Ajouter une carte mapbox dans une application Web Mobile

Retranscription du cours du mardi 4 mai 2021 sur Discord : partage d'écran, question réponse et livecoding

Introduction

Dans ce cours nous allons apprendre à

- intégrer une carte interactive dans une application Ionic + React
- placer un point d'intérêt sur la carte
- géolocaliser l'utilisateur et afficher sa position
- déclencher un évènement si l'utilisateur est suffisamment près du point d'intérêt

Nous apprendrons aussi

- ce qu'est une variable d'environnement et l'utilisateur d'un `.env` et les bonnes pratiques qui vont avec
- à simuler une position utilisateur pour des besoins de test

Une application d'exemple est disponible sur github à l'url suivante :

https://github.com/CNAM-CPN90-2021/geraud-henrion_treasure-hunt/

Les librairies utilisées seront :

- react-map-gl Une intégration de mapbox-gl pour react
- https://turfjs.org/ Une librairie d'utilitaires d'analyse géospatiale

1. Créer une page dédiée

Dans votre projet Ionic + React, ajouter une page sous la forme d'un composant react dans le fichier `src/pages/Map.jsx`.

```
1  /* src/pages/Map.jsx */
2
3  import {
4    IonHeader,
```

```
  IonToolbar,
  IonButtons,
  IonBackButton,
  IonTitle,
  IonContent,
  IonPage,
} from "@ionic/react";

export function Map() {
  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot="start">
            <IonBackButton />
          </IonButtons>
          <IonTitle>Une super carte</IonTitle>
        </IonToolbar>
      </IonHeader>

      <IonContent fullscreen>
        On metttra une carte ici
      </IonContent>
    </IonPage>
  );
}
```

Puis importer celui-ci depuis `src/App.jsx` et référencez-le comme route de la manière suivante :

```
/* src/App.jsx */

import { Map } from "./pages/Map";

/* ... */

  // parmi les autres routes :
  <Route path="/map" component={Map} />

/* ... */

```

En naviguant sur `localhost:3000/map` vous devriez voir votre nouvelle page.

2. Créer un compte mapbox

Mapbox est un service en ligne qui fournit notamment des tuiles de carte (les images qui constituent les cartes) vectorielles. Le format vectoriel rend possible la personnalisation des cartes, ce qui est parfaitement adapté à notre projet de chasse au trésor (vous trouverez un éditeur et de l'inspiration sur leur site).

Mapbox possède une formule gratuite qui nous conviendra. Il existe aussi des alternatives à ce service :

- google maps : payant lui aussi
- maplibre : fork open source de mapbox
- leaflet : outil open source basé sur open street map

Instructions

1. Créez un compte gratuit sur mapbox
2. Récupérez un jeton d'accès (access token) sur la page d'accueil pour la suite

3. Dotenv / .env

Il est déconseillé de conserver des identifiants et autres données sensibles dans un projet public. Dans beaucoup de systèmes, il est courant de fournir ces données à l'éxécution du programme par l'intermédiaire de variables d'environnement système (voir The Twelve-Factor App).

L'usage d'un fichier `.env` pour stocker ces variables s'est démocratisé dans la plupart des sytèmes modernes.

Dans nodejs ces données sont accessibles via l'objet global `process.env`. Mais dans un contexte de type navigateur ce n'est pas disponible, et ne peut pas être injecté à l'exécution.

Notre outil de développement (webpack) prend ce rôle et injecte cet objet au moment de la transformations des fichiers js, et sera livré avec (attention donc aux données sensibles !).

Instructions

1. Créer un fichier `.env` à la racine du projet
2. Ajouter une ligne `.env` dans votre `.gitignore` pour éviter de l'ajouter à votre projet git
3. Créez un fichier `.env.dist` qui servira de fichier `.env` d'exemple pour vous permettre de démarrer rapidement le projet

Dans le fichier `.env` vous pouvez ajouter une série de variables qui seront injectées à l'objet `process.env`. Règle importante : ces variables doivent être préfixées de `REACT_APP_`

```
# .env

# Variable d'exemple :
REACT_APP_NOM_DE_LA_VARIABLE="contenu de la variable"

# Dans notre cas, nous avons besoin de créer une variable MAPBOX_ACCESS_TOKEN :
REACT_APP_MAPBOX_ACCESS_TOKEN="pk.votre.jeton.d'accès"
```

Rédémarrez votre serveur de dev (`npm start`). Vous pouvez maintenant faire `console.log(process.env)` pour voir le contenu de vos variables !

> Le fichier `.env` est lu au démarrage de la commande `npm start`, pensez donc à bien redémarrer le processus à chaque fois pour que vos changements soient pris en compte

Je conseille fortement de regrouper toute la configuration de l'app dans un seul fichier, duquel vous exportez vos variables d'envrionnement ainsi récupérées.

Cela vous permet de :

1. vérifier leur présence et leur validité
2. les transformer si besoin
3. documenter ce que votre application utilise comme configuration

```
/* config.jsx */

// Regrouper ici toute la configuration de l'app, pour ne la gérer qu'à un seul endroit

export const MAPBOX_ACCESS_TOKEN = process.env.REACT_APP_MAPBOX_ACCESS_TOKEN;

if (!MAPBOX_ACCESS_TOKEN) {
  throw new Error("REACT_APP_MAPBOX_ACCESS_TOKEN is missing on process.env. Please get an access token from mapbox website and add it to your `.env` file at the root of the projet, then restart your dev server.");
}
```

4. Afficher une carte

Maintenant que le projet est prêt et configuré, nous allons afficher une carte centrée sur la ville de Mulhouse dans notre page `/map` à l'aide de la librairie `react-map-gl`

Instructions

Installer react-map-gl via npm

```
npm install --save react-map-gl
```

Dans votre fichier `src/pages/Maps.jsx`, importer le composant `ReactMapGL` installé, ainsi que la feuille de style de mapbox-gl, qui a été installée avec. Puis, au sein de votre page, vous pouvez insérer votre composant :

```
/* src/pages/Maps.jsx */

/* ... */
import ReactMapGL from "react-map-gl";
import "mapbox-gl/dist/mapbox-gl.css";

function Map() {
  // Initialisation d'une position par défaut, aux coordonnées de Mulhouse :
  const [viewport, setViewport] = useState({
    latitude: 47.7395389333945,
    longitude: 7.329169414309033,
    zoom: 12
  });

  return (
    <IonPage>
      {/* ... */}
      <IonContent>
        <div style={{ width: "100%", height: "80vh" }}>
          {/* Utilisation du composant. Il est important de lui fournir des dimensions */}
          <ReactMapGL
            {...viewport}
            onViewportChange={(nextViewport) => setViewport(nextViewport)}
            width="100%"
            height="100%"
          >
            {/* contenu de la carte */}
          </ReactMapGL>
        </div>
      </IonContent>
    </IonPage>
  )
}

```

Et voilà, vous affichez votre première carte interactive !

Vous avez la possibilité d'ajouter votre propre style de carte en fournissant l'url de votre style mapbox en paramètre au composant :

```
<ReactMapGL
  mapStyle="mapbox-url"
  ...
```

Pour en savoir plus, react-map-gl possède une documentation et des exemples.

5. Afficher un point d'intérêt sur la carte

Le composant `Marker` fourni par `react-map-gl` permet de placer n'importe quel composant react à des coordonnées choisies sur la carte. Nous allons l'utiliser pour indiquer une destination à notre utilisateur.

Le marqueur prend plusieurs propriétés

- latitude / longitude : permet d'indiquer où placer ce marqueur
- offsetLeft / offsetTop : permet de spécifier le "centre" du marqueur (ex: la pointe d'une épingle, ou le centre d'un rond)

Instructions

1. Importer le composant `Marker` depuis `react-map-gl`

```
- import ReactMapGL from "react-map-gl";
+ import ReactMapGL, { Marker } from "react-map-gl";
```

2. Insérer le composant `<Marker>` directement dans le composant `<ReactMapGL>` précédemment créé, aux coordonnées choisies

```
<ReactMapGL {...viewport}
  onViewportChange={(nextViewport) => setViewport(nextViewport)}
  width="100%"
  height="100%">

  <Marker latitude={47.7395389333945} longitude={7.329169414309033}>
    {/* contenu du marqueur, peut être n'importe quel composant react */}
  </Marker>
</ReactMapGL>
```

3. Nous allong lui donner une apparence et des dimensions (ici 40x40px) via css

```
<div
  style={{
    background: "red",
    width: "40px",
    height: "40px",
    borderRadius: "50%",
  }}
/>
```

4. Et ne pas oublier de spéficier le "centre" du marqueur via `offsetLeft` et `offsetTop`. Dans notre cas c'est un rond, le centre est donc au milieu de celui-ci, à 20px (40 / 2) depuis le haut à gauche.

```
<Marker
  offsetLeft={(-1 * 40) / 2}
  offsetTop={(-1 * 40) / 2}
  ...
```

Vous devriez avoir un rond rouge de 40x40px au centre de votre carte, au milieu de Mulh

ouse !

6. Géolocaliser l'utilisateur

Nous allons géolocaliser l'utilisateur et afficher sa position sur la carte à l'aide d'un second marqueur. `react-map-gl` fournit un composant qui fait déjà cela, mais qui n'offre pas la même flexibilité dont nous aurons besoin pour simuler la position de l'utilisateur.

Instructions

6.1 Créer un marqueur de position statique

Pour commencer nous allons insérer un second `<Marker>` qui représentera la position de notre utilisateur. Sa position est statique pour le moment.

```
<Marker
  latitude={47.745}
  longitude={7.33}
  offsetLeft={(-1 * 30) / 2}
  offsetTop={(-1 * 30) / 2}
  >
  <div
    style={{
```

```
      background: "blue",
      width: "30px",
      height: "30px",
      borderRadius: "50%",
      border: "white 2px solid",
      boxShadow: "0 0 0 15px rgba(0, 0, 255, 0.4)",
    }}
   />
</Marker>
```

6.2 Géolocaliser l'utilisateur

Ensuite nous allons installer le paquet `@capacitor-community/react-hooks` via npm. Cette librairie est un wrapper qui fournit des hooks react pour les plugins capacitor de base, notamment un hook de géolocalisation qui nous intéresse.

```
# le --force est nécessaire car le paquet exige react16, or react17 est installé.
# mais il n'y a pas de problème de compatibilité entre ces 2 versions, vous pouvez donc forcer l'installation

npm install --save @capacitor-community/react-hooks --force
```

La prochaine étape consiste à importer le hook qui nous intéresse et à le tester

```
/* src/pages/Map.jsx */

/* ... */
import { useWatchPosition } from "@capacitor-community/react-hooks/geolocation";

export function Map() {
  /* ... */
  const { currentPosition, startWatch, clearWatch } = useWatchPosition();

  // cet effet sera exécuté à l'instanciation du composant
  useEffect(
    function startWatchOnMount() {
      startWatch();

      // et celui sera exécuté à la destruction du composant, permettant de libérer de la mémoire
      return function clearWatchOnUnmount() {
        clearWatch()
      }
    },
    [startWatch, clearWatch]
  );
```

```
22
23    // ici nous utilisons l"opérateur `?.` (optional chaining) car currentPosition
      est nullable
24    const userCoordinates = currentPosition?.coords
25    console.log(userCoordinates);
26
27    return (
28      /* ... */
29    )
30  }
```

Enregistrez le fichier, votre navigateur va demander à utiliser votre position, puis vos coordonnées s'afficheront dans la console !

6.3 Positionner le marqueur à la position de l'utilisateur

Nous avons un marqueur et nous connaissons la position de l'utilisateur. La dernière étape consiste simplement à donner les coordonnées de l'utilisateur à notre marqueur (si la position est connue), et le tour est joué !

```
1  {/* ... */}
2  <ReactMapGL>
3    {/* ne pas afficher de marqueur tant que la position de l'utilisateur n'est pas
     connue */}
4    {userCoordinates && (
5      <Marker
6        latitude={userCoordinates.latitude}
7        longitude={userCoordinates.longitude}
8        ...
```

7. Simuler la position de l'utilisateur

Nous avons un problème : nous ne pouvons pas nous déplcer à chaque fois que l'on veut développer une fonctionnalité, de plus qu'on n'est pas tous à Mulhouse en cette période d'épidémie. Nous allons donc devoir simuler une fausse position pour continuer à développer.

Pour cela, nous allons abstraire la logique pour récupérer la position de l'utilisateur dans un premier hook `useRealPosition`, que l'on pourra interchanger avec un 2e hook `useSimulatedPosition`, fournissant tous deux des coordonnées gps.

Instructions

7.1 Extraire notre code dans un hook `useRealPosition()`

Extraire le code utilisant `useWatchPosition()` dans sa propre fonction `useRealPosition`, et en retourner simplement les coordonnées utilisateur :

```
function useRealPosition() {
  const { currentPosition, startWatch, clearWatch } = useWatchPosition();

  useEffect(
    function startWatchOnMount() {
      startWatch();

      return function onUnmount() {
        clearWatch();
      };
    },
    [startWatch, clearWatch]
  );

  return currentPosition?.coords;
}

function App() {
  /* ... */
  const userCoordinates = useRealPosition();
  /* ... */
}
```

7.2 Créer le hook `useSimulatedPosition()`

Copiez-collez le code ci-dessous. Il s'agit d'un hook fournissant une série de position dans le temps, interpolées depuis un point de départ `from` jusqu'à un point d'arrivée `to`.

```
function useSimulatedPosition() {
  const from = { latitude: 47.7426476, longitude: 7.3407563 };
  const to = { latitude: 47.7386289, longitude: 7.3293385 };
  const speed = 0.02; // from 0 to 1
  const refresh = 50; // ms

  const [currentPosition, setPosition] = useState(from);

  useEffect(
    () => {
      const intervalID = setInterval(() => {
        setPosition((pos) => ({
```

```
          latitude: pos.latitude + (to.latitude - pos.latitude) * speed, // formule "lerp"
          longitude: pos.longitude + (to.longitude - pos.longitude) * speed,
        }));
      }, refresh);

      return function onUnmount() {
        clearInterval(intervalID);
      };
    },
    // eslint-disable-next-line react-hooks/exhaustive-deps
    [setPosition]
  );

  return currentPosition;
}
```

Vous pouvez maintenant interchanger vos hooks à loisir dans votre page `Map` :

```
function App() {
  /* ... */
  // const userCoordinates = useRealPosition();
  const userCoordinates = useSimulatedPosition();
  /* ... */
}
```

> Plutôt que de commenter / décommenter un bout de code, ce qui peut créer des erreur, il est conseillé d'activer l'un ou l'autre par rapport à un paramètre global, que l'on peut justement définir dans `src/config.jsx`

8. Calculer la distance entre 2 points sur la carte, et déclencher un évènement

Pour calculer la distance entre 2 points, pythagore pourrait nous aider. Mais comme la Terre est ronde, on doit travailler avec des formules plus complexes, et ce n'est pas non plus simple de convertir des coordonnées GPS en mètre.

C'est pourquoi nous allons installer turfjs qui est spécialisé dans l'analyse géospatialepour nous aider.

Instructions

Installer les paquets `@turf/helpers` et `@turf/distance` à l'aide de npm :

```
npm install --save @turf/helpers @turf/distance
```

Puis importer ceux-ci dans `src/pages/Maps.jsx`

```
import distance from "@turf/distance";
import { point } from "@turf/helpers";
```

Nous sommes maintenant en capacité d'écrire une fonction qui retourne la distance en mètres entre deux coordonnées GPS :

```
// mesure la distance entre 2 points GPS. L'unité de mesure est le mètre
function measureDistance(from, to) {
  // Si l'un des deux points n'est pas défini (ex: on ne connaît pas la positio
nde l'utilisateur), on arrête tout
  if(!from || !to) {
    return Infinity
  }

  return distance(
    point([from.latitude, from.longitude]),
    point([to.latitude, to.longitude]),
    { units: "meters" }
  )
}
```

Utilisons cette fonction pour détecter si l'utilisateur est suffisamment proche du point et déclencher un évènement à ce moment là

```
export function Map() {
  /* ... */
  // nous devons stocker les coordonnées de destination dans une variable afin de
calculer sa distance, ne pas oublier de mettre à jour le marqueur plus bas
  const destination = { latitude: 47.7386289, longitude: 7.3293385 }
  const userCoordinates = useSimulatedPosition()

  const distanceToDestination = measureDistance(userCoordinates, destination);
  const isCloseEnough = distanceToDestination < 100;

  // cet effet sera exécuté uniquement si la valeur de `isCloseEnough` change
entre 2 mises à jour.
  useEffect(() => {
    if (isCloseEnough) {
      alert("Bravo !");
    }
  }, [isCloseEnough]);

  return (
```

```
    {/* ... */}
      // les coordonnées du point d'intérêt sont maintenant stockées dans une variable, ne pas oublier de mettre à jour les propriétés du composant pour refléter ce changement
      <Marker
        latitude={destination.latitude}
        longitude={destination.longitude}
        // ...
  )
}
```

Vous pouvez par exemple choisir de rediriger l'utilisateur à ce moment. En tout cas bravo, votre carte est fonctionnelle. Au prochain cours nous implémenterons le scan de QR code à l'aide de Capacitor.

Bonne journée