



SUARD MICKAEL

Rattrapage
Dossier de projet

Sommaire

Sommaire	2
Compétences couvertes par l'alternance.....	3
Abstract	4
Introduction.....	5
Environnement technique	6
Gestion de projet	7
Création du projet :.....	7
Analyse des besoins.....	9
Spécifications fonctionnelles du projet.....	11
Réalisation du projet.....	16
Frontend	22
Backend.....	33
Base de données.....	39
Sécurité	42
Évolution du Projet	43
Déploiement	44
Veille de sécurité	48
Situation de travail ayant nécessité une recherche	49
Conclusion.....	50

Compétences couvertes par l'alternance

- Maquetter une application
- Développer une interface utilisateur de type desktop
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web
- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données
- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches
- Développer une application mobile
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application

Abstract

Who hasn't experienced the journey of train travel? Convenient, but finding a seat next to a companion can be challenging. Securing the ideal spot next to someone becomes difficult when you only have the car and the seat at your disposal.

I thought about creating a website where we could choose our seats on the train after making a reservation, much like we do for airplanes or cinemas. The idea is to offer people the option to easily select and reserve their seats. We aim to provide more flexibility and give travelers control over their seating arrangements.

The objective of this project is to streamline the reservation process while allowing users to choose where they want to sit, understanding the following:

- Ability to register and securely login.
- Allow users to update their personal information.
- Choose a departure and arrival destination.
- Option to select seats in the train.
- View the history of ticket reservations.

Introduction

Qui n'a pas déjà connu l'expérience des voyages en train ? Pratique, mais trouver s'asseoir à côté d'un proche peut être compliqué. Trouver la place idéale à côté de quelqu'un devient difficile lorsque l'on a seulement la voiture et la place à disposition.

J'ai pensé à créer un site web où on pourrait choisir nos places dans le train après réservation, un peu comme on le fait pour les avions ou les cinémas. L'idée, c'est d'offrir aux gens la possibilité de sélectionner et réserver leurs sièges facilement. Apporter plus de flexibilité et donner aux voyageurs le contrôle sur leur emplacement.

L'objectif de ce projet est de rendre la réservation plus simple tout en offrant aux utilisateurs la possibilité de choisir où ils veulent être, notamment en permettant :

- L'inscription sécurisée et la connexion des utilisateurs.
- La modification des informations personnelles de l'utilisateur.
- La sélection d'une destination de départ et d'arrivée.
- Le choix des sièges dans la voiture du train.
- La consultation de l'historique des réservations de billets.

Environnement technique

Dans le cadre de mon projet de création de site, j'ai fait le choix d'utiliser React.js et Node.js en tant que technologies principales. Mon choix de ces langages n'est pas pour des raisons spécifiques si ce n'est que j'ai eu l'occasion de les découvrir pendant ma formation.

L'utilisation de React.js s'est révélée pertinente dans le développement de l'interface utilisateur. Grâce à son approche basée sur des composants, cette bibliothèque a simplifié la création d'éléments interactifs et dynamiques.

Quant à Node.js, il a été sélectionné pour le développement côté serveur. Cette plateforme JavaScript côté serveur a la particularité d'être performante et de gérer efficacement les opérations asynchrones.

Dans le cadre de la réalisation de mon mémoire, j'ai opté pour l'utilisation de React.js et Node.js en tant que principales technologies. Ce choix repose sur la pertinence de ces langages dans le domaine du développement web, renforcée par mes connaissances durant ma formation.

Gestion de projet

En ce qui concerne la gestion de projet, j'ai consacré du temps à visionner de nombreux tutoriels disponibles sur des plateformes telles que YouTube, dans le but d'approfondir mes compétences en React et Node.js. Mon objectif principal était de développer mes connaissances de la création d'un projet de A à Z, en étant autonome pendant mon temps personnel.

Sachant que j'étais seul pour la création de ce projet. Mon implication dans le développement du projet s'est faite de manière flexible, en fonction de mes disponibilités. Bien que je n'aie pas fait de diagramme de Gantt pour suivre précisément le temps dédié à chaque tâche, Je prenais quand même des notes de ce que je faisais dans la journée, ce qui me permettait de maintenir une trace. Même si ce n'était pas facile de monter un projet tout seul, je me débrouillais en cherchant des réponses sur internet. À chaque problème, je cherchais en ligne pour trouver des solutions. Ça m'a vraiment aidé à approfondir mes connaissances, à tester des choses différentes, et à faire avancer mon projet. Même si bosser tout seul peut être parfois compliqué, internet a été un vrai atout pour résoudre les problèmes. Ça m'a appris qu'avec la bonne démarche et un bon usage des ressources en ligne, on peut trouver des solutions efficaces même face à des problèmes difficiles.

Pour le projet, les différentes parties se sont enchaînées suivant chaque tâche accomplie. Bien qu'il n'y ait pas eu un ordre strict imposé pour réaliser le projet, j'ai tout de même suivi une logique en commençant par la création de la maquette, puis en procédant à la création de la base de données. Ensuite, j'ai développé le formulaire et procédé aux tests d'envoi et de traitement des données.

Création du projet :

En ce qui concerne l'organisation de mon travail, j'ai débuté en utilisant Visual Studio Code comme mon environnement de développement intégré (IDE). J'ai ensuite créé deux dossiers distincts, l'un dédié au backend et l'autre au frontend, afin de bien séparer les deux aspects du projet.

J'ai commencé à préparer le dossier backend en utilisant les commandes :

« `npm init -y` » dans le dossier backend qui permet d'initialiser un projet Node.js en utilisant les valeurs par défaut pour le fichier package.json.

« `npm i mysql express` » permet d'installer les packages MySQL et Express dans le dossier en utilisant le gestionnaire de packages npm. Cela ajoute ces modules à la liste des dépendances du projet, les rendant disponibles pour être utilisés dans le code.

« npm i mysql express » permet d'installer les packages CORS et Nodemon. Ces modules sont utiles pour gérer les problèmes de politiques de même origine et pour redémarrer automatiquement le serveur lors de la modification du code.

Ensuite pour le front end j'ai utilisé la commande dans le dossier frontend

« npx create-react-app nom_de_votre_application »

Cette commande utilise npx pour exécuter la commande create-react-app temporairement, sans avoir besoin de l'installer globalement sur votre machine. Elle crée une nouvelle application React avec le nom spécifié dans le répertoire actuel.

Pour permettre à mon projet de fonctionner en local j'ai utilisé XAMP

XAMPP est un ensemble de logiciels open source qui facilite la création d'un serveur web local sur votre machine pour le développement et les tests. XAMPP offre une solution tout-en-un, vous permettant de configurer rapidement un environnement de développement web sur votre machine locale sans avoir à installer et configurer chaque composant séparément. Il est souvent utilisé pour le développement de sites web locaux avant de les déployer sur un serveur distant.

Pour la sécurité, j'ai implémenté une solution basée sur JWT (JSON Web Token). Je détaillerai davantage cette approche dans la partie dédiée au backend, où elle a été utilisée pour établir un système d'authentification basé sur des tokens.

Analyse des besoins

Le projet vise à simplifier le processus de réservation tout en offrant aux utilisateurs la possibilité de choisir leurs sièges. Les principaux besoins fonctionnels comprennent l'inscription sécurisée, la gestion du profil utilisateur, le choix des gares de départ et d'arrivée, la réservation des billets avec sélection de sièges, la consultation de l'historique des réservations, et la mise en place d'une interface utilisateur conviviale. Le backend, structuré avec Node.js et sécurisé par JWT.

1. Inscription et Connexion :

- **Besoin** : Les utilisateurs doivent pouvoir s'inscrire en fournissant des informations nécessaires.
- **Fonctionnalités** :
 - Formulaire d'inscription sécurisé.
 - Système de gestion des comptes utilisateurs.
 - Connexion avec vérification sécurisée des identifiants.

2. Gestion du Profil Utilisateur :

- **Besoin** : Les utilisateurs doivent pouvoir gérer leurs informations personnelles.
- **Fonctionnalités** :
 - Modification du nom, prénom, et autres détails du profil.

3. Choix de la Destination :

- **Besoin** : Les utilisateurs doivent pouvoir choisir leurs gares de départ et d'arrivée.
- **Fonctionnalités** :
 - Interface conviviale pour la sélection des gares.

4. Réservation des Billets :

- **Besoin** : Les utilisateurs doivent pouvoir réserver des billets et choisir leurs sièges.
- **Fonctionnalités** :
 - Sélection interactive des sièges dans la voiture du train.
 - Validation des réservations.

5. Historique des Réservations :

- **Besoin** : Les utilisateurs doivent pouvoir consulter l'historique de leurs réservations.
- **Fonctionnalités** :
 - Affichage des réservations précédentes.

6. Interface Utilisateur (Frontend) :

- **Besoin** : Proposer une interface utilisateur conviviale.
- **Fonctionnalités** :
 - Utilisation de React.js pour une interface interactive.
 - Composants réutilisables pour une expérience utilisateur cohérente.

7. Serveur (Backend) :

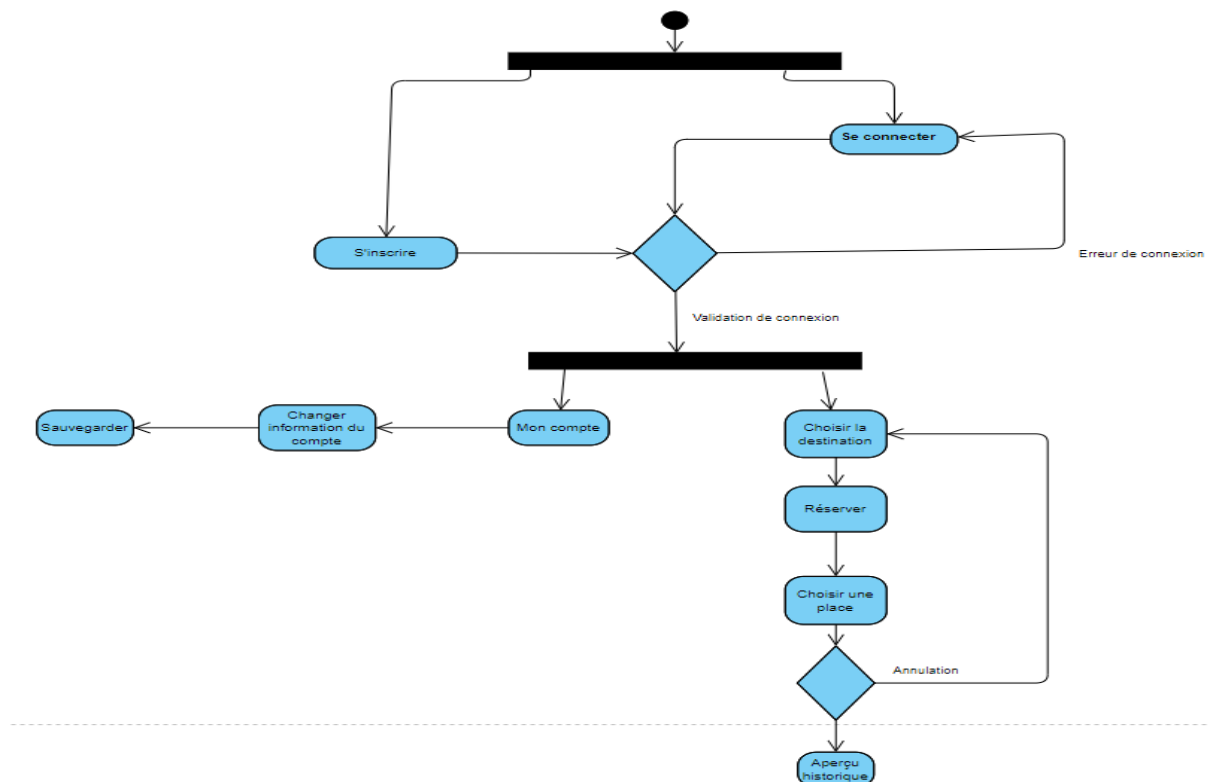
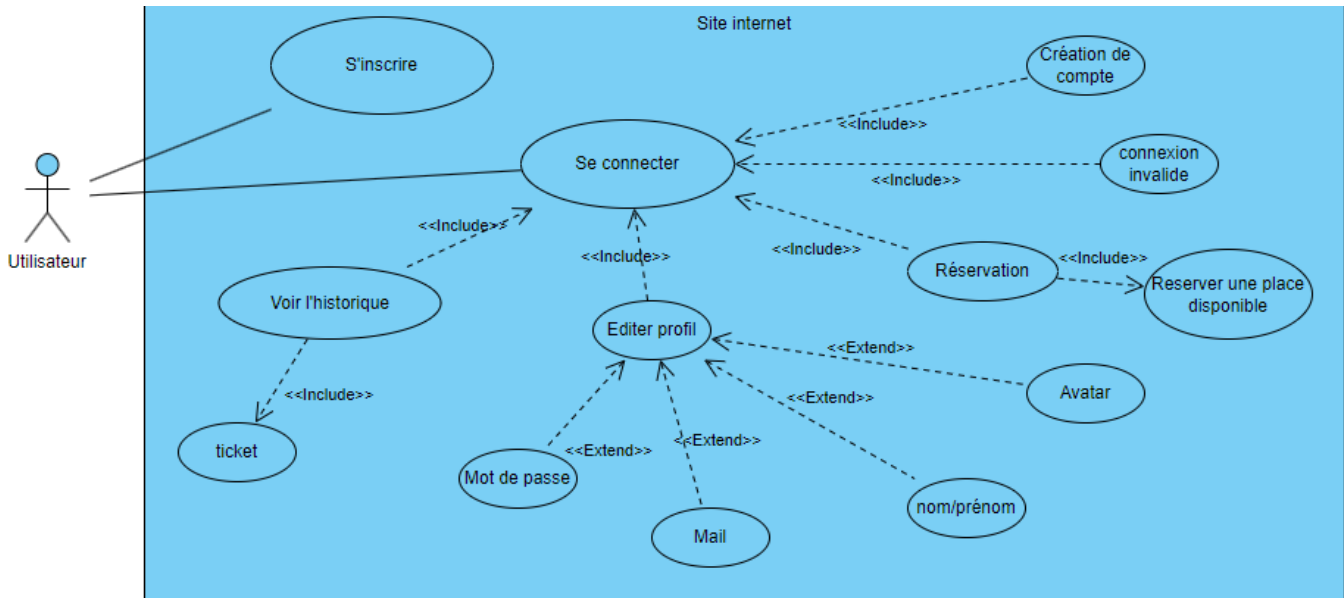
- **Besoin** : Mettre en place un backend robuste pour gérer les opérations côté serveur.
- **Fonctionnalités** :
 - Gestion des requêtes utilisateur.
 - Sécurité avec JWT pour l'authentification.

8. Sécurité :

- **Besoin** : Assurer la sécurité des transactions et de l'authentification.
- **Fonctionnalités** :
 - Utilisation de JWT pour l'authentification basée sur des tokens.

Spécifications fonctionnelles du projet

Pour réaliser cette partie j'ai utilisé Visual Paradigm, cet outil permet de créer des diagrammes, il m'a permis d'établir de nombreux éléments de conception.



Ces diagrammes montrent que l'utilisateur doit être connecté pour accéder au site web. Une fois connecté, il peut visualiser la page d'accueil, explorer son profil pour effectuer des modifications de ses informations ou procéder à une réservation. Après la réservation, il lui est possible de consulter l'historique de toutes ses réservations antérieures.

Diagramme de cas d'utilisation :

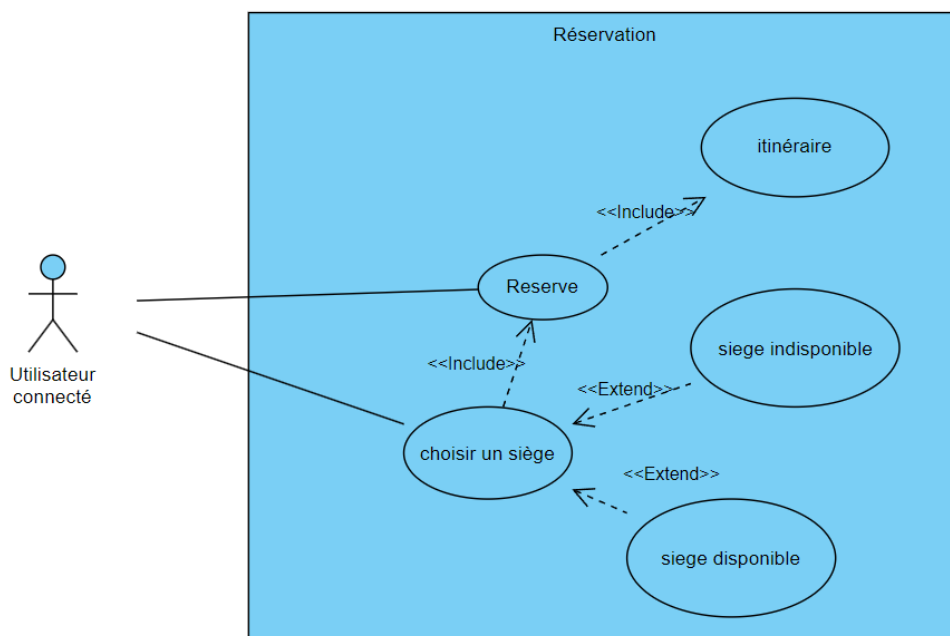
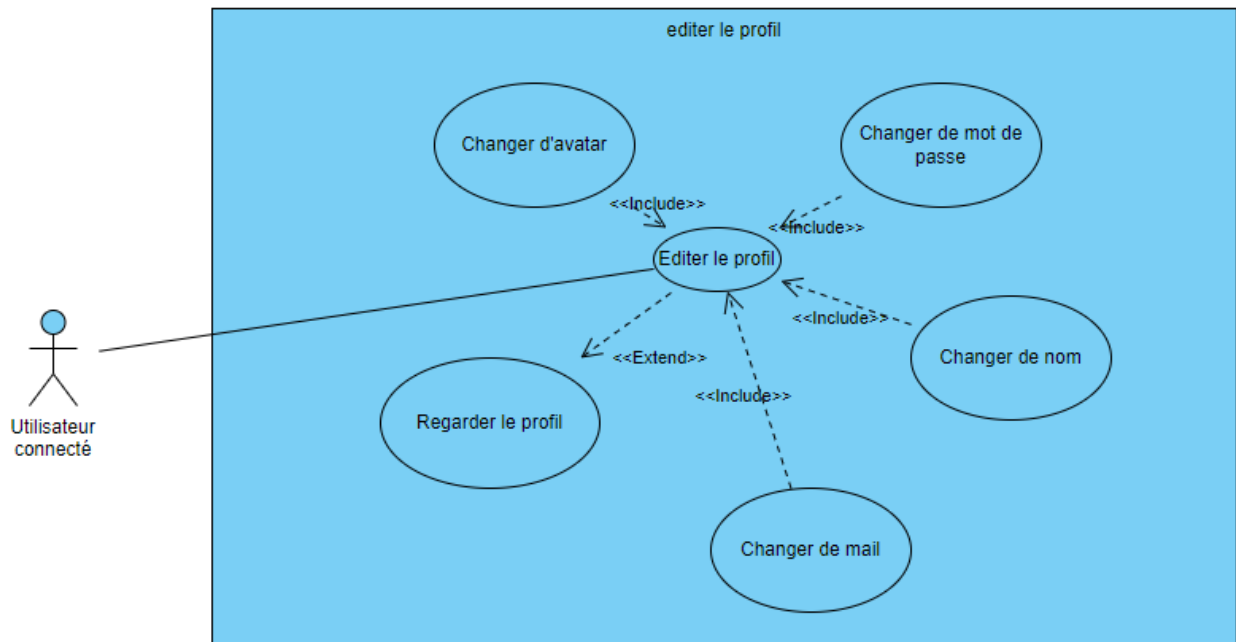
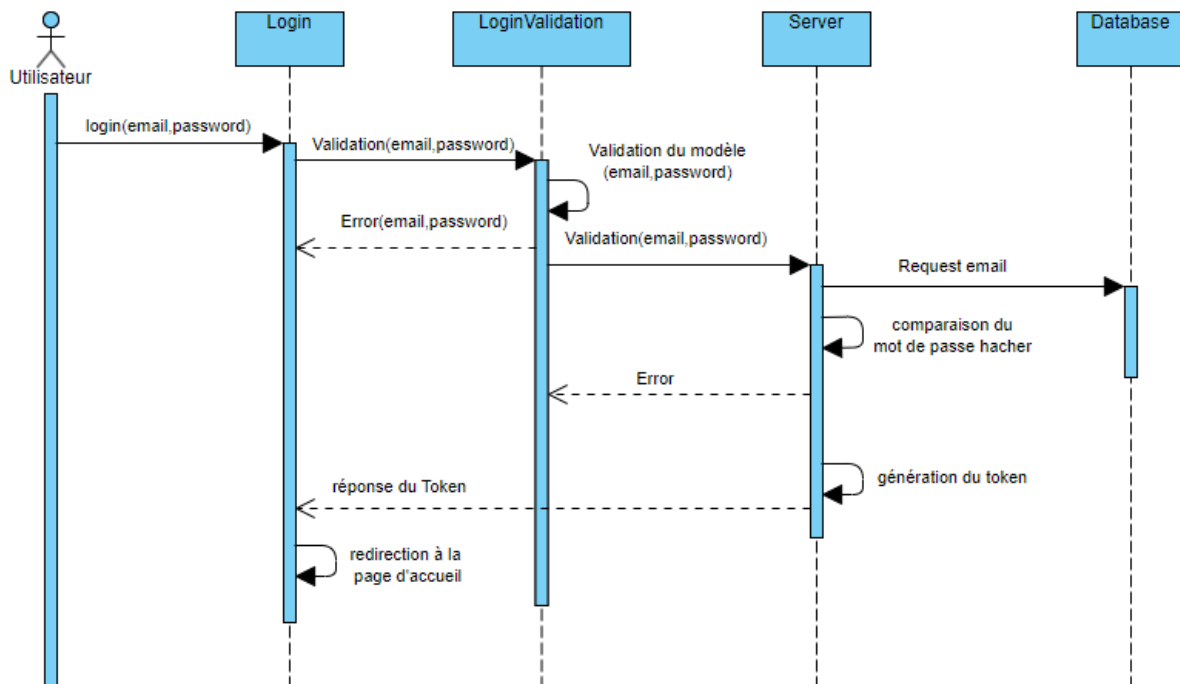


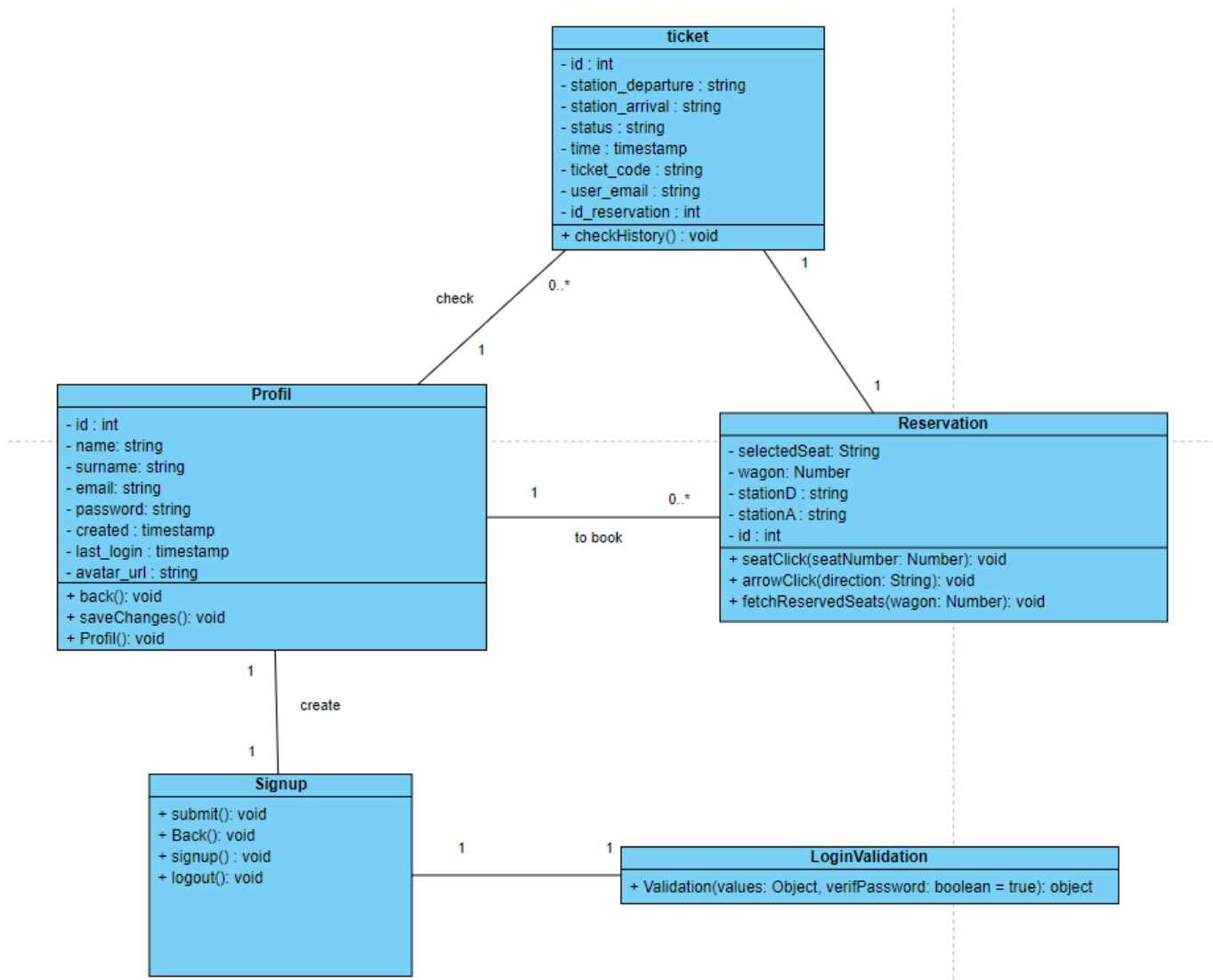
Diagramme de séquence :



Pour expliquer ce diagramme de séquence, une fois que l'utilisateur a saisi son adresse mail et son mot de passe, une validation du modèle de l'adresse mail et du mot de passe est effectuée. Par exemple, le mot de passe doit contenir une majuscule, une minuscule et un chiffre. Si les informations saisies par l'utilisateur sont correctes par rapport au modèle, alors ces informations sont transmises au serveur pour vérifier s'il existe déjà un compte correspondant.

Une fois que le serveur a identifié le compte existant, le mot de passe fourni par l'utilisateur est comparé au mot de passe crypté stocké dans la base de données. En cas de correspondance, un token est généré et associé au compte, comprenant l'adresse mail. Enfin, l'utilisateur est redirigé vers la page d'accueil.

Diagramme de classe :



Le diagramme présenté ci-dessus joue un rôle dans la conception. Il offre une représentation du système en cours de développement. Chaque classe détaille les données dont elle a la responsabilité, ainsi que les opérations qu'elle effectue, tant pour elle-même que pour les autres classes.

Association entre Profil et Ticket :

Nature : Cette association est de type "check", ce qui signifie qu'un utilisateur peut voir 0 ou plusieurs billets, et qu'un billet peut être vu par un profil (Profil "0.." -- "0.." Ticket : check).

Utilité : Un utilisateur peut choisir de posséder plusieurs tickets, et un ticket peut être détenu par plusieurs utilisateurs.

Association entre Ticket et Réservation :

Nature : Cette association est de type "correspond to", ce qui signifie qu'un ticket correspond à une réservation, et une réservation correspond à un ticket (Ticket "1" -- "1" Réservation : correspond to).

Utilité : Chaque réservation est associée à un ticket, et chaque ticket est lié à une réservation unique. Cela permet de suivre l'information selon laquelle un ticket particulier est réservé.

Association entre Profil et Réservation :

Nature : Cette association est de type "to book", ce qui signifie qu'un profil peut réserver zéro ou plusieurs réservations, mais une réservation est effectuée par un et un seul profil (Profil "1" -- "0..*" Réservation: to book).

Utilité : Cette association permet de représenter le fait qu'un utilisateur (profil) peut effectuer plusieurs réservations, mais chaque réservation est attribuée à un seul utilisateur. Ainsi, un utilisateur peut gérer son historique de réservations, et chaque réservation est liée à un profil spécifique qui l'a effectuée.

Association entre Profil et Signup :

Nature : Cette association est de type " associated with" ce qui signifie qu'un profil est associé à une et une seule inscription, et une inscription est associée à un et un seul profil (Profil "1" -- "1" Inscription : associated with).

Utilité : Cette association permet de modéliser la relation entre un profil utilisateur et son processus d'inscription. Chaque utilisateur est associé à une seule inscription, représentant les informations fournies lors de son inscription. D'autre part, chaque inscription est liée à un seul utilisateur, marquant l'association entre les informations d'inscription et le profil correspondant.

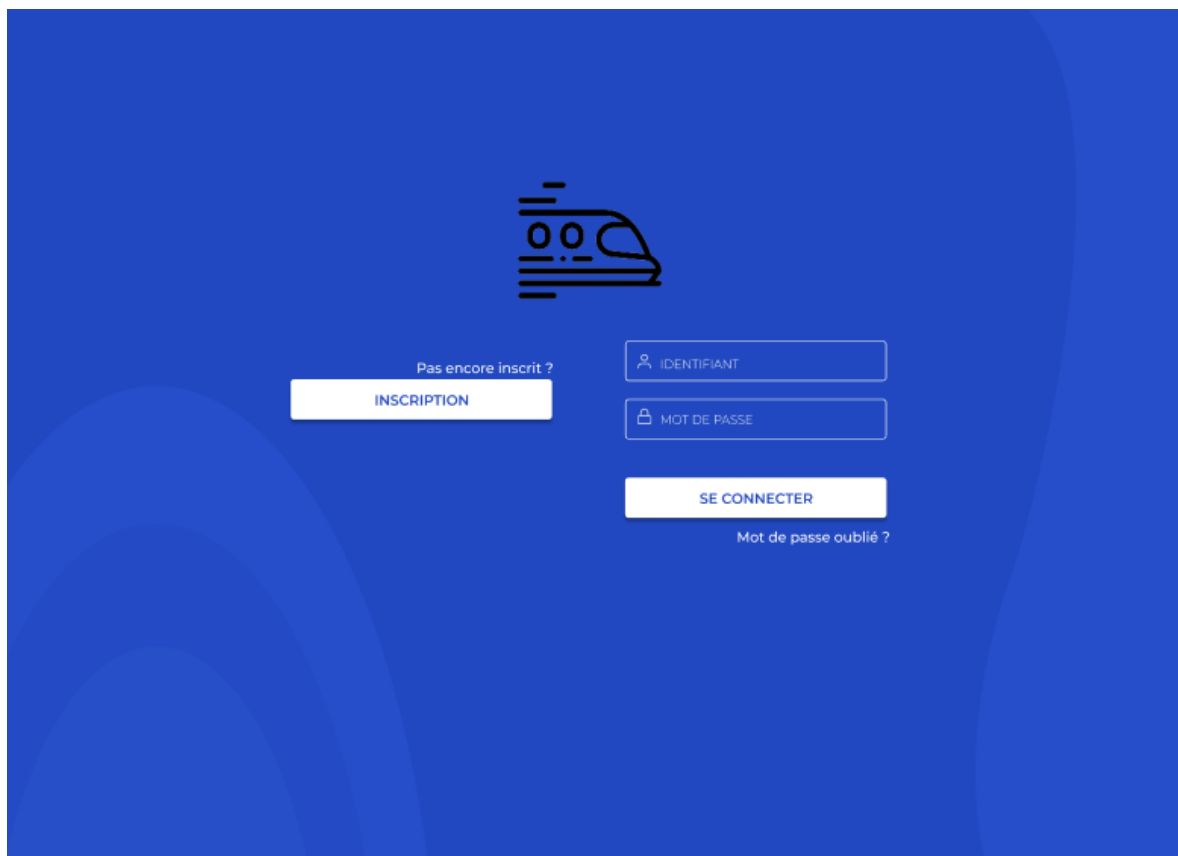
Réalisation du projet

Maquettage :

Pour le maquettage, j'ai opté pour l'utilisation de Figma. Cette plateforme s'est avérée être un outil particulièrement pratique pour concrétiser visuellement ce que j'avais en tête. Figma m'a offert une interface intuitive et des fonctionnalités efficaces, facilitant la création d'une représentation visuelle de mes idées. Grâce à ses nombreuses options de conception, j'ai pu élaborer des maquettes fidèles à ma vision initiale, ce qui s'est avéré essentiel pour clarifier les aspects visuels du projet avant la phase de développement.

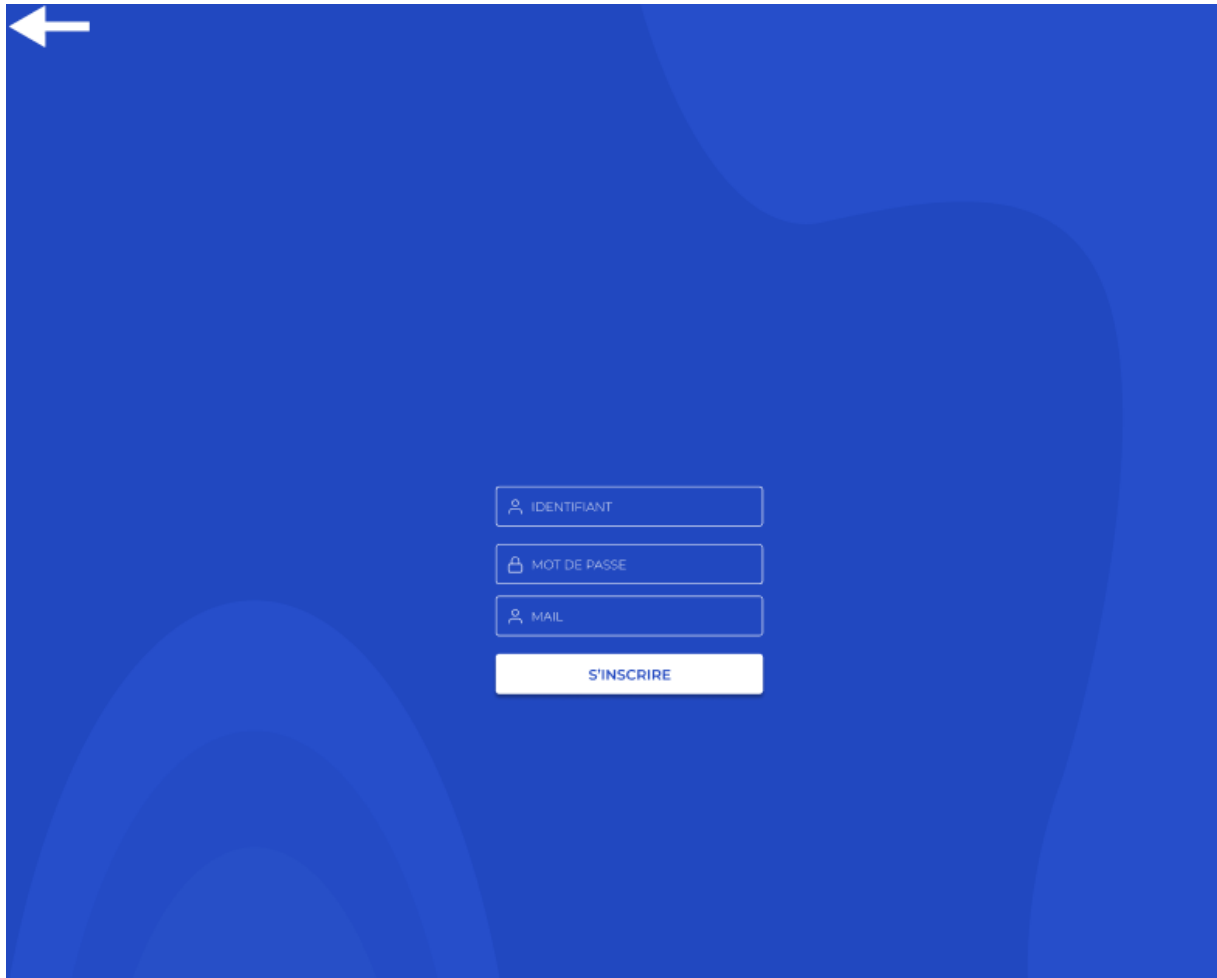
Page de connexion

Ci-dessous se trouve la page de connexion qui offre à l'utilisateur la possibilité de se connecter. En l'absence d'un compte, il peut également accéder à la page d'inscription.



Page d'inscription

La page d'inscription permet de créer un compte en saisissant des informations confidentielles telles que le nom, prénom, adresse mail, et le mot de passe.

A registration page with a solid blue background. In the top-left corner, there is a white left-pointing arrow. Centered on the page are four input fields stacked vertically. The first three fields are outlined in white and contain a small white icon followed by the text 'IDENTIFIANT', 'MOT DE PASSE', and 'MAIL' respectively. The fourth field is a solid white button with the text 'S'INSCRIRE' in blue.

←

IDENTIFIANT


MOT DE PASSE

MAIL

S'INSCRIRE

Page d'accueil

L'utilisateur sera dirigé une fois connecter sur la page d'accueil. Un menu est disponible, offrant l'accès à son compte via le bouton "Mon compte". Un bouton "Se déconnecter" assure une déconnexion sécurisée. Au centre de la page, un bouton "Réserver" permet de passer à la page de réservation, préalablement précédée par la sélection des gares de départ et d'arrivée.

Mon Compte Se déconnecter

Le train, un choix intelligent


pour une mobilité durable pour tous au cœur des territoires.
Ouverture, efficacité et engagement sont nos maîtres mots.

Réservation

Lieu de départ
Lyon

Lieu d'arrivée
Futuroscope

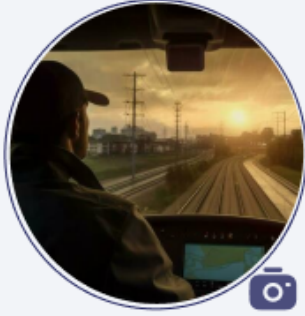
Réserver



Page mon profil

Sur la page "Mon Profil", l'utilisateur connecté peut éditer ses informations personnelles, y compris des détails confidentiels, et également modifier son avatar de profil.

Editer le profil



Identifiant

Nom

Prénom

Email

Mot de passe

Sauvegarder

Page de réservation

La page de réservation s'affiche après que l'utilisateur a sélectionné les gares de départ et d'arrivée depuis la page d'accueil. Les choix de gares sont automatiquement transmis sur cette page. L'utilisateur peut alors choisir son siège parmi les différentes options disponibles. Les sièges sont affichés en deux couleurs distinctes, indiquant leur disponibilité (blanc pour disponible, gris pour indisponible). De plus, l'utilisateur a la possibilité de sélectionner la voiture désirée parmi les huit disponibles.

Il est important de noter que l'utilisateur ne peut pas choisir un siège qui est déjà occupé. Une fois qu'il a fait son choix, les détails de son siège et de sa voiture sélectionnée apparaissent à droite de l'écran pour une confirmation visuelle.

Annuler

Lyon - Futuroscope

☒ Selectionner ☐ Disponible ☒ Indisponible

A	B		C	D
<input type="checkbox"/>	<input checked="" type="checkbox"/>	1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>	4	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input checked="" type="checkbox"/>	5	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>	6	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	7	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>	8	<input checked="" type="checkbox"/>	<input type="checkbox"/>

< Voiture 2 >

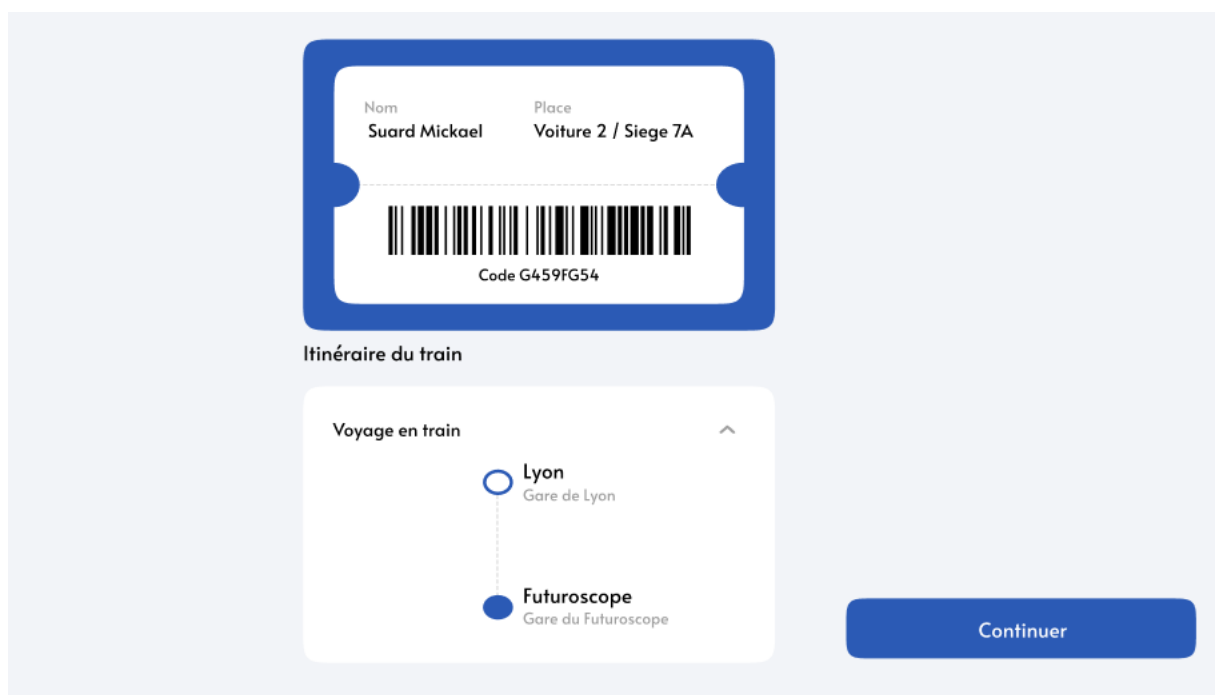
Votre Siege Voiture 5 / Siege 7A

Continuer

Page d'historique

La page d'historique affiche les tickets que l'utilisateur a réservés. Chaque ticket contient des informations spécifiques, notamment les détails de l'utilisateur, le numéro de siège choisi et un code-barres généré de manière aléatoire, garantissant l'unicité du ticket.

Le ticket présente l'itinéraire du train en fonction des gares sélectionnées sur la page d'accueil. Cela offre à l'utilisateur un récapitulatif complet de sa réservation, lui permettant de visualiser rapidement les informations clés liées à son voyage.



Frontend

Une fois le maquettage terminé, j'ai commencé le développement en créant la page de connexion, étant la première interface visible par l'utilisateur. J'avais envisagé d'utiliser Figma avec un plugin permettant de générer le code frontend à partir de nos maquettes. Cependant, bien que le rendu visuel semblait correct, le code généré ne répondait pas à mes attentes. J'ai donc pris la décision de le faire moi-même à partir de zéro en utilisant le CSS que j'avais déjà extrait grâce à Figma.

Page de connexion :

Après avoir achevé la partie visuelle avec la conception des boutons, voici le code qui gère les données à la suite d'un clic sur le bouton de connexion.

```
const handleSubmit = (event) => {  
  event.preventDefault(); // évite le rechargement de la page lorsque le formulaire est soumis  
  setErrors(Validation(values)); // appelle de la fonction permettant de voir si les modèles sont  
  justes  
  
  // si aucun erreur de modele du mail et du mot de passe  
  if (errors.email === '' && errors.password === '') {  
    axios.post('http://localhost:8081/login', values)  
      .then(res => {  
        if (res.data === 'Success') {  
          // Appeler la fonction de gestion de la connexion passée depuis App.js  
          handleLogin();  
          navigate('/home'); //redirection  
        } else {  
          alert("Le compte n'existe pas ou les informations ne sont pas valides");  
        }  
      })  
      .catch((err) => console.log(err));  
  }  
};
```

Cette fonction prend en charge le traitement des valeurs du mail et du mot de passe. Avant d'initier une requête vers le serveur, j'utilise la fonction Validation() pour vérifier la conformité du modèle de mail et du mot de passe.

Par exemple, la fonction vérifie si le mot de passe contient une majuscule, une minuscule et un chiffre. En cas de conformité, elle renverra un message d'erreur vide géré par la fonction setErrors(). En revanche, si un problème est détecté, le message d'erreur correspondant sera retourné.

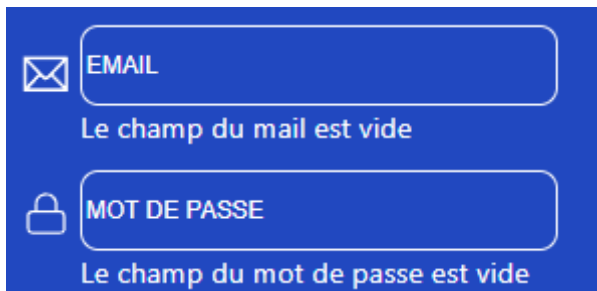
Code de la fonction si dessous

```
function Validation(values, verifPassword = true) {
  let error = {}
  const email_pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/
  const password_pattern = /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*$ / //Password (UpperCase,
  LowerCase and Number)

  // validation mail
  if(values.email==""){
    error.email = "Le champ du mail est vide"
  }
  else if(!email_pattern.test(values.email)){
    error.email = "Email n'est pas valide"
  } else {
    error.email = ""
  }

  // Validation password
  if(verifPassword){
    if(values.password==""){
      error.password = "Le champ du mot de passe est vide"
    }
    else if(!password_pattern.test(values.password)){
      error.password = "Le mot de passe n'est pas valide"
    } else {
      error.password = ""
    }
  }
  return error;
}
```

Exemple si nous avons une erreur



The image shows a blue-themed login interface. At the top left is an envelope icon. Below it is a white rounded rectangular input field containing the text 'EMAIL'. Underneath the input field, the text 'Le champ du mail est vide' is displayed in white. Below this is a white rounded rectangular input field containing the text 'MOT DE PASSE'. To the left of this field is a white padlock icon. Underneath the input field, the text 'Le champ du mot de passe est vide' is displayed in white.

Page d'inscription :

La page d'inscription présente des similitudes avec la page de connexion. Ci-dessous se trouve le code de la fonction responsable de la gestion des valeurs des champs.

```
const handleSubmit =(event) => {
  event.preventDefault();
  setErrors(Validation(values));
  if(errors.name === "" && errors.surname === "" && errors.email === "" && errors.password === "" )
{
  axios.post('http://localhost:8081/signup', values)
    .then(res => {
      if (res.data === 'Success') {
        navigate('/')
      } else {
        alert(res.data);
      }
    })
    .catch(err => console.log(err));
  }
}
```

Avant d'établir une interaction avec le serveur, je m'assure de la validité des modèles du mail et du mot de passe avec la fonction Validation(). En cas d'existence préalable d'un compte associé au même mail saisi par l'utilisateur, une alerte est déclenchée pour informer que le compte correspondant au même mail est déjà enregistré.

Exemple :

L'adresse mail est déjà existant

J'ai développé un composant d'alerte qui affiche des messages d'erreur de manière plus esthétique sur une page, offrant une alternative visuellement agréable à l'utilisation de la fonction de base alert().

```
function Alert({ message, onClose }) {
  useEffect(() => {
    // Efface l'alerte après 5 secondes
    const timeout = setTimeout(() => {
      onClose();
    }, 5000);
    // Nettoie le timeout lorsqu'on quitte le composant
    return () => clearTimeout(timeout);
  }, [message, onClose]);

  return (
    <div className={`alert`} >
      {message}
    </div>
  );
}
```


Le composant utilise `useEffect` pour déclencher une action après que le rendu initial est terminé. À l'intérieur de `useEffect`, un `setTimeout` est utilisé pour programmer la fermeture automatique de l'alerte après 5 secondes.

Lorsque le timeout se déclenche, la fonction `onClose` est appelée. Cela peut être une fonction fournie en tant que prop pour informer le parent de la fermeture de l'alerte.

Le return de la fonction `useEffect` sert à nettoyer le timeout lorsqu'on quitte le composant. Le `clearTimeout` garantit une utilisation efficace des ressources en s'assurant que les alertes sont supprimées de manière fiable après un certain laps de temps, même si l'utilisateur quitte la page.

Le code présenté ci-dessus illustre l'utilisation de la fonction `alert()`. J'ai organisé cette fonction dans un dossier nommé "composant " que j'ai nommé "Alert.js" et placé dans un dossier dédié. Ainsi, il peut être importé et utilisé facilement sur n'importe quelle page du projet.

```
import Alert from './components/Alert'; // Importer le composant d'alerte

const [alertMessage, setAlertMessage] = useState('');
const closeAlert = () => {setAlertMessage('')};

{/* Affichage de l'alerte */}
{alertMessage && <Alert message={alertMessage} onClose={closeAlert} />}
```

importe le composant d'alerte que j'ai créé dans le fichier `Alert.js` situé dans le dossier `components`.

Un état (`alertMessage`) est défini pour stocker le message à afficher dans l'alerte. La fonction `setAlertMessage` est utilisée pour mettre à jour cet état.

Une fonction `closeAlert` est définie pour réinitialiser le message de l'alerte à une chaîne vide, ce qui aura pour effet de fermer l'alerte.

(`{alertMessage && <Alert ... />}`) qui affiche le composant `Alert` uniquement si `alertMessage` n'est pas une chaîne vide. Le composant `Alert` est appelé avec les propriétés `message` et `onClose`.

Page d'accueil :

Une fois l'utilisateur connecté, il est redirigé vers la page d'accueil avec un token associé à son adresse mail. Si l'utilisateur n'est pas connecté et tente d'accéder à la page d'accueil ou à d'autres pages, à l'exception de l'inscription et de la connexion, la page ne s'affichera pas. À la place, un message s'affichera avec un bouton pour retourner à la page de connexion. De même, si l'utilisateur essaie de modifier son jeton généré.

Exemple

vous devez être connecté pour afficher la page

retour

```
function ErrorAuth(){  
  
  const navigate = useNavigate();  
  const Back = useCallback(() => { navigate("/"); }, [navigate]);  
  
  return (  
    <div>  
      <h3>Vous devez être connecté pour afficher la page</h3>  
      <button onClick={() => Back()}>Retour</button>  
    </div>  
  )  
}
```

Le composant ErrorAuth est destiné à être utilisé lorsqu'un utilisateur non connecté tente d'accéder à une page protégée nécessitant une connexion. Il fournit une interface simple pour informer l'utilisateur de la nécessité de se connecter et propose une option de retour à la page d'accueil.

Dans la page App.js, voici le code permettant de sécuriser les pages par un état de connexion

```
// État pour suivre l'état de connexion
const [isLoggedIn, setLoggedIn] = useState(false);

// Fonction pour gérer la connexion de l'utilisateur
const handleLogin = () => {
  setLoggedIn(true);
};
axios.defaults.withCredentials = true; // gestion des cookies , inclure les cookies

useEffect(()=>{
  axios.get(`http://localhost:8081/verify`)
  .then(res => {
    if (res.data.Status === 'Success') {
      setLoggedIn(true);
    } else {
      setLoggedIn(false);
    }
  })
  .catch((err) => console.log(err));
}, []);
```

L'état isLoggedIn est un booléen qui indique si l'utilisateur est connecté ou non.

La fonction setLoggedIn est utilisée pour mettre à jour cet état.

handleLogin est une fonction qui est appelée lorsque l'utilisateur se connecte. Elle met à jour l'état isLoggedIn à true, indiquant que l'utilisateur est connecté.

axios.defaults.withCredentials, Cette ligne configure Axios pour inclure les cookies dans les requêtes, ce qui est important pour la gestion de la session utilisateur.

useEffect est utilisé pour effectuer des actions après le rendu initial du composant. Dans ce cas, il envoie une requête vers http://localhost:8081/verify pour vérifier l'état de connexion côté serveur. Si la réponse indique que l'état est 'Success', l'état isLoggedIn est mis à true, sinon il est mis à false.

Voici ce que retourne le fichier App.js

```
return (  
  <BrowserRouter>  
    <Routes>  
      { /* Page de connexion */  
        <Route path="/" element={<Login handleLogin={handleLogin} isLoggedIn={isLoggedIn} />} ></Route>  
  
        { /* Page d'inscription */  
        <Route path="/signup" element={<Signup />} />  
  
        { /* Page Home - Protégée par l'état de connexion */  
        <Route path="/home" element={isLoggedIn ? <Home /> : <ErrorAuth/>} ></Route>  
  
        { /* Page Profil - Protégée par l'état de connexion */  
        <Route path="/profil" element={isLoggedIn ? <Profil /> : <ErrorAuth/>} ></Route>  
  
        { /* Page Reservation - Protégée par l'état de connexion */  
        <Route path="/reservation" element={isLoggedIn ? <Reservation /> : <ErrorAuth/>} ></Route>  
  
        { /* Page History - Protégée par l'état de connexion */  
        <Route path="/history" element={isLoggedIn ? <History /> : <ErrorAuth/>} ></Route>  
      </Routes>  
    </BrowserRouter>  
  );
```

Chaque composant Route définit une correspondance entre un chemin d'URL spécifique et un composant React à rendre lorsque l'URL correspond.

Par exemple, lorsque l'URL correspond à '/', le composant Login est rendu avec les propriétés handleLogin et isLoggedIn.

Dans chaque route ('/home', '/profil', '/reservation', '/history'), le rendu du composant dépend de l'état de connexion (isLoggedIn).

Si l'utilisateur est connecté (isLoggedIn est true), le composant correspondant (<Home />, <Profil />, <Reservation />, <History />) est rendu.

Sinon, le composant <ErrorAuth /> est rendu, indiquant que l'utilisateur doit être connecté pour accéder à ces pages.

J'utilise React Router pour gérer la navigation. Il définit des itinéraires pour différentes pages, protégeant certaines pages en s'assurant que l'utilisateur est connecté (isLoggedIn est true) avant de leur permettre d'accéder à ces pages. Si l'utilisateur n'est pas connecté, il est redirigé vers la page d'erreur d'authentification (<ErrorAuth />).

Avant de procéder à la réservation, il est nécessaire d'avoir préalablement saisi les valeurs des gares de départ et d'arrivée. Ces valeurs sont incluses dans l'URL de la page de réservation qui permet de récupérer ces paramètres d'URL, par exemple en affichant les gares sélectionnées.

```
const Reserver = useCallback(() => {
  if(stationD.current.value !== "" && stationA.current.value !== ""){
    navigate(`/reservation?departure=${stationD.current.value}&arrival=${stationA.current.value}`);
  } else {
    alert("vous n'avez pas choisi de gare de départ ou d'arrivée");
  }
}, [navigate]);
```

Dans le cas contraire, une alerte sera déclenchée, indiquant qu'aucune gare n'a été sélectionnée

Vous n'avez pas sélectionné de gare de départ ou d'arrivée.

```
const logout = () => {
  axios.get(`${config.URL}/logout`)
    .then(res => {
      navigate("/");
    })
    .catch((err) => console.log(err));
}
```

Dans la page d'accueil, nous retrouverons aussi un bouton permettant la déconnexion sécurisée. Lorsque la fonction logout est appelée, elle envoie une demande de déconnexion au serveur, et si la déconnexion est réussie, l'utilisateur est redirigé vers la page de connexion. En cas d'échec, des erreurs sont affichées dans la console.

Page de réservation :

La page de réservation utilise l'API URLSearchParams pour récupérer les paramètres inclus dans l'URL, ce qui lui permet d'afficher les gares spécifiées. URLSearchParams facilite l'extraction des valeurs des paramètres de requête dans l'URL

```
const queryParameters = new URLSearchParams(window.location.search)
const departure = queryParameters.get("departure")
const arrival = queryParameters.get("arrival")
```

Affichage

```
<div className="station">{departure} - {arrival} </div>
```

```
<div class="conteneur-grid">
  {Array.from({ length: 16 }).map((_, index) => (
    <div
      key={index+1}
      className={`grid-seat ${selectedSeat === index + 1 ? 'selected' : ''}
        ${reservedSeats.includes(index + 1) ? 'reserved' : ''}`}
      onClick={() => handleSeatClick(index+1)}>
      {reservedSeats.includes(index + 1) && ()}
      {index + 1}
    </div>
  ))}
</div>
```

Ce code génère une grille de 16 sièges, chaque siège peut avoir différentes classes ('grid-seat', 'selected', 'reserved') en fonction de son état, et il réagit aux clics en appelant la fonction handleSeatClick avec le numéro du siège correspondant. Si le siège est déjà sélectionné ou réservé, la sélection est annulée sinon le siège est sélectionné.

La fonction `handleSeatClick()` gère la logique de sélection et de désélection des sièges dans la page de réservation en fonction de leur disponibilité. Si un siège est déjà sélectionné, son état est annulé. Si un siège n'est pas déjà sélectionné, il devient le siège sélectionné.

```
const handleSeatClick = (seatNumber) => {  
  // Vérifier si le siège est disponible avant de mettre à jour l'état  
  if (selectedSeat === seatNumber || reservedSeats.includes(seatNumber)) {  
    setSelectedSeat('indéfini');  
  } else {  
    setSelectedSeat(seatNumber);  
  }  
};
```

```
const handleArrowClick = async (direction) => {  
  if (direction === 'up' && wagon < 8) {  
    setWagon((wagonNumber) => wagonNumber + 1);  
  } else if (direction === 'down' && wagon > 1) {  
    setWagon((wagonNumber) => wagonNumber - 1);  
  }  
};
```

Cette fonction permet de changer dynamiquement le numéro de la voiture dans la plage de 1 à 8 en fonction de la direction indiquée par l'utilisateur (haut ou bas), avec des conditions pour s'assurer que le numéro de la voiture reste dans cette plage autorisée.

```
useEffect(() => {  
  fetchReservedSeats(wagon);  
}, [wagon]);
```

`useEffect` est utilisé ici pour s'assurer que la fonction `fetchReservedSeats` est appelée chaque fois que la valeur de `wagon` change, assurant ainsi que les sièges réservés pour la nouvelle voiture sont récupérés et mis à jour correctement.

```

const fetchReservedSeats = async (wagon) => {
  try {
    const response = await axios.get(`http://localhost:8081/reservedSeats?wagon=${wagon}`);
    setReservedSeats(response.data);
  } catch (error) {
    console.error('Erreur lors de la récupération des sièges réservés :', error.message);
  }
};

```

Cette fonction effectue une requête asynchrone pour obtenir les sièges réservés pour une voiture spécifique, met à jour les sièges déjà réservés et gère les erreurs qui pourraient survenir lors de la requête.

```

const handleClick = () => {
  // si aucun siege n'est sélectionné
  if(selectedSeat !== 'indéfini') {
    axios.put('http://localhost:8081/updateSeat', {selectedSeat,wagon} )
      .then(response => {
        // Gere la réponse du serveur ici
        console.log(response.data);
      })
      .catch(error => {
        // Gere les erreurs ici
        console.error('Erreur lors de la requête POST :', error);
      });
  } else {
    alert("vous n'avez pas choisi votre place")
  }
};

```

Cette fonction effectue une requête PUT pour mettre à jour dans la base de données le siège sélectionné, gère les réponses et les erreurs de la requête, et affiche une alerte si aucun siège n'est sélectionné.

Page d'historique :

Actuellement, le développement de la page d'historique n'est pas encore terminé. Sur cette page, les utilisateurs pourront visualiser toutes les réservations effectuées, y compris les détails des sièges choisis. Ils auront aussi la possibilité de supprimer une réservation si nécessaire.

Backend

Au cours de mon développement, j'ai organisé mes fichiers en deux dossiers distincts, l'un dédié à la partie frontend et l'autre à la partie backend.

En ce qui concerne la partie backend de l'application, j'ai opté pour l'utilisation de Node.js. Node.js est un environnement d'exécution JavaScript côté serveur. Il permet de créer des serveurs web et de gérer les opérations côté serveur de manière asynchrone. En utilisant Node.js, j'ai pu mettre en place une infrastructure backend capable de gérer les requêtes HTTP, de se connecter à la base de données et d'effectuer diverses opérations serveur.

Le serveur backend est configuré en utilisant le framework Express.js avec des fonctionnalités supplémentaires pour gérer les requêtes HTTP, les cookies, la sécurisation par token JWT, et la connexion à une base de données MySQL.

Route d'inscription (/signup) :

Lorsqu'un utilisateur soumet un formulaire d'inscription avec son nom, prénom, adresse mail et mot de passe, cette route est appelée. Une requête SQL est utilisée pour vérifier si le mail fourni par l'utilisateur existe déjà dans la base de données. Si le mail existe déjà, une réponse que le mail est déjà existant est envoyée.

```
const checkEmailQuery = "SELECT email FROM profil WHERE email = ?";
db.query(checkEmailQuery, [req.body.email], (checkEmailErr, checkEmailResults) => {
  if (checkEmailErr) {
    return res.json("Error lors du check du mail");
  }
})
```

Si le mail n'existe pas, le code utilise Bcrypt pour générer un hachage sécurisé du mot de passe fourni par l'utilisateur.

```
// Si l'email n'existe pas, continuez avec l'inscription
bcrypt.hash(req.body.password.toString(), salt, (hashErr, hash) => {
  if (hashErr) {
    return res.json("Error lors du mot de passe ");
  }
})
```

Une fois le hachage du mot de passe généré avec succès, une requête SQL est exécutée pour insérer les données de l'utilisateur (nom, prénom, mail et mot de passe haché) dans la base de données. En fonction du résultat de l'insertion, une réponse appropriée est renvoyée au client. En cas de succès, le message "Success" est renvoyé.

```
// Exécution du SQL pour l'inscription
const insertUserQuery = "INSERT INTO profil (name, surname, email, password) VALUES (?)";
const values = [req.body.name, req.body.surname, req.body.email, hash];

db.query(insertUserQuery, [values], (insertErr, data) => {
  if (insertErr) {
    return res.json("Error inserting user");
  }
  return res.json("Success");
});
```

Route de connexion (/login) :

Lorsqu'un utilisateur soumet un formulaire de connexion avec son adresse e-mail et son mot de passe, cette route est appelée. Une requête SQL est utilisée pour récupérer le mail et le mot de passe associé au mail fourni par l'utilisateur dans la base de données. La requête SQL est sécurisée contre les attaques par injection SQL en utilisant des paramètres.

```
const sql = "SELECT email,password FROM profil WHERE email = ? ";
```

Si la base de données renvoie des résultats c'est-à-dire si le mail existe, le code utilise Bcrypt pour comparer le mot de passe fourni par l'utilisateur avec le mot de passe haché stocké dans la base de données.

```
bcrypt.compare(req.body.password.toString(), data[0].password, (err, response) => {
  if (err) {
    return res.json({ Error: "Erreur de comparaison de mot de passe" });
  }
  if (response) {
    const email = data[0].email;
    const token = jwt.sign({email}, "jwt-secret-key",{expiresIn: '1d'});
    res.cookie('token',token);
    return res.json("Success");
  } else {
    return res.json({Error: "Mot de passe incorrect" });
  }
});
```

Selon le résultat de la comparaison des mots de passe, différentes réponses sont renvoyées au client. Si le mot de passe est correct, un jeton JWT est généré et renvoyé dans un cookie HTTP, marquant ainsi la réussite de la connexion.

Si le mot de passe est incorrect, une erreur spécifique est renvoyée. Si le mail n'existe pas, une réponse indiquant une erreur de la connexion est renvoyée.

Middleware :

La fonction middleware verifyUser utilisée pour vérifier la validité du token JWT présent dans les cookies de la requête.

La fonction commence par extraire le token JWT depuis les cookies de la requête.

```
const token = req.cookies.token;
```

Si le token n'est pas présent, que l'utilisateur n'est pas connecté, la fonction renvoie une réponse indiquant que l'utilisateur n'est pas connecté.

```
if (!token) {  
    return res.json({ Error: "Vous n'êtes pas connecté" });  
}
```

Si le token est présent, la fonction utilise la méthode jwt.verify pour vérifier la validité du token en le décodant à l'aide de la clé secrète "jwt-secret-key". Si le token est incorrect, la fonction renvoie une réponse indiquant une erreur liée au token.

```
jwt.verify(token, "jwt-secret-key", (err, decoded) => {  
    if (err) {  
        return res.json({ Error: "Token incorrect" });  
    } else {  
        req.email = decoded.email;  
        next();  
    }  
})
```

Si le token est valide, le mail décodé à partir du token est attaché à la requête (req.email). Cela permet de récupérer l'identité de l'utilisateur dans les routes.

La fonction middleware appelle la fonction suivante next() pour permettre la poursuite du flux de la requête vers la route suivant.

Route de déconnexion (/logout) :

Le code utilise res.clearCookie('token') pour supprimer le cookie du token présent dans la réponse

```
res.clearCookie('token');
```

Cela a pour effet de déconnecter l'utilisateur en supprimant son token d'authentification stocké en tant que cookie sur le côté client. Ensuite, le serveur renvoie une réponse JSON indiquant que la déconnexion a réussi.

Route de récupération (/profil) :

Avant d'accéder à cette route, le middleware `verifyUser` est utilisé pour vérifier si l'utilisateur est connecté en vérifiant la présence du token dans les cookies de la requête.

Si l'utilisateur n'est pas connecté, une réponse JSON avec une erreur est renvoyée. Le mail de l'utilisateur est extrait du token JWT décodé dans le middleware.

```
const userEmail = req.email;
```

Une requête SQL est effectuée pour récupérer les données du profil de l'utilisateur en utilisant son mail. Si une erreur survient lors de la requête SQL, une réponse JSON avec une erreur est renvoyée. En fonction du résultat de la requête, le serveur renvoie une réponse JSON appropriée.

Si l'utilisateur est trouvé, les données du profil (nom, prénom, mail) sont extraites du résultat de la requête et renvoyées dans une réponse JSON.

```
if (result.length === 1) {
    const userData = {
        name: result[0].name,
        surname: result[0].surname,
        email: result[0].email,
        // Ajoutez d'autres propriétés selon votre modèle de données
    };
    res.json(userData);
}
```

Si l'utilisateur n'est pas trouvé, une réponse avec une erreur indiquant "Utilisateur non trouvé" est renvoyée.

S'il y a une erreur qui survient lors de la récupération des données de l'utilisateur, une réponse JSON avec une erreur interne du serveur est renvoyée.

```
if (err) {
    console.error('Erreur lors de la récupération des données de l'utilisateur', err);
    return res.json({ Error: 'Erreur interne du serveur' });
}
```

Route de mise à jour du profil (/updateProfil) :

Avant d'accéder à cette route, le middleware `verifyUser` est utilisé pour vérifier si l'utilisateur est connecté en vérifiant la présence du token dans les cookies de la requête.

Si l'utilisateur n'est pas connecté, une réponse JSON avec une erreur est renvoyée. Les données à mettre à jour, telles que le nom (`name`), le prénom (`surname`), et le mot de passe (`password`), sont extraites du corps de la requête. Le mail de l'utilisateur est déjà extrait du token JWT dans le middleware.

```
const { name, surname, password } = req.body;
```

Si le mot de passe est fourni dans la requête, il est haché avec `bcrypt` avant d'être mis à jour dans la base de données.

```
if (password !== undefined && password !== "") {
  bcrypt.hash(password.toString(), salt, (err, hash) => {
    if (err) {
      return res.json("Error hashing password");
    }

    const values = [
      hash, // mot de passe salé
    ];
    db.query(sql, [hash, userEmail]);
  });
}
```

La requête SQL est utilisée pour mettre à jour le nom et le prénom de l'utilisateur dans la base de données.

```
const updateQuery = 'UPDATE profil SET name = ?, surname = ? WHERE email = ?';
```

La mise à jour est effectuée dans un bloc `try-catch` pour gérer les éventuelles erreurs.

```
try {
  await db.query(updateQuery, [name, surname, userEmail]);

  res.json({ message: 'Profil mis à jour avec succès' });
} catch (error) {
  console.error('Erreur lors de la mise à jour du profil', error);
  res.json({ message: 'Erreur interne du serveur' });
}
```

Route d'insertion des sièges réservés (/updateSeat) :

Avant d'accéder à cette route, le middleware `verifyUser` est utilisé pour vérifier si l'utilisateur est connecté en vérifiant la présence du token dans les cookies de la requête.

Les données nécessaires à l'insertion dans la base de données sont extraites lors de la requête. Dans ce cas, il s'agit du siège sélectionné (`selectedSeat`) et du wagon (wagon) dans lequel le siège est réservé.

```
const { selectedSeat, wagon } = req.body;
```

Une requête SQL est préparée pour insérer les informations du siège réservé dans la table réservation de la base de données.

```
const sql = "INSERT INTO reservation (seat, wagon, user_email) VALUES (?)";  
const values = [selectedSeat, wagon, userEmail];
```

La requête SQL est exécutée avec les valeurs fournies, et le résultat est renvoyé à l'utilisateur. En cas d'erreur lors de l'insertion, une réponse JSON d'erreur, sinon une réponse de succès est renvoyée.

```
db.query(sql, [values], (err, result) => {  
  if (err) {  
    console.error('Erreur lors de l\'insertion dans la base de données : ' + err.message);  
    res.send('Erreur lors de l\'insertion dans la base de données');  
  } else {  
    res.send('Insertion réussie');  
  }  
});
```

Route de récupération des sièges réservés (/reservedSeats) :

Le numéro du wagon (wagon), est extrait de la requête. Une requête SQL est préparée pour récupérer les sièges réservés de la table reservation pour le wagon spécifié.

```
const { wagon } = req.query;  
const sql = 'SELECT seat FROM reservation where wagon= ?';
```

La requête SQL est exécutée avec le numéro du wagon en tant que paramètre. En cas d'erreur lors de la récupération des sièges réservés, une réponse d'erreur est renvoyée. Sinon, une réponse JSON contenant un tableau des sièges réservés est renvoyée.

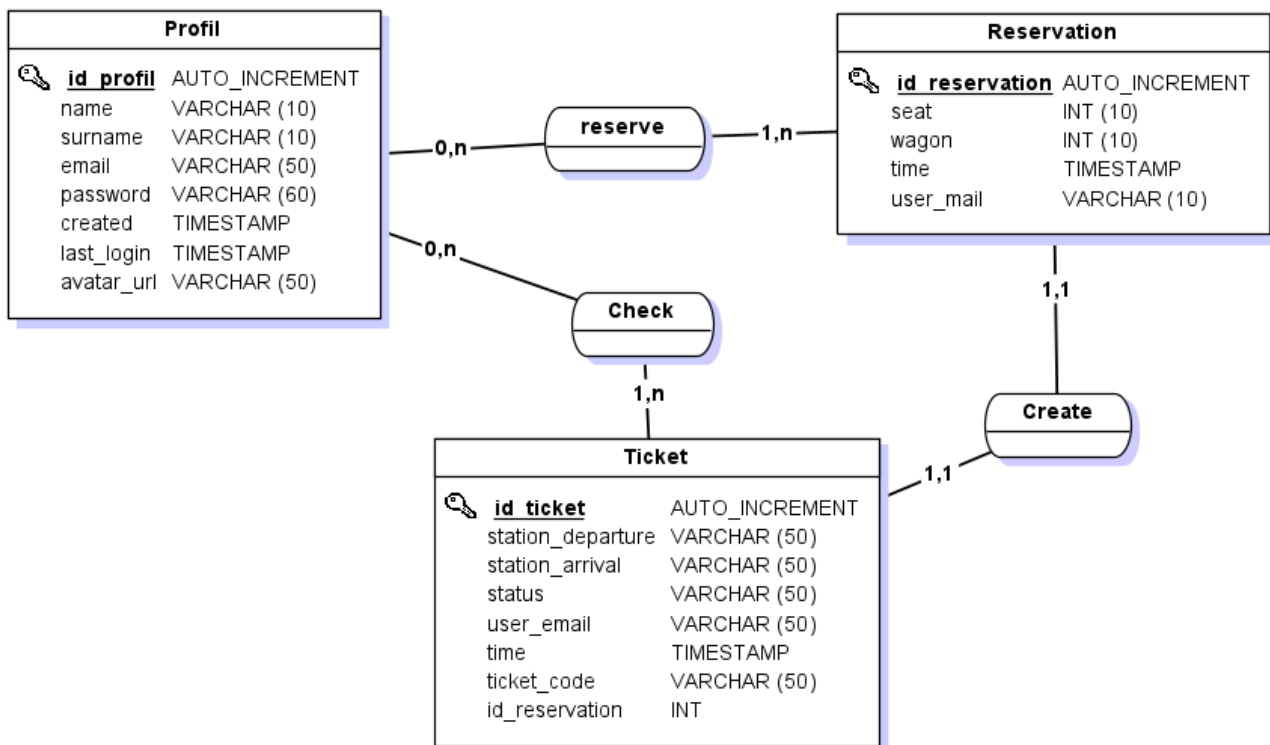
Base de données

En utilisant ces éléments, j'ai élaboré un Modèle Conceptuel de Données MCD avec la méthodologie Merise. Ce modèle finalise la conception et détaille les interactions entre les différentes tables. C'est un des points qui nécessite du temps afin de ne pas avoir à faire la base de données.

Au sein du MCD, divers types d'éléments sont présents, notamment les entités qui rassemblent des objets partageant une nature commune. Ces entités représentent une classe et doivent être identifiables par un identifiant unique.

Les relations représentent des associations de même nature entre deux ou plusieurs d'entités dans le MCD. Une relation peut également posséder des propriétés, également appelées attributs.

Les cardinalités dans le contexte du MCD expriment la participation des occurrences d'une entité aux occurrences d'une relation. Ces cardinalités se composent de termes tels que "0", "1", et "n", formant des paires de termes. Chaque paire de termes représente une entité de la relation et son association respective. Les options de cardinalité se limitent généralement à (0, 1), (0, n), (1, 1), et (1, n). Ces valeurs décrivent le nombre minimum et maximum d'occurrences qu'une entité peut avoir dans la relation.



Les clés primaires sont en gras et soulignées et les clés étrangères en rouge

Grâce au MCD, j'ai pu élaborer le Modèle Logique de Données (MLD). Le MLD représente la structure de la base de données de manière plus concrète, en définissant les tables, les colonnes, les clés primaires, les clés étrangères et les relations entre les différentes entités. Cela permet de détailler davantage la façon dont les données seront organisées et stockées dans la base de données.

Les clés primaires sont en gras et soulignées. Les clés étrangères sont en rouge.

Profil(**id_profil**,name,surname,email,password,created,last_login,avatar_url)

Reservation(**id_reservation**,seat,wagon,time,**#user_mail**)

Ticket(**id_ticket**,station_departure, station_arrival,status,time,ticket_code,**#user_email**, **#id_reservation**)

Convention de nommage :

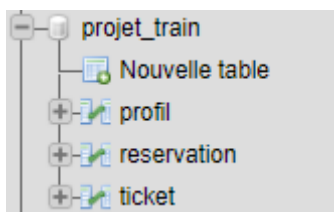
Pour la partie code, les conventions de nommage seront :

- Utilisez la notation CamelCase pour les noms de fonctions et de variables.
- Le nom du fichier doit également être en CamelCase et correspondre au nom du composant.
- Les constantes doivent commencer par une lettre minuscule
- Le nom des variables, fonctions en anglais

Pour la partie Base de données SQL, les conventions de nommages seront :

- Ne pas utiliser les mots réservés (date, delete, add ...).
- Ne pas utiliser de caractères spéciaux.
- Eviter les majuscules, privilégier les underscores pour l'utilisation de deux mots
- Eviter l'utilisation d'abréviation.
- Utiliser un nom représentatif du contenu, utiliser un seul mot si possible, préfixer les noms des tables
- Les tables seront nommées en anglais

Après avoir conceptualisé les schémas des tables et créé le script associé à l'aide de JMerise, j'ai ensuite fait l'implémentation dans le système de gestion de base de données MySQL. Voici ce que j'obtiens



Après avoir configuré les paramètres de connexion à la base de données MySQL, j'ai ensuite procédé à l'établissement de la connexion effective entre le serveur Node.js et la base de données.

```
// Import du module MySQL
const mysql = require('mysql');

// Configuration de la connexion à la base de données
const db = mysql.createConnection({
  host: "localhost",
  user: "user",
  password: "",
  database: "projet_train"
})
```

J'ai établi un profil qui autorise les utilisateurs à effectuer uniquement des opérations SELECT, INSERT et UPDATE sur les tables de la base de données.

Dans la table réservation, J'ai mis en place un déclencheur qui permet à la table "historique" de récupérer toutes les informations, fournissant ainsi une traçabilité complète de toutes les réservations effectuées.

Nom du déclencheur	<input type="text" value="after_reservation_insert"/>
Table	<input type="text" value="reservation"/>
Moment	<input type="text" value="AFTER"/>
Évènement	<input type="text" value="INSERT"/>
Définition	<pre>1 BEGIN 2 INSERT INTO history (id, seat, wagon, time, 3 user_mail) 4 VALUES (NEW.id, NEW.seat, NEW.wagon, NEW.time, 5 NEW.user_email); 6 END</pre>

Sécurité

La sécurité des données est une priorité essentielle dans tout projet informatique. Dans le cadre de mon travail, j'ai pris des mesures pour renforcer la sécurité de l'application en mettant en avant différents aspects.

Pour contrer les attaques par injection SQL, j'ai opté pour l'utilisation de requêtes préparées. Ce choix garantit que les valeurs des variables sont correctement échappées et traitées avant d'être incluses dans les requêtes SQL. Ainsi, toute tentative d'injection SQL est bloquée, renforçant considérablement la sécurité de l'application.

De plus, j'ai pris soin de mettre en place des mesures pour éviter les attaques par XSS (Cross-Site Scripting). Tous les champs de saisie, y compris les champs de saisie invisibles, sont correctement filtrés. Cela signifie que toutes les données entrées par l'utilisateur, sont soumises à un processus de filtrage afin de prévenir toute tentative d'injection de code malveillant ou de saisie de données non conformes. Ce filtrage permet de créer un environnement sûr dans le logiciel tout en garantissant la manipulation sécurisée des données.

La sécurité a été une préoccupation majeure tout au long du développement de l'application. Les requêtes préparées, le contrôle des droits d'accès et la protection contre les attaques par injection SQL et XSS sont autant de mesures que j'ai mises en œuvre pour garantir la sécurité des données et des utilisateurs de l'application.

Évolution du Projet

Le projet en cours n'est pas entièrement terminé car plusieurs idées que j'avais en tête n'ont pas été réalisées faute de temps.

Un exemple est l'absence d'une page permettant la récupération du mot de passe oublié. J'avais l'intention de mettre en place un système où l'utilisateur pourrait recevoir un mail contenant un mot de passe provisoire, lui permettant de se connecter et de changer ensuite son mot de passe.

J'avais également envisagé de mettre l'API de la SNCF dans mon projet qui aurait facilité l'accès à toutes les gares de France, élargissant ainsi les possibilités d'utilisation de ces données dans mon projet.

L'option de personnaliser son image de profil était également prévue, offrant ainsi aux utilisateurs la possibilité de modifier sa photo sur sa page de profil.

En ce qui concerne la page de réservation, mon objectif était de permettre aux utilisateurs de visualiser l'intégralité des billets qu'ils ont réservés, offrant ainsi la possibilité de consulter l'historique complet de toutes leurs réservations.

Je continuerai à améliorer mon projet jusqu'à ce que je sois satisfait et estime avoir concrétisé l'ensemble des éléments que je souhaitais mettre en œuvre dans celui-ci.

Déploiement

Pour déployer mon application, j'ai opté pour un abonnement chez PlanetHoster. Parmi plusieurs options d'hébergeurs disponibles, j'ai choisi PlanetHoster en raison de son prix attractif, et surtout de sa capacité à permettre l'hébergement simultané de plusieurs sites web.

PlanetHoster est un fournisseur de services d'hébergement web et de solutions d'infrastructure en ligne. Ils proposent divers services tels que l'hébergement mutualisé, l'hébergement VPS (serveurs privés virtuels), l'hébergement cloud, ainsi que des solutions de serveurs dédiés.

Après la souscription de l'abonnement, il était nécessaire d'installer l'application Node.js.

Il existait un guide pour faciliter la mise en marche de l'application, bien que son interface ne soit pas intuitive. Malgré cela, j'ai réussi à faire fonctionner mon application Node.js.

<https://kb.n0c.com/knowledge-base/gestion-des-applications-node-js/>

Configuration de l'application Nodejs

VERSION	2
RÉPERTOIRE D'APPLICATION	3
DOMAINE/URL D'APPLICATION	4
FICHIER DE DÉMARRAGE	5
SAUVEGARDER	6 ANNULER

3. Indiquez dans quel **RÉPERTOIRE D'APPLICATION** vous souhaitez que l'application soit créée.
4. Indiquez sous **DOMAINE/URL D'APPLICATION** depuis quelle URL l'application devrait être.
5. Indiquez le nom du **FICHIER DE DÉMARRAGE** (optionnel).

Mon exemple :

RÉPERTOIRE D'APPLICATION*

/home/wdnmbbzg/ backend

URL D'APPLICATION

projettrain.go.yo.fr/ backend

FICHIER DE DÉMARRAGE

server.js

Pour la programmation de l'application j'ai simplement effectué une opération de copier-coller du fichier server.js que j'avais préalablement créé en local. Toutefois, il

était essentiel de prendre en compte que les routes devaient être modifiées en ajoutant /backend, étant donné que le serveur Node.js était hébergé dans le dossier backend.

```
app.post('/backend/signup', (req, res) => { ""})
```

Afin de déployer le frontend sur l'hébergeur, j'ai exécuté la commande "npm run build" et j'avais plus qu'à mettre les fichiers dans l'hébergeur.

La commande "npm run build" est utilisée dans le contexte de Node Package Manager (npm) pour construire et préparer une application pour la production. Elle englobe diverses tâches, notamment la compilation, la minification et l'optimisation des fichiers source, afin de générer une version optimisée et prête à être déployée.

J'ai élaboré un fichier de configuration permettant aux routes de s'ajuster entre l'URL locale et celle de l'hébergeur.

```
const config = {  
  URL: 'https://projettrain.go.yo.fr/backend',  
  //URL : 'http://localhost:8081',  
};  
  
export default config;
```

J'ai généré un fichier .htaccess qui m'a permis de faire des redirections de mes fichiers. Un fichier .htaccess est un fichier de configuration utilisé sur les serveurs web Apache pour définir des règles spécifiques pour un répertoire particulier. Le nom "htaccess" est une abréviation de "hypertext access". Ce fichier permet aux utilisateurs de spécifier différentes directives qui modifient le comportement du serveur web pour le répertoire dans lequel il est placé.

```
RewriteEngine On  
RewriteBase /  
  
# Ne réécrire que si la demande ne commence pas par "/backend/"  
RewriteCond %{REQUEST_URI} !^/backend/  
# Ne réécrire que si le fichier demandé n'existe pas  
RewriteCond %{REQUEST_FILENAME} !-f  
# Ne réécrire que si le répertoire demandé n'existe pas  
RewriteCond %{REQUEST_FILENAME} !-d  
# Réécrire tout le reste vers index.html  
RewriteRule . /index.html [L]
```

Ce fichier .htaccess avec des règles de réécriture est souvent utilisé dans le contexte de projets React, qui sont des Single Page Applications (SPA). Les SPAs ont la particularité de charger une seule page HTML, généralement index.html, et de gérer la

navigation et le rendu des différentes vues côté client, en utilisant JavaScript pour mettre à jour dynamiquement le contenu de la page.

L'utilisation de ce fichier est nécessaire pour garantir que toutes les URL, même celles qui ne correspondent pas à des fichiers ou des répertoires physiquement présents sur le serveur, soient redirigées vers le fichier index.html. Cela est crucial pour assurer le bon fonctionnement du routage côté client dans une application React.

RewriteCond %{REQUEST_URI} !^/backend/ : Évite de réécrire les URL qui commencent par "/backend/", pour permettre l'utilisation du node.js sans erreur.

RewriteCond %{REQUEST_FILENAME} !-f : Assure que la réécriture ne s'applique que si le fichier demandé n'existe pas physiquement sur le serveur. Cela est important pour éviter d'interférer avec les fichiers réels présents sur le serveur.

RewriteCond %{REQUEST_FILENAME} !-d : Empêche la réécriture si le répertoire demandé existe physiquement. Cela garantit que les requêtes vers des répertoires réels sur le serveur ne sont pas redirigées.

Après ces étapes, mon projet est désormais opérationnel via l'URL.

<https://projettrain.go.yo.fr/>

Veille de sécurité

La veille de sécurité pour un projet React est essentielle pour garantir la protection de votre application contre les vulnérabilités et les menaces potentielles. Voici quelques points à prendre en compte dans le cadre de la veille de sécurité pour un projet React :

- Mises à Jour :
Assurez-vous de toujours utiliser la version la plus récente de React ainsi que des bibliothèques tierces. Les mises à jour contiennent souvent des correctifs de sécurité.
- Dépendances NPM :
Surveillez régulièrement les dépendances de votre projet via le registre npm. Les mises à jour peuvent résoudre des problèmes de sécurité.

```
npm outdated
```

- Sécurité des Dépendances :
Utilisez des outils comme npm audit pour identifier et résoudre les problèmes de sécurité dans vos dépendances.

```
npm audit  
npm audit fix
```

- Analyse de Code :
Intégrez des outils d'analyse statique du code pour identifier les vulnérabilités potentielles dans votre code source
- Protection contre les Attaques XSS :
Utilisez des pratiques sécurisées pour protéger votre application contre les attaques Cross-Site Scripting (XSS). Échappez correctement les données avant de les afficher dans le DOM.
- Tests de Sécurité :
Effectuez des tests de sécurité réguliers, tels que des tests d'intrusion et des analyses de vulnérabilités, pour identifier et corriger les éventuelles failles de sécurité.

Situation de travail ayant nécessité une recherche

Afin de réussir ce projet, j'ai commencé par acquérir des compétences en React en suivant des formations disponibles sur YouTube. J'ai commencé en visionnant une vidéo d'une heure de Melvynx, ce qui m'a permis de saisir la manière d'utiliser React ainsi que ses principaux concepts.

<https://www.youtube.com/watch?v=mLZDzZgoCC4&list=WL&index=6&t=1050s>

Par la suite, j'ai consulté divers conseils et astuces provenant de différentes personnes pour apprendre à démarrer avec React, tout en explorant ses avantages et ses inconvénients. Par exemple, j'ai visionné une vidéo où une personne partageait les 10 choses à éviter en travaillant avec React.

<https://www.youtube.com/watch?v=b0lZo2Aho9Y&list=WL&index=5>

Lors de la conception de mes maquettes sur Figma, j'ai entamé mon projet en visionnant une vidéo explicative sur la création sécurisée d'une page de connexion et d'inscription. Cette étape m'a permis de progresser, ce qui m'a ensuite facilité la réalisation autonome des autres pages de mon application.

<https://www.youtube.com/watch?v=F53MPHqOmYI&list=WL&index=4>

J'ai constamment suivi diverses formations et vidéos présentant des projets React ou d'autres langages. Cela me permet de rester à jour, de comprendre les approches utilisées par d'autres développeurs, et de rester informé sur les dernières tendances du domaine.

Conclusion

Ce projet a été une opportunité exceptionnelle pour moi, car il m'a permis de concevoir et réaliser un projet de manière autonome, de sa conception initiale à sa mise en œuvre finale, l'essentiel du travail a été accompli de manière indépendante. Cette expérience m'a offert une formidable occasion de progresser, de trouver des solutions par moi-même et de consolider mes compétences.

Il a une importance particulière pour moi, car il était d'une idée personnelle que j'ai développée avec passion. Travailler sur quelque chose qui était de ma propre créativité a renforcé mon engagement et mon investissement dans chaque étape du développement. C'est une réalisation qui va au-delà du simple achèvement d'un projet technique, car il représente l'expression concrète de ma vision.

Cela témoigne également de ma motivation à ne pas abandonner, même lorsque j'ai dû faire face à des lacunes lors de ma première soutenance. Au lieu de me décourager, j'ai persisté et démontré ma capacité à surmonter les problèmes. Elle a révélé ma volonté de relever les défis, de progresser. Cette démarche a renforcé ma confiance en mes compétences et m'a rappelé que chaque difficulté peut être transformée en une opportunité d'apprentissage et de croissance personnelle.