ECE PARIS
ÉCOLE D'INGÉNIEURS

Outils avancés

2018/2019

APERCU

Dans ce TP, vous découvrirez quelques librairies et fonction javascript bien pratique pour la réalisation de votre projet.

OBJECTIFS

1. Utiliser des routes
2. Intégrer Bootstrap dans votre application
3. Afficher des graphes
4. Envoyer des requêtes API

1. Utiliser des routes

Le routage est le mécanisme par lequel les requêtes (spécifiées par une méthode URL et HTTP) sont routées vers le code qui les gère. Avec React, le routage (par adresse URL) va déterminer quel sera le composant qui sera affiché. Pour effectuer ce routage, vous utiliserez la librairie **react-router** qui permet l'utilisation de composant qui effectuerons le routage pour vous.

Les trois composants principaux de **react-router** vous seront présentés: le composant **BrowserRouter**, le composant **Route** et le composant **Link**. Par exemple:

```
import { BrowserRouter, Route, Link } from 'react-router-dom';
import React, { Component } from 'react';
import './App.css';
import Clock from './Clock/Clock.js';
import Chat from './Chat/Chat.js';

class App extends Component {
  render() {
    return (
      <div className="App">
        <BrowserRouter>
          <div>
            <ul>
              <li><Link to="/">Home</Link></li>
              <li><Link to="/clock">Clock</Link></li>
            </ul>

            <Route exact path="/" component={() => <Chat display={[]}
onSend={()=>{}}/>}/>
            <Route path="/clock" component={() => <Clock date={new Date()}
run={true}/>}/>
          </div>
        </BrowserRouter>
      </div>
    );
  }
}
export default App;
```

Le composant **BrowserRouter** correspond à l'interface de base de la librairie **react-router**. Vous pouvez considérer que tous les autres composants de la librairie (**Route**, **Link**, ...) doivent être contenu dans ce composant pour fonctionner correctement.
Le composant **Route** sert de lien entre une route (une adresse URL spécifique) et un composant qui sera lié à cette route. C'est grâce à ce composant que le composant souhaité sera affiché.
Le composant **Link** permet simplement de créer un lien hypertexte pour accéder à une route particulière.

Documentation: https://reacttraining.com/react-router/web/guides/quick-start

1.1 Routage statique

Une route statique correspond à une route dont l'URL ne changera jamais pour l'affichage d'un composant spécifique. Les routes créées dans l'exemple ci-dessus correspondent à des routes statiques.

1.2 Routage dynamique

Une route dynamique correspond à une route dont un ou plusieurs éléments de l'URL peuvent changer pour l'affichage d'un composant spécifique, cela permet d'utiliser ce qu'on appelle des "variables de routage". Un exemple de route dynamique: les routes "http://example.com/blog/10" et "http://example.com/blog/55" afficheront toutes les deux un article spécifique du blog mais afficheront toujours le même composant "Article" dans le composant "Blog". Ici nous avons donc une variable de routage qui correspond à l'ID de l'article souhaité. Cette variable prendra donc la valeur "10" ou "55".

Pour mettre en place des routes dynamiques, la syntaxe est la suivante:

```
<Route exact path="/blog/:articleId" component={Article}/>
```

Ici une variable de routage nommé **articleId** est transmise dans le composant **Article**. Pour y accéder, il suffit d'aller chercher l'objet **params** dans l'objet **match** dans les propriétés de **Article**:

```
this.props.match.params.articleId
```

Exemple: Pour l'URL /blog/42, **this.props.match.params.articleId** contiendra la valeur **42**.

2. Intégrer Bootstrap dans votre application

Bootstrap est un framework CSS permettant de construire rapidement une interface graphique. On peut considérer Bootstrap comme faisant partie du mouvement artistique Flat Design. Vous utiliserez Bootstrap dans React grâce à la librairie **reactstrap**.

2.1 Le système de grille

L'un des points les plus important de Bootstrap réside dans son système de grille permettant d'organiser les éléments dans une page de manière simple. Le système de grille de Bootstrap utilise une série de **conteneurs**, de **lignes** et de **colonnes** pour mettre en forme et aligner le contenu. Il est construit avec flexbox et est entièrement responsive.

Les **conteneurs** sont les éléments de présentation les plus élémentaires de Bootstrap et sont **obligatoires** pour l'utilisation du système de grille. Choisissez un conteneur responsive de largeur fixe (ce qui signifie que sa largeur maximale change en fonction du type d'écran) ou de largeur fluide (ce qui signifie qu'il prend tout le temps 100% de la largeur de l'écran). Bien que les conteneurs puissent être imbriqués, la plupart des mises en page ne nécessitent pas de conteneur imbriqué. Les **conteneurs** permettent de centrer et de remplir horizontalement le contenu de votre site. Utilisez le composant **Container** pour une largeur de pixel sensible ou définissez la propriété **fluid** du composant à "true" pour une largeur de 100%, quelle que soit la taille de la fenêtre et du périphérique.

Les **lignes** sont des wrappers pour les **colonnes**. Chaque **colonne** a un remplissage horizontal (appelé "gouttière") pour contrôler l'espace entre elles. Ce remplissage est neutralisé sur les **lignes** avec des marges négatives. De cette manière, tout le contenu de vos **colonnes** est aligné visuellement sur le côté gauche. Dans une présentation en grille, le contenu doit être placé dans des **colonnes** et seules les **colonnes** peuvent être des enfants immédiats de **lignes**. Les **colonnes** de la grille sans largeur spécifiée seront **automatiquement** mises en page en **colonnes** de largeur égale. Par exemple, quatre instances du composant **Col** auront automatiquement 25% de largeur. Les classes de **colonnes** indiquent le nombre de **colonnes** que vous souhaitez utiliser sur un **maximum de 12** par **ligne**. Donc, si vous voulez trois **colonnes** de largeur égale, vous pouvez définir la propriété **xs**, **sm**, **md**, **lg** ou **xl** à 4. Les largeurs de **colonne** sont définies en **pourcentage**, elles sont donc toujours fluides et dimensionnées par rapport à leur **élément parent**. Les **colonnes** ont un remplissage horizontal pour créer les gouttières entre les **colonnes** individuelles. Toutefois, vous pouvez supprimer la marge des **lignes** et le remplissage des **colonnes** avec la propriété **noGutters** du composant **Row**.

Pour que la grille soit responsive, il existe cinq types de grille, un pour chaque type d'écran: très petits (**xs**), petits (**sm**), moyens (**md**), grands (**lg**) et très grands (**xl**). Les types de grille sont basés sur la largeur minimale de l'écran, c'est-à-dire qu'elles s'appliquent à tous ceux qui le précèdent (par exemple, **sm** s'applique aux petits, moyens, grands et très grands périphériques, mais pas aux très petits (**xs**)).

Exemple:

```
import {Container, Row, Col} from 'reactstrap';
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        <Container>
          <Row>
            <Col>.col</Col>
            <Col>.col</Col>
            <Col>.col</Col>
            <Col>.col</Col>
          </Row>
          <Row>
            <Col xs="1">.col-1</Col>
            <Col xs="2">.col-2</Col>
            <Col xs="3">.col-3</Col>
            <Col xs="6">.col-6</Col>
          </Row>
        </Container>
      </div>
    );
  }
}
export default App;
```

Documentation: https://getbootstrap.com/docs/4.1/layout/grid/

2.2 Les composants

Les composants Bootstrap correspondent à des briques élémentaires permettant d'afficher votre contenu en fonction de votre besoin. Lorsque vous fabriquerez votre page, votre objectif sera d'utiliser ces composants pour agencer le contenu que vous souhaitez afficher. Pour connaître les composants à votre disposition, ils sont listés ici: http://reactstrap.github.io/components/

3. Afficher des graphes

Pour afficher simplement et rapidement un graphe avec React, un outil très pratique basé sur la librairie JS **Chart.js** est à votre disposition: **react-chartjs**. Il vous permet de créer plusieurs type de graphique en fonction de données stockées dans un objet. Pour l'installation:

```
npm install chart.js@^1.1.1 react-chartjs
```

Et un exemple d'utilisation:

```
import React, { Component } from 'react';
import { Line, Bar, Pie, Doughnut } from "react-chartjs";

class App extends Component {
  render() {
    let lineOrBarData = {
      labels: ["January", "February", "March", "April", "May", "June"],
      datasets: [
        {
          data: [0, 10, 23, 45, 9, 18]
        }
      ]
    };

    let pieOrDoughnutData = [
      { label: "Value 1", value: 75 },
      { label: "Value 2", value: 25 },
      { label: "Value 3", value: 35 },
      { label: "Value 4", value: 55 }
    ];

    return (
      <div className="App">
        <Line data={lineOrBarData} width="600" height="300"/>
        <Bar data={lineOrBarData} width="600" height="300"/>
        <Pie data={pieOrDoughnutData} width="600" height="300"/>
        <Doughnut data={pieOrDoughnutData} width="600" height="300"/>
      </div>
    );
  }
}
export default App;
```

Il est important de respecter la structure des données pour la création de graphes. Pour en savoir plus, vous pouvez regarder le lien redirigeant vers des exemples ou la documentation de la librairie.

Exemples: http://reactcommunity.org/react-chartjs/index.html

Documentation: http://www.chartjs.org/docs/latest/

4. Envoyer des requêtes API

Afin de contacter une API Web durant l'exécution de l'application, il faut avoir la possibilité d'envoyer des requêtes HTTP. Pour cela, il existe la fonction Javascript **fetch**. Son utilisation est très simple:

```
fetch("https://mon-api.com/api/v1/users", { method: "GET" })
    .then((response) => {
        return response.json();
    })
    .then((json) => {
        console.log(JSON.stringify(json));
    });
```

La méthode **then** correspond à l'utilisation d'une promesse (plus d'informations ici) et permet donc d'exécuter un callback une fois que la requête s'est correctement déroulée.

Pour envoyer des données, il suffit simplement d'ajouter la propriété **body** dans les options:

```
fetch("https://mon-api.com/api/v1/users", {
        method: "POST",
        body: { data: "test" }
    })
    .then((response) => {
        return response.json();
    })
    .then((json) => {
        console.log(JSON.stringify(json));
    });
```

Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch