

DOSSIER DE PROJET

Arnaud Guevaer

Code Quest Academy

Développeur Web et Web Mobile

Wild Code School



Introduction

Ce projet a été réalisé par notre groupe d'étudiants en reconversion professionnelle pour le titre de développeur web et web mobile à la Wild Code School.

Après plus de 6 ans à travailler en tant qu'agent logistique, j'ai décidé de me reconvertis dans un domaine qui me plaît vraiment. Le métier de développeur web est apparu pour moi comme une évidence, étant passionnée depuis toujours d'informatique et de nouvelles technologies.

Cette reconversion représente pour moi bien plus qu'un simple changement de carrière : c'est une opportunité de prendre confiance en moi, de sortir de ma zone de confort afin d'exercer un métier qui me motive au quotidien.

J'ai choisi cette formation pour son format court et intense de 5 mois, durant lesquels nous avons appris le développement web full stack.

Nous avons commencé par les bases de la programmation avec HTML et CSS, puis Javascript, jusqu'à maîtriser le framework React avec Typescript pour la partie front-end.

Pour la partie back-end, nous avons appris à utiliser Node.js avec le framework express, et les bases de données avec SQL.

Sommaire

Introduction	<i>page 1</i>
1 - Présentation du projet	<i>page 3</i>
2 - Équipe et organisation	<i>page 4</i>
3 - Compétences mise en oeuvre	<i>page 5</i>
3.1 - Développer la partie front-end d'une application web ou web mobile sécurisée	<i>page 5</i>
3.2 - Développer la partie back-end d'une application web ou web mobile sécurisée	<i>page 9</i>
4 - Les besoins du projet	<i>page 12</i>
5 - L' environnement technique	<i>page 13</i>
5.1 - Technologies utilisées	<i>page 13</i>
5.2 - Architecture du projet	<i>page 14</i>
5.3 - La base de données	<i>page 16</i>
5.4 - Les routes	<i>page 17</i>
5.5 - Sécurité de l'application	<i>page 18</i>
6 - Réalisations permettant la mise en oeuvre des compétences	<i>page 19</i>
6.1 - La page de connexion	<i>page 19</i>
6.2 - La page du profil	<i>page 27</i>
6.3 - La page de jeu	<i>page 29</i>
7 - Jeu d'essai	<i>page 35</i>
8 - Veille sur les vulnérabilités de sécurité	<i>page 37</i>
9 - Conclusion	<i>page 39</i>
9.1 - Résumé des acquis et des compétences développées	<i>page 39</i>
9.2 - Perspectives d'amélioration	<i>page 39</i>
9.3 - Récapitulatif et remerciements	<i>page 40</i>

1. Présentation du projet

Code Quest Academy est un projet de jeu éducatif web destiné aux étudiants et passionnés de développement.

Il propose une expérience immersive où les joueurs doivent résoudre des énigmes et répondre à des questions techniques pour progresser.

L'histoire se déroule dans une école de développement informatique qui a été frappée par un mystérieux bug, bloquant l'accès aux différentes salles de formation.

Chaque salle représente une technologie clé enseignée durant une formation en développement web et web mobile (HTML, CSS, JavaScript, React, Node.js et SQL).

Pour avancer, le joueur doit relever des défis liés à la technologie concernée. À la fin de chaque niveau, un boss incarne la technologie en question et doit être "vaincu" pour débloquer le niveau suivant.

Le projet met l'accent sur la **gamification de l'apprentissage**, rendant la montée en compétence plus ludique et interactive. Il repose sur un système de progression dynamique, une interface engageante et un backend robuste pour gérer les scores et les niveaux.

Il est également truffé de **références à la pop culture et de jeux de mots inspirés des langages informatiques**, rendant l'expérience encore plus fun et engageante.

2. Équipe et organisation

L'équipe de notre projet est constitué de 3 personnes :

Emeric LESAGE, Mathieu PRIEZ et moi-même, Arnaud GUEVAER.



Chacun a pu développer une partie du site et utiliser toutes les connaissances apprises lors de la formation, garantissant une expérience d'apprentissage complète et collaborative.

Nous avons été sensibilisés aux méthodes Agile et Scrum. Nous avons décidé de conserver certains aspects tout en adaptant notre fonctionnement :

- Nous avons mis en place un daily meeting chaque jour pour faire le point sur le travail effectué et définir les prochaines étapes.
- À la fin de chaque semaine, nous réalisions une rétrospective pour évaluer notre progression et ajuster notre organisation si nécessaire.
- Nous avons choisi de ne pas travailler en sprints, cette approche ne nous semblait pas adaptée à notre équipe et à notre niveau d'expérience sur ce type de gestion.

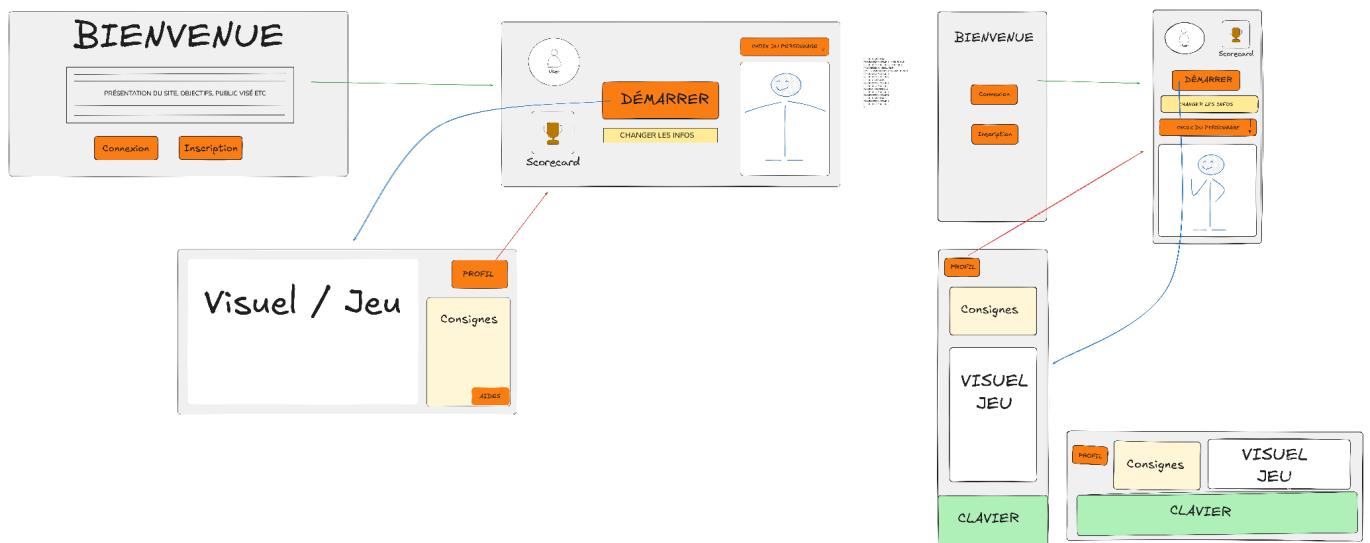
Lors d'un précédent projet, nous avions expérimenté la rotation des rôles de Product Owner (PO) et Scrum Master afin de mieux comprendre leur impact au sein d'une équipe de développement. Cependant, pour ce projet, nous avons décidé collégialement de ne pas reproduire cette organisation, estimant qu'elle n'était pas essentielle dans une équipe réduite de trois développeurs.

3. Compétences mises en oeuvre

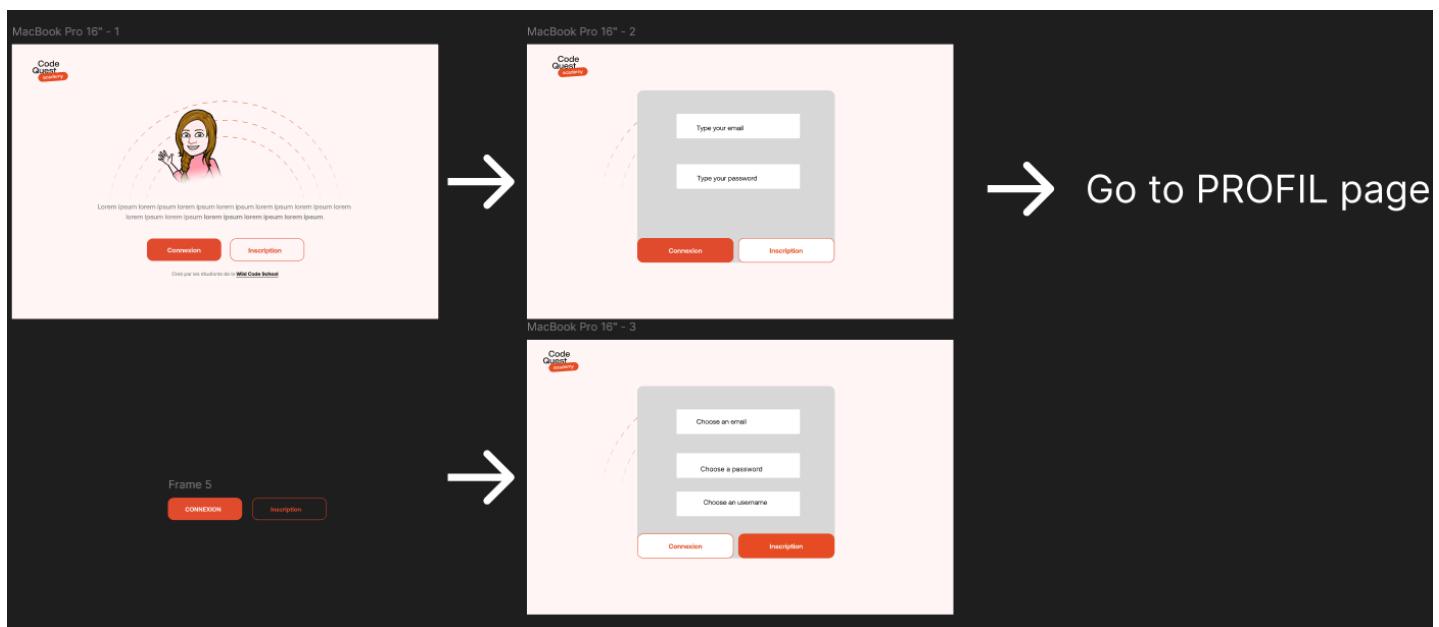
3.1 - Développer la partie front-end d'une application web ou web mobile sécurisée

a) Maquetter des interfaces utilisateur web ou web mobile

Dans un premier temps, nous avons rassemblé toutes nos idées et échangé sur les différentes possibilités, avant de concevoir ensemble un premier wireframe sur **Excalidraw**. Cet outil nous a permis de visualiser rapidement la structure et l'organisation de notre interface, facilitant ainsi les ajustements avant le développement



Nous avons ensuite utilisé **Figma** pour transformer ce brouillon en un prototype plus abouti, offrant un premier aperçu visuel de notre application. Ce prototype nous a servi de référence tout au long du développement, facilitant ainsi la cohérence et l'alignement de notre travail



Page d'accueil, avec les popups de connexion et d'inscription

Page de profil

MacBook Pro 16" - 2

Code Quest Academy

LEVEL 3 QUÊTE : XXXXX

GESTION DES UTILISATEURS

PSEUDO FORMATEUR 1 : XXXXXX
PSEUDO FORMATEUR 2 : XXXXXX

MODIFIER MES INFORMATIONS

PSEUDO : XXXXXX
MOT DE PASSE : XXXXX
EMAIL : XXX

MODIFIER MES INFORMATIONS

ADMINISTRATION →

ID	USERNAME	MAIL	LEVEL
1	ADMIN	ADMIN@GMAIL.COM	7
2	LAPINOUE2	LAPINOUE@HOTMAIL.FR	2
3	LOUIS	LOUISO@LAPOSTE.NET	1

Page de profil admin avec le panneau de gestion des utilisateurs

PROFIL

CONSIGNES

AIDES

COMMANDES

Frame avec img_bg

Frame 2 Quete 1

LAPINOUE2 LEVEL1

Bienvenue à l'université LAPINOUE ! Un mystérieux bug a infecté le campus, qui empêche les étudiants de coder correctement. Pour apprendre... on ne peut accéder qu'au terminal via des requêtes SQL. Aides-nous à résoudre ce mystère !

LAPINOUE2 LEVEL1

POUR QUE LA PAGE HTML FONCTIONNE CORRECTEMENT, ELLE DOIT RESPECTER UNE STRUCTURE JEUNESSE. CEPENDANT, LA PAGE EST CORROMPURE ET MANQUE DES ELEMENTS ESSENTIELS, COMME LA DECLARATION DE TYPE DE DOCUMENT ET LA BALISE <HTML>. VOTRE PREMIERE MISSION EST DE RETROUVER CES ERREURS ET D'AJOUTER LES BALISES MANQUANTES DANS LA STRUCTURE DES PAGES.

CHOISIS LA BONNE BALISE :

```
<H1></H1>
<FOUTEUR></FOUTEUR>
<P></P>
<HTML></HTML>
```

Page de jeu

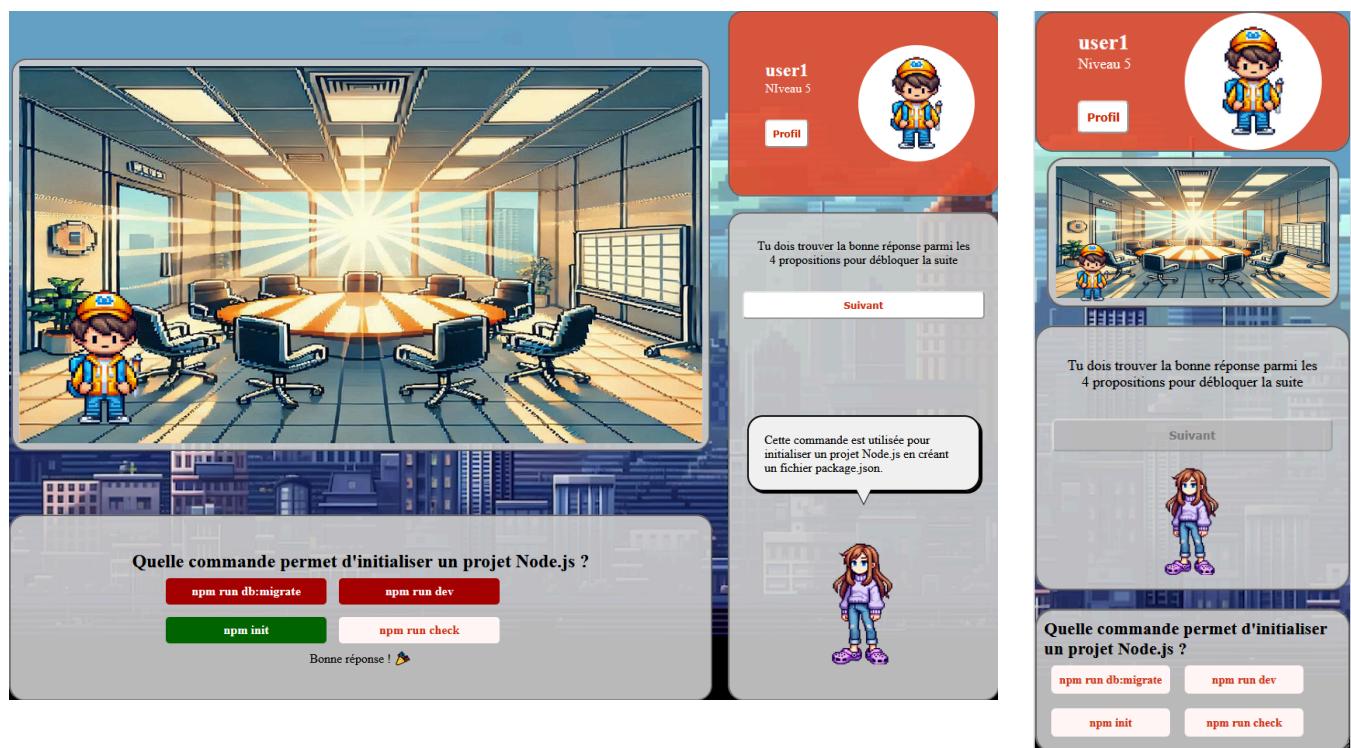
b) Réaliser des interfaces utilisateur statiques web ou web mobile

Après avoir finalisé notre prototype sur Figma, nous avons entrepris la réalisation de la structure des différentes pages. Nous avons veillé à respecter la charte graphique définie et à assurer une cohérence visuelle sur l'ensemble des écrans.

Pour la version web, nous avons mis en place une structure claire et intuitive, facilitant la navigation des utilisateurs. Cependant, l'adaptation mobile et tablette ont posé certaines contraintes, notamment en raison de la complexité de la partie jeu, ce qui nous a conduits à développer le jeu uniquement pour ordinateur.

c) Développer la partie dynamique des interfaces utilisateur web ou web mobile

Une fois notre prototype finalisé, nous avons entamé le développement des différentes pages de notre projet en utilisant le framework **React**. Nous avons veillé à structurer notre code de manière modulaire pour faciliter la maintenance et la réutilisation des composants. De plus, nous avons mis en place une gestion efficace des états et des interactions afin d'assurer une navigation fluide et intuitive pour l'utilisateur.



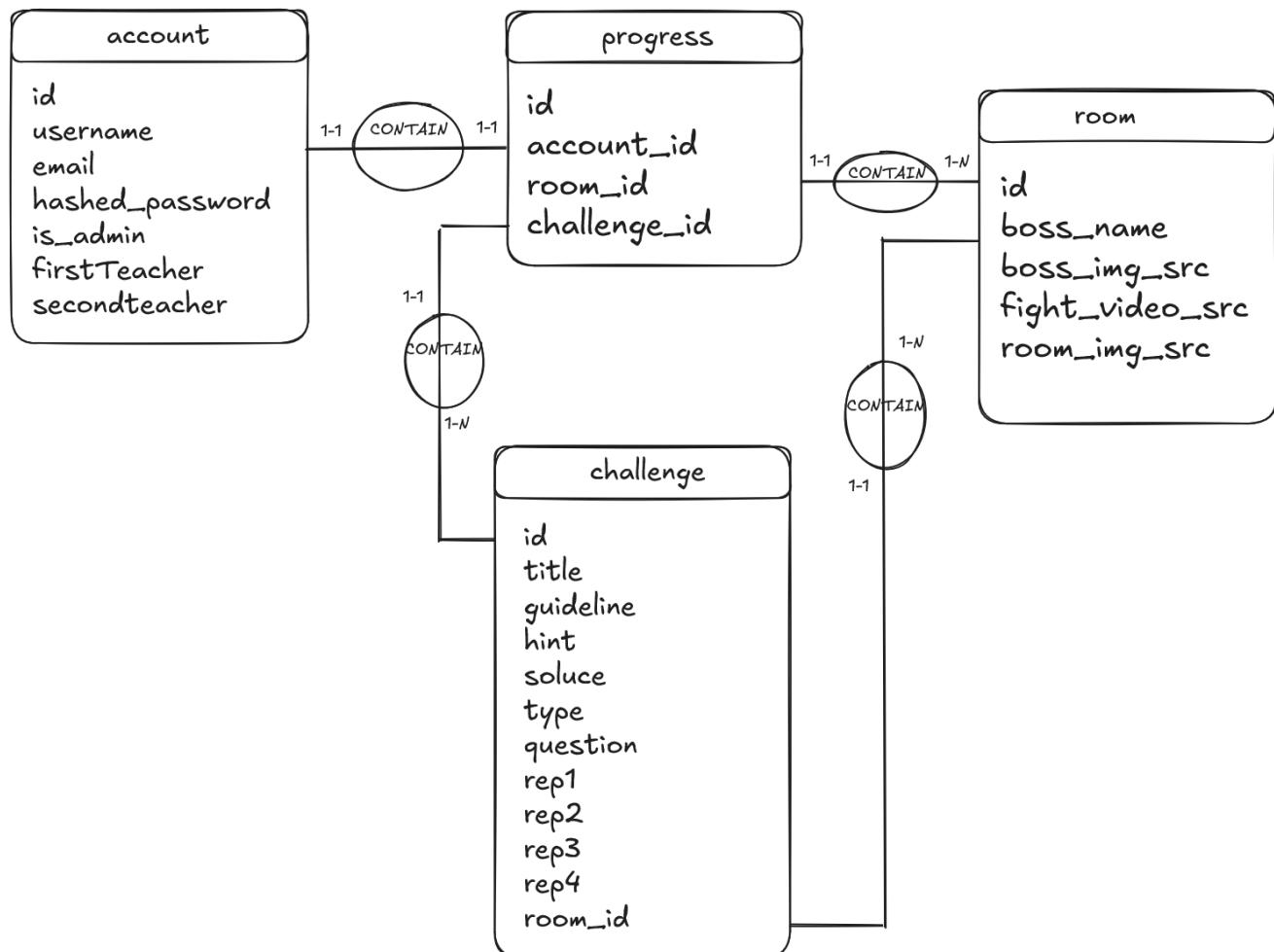
Page de jeu version desktop et mobile

3.2 - Développer la partie back-end d'une application web ou web mobile sécurisée

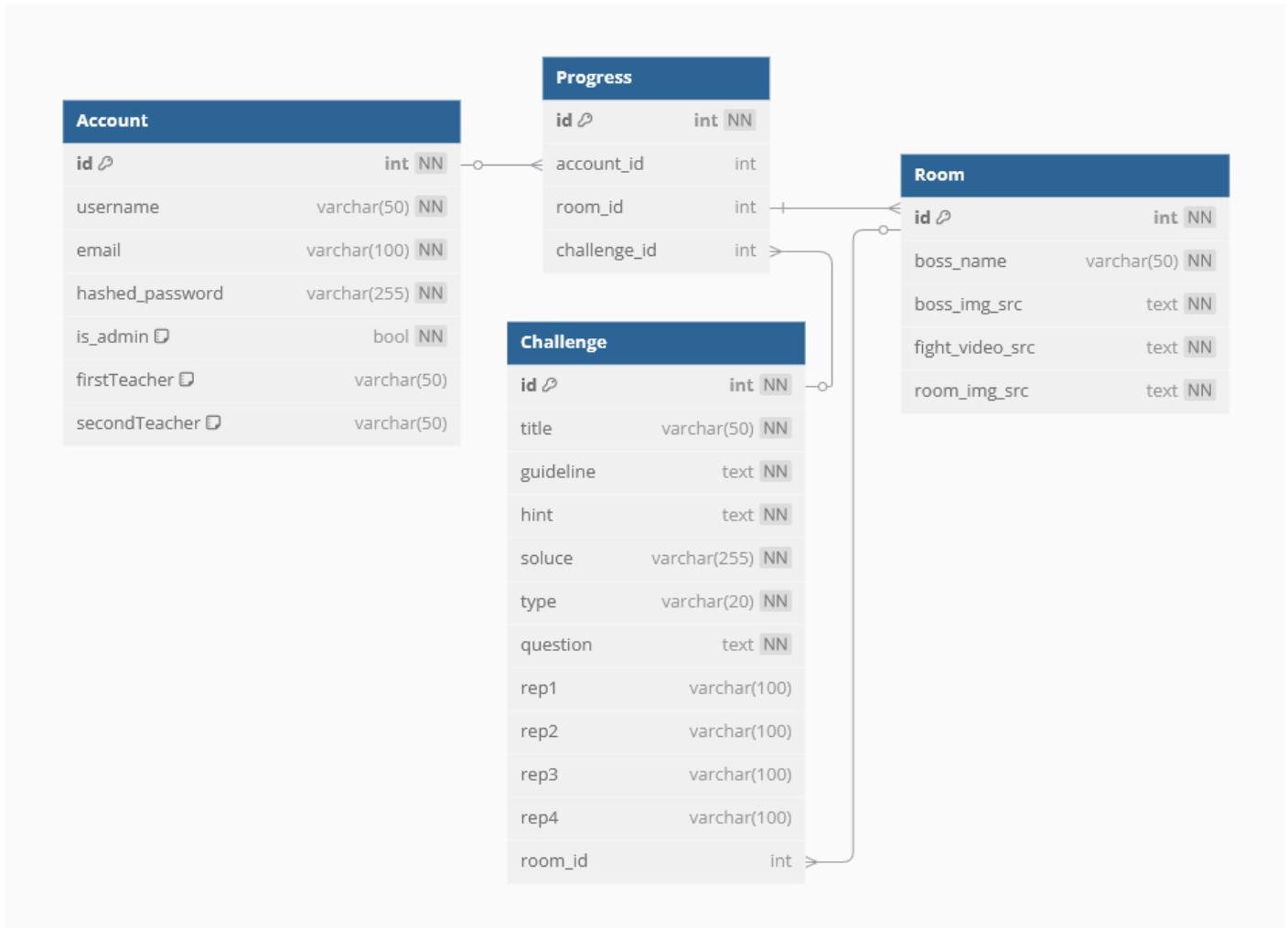
a) Mettre en place une base de données relationnelle

Pour concevoir notre base de données, nous avons d'abord analysé nos besoins afin de déterminer quelles informations devaient être stockées et comment elles seraient utilisées. Nous avons ensuite la **méthode Merise** pour modéliser la structure de la base en identifiant les principales entités et leurs relations, garantissant ainsi une organisation optimale des données.

Voici le **MCD** (Modèle Conceptuel de Donnée) :



Nous avons ensuite utilisé l'outil en ligne **dbdiagram** pour afficher le **MPD** (Modèle Physique de Donnée) :



b) Développer des composants d'accès aux données SQL et NoSQL

Dans ce projet, nous avons développé des **composants d'accès aux données** en adoptant l'architecture **MVC (Modèle-Vue-Contrôleur)**, ce qui permet une séparation nette des responsabilités et simplifie la maintenance du code à long terme.

c) Développer des composants métier côté serveur

Nous avons développé des **composants métier côté serveur** en mettant en place des **routes** pour gérer les différentes requêtes HTTP (POST, PUT, GET, DELETE, etc.), chacune correspondant à une action spécifique dans l'application. Chaque **route** est associée à un **contrôleur**, qui reçoit la requête, effectue les traitements nécessaires et renvoie une réponse appropriée.

Les **composants métier** gèrent la logique métier en validant les données, en appliquant des règles spécifiques et en orchestrant les interactions avec la base de données. Cela permet de centraliser la logique côté serveur, garantissant une séparation claire entre la gestion des données et la présentation ou l'interface utilisateur.

4. Les besoins du projet

Il nous a été demandé de réaliser un jeu éducatif interactif où l'utilisateur peut apprendre et réviser différents langages de programmation à travers des niveaux et des défis. L'objectif est de rendre l'apprentissage des langages informatiques ludique et motivant.

Fonctionnalités principales :

Utilisateur :

- Créer un compte et se connecter.
- Accéder à sa page de profil où il pourra :
 - Modifier ses informations comme son pseudo, mot de passe ou email.
 - Modifier le nom de ses formateurs (2 personnages présents dans le jeu).
 - Modifier sa progression dans le jeu.
 - Se déconnecter.
- Accéder à la page du jeu (via la page profil) où il pourra :
 - Compléter les challenges via un quizz ou un prompt, et ainsi accéder aux suivants.
 - Recevoir une astuce au challenge en cours, via l'image du formateur.
 - Retourner à son profil.

Administrateur :

L'administrateur dispose des mêmes fonctionnalités avec en plus l'accès à la page de gestion des utilisateur où il pourra :

- Voir la liste de tous les comptes créés.
- Modifier les informations d'un compte.
- Supprimer un compte.

5. L'environnement technique

5.1 Technologies utilisées

Pour le **front-end**, nous avons utilisé le framework React.js pour créer des composants réutilisables, ainsi que Typescript pour renforcer la sécurité du code.

Pour le style, nous avons utilisé CSS pour la mise en page et la personnalisation des éléments graphiques.

Nous avons adopté une approche modulaire afin de garantir une meilleure organisation du code et une facilité de maintenance.

Dépendances :

- **React router** : Permet de créer des routes et facilite la navigation entre les pages.

Pour le **back-end**, nous avons utilisé Express.js, un framework léger et rapide qui facilite la création de serveurs et d'API avec Node.js.

Pour la base de données, nous avons utilisé MySQL.

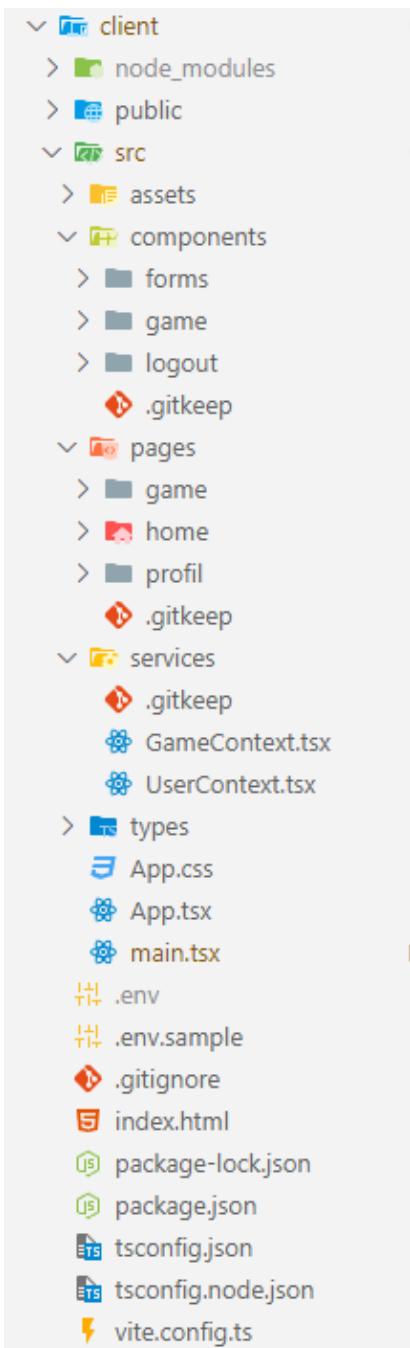
Dépendances :

- **Argon2** : Un algorithme de hachage sécurisé utilisé pour protéger les mots de passe en les transformant en une chaîne de caractères difficile à déchiffrer.
- **Cors** : Un mécanisme de sécurité qui permet d'autoriser l'accès à notre API uniquement depuis notre URL.
- **Dotenv** : Un outil qui permet de charger des variables d'environnement à partir d'un fichier .env, facilitant la gestion de configurations sensibles
- **JsonWebToken** : Une méthode sécurisée pour créer un token crypté qui permet de stocker des données, notamment pour la connexion et la session de l'utilisateur.
- **Mysql2** : Un module qui permet à l'application de se connecter et d'interagir avec la base de données.

Nous avons également utilisé Microsoft Copilot afin de générer les images grâce à l'IA, et j'ai personnellement utilisé Canva pour créer les animations de combat contre les boss.

5.2 L'architecture du projet

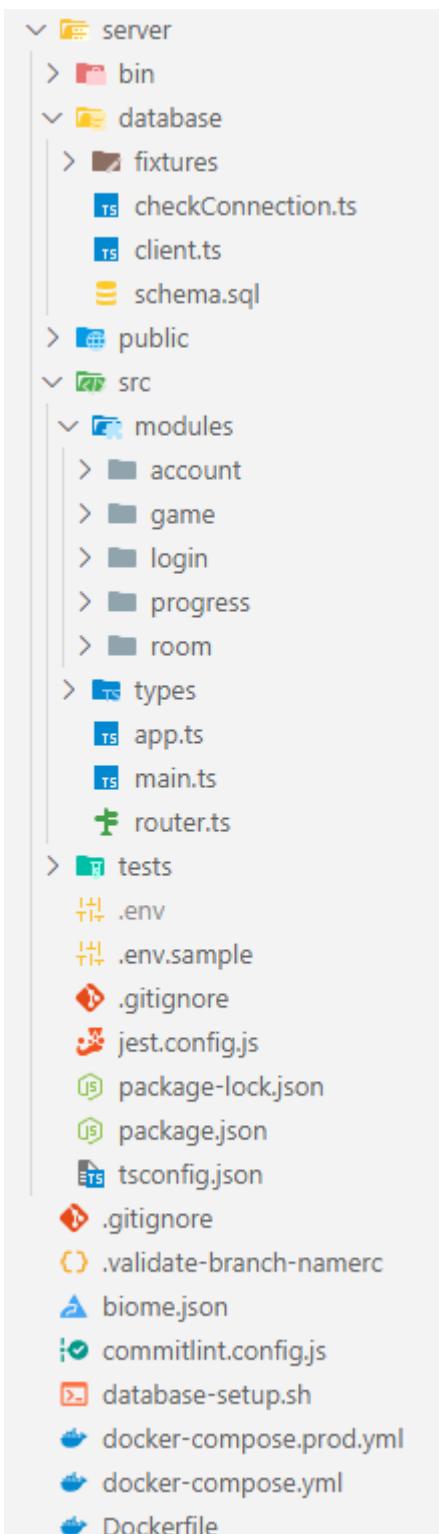
Nous avons adopté l'**architecture MVC (Modèle-Vue-Contrôleur)** afin de structurer notre application de manière claire et efficace. Cette approche permet de séparer la gestion des données (**Modèle**), l'affichage (**Vue**) et la logique métier (**Contrôleur**), ce qui facilite la maintenance, l'évolution et la compréhension du code.



Côté front-end :

L'ensemble du code est organisé dans un dossier **src**, qui contient les éléments suivants :

- **Un dossier assets**, avec toutes les images et vidéos utilisées pour le projet.
- **Un dossier components**, avec tous les composants.
- **Un dossier pages**, qui regroupe les différentes pages de l'application.
- **Un dossier services**, pour les contextes.
- **Un dossier types**, qui regroupe tous les types et les exporte afin de pouvoir les réutiliser.
- **Les fichiers principaux nécessaires au bon fonctionnement de l'application**, app, main, .env... etc.



Côté back-end :

L'ensemble du code back-end est organisé dans un dossier **server**, qui contient les éléments suivants :

- **Un dossier database**, qui contient la base de données(schema.sql) et les fichiers permettant l'interaction avec celle-ci.
- **un dossier src**, avec les modules(contrôleurs) qui permettent de faire la liaison encore la vue et le modèle, les types, les fichiers principaux nécessaires au bon fonctionnement de l'application, app, main, router... etc

5.3 La base de données

Notre base de données contient 4 tables :

Account :

Contient les informations de l'utilisateur

- id** : Identifiant unique de l'utilisateur, généré automatiquement.
- username** : Pseudo de l'utilisateur, c'est un champ obligatoire.
- email** : Adresse email de l'utilisateur, elle est aussi obligatoire et doit être unique.
- hashed_password** : Mot de passe de l'utilisateur, stocké sous forme de hash pour plus de sécurité.
- is_admin** : Booléen pour indiquer si l'utilisateur est un administrateur. Par défaut, il est FALSE.
- firstTeacher** : Nom du premier formateur attribué à l'utilisateur (champ optionnel).
- secondTeacher** : Nom du second formateur attribué à l'utilisateur (champ optionnel).

Progress :

Sert à stocker la progression de l'utilisateur

- id** : Identifiant unique du progrès, généré automatiquement.
- account_id** : L'ID de l'utilisateur auquel le progrès appartient.
- room_id** : L'ID de la salle où l'utilisateur est arrivé.
- challenge_id** : L'ID du challenge auquel l'utilisateur est arrivé.

Room :

Sert à stocker les informations de la salle, essentiellement des images.

- id** : Identifiant unique de la salle, généré automatiquement.
- boss_name** : Nom du boss correspondant à la salle.
- boss_img_src** : Source de l'image du boss.
- fight_video_src** : Source de la vidéo de combat du boss.
- room_img_src** : Source de l'image d'arrière plan correspondant à la salle.

Challenge :

Sert à stocker les informations du challenge en cours

- id** : Identifiant unique du challenge, généré automatiquement.
- title** : Titre du challenge correspondant généralement au nom du langage informatique de la salle.
- guideline** : Consignes ou instructions pour compléter le challenge.
- hint** : Un indice pour aider l'utilisateur s'il est bloqué.
- solute** : La solution du challenge.
- type** : Type du challenge (quizz, prompt...)
- question** : La question principale du challenge.
- rep1, rep2, rep3, rep4** : Réponses possibles (nécessaires pour le quizz, optionnelles pour les autres types).
- room_id** : Fait référence à la salle où se trouve ce challenge.

5.4 Les routes

Pour assurer une navigation fluide et intuitive dans notre application, nous avons structuré les routes de manière claire en utilisant React Router.

Le point d'entrée principal est défini par le composant `<App />`, qui englobe les différentes pages accessibles via des chemins bien définis.

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      {
        path: "/",
        element: <HomePage />,
      },
      {
        path: "/game",
        element: <GameBoard />,
      },
      {
        path: "/profile",
        element: <ProfilPage />,
      },
      {
        path: "/admin/manage",
        element: <AdminManagement />,
      },
      {
        path: "/admin",
        element: <AdminPage />,
      },
    ],
  },
]);
```

L'utilisateur peut ainsi naviguer entre la page d'accueil ("`/`"), son profil (`"/profile"`) et l'espace de jeu (`"/game"`).

Si le compte utilisateur est **administrateur**, c'est la page "`/admin`" qui s'affiche à la place de "`/profile`".

Un espace de gestion administrateur est également mis en place avec la page ("`/admin/manage`").

5.5 Sécurité de l'application

Nous avons mis en place plusieurs mesures afin de protéger les données utilisateur et assurer l'intégrité du système.

Authentification et autorisation : Nous utilisons JSON Web Token (**JWT**) pour créer des tokens cryptés qui stockent les données d'identification. Cela permet une gestion sécurisée des sessions utilisateurs et des accès restreints. De plus, cela permet à l'utilisateur de rester connecté même après avoir actualisé la page.

Cryptage des mots de passe : Les mots de passe utilisateurs sont cryptés avec **Argon2**, un algorithme de hachage sécurisé qui protège contre les attaques de type "brute force" et assure que même en cas de fuite de données, les mots de passe restent illisibles.

Contrôle des accès API : Le package **CORS** (Cross-Origin Resource Sharing) est configuré pour autoriser l'accès à l'API uniquement depuis notre URL spécifique, limitant ainsi les risques d'accès non autorisés provenant de sources externes non fiables.

Protection contre les injections SQL : Nous utilisons des requêtes paramétrées avec des paramètres préparés (notamment avec le symbole ? dans les requêtes SQL), qui permettent de séparer les données des commandes SQL. Cela empêche les attaques par injection SQL. En séparant la logique de la donnée, ces requêtes assurent que les entrées utilisateur sont traitées comme des données, et non comme des parties de code exécutable.

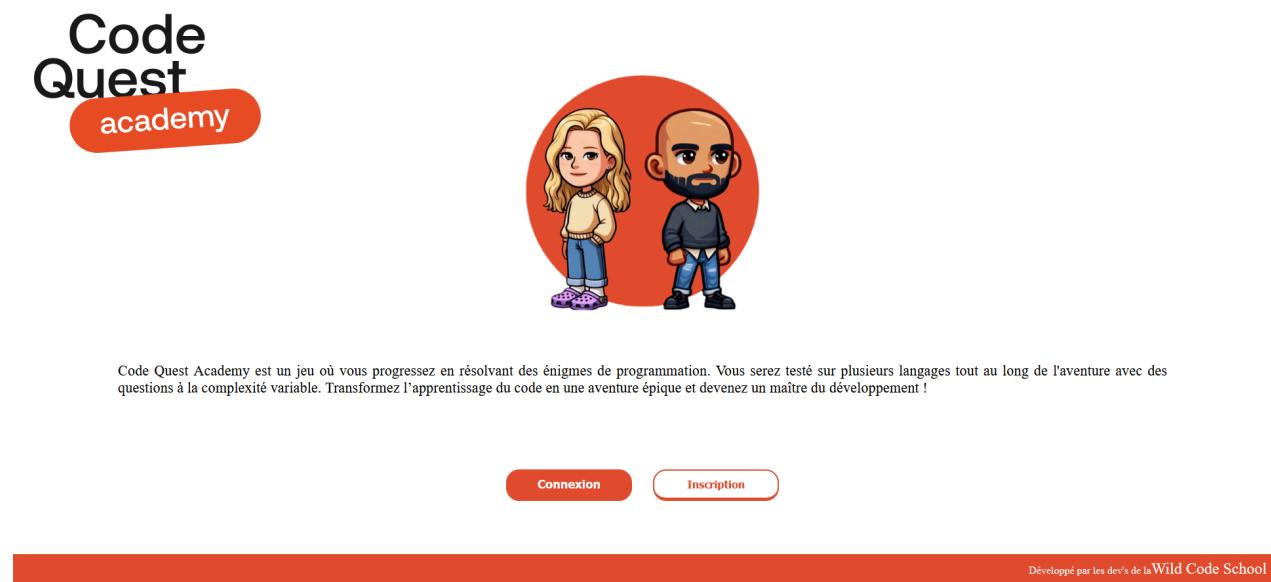
Sécurisation des pages : Les routes réservées aux admin sont sécurisées par des conditions vérifiant les données de l'utilisateur, empêchant ainsi les utilisateurs non autorisés d'y accéder manuellement via l'URL.

6. Réalisations permettant la mise en oeuvre des compétences

6.1 - La page de connexion

Lorsque l'utilisateur arrive sur le site, il est accueilli par une page d'accueil proposant deux options : **se connecter** ou **s'inscrire**.

L'objectif de cette page est de fournir à l'utilisateur un moyen simple et intuitif de créer un compte ou d'accéder à son espace personnel. Une fois connecté, il pourra accéder à son profil, suivre sa progression et interagir avec les différentes fonctionnalités du jeu.



L'ouverture et la fermeture des popup est gérée avec le hook “**useState**” de React.

```
const [activeForm, setActiveForm] = useState<activeForm>(null);  
const closeForm = () => setActiveForm(null);
```

```

<section className="homepage-buttons-container">
  <button
    type="button"
    className="button-type1 homepage-button login-button"
    onClick={() => setActiveForm("login")}
  >
    Connexion
  </button>
  <button
    type="button"
    className="button-type2 homepage-button register-button"
    onClick={() => setActiveForm("signup")}
  >
    Inscription
  </button>
</section>

```

Un clic sur le bouton “Connexion” définit **activeForm** sur “login” et un clic sur le bouton “Inscription” le définit sur “signup”.

```

{activeForm === "login" && (
  <div className="form-container">
    <LoginForm closeForm={closeForm} />
  </div>
)
{activeForm === "signup" && (
  <div className="form-container">
    <SignupForm closeForm={closeForm} />
  </div>
)
}

```

En fonction de la valeur de **activeForm**, la popup de connexion ou d’inscription sera affichée.

La fonction **closeForm** passée en props au composants permet de fermer la popup.

Nous utilisons CSS pour le style, en le positionnant de façon à ce que la popup se place au milieu de l’écran.

Le *background-color* noirci l’arrière-plan et le *z-index* assure que la popup s’affiche au premier plan.

```

.form-container {
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;
  background-color: black;
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

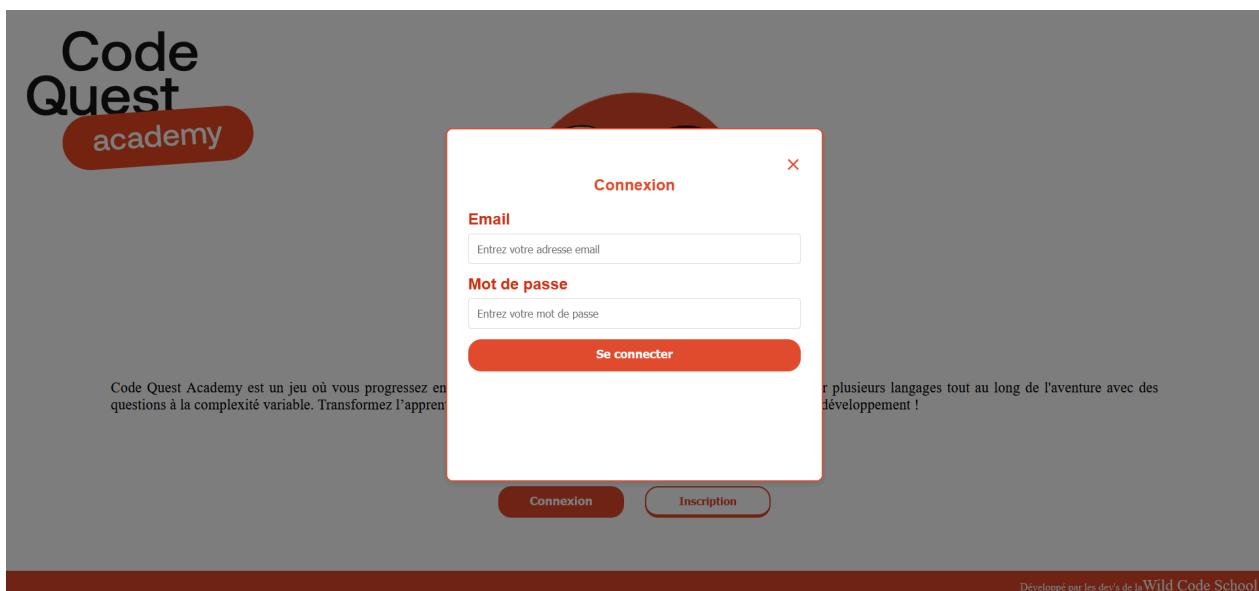
```

Composant de Connexion :

Le composant **LoginForm** permet aux utilisateurs de se connecter à l'application en renseignant leur email et mot de passe. Il envoie ces informations à l'API pour authentification, gère les éventuelles erreurs et stocke le token reçu pour maintenir la session utilisateur.

Il utilise plusieurs hooks et modules :

- `useNavigate (React Router)` : permet de rediriger l'utilisateur après la connexion.
- `useContext (React)` : accède au contexte utilisateur (`UserContext`).
- `useRef (React)` : stocke les références des champs de saisie.
- `useState (React)` : gère les erreurs d'authentification.



Quand l'utilisateur a tapé ses informations et clique sur "se connecter" :

```
const response = await fetch(`import.meta.env.VITE_API_URL}/api/login`,
{
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    email: (emailRef.current as HTMLInputElement).value,
    password: (passwordRef.current as HTMLInputElement).value,
  }),
};

if (!response.ok) {
  const errorMessage = await response.json();
  throw new Error(errorMessage.message || "Erreur de connexion.");
}
```

Effectue une requête **POST** à l'API (/api/login) avec les identifiants de l'utilisateur.

Les valeurs des champs sont récupérées via **emailRef.current.value** et **passwordRef.current.value**.

On vérifie si la réponse est correcte (`response.ok`), si ce n'est pas le cas, on renvoie une erreur à l'utilisateur.

```

const token = response.headers.get("Authorization")?.split("Bearer ")[1];

if (!token) {
| throw new Error("Token non reçu");
}

const user = await response.json();

setUser(user);
setToken(token);

// Save the token in the local storage
try {
  localStorage.setItem("token", token);
} catch (storageError) {
  console.error("Erreur de stockage du token", storageError);
}

navigate(user.is_admin ? "/admin" : "/profile");
} catch (error) {
  console.error("Erreur de connexion : ", error);
  setError(
    error instanceof Error ? error.message : "Une erreur est survenue.",
  );
}
};


```

On récupère ensuite le token de l'en-tête de la réponse.

Si le token est absent, l'utilisateur reçoit une erreur.

On met ensuite à jour l'état utilisateur “**user**” dans le contexte utilisateur, et on stocke le token dans le localStorage, ce qui permet la persistance de la session.

L'utilisateur est ensuite redirigé vers sa page de profil, ou la page de profil admin pour les administrateurs.

Création de la route :

```

import loginActions from "./modules/login/loginActions";
router.post("/api/login", loginActions.login);

```

La route est créée dans le fichier router.ts.

Elle associe la méthode HTTP **POST** à l'endpoint **/api/login**, qui est gérée par la fonction **login** provenant du module **loginActions**.

Le module **loginActions** :

1. Vérifie si l'email existe dans la base de données.

```

const login: RequestHandler = async (req, res, next) => {
  try {
    // Retrieve the user by their email address
    const user = await accountRepository.readByEmail(req.body.email);

    // Check if the user exists in the database
    if (user == null) {
      res.status(422).json({ message: "Adresse mail inconnue" });
      return;
    }
  }
};

```

2. Vérifie si le mot de passe est correct en le comparant au mot de passe chiffré enregistré.

```
// Verify the password
const verified = await argon2.verify(
  user.hashed_password,
  req.body.password,
);

// If the password is correct, return the user's information
if (verified) {
  // Destructure `user` using the spread operator to return all properties except `hashed_password`
  const { hashed_password, ...userWithoutHashedPassword } = user;
```

3. Si tout est bon, un **token JWT** est généré et envoyé dans l'en-tête de la réponse. Ce token permet de s'authentifier sans avoir à saisir ses identifiants à chaque requête.

```
// Token generation code will be added here
const myPayload: MyPayload = {
  sub: user.id.toString(),
  isAdmin: user.is_admin,
};

const token = await jwt.sign(
  myPayload,
  process.env.APP_SECRET as string,
  { expiresIn: "24h" },
);

// Send the token in the Authorization Header
res.setHeader("Authorization", `Bearer ${token}`);

// Send the user's information in the response body
res.json(userWithoutHashedPassword);
} else {
  res.status(422).json({ message: "Mauvais mot de passe." });
}
```

Composant d'inscription :

Le composant **SignupForm** est un formulaire d'inscription permettant aux utilisateurs de créer un compte en renseignant un pseudo, une adresse email et un mot de passe. Il utilise le hook **useState** pour gérer les valeurs des champs et les messages d'erreur ou de succès.



Code Quest Academy est un jeu où vous progressez en résolvant des énigmes et des questions à la complexité variable. Transformez l'apprentissage en jeu !

S'Incrire

Pseudo
Entrez votre pseudo

Email
Entrez votre adresse email

Mot de passe
Entrez votre mot de passe

S'Incrire

Apprenez plusieurs langages tout au long de l'aventure avec des défis de développement !

Développé par les dev's de la Wild Code School

```
const SignupForm = ({ closeForm }: SignupFormProps) => {
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [message, setMessage] = useState<string | null>(null);
```

On définit le composant **SignupForm** et on utilise `useState` pour stocker les valeurs du pseudo, de l'email, du mot de passe, ainsi que le message de retour (succès ou erreur).

```

const handleSubmit = async (event: React.FormEvent) => {
  event.preventDefault();

  try {
    const response = await fetch(
      `${import.meta.env.VITE_API_URL}/api/accounts`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, email, password }),
      },
    );

    const data = await response.json();
  }
}

```

Lorsqu'un utilisateur soumet le formulaire, la fonction **handleSubmit** est exécutée. Elle empêche le rechargement de la page par défaut (**event.preventDefault()**), puis envoie une requête POST à l'API pour créer un compte.

```

if (response.ok) {
  setMessage("Inscription réussie !");
  setUsername("");
  setEmail("");
  setPassword("");
} else {
  setMessage(data.message || "Une erreur est survenue.");
}
} catch (error) {
  console.error("Erreur lors de l'inscription :", error);
  setMessage("Une erreur est survenue. Veuillez réessayer.");
}
}

```

Si l'inscription réussit, un message de succès est affiché, et les champs du formulaire sont réinitialisés

Sinon, un message d'erreur est affiché

Création de la route :

```
router.post("/api/accounts", accountActions.hashPassword, accountActions.add);
```

Cette route utilise la méthode HTTP **POST** et est associée à l'endpoint **/api/accounts**. Lorsqu'une requête est envoyée à cette adresse, elle est d'abord traitée par la fonction **hashPassword**, qui hache le mot de passe de l'utilisateur pour plus de sécurité. Ensuite, la fonction **add** ajoute le nouvel utilisateur à la base de données.

Le module accountActions :

```
const add: RequestHandler = async (req, res, next) => {
  try {
    const newAccount = {
      username: req.body.username,
      email: req.body.email,
      hashed_password: req.body.hashed_password,
      is_admin: req.body.is_admin,
      firstTeacher: req.body.firstTeacher,
      secondTeacher: req.body.secondTeacher,
    };

    const insertId = await accountRepository.create(newAccount);
    res.status(201).json({ success: true, insertId });
  } catch (err) {
    next(err);
  }
};
```

La fonction Add récupère les données envoyées dans le corps de la requête (**req.body**),

Elle crée un nouvel objet **newAccount** contenant ces informations.

Ensuite, elle appelle la méthode **create** du **accountRepository** pour insérer ce nouvel utilisateur dans la base de données.

Si l'insertion réussit, elle renvoie une réponse avec le statut **201** et l'**insertId** du nouvel utilisateur.

Le module accountRepository :

Le *repository* est une couche qui s'occupe directement des interactions avec la base de données. C'est là que la requête SQL réelle est exécutée pour ajouter un utilisateur à la base de données.

```
async create(account: Omit<Account, "id">) {
  const [result] = await databaseClient.query<Result>(
    `INSERT INTO account (username, email, hashed_password)
     VALUES (?, ?, ?);`,
    [account.username, account.email, account.hashed_password],
  );

  // Get the last inserted id
  const [rows] = await databaseClient.query<Rows>(
    "SELECT LAST_INSERT_ID() AS user_id",
  );
  const userId = rows[0].user_id;

  // we add a base progress for new users
  await databaseClient.query(
    `INSERT INTO progress (user_id, room_id, challenge_id)
     VALUES (?, ?, ?);`,
    [userId, 1, 1],
  );

  return result.insertId;
}
```

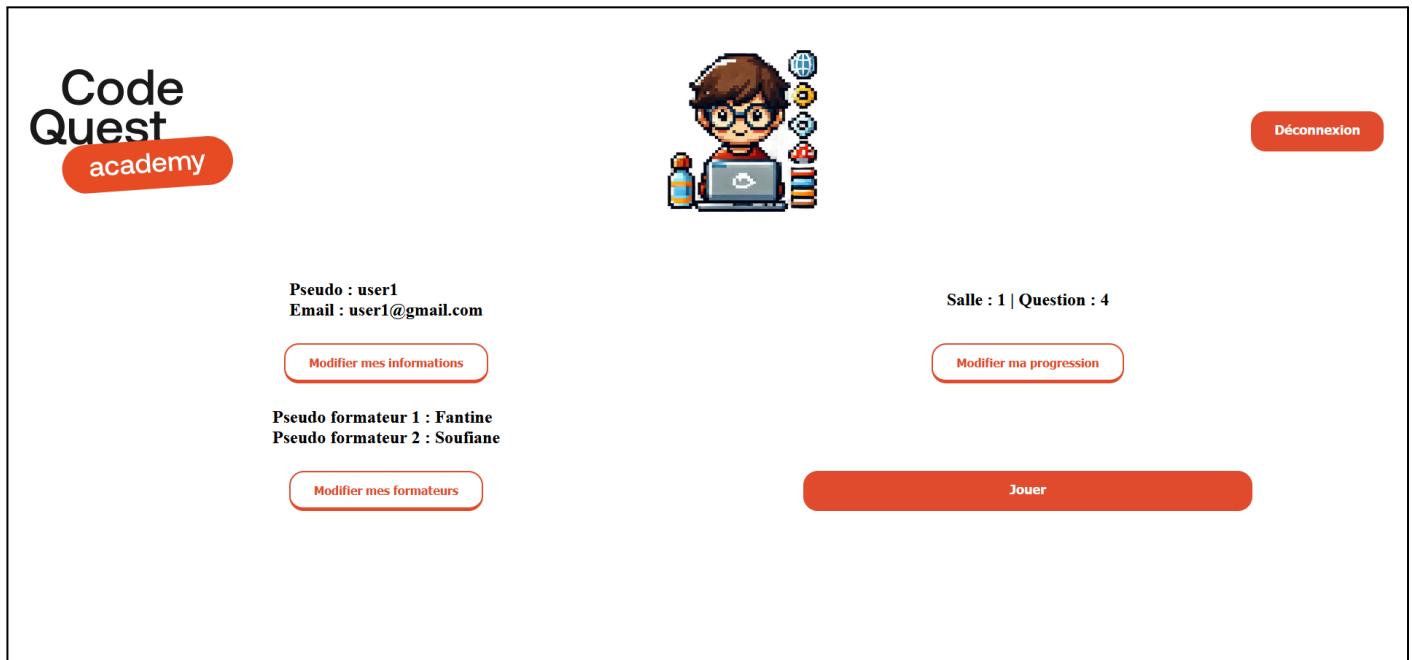
La fonction **create** exécute une requête SQL pour insérer un nouvel utilisateur dans la table **account**.

Ensuite, elle récupère l'ID du dernier utilisateur inséré avec la requête **SELECT LAST_INSERT_ID()**, puis ajoute une ligne dans la table **progress** pour initialiser les données de progression de l'utilisateur.

Enfin, la fonction retourne l'ID de l'insertion pour le renvoyer à l'action qui l'a appelée.

6.2 - La page de profil

Une fois connecté, l'utilisateur arrive sur sa page de profil :



Il a alors la possibilité de :

- **Modifier ses informations** : pseudo, mot de passe, email.
- **Modifier ses formateurs** : le nom des formateurs présents dans le jeu.
- **Modifier sa progression** : toutes les parties du jeu peuvent être recommencées.
- **Se déconnecter**.
- **Lancer le jeu**.

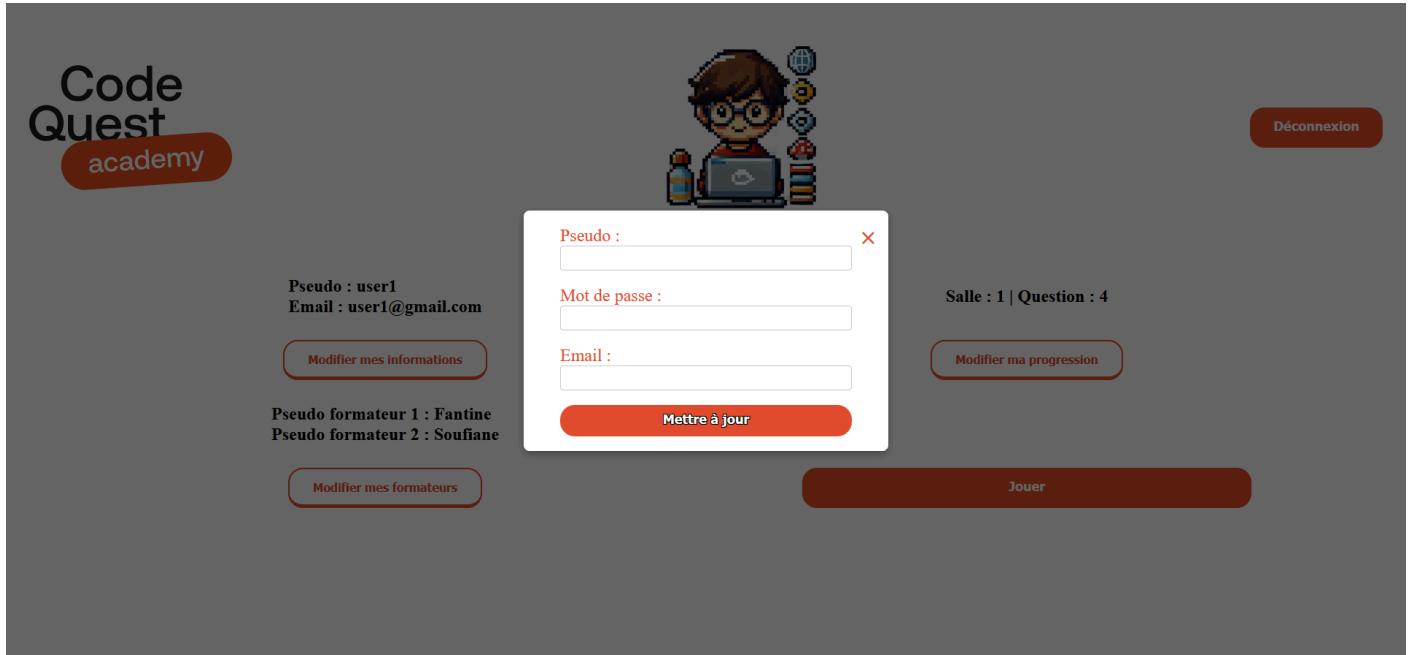
Sur cette page, les informations de l'utilisateur sont récupérés via le UserContext :

```
const userContext = useContext(UserContext);
const { user, progress } = userContext;
```

Ces informations sont ensuite stockés dans des états grâce au hook **useState** :

```
const [username, setUsername] = useState(user?.username);
const [email, setEmail] = useState(user?.email);
const [firstTeacher, setFirstTeacher] = useState(user?.firstTeacher);
const [secondTeacher, setSecondTeacher] = useState(user?.secondTeacher);
const [challengeId, setChallengeId] = useState(progress?.challenge_id);
```

Puis c'est le même principe que pour la page de connexion :
 Un clic sur un bouton de modification ouvre une popup avec des formulaires, où l'utilisateur tape ses nouvelles valeurs.



Une requête PUT est ensuite envoyée à la base de données pour éditer la table en question (ici la table account).

```
try {
  const response = await fetch(
    `${import.meta.env.VITE_API_URL}/api/accounts/${userId}/infos`,
  {
    method: "PUT",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ username, email, password }),
  },
);
```

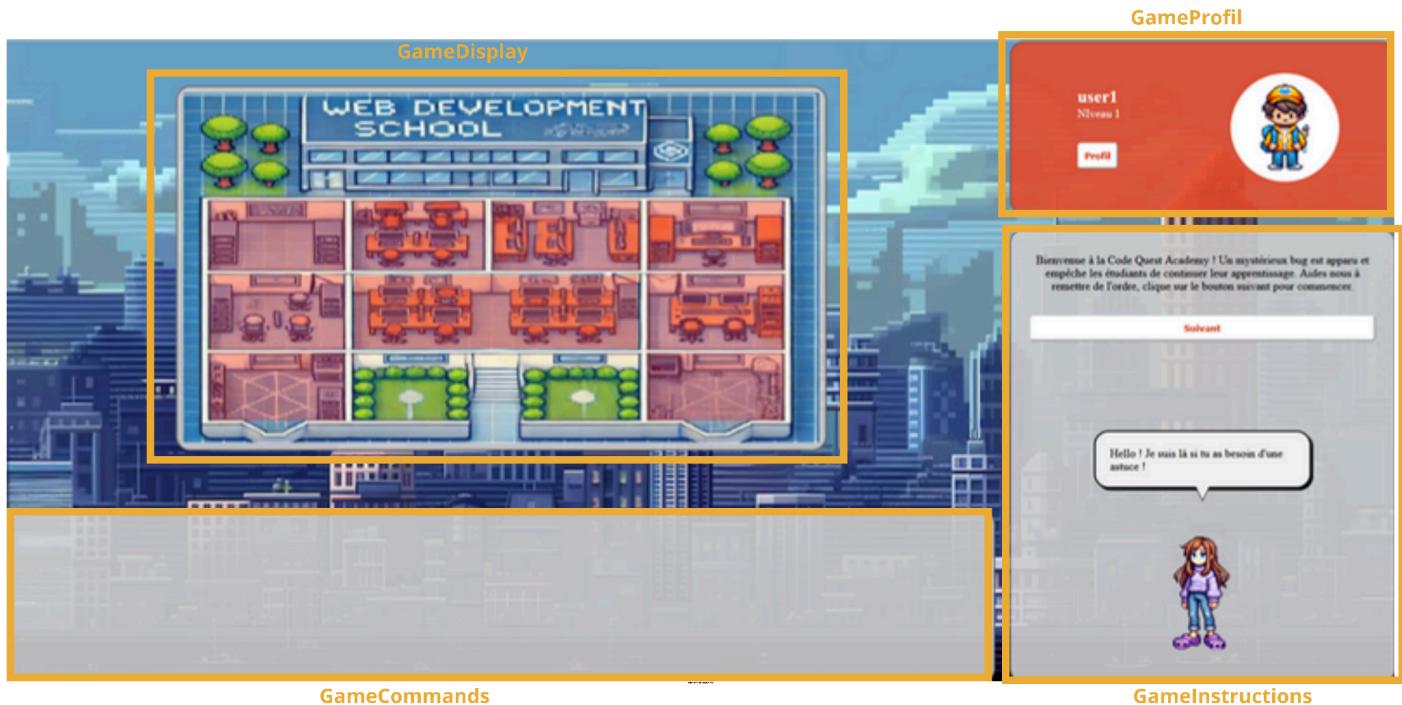
```
useEffect(() => {
  setUsername(user?.username);
  setEmail(user?.email);
  setFirstTeacher(user?.firstTeacher);
  setSecondTeacher(user?.secondTeacher);
  setChallengeId(progress?.challenge_id);
}, [user, progress]);
```

Le hook **useEffect** est utilisé afin de permettre un rendu en direct des modifications.

Cela déclenche un re-render de la page à chaque fois que **user** et **progress** changent.

6.3 - La page de jeu

La page du jeu est découpé en 4 composants :



GameDisplay : Ce composant affiche le visuel du jeu, incluant les images, personnages et vidéos. Chaque salle, associée à un langage, possède un arrière-plan spécifique, un boss et une animation illustrant le combat.

GameProfil : Cette section est dédiée au profil de l'utilisateur. Il y retrouve son pseudo, son niveau, son avatar, ainsi qu'un bouton lui permettant d'accéder à son profil.

GameInstructions : Ce composant fournit plusieurs éléments essentiels :

- Une narration décrivant le contexte du challenge en cours.
- Un bouton "Suivant" permettant de passer au challenge suivant.
- L'image du personnage *FirstTeacher* (par défaut, Fantine, notre formatrice). En cliquant sur cette image, l'utilisateur reçoit une astuce pour l'aider dans le challenge actuel.

GameCommands : Cet espace affiche les questions auxquelles l'utilisateur doit répondre pour progresser dans le jeu.

Excepté le composant GameProfil, le contenu des composants va varier selon les données du challenge en cours. Le bouton "Suivant" sera bloqué tant que l'utilisateur ne trouve pas la bonne réponse, sauf cas spécifiques comme les phases de transition.

Style

Concernant le style de la page, j'ai utilisé **grid** afin de placer correctement les éléments :

```
.gameboard-page {  
    width: 100%;  
    height: 99vh;  
    background: url("../assets/images/background.png") no-repeat center center  
    | fixed;  
    background-size: cover;  
    display: grid;  
    grid-template-columns: repeat(7, 1fr);  
    grid-template-rows: repeat(7, 1fr);  
    grid-column-gap: 0px;  
    grid-row-gap: 0px;  
    gap: 20px;  
}  
  
.gamedisplay-container {  
    grid-area: 1 / 1 / 6 / 6;  
}  
.profil-container {  
    grid-area: 1 / 6 / 3 / 8;  
}  
.instructions-container {  
    grid-area: 3 / 6 / 8 / 8;  
}  
.command-container {  
    grid-area: 6 / 1 / 8 / 6;  
}
```

Fonctionnement

Nous avons vu précédemment que les informations de l'utilisateur étaient récupérées par un contexte “**UserContext**”.

Dans ce même contexte, la progression du joueur est en même temps récupérée via la table progress et stockée dans un état progress :

```
const [progress, setProgress] = useState<ProgressProps | null>(null);  
  
useEffect(() => {  
    if (!user) {  
        return;  
    }  
  
    fetch(`import.meta.env.VITE_API_URL}/api/progress/${user.id}`)  
        .then((response) => response.json())  
        .then((data: ProgressProps | null) => {  
            setProgress(data);  
        });  
}, [user]);  
//
```

Pour les informations de jeu, nous avons créé un deuxième contexte “**GameContext**”, où sont stockées toutes les informations liées au jeu afin d'être réutilisées dans les 4 composants de la page.

```
export const GameContext = createContext<ContextValue | null>(null);

const { user, progress, setProgress } = userContext;

useEffect(() => {
  if (progress && user) {
    fetch(`${import.meta.env.VITE_API_URL}/api/challenge/${progress?.challenge_id}`)
      .then((response) => response.json())
      .then((data: ChallengeProps) => {
        setActualChallenge(data);
      });
  }
}, [user, progress]);
```

D'abord, les informations de l'utilisateur et son progrès sont récupérés du **UserContext**.

On utilise ensuite `useEffect` pour effectuer une requête sur la table challenge dès que les valeurs `user` ou `progress` changent.

Une requête `fetch` est ensuite envoyée sur la route permettant de récupérer le challenge selon la progression de l'utilisateur.

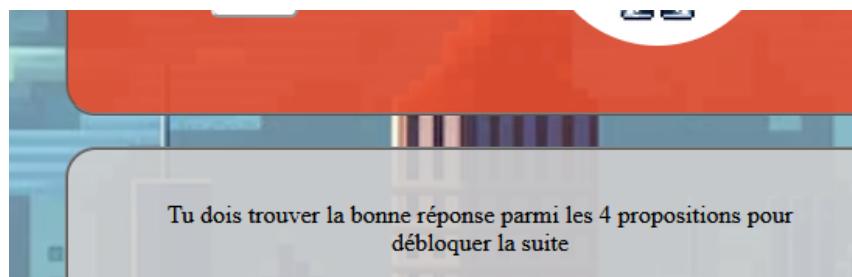
```
router.get("/api/challenge/:id", challengeActions.read);
```

Une fois la réponse reçue, elle est convertie en JSON, puis stockée dans l'état **actualChallenge**.

On l'utilise ensuite directement dans le code afin d'afficher les informations comme par exemple la “guideline” (consigne) :

(Les fichiers utilisent l'extension `.tsx` pour prendre en charge le JSX, une syntaxe qui permet de combiner HTML et JavaScript.)

```
<p className="instructions-text">{actualChallenge.guideline?}</p>
```



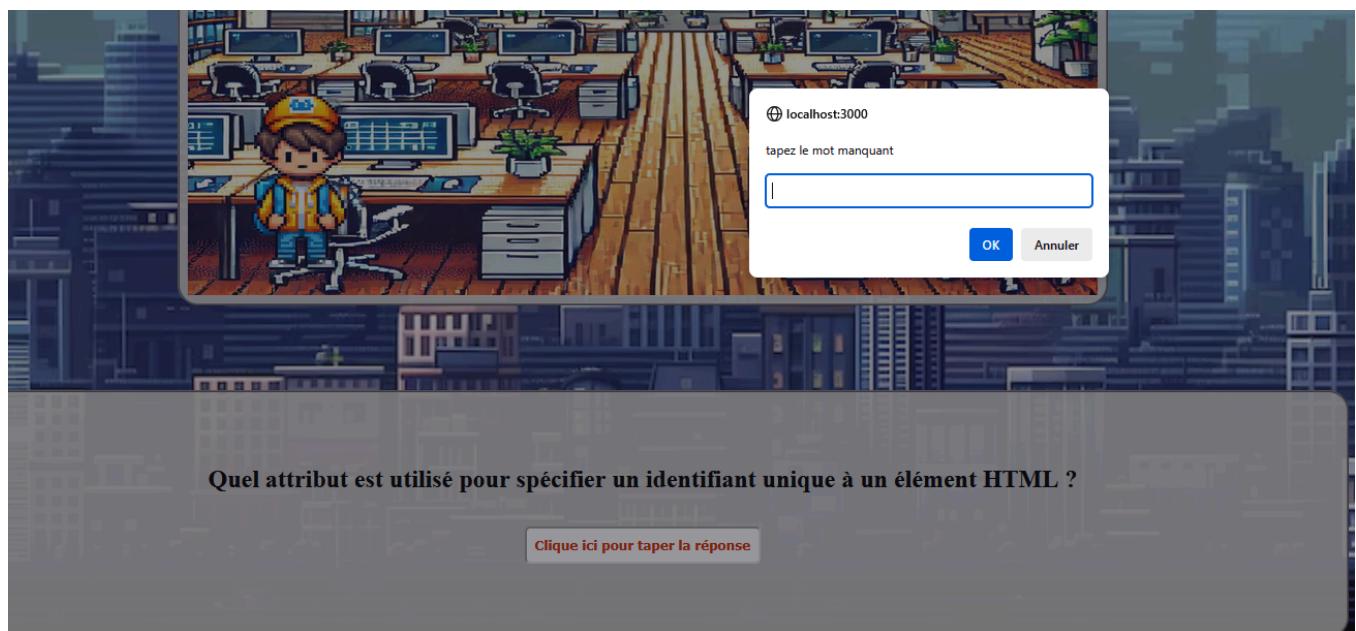
Avancement dans le jeu

Pour progresser dans le jeu, l'utilisateur doit répondre à des questions sous la forme de plusieurs types de jeu :

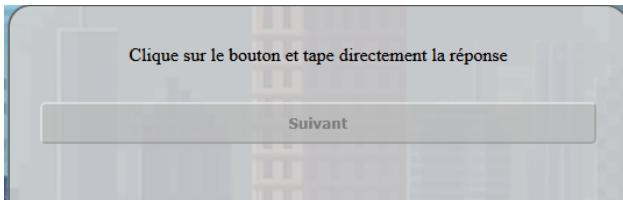
- Un **quizz** avec 4 propositions



- Un **prompt** où l'utilisateur doit taper directement la réponse.



Quand l'utilisateur trouve la bonne réponse, le bouton “**Suivant**” est débloqué et en cliquant dessus, permet à l'utilisateur de passer au challenge suivant.



Le bouton est activé ou désactivé à l'aide d'un état initialisé par un booléen sur “false”:

```
const [isButtonEnabled, setIsButtonEnabled] = useState(false);
```

```
<button
  className={`${instructions-button ${buttonStyles}`}
  onClick={handleProgressUpdate}
  type="button"
  disabled={!isButtonEnabled}
>
  Suivant
</button>
```

La valeur est appliquée dynamiquement sur l'attribut “**disabled**” du bouton.

Si la réponse est correcte, l'état devient “**true**” et rend le bouton cliquable.

Une classe “**buttonStyles**” est en même temps appliquée (grâce à un autre état), permettant de changer le style du bouton.

Dans le fichier CSS :

```
.instructions-button {
  padding: 0.5rem;
  color: #rgb(95, 95, 95);
  background-color: #rgb(185, 185, 185);
  font-weight: bold;
  border-radius: 5px;
  opacity: 0.7;
  width: 90%;
}
```

Le style du bouton de base avec des couleurs grises, montrant le bouton désactivé.

```
.button-enabled {
  color: var(--text-color);
  background-color: #white;
  opacity: 1;
}
```

Une nouvelle classe est appliquée, avec des couleurs vives, montrant le bouton activé.

Le clic sur le bouton exécute la fonction **handleProgressUpdate** :

```
const handleProgressUpdate = async () => {
  setFeedbackMessage("");
  setIsButtonEnabled(false);
  setButtonStyles("");
  setAnswerStyles("");
  setHintVisibility("");
  setIsHintVisible(false);
```

Les états des différents styles sont d'abord réinitialisés, pour éviter que le bouton reste activé à la prochaine question.

```
try {
  const response = await fetch(
    `${import.meta.env.VITE_API_URL}/api/progress/${user?.id}/${progress?.room_id}/${progress?.challenge_id}`,
    {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
    },
  );
```

Une requête **PUT** est envoyée pour modifier la table **progress** en base de données. L'URL est construite dynamiquement avec l'id de l'utilisateur, l'id de la salle (room_id) et celui du challenge (challenge_id).

```
if (!response.ok) {
  throw new Error("Échec de la mise à jour du progrès");
}

const newChallenge = await response.json();

setActualChallenge(newChallenge);
```

Si la mise à jour de la progression échoue, une erreur est envoyée.

Si tout se passe bien, l'état **setActualChallenge** est mis à jour avec les nouvelles données.

```

    await fetchUserProgress();
} catch (error) {
  console.error(
    "Erreur lors de la mise à jour ou récupération du progrès :",
    error,
  );
}

```

Pour finir, une fonction `fetchUserProgress` effectue une requête **GET** afin de récupérer et mettre à jour l'état **progress** avec les nouvelles données.

7 - Jeu d'essai

Fonctionnalité de connexion d'un utilisateur

- **Les données en entrée** : l'action de l'utilisateur sur l'application.
- **Les données attendues** : comment l'application est censée réagir.
- **Les données obtenues** : comment l'application agit réellement.

1 - Les données en entrée

Pour se connecter, l'utilisateur doit saisir une adresse e-mail valide ainsi qu'un mot de passe. Plusieurs cas de figure peuvent se présenter :

- A** - L'utilisateur saisit un email valide et un mot de passe valide
- B** - L'utilisateur saisit un email valide et un mot de passe invalide
- C** - L'utilisateur saisit un email invalide et un mot de passe valide
- D** - L'utilisateur saisit un email invalide et un mot de passe invalide
- E** - L'utilisateur saisit un email vide et un mot de passe valide
- F** - L'utilisateur saisit un email valide et un mot de passe vide
- G** - L'utilisateur saisit un email vide et un mot de passe vide

2 - Les données attendues

- **Cas A** : L'utilisateur est redirigé vers sa page de profil.
- **Cas B, C, D** : Un message d'erreur s'affiche : “*Adresse e-mail inconnue*” ou “*Mauvais mot de passe*”.
- **Cas E, F, G** : Un message d'erreur s'affiche : “*Ce champ est requis*”.

3 - Les données obtenues

- **Cas A** : Conforme aux données attendues.
- **Cas B, C, D** : Conforme aux données attendues.
- **Cas E, F, G** : Un message d'erreur différent apparaît : “*Adresse e-mail inconnue*” ou “*Mauvais mot de passe*” au lieu de “*Ce champ est requis*”.

4 - Analyse des écarts

On constate une divergence pour les cas où un champ est laissé vide. L'application affiche un message d'erreur indiquant une adresse e-mail inconnue ou un mauvais mot de passe, alors qu'elle devrait signaler qu'un champ est requis.

Ce problème est dû à l'absence de l'attribut **required** sur les champs **input** du formulaire, ce qui empêche le navigateur d'effectuer une validation préalable avant l'envoi des données.

5 - Conclusion

Globalement, la fonctionnalité de connexion fonctionne correctement, puisque les utilisateurs peuvent s'authentifier avec des identifiants valides et que les erreurs sont bien gérées dans la majorité des cas.

Cependant, une légère anomalie a été détectée lorsque les champs sont laissés vides : le message d'erreur affiché ne correspond pas exactement à ce qui est attendu. Ce problème peut être facilement corrigé en ajoutant l'attribut **required** aux champs du formulaire.

Malgré cette petite amélioration à apporter, la fonctionnalité reste opérationnelle et conforme aux attentes dans l'ensemble.

8 - Veille sur les vulnérabilités de sécurité

Lors du développement de notre projet, la sécurité a été une préoccupation majeure afin de garantir l'intégrité et la protection des données des utilisateurs. J'ai effectué une veille constante, ce qui m'a permis d'identifier et de corriger plusieurs failles potentielles, garantissant ainsi une application plus robuste et fiable.

Pour cela, j'ai consulté diverses sources d'information, notamment des communautés de développeurs comme **Stack Overflow**, **Reddit**, **GitHub**... etc

J'ai également pris le temps de relire régulièrement les cours et documents vus pendant la formation afin d'appliquer au mieux les bonnes pratiques de sécurité.

Vulnérabilités identifiées et corrigées

Plusieurs mesures ont été mises en place pour renforcer la sécurité de notre application :

- **Authentification sécurisée avec JWT** : Comme vu précédemment, nous avons utilisé JSON Web Tokens (JWT) afin de gérer l'authentification des utilisateurs. Ce système permet de vérifier l'identité de l'utilisateur via un token signé, évitant ainsi l'utilisation de sessions côté serveur et réduisant les risques liés aux attaques par vol de session.

- **Hashage des mots de passe avec Argon2** : Afin de stocker les mots de passe de manière sécurisée, nous avons utilisé Argon2, un algorithme de hachage robuste qui offre une forte résistance aux attaques par force brute et par dictionnaire.

```
const hashPassword: RequestHandler = async (req, res, next) => {
  try {
    // get password from request
    const { password } = req.body;

    if (!password) {
      res
        .status(400)
        .json({ success: false, message: "Le mot de passe est requis" });
    }
    // hash password with hashing option determined sooner
    const hashedPassword = await argon2.hash(password, hashingOptions);
    // replace clear password by hashed password
    req.body.hashed_password = hashedPassword;
    // erase clear password
    req.body.password = undefined;
    next();
  } catch (err) {
    next(err);
  }
};
```

- **Sécurisation des routes front-end** : Pour empêcher l'accès non autorisé à certaines pages de l'application, nous avons mis en place un système de protection des routes côté front-end. Cela empêche un utilisateur malveillant d'accéder aux pages restreintes en tapant directement l'URL dans le navigateur.

```
return (
  <>
  {user?.is_admin !== 1 && (
    <div>
      | Erreur : vous n'avez pas la permission d'accéder à cette page.
    </div>
  )}

  {user?.is_admin === 1 && (
    <>
    <header className="profil-header">
      <img src={logo} alt="Logo" className="logo" />
      <img src={sprite} alt="" className="sprite-admin-page" />
      <button
        | type="button"
        | |
```

9 - Conclusion

La réalisation de Code Quest Academy a été un véritable challenge que nous avons relevé en équipe. Ce projet m'a permis d'approfondir mes compétences en développement web, gestion de projet et design, tout en appliquant concrètement les connaissances acquises durant la formation. Travailler en groupe a été une expérience enrichissante, nous habituant à communiquer, à nous organiser et à surmonter les difficultés ensemble pour mener à bien notre projet.

9.1 - Résumé des acquis et des compétences développées

La réalisation de Code Quest Academy m'a permis de développer de nombreuses compétences, aussi bien sur le plan technique qu'organisationnel.

- **Front-end** : Utilisation de Javascript et Typescript avec React.js, CSS pour styliser et animer l'application.
- **Back-end** : Création de base de données avec MySQL et développement d'APIs sécurisés avec Express.js
- **Gestion de projet** : Organisation du travail avec Jira, utilisation de GitHub pour le versioning et la collaboration en équipe, le tout avec une méthodologie agile.
- **Design** : Création de design et d'animations avec Canva

9.2 - Perspectives d'amélioration

Certaines idées que nous avions imaginées au départ n'ont pas pu être développées par manque de temps, mais restent des pistes intéressantes pour l'avenir.

- **Personnalisation plus poussée** : Offrir à l'utilisateur la possibilité de modifier son avatar de profil, celui de ses formateurs, ainsi que son personnage en jeu, afin de rendre l'expérience plus immersive et unique.

- **Plus de type de jeu** : Enrichir le gameplay en ajoutant d'autres types d'activités, comme la réalisation de requêtes SQL, en plus des quiz et des prompts déjà disponibles.
- **Sécurité du mot de passe accrue** : Renforcer les exigences de création de mot de passe en imposant un nombre minimal de caractères, des chiffres et des caractères spéciaux, afin d'améliorer la protection des comptes utilisateurs.
- **Plus d'interactivité dans le jeu** : Notamment en rendant les combats contre les boss plus stratégiques en intégrant des mécaniques interactives, où l'utilisateur devra réfléchir et prendre des décisions pour remporter la victoire.

9.3 - Récapitulatif et remerciements

La réalisation de Code Quest Academy a été une belle aventure, pleine de défis et d'apprentissages. Ce projet démontre que j'ai su mettre en pratique les connaissances acquises durant la formation, développer de nouvelles compétences et travailler en équipe. C'est une expérience précieuse qui me servira dans mes futures expériences professionnelles et qui m'a conforté dans mon choix de carrière.

Je tiens à remercier chaleureusement ma formatrice, qui a toujours été présente lorsque nous avions un problème, nous guidant avec patience et bienveillance. Un grand merci également à mes collègues, avec qui j'ai partagé non seulement des moments de travail et d'apprentissage, mais aussi beaucoup de fun et de bonne humeur tout au long de la formation.

Merci d'avoir pris le temps de lire ce dossier. J'espère qu'il vous a offert une vision claire du développement du projet et du travail accompli tout au long de cette expérience.

Arnaud GUEVAER

Développeur web et web mobile