Compte rendu programmation web - Client Side

Jason Haenlin

Identifiant GitHub

Pseudo : JasonHaenlin
Url : https://github.com/JasonHaenlin

Tâches effectuées

- mettre en place le CI et Docker pour avoir un déploiement automatique sur mon VPS.
    - CI jenkins en place avec un hook pour déployer automatiquement les changements.
    - Docker pour le client/serveur et aussi global pour build tous les services.
- Ajouter des “loading” sur certaines requêtes qui prennent du temps.
    - Mettre des composants pour montrer que l’élément est en train de charger
- Rendre le site plus responsive
    - Faire en sorte que les graphes (chart, tableau et carte) restent cohérent par rapport à la taille de l’écran
    - Adapter la taille du filtre
    - Changer les composants de la AppBar pour s’adapater sur mobile avec un menu déroulant sur l’écran est plus petit.
- Implémenter une Heatmap pour les régions
    - mettre en place la requête pour récupérer le taux d’incidences par régions
    - Faire la carte avec un SVG et mettre des couleurs en fonction du taux
    - Ajouter un Tooltip quand on survole une région pour avoir davantage d’informations.
- Mettre en place OpenAPI pour générer le document Swagger et générer le contrat pour le client.

Temps de développement

Approximativement 1 jour par semaine.

Stratégie employée pour la gestion des versions avec Git

Concernant la gestion du Git, nous avons fait un Kanban pour écrire les scénarios et tâches pour les réparties dans l’équipe afin de se mettre d’accord des fonctionnalités à faire pour chaque sprint tout au long du projet. Chaque tâche et soumise via une “Pull Request” afin que les autres membres du groupe puissent voir, commenté et conseillé. Cela permet en autre d’avoir une meilleure vision globale du projet même si nous n’avons pas effectué

toutes les tâches. Pour chaque tâche, nous faisions une branche associé, par exemple pour une nouvelle fonctionnalité, la branche s'appelle "feature/US-12" avec 12 le numéro de l'issue. Nous pouvons aussi l'appliqué pour les "task", "tech", "rework" et "bugfix".

Solutions choisies

Bibliothèque de style

Titre	Bibliothèque frontend de composant
Contexte	Pour ce projet, je voulais utiliser une bibliothèque de style pour accélérer le processus de conception et se concentrer sur l'apprentissage et la prise en main de React.
Choix	Nous avons opté pour 3 choix possibles: **Semantic-UI** **Antd** **Material-UI**
Décision	J'ai choisi Material-UI. pour faire mon choix, je me suis basé sur OpenBase en regardant les avis, la popularité et la maintenance du paquet. https://openbase.com/js/@material-ui/core https://openbase.com/js/antd https://openbase.com/js/semantic-ui-react Semantic-UI étant le moins populaire et avec le moins de maintenance, je l'ai enlevé de la liste. Finalement, j'ai préféré prendre Material-UI car déjà, il utilise moins de dépendance externe (12 contre 42 pour Antd). Par ailleurs, selon les avis, Antd est plus compliqué à utiliser. Par rapport à la taille du projet, je préférais utiliser une bibliothèque plus simple et rapide à prendre en main.
Conséquence	Le fait d'avoir mis Material-UI nous contraint à utiliser ses composants material afin de rester consistant dans la manière de développer. Material-UI nous permet aussi de gérer les thèmes à travers de l'application.

Heatmap

Titre	Avoir une visualisation de la France colorisé par région.
Contexte	Il fallait mettre en place une carte avec la possibilité d'appliquer une couleur en fonction du taux d'incidence sur toutes les régions.
Choix	J'avais le choix entre l'API **Google map chart** et **react-simple-maps**
Décision	J'ai opté pour React-Simple-Maps. Google Map est une très bonne API mais a beaucoup de contrainte. Déjà il faut avoir une clé API pour l'utiliser et donc mettre la Carte Bancaire et dans un second cas, il faut

	récupérer les données sur le cloud, ce qui crée une requête de plus dans notre application. React-Simple-Maps permet de créer une carte via un SVG et de coloriser chaque zone du SVG (chaque région) d'une couleur.
Conséquence	Ce choix entraîne quelques difficultés pour trouver un bon SVG avec le bon format. J'ai effectivement fait 3 essaies avant d'avoir la bonne carte. Ensuite, cela ne permet pas de gérer les mises à jour de la carte automatiquement, car le SVG est dans les assets de l'application en brute. Néanmoins, ça économise une requête et je n'ai pas besoin d'enregistrer ma carte bancaire pour l'utiliser.

Requête vers les API

Titre	Faciliter la gestion du contrat entre le client React et le serveur.
Contexte	Au niveau du client, il faut pouvoir requêter notre API
Choix	Il est possible d'utiliser **Fetch**, **Axios** ou une bibliothèque externe tel que **Restful-react**
Décision	J'ai décidé d'utiliser Restful-react car il utilise le principe de hook de react pour les requêtes. Il permet de gérer les requêtes Get et Mutate (post, delete, etc.) un peu comme des **useEffects**. En plus de ça, il est facile de gérer le **loading**, le **refetch** et les **query params**.
Conséquence	Ce paquet offre des hooks très intéressant, mais s'il me faut un comportement différent, il sera peut-être nécessaire d'utiliser Fetch

Difficultés rencontrées

La Heatmap a été une assez difficile à mettre en place. Je suis longtemps resté bloqué sur google map API, mais au vu des contraintes, j'ai préféré changer pour ne plus utiliser une carte mais une image en SVG directement.

MongoDB a été une autre difficulté, car il fallait mettre en forme des données pour la carte, filtrer, regrouper, faire la somme, etc. Étant nouveau sur MongoDB, la prise en main a été longue. Après quelque recherche et avec l'utilisation de **Compass**, j'ai pu élaborer une query fonctionnelle.

Code

Je vais revenir sur 2 composants de l'application
Dans un premier temps, je vais revenir sur le ResponsiveToolbarItem que je trouve bien et adapté. Dans un second temps, je vais parler de GeoMapIncidences qui est selon moi, bien, mais qui m'hérite quelques améliorations.

Composant ResponsiveToolbarItem

Ce composant est pour moi très utile et permet d'utiliser la force de React.
Il est nécessaire pour avoir un rendu responsive sur mobile ou ordinateur en adaptant la AppBar en fonction de la taille de l'écran. Le composant prend un autre composant en paramètre est change la manière dont ils sont affichés pour l'utilisateur. Lorsque le composant est render, un useEffect permet de vérifier la largeur de l'écran et change l'état en fonction. Un eventListener permet de déclencher cette vérification à chaque fois que la l'utilisateur change la taille de l'écran.

```
const displayMobile = (children) => (
  <SimpleMenu>{children}</SimpleMenu>
);
const displayDesktop = (children) => children;

export const ResponsiveToolbarItem = ({ children }) => {
  const [state, setState] = useState({
    mobileView: false,
  });
  const { mobileView } = state;

  useEffect(() => {
    const setResponsiveness = () => {
      return window.innerWidth < 900
        ? setState((prevState) => ({ ...prevState, mobileView: true }))
        : setState((prevState) => ({ ...prevState, mobileView: false }));
    };
    setResponsiveness();
    window.addEventListener('resize', () => setResponsiveness());
  }, []);

  return mobileView ? displayMobile(children) : displayDesktop(children);
};
```

Composant GeoMapIncidences

Ce composant permet d'avoir une carte des régions de la France avec la possibilité d'adapter la couleur en fonction du taux d'incidences de chaque région. Il prend 2 arguments en paramètre, les données de chaque région de la date la plus récente et un setTooltipContent pour mettre à jour le texte de celle-ci en fonction de la région que l'on hover avec la souris. Le point négatif de ce composant est pour moi son manque de flexibilité. Par exemple, pour la gestion de l'intensité de la couleur, j'ai utilisé **ScaleQuantize** pour prendre une couleur selon le taux d'incidences entre 0 et 100, néanmoins, la limite haute a été mise par rapport à l'observation des données, si demain le taux va jusqu'à 1000, alors l'intervalle n'est plus optimal. Ensuite, un point ou j'ai eu du mal est le positionnement

de la carte, j'ai utilisé la propriété **projectionConfig** du composant **ComposableMap** pour rendre la carte plus grande et mieux centrée. Si le SVG change, ces valeurs risques de ne plus être bonnes et nous risquons d'avoir une carte mal positionné, trop grande voire trop petite.

```
const geoUrl = '/assets/gadm36_FRA_1.json';

const colorScale = scaleQuantize()
  .domain([1, 100])
  .range([
    '#ffedea',
    '#ffcec5',
    '#ffad9f',
    '#ff8a75',
    '#ff5533',
    '#e2492d',
    '#be3d26',
    '#9a311f',
    '#782618',
  ]);

export const GeoMapIncidences = ({ data, setTooltipContent }) => {
  return (
    <ComposableMap
      data-tip=""
      projection="geoAzimuthalEqualArea"
      projectionConfig={{
        scale: 2000,
        center: [3, 43],
      }}
    >
      <Geographies geography={geoUrl}>
        {({ geographies }) => geographies.map((geo) => {
          const cur = data.regions.find((s) => mapIdToRegion[s.reg] ===
geo.properties.NAME_1);
          return (
            <Geography
              key={geo.rsmKey}
              geography={geo}
              fill={cur ? colorScale(cur.tx_std) : '#c6c6c6'}
              onMouseEnter={() => {
```

```
                setTooltipContent(`${geo.properties.NAME_1} - ${cur ?
parseFloat(cur.tx_std).toFixed(2) : 'Pas de données'}`);
              }}
              onMouseLeave={() => {
                setTooltipContent('');
              }}
              style={{
                hover: {
                  fill: '#207cd8',
                  outline: 'none',
                },
              }}
            />
          );
        })}
      </Geographies>
    </ComposableMap>
  );
};
```