Documentation Frontend React
CoVoit'Univ

Gabriel Mahalin

7 avril 2025

Table des matières

1 Introduction

Ce document décrit l'organisation et le fonctionnement du frontend en React pour l'application CoVoit'Univ. Il détaille la hiérarchie des composants, leurs rôles et leurs principales interactions.

2 Structure Générale

L'application utilise `React Router` pour définir plusieurs routes et pages principales :

- **Accueil** (`/`) : page d'accueil.
- **Inscription** (`/inscription`) : formulaire pour créer un compte.
- **Connexion** (`/connexion`) : formulaire de connexion.
- **Timeline** (`/TimeLine`) : Vision des trajets à venir.
- **Calendrier** (`/Calendrier`) : vue annuelle, accès aux semaines et aux mois.
- **Interface Conducteur** (`/InterfaceConducteur`) : map + gestion des passagers.

Le point d'entrée principal, `App.js`, englobe l'ensemble des pages dans un `Router` et utilise un `AuthProvider` pour la gestion du contexte d'authentification.

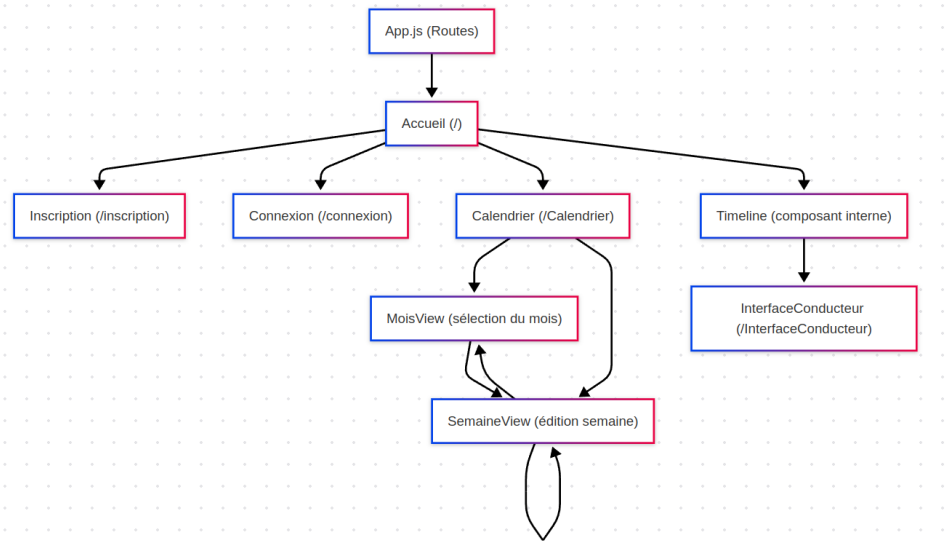

FIGURE 1 – Diagramme d'architecture Front-End

3 App.js

Rôle

- Déclare le `BrowserRouter` (Routes, Route).
- Importe le `AuthProvider` qui fournit les fonctions `login`, `logout` et les informations `userId`.
- Définit la navigation : `Accueil`, `Inscription`, `Connexion`, `ConducteurMap` et `Calendrier`.

Exemple de configuration (extrait)

```
// App.js : Extrait
...
<AuthProvider>
  <Router>
    <div className="App">
      <Header />
      <Routes>
        <Route path="/" element={<Accueil />} />
        <Route path="/inscription" element={<Inscription />} />
        <Route path="/connexion" element={<Connexion />} />
        <Route path="/InterfaceConducteur" element={<ConducteurMap />} />
        <Route path="/Calendrier" element={<Calendrier />} />
      </Routes>
      <Footer />
    </div>
  </Router>
</AuthProvider>
```

4 Contexte d'authentification : `AuthContext.jsx`

Rôle

- Fournit `isAuthenticated`, `userId`, et des méthodes `login`, `logout`.
- Stocke le `token` (pour simplifier, l'ID utilisateur) dans le `localStorage`.

Principe

```
// AuthContext.jsx : Schéma simplifié
export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  // isAuthenticated, userId, login, logout
  // ...
```

```
  return (
    <AuthContext.Provider value={{ ... }}>
      {children}
    </AuthContext.Provider>
  );
};
```

5 Inscription : `Inscription.jsx`

Rôle

- Formulaire pour recueillir l'email, le mot de passe, le nom, l'adresse, et un booléen « conducteur ? ».
- Envoie les données au backend (`/signup`) pour créer un nouvel utilisateur.
- Gère l'affichage des erreurs éventuelles et la validation du formulaire.

Points clés

- `useState` pour gérer les champs de formulaire.
- `fetch` en méthode `POST` vers `/signup`.

6 Connexion : `Connexion.jsx`

Rôle

- Formulaire de connexion : email + mot de passe.
- Requête POST sur `/login` pour valider l'utilisateur.
- Utilisation de `login` depuis `AuthContext` pour stocker le `token`.
- Redirection vers `/Calendrier` en cas de succès.

Technique

```
// Connexion.jsx : Extrait
const { login } = useContext(AuthContext);
...
const handleSubmit = async (e) => {
  e.preventDefault();
  const response = await fetch("/login", {...});
  if (response.ok) {
    login(data.token); // Stockage dans le contexte
    navigate("/calendrier");
  }
};
```

7 Calendrier : `Calendrier.jsx`

Rôle

- Vue annuelle : clique sur un mois pour voir les semaines (`MoisView.jsx`).
- Gère la logique de récupération des données d'emploi du temps pour l'utilisateur courant (`userId`).

Points importants

- Utilise la librairie `dayjs` pour manipuler les dates.
- Vérifie `isAuthenticated` et `userId` pour conditionner l'affichage du contenu.

8 MoisView : `MoisView.jsx`

Rôle

- Liste les semaines du mois sélectionné.
- Permet de naviguer vers `SemaineView` en cliquant sur la semaine.

9 SemaineView : `SemaineView.jsx`

Rôle

- Vue détaillée d'une semaine : heures de départ, heures de retour, choix rôle `conducteur` / `passager`, etc.
- Interaction avec le backend : sauvegarde (`/saveCal`), propagation de la semaine (`/propagateCalendar`), etc.

Points clés

- Utilisation de `useState` et `useEffect` pour charger et stocker les données de chaque jour.
- Gestion du `drag and drop` ou du `mouseEnter` sur les blocs horaires pour définir `startHour` et `endHour`.
- Bouton pour activer / désactiver un jour spécifique.

10 Timeline : `Timeline.jsx`

Rôle

- Affiche en liste les trajets validés par l'utilisateur (prochains jours et semaines).

— Permet de cliquer pour demander un trajet (si passager) ou gérer les passagers (si conducteur).

11 Interface Conducteur : `ConducteurMap.jsx`

Rôle

— Affiche une `Leaflet Map` avec l'itinéraire du conducteur pour un jour donné (aller ou retour).
— Cherche les passagers potentiels (requête POST `/find_passengers`) et propose de leur envoyer des offres.
— Fonctionnalité « itinéraire + waypoints » via `leaflet-routing-machine`.

Points importants

```
// ConducteurMap.jsx : Schéma
useEffect(() => {
  // 1. Créer la carte Leaflet
  // 2. Récupérer la liste des passagers
  // 3. Gérer l'ajout de marqueurs et l'itinéraire
}, []);
```

12 Feuilles de style et assets

`SemaineView.css`

— Définit la mise en page et la gestion du `drag` pour les blocs horaires de la semaine.
— Ajoute des styles `.highlight`, `.disabled-day`, etc.

Autres ressources

— `leaflet.css`, `leaflet-routing-machine.css` : styles pour la carte Leaflet et le routage.
— Icônes et images (marqueurs `.png`) pour `Leaflet`.

13 Conclusion

Le frontend est organisé en composants React distincts, chacun gérant un aspect particulier de l'application (inscription, connexion, calendrier, map conducteur). L'interaction principale avec le backend se fait via `fetch` (`POST`, `GET`) vers l'API Flask, et la gestion d'état d'authentification est centralisée par `AuthContext`.