$

Etape 0 - Installation et premier composant

Package.json

Créez un nouveau dossier pour votre application. Une fois dans ce dossier lancez la commande `npm init`. Répondez aux diverses questions de l'assistant afin d'initialiser votre fichier `package.json`.

Une fois le fichier créé, installez les dépendances de la manière suivante :

```
npm install --save react@0.14.7 react-dom@0.14.7
```

avec cette commande, vous spécifiez à `npm` d'aller chercher la dernière version des paquets `react` et `react-dom` sur `npmjs.com`, de les installer en local dans le dossier `node_modules` local et de les déclarer comme dépendance dans le fichier `package.json` (via l'argument --save).

Une autre possibilité est de créez un fichier `package.json` et de déclarer manuellement les dépendances `react` :

```
{
    "name": "react-workshop",
    "description": "React Workshop",
    "version": "0.1.0",
    "dependencies": {
        "react": "0.14.7",
        "react-dom": "0.14.7"
    }
}
```

puis de lancer la commande `npm install` afin de télécharger localement les dépendances (elles se trouvent dans le répertoire `node_modules`)

Premier composant

wine.js

Dans le répertoire `src/components`, créez le fichier `wine.js` qui contient le code de notre premier composant, nommé `Wine` :

```
import React from 'react';

const WineStyle = {
  padding: 8,
  boxShadow: '0 1px 6px rgba(0,0,0,0.12), 0 1px 4px rgba(0,0,0,0.12)'
};

const Wine = React.createClass({
  propTypes: {
    name: React.PropTypes.string.isRequired
```

```
  },

  render() {
    return (
      <div style={WineStyle}>
          {this.props.name}
      </div>
    );
  }
});

export default Wine;
```

Ce premier composant est volontairement très simple (de type "hello world"), il retourne simplement le nom du vin dans un élément HTML `<div>`. Le nom du vin est passé au composant grâce à une propriété `name`. Cette propriété est définie comme étant de type `string` et obligatoire (partie `propTypes` du composant).

Un style est également appliqué au composant via l'attribut `style`.

Vous pouvez également utiliser directement la fonction `createElement` de l'API `react` :

```
render() {
    return React.createElement(
        'div',
        {style: WineStyle},
        this.props.name
    );
}
```

index.html

Dans le dossier `public`, créez une page HTML basique et ajoutez-y une `<div>` (possédant l'identifiant `main`) dans laquelle nous effectuerons le rendu de notre composant.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8"/>
    <title>React Workshop</title>
</head>
<body>
<h1>Wines</h1>

<div id="main"></div>

</body>
</html>
```

app.js

A la racine du répertoire `src`, créez le fichier `app.js` qui contient le code nécessaire au rendu du composant `Wine` dans la `<div>` créée précédemment :

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

```
import Wine from './components/wine';

ReactDOM.render(
    <Wine name="Château Chevrol Bel Air"/>,
    document.getElementById('main')
);
```

Vous pouvez également utiliser directement la fonction `createElement` de l'API `react` :

```
ReactDOM.render(
    React.createElement(
        Wine,
        {name:'Château Chevrol Bel Air'}
    ),
    document.getElementById('main')
);
```

Build avec Webpack

Nous utilisons l'outil Webpack afin de construire notre application.

En complément de Webpack, nous utilisons Babel, un compilateur Javascript qui permet d'utiliser les dernières nouveautés du langage. Dans notre cas, nous utilisons les plugins `react` et `es2015`.

Dans le fichier `package.json`, ajoutez les dépendances de développement nécessaires au build Webpack :

```
"devDependencies": {
    "webpack": "1.12.14",
    "babel-loader": "6.2.4",
    "babel-preset-es2015": "6.6.0",
    "babel-preset-react": "6.5.0"
}
```

vous pouvez évidemment les ajouter via la ligner de commande :

```
npm install --save-dev webpack@1.12.14 babel-loader@6.2.4 babel-preset-es2015
```

ici l'argument `--save-dev` indique que le dépendance doit être inscrite dans les dépendances du build et non du projet lui même

Créez le fichier `webpack.config.js` permettant de configurer Webpack et Babel :

```
var webpack = require('webpack');

module.exports = {
    output: {
        path: './public/js/',
        publicPath: '/js/',
        filename: 'bundle.js'
    },
    entry: {
        app: ['./src/app.js']
```

```
    },
    resolve: {
        extensions: ['', '.js', '.jsx']
    },
    module: {
        loaders: [
            {
                test: /\.js/,
                exclude: /node_modules/,
                loader: 'babel',
                query: {
                    presets: ['react', 'es2015']
                }
            }
        ]
    }
};
```

La configuration de Webpack est simple :

- Le point d'entrée est le fichier `src/app.js`
- Le fichier `bundle.js` est généré dans le répertoire `public/js`
- Le build exécute le lanceur `babel` avec les plugins `react` et `es2015`
    - *Remarque : les plugins babel peuvent également être définis dans le fichier de configuration `.babelrc`*

Vous pouvez ajouter les commandes Webpack sous forme de scripts dans le fichier `package.json`. Par exemple :

```
"scripts": {
    "bundle": "webpack -p --colors --progress"
}
```

Ainsi, la commande `npm run bundle` permet de construire le fichier `bundle.js`

Enfin, pensez à référencer le script `bundle.js` dans le fichier `index.html` :

```
<body>
<h1>Wines</h1>

<div id="main"></div>

<script src="/js/bundle.js"></script>
</body>
```

Exécution avec Webpack Dev Server

Afin de rendre la page `index.html` dans un navigateur, nous utilisons Webpack Dev Server.

Ajoutez la dépendance à `webpack-dev-server` dans le fichier `package.json` :

```
"devDependencies": {
    "webpack-dev-server": "1.14.1"
}
```

ou via la command `npm install --save-dev webpack-dev-server@1.14.1`

il faudra également rajouter un peu de configuration pour plus tard dans le fichier `webpack.config.js`

```
devServer: {
    historyApiFallback: true,
    proxy: {
        '/api/*': {
            target: 'http://localhost:3000'
        }
    }
}
```

cette ajout ne nous est pas tout de suite utile, mais le deviendra dans les prochaines étapes. Il permet a notre serveur de développement de proxyfier tous les appels à `http://localhost:8080/api/*` vers `http://localhost:3000/api/*` qui se trouve être notre serveur de données sur les vins (d'ailleurs n'oubliez pas d'aller lancer ce serveur dans le dossier `api` via la commande `npm start`). Cette astuce permet d'éviter a avoir à gérer des appels `CORS` en développement. La configuration `historyApiFallback` permet au serveur de renvoyer la page d'index en cas de page non trouvée. Cette astuce nous permettra d'utiliser facilement l'API `history` du navigateur.

Ajoutez un nouveau script permettant de lancer le serveur Webpack :

```
"scripts": {
    "start": "webpack-dev-server -d --colors --inline --content-base public"
}
```

Lancez enfin la commande `npm start` et ouvrez la page `http://localhost:8080`.

Modifiez le code du composant `Wine` et observez les modifications en live dans votre navigateur !

Attention

Si vous souhaitez travailler sur un environnement déporté, il sera nécessaire de pouvoir accéder au `webpack-dev-server` depuis l'exterieur. Par défaut, `webpack-dev-server` n'écoute que `localhost`. Afin d'éviter ce comportement, vous pouvez spécifier sur quel host `webpack-dev-server` doit écouter. Changez le script `start` par :

```
"scripts": {
    "start": "webpack-dev-server -d --colors --inline --content-base public -
}
```

ESLint

ESLint est un outil qui permet d'analyser votre code Javascript selon un certains nombre de règles.

Dans notre cas, nous allons l'utiliser avec le plugin `eslint-plugin-react` qui propose des règles spécifiques au développement de composants `react`.

Pour commencer, ajoutez les dépendances nécessaires dans le fichier `package.json` :

```
"devDependencies": {
    "eslint": "2.4.0",
    "eslint-plugin-react": "4.2.3"
}
```

ou via la commande `npm install --save-dev eslint@2.4.0 eslint-plugin-react@4.2.3`

Créez ensuite le fichier `.eslintrc` qui permet de configurer ESLint :

```
{
  "extends": "eslint:recommended",
  "env": {
    "browser": true,
    "node": true,
    "es6": true
  },
  "plugins": [
    "react"
  ],
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module",
    "ecmaFeatures": {
      "jsx": true,
      "experimentalObjectRestSpread": true
    },
  },
  "rules": {
    "react/display-name": 0,
    "react/forbid-prop-types": 1,
    "react/jsx-boolean-value": 1,
    "react/jsx-closing-bracket-location": 1,
    "react/jsx-curly-spacing": 1,
    "react/jsx-handler-names": 1,
    "react/jsx-indent-props": 1,
    "react/jsx-key": 1,
    "react/jsx-max-props-per-line": 1,
    "react/jsx-no-bind": 1,
    "react/jsx-no-duplicate-props": 1,
    "react/jsx-no-literals": 1,
    "react/jsx-no-undef": 1,
    "react/jsx-pascal-case": 1,
    "jsx-quotes": 1,
    "react/jsx-sort-prop-types": 1,
    "react/jsx-sort-props": 1,
    "react/jsx-uses-react": 1,
    "react/jsx-uses-vars": 1,
    "react/no-danger": 1,
    "react/no-did-mount-set-state": 1,
    "react/no-did-update-set-state": 1,
    "react/no-direct-mutation-state": 1,
    "react/no-multi-comp": 1,
    "react/no-set-state": 1,
    "react/no-unknown-property": 1,
    "react/prefer-es6-class": 0,
    "react/prop-types": 1,
    "react/react-in-jsx-scope": 1,
    "react/require-extension": 1,
    "react/self-closing-comp": 1,
    "react/sort-comp": 1,
    "react/wrap-multilines": 1
  }
}
```

- L'attribut `extends` permet d'hériter d'une configuration existante. `eslint:recommended` contient les règles recommandée par ESLint.
- La partie `env` permet de définir quelles variables globales sont potentiellement utilisées dans le code. Ici nous ajoutons celles du navigateur et celle de node.
- La partie `plugins` permet d'ajouter des plugins ESLint. Ici nous ajoutons le plugin `react`.
- La partie `ecmaFeatures` permet de définir les options du langage Javascript supportées lors de l'analyse. Ici nous activons la syntaxe JSX ainsi que les modules ES6.
- La partie `rules` permet de définir les règles à appliquer lors de l'analyse du code. Pour plus de détails sur les règles disponibles : https://www.npmjs.com/package/eslint-plugin-react

Il est possible d'exclure certains fichiers ou dossiers de l'analyse, grâce au fichier `.eslintignore`. Exemple :

```
node_modules
webpack.config.js
public
```

Enfin, ajoutez un script dans le fichier `package.json` permettant d'exécuter ESLint grâce à la commande `npm run lint` :

```
"scripts": {
  "lint": "eslint src"
}
```

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 ;-)

Prochaine étape

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