Révision et complément sur Redux

Ce TP est un travail de révision et d'approfondissement des notions de Redux dans React.

Contexte

Vous allez développer une application **Num-pad** de calcul mental. Chaque réponse exacte de l'utilisateur augmentera son score de +1, si l'utilisateur se trompe son score sera inchangé.

Conseil

- Utilisez l'extension Chrome React pour le développement : https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

Puis allez dans l'inspecteur de votre navigateur pour l'utiliser.

Contraintes techniques

- Vous utiliserez **create-react-app** pour définir un squelette d'application.
- Vous utiliserez **Redux** et le module **react-redux** pour utiliser Redux avec React.
- Vous utiliserez bootstrap twitter pour les CSS/HTML, pensez à l'installer dans l'application :

```
npm i bootstrap
```

- Vous découperez au maximun le projet en Components.
- Vous utiliserez un reducer **caclul.js** que vous placerez dans un dossier **reducers**.
- Vous centraliserez les constantes et les actions dans deux dossiers séparés :

```
num-pad/
    src/
        actions/
            actions.js
        constants/
            action-type.js
    ...
```

Exemple d'action dans le fichier actions.js :

```
import {
SEND_NUMBER,
/* d'autres constantes d'actions ...*/
} from '../constants/action-type'

export const sendNumber = (payload) => {
    return {
        type: SEND_NUMBER, payload
    }
};

// ...
```

La valeur **payload**, ci-dessus, dans le reducer sera récupérée dans l'objet **action** du Reducer (fichier **calcul.js**) :

```
// reducer extrait
export default (state = stateInit, action = {}) => {
    switch (action.type) {

        case SEND_NUMBER:

            return {
                ...state,
                values : [ ...state.values, action.payload  ]
            }
```

Contextualiser l'application

Dans le fichier **index.js** "contextualisez" l'application avec le **store** de Redux, *rappelons qu'il faut deux choses pour utiliser Redux dans React : Redux et le module react-redux* :

```
import React from 'react'
import { render } from 'react-dom';
import { createStore } from 'redux';
import { Provider } from 'react-redux';
// reducer
import calcul from './reducers/calcul';

import App from './App';
import 'bootstrap/dist/css/bootstrap.css';

const store = createStore(calcul);
```

```
render(
  // Contextualiser le sotre pour l'application
  // c'est obligatoire.
  <Provider store={store}>
    <App />
  </Provider>
  ,
  document.getElementById('root')
);
```

Première partie afficher le pad numérique

Développez le **num pad**, pavé numériques de 10 chiffres. Faites en sorte que lorsqu'on clique sur l'un de ses chiffres on "récupère" cette valeur dans le state de Redux.

Pour contrôler que tout marche bien dans le component App.js affichez dans un console.log les props que vous récupèrez en lecture à partir de Redux (méthode connect du module react-redux).

Une fois que cette fonctionnalité est en place, affichez dans un component **Message** le résultat de la saisie. Pensez à concaténer les chiftres pour les afficher dans ce component.

Deuxième partie

Mettez en place un bouton permettant d'effacer la proposition numérique de l'utilisateur.

Troisième partie

Générez des multiplications distinctes, créez l'algorithmique dans un fichier multiplications.js dans un dossier utils de l'application. Vous proposerez une multiplication de manière aléatoire à l'utilisateur en passant par les states de Redux.

Faites en sorte qu'à chaque fois que l'on propose une multiplication à trouver, le nombre de multiplications que vous avez généré décroit.

Quatrième partie

Affichez des messages d'échec et de succès lorsque l'utilisateur respectivement se trompe et réussi dans un unique component spécifique.

Rappel : vous incrémenterez le score de l'utilisateur en fonction des réussites +1 si il réussit et rien sinon.

Cinquième partie

Lorsqu'il n'y a plus de multiplication à afficher alors la partie se termine. Affichez dans ce cas un component spécifique avec le score de l'utilisateur et un bouton pour réinitialiser le jeu.

Sixième partie

Imagniez maintenant un système de persistance avec Firebase pour sauvegarder le score de l'utilisateur.

Pour terminer

Vous allez maintenant passer en production. Tapez la ligne de code suivante :

```
npm run build
```

React "compile" votre code (optimise et mimifie les sources). Un dossier build est alors créé. Placez-vous dans le dossier **build** et tapez alors la ligne de code suivante :

```
php -S localhost:7000
```

Vérifiez que votre application est fonctionnelle.

Wireframes

Ci-dessous quelques wireframes pour vous aider à mettre en place les fonctionnalités de l'application, ce sont des exemples vous n'êtes pas obligé de les suivre à la lettre :

Si l'utilisateur se trompe on affiche un feedback avec la bonne réponse, une alerte s'affiche également mentionnant si on a réussi ou si on c'est trompé, ici on c'est trompé...

Dans le cas où on ne se trompe pas le message change :

Dans tous les cas on rappelle l'opération que l'on vient de faire et on affiche la suite à calculer.

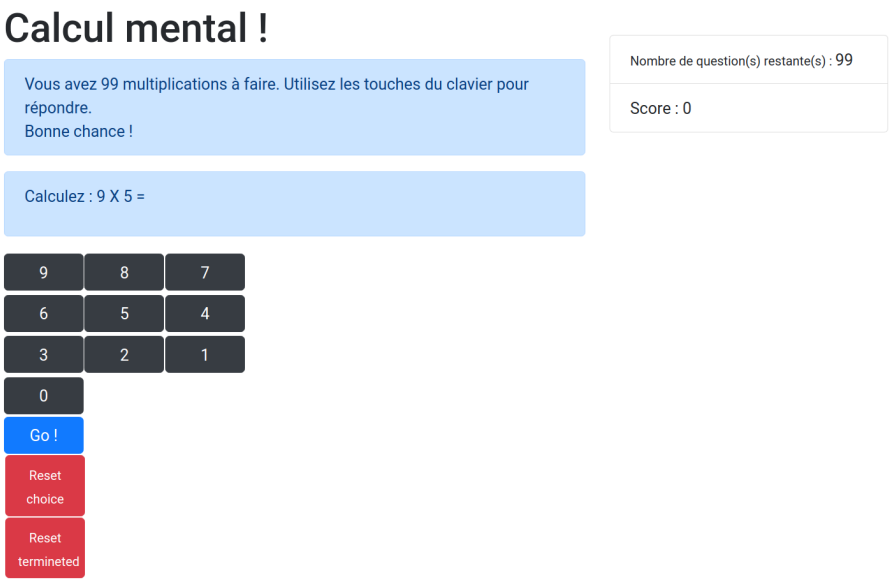

Figure 1: pad

Figure 2: feedback

Figure 3: feedback