Jules HIRTZ
Yann MIJATOVIC
Théo PINCHON
Alexandre PERROT

Mimir
Tuteur : Yann BONIFACE
Année 2023 - 2024

MIMIR

Rapport de fin de projet

IUT nancy Charlemagne

Table des matières

Introduction

Le projet que nous avons imaginé est de proposer une plateforme éducative. L'application se nomme Mimir en référence à la divinité nordique responsable de la sagesse et de la connaissance.

Dans un premier temps l'objectif principal est de créer un système de flashcards (questions / réponses), dans des decks. (*un deck est un paquet de cartes*)

Prenons l'exemple d'un étudiant ayant pour but d'aller dans des écoles d'ingénieur à la suite de son BUT à l'IUT Nancy-Charlemagne dans la branche ingénierie logicielle. Afin de réviser et d'apprendre son cours de logique le mieux possible, il aura simplement à se connecter à l'outil et ses révisions seraient organisées automatiquement par le système. Plus besoin de perdre des heures à faire des fiches de révisions, plus de stress pour savoir par quels cours commencer, comment se tester, ou si nos connaissances sont réellement apprises et mémorisées ou si c'est dans notre mémoire instantanée.

Notre application gérera tous les aspects de l'apprentissage profond et de la compréhension des concepts à travers une sorte de quizz (à l'image d'ANKI (https://apps.ankiweb.net/), en mieux).

Par ailleurs, si le système venait à être utilisé par plusieurs personnes, on pourrait imaginer un partage de decks (groupe de flashcards) entre étudiants, voire entre professeurs et étudiants. C'est-à-dire qu'un professeur pourrait préparer facilement les cartes liées à la théorie du cours et partager le deck à ses groupes de classe, s'assurant ainsi de la qualité des informations directement retenus par les étudiants. En plus de vouloir créer un outil que nous aurions aimé avoir, nous cherchons à faire d'une pierre deux coups en utilisant le plus possible les technologies les plus efficaces et en vogue dans le domaine de l'ingénierie logicielle web. De plus nous avons déjà pensé à une architecture facilement scalable afin de pouvoir continuer à maintenir le projet dans les années à venir si besoin.

Analyse

Visuels

Maquette Figma

Nous avons décidé de créer une maquette sur l'application web Figma (https://www.figma.com/file/Wifj3LLdOS5zvpNooBIMam/mimir?type=design&node-id=0-1&mode=design) afin d'avoir une idée de la charte graphique du site. Nous avions déjà pu en créer une lors de l'étude préalable de l'application mais celle-ci était trop légère et elle ne correspondait pas à l'idée que nous avions de l'application, c'est-à-dire minimaliste et intuitive pour l'utilisateur.

Nous avons donc recréé une maquette lors de notre deuxième itération pour avoir une idée claire du visuel du site. Nous voulons que l'utilisateur ne se perde pas dans le site, notamment dans les statistiques. Il faut que l'utilisateur ait à sa disposition seulement les informations dont il a besoin. S' il veut afficher des choses en plus, libre à lui de cliquer sur les informations supplémentaires.

La maquette est aussi utile pour nous, développeurs, pour avoir une idée claire de ce qu'on doit coder, surtout au niveau des pages et des composants React. Cette maquette, malgré le temps nécessaire à sa création, fut un gain de temps énorme sur le développement du site.

Composants React

Le but de créer la maquette était donc notamment de créer les composants React. Nous avions choisi d'utiliser React justement pour ce genre de cas. Par exemple, on voit sur la maquette que nous avons créé qu'il y a beaucoup de decks qui sont les mêmes visuellement à chaque fois.

Pour ne pas réécrire le code, nous créons donc des composants React qui seront des gains de temps précieux pour les différents onglets de notre application (voir UI/UX de la partie Réalisation pour plus d'informations)

Utilisation de l'intelligence artificielle

Génération des cartes à partir d'un PDF

On souhaite pouvoir générer des cartes à partir d'un PDF pour plusieurs raisons.

Dans un premier temps, on souhaite que notre application soit unique en son genre, que ca soit en terme de projet universitaire et en terme de concurrence (Quizlet, Anki, … qui sont également des plateformes éducatives où l'on apprend ses cours via des cartes).

Dans un second temps, on souhaite accélérer le processus de création des cartes de l'utilisateur parce que l'objectif principal de l'application, ce n'est pas passer son temps à créer des cartes. Mais plutôt de réviser son cours par le biais des cartes. Pour cela, le support de cours des étudiants est souvent des PDF ainsi que des PowerPoint, donc l'idée est d'utiliser le contenu de ces support, d'en extraire les informations et de garder uniquement les questions pertinentes ainsi que leur réponse et de créer les cartes automatiquement à la place de l'utilisateur.

Une nouvelle technologie émergente est apparue en 2018 mais de plus en plus accessible, les LLM (Large Language Model) qui nous permet de générer du contenu en fonction d'une requête et d'un contexte donnée. C'est la solution idéale pour ce qu'on souhaite faire, puisque cette technologie, si elle est utilisée correctement et avec les bonnes ressources,

pouvait générer des questions avec leur réponse en cohérence avec le support de cours fourni.

Au fur et à mesure de nos essais sur différents modèles de LLM, on s'est rendus compte que les LLM sont moyennement bons pour générer des questions, et vraiment mauvais pour générer des réponses à des questions générées par lui-même. Nous avons donc pris la décision de conserver notre travail en faisant en sorte que cette IA possède un rôle d'assistant à la création de decks, qui propose des questions et des réponses à l'utilisateur qui seront modifiables par la suite. L'IA sera juste utile pour générer un squelette au deck et possiblement donner des idées de questions à poser par rapport à un cours PDF donné.

Vérification des réponses

On souhaite également pouvoir vérifier la réponse de l'utilisateur par rapport à la réponse attendue de la carte dans un contexte sémantique et non syntaxique dans une question ouverte. Imaginons qu'on demande à un utilisateur comment fonctionne la descente de gradient, la réponse que l'utilisateur propose et la réponse attendue auront sensiblement la même idée mais pas forcément la même construction syntaxique, et notre vérificateur doit pallier ce problème et valider la réponse.

Dans notre application, il y a plusieurs types de réponse :

- Réponse syntaxique strictement exacte
- Auto-évaluation par l'utilisateur
- Réponse sémantique similaire à celle attendue
- Réponse à choix multiple

Pour pouvoir comparer sémantiquement deux réponses entre elles, il faut utiliser un certain type d'intelligence artificielle, pas forcément un LLM puisque ce n'est pas son objectif de base, mais plutôt un NLP (Natural Language Processing) où son objectif est de traiter du texte, cela sera plus précis et moins coûteux en ressources.

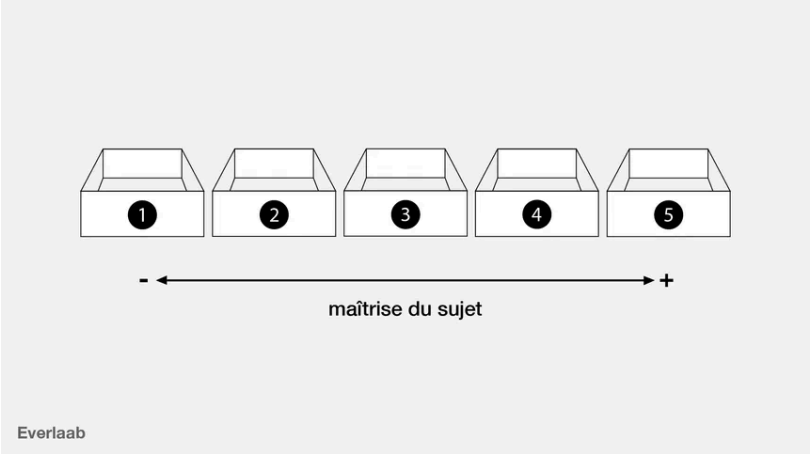
Si jamais le NLP n'est pas suffisant pour évaluer la similarité d'une réponse, on envisage d'utiliser la distance de Levenshtein qui est une fonction mathématique permettant d'évaluer la distance entre deux chaînes de caractère et plus cette distance est minime, plus les chaîne sont similaire, mais cette méthode compare syntaxiquement et non sémantiquement comme le NLP.

Méthodologie

Système de Leitner

Le système de Leitner est une méthode de tri dans des boîtes assez intéressantes qui permet de trier les cartes selon si elles ont été bien révisées par l'utilisateur. Plus l'utilisateur répond juste aux cartes, moins elles reviendront souvent lors de ses prochaines révisions de

ce deck. Ce système est très important pour notre application car il permet d'accentuer l'apprentissage des utilisateurs en lui permettant de réviser au maximum ses points faibles sans pour autant négliger ses points forts.



Ce système permet aussi à notre application de se démarquer de la concurrence en proposant un système unique d'apprentissage.

Social

Lien entre utilisateur

Le lien entre utilisateur dans notre application se fait sur plusieurs points. Tout d'abord, sur le fait d'avoir des contacts en tant que followers / following (comme un réseau social). Ce système permet aux utilisateurs de suivre les sorties de decks d'utilisateurs très suivis, qui sont plus susceptibles de créer des decks de qualités.

L'application permet aussi le fait de pouvoir mettre ses decks en public afin qu'ils soient répertoriés dans le marketplace. Ce marketplace permet d'accéder à tous les decks publics de l'application. Il permet donc d'accéder à ce dont l'utilisateur a besoin. Si l'utilisateur désire accéder à des mathématiques, il peut directement le faire en recherchant des decks sur le marketplace.

Réalisation

UI/UX

Avant d'avoir commencé ces itérations, nous avions réalisé une maquette de l'application Web sous Figma dans l'étude préalable qui décrivait vaguement l'application Web à réaliser. Mais lors de la première itération, nous nous sommes rendus compte que la maquette n'allait pas dans le sens du projet. Nous avons donc refait une maquette entièrement afin qu'elle soit dans un premier temps plus esthétique que l'ancienne et que l'équipe puisse avoir un support pour rédiger plus facilement les composants React.

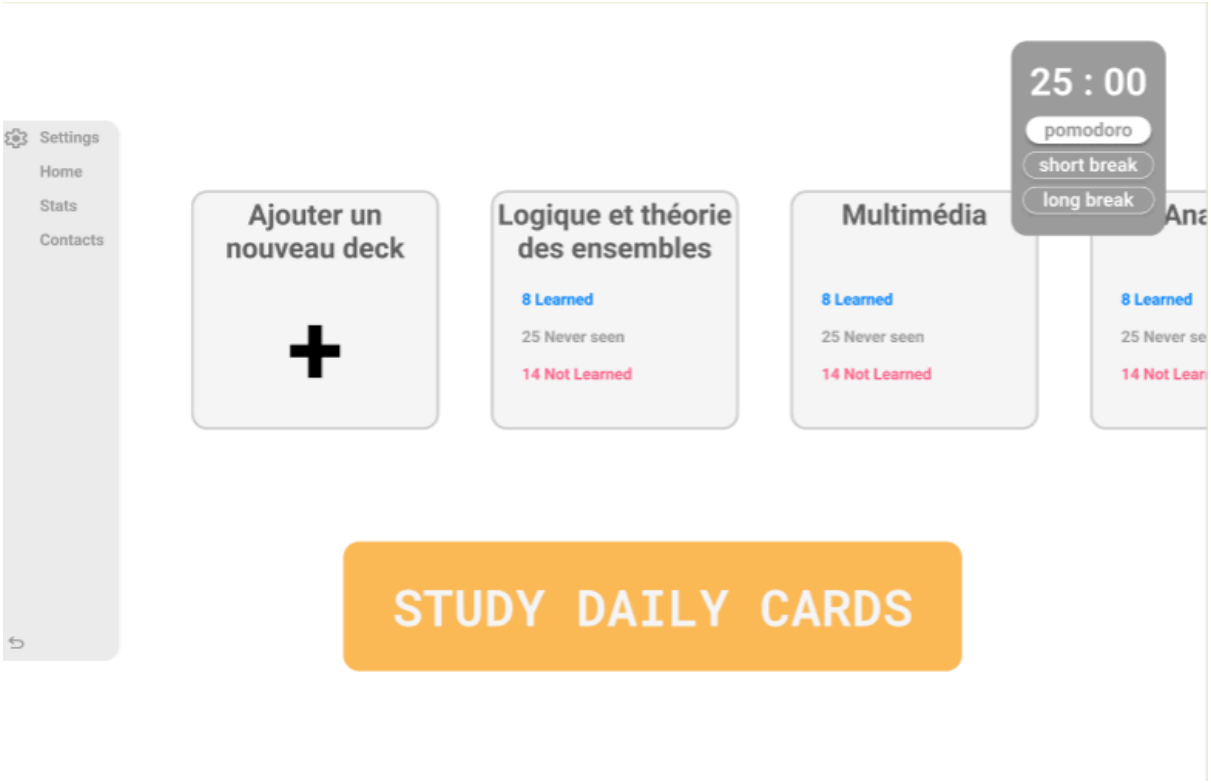


Figure 1 : La première version de la maquette

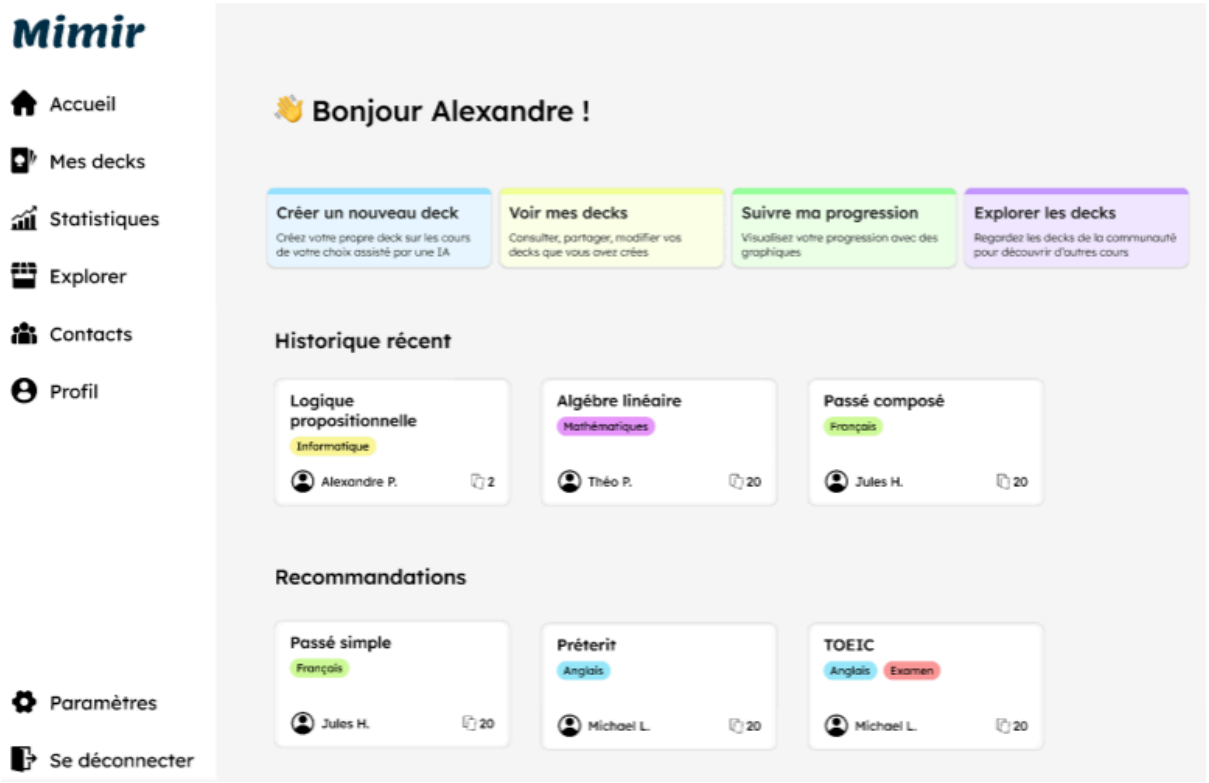


Figure 2 : La deuxième et dernière version de la maquette

Ensuite nous avons passé une itération à implémenter la maquette sous React, qui est une librairie sous Javascript nous permettant de mettre en place des pages Web rapidement sous forme de composants réutilisables afin d'éviter de recopier du code en boucle (par exemple, les decks que vous voyez dans la figure 2). En effet, en plus d'inclure toutes les balises HTML, on peut découper ces balises afin d'éviter la redondance, par exemple l'affichage d'une carte qui est présente quasiment partout sur notre site Web. On déclare une carte sous forme d'un composant React, on écrit le code HTML (plus précisément sous forme de JSX qui est une extension du code Javascript pour React et qui ressemble au HTML) et on peut utiliser ce composant sur n'importe quelle page qu'on souhaite. Cela permet de rendre le projet beaucoup plus maintenable d'un point de vue développeur. On utilise également Tailwind CSS qui est un framework CSS et qui nous permet d'implémenter du CSS rapidement sur les balises HTML, l'avantage de Tailwind CSS et de React est que, comme on travaille par composant, on applique le CSS dans les balises HTML une fois et cela rend le projet plus maintenable que si on travaillait avec des fichiers CSS individuel.

Lien vers TailWind : https://tailwindcss.com/
Lien vers React : https://fr.react.dev/

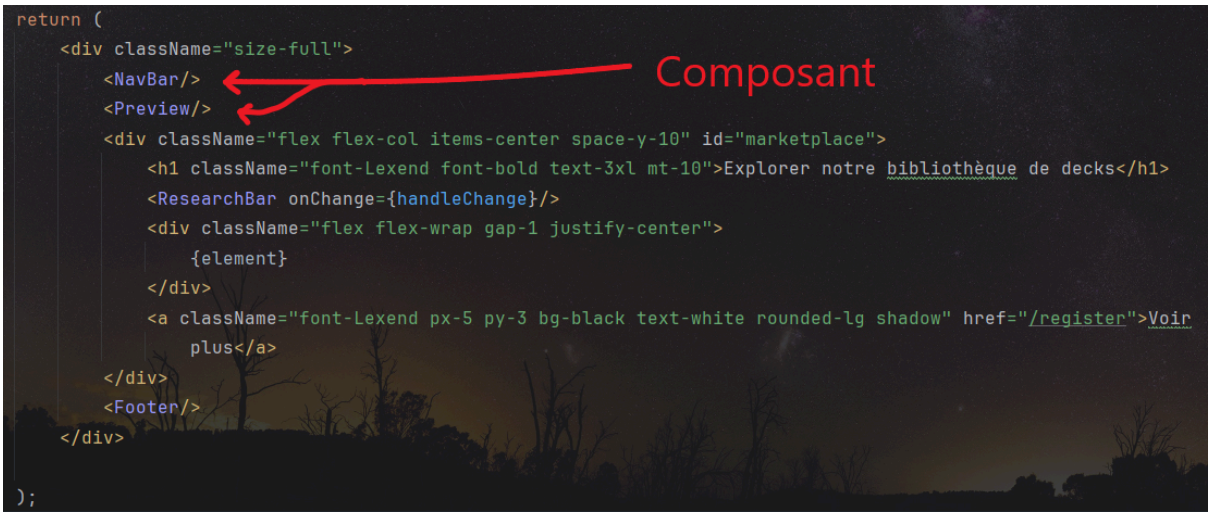


Figure 3 : Exemple d'une page écrite sous React, la page d'accueil, où l'on peut voir que certains éléments ont été écrits sous composant pour pouvoir les réutiliser, comme la barre de navigation, la barre de recherche, etc…

Lors de l'itération 5, nous avions des problèmes d'affichage avec notre application Web. En effet, le site n'était pas vraiment responsive, c'est-à-dire qu'il ne s'adaptait pas à toutes les tailles d'écrans. Nous avons donc passé un bon moment à fixer ce problème pour que l'application s'adapte au mieux sur tous les appareils.

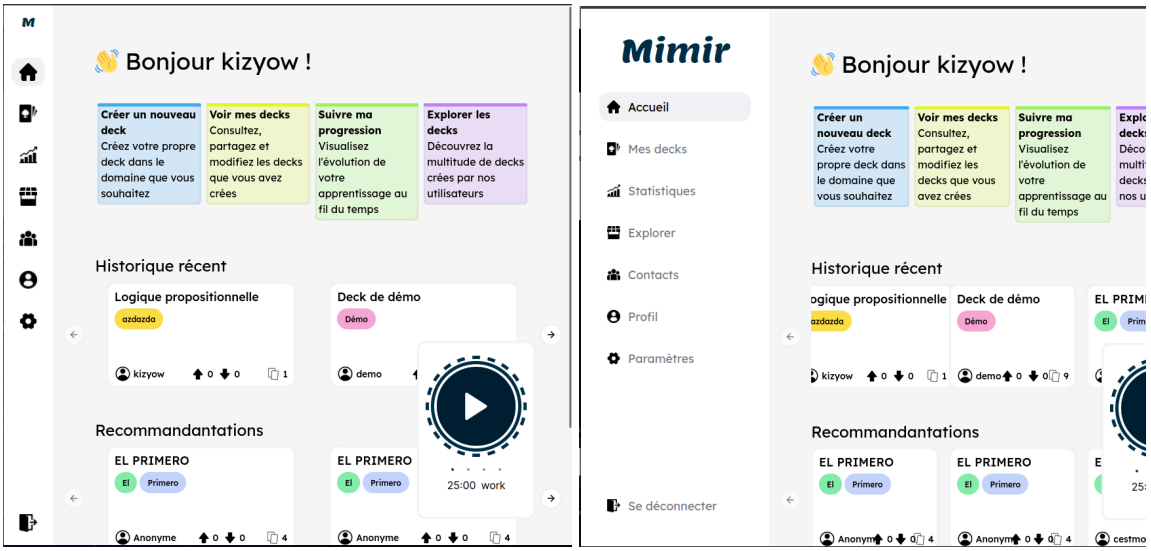
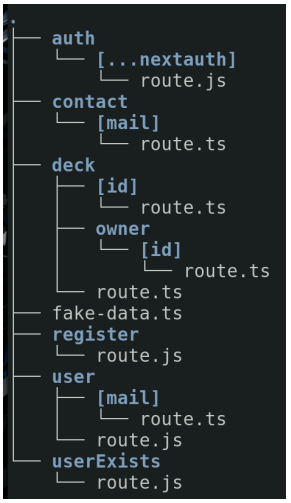


Figure 4 : Illustration de la barre gauche qui se rétracte lorsque l'écran devient trop petit.

Logique métier / Back-end

Parler de "back-end" sous-entend généralement la liaison entre l'interface utilisateur et la base de données.
Nous avons choisi une implémentation No-Sql de la BDD (Base de données). Dans un environnement de développement, cela a pour intérêt de pouvoir facilement adapter ou modifier la structure de documents manipulés selon des imprévus.
La BDD est stockée en cloud grâce au serveur gratuit de Mongo-Atlas.

Côté serveur, la librairie mongoose permet d'effectuer la connexion au serveur et le transfert des différents documents au serveur de la même façon qu'en local. La librairie propose tout un panel de fonctions similaires à celles d'une base Sql telles que find, delete ou encore update. Malgré la simplicité d'utilisation de ces fonctions, l'environnement next construit tous ses composants par défaut du côté serveur donc pouvant utiliser les fonctions de la BDD. Pour un souci de rendu dynamique, de nombreux composants restent définis du côté client, il faut donc instaurer des routes permettant au client web de communiquer avec le serveur et ainsi pouvoir accéder à l'ensemble des données par l'utilisation de fonctions dites "fetch".



En NextJs, l'implémentation de routes est relativement simple. Comme tout composant est Serveur, il suffit de créer une nouvelle page dans laquelle on ne définit pas la fonction Page ni de retour Visuel (JSX Element). Après la création des différentes pages utiles, on obtient donc la hiérarchie suivante pour le dossier "api".
Chaque page définit uniquement les méthodes HTTP qui lui sont utiles. Prenons l'exemple de la page /deck/route.ts, elle représente l'ensemble des decks. Sur cet ensemble, il est possible d'ajouter un nouveau deck ou de récupérer la liste entière.
Par respect de la syntaxe au fetch que l'on effectue sur cette page, il est important de ne définir que des méthodes HTTP comme par exemple : PUT, POST, DELETE et bien d'autres. Par semantique, notre page exemple va devoir définir respectivement les méthodes POST et GET.
De la même façon, toute action concernant un seul et unique deck devra passer par la page /deck/[id]/route.ts. La syntaxe [id] est appelée slug, dans un environnement NextJS elle représente une variable présente dans les paramètres de recherche.
Nous savons maintenant comment communiquer avec la BDD mais utilisant du ts (TypeScript) pour les composants visuel, le serveur doit encore convertir le format json reçu de la BDD en objet. Pour cela, nous utilisons des modèles.

La librairie mongoose utilise un système de "models" qui ne sont compilés qu'une seule fois puis toujours transmis à la manière d'un patron singleton. Un modèle prend 2 paramètres à sa construction :

- une interface nécessaire à l'utilisation du document dans le context ts.
- un schéma décrivant la structure du document présent en bdd.

De cette façon, toute page nécessitant des données de la bdd utilise la fonction useEffect pour exécuter la requête au serveur sans retarder l'affichage de la page. Une fois les données récupérées, la page se charge de nouveau avec les nouvelles informations.

La librairie NextAuth a permis de mettre en place les pages de login et de registrer ainsi que la protection des pages de l'application pour les utilisateurs non connectés. Nous avons dû mettre en place de nouvelles routes api et configurer les fichiers de base de next pour automatiser la recherche de l'utilisateur dans la bdd pour savoir ensuite s' il était connecté. Cette session nous permet également de récupérer facilement différentes informations à propos de l'utilisateur courant.

Intelligence artificielle

Nous avons mis en place une fonctionnalité de génération de cartes à partir d'un PDF, à l'aide d'une intelligence artificielle, plus précisément un LLM (Large Language Model) dont l'objectif est de pouvoir générer du contenu à partir d'un contexte et d'une requête spécifique. Dans notre cas nous utilisons Vigogne, un modèle fine-tuné en français, en qualité moyenne afin qu'il puisse générer rapidement les cartes et parce que nous avons des ressources limités (un serveur tournant sous Ubuntu avec 32Go de RAM et une carte graphique NVIDIA RTX 3060 Ti). Un LLM consomme beaucoup de puissance surtout si le modèle est plus gros, cela peut remplir 60Go de RAM et peut être très long à générer une réponse (une vingtaine de minutes). Mais grâce à une plateforme nommée Ollama, cela nous permet d'intégrer des modèles de LLM optimisés et ainsi qui ne remplissent pas toute la RAM du serveur.

Pour pouvoir délivrer le contexte au LLM, il faut pouvoir extraire les informations d'un PDF, et pour cela, nous utilisons une librairie nommée PyMuPDF, qui est également sous Python et qui est la librairie d'extraction le plus rapide et le plus fiable du marché (quelques millisecondes et 95% d'extraction totale).

Ensuite il fallait mettre en place un mini serveur Web pour pouvoir communiquer entre notre application Web (qui est sous Next.js / Typescript), Ollama (qui possède son propre REST API), l'extracteur de texte et le vérificateur de réponse. Pour faire cela, nous avons utilisé Flask qui est un micro-framework de serveur Web facile à mettre en place sous Python, et qui nous permettra de transmettre toutes ces informations entre les différents services.

Lien vers PyMuPDF : https://pymupdf.readthedocs.io/en/latest/
Lien vers Ollama : https://ollama.com/

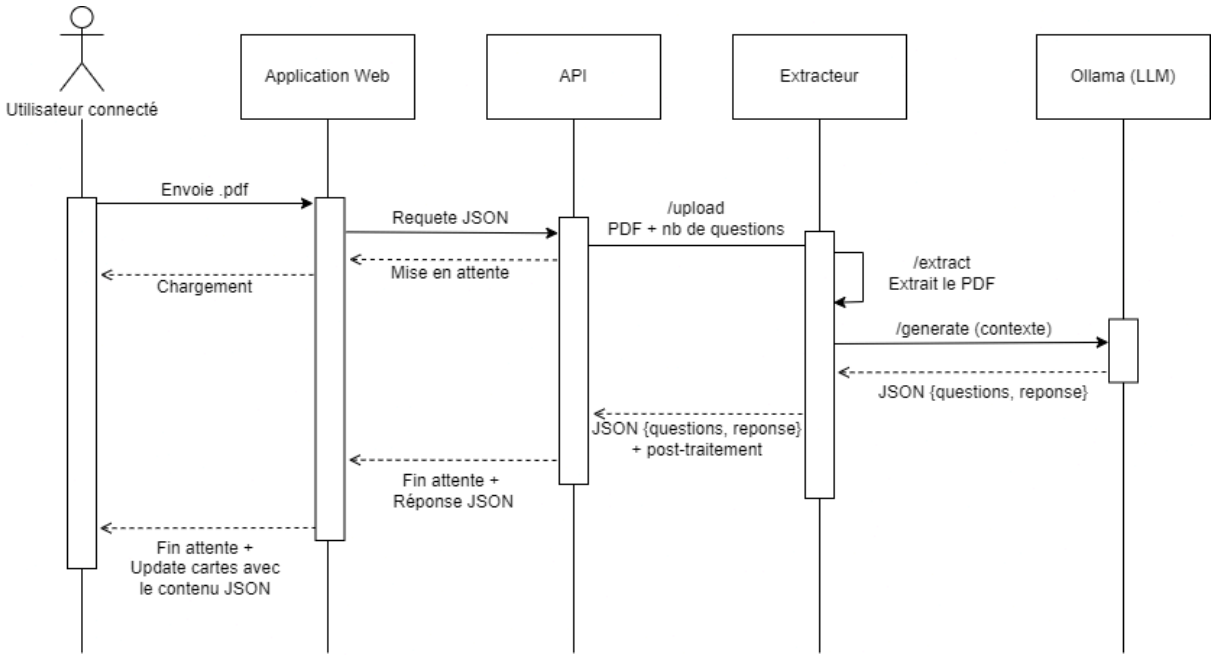


Figure 5 : Diagramme de séquence sur le fonctionnement de la génération des cartes

Nous avons également utilisé un autre type d'intelligence artificielle pour la vérification des réponses à des cartes. En effet, on souhaite pouvoir évaluer les réponses des l'utilisateur en similarité pour les questions ouvertes au lieu d'une égalité stricte. On avait pensé à réutiliser le LLM pour pouvoir comparer les deux réponses et de lui demander son avis sur le fait qu'elle soit similaire, sur le papier, ça semblait être une bonne idée, mais malheureusement à l'implémentation, cela a été un échec complet, lorsqu'on répondait "je sais pas", l'IA validait la réponse alors que cela n'aurait pas dû.

Donc nous avons recherché d'autres moyens pour comparer sémantiquement des phrases. Nous avons trouvé et utilisé le framework SentenceBERT, qui est le le meilleur en ce qui concerne le traitement des mots, des phrases et des images. Et qui nous permet d'évaluer deux phrases (la réponse attendue et la réponse donnée de l'utilisateur) par un pourcentage de similitude sémantique, cela signifie que, bien que la phrase soit différente, si elle veut dire la même chose que la réponse attendue, alors elle aura un pourcentage de similitude élevée. Et enfin si la similitude atteint un certain seuil (dans notre cas, 75%, alors on valide la réponse donnée par l'utilisateur). Cette technologie est simple à mettre en place puisqu'elle consomme beaucoup moins de ressources qu'un LLM comme décrit ci-dessus et s'implémente en Python.

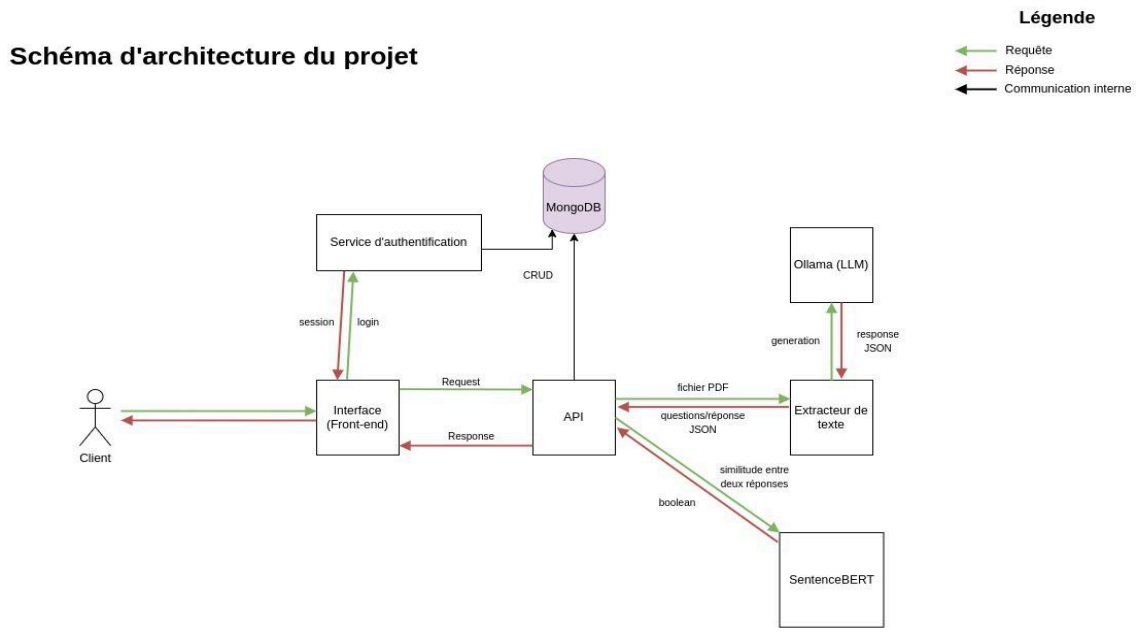


Figure 6 : Un schéma de l'architecture du projet dans l'ensemble

Docker

Notre projet comporte plusieurs sous-projets, à savoir, l'application Web en elle-même ainsi que l'extracteur de texte, le vérificateur de réponse ainsi que l'IA générative. Pour qu'un

autre développeur ou même un utilisateur lambda installe le projet, il fallait compter plusieurs heures et beaucoup de connaissances techniques pour que tous ces projets puissent fonctionner correctement et puissent communiquer entre eux, ce qui était très complexe.

Nous avons donc décidé de mettre en place Docker, qui est un logiciel de conteneurisation et qui permet d'isoler des applications, ce logiciel est très utile puisqu'il nous permet d'abord d'isoler l'application de toutes dépendances de notre système, et ainsi on garantit que l'installation du projet fonctionnera sur tous les systèmes d'exploitation. Il nous permet également de configurer le réseau interne des différentes technologies à notre guise et va faciliter la communication des différents projets entre eux.

Finalement, on peut lancer le projet à partir d'une seule ligne de commande et tout le projet se lance, sans que le développeur et l'utilisateur final n'ait eu à faire quoi ce soit (excepté de configurer les variables d'environnement)

Pour mettre en place Docker, on a d'abord créé plusieurs Dockerfile pour les différents projets, ce sont des fichiers qui décrivent comment doit s'installer ce projet, avec quelles dépendances, etc…

Et nous avons mis en place Docker Compose, qui permet de contrôler le stack plus précisément, au lieu de lancer un par un les containers, le Docker Compose s'occupe de lancer tous les services, d'établir un réseau pour que les projets puissent communiquer ensemble.

On a configuré exactement deux Docker Compose :

- Le compose par défaut, est en mode CPU-only, c'est à dire qu'il fonctionnera uniquement avec le processeur, donc il démarre le projet sans l'IA générative car elle requiert une carte graphique pour pouvoir fonctionner correctement, avec uniquement le processeur, non seulement elle est moins rapide, mais prend encore plus de ressources au système pour un résultat moins pertinent.
- Le compose-gpu qui contient l'intégralité du projet utilise la carte graphique, pour pouvoir profiter de l'IA générative.

Le fait d'avoir mis en place Docker pour le projet simplifie énormément l'installation pour quiconque puisqu'on peut expliquer les étapes facilement et que l'installation peut fonctionner peu importe le matériel qu'on utilise.

Statistiques

Le principe premier de cette application web est de favoriser l'apprentissage pour les étudiants (ainsi que tout autre utilisateur désireux d'apprendre).
C'est donc tout naturellement qu'un système de statistiques existe afin de suivre sa progression au sein d'un deck.
Nous pensions tout d'abord que le système de statistiques était quelque chose de simple sachant que nous avons toutes les informations. Nous nous sommes vite rendus compte

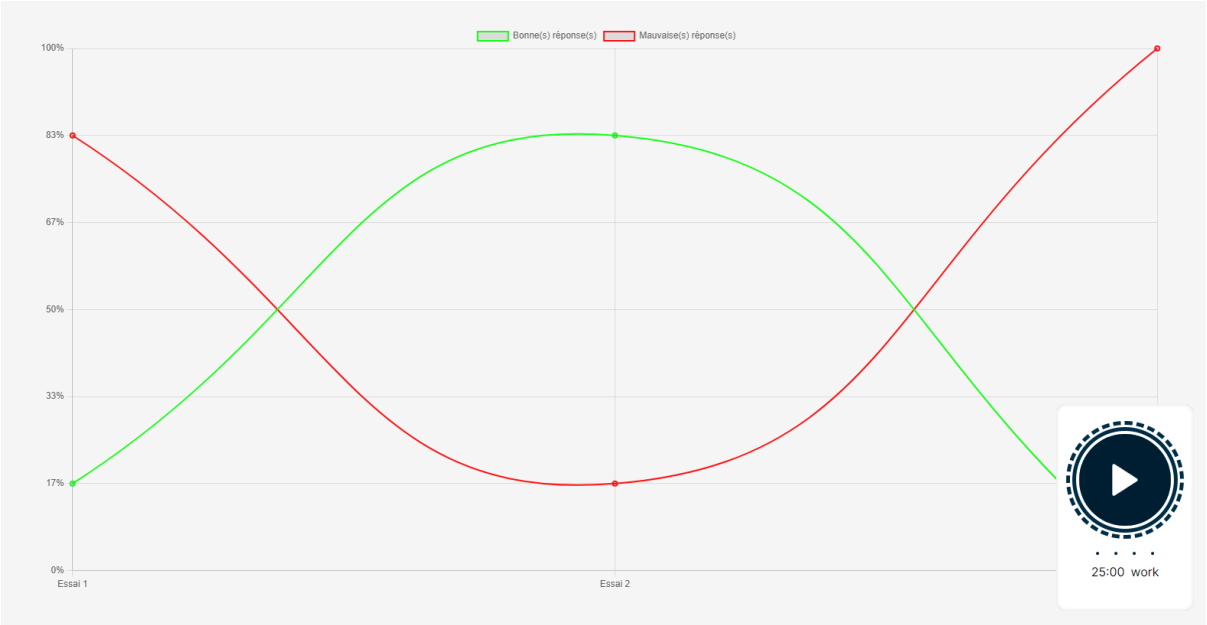
que le système de decks que nous avons mis en place n'allait pas convenir aux statistiques que nous voulions implémenter.

Nous avons donc dû revisiter la structure des decks afin de pouvoir conserver les parcours de decks de tous les utilisateurs, sachant qu'un deck peut être public ou privé et peut passer d'un état à l'autre selon les choix du créateur.

Nous avions eu au départ l'idée de différencier le public du privé mais c'était donc finalement impossible. A la place, nous avons laissé la possibilité à tous les decks d'avoir un suivi personnalisé pour tous les utilisateurs l'ayant parcouru au moins une fois.

L'objectif était d'avoir un suivi clair et interactif pour que l'utilisateur comprenne les données affichées à l'écran au premier coup d'œil. Nous avons utilisé pour cela ChartJS qui est une librairie permettant de créer des graphiques à partir de données.

Ces graphiques s'affichent au clic sur un des decks présents dans l'historique. Ils sont donc déterminés par les données des decks inhérentes à l'utilisateur. L'utilisateur peut donc parcourir ses decks et afficher le graphique du deck qu'il désire.



Exemple de graphique après clic sur un des decks

Planning et répartition du travail

Itération 1

Création de l'architecture de l'application (**Jules**)

Mise en place d'une application morte (visuel) (**Théo**)

Mise en place de la base de donnée MongoDB (**Jules**)

Preuve de concept de l'intelligence artificielle
- Tests des différents modèles de LLM (**Yann**)
- Mise en place de l'extracteur de texte d'un PDF (**Alexandre**)
- Liaison entre l'extracteur et le LLM (**Yann / Alexandre**)

Itération 2

Intégration du système d'authentification (**Jules**)
- Adresse e-mail / Mot de passe
- Stockage dans la session

Implémentation des classes métier (**Théo**)
- Informations utilisateur
- Informations d'un deck

Optimisation de l'IA
- Optimisation de la génération du LLM (**Yann**)
- Envoi des données vers la base de donnée (**Yann**)
- Retirer les infos inutiles d'un PDF (**Alexandre**)

Refonte de la charte graphique (**Alexandre**)
- Maquette beaucoup plus détaillée et esthétique sur Figma

Itération 3

Mise en place du nouveau visuel (**Yann / Alexandre**)
- Coder les composants React pour chaque page de la maquette

Mise en place de plusieurs endpoint pour faciliter l'intégration (**Théo**)
- Informations d'un deck et de ses cartes
- Informations d'un utilisateur

Merge de tout le travail du groupe (back-end / front-end)

Liaison du front-end avec le back-end
- Système de contacts (**Alexandre**)
- Système de création / modification d'un deck (**Théo**)
- Page "Accueil" (**Théo**)
- Page "Mes decks" (**Théo**)
- Page "Profil" (**Jules**)

Itération 4

Continuer la liaison du front-end avec le back-end
- Page “Explorer” (**Yann**)

Intégration de l’IA dans l’application (**Alexandre**)

Vérificateur de réponse à une carte (**Yann / Alexandre**)

Régler les bugs visuels de l’application (**Jules**)

Modification de la logique back-end sur le deck (**Yann**)

Test de l’application **(Alexandre)**

Répondre à une carte (**Théo**)

Itération 5

Changement dans la gestion des données des decks et utilisateurs (**Yann**)

Responsive et modification de la structure du projet pour intégrer navbar (**Alexandre**)

Gestion et partage de decks (**Alexandre**)

Intégration de Google et Github pour l’authentification (**Alexandre**)

Vérification par l’adresse mail du compte (**Théo**)

Editeur complexe (**Jules**)

Itération 6

Dockerisation du projet (**Alexandre**)

Gestion de Decks (**Alexandre**)

Page de statistiques (**Yann**)

Pomodoro et mise en production (**Théo**)

Intégration de l’éditeur de texte (**Jules**)

Élément original du projet

Alexandre : Le point dont je suis le plus fier est d'avoir pu implémenter une intelligence artificielle pour la génération des cartes avec mon camarade Yann. Ce n'était pas évident lors de la première itération, on ne savait pas si ça allait fonctionner comme on souhaitait, qu'on ne perdait pas du temps pour rien.

Mais au final, nous avons réussi à implémenter un modèle assez puissant pour notre tâche bien qu'il soit limité en connaissance et en ressources, car nous avons constaté qu'on ne pouvait pas faire tourner de LLM sur nos PC portable personnels ni sur les PC de l'IUT, nous avons donc dû utiliser mon PC que j'ai chez moi et qui possède une carte graphique puissante pour faire tourner l'IA.

Bien que la génération ne soit pas forcément cohérente avec le contenu du PDF, je suis convaincu que si nous y passions plus de temps à peaufiner le modèle et qu'on avait de meilleures ressources, l'IA serait nettement meilleure.

J'ai pu apprendre beaucoup plus dans l'univers de l'intelligence artificielle notamment celui des Large-Language-Model, découvrir ses limites et comment l'implémenter dans un tel projet.

Je suis également fier de la mise en place de Docker sur notre projet puisque cela signifie que notre projet peut être utilisable par tous.

Yann : Tout comme Alexandre, je suis plutôt fier d'avoir pu intégrer une intelligence artificielle à notre application Mimir. Le fait de pouvoir tester différents modèles de LLM m'a aussi apporté beaucoup sur la compréhension de cette technologie en pleine émergence. Cela m'a cependant pris beaucoup de temps (une itération) à cause de mon manque d'expérience dans ce domaine mais aussi aux capacités de mon ordinateur personnel qui ne possède pas de carte graphique, ce qui ralentissait drastiquement l'exécution des modèles d'IA testées.

J'ai appris qu'il y avait plein de paramètres différents à définir pour une IA qui peuvent complètement changer son fonctionnement et les résultats de sortie.

Ce défaut technique nous a permis d'envisager des solutions, c'est-à-dire une machine assez puissante pour que le traitement de l'IA se fasse le plus rapidement possible, pour éviter à l'utilisateur d'attendre trop longtemps que l'IA donne une réponse.

Nous avons dû aussi penser à plein de détails pour optimiser au maximum la sortie un peu aléatoire que peut nous fournir l'IA, comme le format du prompt transmis et aussi le template de départ du modèle, qui permet de lui donner son comportement (voir Modelfile de la plateforme Ollama).

```
FROM ./vigogne.gguf

TEMPLATE """
Ton but est de générer les questions et leurs réponses par rapport à un
contexte donné. Le format sera toujours "Q : (question)   A : (réponse)"
pour chaque question/réponse. Génère le maximum de questions/réponses
possibles avec les informations données.Le contexte est :
### Instruction:

{{ .Prompt }}

### Response:
"""
```

Contenu du Modelfile

Théo : Pour ma part, ce projet a fait l'objet de beaucoup d'expériences. Pour commencer avec le côté programmation, la majorité des technologies utilisées m'étaient inconnues ou que vaguement abordées. Je suis donc agréablement surpris de la rapidité et de l'efficacité avec lesquelles, l'intégralité de mon groupe a su s'imprégner de ces nouveautés. De plus, les frameworks js (react) ou le langage Typescript sont en pleine expansion dans le domaine de la programmation web. La maîtrise de ces technologies est donc un grand avantage pour notre carrière future.

Je tiens également à souligner l'importance de l'architecture de l'application. L'utilisation du framework NextJs permet de déléguer par exemple le routing de l'application, celle-ci se gère automatiquement selon l'arborescence des fichiers. Chaque dossier correspond à une page de l'application. Ce système permet notamment de gérer les appels à la base de données de façon similaire à une page de l'interface web.

Tout ce nouvel environnement de travail force à l'adaptation, la veille technologique et la communication au sein de l'équipe. Cela correspond finalement à tout ce qui est requis lors d'un travail en entreprise.

Jules : La partie qui m'a le plus marqué durant le développement de l'application Mimir fût la gestion autour de projet. Pas seulement l'organisation et la répartition des tâches , mais aussi et surtout la gestion des différents supports, que ce soit nos dépendances à travers la recherche et l'apprentissage de librairies utilisées par les professionnels du secteur. J'ai eu l'occasion de me rendre compte que des bases solides et un investissement majeur de mon temps dans le setup du projet ont permis d'avoir une certaine vélocité dès les premières itérations.

Compte rendu entre attente et finalité

Un projet aussi conséquent que le projet tutoré implique très souvent une bonne gestion du temps et des tâches à réaliser. Dans notre cas, c'était notre principal ennemi et il a décimé plusieurs de nos fonctionnalités phares.

Sur le plan social, mimir est une application servant aux étudiants à apprendre leurs cours sous forme de quiz. L'une de nos idées était donc de pouvoir créer des groupes et ainsi limiter la portée d'un deck uniquement à un groupe d'amis ou de classe. Cette fonction intéressante pour toutes les écoles et universités a donc été abandonnée suite à l'implémentation d'une autre fonctionnalité en lien : création de deck privé ou publique. Afin de rester dans le domaine de l'éducation, le but secondaire de l'application était d'apprendre à apprendre.

Cela ne se fait pas aussi naturellement que l'on pourrait le penser et c'est pourquoi il était dans la continuité du projet d'ajouter à l'application toute une documentation décrivant les principales méthodologies d'apprentissage. Bien que celle du pomodoro (chronomètre permettant d'alterner entre des phases de travail et de repos) ait été implémentée, aucune autre n'a pu être explicitement détaillée au sein du projet. Le système de leitner et l'idée de Roadmap entre les decks ont tout de même une place réservée pour une implémentation tardive.

Pour le dernier point concernant les fonctionnalités non délivrées, il ne se justifie pas par un manque de temps mais plus par une mauvaise estimation de son importance dans le projet. Pour les questions l'idée était simple, pouvoir y mettre tout et n'importe quoi, que ce soit des images, des formules, du texte stylisé ou encore des vidéos, il ne fallait pas limiter l'utilisateur à sa question. En parallèle, la réponse devait contenir la même forme de données malgré les différents types de réponses permettant ainsi de rendre l'apprentissage plus ludique avec la présence de questionnaire à choix multiple. Malheureusement, ces fonctionnalités n'ont pas pu rejoindre celles déjà en place telles que vérification de la réponse par Intelligence artificielle, auto-évaluation et réponse formelle sous forme écrite.

Conclusion

Notre application repose sur plusieurs principes : le partage entre utilisateurs, la liberté de création et d'utilisation du travail des autres utilisateurs, l'apprentissage et le suivi de son apprentissage.

Nous avons pu mettre sur papier ces aspects dans les fonctionnalités évoqués lors de l'étude préalable ainsi que dans les différents diagrammes réalisés. Nous avions pu déterminer que cela ne serait pas si simple à mettre en place entre l'apprentissage de nouvelles technologies et le peu de temps disponible par rapport au projet envisagé.

Tout au long du développement de notre projet, nous nous sommes rendu compte que certains points dont nous pensions que la charge de travail nécessaire à leur réalisation serait minime, étaient en fait bien plus chronophages que nous le pensions, alors que d'autres fonctionnalités furent bien plus rapides à mettre en place. Certaines tâches ont, au contraire, pris le temps prévu pour être développées.

Précédemment, nous avons détaillé l'écart entre l'application que nous voulions créer et celle que nous avons à ce dernier stade du projet tutoré. Bien que déçu de ne pas avoir pu

tout implémenter, nous sommes fiers de notre application et surtout conscients des erreurs commises allant du simple manque de communication au non-respect des principes fondamentaux de la programmation SOLID que nous avons vu en cours.

La consécration de plus de 200 heures de travail sur ce projet nous a permis de nous exercer et de nous auto-évaluer quant à notre capacité de travail sur un projet beaucoup plus volumineux et complexe que ce dont nous avons l'habitude. C'est une expérience que nous sommes fiers d'avoir mené à son terme et qui sera peut-être, le brouillon d'une future application d'auto-apprentissage nationale !

Important : Tout le détail lié à l'installation et à l'hébergement du projet est indiqué dans le readMe Github. (https://github.com/wartt88/Mimir).