

# Conquête : Math assistant.

**Durée :** 1.5 Semaines (11 Jours)

## La Mission

Bienvenue dans la "Vibe-code Conquest" ! Votre défi est de construire un "Assistant IA de Mathématiques" pour le web, inspiré d'outils comme Foondamate. L'objectif est simple : un étudiant est bloqué sur un problème de maths dans son livre, il prend une photo, et votre application non seulement *résout* le problème, mais lui *explique* comment y arriver.

Ce projet est une excellente opportunité d'apprendre en construisant, en particulier pour des développeurs juniors. Vous allez manipuler des APIs externes, connecter un frontend et un backend, et créer une application complète.

Vous pouvez utiliser **Cursor ou autre IDE avec I.A intégrée** pour vous aider dans cette tâche.

## Flux Utilisateur Principal

1. **Capture** : L'utilisateur envoie (upload) une image d'un problème de maths (ex:  $f(x) = x^2$ , trouver  $f'(x)$ ).
2. **Traduction** : L'application analyse l'image, extrait l'équation et l'affiche en format LaTeX propre (pour que l'utilisateur confirme que c'est correct).
3. **Résolution & Explication** : L'utilisateur clique sur "Résoudre". L'application fournit une réponse et, surtout, une explication détaillée étape par étape.
4. **Exportation** : L'utilisateur peut sauvegarder cette solution dans un fichier PDF.

## Tech Stack - OBLIGATOIRE

Votre projet **doit** utiliser les technologies suivantes :

- **Frontend : React + Vite**
- **Backend : Python** (avec Flask ou FastAPI) ou **JavaScript** (avec **Node.js** + Express)
- **IDE : Cursor** (vous devez l'utiliser pour développer)
- **Conversion Image -> LaTeX : API Mathpix** (C'est le service que vous devez utiliser pour l'OCR de maths)
- **Moteur de Solution (Hybride)** : Vous devez utiliser les **deux** :
  1. **API WolframAlpha** (pour le calcul symbolique et la réponse "pure")
  2. **Un LLM** (API Gemini ou OpenAI) (pour comprendre la question, générer l'explication et *tutoyer* l'utilisateur)

## Exigences Fonctionnelles

Votre projet final **doit** inclure ces fonctionnalités :

- **F1. Upload d'Image** : Une interface utilisateur claire pour envoyer un fichier image.
  - *Bonus* : Permettre à l'utilisateur d'utiliser directement la caméra de son téléphone ou de son ordinateur.
- **F2. Traduction Image-vers-LaTeX** : Le backend doit envoyer l'image à l'**API Mathpix** et récupérer la chaîne de caractères LaTeX.
- **F3. Rendu LaTeX** : Le frontend React doit afficher la chaîne LaTeX (ex: " $\frac{1}{2}$ ") de manière lisible, en utilisant une bibliothèque comme **KaTeX** ou **MathJax**.
- **F4. Moteur de Solution Hybride** : C'est le cœur du projet.
  1. Le backend doit envoyer le problème LaTeX à l'**API WolframAlpha** pour obtenir une solution calculée.
  2. Le backend doit *aussi* envoyer le problème (et peut-être la réponse de WolframAlpha) à un **LLM** (Gemini/OpenAI) avec un prompt lui demandant de générer une explication pédagogique, étape par étape.
  3. Votre app doit combiner intelligemment ces deux sources.
- **F5. Affichage Étape par Étape** : Ne montrez pas seulement la réponse finale. L'interface doit présenter l'explication de manière claire, en formatant le texte et les équations.
- **F6. Export PDF** : Un bouton "Télécharger en PDF" qui sauvegarde la solution (problème + étapes + réponse) dans un fichier PDF (par exemple, en utilisant jsPDF côté frontend).

## Exigences de Design (UI / UX)

L'expérience utilisateur est cruciale. Une solution puissante mais inutilisable est un échec.

- **D1. Clarté et Simplicité (Minimalisme)** :
  - L'interface doit être épurée et aller droit au but. La priorité absolue est de guider l'utilisateur à travers le flux (Upload -> Résultat) sans aucune friction.
  - Évitez les éléments visuels superflus qui ne servent pas directement l'objectif.
- **D2. "Mobile-First" (Responsive Design)** :
  - Votre public cible (les étudiants) utilisera majoritairement son **téléphone mobile**.
  - Votre application doit être *parfaitement* utilisable et esthétique sur un petit écran. Pensez à "Mobile-First", puis adaptez le design pour les écrans plus larges (tablette, bureau).
- **D3. Lisibilité Maximale** :
  - C'est une application de lecture. Le texte et les formules mathématiques doivent être grands, clairs et bien espacés.
  - Utilisez une police de caractères simple et très lisible (ex: Inter, Lato, ou les polices système).
  - Le rendu LaTeX (via KaTeX ou MathJax) doit être impeccable et s'aligner correctement avec le texte.
- **D4. Retours Visuels (Feedback)** :

- L'utilisateur ne doit jamais se demander "Est-ce que ça marche ?".
- Affichez des **indicateurs de chargement** (spinners, barres de progression) clairs lorsque le backend travaille (upload, appel API Mathpix, appels IA).
- Affichez des **messages d'erreur clairs** et amicaux (ex: "Oups, l'image est trop floue. Essayez de prendre une photo plus nette." au lieu de "Erreur 500").
- **D5. Zone d'Upload Intuitive :**
  - L'élément central de la page d'accueil doit être la zone de capture/upload. Rendez-la évidente.
  - Utilisez un gros bouton "Prendre une photo" ou une zone "Glisser-déposer" claire.

## Bonnes Pratiques de Développement (Important !)

Puisque vous êtes développeur junior, c'est le moment parfait pour adopter de bonnes habitudes.

- **Gestion de Source (Git) :**
  - Utilisez **GitHub** pour héberger votre code.
  - Faites des petits et fréquents commits.
  - Écrivez des messages de commit clairs (ex: "Feat: Ajout du bouton d'upload au lieu de "maj").
  - N'hésitez pas à utiliser des branches pour les nouvelles fonctionnalités (feat/login, fix/pdf-export, etc.).
- **Qualité de Code (Linting) :**
  - Vous devez configurer un *linter* pour votre projet.
  - **Frontend (React) :** Utilisez **ESLint** (souvent déjà inclus ou facile à ajouter avec Vite) pour garder votre code JS/React propre.
  - **Backend (Python) :** Utilisez **Flake8** (pour détecter les erreurs) et **Black** (pour formater automatiquement)
  - **Backend (Node.js) :** Utilisez **ESLint** avec les plugins appropriés.
- **Lisibilité :** Votre code doit être facile à lire. Ajoutez des commentaires là où c'est nécessaire.

## Critères de Victoire

Votre projet sera évalué sur les points suivants.

### (40%) - Fonctionnalité de Base & Fiabilité

- Est-ce que le flux principal (Image -> LaTeX -> Résolution -> PDF) fonctionne sans planter ?
- La connexion avec **Mathpix**, **WolframAlpha** et le **LLM** est-elle réussie et stable ?
- L'application gère-t-elle les erreurs (ex: une image illisible) ?

### (30%) - Qualité de la Solution (L'Approche Hybride)

- C'est le critère le plus important.
- Est-ce que l'application se contente de cracher la réponse de WolframAlpha (score faible) ?
- Ou est-ce qu'elle utilise le **LLM pour synthétiser** une explication humaine et pédagogique, en s'appuyant sur l'exactitude de WolframAlpha (score élevé) ? C'est ce qu'on veut voir !

### (15%) - Bonnes Pratiques & Qualité du Code

- Avez-vous suivi les bonnes pratiques de développement ?
- Le dépôt GitHub est-il propre, avec un historique de commits compréhensible ?
- Le code est-il *linté* et formaté correctement ?
- Le code est-il lisible et bien organisé ?

### (15%) - Expérience Utilisateur (UX/UI) & "Wow" Factor

- L'application est-elle intuitive, propre et *responsive* (utilisable sur mobile) ?
- L'affichage des maths et du texte est-il agréable à lire ?
- *Facteur "Wow" (Bonus)* : Avez-vous ajouté quelque chose en plus ?
  - Ex: Une interface de "chat" pour poser des questions de suivi sur la solution.
  - Ex: Une gestion des erreurs très soignée pour l'utilisateur.

## Livrables

1. Un lien vers votre dépôt GitHub (public ou privé, en donnant l'accès aux juges).
2. Une courte vidéo de démonstration (3-5 minutes) de votre application.

## Resources

<https://mathpix.com/>

<https://products.wolframalpha.com/simple-api/documentation>

<https://ai.google.dev/gemini-api/docs>