Documentation technique

Planificateur intelligent – Frontend

Par

Arnaud DUHAMEL

Table des matières

1. Environnement de développement

Le serveur frontend de ce projet est basé sous **React**. Pour développer cette partie, nous avons utilisé des éditeurs de texte classiques, comme **Sublime Text** ou **Notepad++**. Vous pouvez télécharger Sublime Text en suivant le lien suivant :

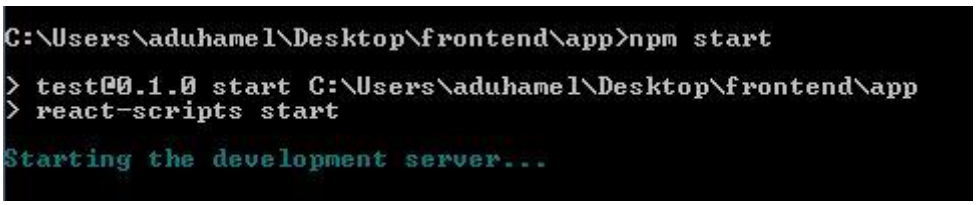
https://www.sublimetext.com/

Afin de télécharger les dépendances nécessaires au bon fonctionnement du projet, il vous faut utiliser **npm**. C'est le **gestionnaire de paquets** de node.js, donc lorsque vous installez Node.js, vous obtenez directement npm.

https://nodejs.org/en/

2. Déployer le serveur front

Pour déployer l'application, il vous suffit de vous placer dans le dossier **/frontend/app** situé à la racine du projet, et d'exécuter la ligne de commande *npm start*.



3. Architecture du front

La structure du front se divise en plusieurs sous-dossiers.

Le dossier **nodes_modules/** contient toutes les dépendances installées avec npm et nécessaires au bon fonctionnement du projet.

Le dossier **Public/** contient une page de support pour le tout, ce qui permet d'avoir un modèle de page HTML, ainsi qu'un manifeste décrivant l'application.

Le dossier **src/** contient tous les fichiers sources. Ils doivent absolument être placés ici, sans quoi le **webpack**, permettant de les compiler, ne les verra pas. Le fichier index.js est indispensable au bon fonctionnement de l'application, car il s'agit du **point d'entrée** appelant les autres composants. Ainsi, tous les composants correspondant aux fonctionnalités (login, register, formulaire, carte) sont placés dans ce dossier.

```
\---App
    |   package.json
    |   README.md
    |
    +---node_modules
    +---public
    |       favicon.ico
    |       index.html
    |       manifest.json
    |
    \---src
        |   App.test.js
        |   index.css
        |   index.js
        |   logo.svg
        |   registerServiceWorker.js
        |
        +---Form
        |       Form.css
        |       Form.js
        |
        +---Login
        |       Login.css
        |       Login.js
        |
        +---Map
        |       Map.css
        |       Map.js
        |
        +---Register
        |       Register.css
        |       Register.js
        |
        \---static
            \---img
                    video.mp4
```

4. Description d'un composant

Un composant se présente sous la forme d'une **classe JavaScript** qui étend la classe *Component* de React. Il possède un constructeur qui appelle le constructeur de la classe dont elle hérite via la méthode **super** et un attribut ***state*** correspondant à l'état interne du composant.

```
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            tags: [],
            add_dep: '',
```

Le state ne se modifie qu'avec l'attribut ***setState*** lorsqu'un évènement se produit (la soumission d'un formulaire par exemple)

```
handleChangeAdd_dep(event) {
    this.setState({add_dep: event.target.value});
}
```

Ensuite, on trouve la méthode render, méthode **indispensable** dans tout composant React. C'est cette fonction qui sera appelée pour rendre le composant. On peut y insérer du code HTML pour créer les entrées nécessaires, par exemple les champs du formulaire.

5. Ajout d'une route

Pour pouvoir aisément naviguer vers les composants créés, il faut intégrer les routes à l'application. Pour cela, il faut modifier le fichier **index.js** situé dans le dossier **src/**.

Il faut importer le composant souhaité de la façon suivante :

```
import App from './App.js';
```

Nous utilisons le package **react-router** et sa librairie **browserHistory** afin de créer chaque route.

```
import {Router, Route, browserHistory} from 'react-router';

  ReactDom.render(
      <Router history={browserHistory}>
          <Route path="/" component={Login}/>
          <Route path="/register" component={Register}/>
          <Route path="/form" component={App}/>
          <Route path="/map" component={Map}/>
          <Route path="/logout" component={Logout}/>
          <Route path="/test" component={Test}/>
      </Router>,
      document.querySelector('#root')
  );
```

6. Reporting

A l'heure actuelle, nous avons créé les composants suivants :

- Login
- Register
- Formulaire
- Map

Les trois premiers composants comportent des formulaires. Ainsi lorsque l'utilisateur soumet ses résultats, nous effectuons une requête de type **fetch** pour communiquer avec l'API.

Cette requête se décompose de la façon suivante.

fetch(url, method, body)

url : l'url de la route créée avec l'API Flask
method : GET ou POST
body : les données à transmettre

Nous pouvons ainsi correctement interroger l'API et nous recevons des réponses correctes.

Au niveau de la carte, nous sommes capables d'afficher un trajet partant d'un point A à un point B avec une ou plusieurs escales. Cependant, nous ne parvenons pas encore à transmettre le résultat de la requête (le tableau comportant chaque du trajet) pour ensuite les afficher sur la carte.

7. Problèmes rencontrés

En plus des problèmes rencontrés pour le développement de la partie back et détaillés dans la documentation consacrée, nous avons d'autres difficultés avec le **proxy**. En effet, il empêche de correctement récupérer la réponse à la requête fetch à l'envoi d'un formulaire. Pour résoudre cela, il faut se mettre en localhost et vous serez capable d'afficher correctement la réponse à la requête au format JSON.

8. Améliorations/modifications

Par rapport à la version actuelle, les fonctionnalités que nous n'avons pas pu mettre en place sont :

- La personnalisation de la carte avec le trajet proposé par l'algorithme
- La personnalisation des marqueurs
- La timeline rappelant les étapes du trajet à côté de la carte

Bien que les possibilités de modifications soient nombreuses, nous vous conseillons de recommencer entièrement le développement de cette partie. En effet, après avoir une première version fonctionnelle entièrement sous Flask, nous avons manqué de temps pour réaliser un travail suffisamment complet avec React.

Ainsi une nouvelle version de l'application sera plus rapide et plus facile à implémenter.