

Travail de Bachelor

Rapport Intermédiaire

**Outil d'aide à la génération automatique de quiz**

|  |  |
| --- | --- |
| **Département** | TIC |
| **Filière** | Informatique et systèmes de communication |
| **Orientation** | Informatique logicielle |
|  |  |
| **Etudiant** | Boris Hutzli |
| **Supervisé par** | Nastaran Fatemi |
| **Date** | 23.05.2025 |

**Table des matières**

[1 Introduction 7](#_Toc198911756)

[1.1 Description du projet 7](#_Toc198911757)

[1.2 Problématique 7](#_Toc198911758)

[1.3 Objectif 7](#_Toc198911759)

[1.4 Portée 8](#_Toc198911760)

[2 État de l’art 9](#_Toc198911761)

[2.1 Définitions et concepts 9](#_Toc198911762)

[2.1.1 Critères de qualité d’une évaluation pédagogique 9](#_Toc198911763)

[2.1.2 Grand modèle de langage (LLM) 9](#_Toc198911764)

[2.1.3 Approches d’adaptation des LLMs 11](#_Toc198911765)

[2.1.4 Vectorisation et recherche de similarités 12](#_Toc198911766)

[2.1.5 Prompt Engineering 14](#_Toc198911767)

[2.2 Travaux et technologies existantes 15](#_Toc198911768)

[2.2.1 Plateformes de génération de contenu pédagogiques 15](#_Toc198911769)

[2.2.2 Comparaison des plateformes existantes 15](#_Toc198911770)

[2.2.3 Modèles LLMs exitstants 17](#_Toc198911771)

[3 Planification initiale 19](#_Toc198911772)

[3.1.1 Liste des tâches 19](#_Toc198911773)

[4 Conception 21](#_Toc198911774)

[4.1 Architecture 21](#_Toc198911775)

[4.2 Choix technologiques 22](#_Toc198911776)

[4.2.1 Application 22](#_Toc198911777)

[4.2.2 Hébergement 22](#_Toc198911778)

[4.2.3 Stockage des fichiers 23](#_Toc198911779)

[4.2.4 Base de données 24](#_Toc198911780)

[4.2.5 ORM 24](#_Toc198911781)

[4.2.6 Modèle LLM 25](#_Toc198911782)

[4.3 Modélisation des données 25](#_Toc198911783)

[4.3.1 Modèle conceptuel 26](#_Toc198911784)

[4.3.2 Schéma JSON des évaluations 27](#_Toc198911785)

[4.4 Maquette initiale 29](#_Toc198911786)

[5 Implémentation actuelle 35](#_Toc198911787)

[5.1.1 Authentification 35](#_Toc198911788)

[5.1.2 Routes API 37](#_Toc198911789)

[5.1.3 Base de données 37](#_Toc198911790)

[5.1.4 Dashboard évaluations 39](#_Toc198911791)

[5.1.5 Génération des évaluations 39](#_Toc198911792)

[5.1.6 Mode affichage 42](#_Toc198911793)

[5.1.7 Dashboard fichiers 44](#_Toc198911794)

[6 Conclusion 45](#_Toc198911795)

[7 Utilisation de l’IA 46](#_Toc198911796)

[8 Bibliographie 47](#_Toc198911797)

**Table des figures**

[Figure 1 Architecture d'un modèle basé sur un Transformer 10](#_Toc198911824)

[Figure 2 Total heures planifiées par catégorie de tâche 20](#_Toc198911825)

[Figure 3 Schéma d’architecture du projet 21](#_Toc198911826)

[Figure 4 Modèle conceptuel 26](#_Toc198911827)

[Figure 5 Dashboard évaluations 29](#_Toc198911828)

[Figure 6 Nouvelle évaluation 30](#_Toc198911829)

[Figure 7 Mode affichage 30](#_Toc198911830)

[Figure 8 Mode test 31](#_Toc198911831)

[Figure 9 Mode test agrandi 32](#_Toc198911832)

[Figure 10 Résultats 32](#_Toc198911833)

[Figure 11 Résultats détails 33](#_Toc198911834)

[Figure 12 Dashboard fichiers 34](#_Toc198911835)

[Figure 13 Implémentation actuelle du dashboard des évaluations 39](#_Toc198911836)

[Figure 14 Formulaire de création d'une évaluation 39](#_Toc198911837)

[Figure 15 Mode affichage d'une évaluation 42](#_Toc198911838)

[Figure 16 Edition d'une question 42](#_Toc198911839)

[Figure 17 Démonstration de régénération de question 43](#_Toc198911840)

[Figure 18 Démonstration de régénération de question 43](#_Toc198911841)

[Figure 19 Code de régénération d'une question 44](#_Toc198911842)

[Figure 20 Implémentation actuelle du dashboard des fichiers 44](#_Toc198911843)

**Table des tableaux**

[Tableau 1 Techniques de prompt engineering 14](#_Toc198911844)

[Tableau 2 Résumé des plateformes existantes 16](#_Toc198911845)

[Tableau 3 Liste des tâches 20](#_Toc198911846)

[Tableau 4 Routes API liées aux évaluations 37](#_Toc198911847)

[Tableau 5 Routes API liées aux utilisateurs 37](#_Toc198911848)

**Table des listings**

[Listing 1 Schéma JSON d’une évaluation au format « JSON Schema Draft 2020-12 » 28](#_Toc198904736)

[Listing 2 Création d'un JWT 35](#_Toc198904737)

[Listing 3 Pseudocode middleware 36](#_Toc198904738)

[Listing 4 Schéma Prisma actuel 38](#_Toc198904739)

[Listing 5 Mise en place du client Prisma 38](#_Toc198904740)

[Listing 6 Génération de contexte 40](#_Toc198904741)

[Listing 7 Extrait de contexte généré 41](#_Toc198904742)

[Listing 8 Génération d'une évaluation 41](#_Toc198904743)

# Introduction

## Description du projet

Le projet consiste à développer une application web de génération, gestion et personnalisation d'évaluations (quiz, exercices, travaux écrits) en utilisant des technologies basées sur des modèles de langage (LLMs).

L'objectif n'est pas simplement de générer du contenu automatiquement, mais de proposer un environnement interactif, structuré et contrôlable pour accompagner les utilisateurs dans la conception efficace et sur-mesure d'évaluations pédagogiques adaptées à leur contexte.

Cette plateforme combinera la puissance de l'IA générative avec une interface ergonomique et des outils de révision, de personnalisation et de gestion pour répondre aux exigences pédagogiques des utilisateurs.

## Problématique

Aujourd'hui, de nombreux enseignants passent un temps considérable à préparer des quiz, exercices et travaux écrits. Il leur faut inventer des questions claires, équilibrées, variées, et adaptées au niveau des étudiants et aux objectifs pédagogiques. En outre, les étudiants manquent souvent des exercices et des évaluations d'entraînement pertinents pour les permettre de s'entraîner de manière efficace et autonome.

Les LLMs existants sont des outils puissants pour la génération automatique de contenu, mais leur utilisation brut présente plusieurs limites, telles que :

* Manque de structure/cohérence dans les évaluations générées
* Risque d'erreurs ou de contenu non conforme aux attentes
* Gestion et sauvegarde des évaluations générées très limité
* Interface peu adaptée pour une utilisation récurrente et collaborative

## Objectif

C'est ici que cette application apporte une réelle plus-value. Elle offre une interface dédiée à la création d'évaluations assistée par IA, qui guide les utilisateurs dans la formulation de leurs besoins, tout en leur offrant :

* Une base de données pour gérer et retrouver leurs évaluations
* Un système de modification et régénération fine de chaque question spécifique
* Un espace de gestion des fichiers de cours liés
* Des garanties de structure et de format des évaluations générées
* Une personnalisation du comportement du LLM pour répondre aux attentes (niveau de difficulté, types de questions, quantité de questions, etc)
* Un moyen d'effectuer l'évaluation directement sur la plateforme avec un retour sur les réponses fournies

## Portée

La portée du projet se concentre principalement sur la génération de contenu pédagogique dans un cadre technique, plus précisément pour les étudiants et enseignants au sein de la HEIG-VD dans la filière d’informatique et systèmes de communication. Cette spécification du public cible permettrait d’optimiser plus facilement la pertinence du contenu généré, en l’adaptant aux besoins spécifiques d’un domaine.

À l’inverse, l’implémentation d’une solution plus généraliste destinée à un public cible interdisciplinaire, nécessiterait la prise en compte de multiples points de vue des différentes filières afin de garantir la pertinence du contenu généré. Ceci complexifierait considérablement la conception et risquerait de restreindre la qualité du produit.

# État de l’art

## Définitions et concepts

### Critères de qualité d’une évaluation pédagogique

Comprendre les différents critères de qualité d’une évaluation pédagogique va servir à guider le développement de la plateforme, où il est nécessaire de générer du contenu adéquat, efficace, et de qualité.

Les critères principaux trouvés, après plusieurs recherches, sont les suivantes (1) (2) (3) :

* **Qualité de la langue :** Les règles de la langue doivent être respectées, que cela soit au niveau de la grammaire, l’orthographe, la syntaxe ou la ponctuation.
* **Validité :** L’évaluation doit mesurer les objectifs d’apprentissage du cours et uniquement ceci. L’évaluation doit être pertinente par rapport au contenu et les compétences à évaluer, et la difficulté doit être ajustée pour correspondre au mieux au niveau des apprenants.
* **Discriminativité :** L’évaluation doit pouvoir optimalement déterminer les apprenants ayant bien compris le sujet et ceux l’ayant pas maîtrisé. Donc il faut éviter autant que possible le hasard, et des questions non pertinents au sujet. Un apprenant qui ne connaît pas la bonne réponse ne devrait pas pouvoir la déduire.
* **Clarté :** Les questions doivent être compréhensible et précis, évitant toute ambiguïté. Idéalement fournir des exemples pour les questions plus tordues.
* **Structure :** Les questions doivent être indépendantes les unes les autres, ne pas faire référence à une autre question ou donner un indice qui peut être utilisé plus tard.

### Grand modèle de langage (LLM)

Un modèle de langage (Large Language Model, LLM) est une catégorie d’IA entraîné pour la reconnaissance, compréhension et génération de texte ou de langages humains. Ces modèles s’appuient sur l’apprentissage en profondeur (Deep Learning) sur un très grand ensemble de données, lui permettant de prédire, mot après mot, le texte le plus pertinent à générer à partir d’une entrée. (4)

**Architecture d’un LLM**

L’architecture d’un LLM est déterminée par plusieurs facteurs, tels que les objectifs du modèle (résumé, traduction, etc), les ressources computationnelles disponibles, et le type de tâche de traitement de langage à accomplir. Les éléments importants qui influencent l’architecture d’un LLM sont (5) :

* La taille du modèle (nombre de paramètres)
* La représentation des données d’entrée (tokenization)
* Les objectifs d’apprentissage
* Efficacité computationnelle (vitesse, parallélisation)
* Le décodage et la génération de texte

**Transformer**

Les modèles LLM actuels s’appuient principalement sur l’architecture Transformers, qui a révolutionné les tâches de traitement de langage naturel. Cette architecture introduit le mécanisme de self-attention, qui permet au modèle de comprendre le contexte global d’une phrase en analysant tous les mots en parallèle, plutôt que séquentiellement comme dans les RNN (Recurrent Neural Networks) (6).

Les anciens modèles (RNN) lisent le texte mot par mot, ce qui les rend vulnérables au problème du vanishing gradient. Cela les empêche souvent de faire des liens entre des mots ou expressions dépendants mais éloignés dans une phrase.

Par exemple dans la phrase :

« L’étudiant est allé en Suisse en 2019 lorsqu’il n’y avait aucun cas de COVID, et il a visité la capitale du pays »

Ici, le terme « du pays » fait référence à « Suisse » qui a été mentionné plus tôt dans la phrase. Cependant un modèle RNN aurait de la peine à faire le lien entre ces deux termes étant donné qu’il traite chaque mot séquentiellement, menant à une perte de contexte dans des longues phrases. Les modèles Transformers, grâce à leurs mécanismes de self-attention, traitent les phrases entières en parallèle augmentant beaucoup la capacité à comprendre le contexte.

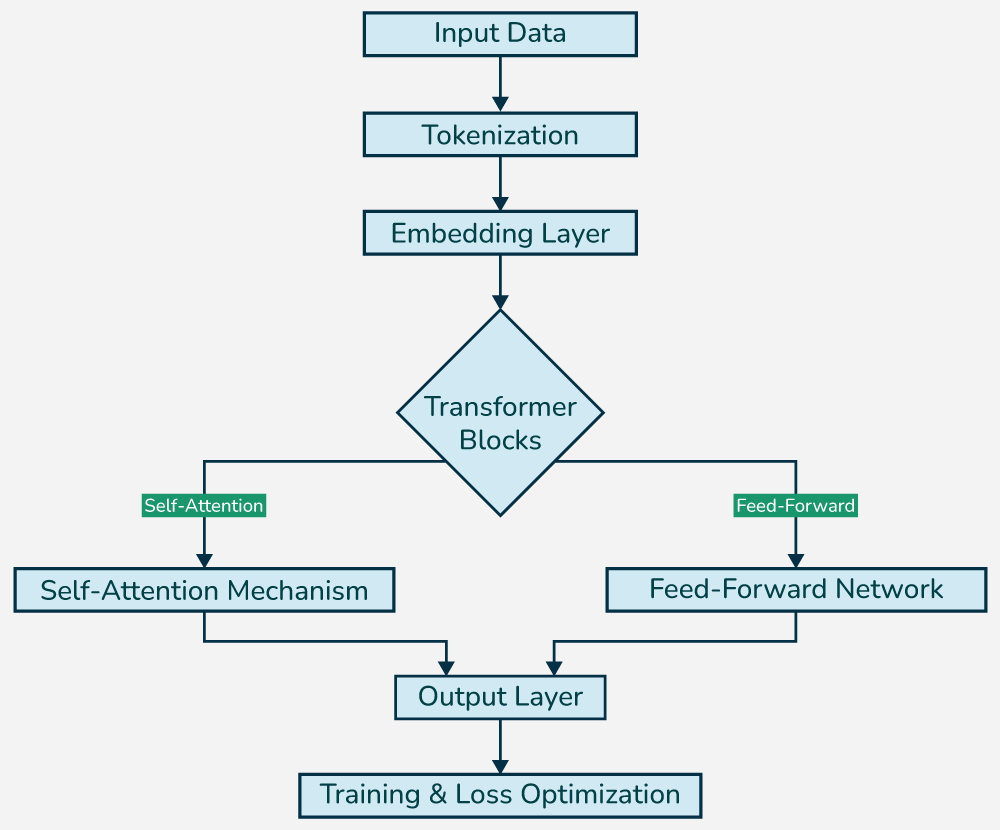


Figure 1 Architecture d'un modèle basé sur un Transformer

La Figure 1 (7) démontre l’architecture typique d’un modèle basé sur un Transformer. Les différents composants représentés par ce modèle sont :

1. **Input Data et Tokenization** : Les données d’entrée est découpé en unités nommées tokens. Ces tokens peuvent être des mots, des parties de mots, ou des caractères.
2. **Embedding Layer**: Chaque token est transformé en vecteur numérique, représentant sa définition sémantique. Vu que les transformers ne comprennent pas de base l’ordre des tokens, des Positional Embeddings sont ajoutés afin que le modèle ait les informations des positions des tokens dans une phrase.
3. **Mécanisme de Self-Attention** : Le modèle calcule des scores d’attention qui permet de déterminer combien d’attention chaque mot doit donner pour les autres mots de la séquence.
4. **Feedforward** : Suite au mécanisme de « Self-Attention », la sortie est passée à un feedforward neural network (une série de couches), appliqué indépendamment à chaque position.
5. **Blocks Transformers** : L’architecture inclut souvent plusieurs couches transformers empilées. Chaque block consiste de leurs propres mécanismes d’attention et de feedforward nerual network. Ceci permet au modèle d’apprendre des relations de données complexes.
6. **Output Layer** : La dernière couche est typiquement une fonction « softmax », qui converti les sorties en distribution de probabilités sur le vocabulaire, lui permettant de choisir le token de sortie le plus probable.

**Défis**

L’entrainement d’un LLM requiert un investissement conséquent, que cela soit d’un point de vu computationnel avec un entrainement pouvant parfois prendre plusieurs mois. En outre, l’obtention d’un dataset suffisamment large et diversifié pour l’entrainement est difficile, et soulève parfois des problèmes éthiques ou juridiques (données protégées par le droit d’auteur, par exemple). L’impact environnementale (consommation d’énergie, refroidissement) est également un sujet conséquent et à prendre en compte.

### Approches d’adaptation des LLMs

Malgré la capacité des LLMs à répondre au langage humain naturel non-structuré, la cohérence et la fiabilité des réponses dépendent entièrement des données sur lesquelles ses modèles se sont entraînés. C’est-à-dire que si les données d’entraînement fournis sont de nature erronée ou biaisée, les réponses du modèle le seront aussi. En outre du problème de fiabilité, il n’y a aucune garantie que le modèle est suffisamment entraîné sur un thème ou sujet donné. Donc s’il était amené à générer du contenu dessus, il y aurait des fortes chances « d’hallucination », où il invente des faussent informations.

Ceci est problématique dans, entre autres, le contexte de génération de contenu pédagogique. Certains cours peuvent être basés sur des sujets très spécifiques et/ou peu connus/documentés, diminuant les chances de pertinence dans le contenu généré par le modèle.

**Fine-tuning**

Ce problème peut être contourné par plusieurs façons. Une des façons est le fine-tuning, qui consiste à réentraîner un modèle déjà pré-entraîné sur un dataset spécifique, afin de l’adapter à une tâche ou un domaine spécifique. Cette méthode améliore considérablement la précision du modèle pour un usage donné, mais présente plusieurs inconvénients :

* Nécessite du temps et des ressources matérielles pour le réentraînement.
* Peu de flexibilité, car des réentrainements seraient nécessaires pour des changements de tâches spécifiques.

Cette approche est donc adaptée pour des cas dans lesquels les tâches spécifiques sont prédéfinies, et qu’une haute précision est requise. (8)

**Contextualisation par prompt (contextual prompting)**

Une autre approche est la contextualisation par prompt (contextual prompting), qui ne modifie pas les paramètres du modèle tel que le Fine-Tuning le fait, mais consiste à fournir le contexte avec la requête (prompt) d’entrée. Cette approche permet d’éviter le réentrainement d’un modèle et offre plus de flexibilité dans les tâches du modèle. Il y existe plusieurs techniques, telles que :

* **Prompt injection :** Le fait d’introduire des éléments spécifiques dans le prompt afin d’orienter la réponse.
* **Few-shot prompting :** Fournir quelques exemples dans le prompt pour aider le modèle à comprendre la tâche.
* **Contextual grounding :** Intégrer un extrait de texte (cours, documentation, article, etc.) dans le pormpt pour guider la génération.
* **Retrieval-Augmented Generation (RAG) :** Méthode plus avancée qui consiste à rechercher des informations pertinentes de façon automatisé dans des dispositifs de stockage (stockage local ou base de données vectorielle), et de les insérer avec le prompt comme contexte.

### Vectorisation et recherche de similarités

La vectorisation consiste à transformer un extrait de texte en une représentation vectorielle en nombre réels (embedding) dans un espace de grande dimension. Cette représentation permet de capturer la signification sémantique du texte de manière à pouvoir la manipuler numériquement dans des algorithmes mathématiques et statistiques.

Concrètement, chaque extrait de texte est projeté dans un espace vectoriel ℝⁿ, où des textes ayant un sens proche se retrouvent dans des régions proches du vecteur espace. Ces embeddings sont généralement produits par des modèles entraînés à encoder la signification linguistique.

La recherche de similarité entre deux textes se repose ensuite sur une mesure de proximité entre leurs deux vecteurs respectifs. La plus couramment utilisée est la similarité cosinus, définie comme :

où et sont les vecteurs correspondants aux textes. Cette approche mesure le cosinus de l’angle entre les deux vecteurs. Plus le cosinus est proche de 1, plus les deux vecteurs pointent dans la même direction, ce qui signifie que les deux extraits de texte sont sémantiquement similaires.

Cette approche est particulièrement intéressante dans le cadre du Retrieval-Augmented Generation (RAG), où l’objectif est d’améliorer la qualité et la pertinence des réponses générées par un LLM. Lorsqu’une requête est soumise, elle est vectorisée, puis comparée à un ensemble de documents pré-vectorisés à l’aide d’une base de données vectorielle. Les documents jugés pertinents sont ensuite utilisés comme contexte dans le prompt envoyé au modèle, améliorant donc la précision de la réponse générée.

### Prompt Engineering

Le prompt engineering représente la procédure de développer et optimiser des prompts afin d’utiliser le plus efficacement possible les LLMs pour générer du contenu désiré. Cela est crucial car les LLMs, malgré leur puissance dans l’interprétation de texte non-structuré, nécessitent tout-de-même des instructions claires, spécifiques, et non-ambiguës afin de produire un résultat spécifique voulu.

Il existe une grande multitude de techniques de prompt engineering, qui sont les suivantes (très brièvement expliquées) (9) (10) :

|  |  |  |
| --- | --- | --- |
| Technique | But | Cas d’utilisation |
| Zero-shot prompting | Réponse directe (façon classique). | Question et réponses simples. |
| Few-shot prompting | Guider le modèle avec des exemples. | Structure de réponse spécifique |
| Chain-of-thought prompting | Encourager le raisonnement étape par étape. | Mathématiques, raisonnement logique. |
| Instruction-based prompting | Focaliser le modèle sur des tâches spécifiques. | Réponses spécifiques à une tâche, génération de contenu. |
| Role-based prompting | Assigner un rôle au modèle afin guider le ton et contenu des réponses. | Réponses doivent simuler la perspective d’une entité spécifique. |
| Contextual prompting | Fournir de l’information en plus au modèle pour élargir son contexte. | Tâches spécifiques et détaillées. |
| Meta prompting | Guider le modèle pour la création/amélioration de prompts. | Automation du processus de création de prompt. |
| Self-consistency prompting | Sélectionner la réponse la plus consistante. | Tâche nécessitant réponses définitives. |
| Generated knowledge prompting | Génération intermédiaire de contexte avant de répondre. | Réponse finale dépend sur des connaissances pas fournis par l’utilisateur. |
| Dynamic prompt optimization | Ajuster les prompts en temps-réel. | Tâches itératives et workflows flexibles |
| Automatic prompt engineering | Programmer la génération/amélioration des prompts. | Amélioration de la pertinence/performance des réponses générées |
| Multi-prompt fusion | Fusionner plusieurs prompts indépendantes pour l’équilibre. | Amélioration de la précision de réponse. Chaque prompt reformule la demande. |
| Prompt chaining | Structurer des étapes indépendantes. (Output d’un prompt devient l’input d’un autre). | Raisonnements multi-étapes. |
| Directional stimulus prompting | Guider le ton et processus de raisonnement des réponses. | Réponse focalisant sur un certain style ou points clés. |
| Graph prompting | Structurer le prompt dans un format de graphe pour aider le modèle à interpréter les dépendances. | Analyse de données relationnelles. |

Tableau 1 Techniques de prompt engineering

## Travaux et technologies existantes

### Plateformes de génération de contenu pédagogiques

Il existe déjà une multitude de plateformes offrant la génération d’évaluations pédagogiques à l’aide des technologies LLMs. Une sélection de quelques-unes de ces plateformes a été faite sur la base de pertinence et de similarité du besoin à combler, afin d’y analyser les fonctionnalités offertes/manquantes.

* **Revisely :** Revisely est une plateforme permettant la génération automatisée de résumés, quizz, flashcards en se basant sur l’IA. Elle offre également des outils pour fournir des explications détaillées sur des contenu du cours, qui peuvent être uploadés sous une vaste gamme de formats que cela soit des documents, notes, ou vidéos.
* **Khanmingo :** Khanmingo est une plateforme offrant deux outils séparées, un outil gratuit dédié au prof ainsi qu’un outil payant dédié aux étudiants. Les deux servent d’assistant basé sur IA afin de guider leur public cible respectif. Il offre une vaste gamme de fonctionnalités qui sortent du cadre de génération de contenu pédagogique (création de plans de cours, chatbot assistant, analyse de classe, jeux éducatifs, etc.)..Malheureusement aucun essai concret a pu être effectué car la plateforme n’est pas disponible en Suisse.
* **Quizgecko :** Quizgecko est une plateforme, similaire aux autres, aider aux révisions en offrant la possibilité de générer des quiz à partir des notes, documents, et pdf. Offre également une gestion de devoirs.Questions d’écriture de code au format de question ouverte, types de questions variées, versions payantes relativement chères.
* **Pdftoquiz :** Outil simple et, comme son nom l’indique, propose une fonctionnalité spécifique qui est la génération de quiz à partir d’un pdf.

### Comparaison des plateformes existantes

Dans toutes les plateformes testées, leur utilisation n’est pas adaptée pour du contenu technique et spécifique à l’informatique, qui est l’axe principale de ce projet. Leur utilisation est principalement adaptée pour des sujets nécessitant plus de l’appris par cœur plutôt que des cours de logique.

Durant les tests effectués avec des fichiers de cours techniques, les questions générées avaient tendance à ne pas respecter les critères de qualité d’une évaluation (chapitre 2.1.1). Les questions manquaient souvent de clarté, les questions étaient très simples et nécessitaient très peu de réflexion, ou sinon les questions étaient tout simplement erronées.

En outre, aucune d’entre elles n’offrent des options de génération d’évaluations inspirées par d’autres évaluations, qui serait une fonctionnalité très utile pour à la fois les enseignants qui viseraient gagner du temps, et les étudiants qui chercheraient des exercices et types de questions similaires auxquelles ils ont l’habitude pour un cours donné.

Par ailleurs, elles n’offrent pas d’option de régénération de question spécifique d’un quiz, qui serait une fonctionnalité très utile pour raffiner un quiz/évaluation aux besoins des utilisateurs.

Le Tableau 2 montre une comparaison globale des différentes plateformes, listant des avantages et inconvénients qui ont pu être constatés.

|  |  |  |
| --- | --- | --- |
| Plateforme | Avantages | Inconvénients |
| Revisely | Offre possibilité de création de flashcards en plus des quiz. | Seulement un type de question peut être utilisé pour générer un quiz.  Seulement un fichier peut être utilisé comme source de contenu pour génération de quiz.  Ciblé exclusivement pour les étudiants.  Critères de qualité d’évaluation pédagogique pas respectés pour contenu technique/code. |
| Khanmingo | Vaste gamme de fonctionnalités sortant du contexte de génération de quiz/évaluation  Espaces séparés pour étudiants et enseignants. | Payant pour les étudiants.  Pas disponible en suisse. |
| Quizgecko | Possibilité de générer des flashcards pour aider aux révisions.  Beaucoup de choix pour l’input du quiz (Fichier, importation de question à partir d’un fichier, texte, URL, vidéo youtube, sujet aribtraire).  Quiz générés mélangent les différents types de questions.  Questions d’écriture de code supportés au format de question ouverte mais très basiques.  Choix de la langue du quiz | Version gratuite très limitée, une génération par jour.  Seulement un fichier peut être utilisé comme contexte de génération  Critères de qualité d’évaluation pédagogique pas respectés pour contenu technique/code, surtout au niveau de la clarté. |
| Pdftoquiz | Simple et droit au but.  Contexte peut être un fichier, un URL, ou du texte arbitraire.  Choix de la langue du quiz | Seulement questions QCM supportées dans la version gratuite.  Seulement un type de question pour tout le quiz, aucun mélange.  Critères de qualité d’évaluation pédagogique pas respectés pour contenu technique/code. |

Tableau 2 Résumé des plateformes existantes

### Modèles LLMs exitstants

Dans le cadre de ce projet, l’utilisation d’un LLM est un pilier incontournable pour permettre la génération de contenu. De ce fait, Il est nécessaire d’explorer les divers modèles, les potentiels APIs que leurs fournisseurs offrent, ainsi que leurs points forts et leurs points faibles afin de déterminer lequel de ces modèles serait le plus pertinent à utiliser pour la mise en œuvre de ce projet.

Deux catégories de modèles vont être explorées. Nous avons tout d’abord la catégorie des modèles qui sont uniquement disponibles en tant que LLMaaS (LLM as a Service), et la catégorie des modèles pouvant être hébergés localement. Les LLMaaS sont des modèles LLMs qui sont fourni par des infrastructures clouds et sont accessibles via des appels API (11).

**Gemini**

Gemini est une intelligence artificielle générative développée sur Google qui est offerte en tant que LLMaaS uniquement.

Les appels API font face à des limites de débits (rate limits). Ces limites sont basées sur plusieurs critères (12), qui sont le nombre de requêtes par minute (RPM), nombre de requêtes par jour (RPD), tokens par minute (TPM), et tokens par jour (TPD). Ces limites de débits sont liées au niveau d’utilisation des APIs, où le développeur peu changer le niveau d’utilisation du projet par rapport à la quantité d’argent dépensé. Les niveaux de projets sont disponibles en quatre catégories :

* Gratuit : Pour tous les utilisateurs dans les pays éligibles
* Niveau 1 : Nécessite un compte de facturation associé au projet
* Niveau 2 : Nécessite au minimum 250$ de dépenses et au moins 30 jours écoulés depuis le paiement
* Niveau 3 : Dépenses totales de 1000$ et également au moins 30 jours écoulés depuis le paiement

Les modèles explorés sont les suivants (12) (13) :

* **Gemini 2.5 Pro**: Le modèle de réflexion le plus puissant de Google, offrant une précision de réponse maximale. Optimisé surtout pour de la résolution de problèmes complexes, compréhension d’input multimodale complexe, et de la programmation avancée. Pas disponible pour un niveau de projet gratuit.
* **Gemini 2.5 Flash** : Meilleur modèle en termes de rapport performances/prix. Optimisé pour du raisonnement adaptatif et la rentabilité, et offre également de la compréhension multimodale mais moins avancé que le modèle Pro. Disponible pour niveau de projet gratuit mais avec des limites de débit de 10 RPM, 250'000 TPM, et 500 RPD.
* **Gemini 2.0 Flash** : Optimisé pour la rapidité, le raisonnement, et permet également les inputs multimodaux. Disponible pour un niveau de projet gratuit avec les limites de débits étant 15 RPM, 1'000'000 TPM, et 1'500 RPD

Les avantages des modèles Gemini est en premier lieu l’API fourni, qui est riche en fonctionnalités et une documentation facile à comprendre. Dans le cadre ce projet, les avantages sont la capabilité d’utilisation d’une grande quantité token comme input, idéale pour l’envoi de contexte. Un autre avantage est également la possibilité de structurer les outputs des modèles en JSON avec un système de schéma intégré dans l’API de Google fourni. L’API fourni également la compréhension de documents PDF intégrés, ayant la possibilité d’analyser les diagrammes et les tables dans les documents puis y extraire les informations, quelque chose qui serait idéale dans le cadre d’analyse de support de cours.

Les inconvénients sont surtout que ces modèles sont purement cloud-based, donc il n’y a aucun moyen d’héberger ces modèles localement. Ceci peut engendrer des problèmes où les modèles sont surchargés et ne peuvent temporairement pas être utilisés même si les limites de débits n’ont pas été dépassées. Les limites de débits sont un autre inconvénient. Les limites pour un niveau de projet gratuit étant limités doivent être adaptés par rapport aux taux d’utilisation de l’application. Il serait nécessaire de passer sur des niveaux de projets payants afin d’augmenter ces limites, et les niveaux payants vont engendrer des frais sur chaque million de tokens utilisés.

**Mistral**

Mistral AI, contrairement à Gemini, offre ses modèles à la fois en tant que LLMaaS, ou laisse la possibilité de les héberger localement. L’utilisation de leurs modèles en tant que LLMaaS engendre des couts d’utilisation, contrairement à leur utilisation en local, qui est gratuit. Cependant pas tous les modèles sont utilisables localement, et certains de ces modèles sont licenciés sous la licence « Mistral Research License », voulant dire que leurs utilisations se limite à des fins non commerciales ou de recherche, ce qui ne poserait aucun problème dans le cadre de ce projet. D’autres sont des modèles ouverts (sous licence Apache2), donc utilisation gratuite complète. Quelques-uns de leurs modèles disponibles explorés sont (14) :

* **Mistral Large** : Modèle le plus avancé de Mistral, optimisé en programmation, raisonnement avancé, et tokens d’input de 128k. Utilise 123 milliards de paramètres. Modèle lourd (73GB) et nécessiterait une machine puissante pour l’héberger. (Licencié sous Mistral Research License).
* **Mixtral** : Modèle de MoE (Mixture of Experts, donc regroupe plusieurs sous-modèles et en active qu’une partie par requête). Possible en modèles 8x7b (8 sous-modèles de 7 milliards de paramètres chacun) et 8x22b. Courant en plusieurs langues, bonnes capacités en maths et en programmation. Capactié de 64k tokens d’input. (Licencié sous Apache2).
* **Mistral NeMo** : Modèle relativement légé construit en collaboration avec NVIDIA sur 12b paramètres. Offre 128k tokens d’input. Capacités excellentes de raisonnement, programmation, multilinguisme, et connaissances générales par rapport à sa taille (7.1 GB). (Licencié sous Apache2).

# Planification initiale

### Liste des tâches

La liste des tâches permettant la mise en œuvre de ce projet a été définit (Tableau 3), et des priorités leurs ont été attribuées (1 étant le plus prioritaire). Elles ont également été séparées en catégories afin de permettre leur regroupement pour des points de vue plus globales.

|  |  |  |  |
| --- | --- | --- | --- |
| Tâche | Priorité | Durée estimée | Catégorie |
| Mise en place de la DB relationnelle | 1 | 4 h | Base de données |
| Mise en place de la base de données vectorielle | 1 | 8 h | Base de données |
| Implémentation de l’authentification | 2 | 8 h | Sécurité |
| Implémentation de l’upload des fichiers | 1 | 8 h | Gestion des fichiers |
| Stockage et indexation des fichiers dans la DB vectorielle | 1 | 8 h | Gestion des fichiers |
| Implémentation de la suppression/modification des fichiers | 3 | 14 h | Gestion des fichiers |
| Implémentation de génération d’évaluation à partir d’un contenu | 1 | 14 h | Génération d’évaluations |
| Implémentation de génération d’évaluation inspiré d’un fichier d’évaluation fourni | 2 | 24 h | Génération d’évaluations |
| Implémentation des options de génération | 2 | 14 h | Génération d’évaluations |
| Implémentation de régénération de question à partir de feedback utilisateur | 2 | 14 h | Génération d’évaluations |
| Implémentation de système d’ajout de question avec un prompt utilisateur | 3 | 10 h | Gestion d’évaluations |
| Implémentation du tableau de bord utilisateur | 2 | 14 h | Gestion d’évaluations |
| Implémentation de la modification/suppression de questions | 3 | 8 h | Gestion d’évaluations |
| Implémentation de mode « test » d’évaluation | 3 | 24 h | Gestions d’évaluations |
| Implémentation de la correction automatique des réponses utilisateurs | 3 | 24 h | Gestion d’évaluations |
| Implémentation du mode affichage (questions + réponses) | 1 | 6 h | Gestion d’évaluations |
| Implémentation sauvegarde des évaluations en fichier local | 3 | 12 h | Gestion d’évaluations |
| Implémentation partage des évaluations entre utilisateurs | 3 | 12 h | Gestion d’évaluations |
| Création de la maquette de l'application | 1 | 6 h | Conception |
| Mise ne place des tests unitaires | 2 | 16 h | Tests |
| Mise ne place des tests E2E | 2 | 20 h | Tests |
| Amélioration expérience utilisateur | 3 | 10 h | UX |
| Déploiement de la DB | 2 | 12 h | DevOps |
| Mise en place de CI/CD | 3 | 10 h | DevOps |
| Hébergement du LLM choisi sur serveur local | 3 | 10 h | DevOps |
| Documentation du rapport | 1 | 120 h | Documentation |

Tableau 3 Liste des tâches

Figure 2 Total heures planifiées par catégorie de tâche

La Figure 2 montre la répartition des heures pour chaque catégorie de tâche. Sans surprise, la documentation est planifiée à prendre la majorité du temps, étant donné que c’est une tâche qui dure tout le long du projet en parallèle de toutes les autres tâches à implémenter. Les tâches de catégorie gestion d’évaluations se trouve en deuxième place après la documentation. Beaucoup de marge a été attribuée à cette catégorie étant donné que cela implique le « mode test » des évaluations, et cela nécessiterait la mise en place de la correction automatique des questions, où il y a des fortes chances de faire beaucoup de « trial and error » afin de se retrouver avec une correction efficace.

# Conception

## Architecture

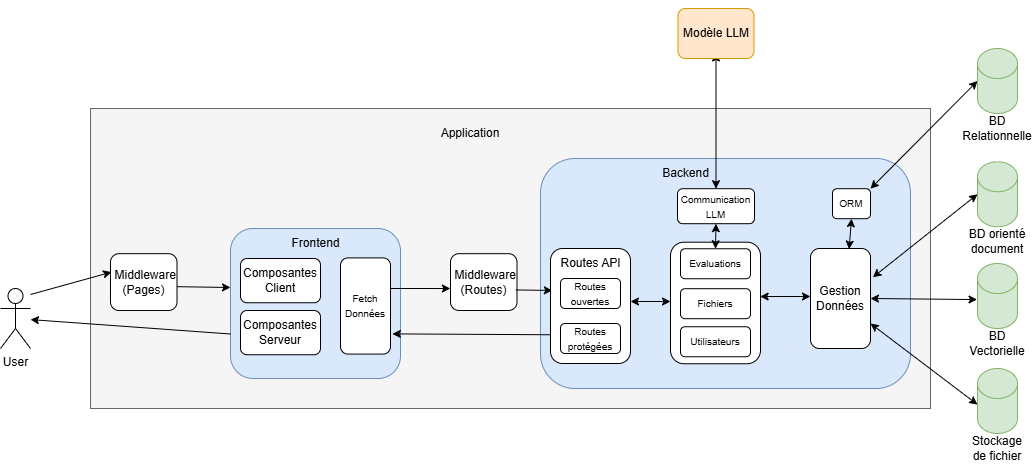


Figure 3 Schéma d’architecture du projet

La Figure 3 démontre le diagramme d’architecture abstrait du projet, représentant donc le regroupement et l’organisation des différents éléments principaux composants le projet, ainsi que leurs interactions (ne démontre pas forcément le parcours spécifique d’une requête utilisateur.

Le frontend (interface utilisateur) sert de point d’entrée de l’application. Elle permet aux utilisateurs (étudiants ou enseignants) d’interagir avec les diverses fonctionnalités proposées par l’application. Il est composé des *Composantes Client* (les composantes devant être rendu côté client), *Composantes Serveur* (composantes devant être rendu côté serveur), et d’un module s’occupant la communication avec les routes API du backend. L’accès à différents composants sera protégé par un middleware, permettant de garantir l’authentification et l’autorisation de l’utilisateur

Le backend est responsable de gérer toute la logique métier. Il traite les requêtes utilisateur venant du frontend et gère les accès aux données et les appels aux services externes. Il est composé tout d’abord des routes API servant d’endpoint pour les appels venant du frontend. Certaines routes seront ouvertes (nécessitant aucune connexion utilisateur) et des routes protégées (nécessitant une authentification utilisateur) pour des opérations spécifiques, qui sera géré par un middleware. Ces routes perme de manipuler les différents éléments tels que les modules d’évaluation, fichier, et utilisateur. Le module d’évaluation fera appel à un module de communication LLM, permettant de gérer toutes les interactions avec le modèle LLM. Sinon les trois éléments principaux communiquent avec un module de gestion des données, gérant toutes les interactions avec les dispositifs de stockage. Un ORM sera utilisé afin de faciliter la gestion, représentation et le traitement de la base de données relationnelle.

Le projet nécessite plusieurs composants de stockage spécialisés, chacun dédié à un type de données bien précis :

* Base de données relationnelle : Utilisé pour stocker les entités structurées telles que les utilisateurs, les métadonnées des évaluations et les métadonnées des fichiers.
* Base de données orientée document : Utilisé pour stocker les évaluations, car elles ne suivent pas de structure rigide. En effet, elles peuvent beaucoup varier que ça soit dans la quantité ou types de questions. Cela offre une grande flexibilité dans leur gestion.
* Base de données vectorielle : Utilisé pour stocker les embeddings générés des fichiers uploadés par les utilisateurs, permettant ainsi la recherche de similarité sémantique pour le Retrieval-Augmented Generation (RAG) (chapitre 2.1.4).
* Dispositif de stockage de fichiers : Utilisé pour stocker la version brute des fichiers uploadés (PDF, DOCX, etc), en parallèle de leurs embeddings stockés dans la DB vectorielle. Le contenu textuel de ces fichiers sont utilisés pour le RAG.

Ces dispositifs sont représentés séparément dans la Figure 3, mais cela ne veut pas forcément dire qu’ils sont hébergés sur des machines différentes.

## Choix technologiques

Ce chapitre vise à expliquer les choix technologiques prises pour les différentes composantes de l’architecture décrite dans le chapitre précédent (4.1). Chaque choix se repose sur des critères de pertinence technique, de maîtrise des outils, ainsi que la portée spécifique du projet.

### Application

Le projet étant une application web interactive, le framework Next.js a été utilisé. Il centralise le développement frontend et backend dans un environnement unifié, basé sur React. La version moderne de Next.js, appelée App Router, est utilisée ici. Elle se base sur les fonctionnalités récentes de React telles que les composants Server/Client, le streaming, les layouts imbriqués, et le support natif de l’asynchronisme au niveau des composants. Les performances peuvent également être améliorées grâce aux différents types de rendering disponibles (SSR, SSG, IST ou CSR) selon les besoins. Contrairement à l’ancien système Pages Router, l’App Router permet également le routing des pages et des routes API par rapport à la structure du système de fichiers. Le choix de ce framework se repose également sur mes connaissances de celui-ci, ce qui permet de réduire la courbe d’apprentissage et de garantir une meilleure productivité.

### Hébergement

Le public cible étant principalement les étudiants et enseignants en informatique à la HEIG-VD, une stratégie d’hébergement local via conteneurs est adoptée. L’application web ainsi que la base de données seront containerisées avec Docker, puis exécutées localement par les utilisateurs. Cette approche offre plusieurs avantages par rapport à un hébergement public, étant principalement :

* Aucune nécessité d’héberger les différentes composantes sur des services payants pour une plateforme en production (Serveur web, base de données, stockage de fichiers).
* Evite la nécessité de la mise en place d’un système d’authentification, d’autorisation, et toutes les notions de sécurité qui vont avec, donc le développement peut être centré sur les éléments clés de l’application (génération d’évaluation), augmentant potentiellement sa pertinence.

Bien évidemment cela présente également quelques inconvénients, tels que :

* Le public cible est forcément restreint sur des utilisateurs ayant des connaissances sur l’utilisation de conteneurs.
* Toute fonctionnalité d’interaction entre les utilisateurs sur l’application, qui étaient initialement prévues en tant que plus-value, ne seront plus pertinents à mettre en place.
* Les utilisateurs nécessiteront quand même une connexion internet afin de faire appel au modèle LLM, qui serait trop lourd de lancer localement sur la machine locale de l’utilisateur.
* Toutes les données sont stockées localement, donc en cas de perte de données ou panne de machine, les utilisateurs n’ont aucun moyen de récupérer leurs données à distance.

### Stockage des fichiers

Au niveau du stockage des fichiers bruts envoyés par les utilisateurs, il y a plusieurs options qui se présentent, chacune ayant ses propres avantages et inconvénients selon le contexte.

**Stockage en binaire dans une base de données**

* Cette solution offre des avantages dans la simplicité de la gestion car toutes les données sont centralisées dans une base de données.
* Cependant, cette solution impacte la scalabilité, où le stockage d’une grande quantité de fichiers volumineux va rapidement alourdir la base données, diminuant ainsi sa performance globale.

**Stockage dans le système de fichiers local**

* Les avantages de cette solution sont la facilité d’implémentation car elle requiert une mise en place minimale, ainsi que la performance car l’utilisation du système de fichier local reste plus rapide que l’utilisation d’une plateforme remote pour l’accès aux fichiers.
* Les inconvénients sont également l’impact sur la scalabilité, car cette approche est limitée par la capacité de stockage du serveur local.

**Stockage cloud**

* Les avantages de stocker les fichiers dans le cloud (tel que Amazon S3 ou Google Cloud Storage) sont tout d’abord la scalabilité, car les services cloud vont dynamiquement scale up ou scale down les dispositifs de stockage par rapport à la demande. Le deuxième avantage est la disponibilité, car les services cloud sont de nature distribuées, offrant donc la redondance et la tolérance aux pannes.
* Les inconvénients sont les coûts d’utilisation, la mise en place qui peut être plus complexe que les méthodes décrites précédemment, ainsi que la performance car l’accès aux fichiers stockés dans le cloud peut potentiellement prendre plus de temps qu’un accès locale ou en base de données.

Vu le choix de déployer la plateforme localement comme décrit dans la section 4.2.3, la solution retenue est le **stockage sur le système de fichier local du serveur.** Cette solution est la plus adapté à ce cas-là, car l’inconvénient de la scalabilité est largement diminuée. En effet, chaque instance locale ne doit que stocker les fichiers d’un seul utilisateur. La performance au niveau des accès aux fichiers reste également très élevée, comparée à la solution de stocker les fichiers en binaire dans la base de données.

Si la plateforme avait à être déployée publiquement, la solution cloud aurait été la plus pertinente pour stocker les fichiers d’une grande quantité d’utilisateurs vu ses offres de scalabilité et de disponibilité.

### Base de données

Le système de gestion de base de données choisi est PostgreSQL, surtout pour ses fonctionnalités avancées permettant la gestion des différents types de données nécessaires pour ce projet. Il permet principalement le stockage des données relationnelles pour le stockage des différentes entités de l’application (métadonnées fichiers, métadonnées évaluations, etc.). PostgreSQL supporte également le type JSONB, offrant un type de stockage semi-structuré tel que les bases de données NoSql basées documents le permettent. Ceci est idéale pour le stockage des évaluations, comme décrit dans le chapitre 4.1. Finalement, elle offre l’extension pgvector, permettant de stocker les embeddings des fichiers uploadés par les utilisateurs.

Cette combinaison de fonctionnalités permet de centraliser les différents types de données dans un système unique, tout en gardant la flexibilité nécessaire au projet.

### ORM

L’ORM retenu pour ce projet est Prisma, principalement en raison de ses avantages en termes de productivité, sécurité et de performance dans un environnement TypeScript.

Prisma génère automatiquement un client TypeScript fortement typé à partir du schéma de la base de données. Cela offre une expérience de développement plus fiable, avec détection d’erreurs à la compilation et de l’auto-complétion.

Prisma fournit également une syntaxe intuitive et déclarative pour effectuer des opérations sur la base de données (CRUD, filtres, relations, etc.), réduisant donc la complexité du code backend tout en conservant une bonne lisibilité.

Un système de migration de base de données est également inclus dans Prisma, ce qui permet de versionner et synchroniser l’évolution du schéma avec le code source.

Un outil Prisma Studio est également disponible avec Prisma, permettant de visualiser facilement la base de données lors du développement.

### Modèle LLM

Le modèle LLM à utiliser n’a pas encore été complétement déterminé. Malgré l’implémentation actuelle (chapitre 5) faisant déjà utilisation du modèle Gemini 2.0-flash, il serait potentiellement préférable de changer de modèle (surtout de fournisseur) permettant l’hébergement local des modèles. Ceci permettrait dans le cas de la mise en production de l’application d’éviter des limites de débits, des frais d’utilisations, ou des indisponibilités dues aux surcharges des modèles.

Cependant si le modèle devait être hébergé localement, cela nécessiterait la disponibilité d’un serveur qui soit assez puissant pour l’héberger, et bien sûr cela nécessiterait également de la maintenance.

Idéalement le modèle devrait pouvoir être « fine-tuné » au niveau de la génération des évaluations, permettant d’améliorer leur pertinence afin de satisfaire au mieux les critères de qualité d’une évaluation pédagogique (chapitre 2.1.1). Ceci serait un cas idéal de fine-tuning vu la spécificité des critères de qualité (plus de détail dans le chapitre 2.1.3), mais nécessiterait un vaste dataset d’évaluations respectant ces critères, qui pourrait potentiellement être difficile à trouver. Ceci pourrait être combiné avec la contextualisation via prompt pour les contenus des cours spécifiques, qui serait plus compliqué à mettre en place avec le fine-tuning étant donné que les sujets et thèmes des cours peut être vaste doit rester flexible.

## Modélisation des données

Ce projet nécessite plusieurs modélisations de données, qui sont donc en premier lieu le modèle conceptuel pour la base de données relationnelle, ainsi que les schémas JSON afin de définir le format des évaluations, qui seront donc stockée en format JSONB dans PostgreSQL comme décrit dans le chapitre 4.2.4.

### Modèle conceptuel

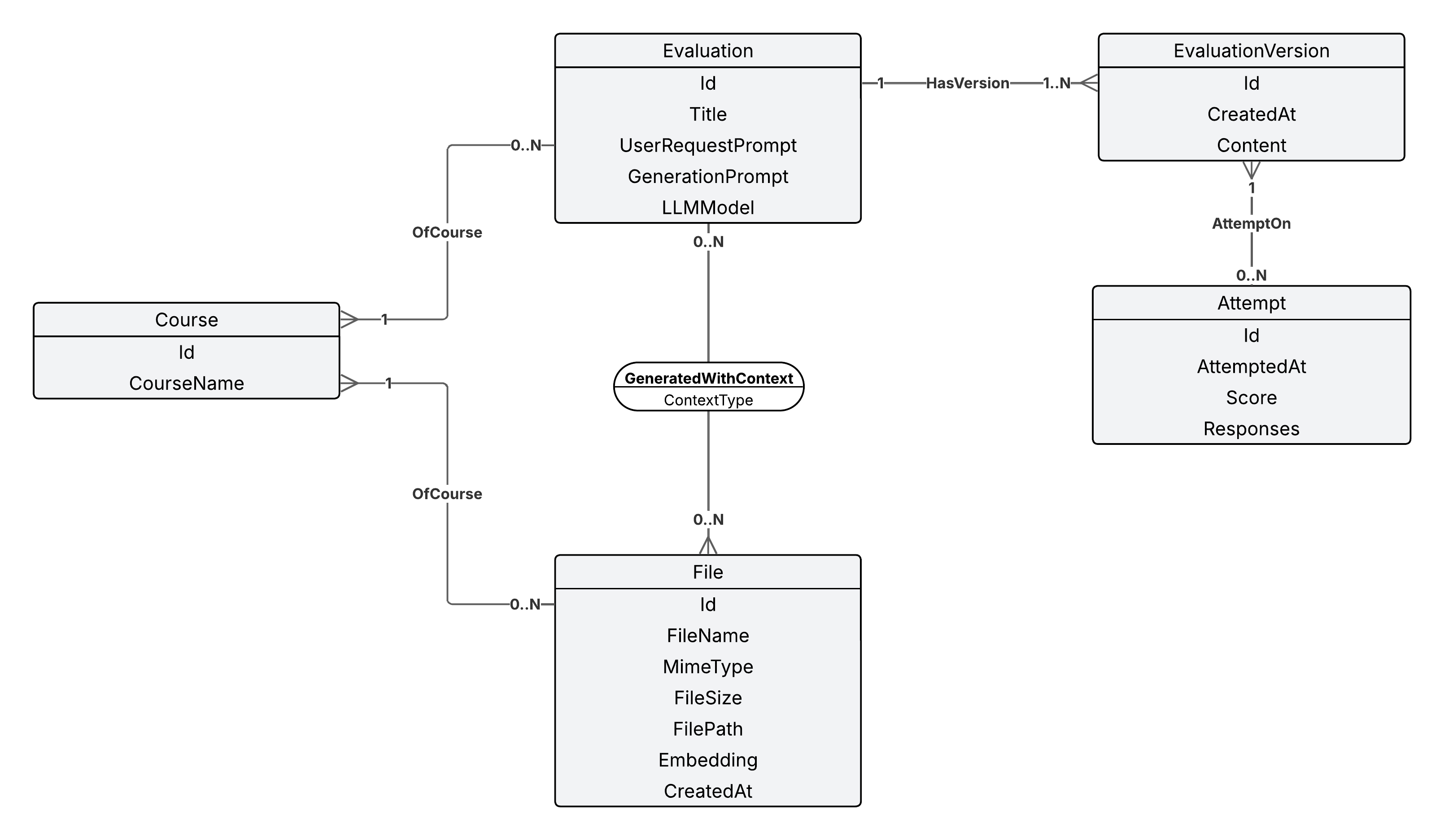


Figure 4 Modèle conceptuel

La **Erreur ! Source du renvoi introuvable.** démontre le modèle conceptuel de la base de données relationnel. En premier lieu il contient l’entité « Evaluation », qui va servir de contenir toutes les métadonnées d’une évaluation créée. Ces métadonnées sont le titre de l’évaluation, le prompt de requête utilisateur, le prompt utilisé par l’application pour effecteur la génération, ainsi que le modèle LLM utilisé pour générer l’évaluation. Ces trois derniers champs décrits servent plus à un avoir un suivi des évaluations avec leurs paramètres de génération et ne seront pas forcément utilisée par l’application.

Une évaluation est créée à partir de fichiers de contexte, qui est représenté par l’entité « File ». Une évaluation peut être généré avec deux types de contexte différents :

* Contexte de contenu : Indique que le fichier est utilisé comme contexte de contenu, donc la génération de l’évaluation va se baser sur le contenu du fichier. (Pour les fichiers contenant les contenus du cours)
* Contexte d’inspiration : Indique que le fichier est utilisé pour inspirer le type, le format, et la forme des questions à poser. (Pour les fichiers étant des travaux écrits/exercices).

Les métadonnées des fichiers sont stockées dans l’entité « File », étant donc le nom du fichier, le type MIME du fichier, ainsi que la date de création. L’entité contient également un champ « Embedding » afin de stocker les vecteurs des fichiers. Elle contient également un champ « FilePath » permettant de localiser le fichier brut stocké dans le système de fichiers du serveur local. Le champ « FileSize » stocke la taille du fichier.

L’historisation des évaluations est également possible grâce à l’entité « EvaluationVersions ». Cette entité contiendrait le contenu des évaluations, ainsi que la date de création. Chaque évaluation créée aura la version initiale créée également. Ceci est utile pour que les utilisateurs puissent avoir un historique des modifications effectuées sur leurs évaluations, et leur permettant donc de retourner sur une version antérieure si besoin.

Les évaluations et les fichiers font toutes les deux parties d’un cours, permettant de les catégoriser. Les cours sont représentés par l’entité « Course », qui stocke tout simplement le nom du cours. Ces cours peuvent être ajoutés/supprimés par les utilisateurs.

L’entité « Attempt » représente les évaluations effectuées en mode « test », permettant de garder un historique de tous les essais effectués, le timestamp du moment ou il a été fait, ainsi que les réponses fournies par l’utilisateur qui seraient aussi stockées au format JSON.

### Schéma JSON des évaluations

Etant donné que les évaluations sont stockées au format JSON, il est nécessaire de créer leurs schémas JSON que les modèles LLMs doivent respecter lors de leur génération. Ceci est crucial afin de garantir une cohérence dans les évaluations. Le schéma JSON démontré dans la ci-dessous adhère au standard « JSON Schema Draft 2020-12 » (15).

Listing 1 Schéma JSON d’une évaluation au format « JSON Schema Draft 2020-12 »

{

"$schema": "https://json-schema.org/draft/2020-12/schema",

"type": "object",

"required": ["content"],

"properties": {

"content": {

"type": "array",

"description": "Array of questions for the quiz.",

"items": {

"type": "object",

"required": ["number", "questionText", "questionType", "correctAnswer", "explanation"],

"properties": {

"number": {

"type": "string",

"description": "The question number."

},

"questionText": {

"type": "string",

"description": "The text of the question."

},

"questionType": {

"type": "string",

"description": "The type of the question.",

"enum": ["mcq", "open", "codeComprehension", "codeWriting"]

},

"options": {

"type": "array",

"description": "Array of answer options for multiple choice questions.",

"items": {

"type": "string"

},

"minItems": 0

},

"correctAnswer": {

"type": "string",

"description": "The correct answer for the question."

},

"explanation": {

"type": "string",

"description": "An explanation for the correct answer."

}

}

}

}

}

}

Le schéma JSON montré dans le Listing 1 permet de structurer et valider les évaluations générées par les modèles LLM. L’objet JSON généré doit donc contenir un tableau « content » qui va contenir toutes les questions. A noter que le schéma pourrait être dynamiquement adapté dépendant des choix utilisateurs (par exemple définir dynamiquement « minItems » pour le tableau « content » par rapport à une indication utilisateur qu’il souhaite un minimum de x questions).

Les questions ont comme propriétés :

* « number » : Numéro permettant d’identifier une question dans une évaluation, ainsi que de leur garantir un ordre. (Propriété obligatoire)
* « questionText » : Le texte de la question en elle-même. (Propriété obligatoire)
* « questionType » : Le type de la question, qui peut être soit un QCM, une question ouverte, une question de compréhension de code, une question d’écriture de code. (Propriété obligatoire)
* « options » : Tableau stockant toutes les réponses possibles dans le cas d’une question QCM. Cette propriété n’est donc pas indiquée comme étant obligatoire, et son keyword « minItems » est définie à 0.
* « correctAnswer » : Réponse correcte de la question. (Propriété obligatoire)
* « explanation » : Une explication à la réponse correcte. (Propriété obligatoire)

## Maquette initiale

Afin de faciliter le développement et de visualiser les fonctionnalités à implémenter, une maquette initiale de l’application a été mis en place via l’outil Figma. A noter que le nom de la plateforme indiqué sur la maquette est « Evagen », mais cela n’est pas définitif.

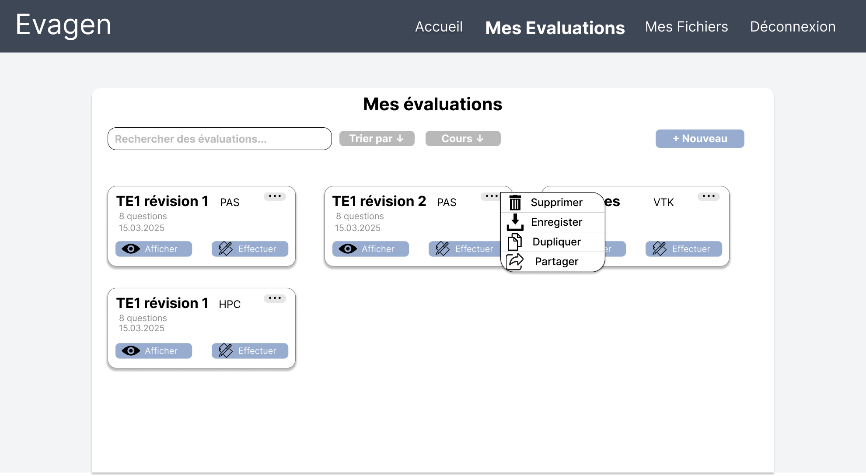


Figure 5 Dashboard évaluations

La Figure 5 démontre la page de dashboard des évaluations. Cette page permet à l’utilisateur de manipuler ses évaluations. Ses possibilités sont :

* Rechercher ses évaluations via une bar de recherche.
* Trier ses évaluations à partir de divers critères (Date de création, date de modification, nombre de questions, etc).
* Afficher toutes les évaluations d’un cours en particulier
* Créer une nouvelle évaluation
* Diverses options sur une évaluation telles que la supprimer, l’enregistrer sur la machine locale (en format JSON ou en format PDF), dupliquer une évaluation, ou là partager.
* Passer à un affichage complet d’une évaluation (plus d’information dans la description de la Figure 10)
* Effectuer une évaluation (plus d’information dans la description de la Figure 11)

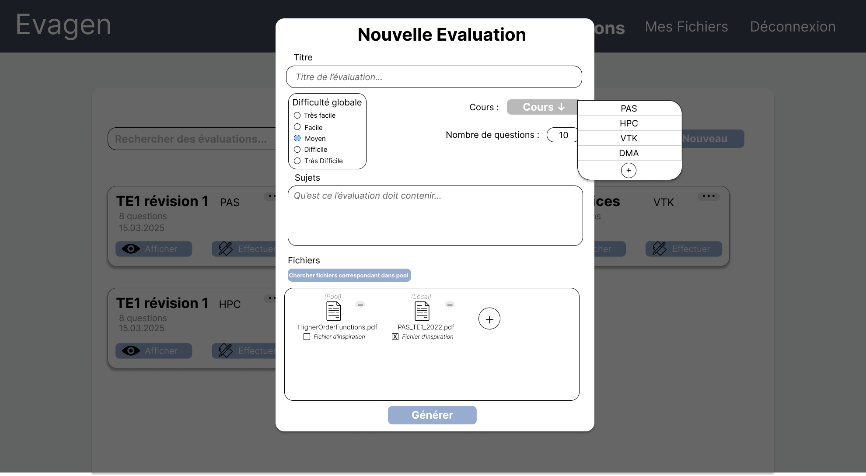


Figure 6 Nouvelle évaluation

La Figure 6 démontre le formulaire de création d’une évaluation. L’utilisateur y défini les paramètres tels que la difficulté globale des questions, le cours auquel l’évaluation appartient, le nombre de questions et les sujets sur lesquels l’évaluation va porter. Au niveau des fichiers de contexte, l’utilisateur peut automatiquement rechercher les fichiers de contexte correspondant dans son pool de fichier, ou sinon les ajouter manuellement, qu’ils proviennent du pool de fichier ou de sa machine locale. L’utilisateur a la possibilité de définir lesquels de ces fichiers doivent être considérés comme fichier d’inspiration.

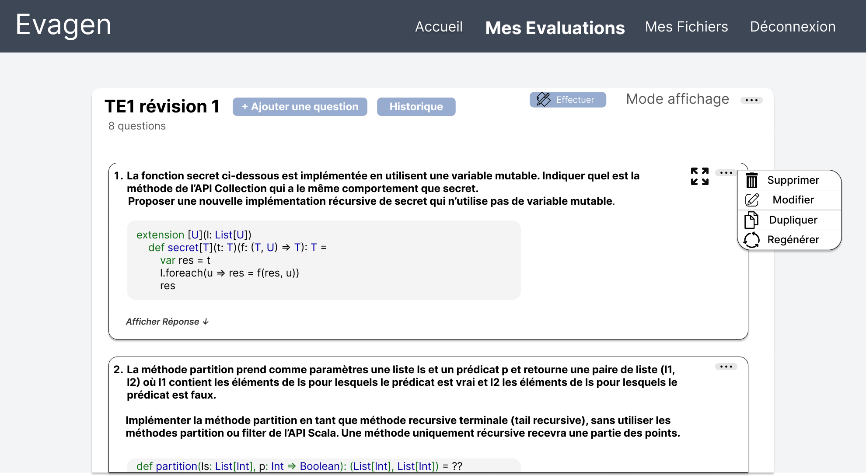


Figure 7 Mode affichage

La Figure 7 démontre un exemple du mode « affichage » d’une évaluation. Cette page permet de visualiser une évaluation spécifique entièrement, ainsi que y effectuer des modifications. Ici l’utilisateur a la possibilité d’ajouter une nouvelle question, visualiser l’historique des versions de l’évaluation et retourner à une version antérieure si besoin. Chaque question est affichée et l’utilisateur à la possibilité d’afficher leurs réponses. L’utilisateur a également la possibilité de manipuler des questions spécifiques, tel que les supprimer, les modifier manuellement (modification des textes), dupliquer une question, et les regénérer avec un prompt spécifique permettant d’indiquer au modèle LLM les modifications à effectuer.

Une fois tous les changements effectués sur l’évaluation, l’utilisateur à la possibilité d’enregistrer les changements ou de les annuler. L’enregistrement engendrera la création d’une nouvelle version de l’évaluation qui fera office de version actuelle de l’évaluation.

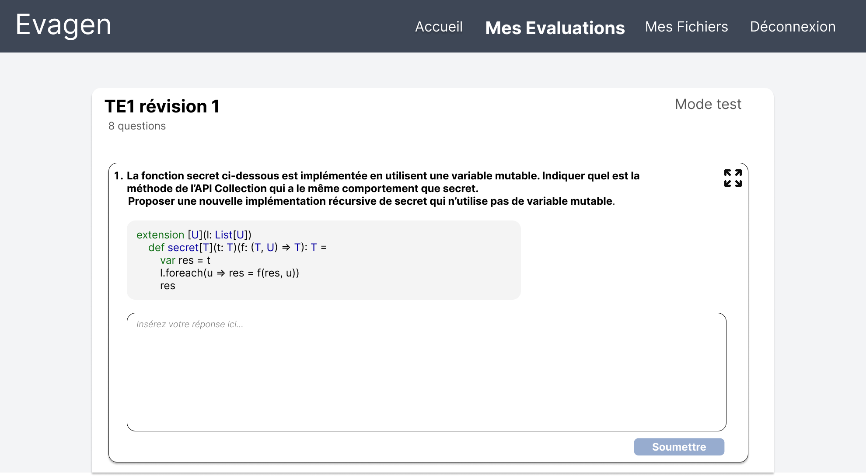


Figure 8 Mode test

La Figure 8 démontre le mode « test » d’une évaluation. Ici, l’utilisateur ne peut pas modifier l’évaluation ou avoir accès aux réponses. Dans ce mode il répond aux questions et soumets les questions dès qu’elles été répondues. Dès que l’utilisateur a répondu à toutes les questions, il peut soumettre l’évaluation en entier afin d’y récupérer un retour.

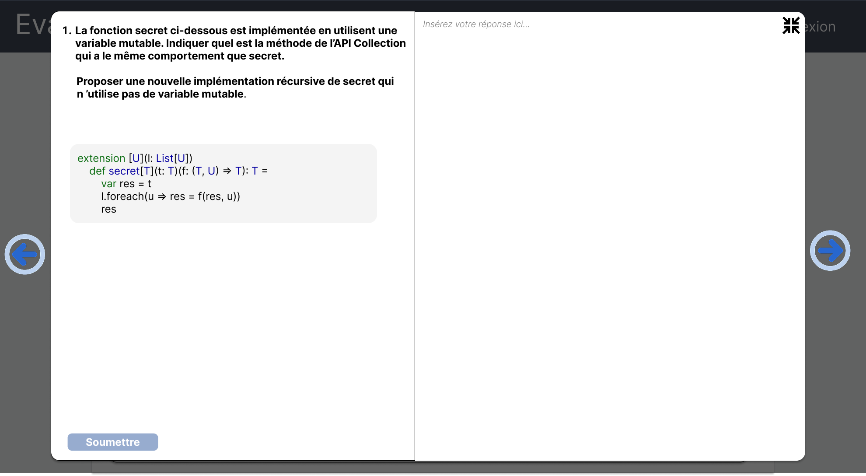


Figure 9 Mode test agrandi

La Figure 9 démontre la version « agrandie » d’une question, lui donnant plus de place pour visualiser sa réponse si nécessaire. Il a la possibilité de changer de question via les flèches qui se trouvent aux deux côtés, sans retourner dans la version « minimisée ».

L’utilisateur a également possibilité d’entrer dans cette version agrandie dans le mode « affichage » d’une évaluation. Dans ce cas-là, la partie de droite contiendrait la réponse de la question et pas un input comme dans le mode « test ».

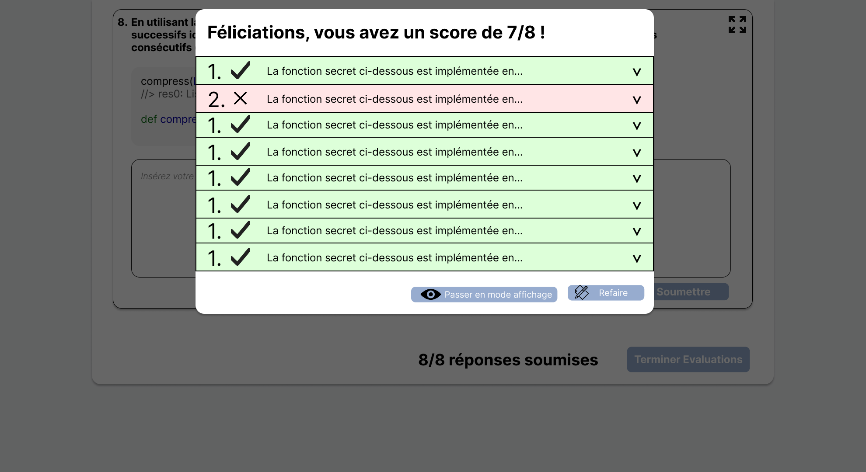


Figure 10 Résultats

La Figure 10 démontre l’affichage du score utilisateur après avoir effectué une évaluation. Il a la possibilité de regarder plus en détail les questions, ou sinon refaire l’évaluation en mode test ou sinon passer au mode « affichage ». Le score représente simplement le quantité de questions répondues correctement.

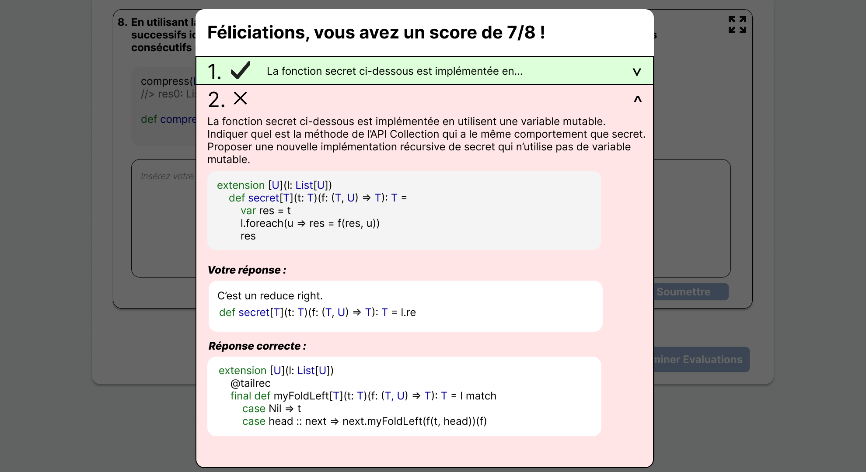


Figure 11 Résultats détails

La Figure 11 montre un exemple de visualisation de question plus détaillée. La question est répétée et affiche la réponse fournie par l’utilisateur, ainsi que la réponse correcte attendue.

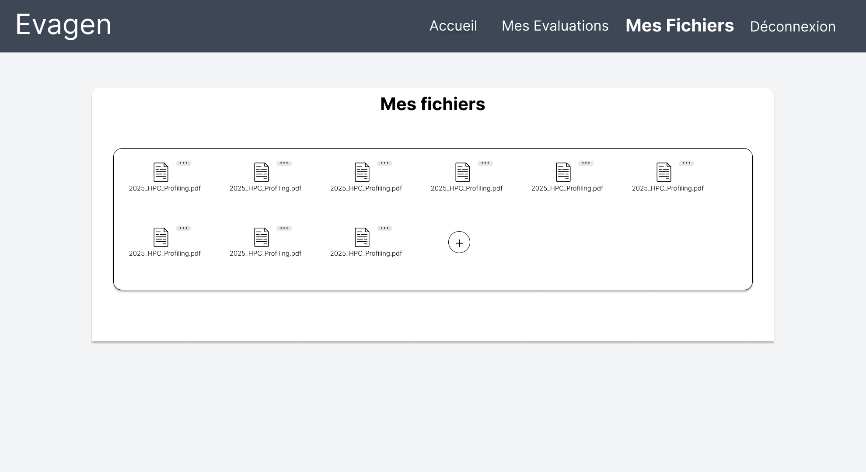


Figure 12 Dashboard fichiers

La Figure 12 démontre la page de dashboard des fichiers. Ici, l’utilisateur a la possibilité de manipuler son pool de fichier. Il peut y ajouter des nouveaux fichiers, ou modifier/supprimer ses fichiers déjà présents.

# Implémentation actuelle

Une première itération d’implémentation basique a déjà été mis en place.

### Authentification

L’implémentation actuelle comprend un système d’authentification en utilisant les JWT. Cette implémentation a été effectuée avant la décision de partir sur une solution d’hébergement locale, de ce fait le code d’authentification n’est plus nécessaire. Cependant, au moins le code et le processus d’authentification existe si un besoin survient de le réimplémenter, tel que potentiellement la mise en place d’une authentification locale.

**Connexion**

L’implémentation de l’authentification comprend plusieurs étapes. Tout d’abord le système de login. L’email et le mot de passe de l’utilisateur est envoyé au backend via une route POST. L’implémentation de la route de la route de login fonctionne de la façon suivante :

1. Vérification que l’email et le mot de passe fourni n’est pas nul
2. Vérification que l’utilisateur existe dans la base de données
3. Vérification que le hash du mot de passe correspond au hash de l’utilisateur dans la DB (utilisant la libraire bcrypt)
4. Création d’un token JWT signé avec un code secret stocké dans les variables environnement de l’application (Listing 2)
5. Retour d’une réponse succès en stockant comme cookie le token JWT (Listing 2)

Listing 2 Création d'un JWT

// Create JWT token for user

**const** jwtSecret **=** **new** TextEncoder**().**encode**(**secret**);**

**const** token **=** **await** **new** SignJWT**({**userId**:** user**.**id**})**

**.**setProtectedHeader**({**alg**:** "HS256"**})**

**.**setExpirationTime**(**'1h'**)**

**.**sign**(**jwtSecret**);**

// Return response with JWT

**const** response **=** NextResponse**.**json**({**message**:** "Login Successful!"**});**

response**.**cookies**.set({**

name**:** "authToken"**,**

value**:** token**,**

path**:** '/'**,**

httpOnly**:** **true,**

maxAge**:** 3600**,**

secure**:** **true**

**}** **as** any**);**

**Enregistrement**

Le processus de l’enregistrement fonctionne de telle manière :

1. Appel POST de la route gérant l’enregistrement, en y envoyant l’email, le nom d’utilisateur, et le mot de passe
2. Vérification que l’utilisateur n’existe pas déjà
3. Création de l’utilisateur et l’insertion de celui-ci dans la DB

Si l’appel de cette route d’enregistrement retourne un succès, un appel successif est envoyé à la route gérant le login afin de connecter directement l’utilisateur après son enregistrement.

**Middleware**

Un middleware est créé afin de protéger les pages nécessitant une connexion utilisateur. Ce middleware est effectué avant chaque demande d’accès à une page. Le middleware définit quelles pages sont publics (accessibles aux utilisateurs non-connectés) et quelles pages sont cachées aux utilisateurs connectés (page de login et d’enregistrement ne doivent plus être accessible aux utilisateurs connectés). D’un point de vue algorithmique, le Listing 3 montre comment le middleware fonctionne.

Listing 3 Pseudocode middleware

Définitions :

- Soit P la page demandée

- Soit PP l’ensemble des pages publiques

- Soit HP l’ensemble des pages cachées aux utilisateurs connectées

- Soit T le token JWT

- Soit isValid(T) vérifiant que T est présent, valide, et que l’expiration n’ait pas échouée

Pseudocode :

Si T ∉ cookies et P ∉ PP

Redirection vers /login

Sinon si T ∈ cookies

Si !isValid(T)

Redirection vers /login

Sinon si P ∈ HP

Redirection vers /

Sinon

Laisser passer

Sinon

Laisser passer

Actuellement seules les pages sont protégées par le middleware. Idéalement il faudrait également y intégrer la protection des routes API, qui sont actuellement manuellement protégées dans leurs implémentations.

### Routes API

L’applications expose plusieurs routes API RESTful permettant la gestion des utilisateurs et des évaluations. Ces routes sont sécurisées par un système de token (JWT) pour garantir que seules les opérations autorisées sont possibles

**Routes liées aux évaluations**

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode HTTP | Route | Auth. requise | Description |
| POST | /quiz | Oui | Crée une nouvelle évaluation. Les paramètres dans le body sont : titles (titre de l’éval) contentFiles (liste des fichiers de contexte) |
| GET | /quiz/[id] | Oui | Récupère une évaluation existante. L’utilisateur doit être le créateur de l’évaluation |
| PUT | /quiz/[id] | Oui | Permet la modification d’une question spécifique dans une évaluation. Paramètre attendu : editQuestion (données de la question modifiée). |
| PUT | /quiz/[id]/regenerate | Oui | Permet la régénération d’une question é partir d’un prompt personnalisé. Paramètres attendus : questionNumber (index de la question à régénérer), prompt (instructions de régénération). |
| GET | /user/quiz | Oui | Récupère toutes les évaluations créées pas l’utilisateur actuellement connecté. |

Tableau 4 Routes API liées aux évaluations

**Routes liées à l’utilisateur**

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode HTTP | Route | Auth. Requise | Description |
| GET | /user | Oui | Retourne les informations du compte utilisateur connecté. |
| GET | /user/files | Oui | Récupère les fichiers liés à l’utilisateur. Utilisé comme contexte pour la génération d’évaluations. |
| POST | /user/login | Non | Authentifie un utilisateur via les paramètres email et password, puis retourne un JWT. |
| POST | /user/register | Non | Crée un nouvel utilisateur avec email, username, et password. |
| GET | /user/logout | Oui | Déconnecte l’utilisateur et supprime son JWT. |

Tableau 5 Routes API liées aux utilisateurs

### Base de données

**Schéma de la base de données**

Le schéma prisma actuel de la base de donnée est tout simplement une entité User représentant un utilisateur, et Quiz représentant une évaluation, tel que démontré dans le Listing 4.

L’entité « User » a comme champ son nom d’utilisateur, son mot de passe (qui est stocké en tant que hash), et son email. Il a également le champ « quizes » représentant toutes les évaluations qu’il a générés (par une association one-to-many), mais ne sont pas stockées avec les utilisateurs. C’est l’ORM Prisma qui se charge a récupérer les quiz au runtime lorsque cela est nécessaire.

Listing 4 Schéma Prisma actuel

model User {

id Int @id @default(autoincrement())

username String

password String

email String @unique

quizes Quiz[]

}

model Quiz {

id Int @id @default(autoincrement())

title String

content Json?

createdAt DateTime @default(now())

prompts Json?

genModel String?

authorId Int

author User @relation(fields: [authorId], references: [id])

}

L’entité Quiz, comme décrit dans le chapitre 4.3.1 parlant du modèle conceptuel, stock le titre, le contenu en json, le timestamp de création, les prompts qui ont été utilisés pour sa génération, le modèle LLM l’ayant généré, ainsi que l’utilisateur l’ayant créé.

Bien évidemment c’est une version initiale de la base de données qui nécessite encore beaucoup de changements afin d’implémenter correctement le modèle conceptuel décrit dans le chapitre 4.3.1.

**Manipulation de l’ORM**

Next.js fait utilisation du « hot-reloading » lorsque l’application est en phase de développement (16). Cette fonctionnalité permet de refléter les modifications de code instantanément. Cependant, cela risque de créer des multiples instances de client Prisma, ce qui consomme des ressources et peut provoquer des comportements inattendus. Afin de gérer ceci, la documentation officielle de Prisma propose de traiter les clients Prisma en tant que singleton, garantissant qu’un seul client est utilisé. Cette instante est créé en utilisation une variable globale, telle que démontré dans le Listing 5.

Listing 5 Mise en place du client Prisma

// lib/prisma.ts

**import** **{** PrismaClient **}** **from** "@prisma/client"**;**

**const** globalForPrisma **=** global **as** unknown **as** **{** prisma**:** PrismaClient **};**

**export** **const** prisma **=**

globalForPrisma**.**prisma **||** **new** PrismaClient**();**

**if** **(**process**.**env**.**NODE\_ENV **!==** "production"**)** globalForPrisma**.**prisma **=** prisma**;**

Ensuite il suffit simplement importer Prisma depuis ce fichier, dans le fichier nécessitant une communication avec la base de données.

### Dashboard évaluations

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Figure 13 Implémentation actuelle du dashboard des évaluations

La Figure 13 montre l’implémentation actuelle du dashboard d’évaluation. En ce moment l’utilisateur a la possibilité d’afficher une évaluation, les rechercher par leur titre, ou générer une nouvelle évaluation.

### Génération des évaluations

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure 14 Formulaire de création d'une évaluation

La Figure 14 montre le formulaire de création d’évaluation actuel. Ce formulaire se trouve sur sa propre page, mais il serait plus judicieux d’un point de vue expérience utilisateur de le faire apparaitre comme pop-up sur la page du dashboard des évaluations. Ce formulaire permet une première version de création d’évaluation en fournissant un champ pour le titre, et un bouton pour uploader des fichiers qui seront utilisés comme contexte (pour l’instant ce sont que des fichiers provenant de la machine locale de l’utilisateur, et ils sont uniquement utilisé comme contexte de contenu et pas comme contexte d’inspiration).

Du côté backend, plusieurs étapes sont entreprises afin de générer l’évaluation, et plusieurs techniques de prompt engineering et de contextual prompting sont utilisées.

Les fichiers fournis sont en premier lieu uploadés en utilisant le « GoogleAIFileManager » provenant de l’API google. Cette fonctionnalité permet de temporairement uploader des fichiers pour utilisation dans une requête dédiée au LLM.

Suite à ça, une première requête de génération est envoyée en fournissant les fichiers précédemment uploadés comme contexte, comme démontré par le Listing 6. Cette première requête a pour but de déterminer les thèmes et les points clés des fichiers fournis, pour être par la suite utilisé comme contexte. Ceci est un cas classique de « contextual grounding », une technique de contextual prompting comme expliqué dans le chapitre 2.1.3.

Listing 6 Génération de contexte

**const** contextPrompt **=** `

Analysez le contenu des fichiers fournis et générez un résumé structuré des concepts clés abordés.

- Mélangez les thèmes des différents fichiers de manière cohérente.

- Identifiez les notions principales et expliquez-les clairement.

- Mettez en avant les points importants qui peuvent être utilisés pour créer une évaluation.

- Ne copiez pas directement le contenu des fichiers, mais reformulez et synthétisez.

- Présentez le résumé sous une forme organisée et lisible.

`;

// Generate context with uploaded files

**const** contextResult **=** **await** contextModel**.**generateContent**([**

contextPrompt**,**

**...**fileUploadResults**.**map**(**uploadResult **=>** **({** fileData**:** **{**

fileUri**:** uploadResult**.**file**.**uri**,**

mimeType**:** uploadResult**.**file**.**mimeType**,**

**}** **}))**

**]);**

Le prompt démontré dans le Listing 6 demande une génération des concepts clés des fichiers en utilisant une technique de prompt engineering nommée instruction-based prompting (chapitre 2.1.5). Voici l’objectif de chaque instruction utilisée :

* **Mélangez les thèmes**: Cela a été ajouté comme instruction afin d’éviter que les questions des évaluations générées suivent l’ordre logique des fichiers fournis, ce qui est le cas sans cette instruction. Cela permet d’avoir des évaluations avec des ordres bien diversifiées.
* **Identifiez des notions principales**: Permet d’extraire des concepts fondamentaux qui serviront de base à la génération des questions.
* **Mettez en avant points importants pour l’évaluation**: Aide à focaliser l’attention du modèle sur les éléments évaluables (définitions, raisonnements, etc.).
* **Pas de copie directe** : Assure que le texte généré ne soit pas un simple copier-coller du contenu source, mais une synthèse.
* **Présentation organisée et lisible** : Garantit la lisibilité du contexte généré afin de faciliter sa visualisation, et peut être utile dans le cas où une fonctionnalité permettant aux utilisateurs de visualiser/modifier le contexte intermédiaire serait implémentée.

Le Listing 7 montre un extrait d’un contexte généré via ce prompt, à partir de deux fichiers d’exercices.

Listing 7 Extrait de contexte généré

\*\*Concepts Clés et Explications\*\*

1. \*\*Listes et Récursion:\*\*

\* Les listes sont une structure de données fondamentale en programmation fonctionnelle.

\* La récursion est une technique essentielle pour traiter les listes, en définissant une fonction qui s'appelle elle-même avec une version plus petite de la liste jusqu'à atteindre un cas de base (souvent la liste vide).

\* \*\*Exemple:\*\* Écrire une fonction récursive pour combiner les éléments d'un vecteur et d'une liste en une liste de paires (Int, String).

\* \*\*Quiz Possible:\*\* Comment traiter une liste de manière récursive ? Quel est le cas de base typique ?

2. \*\*Pattern Matching:\*\*

\* Le pattern matching est une fonctionnalité puissante en Scala qui permet de décomposer des structures de données (comme les listes) et d'exécuter différents blocs de code en fonction de la structure.

\* \*\*Exemple:\*\* Implémenter l'opération `init` (retourne une liste contenant tous les éléments sauf le dernier) en utilisant le pattern matching pour gérer les cas de liste vide, liste à un élément, et liste avec plusieurs éléments.

\* \*\*Quiz Possible:\*\* Expliquer le fonctionnement du pattern matching avec les listes. Donner des exemples de patterns courants (::, Nil).

**…**

(Listing 8) Une fois le contexte généré, la technique de prompt engineering « prompt chaining » est utilisée, où la sortie d’un prompt devient l’input dans prompt, mélangé avec la technique de « contextual grounding » du contextual prompting. La technique de prompt chaining est utilisée ici afin de faire abstraction du contenu spécifique des fichiers. En-effet, durant les tests des résultats des différents prompts, la génération des évaluations directement avec les fichiers fournis avait tendance à générer des questions qui reprenait des exemples ou des exercices déjà présents dans les fichiers fournis.

La technique de schema-guided generation est également utilisée afin de faire en sorte que la sortie du prompt satisfait le schéma JSON d’une évaluation (chapitre 4.3.2). Ce schéma de sortie n’est pas indiqué dans le prompt en lui-même, mais dans la construction du modèle grâce à l’API fourni par Google.

Listing 8 Génération d'une évaluation

**const** quizPrompt **=** `

Générez un quiz basé sur le résumé suivant :

${contextText}

- Incluez au moins 10 questions.

- L'ordre des questions doit être indépendant de l'ordre des chapitres fournis.

- Assurez-vous que les questions couvrent les concepts du résumé.

- Variez le type de questions : QCM, questions ouvertes, compréhension de code, écriture de code.

- La quantité des différents types de questions doit être équilibrée.

- Les exercices doivent être difficiles est doivent nécessiter beaucoup de reflexion.`;

**const** quizModel **=** genAI**.**getGenerativeModel**({**

model**:** genModel**,**

generationConfig**:** **{**

responseMimeType**:** "application/json"**,**

responseSchema**:** quizSchema

**}**

**});**

**const** quizResult **=** **await** quizModel**.**generateContent**([**

quizPrompt

**]);**

Une fois l’évaluation généré, le JSON de l’évaluation est validé, puis inséré dans la base de données.

### Mode affichage

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

Figure 15 Mode affichage d'une évaluation

Actuellement le mode affichage d’une évaluation, comme montré dans la Figure 15, permet à l’utilisateur de visualiser l’intégralité des questions d’une évaluation, de voir les réponses aux questions, de les modifier et les regénérer. Aucun formatage de code est effectué et les autres fonctionnalités prévues lors de la conception de la maquette ne sont pas encore mis en place.

La Figure 16 montre le fonctionnement de l’édition d’une question. Des inputs apparaissent avec le texte (question et la réponse), puis l’utilisateur peux modifier les textes comme cela lui convient. Il peut soit annuler la modification puis retourner la question à l’état qu’elle était avant le commencement de modification, ou enregistrer les modifications, et les modifications seront enregistrées dans la base de données

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure 16 Edition d'une question

La Figure 17 et la Figure 18 montrent le fonctionnement de la régénération d’une évaluation. Lors de la demande d’une régénération, l’utilisateur peut décrire les modifications à effectuer à la question.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure 17 Démonstration de régénération de question

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure 18 Démonstration de régénération de question

La Figure 19 montre le code permettant la régénération d’une question. Ici les techniques de prompt engineering utilisées sont à nouveau le contextual prompting en fournissant la question à améliorer et ses données, l’instruction-based prompting en définissant les instructions de régénération, ainsi que le schema-based generation en fournissant le schéma JSON des questions.

Figure 19 Code de régénération d'une question

**const** questionRegenPrompt **=** `

Ta tâche est d'améliorer la question fournie en suivant précisément ces consignes :

- Respecte le format : Garde le même type de question (${question.questionType}).

- Améliore la clarté et la pertinence : Reformule si nécessaire pour plus de précision et d'intelligibilité.

- Respecte les contraintes utilisateur : Applique ces modifications demandées : "${prompt}".

Question originale à améliorer :

${JSON.stringify(question)}

`;

**const** regenModel **=** genAI**.**getGenerativeModel**({**

model**:** genModel**,**

generationConfig**:** **{**

responseMimeType**:** "application/json"**,**

responseSchema**:** questionSchema

**}**

**});**

Après régénération, la question est mise à jour dans la base données et retournée au frontend afin que l’affichage puisse être directement adaptée sans refresh de la page.

### Dashboard fichiers

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

Figure 20 Implémentation actuelle du dashboard des fichiers

La Figure 20 montre l’implémentation du dashboard des fichiers. Pour l’instant seule la partie frontend est implémentée, et aucun fichier n’est vectorisé ou enregistré. Les fichiers sont séparés par les cours auxquelles elles appartiennent.

# Conclusion

Ce travail a permis de poser les bases du projet, en définissant une première architecture, en explorant les outils et technologies disponibles, et en réalisant une implémentation initiale. Plusieurs choix technologiques ont pu être validés (framework, ORM, base de données), tandis que d’autres restent encore à finaliser, notamment au niveau du modèle LLM.

Pour la suite, il reste encore l’implémenter :

* Gestion complète des fichiers (Vectorisation, stockage, etc.)
* Intégration du pipeline RAG
* Choix définitif du modèle LLM (ainsi que son hébergement et éventuellement son fine-tuning)
* Amélioration des prompts afin de maximiser la pertinence des générations
* Mise en place du système de contexte d’inspiration
* Mise en place des tests unitaires et E2E
* Containerisation de l’application et de la base de données
* Implémentation du « mode test » si le temps le permet

Si le processus jusqu’à présent était à refaire, plus de temps aurait été consacré à la conception et la recherche avant de se lancer dans l’implémentation. Une phase de conception a été faite initialement, mais la rédaction de ce rapport fait remarquer qu’elle n’a pas été assez approfondie, et que cela aurait pu engendrer des implémentations pas forcément nécessaires tels que l’implémentation de l’authentification alors qu’au final une stratégie d’hébergement locale est adoptée, ainsi que l’utilisation de Gemini et l’API offerte par Google qui est potentiellement prémature étant donné qu’au final elle n’offre aucun moyen d’hébergement local. Cependant, cette première implémentation a permis de bien visualiser la structure globale, et peu servir comme première itération.

# Utilisation de l’IA

Des modèles IA sont utilisés afin de faciliter la réalisation du projet, que cela soit GitHub copilot permettant d’augmenter la productivité de l’implémentation, ou des modèles en-ligne facilitant la rédaction du rapport (tournures de phrases, grammaire, orthographe, etc.).

# Bibliographie

1. Critères d'évaluation. *CollegeAhuntsic.* [En ligne] [Citation : 23 05 2025.] https://www.collegeahuntsic.qc.ca/departement-de-philosophie/livret-departemental/criteres-devaluation.

2. Critères I Evaluation de ressources pédagogiques. *education21.* [En ligne] 2012. [Citation : 23 05 2025.] https://www.education21.ch/sites/default/files/uploads/pdf\_fr/Criteres\_evaluation\_ressources\_ECM%26EE.pdf.

3. Dorvaux, Aurélien. Comment concevoir un quiz efficace ? *sydologie.* [En ligne] 03 06 2024. [Citation : 23 06 2025.] https://sydologie.com/2024/06/comment-concevoir-un-quiz-efficace/.

4. Qu'est-ce qu'un grand modèle linguistique (LLM) ? *cloudflare.* [En ligne] [Citation : 23 05 2025.] https://www.cloudflare.com/fr-fr/learning/ai/what-is-large-language-model/.

5. What is a Large Language Model (LLM). *GeeksForGeeks.* [En ligne] 22 01 2025. [Citation : 23 05 2025.] https://www.geeksforgeeks.org/large-language-model-llm/.

6. Transformers in Machine Learning. *GeeksForGeeks.* [En ligne] 27 02 2025. [Citation : 23 05 2025.] https://www.geeksforgeeks.org/getting-started-with-transformers/.

7. LLM Architecture: Exploring the Technical Architecture Behind Large Language Models. *GeeksForGeeks.* [En ligne] 20 09 2024. [Citation : 23 05 2025.] https://www.geeksforgeeks.org/exploring-the-technical-architecture-behind-large-language-models/.

8. Zarecki, Iris. Prompt Engineering vs Fine-Tuning: Understanding the Pros and Cons. *k2view.* [En ligne] 21 04 2025. [Citation : 23 05 2025.] https://www.k2view.com/blog/prompt-engineering-vs-fine-tuning/#Prompt-engineering-defined.

9. Shah, Drishti. Prompt engineering techniques for effective AI outputs. *portkey.* [En ligne] 24 12 2024. [Citation : 23 05 2025.] https://portkey.ai/blog/prompt-engineering-techniques#:~:text=Contextual%20Prompting,-Contextual%20prompting%20enriches&text=The%20prompt%20starts%20with%20essential,closely%20with%20the%20user's%20expectations.

10. Prompt Engineering Guide. *https://www.promptingguide.ai/.* [En ligne] 24 04 2025. [Citation : 23 05 2025.] https://www.promptingguide.ai/fr.

11. LLM-as-a-Service. *deepchecks.* [En ligne] [Citation : 23 05 2025.] https://www.deepchecks.com/glossary/llm-as-a-service/.

12. Rate limits. *Google.* [En ligne] 21 05 2025. [Citation : 23 05 2025.] https://ai.google.dev/gemini-api/docs/rate-limits.

13. Gemini models. *Google.* [En ligne] 21 05 2025. [Citation : 23 05 2025.] https://ai.google.dev/gemini-api/docs/models.

14. Models Overview. *Mistral.* [En ligne] [Citation : 23 05 2025.] https://docs.mistral.ai/getting-started/models/models\_overview/.

15. Draft 2020-12. *JSON Schema.* [En ligne] [Citation : 23 05 2025.] https://json-schema.org/draft/2020-12.

16. Comprehensive Guide to Using Prisma ORM with Next.js. *Prisma.* [En ligne] [Citation : 23 05 2025.] https://www.prisma.io/docs/orm/more/help-and-troubleshooting/nextjs-help.