

Cours Inf3522 - Développement d'Applications N tiers

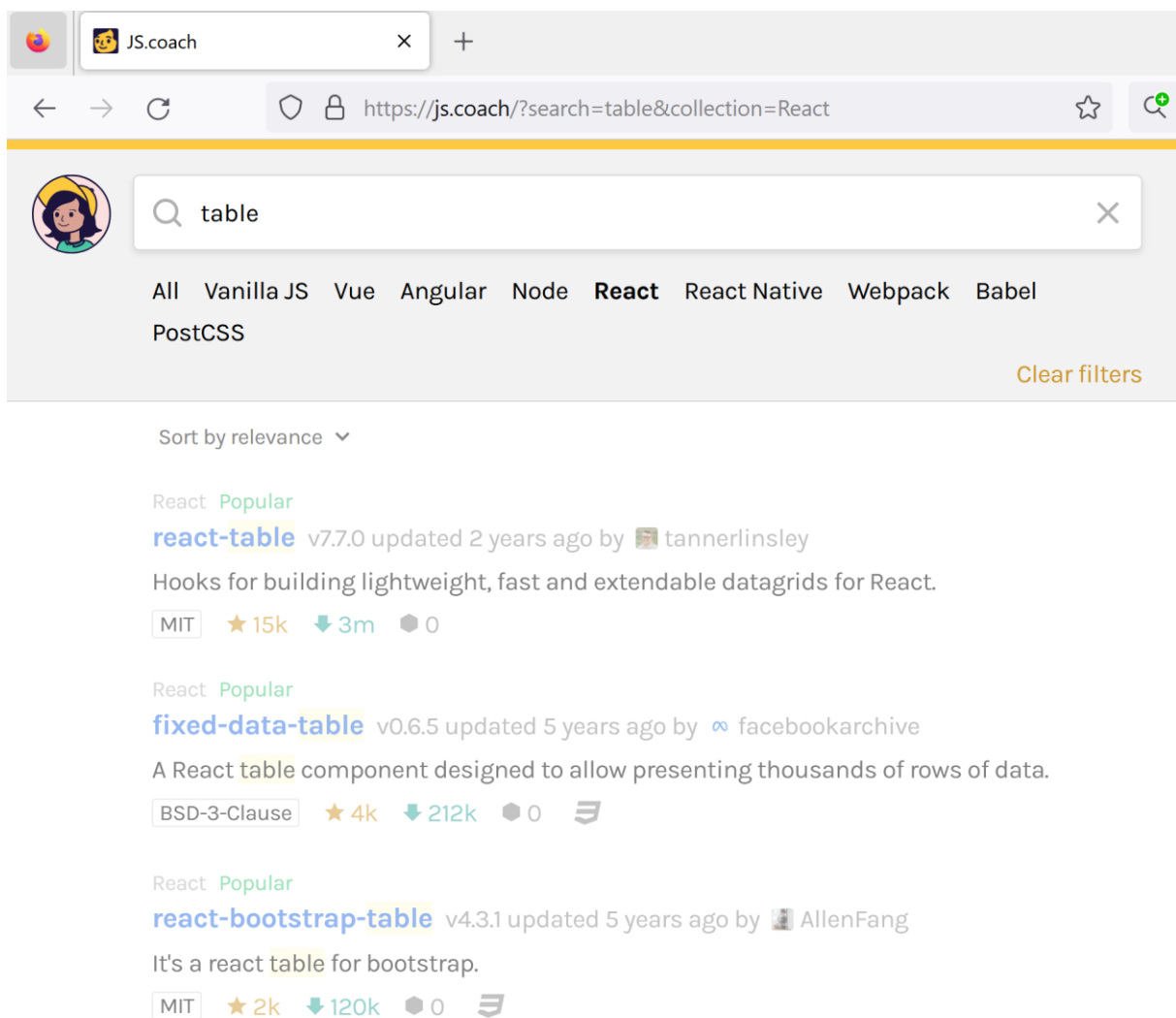
Lab 11 : Composants utiles pour le développement avec React

React est basé sur les composants, et nous pouvons trouver beaucoup de composants tiers utiles que nous pouvons utiliser dans nos applications. Dans ce lab, nous allons examiner plusieurs composants que nous allons utiliser dans notre tiers présentation. Nous examinerons comment trouver des composants appropriés et comment les utiliser dans vos propres applications.

Utilisation de composants React tiers

Il existe de nombreux composants React intéressants disponibles pour différentes utilisations. Notre première tâche est de trouver un composant adapté à vos besoins. Un bon site pour rechercher des composants est JS.coach (<https://js.coach/>). Il vous suffit de saisir un mot-clé, de lancer la recherche, puis de sélectionner React dans la liste des bibliothèques.

Dans la capture d'écran suivante, vous pouvez voir une recherche de composants de table pour React



Une autre bonne source de composants React est "awesome-react-components" (<https://github.com/brillout/awesome-react-components>). Les composants ont souvent une bonne documentation qui vous aide à les utiliser dans votre propre application React. Voyons comment installer un composant tiers dans notre application et commencer à l'utiliser, comme suit :

1. Accédez au site JS.coach, saisissez "date" dans le champ de recherche, et filtrez par React. Parmi les résultats de la recherche, vous trouverez un composant de liste appelé "react-date-picker"
2. Cliquez sur le lien du composant pour voir des informations plus détaillées à son sujet. Vous devriez y trouver les instructions d'installation ainsi que des exemples simples sur la façon d'utiliser le composant.
3. Comme vous pouvez le voir sur la page d'informations du composant, les composants sont installés à l'aide du package npm. La syntaxe de la commande pour installer les composants est la suivante :

npm install nom_du_composant

Ou, si vous utilisez yarn, cela ressemble à ceci :

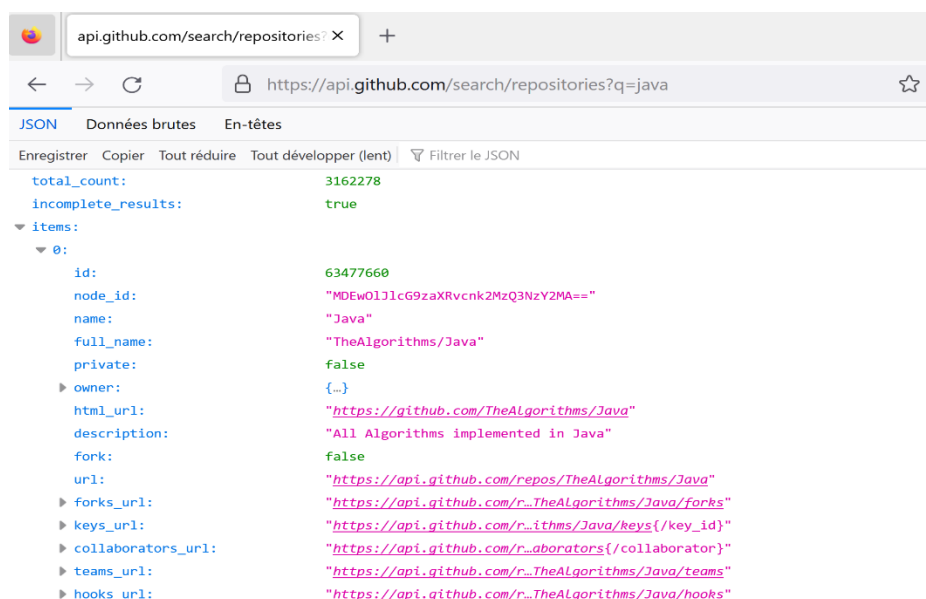
yarn add nom_du_composant

Les commandes npm install et yarn add enregistrent la dépendance du composant dans le fichier **package.json** qui se trouve à la racine de votre application React.

Travailler avec AG Grid

AG Grid (<https://www.ag-grid.com/>) est un composant de grille flexible pour les applications React. Il possède de nombreuses fonctionnalités utiles, telles que le filtrage, le tri et le regroupement. Nous utiliserons la version communautaire, qui est gratuite (licence Massachusetts Institute of Technology (MIT)).

Pour cet exemple, créer une application GitHub REST API, nommé "restgithub". Si vous essayez d'accéder au lien de l'API vous aurez :



Le code de l'application permettant de consommer l'API est donné ci-dessous.

```
import React, { useState } from "react";
import "./App.css";

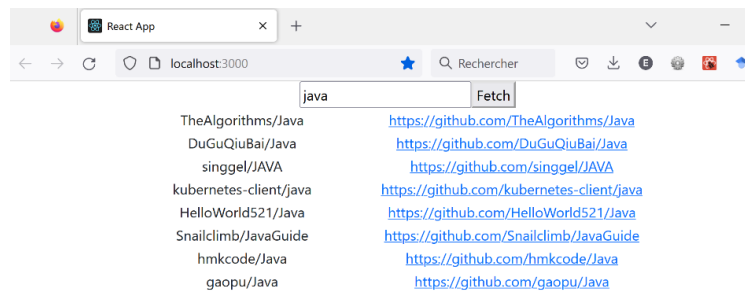
function App() {
  const [keyword, setKeyword] = useState("");
  const [data, setData] = useState([]);
```

```

const fetchData = () => {
  fetch(`https://api.github.com/search/repositories?q=${keyword}`)
    .then(response => response.json())
    .then(data => setData(data.items))
    .catch(err => console.error(err));
};
return (
  <div className="App">
    <input value={keyword} onChange={e => setKeyword(e.target.value)} />
    <button onClick={fetchData}>Fetch</button>
    <table style={{ margin: "auto" }}>
      <tbody>
        {data.map(repo => (
          <tr key={repo.id}>
            <td>{repo.full_name}</td>
            <td>
              <a href={repo.html_url}>{repo.html_url}</a>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
);
}
export default App;

```

Si vous lancez l'application, vous devez obtenir l'interface ci-dessous :



1. Pour installer le composant AG Grid communautaire, ouvrez PowerShell et rendez-vous dans le dossier "restgithub", qui est le dossier racine de l'application. Installez le composant en saisissant la commande suivante :

npm install --save ag-grid-community

npm install --save ag-grid-react

2. Ouvrez le fichier App.js avec Visual Studio Code (VS Code) et supprimez l'élément table à l'intérieur de l'instruction return. Le fichier App.js devrait maintenant ressembler à ceci :

```

1 import React, { useState } from "react";
2 import "../App.css";
3
4 function App() {
5   const [keyword, setKeyword] = useState("");
6   const [data, setData] = useState([]);
7
8   const fetchData = () => {
9     fetch("https://api.github.com/search/repositories?q=${keyword}")
10      .then(response => response.json())
11      .then(data => setData(data.items))
12      .catch(err => console.error(err));
13   };
14
15   return (
16     <div className="App">
17       <input value={keyword} onChange={e => setKeyword(e.target.value)} />
18       <button onClick={fetchData}>Fetch</button>
19     </div>
20   );
21 }
22
23 export default App;

```

3. Importer le composant ag-grid ainsi que les feuilles de style en ajoutant les lignes de code suivantes au début du fichier App.js. ag-grid propose différents styles prédéfinis, et nous utilisons un design de type "material design".

```

import { AgGridReact } from "ag-grid-react";
import "ag-grid-community/styles/ag-grid.css";
import "ag-grid-community/styles/ag-theme-material.css";

```

4. Ensuite, nous ajoutons le composant AgGridReact importé à l'instruction de retour (return). Pour remplir le composant ag-grid avec des données, vous devez passer la prop "rowData" au composant. Les données peuvent être un tableau d'objets, donc nous pouvons utiliser notre état (state) appelé "data". Le composant ag-grid devrait être enveloppé à l'intérieur de l'élément div qui définit le style. Le code est illustré dans l'extrait suivant :

```

<div className="ag-theme-material" style={{ height: 500, width: "90%" }}>
  <AgGridReact rowData={data} columnDefs={columns} />
</div>

```

5. Ensuite, nous allons définir les colonnes pour ag-grid. Nous allons définir une constante appelée "columns" qui est un tableau d'objets de colonne. Dans un objet de colonne, vous devez au moins définir l'accessoire de données en utilisant la propriété "field". Les valeurs de "field" proviennent des données de réponse de notre API REST. Vous pouvez voir dans l'extrait de code suivant que nos données de réponse contiennent un objet appelé "owner", et nous pouvons afficher ces valeurs en utilisant la syntaxe "owner.field_name" :

```

const columns = [
  { field: 'full_name' },
  { field: 'html_url' },
  { field: 'owner.login' }
];

```

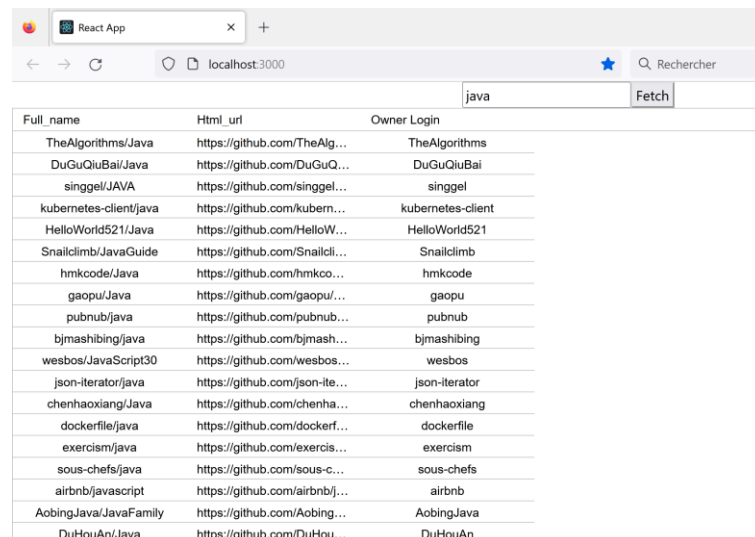
6. Enfin, nous utiliserons la prop "columnDefs" d'ag-grid pour définir ces colonnes, comme suit

```

<AgGridReact
  rowData={data}
  columnDefs={columns}
/>

```

7. Exécutez l'application et accédez à <http://localhost:3000>. Le tableau a un aspect assez agréable par défaut, comme indiqué dans la capture d'écran suivante :



Full_name	Html_url	Owner Login
TheAlgorithms/Java	https://github.com/TheAlg...	TheAlgorithms
DuGuQiuBai/Java	https://github.com/DuGuQ...	DuGuQiuBai
singgel/JAVA	https://github.com/singgel...	singgel
kubernetes-client/java	https://github.com/kubern...	kubernetes-client
HelloWorld521/Java	https://github.com/HelloW...	HelloWorld521
Snailclimb/JavaGuide	https://github.com/Snailcli...	Snailclimb
hmkcode/Java	https://github.com/hmkco...	hmkcode
gaopu/Java	https://github.com/gaopu/...	gaopu
pubnub/java	https://github.com/pubnub...	pubnub
bjmashibing/java	https://github.com/bjmash...	bjmashibing
wesbos/JavaScript30	https://github.com/wesbos...	wesbos
json-iterator/java	https://github.com/json-ite...	json-iterator
chenhaoxiang/Java	https://github.com/chenha...	chenhaoxiang
dockterfile/java	https://github.com/dockterf...	dockterfile
exercism/java	https://github.com/exercis...	exercism
sous-chefs/java	https://github.com/sous-c...	sous-chefs
airbnb/javascript	https://github.com/airbnb/j...	airbnb
AobingJava/JavaFamily	https://github.com/Aobing...	AobingJava
DuHouAn/Java	https://github.com/DuHou...	DuHouAn

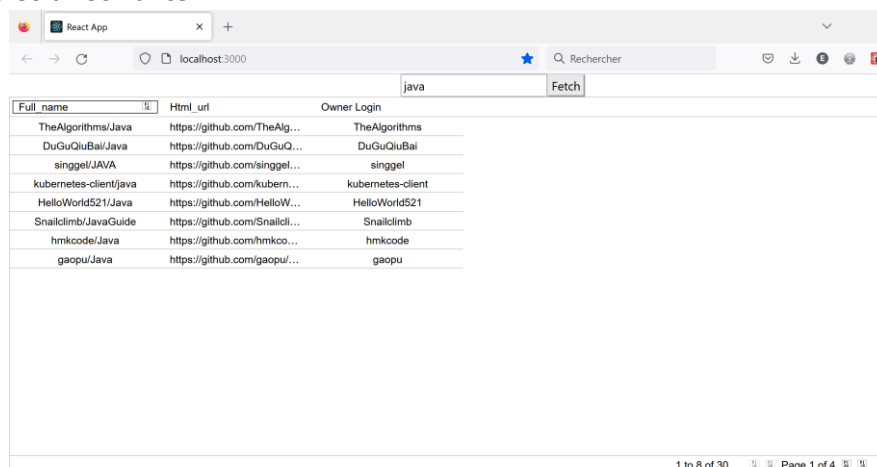
8. Le tri et le filtrage sont désactivés par défaut, mais vous pouvez les activer en utilisant les propriétés "sortable" (tri) et "filter" (filtre) dans les colonnes ag-grid, comme suit :

```
const columns = [
  { field: 'full_name', sortable: true, filter: true },
  { field: 'html_url', sortable: true, filter: true },
  { field: 'owner.login', sortable: true, filter: true }
];
```

Désormais, vous pouvez trier et filtrer les colonnes du tableau en cliquant sur l'en-tête de la colonne. Vous pouvez également activer la pagination et définir la taille de la page dans ag-grid en utilisant les propriétés "pagination" et "paginationPageSize", comme suit :

```
<AgGridReact
  rowData={data}
  columnDefs={columns}
  pagination={true}
  paginationPageSize={8}
/>
```

Maintenant, vous devriez voir la pagination dans votre tableau, comme illustré dans la capture d'écran suivante :



Full_name	Html_url	Owner Login
TheAlgorithms/Java	https://github.com/TheAlg...	TheAlgorithms
DuGuQiuBai/Java	https://github.com/DuGuQ...	DuGuQiuBai
singgel/JAVA	https://github.com/singgel...	singgel
kubernetes-client/java	https://github.com/kubern...	kubernetes-client
HelloWorld521/Java	https://github.com/HelloW...	HelloWorld521
Snailclimb/JavaGuide	https://github.com/Snailcli...	Snailclimb
hmkcode/Java	https://github.com/hmkco...	hmkcode
gaopu/Java	https://github.com/gaopu/...	gaopu

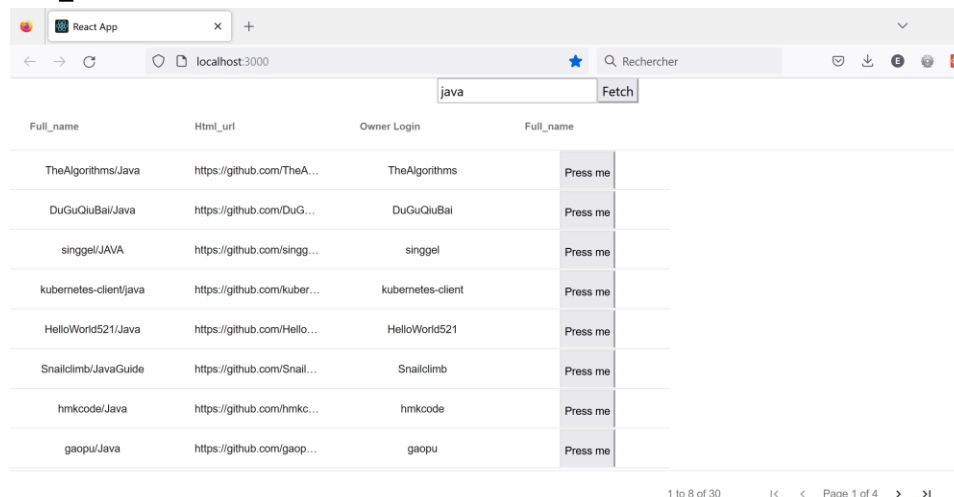
1 to 8 of 30 Page 1 of 4

Vous pouvez trouver la documentation concernant les différentes propriétés de grille (grid) et de colonne (column) sur le site AG Grid.

9. La prop "cellRendererFramework" peut être utilisée pour personnaliser le contenu d'une cellule du tableau. L'exemple suivant montre comment vous pouvez rendre un bouton dans une cellule du tableau. La fonction dans le cellRenderer reçoit "params" en argument. "params.value" sera la valeur de la cellule "full_name", qui est définie dans l'accessoire (accessor) de la colonne. Vous pouvez également utiliser "params.row", qui est l'objet de la ligne entière. Lorsque le bouton est pressé, une alerte s'ouvrira affichant la valeur de la cellule "full_name" :

```
const columns = [
  { field: 'full_name', sortable: true, filter: true },
  { field: 'html_url', sortable: true, filter: true },
  { field: 'owner.login', sortable: true, filter: true },
  {
    field: 'full_name',
    cellRenderer: params => (
      <button onClick={() => alert(params.value)}>
        Press me
      </button>
    )
  }
]
```

Voici une capture d'écran du tableau avec des boutons. Vous pouvez essayer de cliquer sur n'importe quel bouton, et vous devriez voir une alerte affichant la valeur de la cellule "full_name" :



Full_name	Html_url	Owner Login	Full_name
TheAlgorithms/Java	https://github.com/TheA...	TheAlgorithms	Press me
DuGuQiuBai/Java	https://github.com/DuG...	DuGuQiuBai	Press me
singgel/JAVA	https://github.com/singg...	singgel	Press me
kubernetes-client/java	https://github.com/kuber...	kubernetes-client	Press me
HelloWorld521/Java	https://github.com/Hello...	HelloWorld521	Press me
Snailclimb/JavaGuide	https://github.com/Snail...	Snailclimb	Press me
hmkcode/Java	https://github.com/hmkc...	hmkcode	Press me
gaopu/Java	https://github.com/gaop...	gaopu	Press me

Maintenant, la colonne des boutons a un en-tête "Full_name" car, par défaut, le nom du champ sera utilisé comme nom d'en-tête. Si vous souhaitez utiliser autre chose, vous pouvez utiliser la propriété "headerName" dans la définition de la colonne.

Utilisation de la bibliothèque de composants MUI

MUI (<https://mui.com/>) est une bibliothèque de composants React qui met en œuvre le langage de conception Material Design de Google. MUI contient de nombreux composants différents, tels que des boutons, des listes, des tableaux et des cartes, que vous pouvez utiliser pour obtenir une interface utilisateur (UI) agréable et uniforme.

Nous allons créer une petite application de liste de courses et styliser l'interface utilisateur en utilisant les composants MUI, comme suit :

1. Créez une nouvelle application React appelée **"shoppinglist"** en exécutant la commande suivante :

npx create-react-app shoppinglist

2. Ouvrez l'application de liste de courses avec VS Code. Installez MUI en tapant la commande suivante dans le dossier racine du projet dans PowerShell ou tout autre terminal adapté que vous utilisez :

npm install @mui/material @emotion/react @emotion/styled

3. Ouvrez le fichier "App.js" et supprimez tout le code à l'intérieur de la balise div "App". Maintenant, votre fichier "App.js" devrait ressembler à ceci, et vous devriez voir une page vide dans le navigateur :

```
import './App.css';
function App() {
  return (
    <div className="App">
      </div>
    );
  }
  export default App;
```

4. MUI propose différents composants de disposition, et le composant de disposition de base est "Container". Celui-ci est utilisé pour centrer votre contenu horizontalement, et vous pouvez spécifier la largeur maximale d'un conteneur en utilisant la prop "maxWidth" ; la valeur par défaut est "lg", ce qui nous convient. Utilisons le composant **"Container"** dans notre fichier **"App.js"**, comme suit :

```
import Container from '@mui/material/Container';
function App() {
  return (
    <Container>
      </Container>
    );
  }
  export default App;
```

5. Nous utiliserons le composant "AppBar" de MUI pour afficher la barre d'outils dans notre application. Importez les composants "AppBar", "ToolBar" et "Typography" dans votre fichier "App.js". Le code est illustré dans l'extrait suivant :

```
import React, { useState } from 'react';
import Container from '@mui/material/Container';
import AppBar from '@mui/material/AppBar';
import Toolbar from '@mui/material/Toolbar';
import Typography from '@mui/material/Typography';
```

6. Ajoutez le code suivant à l'instruction de retour de votre composant "App". Le composant "Typography" fournit des tailles de texte prédéfinies, et nous l'utiliserons dans notre texte de la barre d'outils. La prop "variant" peut être utilisée pour définir la taille du texte :

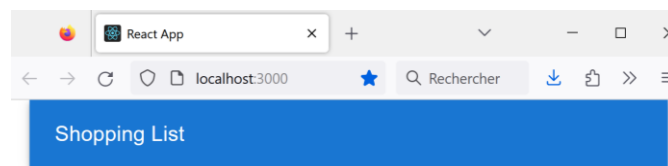
```
function App() {
  return (
    <Container>
```

```

<AppBar position="static">
  <Toolbar>
    <Typography variant="h6">
      Liste de Courses
    </Typography>
  </Toolbar>
</AppBar>
</Container>
);
}

```

Votre application devrait maintenant ressembler à ceci :



7. Dans le composant App, nous avons besoin d'un seul état pour conserver les articles de la liste de courses. Un élément de la liste de courses contient deux champs : "produit" et "quantité". Nous avons également besoin d'une méthode pour ajouter un nouvel élément à l'état "items". Voici le code source pour définir l'état et une fonction pour ajouter un nouvel élément à l'état. Dans la fonction "addItem", nous utilisons la notation « ... », pour ajouter un nouvel élément au début d'un tableau existant :

```

const [items, setItems] = useState([]);
const addItem = (item) => {
  setItems([item, ...items]);
};

```

8. Ajoutez un nouveau composant pour ajouter des articles de courses. Créez un nouveau fichier appelé "AddItem.js" dans le dossier racine de l'application et ajoutez le code suivant à votre fichier "AddItem.js". La fonction du composant AddItem reçoit également des "props" de son composant parent. Le code est illustré dans l'extrait suivant :

```

import React from 'react';
function AddItem(props) {
  return (
    <div></div>
  );
}
export default AddItem;

```

Le composant AddItem utilisera la boîte de dialogue modale de MUI pour collecter les données. Dans le formulaire, nous ajouterons deux champs d'entrée (produit et quantité) et un bouton qui appelle la fonction "addItem" du composant App. Pour pouvoir appeler la fonction "addItem", qui se trouve dans le composant App, nous devons la passer dans les "props" lors du rendu du composant AddItem. En dehors du composant modal Dialog, nous ajouterons un bouton qui ouvre le formulaire modal lorsqu'il est pressé. Ce bouton est le seul élément visible lors du rendu initial du composant.

Le composant Dialog a une propriété appelée "open", et si la valeur est "true", la boîte de dialogue est visible. La valeur par défaut de la propriété "open" est "false", et la boîte de dialogue est masquée. Le

bouton qui ouvre la boîte de dialogue modale définit la valeur de l'état "open" à "true", et la boîte de dialogue s'ouvre.

Nous devons également gérer l'événement de changement des éléments d'entrée (inputs) afin de pouvoir accéder aux valeurs qui ont été saisies. Lorsque le bouton à l'intérieur du formulaire modal est cliqué, la fonction "addItem" est appelée, et le formulaire modal est fermé en définissant la valeur de l'état "open" à "false". La fonction crée un objet à partir des valeurs des champs d'entrée et appelle la fonction "addItem" du composant App, qui ajoute enfin un nouvel élément au tableau d'état et réaffiche l'interface utilisateur.

Les étapes suivantes décrivent la mise en œuvre du formulaire modal :

1. Nous devons importer les composants MUI suivants pour le formulaire modal : Dialog, DialogActions, DialogContent et DialogTitle. En ce qui concerne l'interface utilisateur du formulaire modal, nous avons besoin des composants suivants : Button et TextField. Ajoutez les importations suivantes à votre fichier "AddItem.js" :

```
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Dialog from '@mui/material/Dialog';
import DialogActions from '@mui/material/DialogActions';
import DialogContent from '@mui/material/DialogContent';
import DialogTitle from '@mui/material/DialogTitle';
```

2. Ensuite, nous déclarerons un état appelé "open" et deux fonctions pour ouvrir et fermer la boîte de dialogue modale. La valeur par défaut de l'état "open" est "false". La fonction "handleOpen" définit l'état "open" à "true", et la fonction "handleClose" le définit à "false". Le code est illustré dans l'extrait suivant :

```
// AddItem.js
const [open, setOpen] = React.useState(false);
const handleOpen = () => {
  setOpen(true);
}
const handleClose = () => {
  setOpen(false);
}
```

Votre composant "AddItem.js" devrait maintenant avoir importé les composants nécessaires et défini l'état et les fonctions pour gérer l'ouverture et la fermeture de la boîte de dialogue modale.

3. Nous allons ajouter les composants Dialog et Button à l'intérieur de l'instruction de retour (return). Nous avons un bouton en dehors de la boîte de dialogue qui sera visible lorsque le composant est rendu pour la première fois. Lorsque le bouton est pressé, il appelle la fonction "handleOpen", qui ouvre la boîte de dialogue. À l'intérieur de la boîte de dialogue, nous avons deux boutons : un pour annuler et un pour ajouter un nouvel élément. Le bouton "Add" appelle la fonction "addItem", que nous mettrons en œuvre plus tard. Le code est illustré dans l'extrait suivant :

```
return (
  <div>
    <Button onClick={handleOpen}>
      Ajouter un élément
    </Button>
```

```

<Dialog open={open} onClose={handleClose}>
  <DialogTitle>Nouvel élément</DialogTitle>
  <DialogContent>
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose}>
      Annuler
    </Button>
    <Button onClick={addItem}>
      Ajouter
    </Button>
  </DialogActions>
</Dialog>
</div>
);

```

4. Pour collecter les données d'un utilisateur, nous devons déclarer un nouvel état. L'état est un objet avec deux attributs : "product" (produit) et "amount" (quantité). Ajoutez la ligne de code suivante après la ligne où vous avez déclaré l'état "open" :

```

const [item, setItem] = React.useState({
  product: "",
  amount: ""
});

```

5. À l'intérieur du composant **DialogContent**, nous allons ajouter deux champs d'entrée pour collecter les données d'un utilisateur. Nous utilisons le composant **TextField** de MUI que nous avons déjà importé. La prop "**margin**" est utilisée pour définir l'espacement vertical des champs de texte, et la prop "**fullwidth**" est utilisée pour que le champ d'entrée prenne toute la largeur de son conteneur. Vous pouvez trouver toutes les props dans la documentation Material Design. Les props "value" des champs de texte doivent être les mêmes que l'état où nous voulons enregistrer la valeur saisie. Dans le champ "product", c'est "item.product", et dans le champ "amount", c'est "item.amount". Nous utiliserons également des props "name", comme nous l'avons fait précédemment avec les formulaires. Le code est illustré dans l'extrait suivant :

```

<DialogContent>
  <TextField value={item.product} margin="dense" onChange={handleChange} name="product" label="Produit" fullWidth />
  <TextField value={item.amount} margin="dense" onChange={handleChange} name="amount" label="Quantité" fullWidth />
</DialogContent>

```

6. Ensuite, nous devons mettre en œuvre la fonction "**handleChange**", qui est invoquée lorsque nous tapons quelque chose dans les champs d'entrée. Comme nous l'avons déjà vu aux labs précédents, la fonction suivante enregistre les valeurs du champ d'entrée dans l'état "**item**" :

```

const handleChange = (e) => {
  setItem({ ...item, [e.target.name]: e.target.value });
}

```

7. Enfin, nous devons ajouter une fonction qui appelle la fonction "**addItem**" que nous obtenons dans la prop "**props**" et passe un nouvel élément dans cette fonction. Le nouvel élément est maintenant l'état "**item**" qui contient l'élément de la liste de courses que l'utilisateur a saisi. Comme nous obtenons la fonction "**addItem**" à partir des props, nous pouvons l'appeler en utilisant le mot-clé "**props**". Nous

appellerons également la fonction **"handleClose"**, qui ferme la boîte de dialogue modale. Le code est illustré dans l'extrait suivant :

// Appelle la fonction addItem (dans props) et passe l'état item à l'intérieur

```
const addItem = () => {  
  props.addItem(item);  
  setItem({ product: "", amount: "" }); // Effacer les champs de texte  
  handleClose();  
}
```

Avec ces étapes, le formulaire modal pour ajouter des éléments à la liste de courses est maintenant fonctionnel avec des champs d'entrée pour le produit et la quantité, ainsi que des boutons pour ajouter et annuler.

8. Notre composant **AddItem** est maintenant prêt, et nous devons l'importer dans notre fichier **App.js** et le rendre là-bas. Ajoutez la déclaration d'importation suivante dans votre fichier **App.js** :

```
import AddItem from './AddItem';
```

9. Ajoutez le composant **AddItem** à l'instruction return dans le fichier **App.js**. Passez la fonction **addItem** dans une prop au composant **AddItem**, comme suit :

```
return (  
  <Container>  
    <AppBar position="static">  
      <Toolbar>  
        <Typography variant="h6">  
          Shopping List  
        </Typography>  
      </Toolbar>  
    </AppBar>  
    <AddItem addItem={addItem} />  
  </Container>  
);
```

10. Si nous voulons centrer le bouton « ADD ITEM », nous pouvons utiliser le composant MUI Stack. Il peut être utilisé pour définir la disposition de ses composants enfants. La direction par défaut est la colonne, et vous pouvez la modifier en utilisant la prop **direction**. Vous pouvez utiliser la prop **alignItems** pour définir l'alignement horizontal. Le code est illustré dans l'extrait suivant :

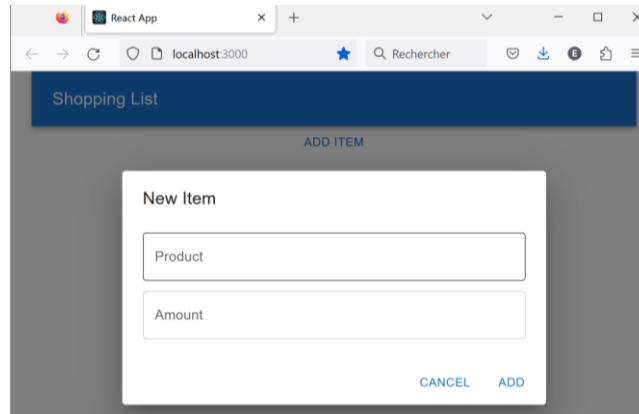
```
import Stack from '@mui/material/Stack';
```

```
return (  
  <Container>  
    <AppBar position="static">  
      <Toolbar>  
        <Typography variant="h6">  
          Shopping List  
        </Typography>  
      </Toolbar>  
    </AppBar>  
    <Stack alignItems="center">  
      <AddItem addItem={addItem} />  
    </Stack>  
  </Container>  
);
```

`</Container>`

`);`

11. Maintenant, si vous ouvrez votre application dans le navigateur et que vous appuyez sur le bouton « ADD ITEM », vous verrez le formulaire modal s'ouvrir et vous pourrez saisir un nouvel élément, comme illustré dans la capture d'écran suivante. Le formulaire modal se ferme lorsque vous appuyez sur le bouton « ADD » :



12. Ensuite, nous ajouterons une liste au composant App qui affiche nos articles de shopping. Pour cela, nous utiliserons les composants MUI **List**, **ListItem** et **ListItemText**.

Importez ces composants dans le fichier App.js. Voici le code dont vous aurez besoin :

```
import List from '@mui/material/List';
import ListItem from '@mui/material/ListItem';
import ListItemText from '@mui/material/ListItemText';
```

13. Ensuite, nous rendrons le composant List, et à l'intérieur, nous utiliserons la fonction map pour générer des composants ListItem. Chaque composant ListItem doit avoir une prop key unique, et nous utiliserons la prop divider pour obtenir une séparation à la fin de chaque élément de la liste. Nous afficherons le produit dans le texte principal (primary text) et la quantité dans le texte secondaire (secondary text) du composant ListItemText. Le code est illustré dans l'extrait suivant :

```
return (
  <Container>
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6">
          Shopping List
        </Typography>
      </Toolbar>
    </AppBar>
    <Stack alignItems="center">
      <AddItem addItem={addItem} />
      <List>
        {
          items.map((item, index) =>
            <ListItem key={index} divider>
              <ListItemText
                primary={item.product}

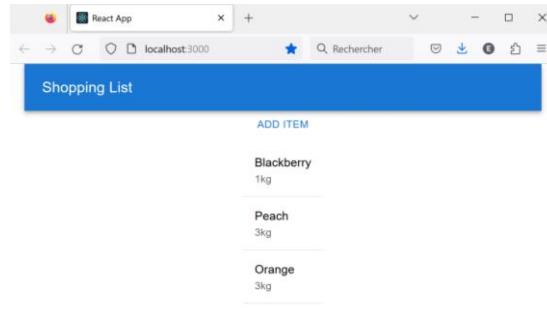
```

```

        secondary={item.amount}
      />
    </ListItem>
  )
}
</List>
</Stack>
</Container>
);

```

Maintenant, l'interface ressemble à ceci :



Le composant MUI Button a trois variantes : text, contained et outlined. La variante text est celle par défaut, et vous pouvez la changer en utilisant la prop variant. Par exemple, si vous voulez avoir un bouton ADD ITEM avec une bordure (outlined), modifiez la prop variant dans le fichier AddItem.js, comme ceci :

```

<Button variant="outlined" onClick={handleOpen}>
  Add Item
</Button>

```

Ensuite, nous apprendrons comment utiliser React Router, une bibliothèque de routage populaire.

Gestion du routage dans React

Il existe plusieurs solutions disponibles pour le routage dans React. La plus populaire, que nous utilisons, est React Router (<https://github.com/ReactTraining/react-router>). Pour les applications web, React Router fournit un package appelé react-router-dom.

Pour commencer à utiliser React Router, nous devons installer les dépendances à l'aide de la commande suivante.

npm install react-router-dom history

Quatre composants différents du package react-router-dom sont nécessaires pour implémenter le routage. BrowserRouter est le routeur pour les applications basées sur le web. Le composant Route rend le composant défini si les emplacements donnés correspondent. L'exemple suivant montre l'utilisation du composant Route. La prop element définit un composant rendu lorsque l'utilisateur accède à l'endpoint contact, défini dans la prop path. Le chemin est relatif au routeur parent qui les rend :

```

<Route path="contact" element={<Contact />} />

```

Vous pouvez utiliser un astérisque * à la fin de la prop path, comme ceci :

```

<Route path="/contact/*" element={<Contact />} />

```

Maintenant, cela correspondra à tous les endpoints sous contact, par exemple contact/daffy, contact/bugs, etc.

Le composant Routes enveloppe plusieurs composants Route. Le composant Link permet la navigation dans votre application. L'exemple suivant montre le lien Contact qui navigue vers l'endpoint /contact lorsque le lien est cliqué :

```
<Link to="/contact">Contact</Link>
```

L'exemple suivant montre comment utiliser ces composants en pratique. Créez une nouvelle application React appelée **routerapp** en utilisant **create-react-app**, comme suit :

1. Ouvrez le dossier src avec VS Code et ouvrez le fichier **App.js** dans la vue de l'éditeur. Importez les composants du package react-router-dom et supprimez le code superflu de l'instruction return. Après ces modifications, votre code source App.js devrait ressembler à ceci :

```
import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import './App.css';
function App() {
  return (
    <div className="App">
      </div>
    );
}
export default App;
```

2. Créez d'abord deux composants simples que nous pouvons utiliser dans le routage. Créez deux nouveaux fichiers, Home.js et Contact.js, dans le dossier src de l'application. Ensuite, ajoutez des entêtes aux instructions return pour afficher le nom du composant. Le code des composants ressemble à ceci :

```
// Home.js
function Home() {
  return <h1>Home.js</h1>;
}
export default Home;
```

```
// Contact.js
function Contact() {
  return <h1>Contact.js</h1>;
}
export default Contact;
```

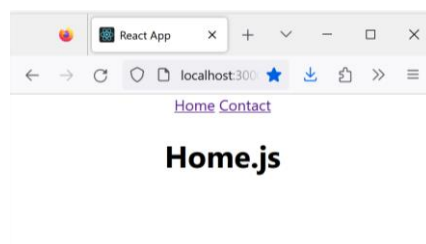
3. Ouvrez le fichier App.js, puis ajoutez un routeur qui nous permet de naviguer entre les composants, comme suit :

```
import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import Contact from './Contact';
import './App.css';
```

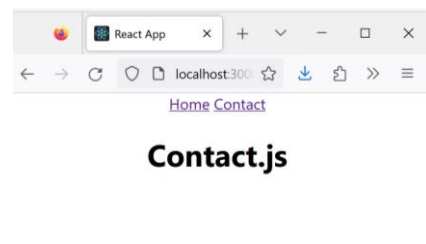
```
function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <nav>
          <Link to="/">Home</Link>{' '}
          <Link to="/contact">Contact</Link>{' '}
        </nav>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="contact" element={<Contact />} />
        </Routes>
      </BrowserRouter>
    </div>
  );
}
```

export default App;

4. Maintenant, lorsque vous démarrez l'application, vous verrez les liens et le composant Home, qui s'affiche à l'endpoint racine (localhost:3000), comme défini dans le premier composant Route. Vous pouvez voir une représentation de ceci dans la capture d'écran suivante :



5. Lorsque vous cliquez sur le lien Contact, le composant Contact est rendu, comme illustré ici :



Vous pouvez avoir une route PageNotFound en utilisant un astérisque * dans la prop path. Dans l'exemple suivant, si aucune autre route ne correspond, la dernière est utilisée :

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="contact" element={<Contact />} />
  <Route path="*" element={<PageNotFound />} />
</Routes>
```

Vous pouvez également avoir des routes imbriquées comme celles montrées dans l'exemple suivant :

```
<Routes>
  <Route path="contact" element={<Contact />}>
    <Route path="london" element={<ContactLondon />} />
    <Route path="paris" element={<ContactParis />} />
  </Route>
</Routes>
```

Vous pouvez également utiliser le hook **useRoutes()** pour déclarer des routes en utilisant des objets JavaScript au lieu d'éléments React, mais nous n'abordons pas ici.

À ce stade, nous avons appris comment installer et utiliser des composants tiers avec React. Ces compétences seront nécessaires dans les labs suivants lorsque nous commencerons à construire notre frontend pour notre backend développé avec spring boot.