Etape 1 - Testing

Pré-requis

L'étape 0 du workshop a permis de développer un composant `react` basique `Wine` :

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

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

Le code disponible ici correspond au résultat attendu de l'étape 0. L'objectif de l'étape 1 est de le compléter pour ajouter le test unitaire du composant `Wine`.

Pour lancer l'application de l'étape 0, exécutez la commande `npm start` (après avoir fait un `npm install`). Ouvrez ensuite l'URL http://localhost:8080 dans votre navigateur.

Test unitaire

Les tests unitaires sont primordiaux dans le développement. Ils ne doivent en aucun cas être négligés, c'est pourquoi nous les introduisons dès le début du workshop.

L'objectif est de mettre en place le test unitaire du composant `Wine`. Pour cela, nous allons nous appuyer sur les librairies suivantes :

- react-addons-test-utils : addon `react` facilitant les tests de composants `react`.
- jsdom : librairie implémentant les standards DOM et HTML, qui permettra de créer un document HTML dans lequel faire le rendu des componsants à tester.
- Chai : librairie d'assertions, orientée BDD/TDD.
- Mocha : framework Javascript de tests unitaires.
- babel-register : permet d'utiliser Babel lors de l'exécution des tests avec Mocha.

Commencez par ajouter ces librairies au fichier package.json :

```
"devDependencies": {
    "babel-register": "6.7.2",
    "jsdom": "8.1.0",
    "mocha": "2.4.5",
    "chai": "3.4.1",
    "react-addons-test-utils": "0.14.7"
}
```

vous pouvez évidemment les ajouter via la ligner de commande :

```
npm install --save-dev babel-register@6.7.2 jsdom@8.1.0 mocha@2.4.5 chai@3.4.
```

Ajoutez la configuration nécessaire à `babel-register` dans le fichier `.babelrc` :

```
{
  "presets": ["es2015", "react"]
}
```

Cette configuration est globale pour babel et sera utilisée si aucune autre configuration n'est passée aux outils babel. Ainsi vous pouvez enlever la partie configuration du `babel-loader` dans `webpack.config.js` afin que tous vos outils utilisent la même configuration babel.

Créez un dossier `tests` dédiés aux tests unitaires de vos composants. Ecrivez ensuite le test du composant `Wine` dans un fichier `tests/components/wine.spec.js`, en utilisant :

- la syntaxe Mocha pour décrire le test,
- ReactTestUtils pour effectuer le rendu et parcourir l'arbre DOM du composant `Wine`,
- Chai pour vérifier le texte affiché.

```
import React from 'react';
import { expect } from 'chai';
import ReactTestUtils from 'react-addons-test-utils';

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

describe('Wine', () => {
  it('affiche le nom du vin', () => {
    const wine = ReactTestUtils.renderIntoDocument(<Wine name="Un bon Bourgog
    const div = ReactTestUtils.findRenderedDOMComponentWithTag(wine, 'div');
    expect(div.textContent).to.be.equal('Un bon Bourgogne');
  });
});
```

Pour s'exécuter correctement, le test précédent nécessite de disposer globalement des objets `window` et `window.document`, ainsi que de la fonction `window.document.createElement`. Pour cela, créez un fichier `bootstrap.js` qui se base sur la librairie `jsdom` pour créer l'environnement DOM nécessaire au bon fonctionnement de ReactTestUtils.

```
import jsdom from 'jsdom';

export function bootstrapEnv(body = '') {
  const doc = jsdom.jsdom(`<!doctype html><html><body>${body}</body></html>`)
  const win = doc.defaultView;
  function propagateToGlobal(window) {
    for (const key in window) {
      if (!window.hasOwnProperty(key)) continue;
      if (key in global) continue;
      global[key] = window[key];
    }
  }
  global.document = doc;
  global.window = win;
  propagateToGlobal(win);
  console.log('\nENV setup is done !!!');
```

```
}
```

Créez enfin un fichier `index.js`, point d'entrée permettant d'exécuter l'ensemble des tests unitaires :

```
import { bootstrapEnv } from './bootstrap';

bootstrapEnv();

const tests = [
  require('./components/wine.spec.js')
];
```

Exécution des tests via NPM

Ajoutez un nouveau script dans le fichier `package.json` permettant de lancer les tests à l'aide de la commande `npm test` :

```
"scripts": {
    "test": "mocha --compilers js:babel-register tests/index.js"
}
```

Vous pouvez également préciser à ESLint qu'il doit désormais également traiter le dossier `tests` :

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

Pour plus tard ...

`ReactTestUtils` permet également de simuler des clics sur des éléments du DOM :

```
ReactTestUtils.Simulate.click(button);
```

Si vous voulez aller plus loin dans les tests, il pourrait être intéressant de regarder du côté d'enzyme, un utilitaire de test pour `react` créé par Airbnb et qui permet de manipuler plus facilement votre arbre de composant pour le tester. En effet `enzyme` s'inspire de l'API de `jQuery` pour pouvoir requêter l'arbre de composants.

```
import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';

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

describe('Wine', () => {
  it('affiche le nom du vin', () => {
    const wine = shallow(<Wine name="Un bon Bourgogne" />);
    expect(wine.contains(<div>Un bon Bourgogne</div>)).to.be.true;
    expect(wine.find('div.wine')).to.have.length(1);

  });
```

```
});
```

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