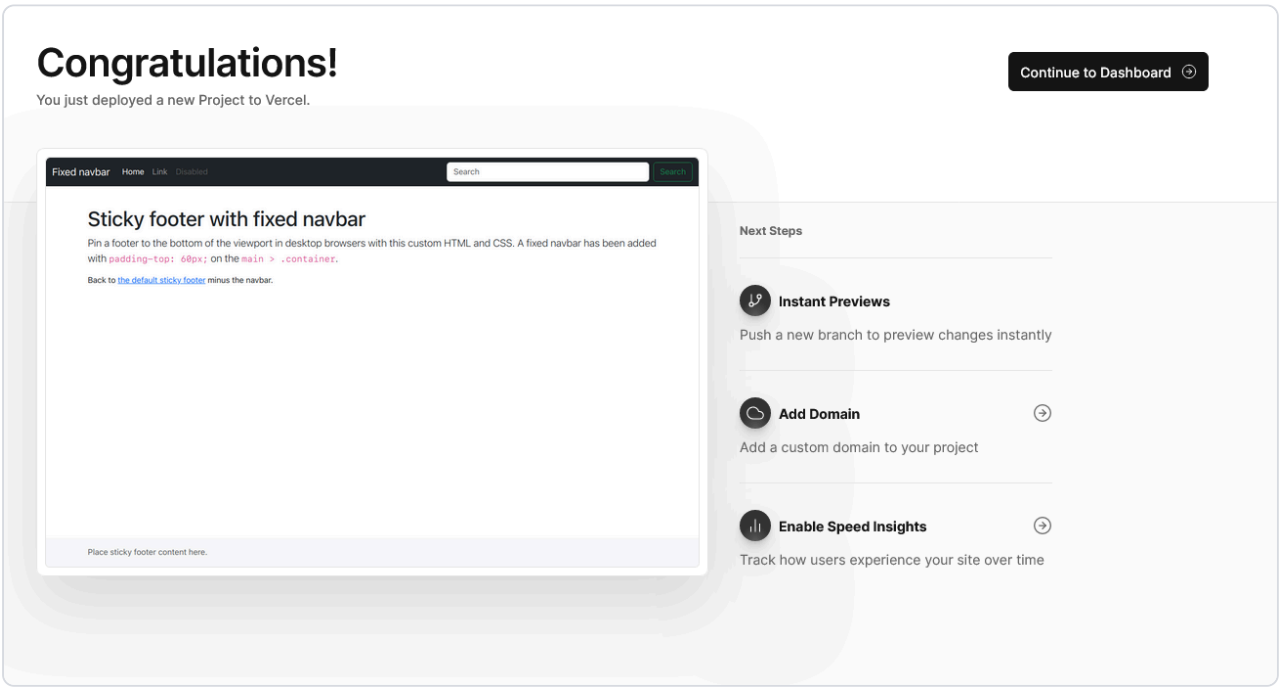
3. Une fois installé, vous verrez une liste de vos projets GitHub.



4. Cliquez sur le bouton "Import" du projet que vous venez de créer dans StackBlitz.
5. Vous serez redirigé vers la page de configuration de votre projet.



6. Vercel détectera automatiquement que vous utilisez Vue.js.
7. Cliquez sur le bouton "Deploy" pour déployer votre projet.
8. Attendez quelques secondes et normalement, si tout s'est bien passé, vous pourrez accéder à votre site en cliquant sur le lien de prévisualisation fourni par Vercel.



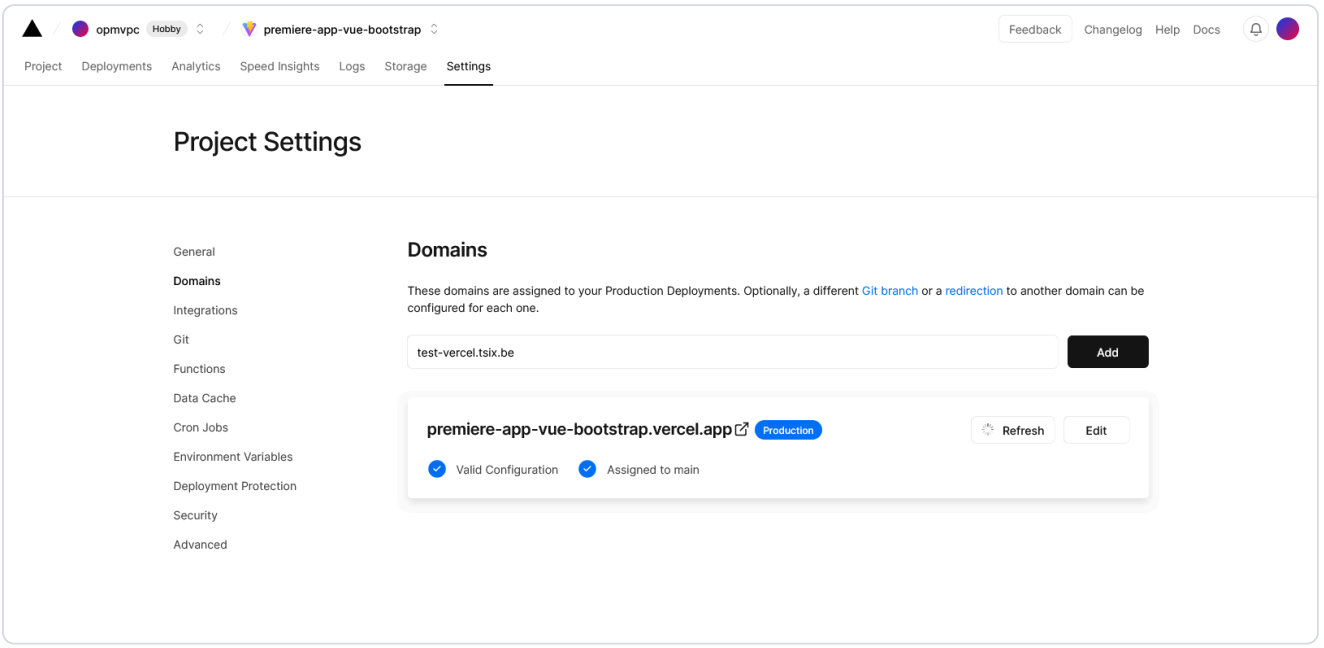
Et voilà ! Vous avez maintenant hébergé votre projet sur Vercel. Vous pouvez partager le lien de prévisualisation avec d'autres personnes pour qu'elles puissent voir votre site en ligne.

Chaque fois que vous ferez des modifications sur cette application, la nouvelle version sera automatiquement mise en ligne.

Nom de domaine

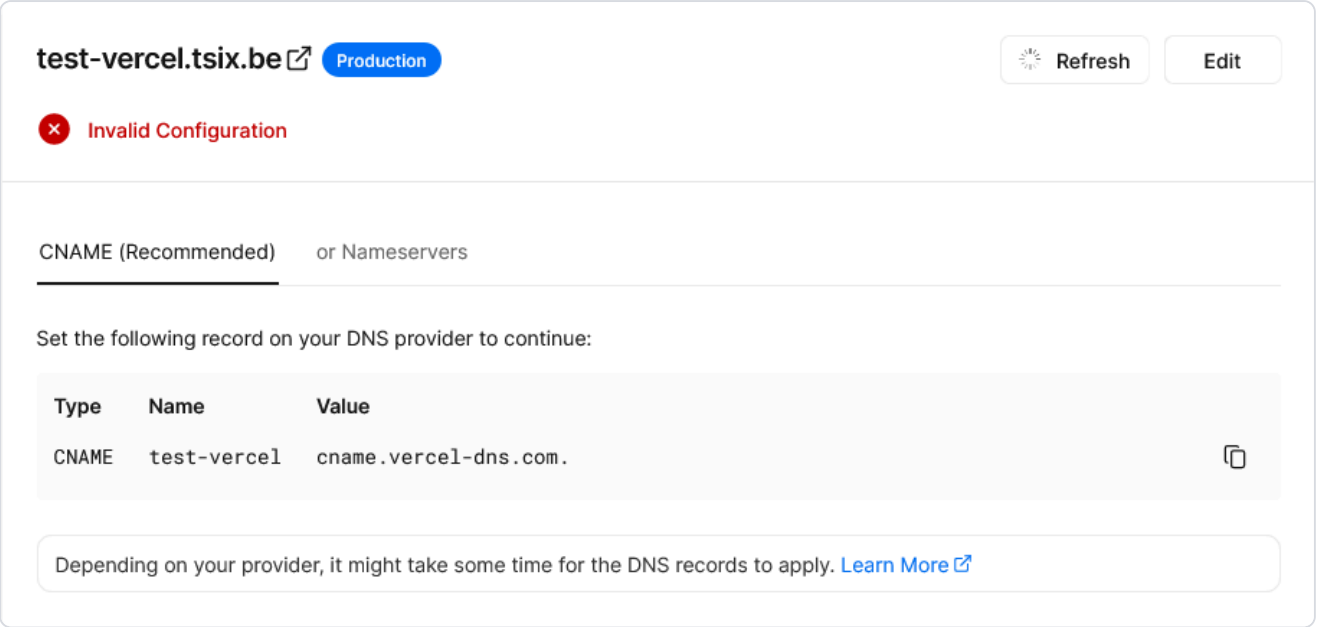
Vous pouvez lier votre nom de domaine à un site hébergé sur Vercel.

Pour cela, allez dans les paramètres de votre projet, puis dans l'onglet "Domains".

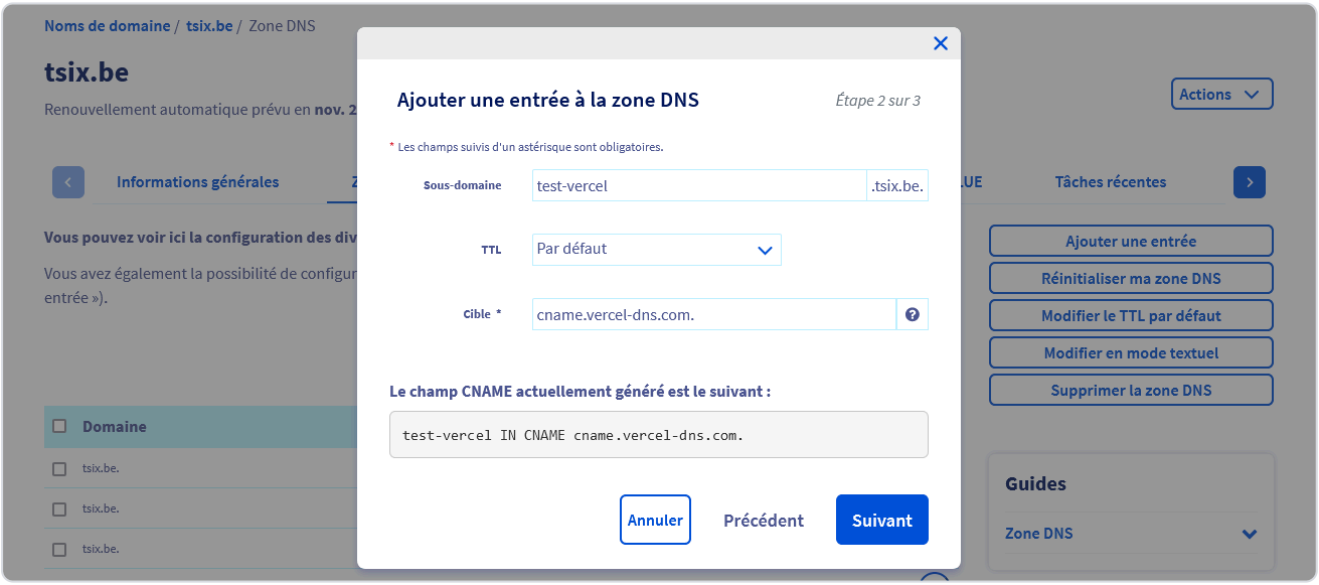


Ajoutez votre nom de domaine avec un sous domaine, puis cliquez sur “Add”.

Nous devons ensuite suivre les instructions.



Allez dans l’interface de gestion de votre nom de domaine et ajoutez un champs CNAME avec les paramètres fournis.



Nous pouvons désormais accéder à notre application avec notre nom de domaine. Vercel émet automatiquement un certificat SSL pour notre site.

CHAPITRE 4

Bloc 4 — Et après ?

Durée : 15 min | **Type** : Théorie + partage | **Slides** : ~5-6



Slides du chapitre

- Et après.pdf

4.1 — Partage des créations (5 min)

Chacun montre sa meilleure réalisation. Les apps web sont accessibles par URL, les apps Python en démo sur l'écran du participant.

Pas besoin que ce soit fini — ce qui compte c'est le potentiel. Vous avez 50 minutes d'expérience. Imaginez ce que vous ferez en une journée.

4.2 — Et après ? Inspiration et pistes (10 min)

Vous avez maintenant les bases : la méthode, les outils, les fichiers de pilotage. Voici les pistes pour aller plus loin.

🌐 Ajouter un navigateur à votre agent (MCP)

Par défaut, votre agent sait lire/écrire des fichiers et exécuter des commandes dans le terminal. Avec **MCP** (Model Context Protocol), vous pouvez lui ajouter des super-pouvoirs — et le plus utile pour commencer, c'est un **navigateur**.

Un agent avec un navigateur peut :

- Tester votre site automatiquement (naviguer, cliquer, vérifier le résultat)
- Faire des recherches web pendant qu'il code
- Prendre des screenshots de votre app pour debug
- Interagir avec des services en ligne

Comment ajouter Chrome DevTools MCP

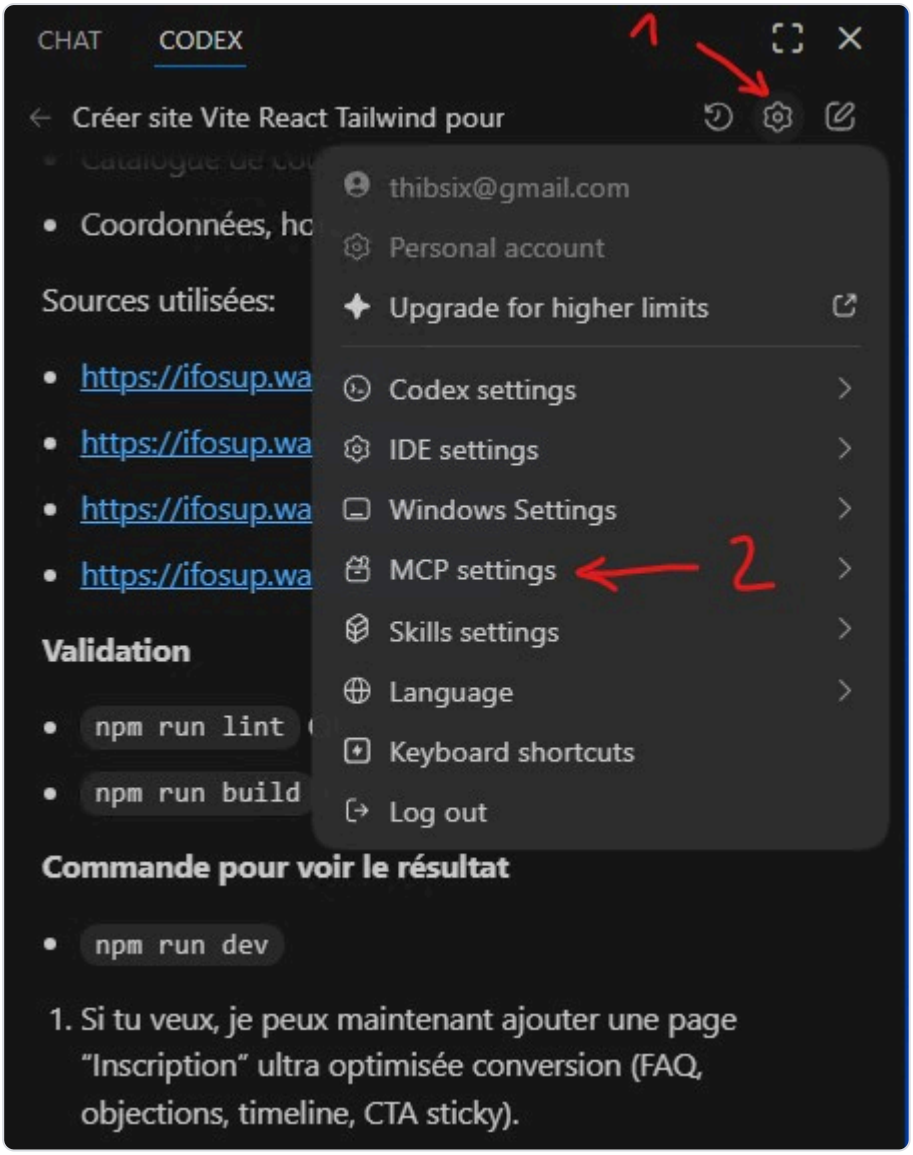
Le serveur MCP **Chrome DevTools** est le plus simple à configurer. Voici comment faire selon votre agent :

Codex (dans VSCode)

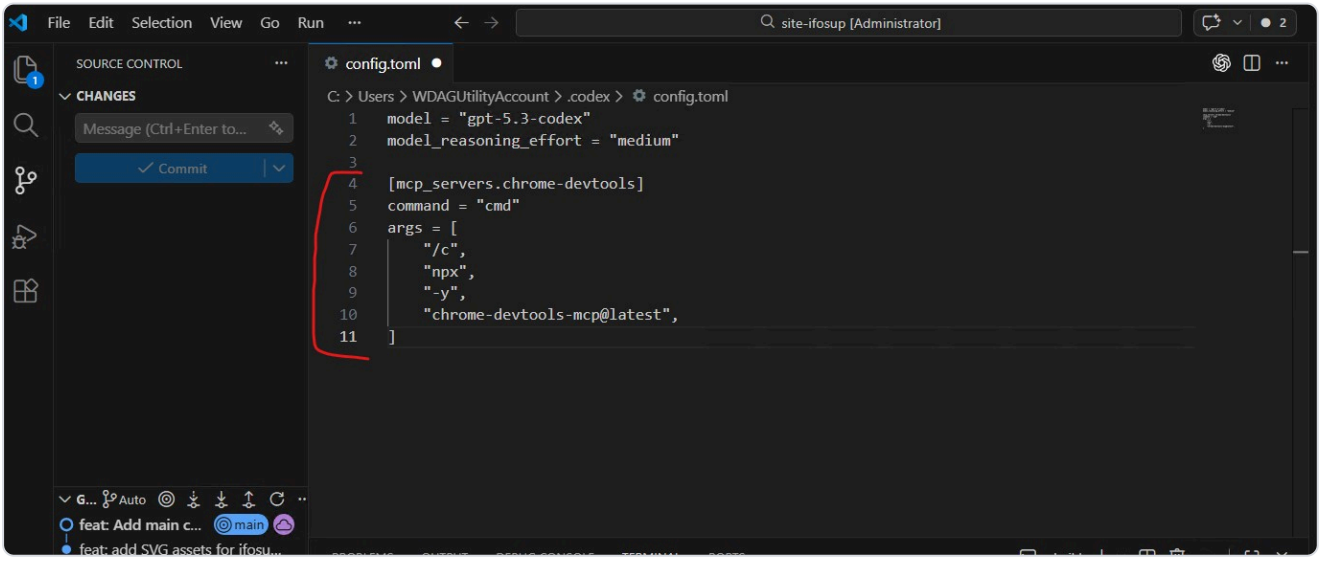
Deux options :

Option 1 — Via l'interface :

1. Ouvrez le panneau Codex
2. Cliquez sur l'engrenage ⚙️ en haut à droite
3. Sélectionnez **"MCP settings"**
4. Cliquez **"Open config.toml"**
5. Ajoutez la configuration



Le fichier `config.toml` ressemble à ça :



Ajoutez ceci puis relancez VS code :

```
[mcp_servers.chrome-devtools]
command = "cmd"
args = [
  "/c",
  "npx",
  "-y",
  "chrome-devtools-mcp@latest",
]
```

Option 2 — Via le terminal (si vous avez le CLI Codex installé) :

```bash
codex mcp add chrome-devtools -- npx chrome-devtools-mcp@latest
```

Claude Code (terminal)

```bash
claude mcp add chrome-devtools --scope user npx chrome-devtools-mcp@latest
```

GitHub Copilot / Cursor (VSCode natif)

VSCode a son propre système MCP. Ouvrez la palette de commandes (`Ctrl+Shift+P`), tapez **"MCP: Add Server"**, et suivez les instructions. Ou créez un fichier `.vscode/mcp.json` :

```json
{
  "servers": {
    "chrome-devtools": {
      "command": "npx",
      "args": ["-y", "chrome-devtools-mcp@latest"]
    }
  }
}
```

💡 **Il existe des centaines de serveurs MCP** : GitHub, bases de données, Slack, Google Drive, Figma... Explorez sur https://github.com/modelcontextprotocol/servers

🔀 Multitasking : travailler pendant que l'agent code

Quand l'agent travaille sur une tâche longue, vous n'êtes pas obligé de le regarder faire. Pendant qu'il code :

- Réfléchissez à la phase suivante
- Préparez vos données de test
- Travaillez sur un autre projet dans une autre fenêtre VSCode
- Rédigez la documentation

C'est un changement fondamental : votre temps de travail n'est plus bloqué par le temps de codage. L'agent code, vous orchestrez.

🤖 Multi-agents : la vraie puissance



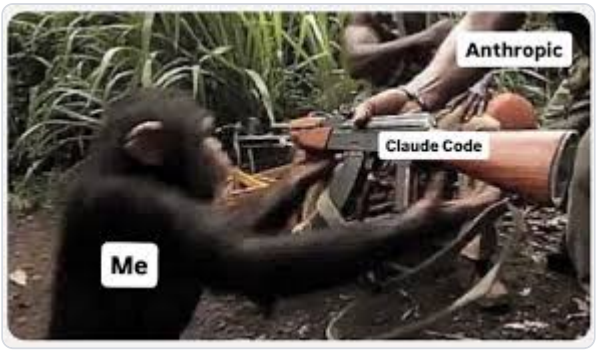
Ça, c'est le niveau suivant. Un développeur qui pilote **plusieurs agents en parallèle** sur le même projet ou sur des projets différents. Chaque fenêtre VSCode a son propre agent qui travaille de manière autonome.

Pour éviter que les agents se marchent dessus sur un même projet, on utilise des **git worktrees** — des copies de travail parallèles du même dépôt, chacune sur sa propre branche :

```bash
# Agent 1 travaille sur le frontend
git worktree add ../mon-projet-frontend feature/frontend

# Agent 2 travaille sur l'API
git worktree add ../mon-projet-api feature/api
```

Quand les deux ont fini, on fusionne les branches. C'est de l'orchestration avancée, mais c'est là que les gains de productivité deviennent exponentiels.



💡 *Le mème est drôle, mais il dit quelque chose de vrai : les outils sont puissants, peut-être même plus que ce qu'on sait en faire aujourd'hui. L'important c'est d'apprendre à les manier — c'est exactement ce qu'on vient de commencer.*

📝 L'agent ne code pas que du code

L'agent peut aussi vous aider pour des tâches non-code :

- **Rédaction de cours** : donnez-lui vos références, il structure un cours complet en markdown
- **Correction de travaux** : il analyse les soumissions et produit un rapport
- **Traitement de données** : nettoyage, fusion, analyse de fichiers Excel/CSV
- **Documentation** : il documente votre code, crée des README, des guides utilisateur
- **Préparation de réunions** : résumés, ordres du jour, comptes-rendus

🔮 Le futur : ça va devenir de plus en plus facile

Les modèles progressent à une vitesse exponentielle (on l'a vu avec les benchmarks au début du cours). Ce qui est compliqué aujourd'hui sera trivial dans 6 mois. Les outils deviennent plus intégrés, plus simples, plus accessibles.

Votre avantage : vous avez la **méthode**. Les outils changent, la méthode reste.

Le message final

> *Après ce cours, je ne serai plus là pour vous aider au quotidien. Mais votre IA, elle, sera toujours là.*
>
> *Vous avez maintenant la méthode et les outils. Ce n'est qu'une question de **curiosité**, d'**imagination** et de **persévérance**.*
>
> *Lancez-vous, expérimentez, et n'ayez pas peur de casser des trucs — c'est comme ça qu'on apprend.*

Conseils pratiques pour la suite

Conseil	Pourquoi
Apps jetables vs maintenance	Parfois c'est plus rapide de refaire en mieux que de maintenir. Vous connaissez mieux le problème, vous évitez les erreurs.
Commencez petit	Un outil simple qui marche > un projet ambitieux jamais fini
Combinez les abonnements	Un seul abo suffit pour démarrer. Pour travailler sérieusement : 2-3 abos complémentaires.
Les prix et modèles changent	Ce qui est cher ou limité aujourd'hui sera peut-être gratuit dans 3 mois. Restez curieux.
Partagez avec vos collègues	Un outil que vous créez peut servir à toute l'équipe.

Ressources

- **MCP** : https://modelcontextprotocol.io — le standard pour étendre les agents
- **Chrome DevTools MCP** : https://github.com/anthropics/anthropic-cookbook — navigateur pour agents
- **Codex** : https://developers.openai.com/codex — documentation officielle
- **Claude Code** : https://code.claude.com — documentation officielle
- **Cursor** : https://cursor.com — éditeur avec IA intégrée
- **Git & GitHub** : https://docs.github.com — versioning
- **Vercel** : https://vercel.com/docs — déploiement web
- **Coolors** : https://coolors.co — palettes de couleurs