Etape 2 - Développer une application `react`

Pré-requis

Vous devez maîtriser les étapes 0 et 1 du workshop afin de pouvoir réaliser l'étape 2.

Le code disponible dans cette étape correspond au résultat attendu des étapes 0 et 1. Vous pouvez partir de cette base pour développer l'étape 2.

Pour lancer l'application de l'étape 1, exécutez la commande `npm start` (après avoir fait un `npm install`). Ouvrez ensuite l'URL http://localhost:8080 dans votre navigateur.

Dans cette étape, vous allez avoir besoin de l'API. Pour l'exécuter, lancez la commande `npm start` dans le dossier `api`.

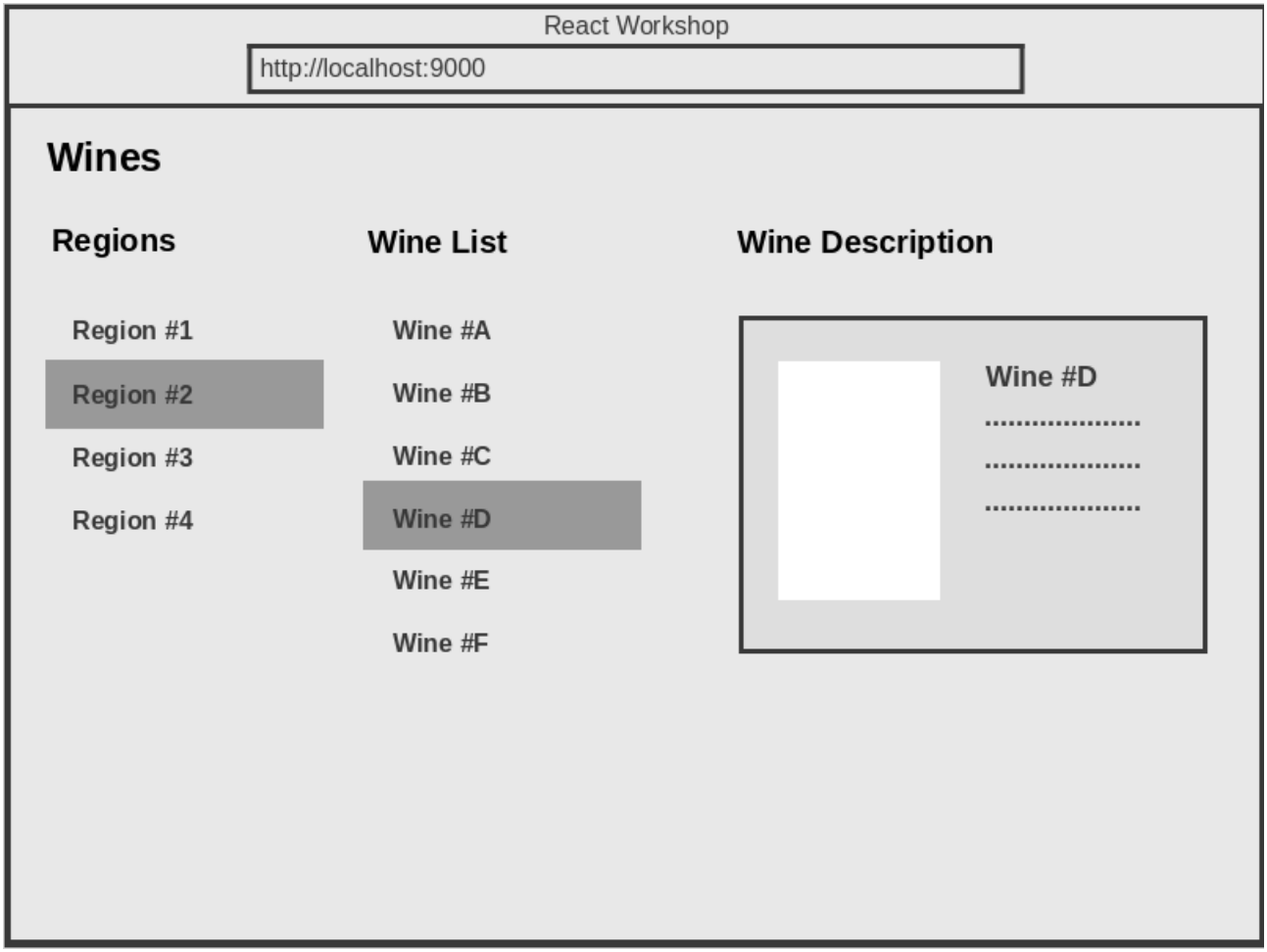
Vous avez également la possibilité de lancer les tests de cette étape (que nous avons rédigé pour vous) en utilisant la commande `npm test` afin de voir quelles parties de l'étape fonctionnent et quelles parties ne fonctionnent pas du tout. N'hésitez pas à lire le code des tests afin d'avoir quelques indications en plus sur la façon d'écrire votre application.

Objectif

L'objectif de cette étape est de développer une application permettant d'afficher la fiche descriptive d'un vin, chaque vin étant catégorisé dans une région viticole.

Voici une maquette de l'application :

- Une première colonne listant les régions viticoles. Chaque région est sélectionnable par un clic.
- Une deuxième colonne listant les vins de la région viticole sélectionnée. Chaque vin est sélectionnable par un clic.
- Une troisième colonne affichant la fiche descriptive du vin sélectionné, contenant :
  - Le nom du vin
  - Le type de vin (rouge, blanc, rosé, etc ...)
  - L'appellation du vin (nom de l'appellation et région)
  - La liste des cépages du vin
  - Une image du vin



Avant de partir la tête baissée dans le développement, il est nécessaire de se poser quelques questions :

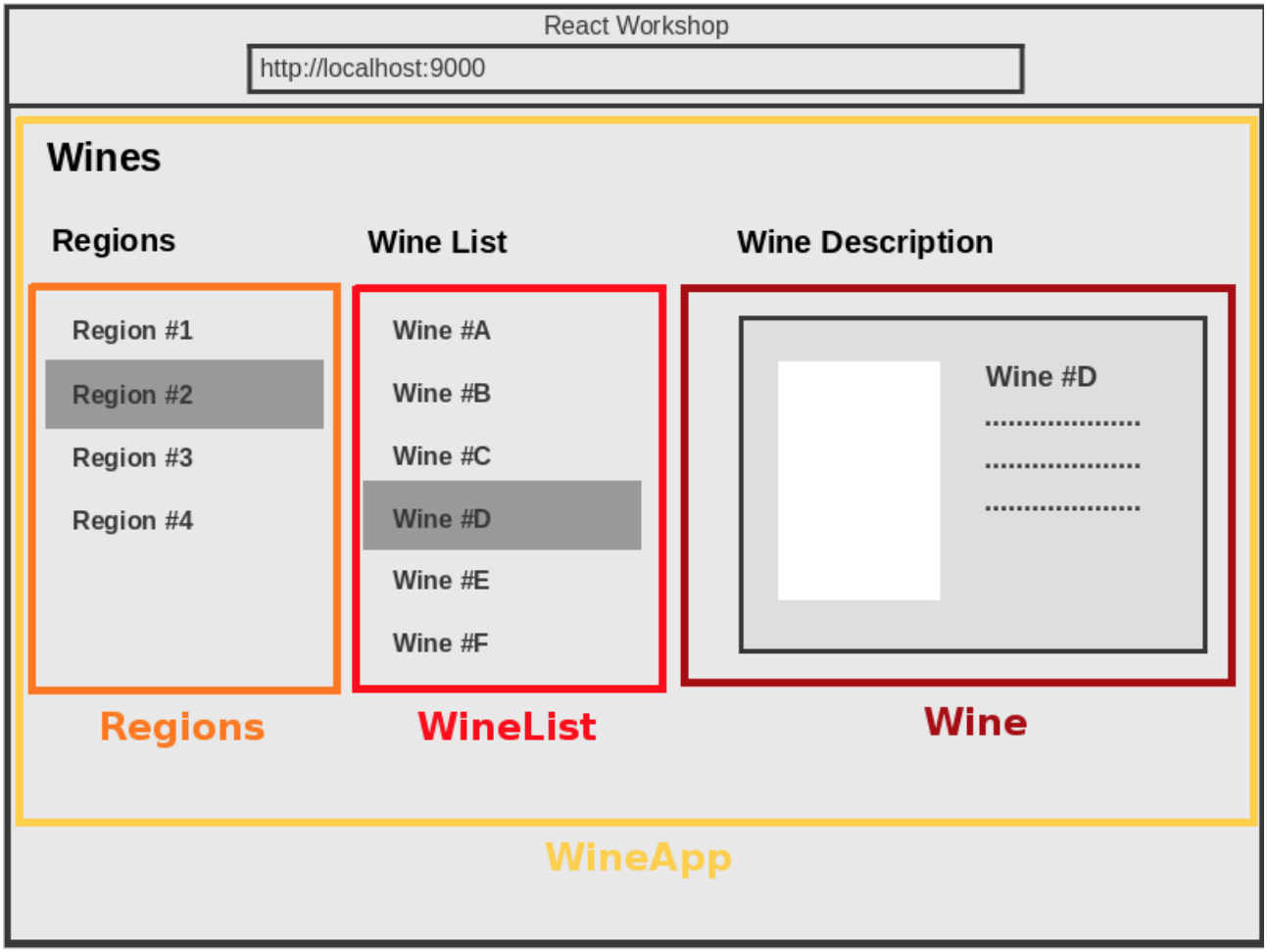
- Quels sont les composants de mon application et comment sont-ils liés ?
- Quelles sont les données gérées par mes composants ? Ces données sont-elles gérées via les `props` ou via le `state` ?
- Quelles sont les interactions entre mes composants ? Quels événements doivent-être gérés ?
- Ok, je suis prêt pour développer, mais par quoi je commence ?

Hiérarchie de composants

La première chose à faire est de mettre son cerveau en **mode React** et de **penser composants** :-)

A partir de la maquette fournie, nous pouvons identifier les principaux composants de notre application :

- `Regions` : le composant gérant la liste des régions,
- `WineList` : le composant gérant la liste des vins,
- `Wine` : le composant gérant la fiche descriptive d'un vin,
- `WineApp` : le composant parent permettant d'assembler les autres composants, qui représente au final notre application.



Props et state

Les données gérées par l'application sont les suivantes :

- la liste des régions viticoles,
- la référence à la région sélectionnée dans la liste des régions,
- la liste des vins de la région viticole sélectionnée,
- la description du vin sélectionné dans la liste des vins.

Il reste à savoir de quelle manière ces données doivent être gérées au niveau de chaque composant : via les `props` ou via le `state` ?

Les `props` d'un composant sont immutables contrairement au `state` qui lui est mutable. Il est donc généralement conseillé de définir deux types de composants dans une application `react` :

- Les composants dédiés purement à la présentation, qui s'appuieront uniquement sur leur `props`. Nous les surnommerons `Dumb components`.
- Les composants plutôt orientés "conteneurs", qui sont responsables du fonctionnement de l'application, qui s'appuient fortement sur leur `state`. Nous les surnommerons `Smart components`.

Vous pouvez lire un article très intéressant sur le sujet : Smart and Dumb Components

Dans notre cas, `Regions`, `WineList` et `Wine` sont des composants orientés présentation et utiliseront uniquement leurs `props`. `WineApp` est le conteneur et utilisera donc son `state`.

Interactions entre les composants

Les interactions entre les composants sont les suivantes :

- La sélection d'une région provoque la mise à jour de la liste des vins (affichage de la liste des vins de la régions sélectionnée) et de la description du vin (affichage de la description du premier vin de la liste)
- La sélection d'un vin provoque la mise à jour de la description du vin.

Par où commencer le développement ?

Par expérience, une bonne façon de démarrer le développement d'une application `react` est la suivante :

- Construire une version statique de l'application en créant l'ensemble des composants définis lors des précédentes étapes de conception.
- Définir l'état initial des composants de type conteneur et utiliser cet état dans le rendu des composants.
- Charger dynamiquement les données depuis l'API (appels Ajax) et mettre à jour l'état des composants.
- Gérer les événements pour rendre l'application interactive.

Version statique de l'application

Dumbs

Créez l'ensemble des composants de type présentation: `Regions` , `WineList` , et `Wine` et implémentez la méthode `render()` de chacun d'eux, en vous appuyant sur les `props` .

Par exemple pour le composant `Regions` :

```
const Regions = React.createClass({
  render () {
    return (
      <div>
        { this.props.regions.map(region => <div key={region}>{region}</div>)
      </div>
    )
  }
})

export default Regions
```

Lors du rendu d'une liste de composants, `react` a besoin d'une propriété unique `key` sur chaque composant de la liste. Pour plus de détails techniques, lisez la documentation

PropTypes

Il est possible de définir plus précisément le format attendu dans les `props` d'un composant, grâce aux `PropTypes` .

Par exemple sur notre composant `Wine` qui gère l'affichage de la description d'un vin :

```
imoort React, { PropTypes } from 'react';

const Wine = React.createClass({
```

```
  propTypes: {
    wine: PropTypes.shape({
      id: PropTypes.string,
      name: PropTypes.string,
      type: PropTypes.oneOf(['Rouge', 'Blanc', 'Rosé', 'Effervescent', 'Moell
      appellation: PropTypes.shape({
        name: PropTypes.string,
        region: PropTypes.string
      }),
      grapes: PropTypes.arrayOf(PropTypes.string)
    })
  },
  // ...
}
```

Définissez les `propTypes` de chaque composant, en vous appuyant sur le format des données remontées par l'API.

Attention : les `propTypes` ne sont vérifiées qu'en mode développement pour aider le développeur à utiliser correctement les composants.

Smarts

Créez le composant conteneur `WineApp` qui permet d'assembler l'ensemble des composants.

C'est ce composant qui sera utilisé pour effectuer le rendu de l'application dans le DOM, dans le fichier `app.js` :

```
import React from 'react';
import ReactDOM from 'react-dom';

import WineApp from './components/wine-app';

ReactDOM.render(
    <WineApp />,
    document.getElementById('main')
);
```

Afin de pouvoir tester rapidement de rendu de l'application, vous pouvez utiliser des valeurs "en dur" pour alimenter les `props` des composants. Par exemple :

```
const WineApp = React.createClass({
  render() {
    return (
      ...
      <Regions regions={["Bordeaux", "Bourgogne"]} />
      ...
    )
  }
})
```

Etat initial des composants

L'étape suivante consiste à définir l'état initial des composants de type conteneur, donc `WineApp` dans notre cas.

Cela se fait via la méthode `getInitialState()` du composant :

```
const WineApp = React.createClass({
  getInitialState() {
    return {
      regions: [],
      selectedRegion: null,
      wines:[],
      selectedWine: null
    };
  },
  // ...
})
```

Utilisez ensuite le `state` dans le rendu. Par exemple pour alimenter la liste des régions :

```
const WineApp = React.createClass({
  // ...
  render() {
    return (
      ...
      <Regions regions={this.state.regions} />
      ...
    }
  }
})
```

Récupération des données via l'API

Les données doivent être récupérées en Ajax au travers de l'API mise à disposition.

Dans un composant `react`, les appels Ajax se font généralement dans la méthode `componentDidMount()` du cycle de vie du composant.

Nous utilisons `fetch` pour effectuer les appels Ajax. Par exemple pour charger la liste des régions :

```
const WineApp = React.createClass({
  // ...
  componentDidMount() {
      fetch('/api/regions')
        .then(r => r.json())
        .then(data => {
          this.setState({
            regions: data,
            selectedRegion: data[0]
          });
          // TODO Charger les vins de la région : this.loadWinesByRegion(data
        })
        .catch(response => {
          console.error(response);
        });
    },
  // ...
})
```

La méthode `setState()` permet de mettre à jour l'état du composant, ce qui provoque son re-rendu.

Complétez le code du composant `WineApp` afin de gérer l'ensemble des appels Ajax permettant d'alimenter correctement le `state`.

Gestion des événements

La dernière étape consiste à gérer les événements afin de rendre l'application interactive.

Ce sont les composants de type présentation qui vont capter les événements. Mais ce ne sont pas eux qui effectueront le traitement lié aux événements : ils se contenteront de déléguer ce traitement au composant parent, via une fonction de type "handler" passée via les `props`.

Par exemple pour le composant `Regions`, il faut déterminer ce que l'on fait lorsque l'utilisateur sélectionne une région (événement `onClick`).

Au niveau du composant `Regions` cela donne simplement :

```
const Regions = React.createClass({
  handleRegionClick(event) {
    this.props.onRegionChange(event.target.textContent);
  },

  render () {
    return (
      <div>
        {
          this.props.regions.map(region =>
            <div key={region} onClick={this.handleRegionClick}>
              {region}
            </div>
          )
        }
      </div>
    )
  }
})
```

Et au niveau du composant `WineApp` :

```
const WineApp = React.createClass({
  // ...
  handleRegionChange(region) {
    this.setState({
      selectedRegion: region
    });
    // TODO Recharger les vins de la région sélectionnée : this.loadWinesByRe
  },

  render() {
    return (
      ...
      <Regions regions={this.state.regions} onRegionChange={this.handleRegion
      ...
    }
  }
})
```

A vous de jouer !

Surtout ne restez pas bloqués ! N'hésitez pas à demander de l'aide aux organisateurs du workshop ou bien à jeter un oeil au code disponible dans la version corrigée ;-)

Pour le style de l'application, ne vous prenez pas la tête, l'enjeu n'est pas là :-) Pour

l'organisation en colonne, vous pouvez utilisez le framework CSS avalanche. Le CSS est disponible dans le dossier `public/css/avalanche.css`.

Exemple de grille avec Avalanche :

```
<div className="grid">
  <div className="1/4 grid__cell">
    <h2>Regions</h2>
    <!-- ... -->
  </div>
  <div className="1/3 grid__cell">
    <h2>Wine List</h2>
    <!-- ... -->
  </div>
  <div className="5/12 grid__cell">
    <h2>Wine Description</h2>
    <!-- ... -->
  </div>
</div>
```

Prochaine étape

Une fois cette étape terminée, vous pouvez aller consulter la version corrigée puis aller jusqu'à l'étape suivante