

Session d'hiver 2024

MainParlé

Document d'analyse



Auteurs (équipe 1) :

Adam Brichi ; Adam Jihad Ibrahim ; Selmane Saïb
Merabet; Martial Romarik Poupale

Destiné à :

Raouf Babari



Institution :

Collège de Bois-de-Boulogne

Cours :

420-204-RE PROJET D'INTÉGRATION EN SCIENCES
INFORMATIQUES ET MATHÉMATIQUES groupe
00003

Table des matières

Présentation du projet	4
L'idée	4
L'utilité.....	4
L'innovation.....	4
Public ciblé	4
Secteur de la santé.....	4
Secteur public.....	4
Secteur de l'éducation	4
Agences médiatiques	4
Ressources humaines.....	4
Technologies	5
Next.js 	5
Node.js  (+ NPM ).....	5
React 	5
PostCSS  (+ Autoprefixer )	5
Fonctionnalités Next.js.....	5
Tailwind CSS 	7
TypeScript 	7
ESLint 	7
Mantine 	7
MediaPipe  (+ TensorFlow )	8
MongoDB 	8
Vercel 	8
Diagrammes	9
Diagramme de Gantt.....	9
Diagramme des classes	9
Diagramme de la base de données MongoDB.....	10
Cas d'utilisation	Erreur ! Signet non défini.
Cas d'utilisation pour tous les utilisateurs	Erreur ! Signet non défini.
Cas d'utilisation pour les élèves.....	Erreur ! Signet non défini.
Cas d'utilisation pour les administrateurs (enseignants) (quels enseignants ?)	Erreur ! Signet non défini.

Annexe.....	10
Annexe 1 – Configuration du projet la première fois	16
Configuration de Next.js	16
Installation de dépendances	16
Annexe 2 – Exécution du projet	17
Annexe 3 – Installation d’extensions Visual Studio Code	17
Annexe 4 – Entraînement du modèle IA	18
Annexe 5 – Déploiement sur Vercel.....	18
Ajouter un domaine	18
Ajouter des variables d’environnement	18
Annexe 6 – Intégration de MongoDB avec Vercel et Next.js.....	18
Annexe 7 – Fonctionnement de Prisma	18
Installation de Prisma.....	18
Utiliser Prisma avec Next.js	19
Commandes utiles.....	19
Relations.....	19
Extensions	20
Annexe 8 – Authentification avec Clerk	20
Annexe 9 – Connexion entre Clerk et MongoDB	22

Présentation du projet

L'idée

Le concept du projet s'agit de créer un site web interactif et amusant dont le but serait l'apprentissage de la langue des signes américaine (ASL). Cela se fera en utilisant un modèle d'IA qu'on entrainera et qui est capable de reconnaître et valider les signes de main qu'effectue le client à partir de sa caméra.

L'utilité

L'utilité de ce projet est d'offrir à nos clients une manière efficace d'apprendre le langage des signes.

L'innovation

L'utilisation d'une interface comportant la caméra de l'utilisateur, couplée à un modèle d'intelligence artificielle, fait en sorte que l'apprentissage s'effectue d'une manière plus « pratique » que seulement « théorique ». En effet, une des meilleures manières pour apprendre une nouvelle langue est de pratiquer en ayant des conversations avec quelqu'un qui parle déjà cette langue (dans ce cas, notre modèle d'intelligence artificielle entraîné pour comprendre le langage des signes).

Public ciblé

Secteur de la santé

Les professionnels de la santé bénéficieront de notre site web d'apprentissage de l'ASL en améliorant leur capacité à communiquer avec les patients malentendants. Cela se traduira par des interactions plus efficaces et empathiques, contribuant ainsi à une meilleure qualité de soins et à une expérience positive pour les patients.

Secteur public

Les services publics seront améliorés grâce à notre site web d'apprentissage de l'ASL, permettant aux fonctionnaires de mieux communiquer avec les citoyens malentendants. Cette initiative favorisera une participation citoyenne accrue et une société plus inclusive pour tous.

Secteur de l'éducation

Les enseignants et les élèves bénéficieront de notre site web d'apprentissage de l'ASL en favorisant l'inclusion en classe. En apprenant l'ASL, les enseignants pourront mieux communiquer avec les élèves malentendants, créant ainsi un environnement éducatif plus inclusif et propice à l'apprentissage pour tous les élèves.

Agences médiatiques

Les agences médiatiques pourront améliorer leur communication avec les auditoires malentendants en utilisant notre site web d'apprentissage de l'ASL. Cela permettra une représentation plus précise et une diffusion d'informations plus inclusive, contribuant ainsi à une société plus diversifiée et équitable.

Ressources humaines

Les professionnels des ressources humaines pourront favoriser l'inclusion en milieu de travail en apprenant l'ASL. Cela leur permettra de mieux communiquer avec les employés malentendants, créant ainsi un environnement de travail plus inclusif et diversifié pour tous les employés.

Technologies

Voici les différentes technologies que nous avons choisi d'apprendre et d'adopter pour notre projet :

Next.js

Next.js est un framework *full-stack* qui permet de développer des applications web tout en offrant une meilleure expérience développeur et client. Il utilise le langage de programmation JavaScript en se basant sur la bibliothèque React pour le frontend et sur la technologie Node.js pour le backend.

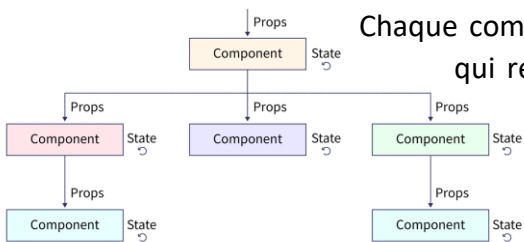
En utilisant l'outil *create-next-app* (voir [CONFIGURATION DE Next.js](#)), on peut créer en moins d'une minute le squelette d'une application incluant toutes les technologies suivantes :

Node.js (+ NPM)

Node.js est une plateforme de développement qui permet d'exécuter du JavaScript en *backend* dans un serveur. Elle est équipée de NPM, un gestionnaire de paquets qui permet de gérer des dépendances du projet, c'est-à-dire qu'il s'occupe de l'installation de tous les autres modules JavaScript dont notre code a besoin (ex. : Next.js, Mantine, MediaPipe, ainsi que leurs dépendances respectives).

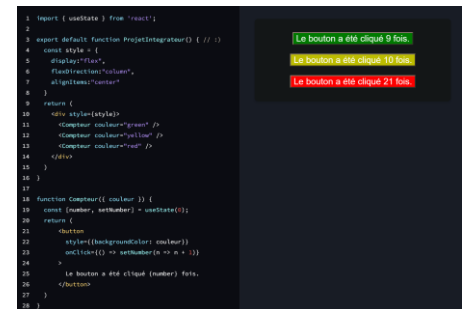
React

React est une bibliothèque JavaScript *frontend* qui permet de créer facilement des interfaces interactives. Elle permet de développer une application web sous forme d'un arbre de composants réutilisables, qui comprennent chacun des données qui peuvent changer avec le temps.



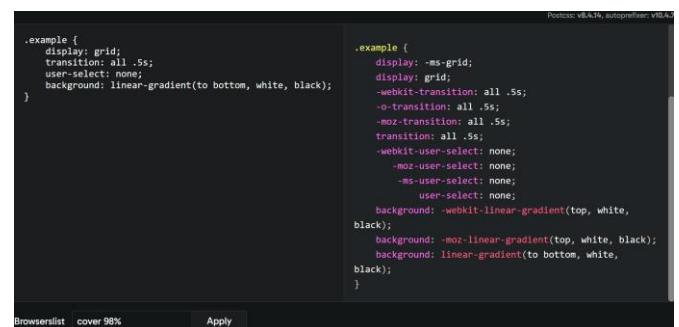
Chaque composant est une méthode JavaScript qui reçoit des données du parent et qui

retourne un élément ou un regroupement d'éléments HTML. Créer un composant permet d'extraire et d'encapsuler une fonctionnalité, évitant de tout mettre dans le même fichier HTML tout en rendant le code plus lisible et moins répétitif.



PostCSS (+ Autoprefixer)

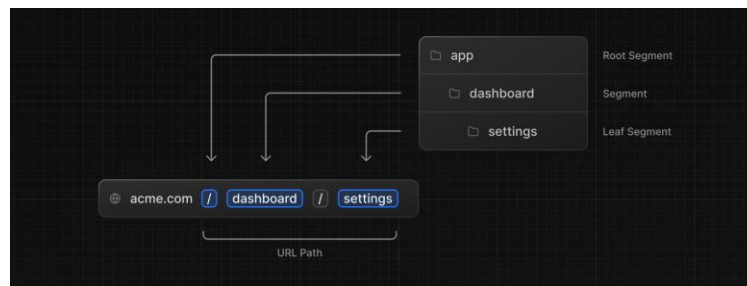
PostCSS est un outil de développement qui permet d'ajouter des *plugins* JavaScript pour automatiser des opérations CSS. Par exemple, les navigateurs web utilisent souvent des préfixes pour donner accès à des fonctionnalités CSS qui ne sont pas encore considérées comme stables ; le plugin Autoprefixer les ajoute automatiquement lorsque le code est compilé à partir des données du site caniuse.com.



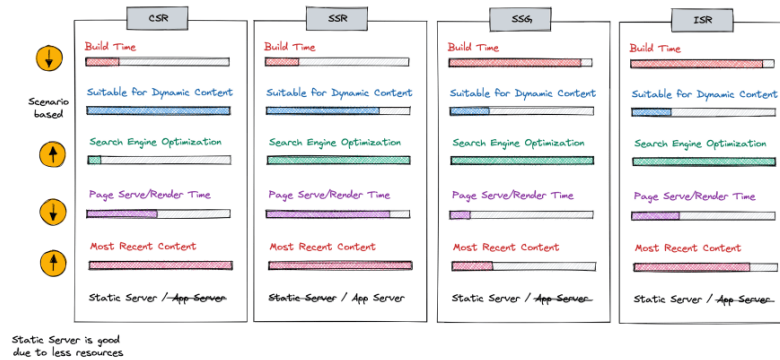
Fonctionnalités Next.js

Next.js est fondé sur les technologies susmentionnées, mais apporte également beaucoup de fonctionnalités utiles qui facilitent le travail du développeur :

- un système de routage optimisé et intuitif



- différentes techniques de rendu (options flexibles pour déterminer pour chaque composante ce qui sera exécuté en frontend ou en backend)



- des Server Actions (méthodes qui sont exécutées seulement en backend)
- la création d'un API
- des composantes pour optimiser le chargement (liens, images, polices, scripts)
- etc.

Built-in Optimizations
Automatic Image, Font, and Script Optimizations for improved UX and Core Web Vitals.

Dynamic HTML Streaming
Instantly stream UI from the server, integrated with the App Router and React Suspense.

React Server Components
Add components without sending additional client-side JavaScript. Built on the latest React features.

Data Fetching
Make your React component async and await your data. Next.js supports both server and client data fetching.

CSS Support
Style your application with your favorite tools, including support for CSS Modules, Tailwind CSS, and popular community libraries.

Client and Server Rendering
Flexible rendering and caching options, including Incremental Static Regeneration (ISR), on a per-page level.

Server Actions
Run server code by calling a function. Skip the API. Then, easily invalidate cached data and update your UI in one network roundtrip.

Route Handlers
Build API endpoints to securely connect with third-party services for handling auth or listening for webhooks.

Next.js 14
The power of full-stack to the frontend. Read the release notes.

Advanced Routing & Nested Layouts
Create routes using the file system, including support for more advanced routing patterns and UI layouts.

Middleware
Take control of the incoming request. Use code to define routing and access rules for authentication, experimentation, and internationalization.

Tailwind CSS

Tailwind CSS est un framework qui permet d'écrire du CSS directement dans le HTML à l'aide de classes utilitaires. Cela permet non seulement d'écrire moins de code, mais aussi d'éviter de trouver des noms de variables (id et classes CSS) et d'avoir à sauter d'un fichier à l'autre. On pourrait croire que ne pas créer de variables rendrait le code répétitif, mais avec un framework ou une bibliothèque comme React, où le code est divisé en composants réutilisables, le code n'a besoin d'être écrit qu'une seule fois. Et même si on choisit de créer des variables, il y a moins de code à écrire.

With Tailwind

Code:

```
index.html
<button class="bg-blue-500 text-white py-2 px-4 rounded">
  Primary
</button>
```

Output:

Primary

Without Tailwind

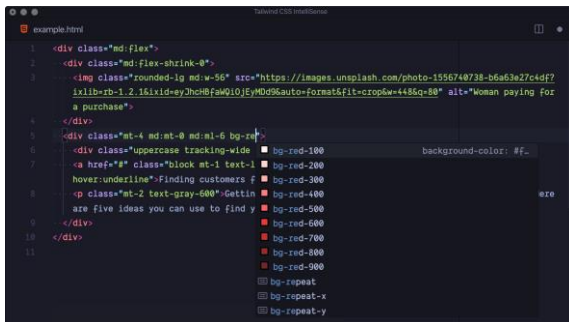
Code:

```
index.html
<button class="button">Primary</button>
```

```
style.css
.button {
  background-color: #4285f4;
  color: #fff;
  border-radius: 0.25rem;
  padding-top: 0.5rem;
  padding-bottom: 0.5rem;
  padding-left: 0.75rem;
  padding-right: 0.75rem;
}
```

Output:

Primary



TypeScript

TypeScript est une extension du langage JavaScript servant à améliorer et à sécuriser la production de code. Il permet de transformer JavaScript, un langage dynamiquement typé, en un langage statiquement typé. En d'autres termes, plutôt que de déterminer les types des variables (number, string, array, object, etc.) pendant l'exécution du code, ceux-ci seront déterminés pendant la compilation. Ainsi, même si ça peut demander plus de code, puisqu'il faut souvent écrire explicitement le type des variables, TypeScript permet de prévenir des erreurs et d'offrir de la complétion automatique.

```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor",
}

console.log(user.name)

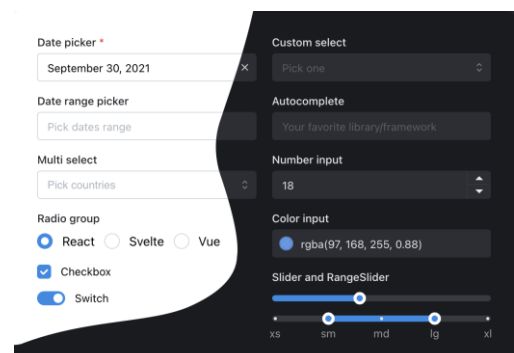
Property 'name' does not exist on type '{ firstName: string; lastName: string; role: string; }'.
```

ESLint

ESLint est un outil qui permet d'analyser statiquement le code pour trouver des problèmes et proposer automatiquement des solutions, ou pour faire respecter des normes préconfigurées. La plupart des problèmes sont déjà détectés par TypeScript, mais ça rajoute une couche de vérifications supplémentaires.

Mantine

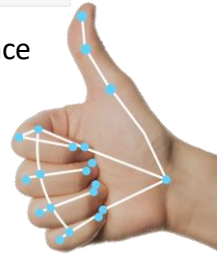
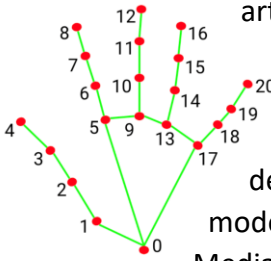
Mantine est une bibliothèque de composants React. Il offre une centaine de composants fonctionnelles et stylisées qui permettent de créer rapidement et facilement une interface élégante sans avoir à tout créer à partir de zéro en CSS et en JavaScript. Elle offre aussi des React hooks qui permettent d'accéder facilement à des fonctions JavaScript.



MediaPipe 🍷 (+ TensorFlow 🏠)

Thumbs up 63%

MediaPipe est une suite de bibliothèques et d'outils permettant d'appliquer et de personnaliser rapidement et facilement des techniques d'intelligence artificielle (IA) et d'apprentissage automatique (ML) dans son application. Dans notre cas, nous utiliserons le [Gesture recognition task](#) pour la reconnaissance des signes de la main ; MediaPipe se chargera de lire la webcam de l'utilisateur, de détecter la main, de la dessiner, puis de reconnaître un signe selon le modèle ML fourni. Le modèle par défaut est capable de reconnaître 8 gestes, mais l'outil MediaPipe Model Maker permet d'entraîner son propre modèle personnalisé à l'aide de TensorFlow, un outil d'apprentissage automatique qui permet de créer des modèles. (Voir [ANNEXE 4 – Entraînement du modèle IA](#))



MongoDB 🍇 (+ Prisma 🏠)

MongoDB est une base de données NoSQL, c'est-à-dire un système de stockage de données semi-structurées offrant une structure flexible. C'est ce qui va nous permettre de stocker les informations relatives aux utilisateurs et aux leçons.

Prisma est un ORM, c'est-à-dire un programme qui permet de manipuler la structure et les données d'une base de données de manière orientée objet. Prisma permet d'imposer une structure précise aux données stockées sur MongoDB dans un « schema », puis de générer automatiquement un « Prisma Client », qui permet d'interagir facilement avec la base de données (à l'aide de la complétion automatique TypeScript). (Voir [COMMANDES UTILES](#))

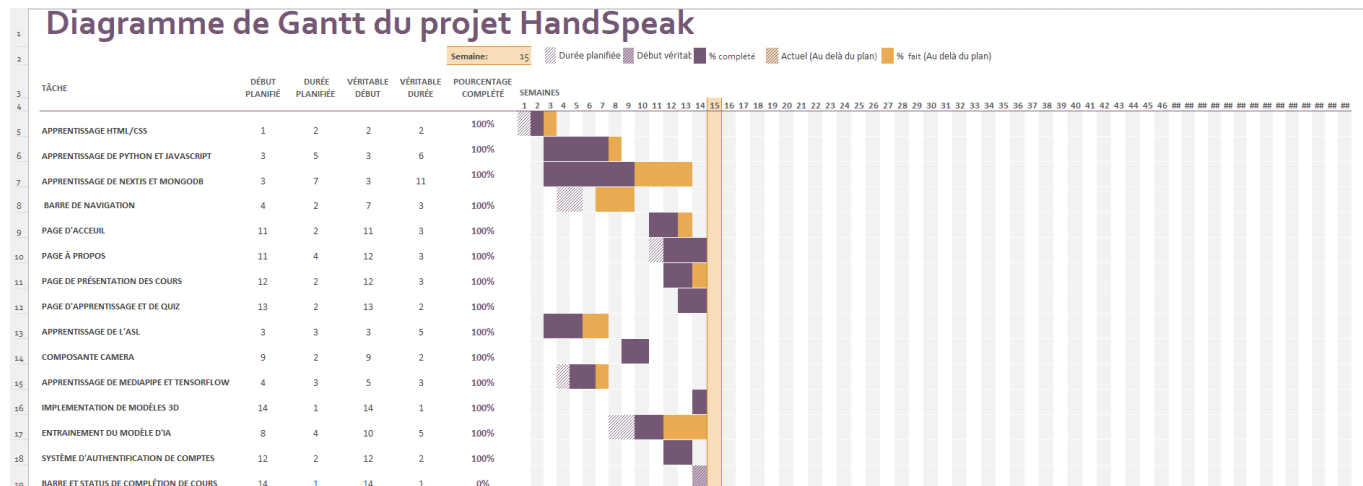
Vercel 🏠

Vercel est une plateforme d'hébergement et de déploiement de sites web.

Diagrammes

Diagramme de Gantt (échancier)

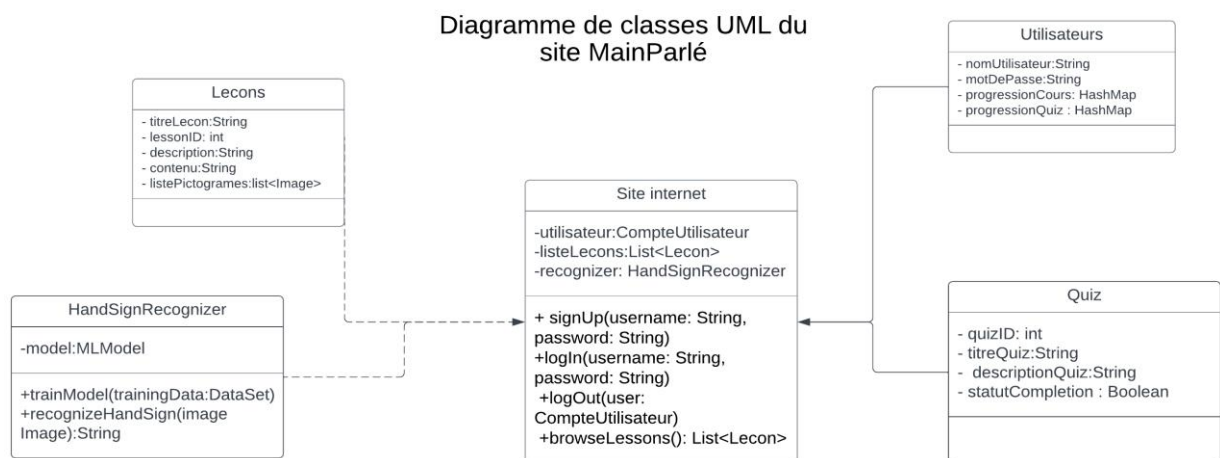
Le diagramme de Gantt, couramment utilisé en gestion de projet, est l'un des outils les plus efficaces pour représenter visuellement l'état d'avancement des différentes activités (tâches) qui constituent un projet. La colonne de gauche du diagramme énumère toutes les tâches à effectuer, tandis que la ligne d'en-tête représente les unités de temps les plus adaptées au projet (jours, semaines, mois, etc.). Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin.



bdebgcca-my.sharepoint.com/:x/g/personal/2276775_bdeb_qc_ca/EbBorT52ulZKviPKUwP3JeQB_np5l6jmiLqg04E-gJCBUw?e=2FNTkz

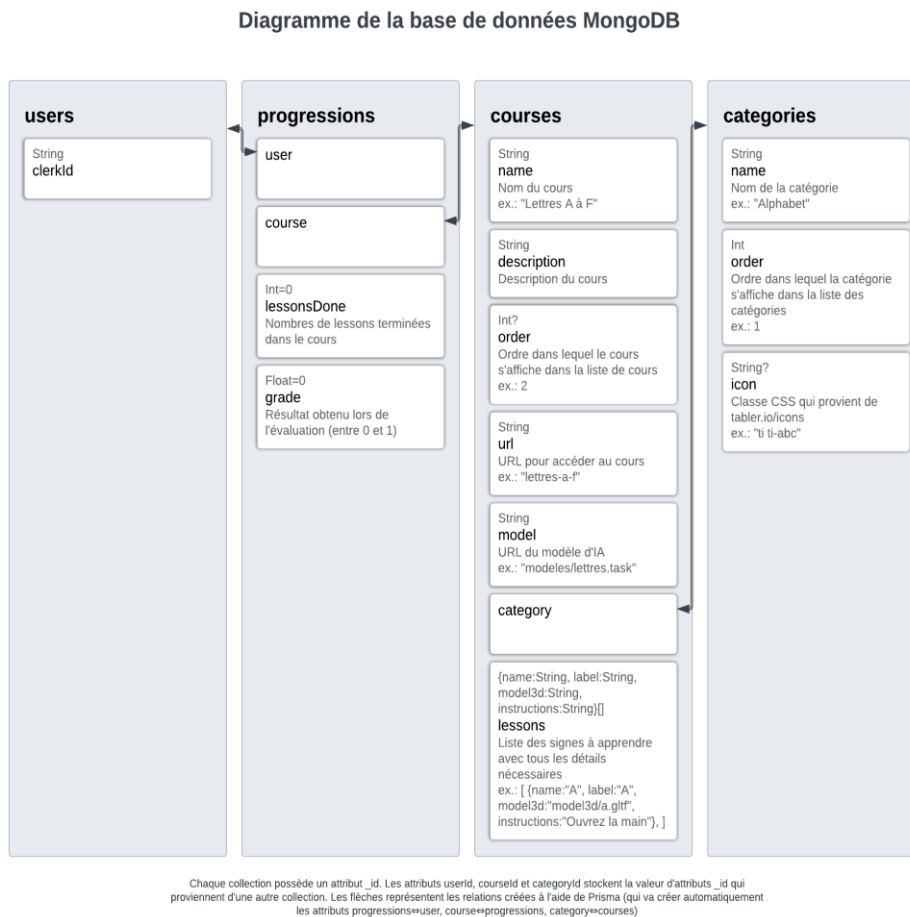
Diagramme des classes (UML)

Les diagrammes de classes sont l'un des types de diagrammes UML les plus utiles, car ils décrivent clairement la structure d'un système particulier en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets.



lucid.app/documents/embedded/28df3f1a-22be-416b-8799-9cfffab82d79

Diagramme de la base de données MongoDB



lucid.app/documents/embedded/7963b9fe-0ae4-4a83-9496-b83c8d82708d

Cas d'utilisation

Un cas d'utilisation représente une représentation d'une interaction entre un utilisateur du service et un système. Ainsi, dans un diagramme de cas d'utilisation, les utilisateurs sont appelés acteurs, et ils apparaissent dans les cas d'utilisation.

Cas d'utilisation pour tous les utilisateurs

- Inscription, connexion et suppression du compte
- Consultation du catalogue des cours offerts
- Consultation des pages sur la barre de navigation (pages à propos, progression, informations sur le compte, etc.)

Cas d'utilisation pour les élèves

- Peuvent faire tout ce que les utilisateurs peuvent faire
- Peuvent participer à l'apprentissage des cours
- Peuvent compléter les quiz afin de passer à la prochaine leçon
- Peuvent sauvegarder leur progression à chaque cours (cela se fait automatiquement)

Preuves d'exécution du projet

Cette section traitera la documentation liée au développement et à l'exécution du projet. Les preuves d'exécution sont démontrées à travers des vidéos, des captures d'écrans ainsi que des liens.

Site

Le lien suivant a été créé à la suite du déploiement du projet grâce à Vercel : handspeak.vercel.app

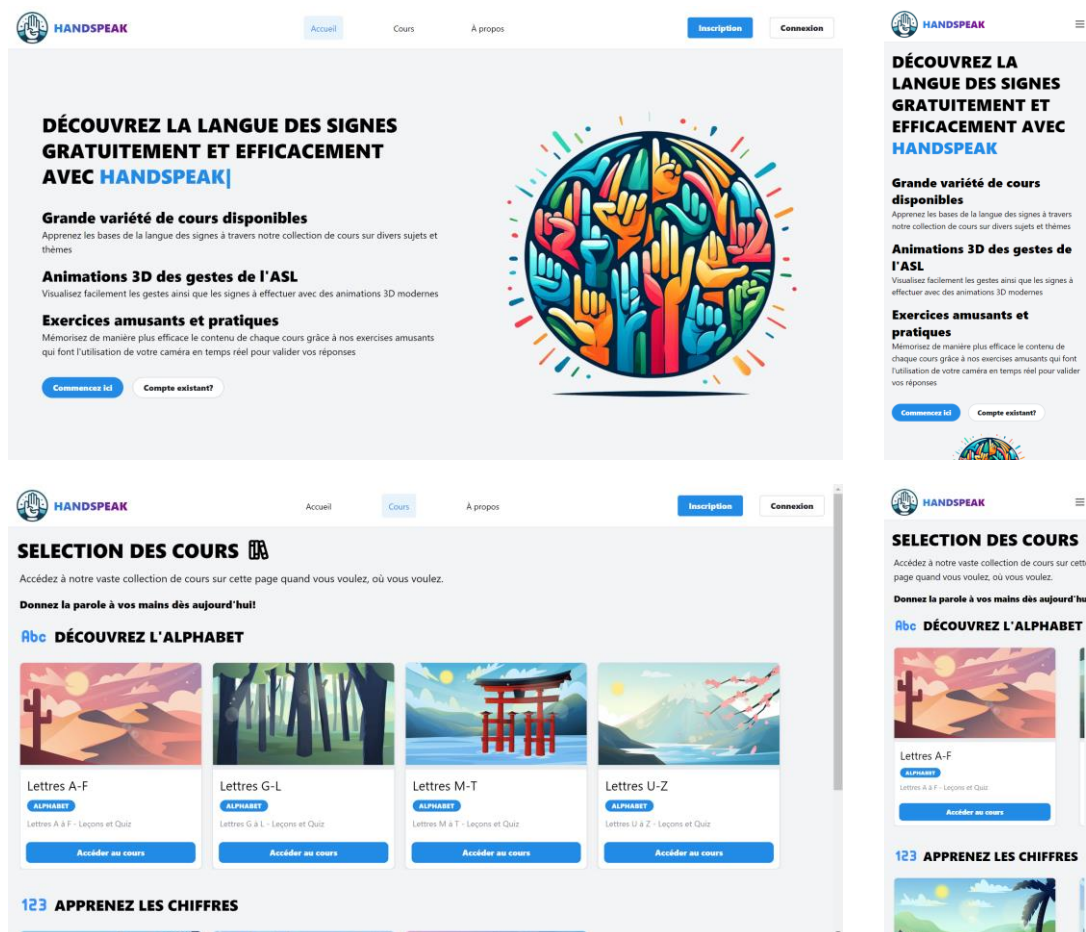
Dépôt GitHub

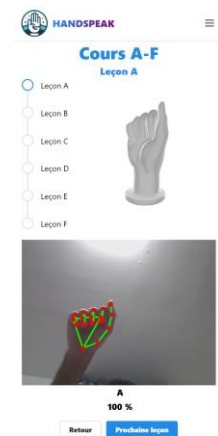
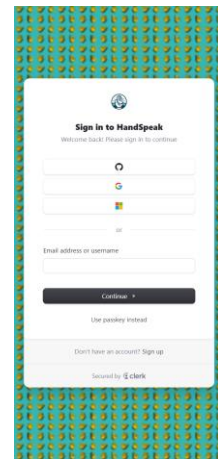
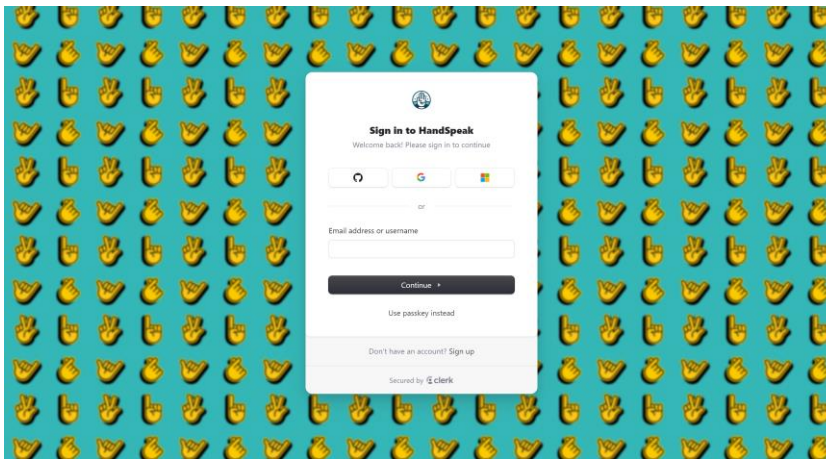
Le dépôt GitHub ci-dessous contient des informations sur les diverses branches créées au cours du développement du projet. Les nombreux « commits » effectués par les membres de l'équipe témoignent de l'exécution du projet. Des captures d'écrans ainsi que des vidéos des versions antérieures du site sont disponibles dans les commentaires laissés par les développeurs.

github.com/ADecametre/H24-204-GR1-HandSpeak

Captures d'écran

Comme il l'a été mentionné précédemment, de nombreuses captures d'écrans et vidéos sont affichées dans les onglets commentaires des « commits » dans le dépôt GitHub. Cette section contiendra quelques exemples de ces captures d'écrans comme preuves d'exécution du projet.





Vidéos

NOTE : UNE VIDÉO QUI COUVRE TOUTE LA VERSION FINALE DU PROJET SERA ENVOYÉE SUR TEAMS LE 21 MAI.

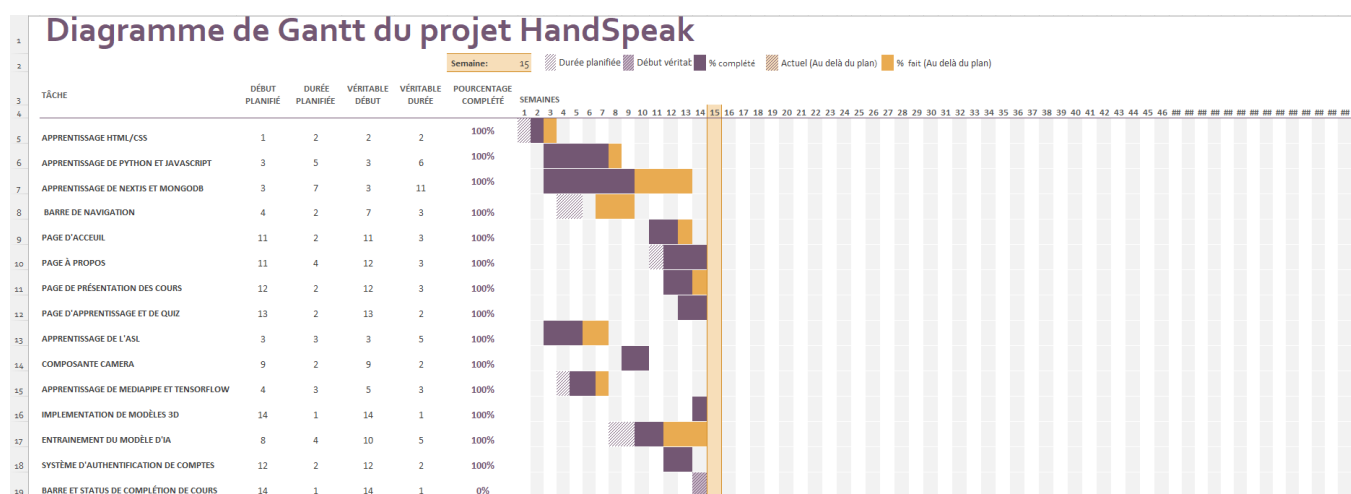


Difficultés rencontrées

Les principales difficultés rencontrées par l'équipe de développement étaient en lien avec la gestion de temps et l'apprentissage d'une multitude de nouvelles technologies en un intervalle de temps très restreint. En effet, la découverte de plus d'une dizaine de technologies et de langues de programmation différentes (HTML, CSS, Python, Javascript, Typescript, etc.) en moins de quatre mois aurait entraîné plusieurs retards dans les dates des objectifs de l'échéancier. Malgré cela, la majorité des sous-projets de l'échéancier ont été complétés à temps avant la remise finale du projet.

Retards mineurs dans l'échéancier

Les cases orange représentent les retards, mais comme l'indique le diagramme, tout a été complété avant la remise finale du projet.



Ressources utilisées pour l'apprentissage de la langue des signes

L'apprentissage de la langue des signes fut un obstacle plutôt mineur. Plusieurs ressources disponibles gratuitement en ligne ont été utilisées comme matériel éducatif dans le but de familiariser l'équipe de développement avec la langue des signes.

https://www.youtube.com/watch?v=6_gXiBe9y9A&ab_channel=OurBergLife

https://www.youtube.com/watch?v=hFCXyB6q2nU&ab_channel=ASLTHAT

https://www.youtube.com/watch?v=Y4stD_yPaAI&ab_channel=LearnHowtoSign

Perspectives d'amélioration

Cette section traitera les perspectives et les possibilités d'améliorations envisageables si l'intervalle de temps disponible pour le développement du projet était doublé.

Ajout de cours plus avancés

Le projet aurait pu être amélioré par l'addition de cours beaucoup plus rigoureux qui traitent des sujets plus avancés. Par exemple, l'ajout de cours pour des étudiants avec un niveau de connaissances plus élevé dans la langue des signes aurait été intéressant. Cela est un changement qui nécessitera plus de temps afin d'entraîner le modèle d'IA, mais il peut quand même être considéré. De plus, il serait aussi possible d'intégrer des vidéos de conférences, des livres électroniques, et des articles académiques pour approfondir les connaissances de nos utilisateurs.

Adaptation du site internet pour des institutions académiques

Une autre amélioration envisageable serait d'adapter le logiciel à des salles de classe en ajoutant des fonctionnalités pour les professeurs. Ces derniers pourront créer des groupes d'enseignements avec leurs élèves afin de superviser attentivement leur apprentissage. Cela pourrait se faire en leur donnant la capacité de créer des cours spécialisés aux besoins de leurs élèves tout en leur donnant accès à la progression ainsi qu'à la performance de ceux-ci. Il serait aussi une bonne idée de donner la capacité aux enseignants de pouvoir organiser des sessions en direct avec leurs élèves pour des interactions en temps réel.

Accessibilité pour une plus grande clientèle

Le site internet pourrait bénéficier de plusieurs fonctionnalités afin de rendre le contenu accessible à une clientèle plus large. L'ajout d'un sélecteur de diverses langues (français, anglais, espagnol, etc.) pour l'affichage du site rendrait celui-ci plus accessible pour une clientèle internationale. Cela ferait en sorte que notre base d'utilisateurs ne serait pas seulement limitée à des régions francophones dans le monde. De plus, le site pourrait être adapté pour faciliter la navigation des utilisateurs malvoyants en employant des lecteurs d'écran auditif ainsi que des boutons de type « Text-to-Speech » afin de lire le contenu de la page à voix haute. Le site pourrait aussi bénéficier d'un mode de couleurs daltoniennes afin de rendre le site plus accessible pour ces utilisateurs.

Conclusion

Pour conclure, le logiciel HandSpeak sert de site web pour l'apprentissage gratuit et efficace de la langue des signes. L'implémentation de modèles à trois dimensions et d'un détecteur de mouvements et de gestes à partir d'un modèle d'intelligence artificielle rend l'apprentissage beaucoup plus immersif. Le développement du site web HandSpeak serait donc un succès puisque la grande majorité des attentes de l'échéancier ont été satisfaites à temps.

Annexe

Annexe 1 – Configuration du projet la première fois

1. Si ce n'est pas déjà fait, installer Node.js et NPM (nodejs.org/en/download).
2. Si ce n'est pas fait, installer l'environnement de développement intégré Visual Studio Code (code.visualstudio.com/download).
3. Cloner localement un dépôt GitHub (en se servant par exemple de GitHub Desktop : desktop.github.com).
4. Ouvrir le dossier cloné avec Visual Studio Code.

Configuration de Next.js

NOTE : CES ÉTAPES NE DOIVENT ÊTRE APPLIQUÉES QU'UNE SEULE FOIS ET PAR UN SEUL MEMBRE DE L'ÉQUIPE.

5. Ouvrir le terminal.
6. Installer l'outil *create-next-app* avec NPM en tapant la commande *npm i create-next-app*.
7. Exécuter *npm create-next-app@latest*, puis sélectionner les options souhaitées.

```
PS C:\Users\b-ada\Documents\GitHub\H24-204-GR1-HandSpeak> npm create-next-app@latest
? What is your project named? ... handspeak
? Would you like to use TypeScript? ... No / Yes
? Would you like to use ESLint? ... No / Yes
? Would you like to use Tailwind CSS? ... No / Yes
? Would you like to use 'src/' directory? ... No / Yes
? Would you like to use App Router? (recommended) ... No / Yes
? Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\Users\b-ada\Documents\GitHub\H24-204-GR1-HandSpeak\handspeak.

Using npm.

Initializing project with template: app-tw

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- autoprefixer
- postcss
- tailwindcss
- eslint
- eslint-config-next

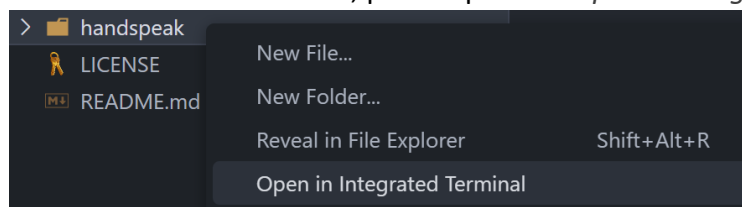
added 366 packages, and audited 367 packages in 18s

132 packages are looking for funding
  run npm fund for details

found 0 vulnerabilities
Success! Created handspeak at C:\Users\b-ada\Documents\GitHub\H24-204-GR1-HandSpeak\handspeak
```

Installation de dépendances

8. Ouvrir le terminal dans le dossier du projet (*handspeak*) ; ça peut se faire de deux façons :
 - a. Faire un clic droit sur le dossier, puis cliquer sur *Open in Integrated Terminal* :



- b. Ouvrir le terminal, puis exécuter *cd handspeak* pour changer de dossier :

```
\GitHub\H24-204-GR1-HandSpeak> cd handspeak
\GitHub\H24-204-GR1-HandSpeak\handspeak> |
```

9. Installer les dépendances nécessaires avec NPM (voir **TECHNOLOGIES** pour comprendre l'utilité de chacune)
 - a. *npm i @mediapipe/tasks-vision*
 - b. *npm i @mantine/core*

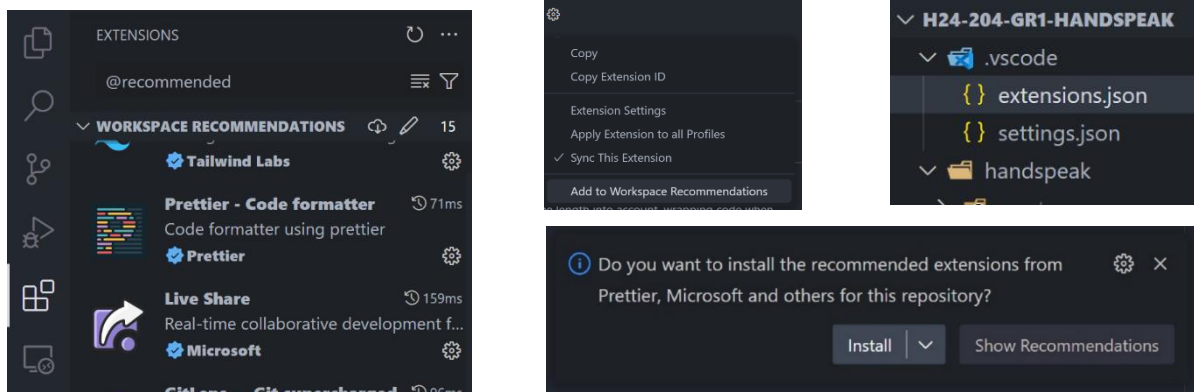
Annexe 2 – Exécution du projet

1. Si ce n'est pas déjà fait, installer les logiciels requis en suivant les étapes 1, 2, 3 de l'Annexe 1 – Configuration du projet la première fois.
2. Ouvrir le projet et le terminal en suivant les étapes 4 et 8 de l'Annexe 1 – Configuration du projet la première fois.
3. S'il manque des dépendances au projet (ex. : s'il y a une vingtaine d'erreurs dans le projet), exécuter `npm i`.
 - Toutes les dépendances sont stockées localement dans le dossier `node_modules`. Ce dossier n'est pas synchronisé avec GitHub ; sinon, il y aurait des centaines de fichiers à téléverser. NPM se charge donc de tout le stockage et seuls deux fichiers sont synchronisés avec GitHub : `package.json` et `package-lock.json`, qui contiennent la liste des dépendances nécessaires.
 - Le fichier `.gitignore` contient d'ailleurs la liste des dossiers et fichiers qui ne doivent pas être stockés sur GitHub parce qu'ils sont générés localement ou parce qu'ils contiennent des secrets (variables d'environnement `.env.local`) qui ne doivent pas être publiés ; ceux-ci sont affichés en gris par Visual Studio Code.
4. Compiler le projet
 - a. en tant que développeur : exécuter `npm run dev`.
 - Cela permet d'exécuter rapidement le projet pour avoir un aperçu non optimisé.
 - Le site sera automatiquement mis à jour pour refléter les modifications du code.
 - b. en tant que client : exécuter `npm run build`, puis `npm start`.
 - Cela permet de compiler la version finale du site auquel accédera le client.
 - Ça prendra beaucoup de temps à compiler, mais le site sera optimisé et fluide.
5. Ouvrir la page localhost:3000 sur un navigateur.

Annexe 3 – Installation d'extensions Visual Studio Code

L'environnement de développement intégré Visual Studio Code offre un *Marketplace* d'extensions. Celles-ci permettent de rajouter toutes sortes de fonctionnalités supplémentaires au logiciel, que ce soit pour du formatage de code, des raccourcis clavier, des suggestions de code, de la saisie automatique, des outils de collaboration, des thèmes, etc.

Afin que tous les membres de l'équipe aient accès aux mêmes extensions, il est possible de créer des Workspace Recommendations. Cela crée un fichier `extensions.json`, qui fournit une liste d'extensions aux autres développeurs. Pour les installer, ils n'ont qu'à cliquer sur *Install* lorsque un message apparaît en bas de l'écran ou à taper `@recommended` dans le volet *Extensions*.



Annexe 4 – Entraînement du modèle IA

Pour entraîner le modèle, nous avons utilisé ce [guide](#) qui nous permet d’entraîner une IA en utilisant les technologies MediaPipe et TensorFlow (voir [MEDIAPIPE](#) 🏗️ (+ [TENSORFLOW](#) 🏗️)).

Voici donc les étapes que nous avons entreprises :

1. Créer des données (images de nos mains qui font les signes) grâce à la webcam.
2. Entraîner le modèle avec ces données.
3. Tester et améliorer la performance du modèle.
4. Télécharger le modèle pour l’intégrer dans notre projet.

Annexe 5 – Déploiement sur Vercel

1. Créer un nouveau projet sur Vercel.
2. Connecter son compte GitHub à Vercel.
3. Sélectionner le dépôt GitHub.
4. Sélectionner le dossier du projet Next.js.
5. Déployer.

Ajouter un domaine

1. Ouvrir les paramètres du projet.
2. Cliquer sur *Domains*.
3. Ajouter un domaine qui se termine par [.vercel.app](#).

Ajouter des variables d’environnement

1. Ouvrir les paramètres du projet.
2. Cliquer sur *Environment Variables*.
3. Copier-coller le contenu du fichier *.env.local*.

Annexe 6 – Intégration de MongoDB avec Vercel et Next.js

1. Ouvrir la *Team* dans lequel se trouve le projet.
2. Cliquer sur l’onglet *Integrations*.
3. Ajouter l’intégration *MongoDB Atlas* et connecter son compte MongoDB à Vercel.
4. Donner accès au projet à MongoDB.

5. Ouvrir les paramètres du projet, puis cliquer sur *Environment Variables*.
6. Copier la variable `MONGODB_URI`, qui a été créée automatiquement, dans le fichier `.env.local`.
 - Attention : Comme expliqué [précédemment](#), les variables d'environnement ne doivent pas être publiées sur GitHub (le fichier `.env.local` est inclus dans `.gitignore`). Sinon, toute personne qui a accès au GitHub pourrait modifier la base de données.

Annexe 7 – Fonctionnement de Prisma

Installation de Prisma

1. Installer le package avec la commande `npm i prisma -D`.
 - `-D` (ou `--save-dev`), va enregistrer le package dans `devDependencies` (dans `package.json`) c'est-à-dire une dépendance qui est nécessaire en développement, mais inutile en production (utile pour le développeur, mais pas pour l'utilisateur).
2. Initialiser Prisma avec la commande `prisma init`.
 - Cette commande va créer un fichier `schema.prisma` qui contient le « schéma » du projet. Un « schéma », c'est tout simplement la structure des données sur la base de données. (prisma.io/docs/orm/prisma-schema)
3. Dans le fichier `schema.prisma`, remplacer `postgresql` par `mongodb` et `DATABASE_URL` par `MONGODB_URI`.

Prisma s'attend à ce que la variable `MONGODB_URI` se trouve dans un fichier `.env`, mais Next.js utilise un fichier `.env.local`. Pour forcer Prisma à utiliser le fichier `.env.local` :

4. Installer dotenv avec la commande `npm i dotenv-cli -D`.
5. Créer une commande personnalisée, en rajoutant dans le fichier `package.json`, sous `"scripts"`, `"prisma": "dotenv -e .env.local -- prisma"`.

Maintenant, pour les commandes qui ne nécessitent une connexion à la base de données, il est désormais possible d'utiliser la commande `npm run prisma` à la place de `npm prisma`.

Utiliser Prisma avec Vercel

Si on déploie le projet sur Vercel, on obtient l'erreur suivante :

```
PrismaClientInitializationError: Prisma has detected that this project was built on Vercel, which caches dependencies. This leads to an outdated Prisma Client because Prisma's auto-generation isn't triggered. To fix this, make sure to run the 'prisma generate' command during the build process.
```

Pour régler ce problème, il faut simplement rajouter dans `package.json`, sous `"scripts"`, `"postinstall": "prisma generate"`.

Utiliser Prisma avec Next.js

- prisma.io/docs/orm/more/help-and-troubleshooting/help-articles/nextjs-prisma-client-dev-practices

Commandes utiles

- Générer `schema.prisma` à partir de MongoDB : `npm run prisma db pull`
- Mettre à jour MongoDB avec `schema.prisma` : `npm run prisma db push`
- Vérifier que le `schema.prisma` correspond aux données MongoDB : `npm run prisma validate`

- Visualiser et modifier les données : `npm run prisma studio`
- Formater le fichier `schema.prisma` : `npx prisma format`
- Générer Prisma Client (ce qu'il faut importer) : `npx prisma generate`
 - Ça va générer à partir de `schema.prisma` tout le code TypeScript nécessaire pour interagir facilement avec la base de données (voir l'image ci-contre).

```
const categories = await db.categories.findMany({
  include: { courses: true },
});
categories.forEach((category) => {
  (property) courses: {
    id: string;
    order: number | null;
    name: string;
    url: string;
    categoryId: string;
  } & {
    lessons: {
      label: string;
      name: string;
      image: string | null;
      instructions: string | null;
    }[];
  }[];
});
```

Relations

Prisma permet de créer des « relations » entre différentes collections MongoDB. Par exemple, si on a une collection de `courses` qui ont chacun une `category`, on peut stocker dans MongoDB l'attribut `categoryId`, puis indiquer à Prisma de créer une `@relation` entre les deux collections et de `include` automatiquement des attributs `category` et `courses[]`, qui ne sont pas sur MongoDB. Cela permet d'accéder facilement à deux collections en même temps (voir les trois images ci-contre). (prisma.io/docs/orm/prisma-schema/data-model/relations)

```
model categories {
  courses courses[] @relation
```

```
model courses {
  categoryId String @db.ObjectId
  category categories @relation(fields: [categoryId], references: [id])
```

Extensions

Prisma permet de créer des « extensions », c'est-à-dire des méthodes personnalisées directement à l'intérieur du Prisma Client, plutôt que de devoir définir des méthodes séparément dans un autre fichier. Par exemple, on peut créer une méthode `.users.getUtilisateur()`, qui retourne un utilisateur stocké sur MongoDB à l'aide des données de l'utilisateur authentifié (avec Clerk), ou bien `.courses.getCoursParURL(url)`, qui retourne simplement un cours stocké sur MongoDB selon son attribut `url`. (prisma.io/docs/orm/prisma-client/client-extensions)

```
model: {
  courses: {
    async getCoursParURL(url: string) {
      const context = Prisma.getExtensionContext(this);
      const user = await context.$parent.users.getUtilisateur();

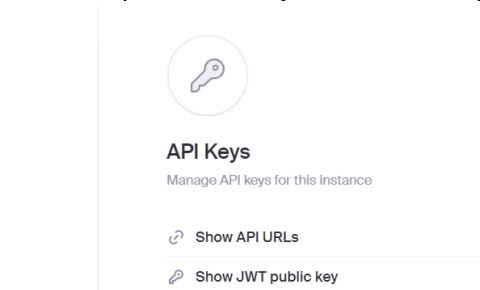
      return context.findUnique({
        where: { url },
        include: {
          category: true,
          progressions: { where: { userId: user.id } },
        },
      });
    },
  },
},
```

Annexe 8 – Authentification avec Clerk

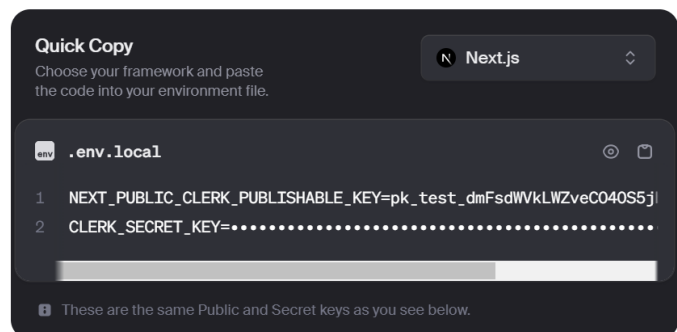
Installation de Clerk

1. Installer le package avec la commande `npm install @clerk/nextjs`
2. Configurer les variables d'environnement, c'est-à-dire les clés suivantes au fichier : `.env.local`:
`NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=YOUR_PUBLISHABLE_KEY`
`CLERK_SECRET_KEY=YOUR_SECRET_KEY`

Ces clés peuvent toujours être récupérées depuis la page des clés API de notre tableau de bord



Clerk :



Ajout de Middleware :

`clerkMiddleware()` nous permet d'accéder à l'état d'authentification des utilisateurs dans tout notre site, sur n'importe quelle page. Il nous permet également de protéger certaines pages des utilisateurs non authentifiés (ex : ne pas laisser les utilisateurs non authentifiés accéder à la page des cours). Pour ajouter `clerkMiddleware()` à notre application, nous avons suivi ces étapes :

1. Création du fichier `middleware.ts`
2. Dans `middleware.ts`, on exporte l'assistant `clerkMiddleware()` de Clerk :
3. On met le code pris de [documentation de Clerk](#) :

```
import { clerkMiddleware, createRouteMatcher } from "@clerk/nextjs/server";

const isProtectedRoute = createRouteMatcher([
  "/cours(.*)",
]);

export default clerkMiddleware((auth, req) => {
  if (isProtectedRoute(req)) auth().protect();
});

export const config = {
  matcher: ["/((?!\\.+\\.w+$|_next).*)", "/", "/(api|trpc)(.*)"],
};
```

4. Dans la constante `isProtectedRoute`, on mets l'url des pages que l'on veut que seuls les utilisateurs connectés peuvent voir (sinon ils sont redirigés vers la page `sign-in`)

Ajout des composantes de Clerk:

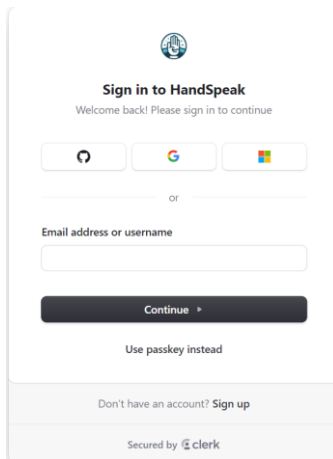
1. Tous les hooks et composants Clerk doivent être des enfants du composant `<ClerkProvider>`, qui fournit le contexte de session et d'utilisateur actif.
2. Maintenant, nous sommes prêts à utiliser les composantes de clerk, voici ceux qu'on a utilisés :
`<SignedIn>` : Les enfants de ce composant ne peuvent être vus que lorsqu'ils sont connectés.
`<SignedOut>` : Les enfants de ce composant ne peuvent être vus que lorsqu'ils sont déconnectés.
`<UserButton />` : Un composant préconstruit pour afficher l'avatar du compte avec lequel l'utilisateur est connecté.

Saib Merabet

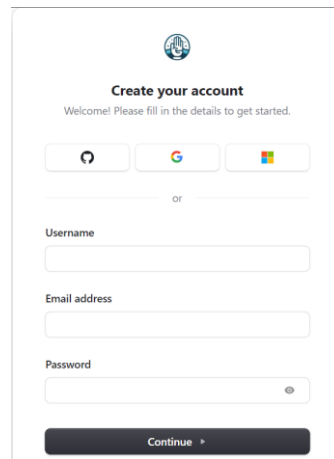


`<SignInButton />` : Un composant qui renvoie à la page de connexion ou affiche la fenêtre de connexion.

`<SignUpButton />` : Un composant qui renvoie à la page d'inscription ou affiche la fenêtre d'inscription.



The image shows a 'Sign in to HandSpeak' form. At the top is the HandSpeak logo and the title 'Sign in to HandSpeak' with the subtitle 'Welcome back! Please sign in to continue'. Below this are three social login buttons (Apple, Google, Microsoft) with an 'or' separator. A text input field is labeled 'Email address or username'. Below the field is a dark 'Continue' button with a right arrow. Underneath is a link 'Use passkey instead'. At the bottom, there is a link 'Don't have an account? Sign up' and a footer 'Secured by clerk' with the Clerk logo.



The image shows a 'Create your account' form. At the top is the HandSpeak logo and the title 'Create your account' with the subtitle 'Welcome! Please fill in the details to get started.' Below this are three social login buttons (Apple, Google, Microsoft) with an 'or' separator. The form has three text input fields: 'Username', 'Email address', and 'Password'. The 'Password' field has a toggle icon for visibility. At the bottom is a dark 'Continue' button with a right arrow.

`<SignIn/>` : Ce composant est mis dans la page de connexion, et montre la page de connexion de Clerk.

`<SignUp/>` : Ce composant est mis dans la page d'inscription, et montre la page d'inscription de Clerk.

Annexe 9 – Connexion entre Clerk et MongoDB

Pour pouvoir stocker des informations sur les utilisateurs sur MongoDB, il faut avoir accès aux utilisateurs Clerk sur MongoDB. Dans notre cas, la seule information qui sera nécessaire à enregistrer sur MongoDB sera le *User ID* stocké sur Clerk. Pour ce faire, la méthode recommandée par Clerk est l'utilisation de webhooks, c'est-à-dire des requêtes envoyées automatiquement lorsqu'un événement se produit. Sur le site de Clerk, on va créer un webhook pour les événements *user.created* et *user.deleted*. Et avec Next.js, on va créer un API qui va recevoir les requêtes et créer ou supprimer un utilisateur avec un *clerkId*.

(clerk.com/docs/integrations/webhooks/sync-data)*

*Pour tester le webhook localement, on utilise *ngrok*, mais une fois que l'API est déployée sur Vercel, on peut désinstaller *ngrok*.