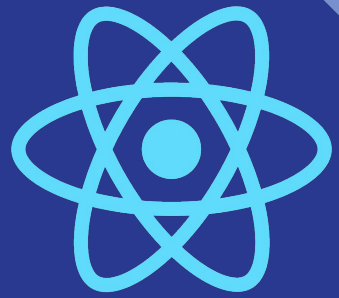
Formation React-Native
Gestion avancée des données



Bastien NICOLAS
bastien.nicolas@edu.esiee-it.fr

CODING FACTORY by ESIEE [tech]

Introduction

- Données persistantes avec AsyncStorage
- Partager des données dans l'application avec Redux
- Exercices

Données persistantes avec AsyncStorage

- AsyncStorage nous permet de conserver des données après que l'application soit fermée
- On peut entre autre sauvegarder (setItem), récuperer (getItem), supprimer (removeItem) et fusionner (mergeItem) des données à l'aide de clés/valeurs
- Ce module ne gère que des données textes !
- Des objets ou tableaux JS peuvent être convertis en JSON (texte) avant d'être enregistrés : JSON.parse(), JSON.stringlify()
- Doc : https://react-native-async-storage.github.io/async-storage/docs/install/
- Il existe d'autres solution pour persister des données en React-native mais elles utilisent souvent AsyncStorage

Données persistantes avec AsyncStorage

```
import AsyncStorage from '@react-native-async-storage/async-storage';
```

```
const saveData = async data => {
  const json = JSON.stringify(data);
  await AsyncStorage.setItem('myData', json);
};
```

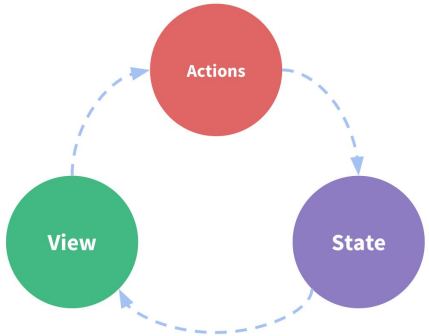
```
const loadData = async () => {
  const json = await AsyncStorage.getItem('myData');
  return JSON.parse(json);
};
```

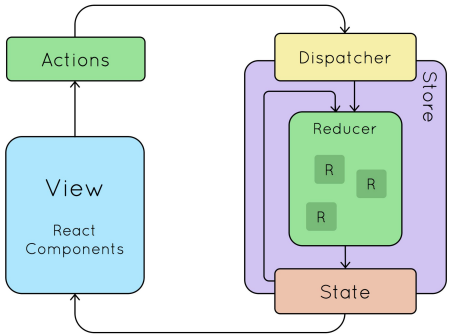
Partager des données dans l'application avec Redux

- useState permet de gérer des données au sein d'un composant
- Les props nous permettent de passer des données d'un composant parent à un enfant
- Il est compliqué de partager des données dans toute l'application de cette manière
- Il existe plusieurs solutions en React pour partager des données
  - Redux
  - MobX
  - React Context
- Nous allons utiliser Redux qui est le plus ancien et le plus utilisé par la communauté

Partager des données dans l'application avec Redux

- Le principe de redux consiste à avoir un **store** contenant des **states** à l'échelle de l'application
- Depuis nos composants, nous pouvons récupérer les states qui nous intéressent, le composant est mis à jour quand les states évoluent
- Depuis nos composants, nous pouvons **dispatch** des **actions** vers des **reducers** qui vont mettre à jour nos states
- L'ensemble de l'application doit être contenue dan un composant <Provider> fournir par react-redux





Partager des données dans l'application avec Redux

App

Store

```
import {createStore} from 'redux';

import reducer from './reducer';

export const store = createStore(reducer);
```

```
import {Provider} from 'react-redux';

import {store} from './src/redux/store';

const App = () => {
  return (
    <Provider store={store}>
      <NavigationContainer>
        {
          //... reste de l'application
        }
      </NavigationContainer>
    </Provider>
  );
};

export default App;
```

Partager des données dans l'application avec Redux

Reducer (c'est une fonction) :

- Il reçoit deux paramètres :
  - La valeur actuelle du state (si il est vide on peut définir une valeur initiale)
  - L'action à traiter (avec un type et des données)
- Il renvoie la nouvelle valeur du state après avoir traiter l'action

```
const initialState = {
  incrementValue: 0,
};

const increment = (state, action) => {
  return {
    ...state,
    incrementValue: state.incrementValue + action.value,
  };
};

const decrement = (state, action) => {
  return {
    ...state,
    incrementValue: state.incrementValue - action.value,
  };
};

const reducer = (state : {incrementValue: number} = initialState, action) => {
  switch (action.type) {
    case 'increment':
      return increment(state, action);
    case 'decrement':
      return decrement(state, action);
    default:
      return state;
  }
};

export default reducer;
```

Partager des données dans l'application avec Redux

Deux hooks fournis par react-redux :

- useSelector : permet de récupérer un state depuis le store
- useDispatch : récupère la fonction dispatch qui nous permet d'envoyer des actions aux reducers

```
import {useDispatch, useSelector} from 'react-redux';

const Increment = () => {
  const incrementValue = useSelector( selector: s => s.incrementValue);
  const dispatch = useDispatch();

  const increment = useCallback( callback: () => {
    dispatch({type: 'increment', value: 1});
  }, deps: [dispatch]);

  const decrement = useCallback( callback: () => {
    dispatch({type: 'decrement', value: 1});
  }, deps: [dispatch]);

  return (
    <View style={styles.container}>
      <Text>Composant Increment</Text>
      <Text style={styles.text}>{incrementValue}</Text>
      <View style={styles.btnContainer}>
        <TouchableOpacity style={styles.btn} onPress={increment}>
          <Text>Incrémenter</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.btn} onPress={decrement}>
          <Text>Décrémenter</Text>
        </TouchableOpacity>
      </View>
    </View>
  );
};

export default Increment;
```

Partager des données dans l'application avec Redux

En appelant les composants de cette manière, les deux instances de <Increment /> vont rester synchronisés car ils partagent un state commun

```
const MyScreen = () => {
  return (
    <SafeAreaView style={styles.container}>
      <Increment />
      <Increment />
    </SafeAreaView>
  );
};
```

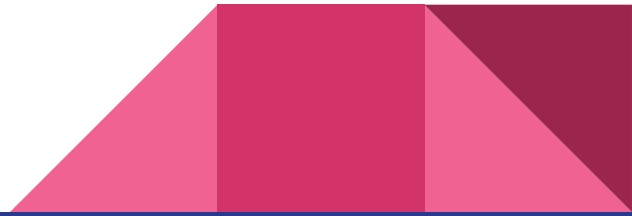
Partager des données dans l'application avec Redux

Il est possible de combiner plusieurs reducers dans le store

```
import {combineReducers, createStore} from 'redux';

import authReducer from './reducers/authReducer';
import dataReducer from './reducers/dataReducer';

const rootReducer = combineReducers( reducers: {
  auth: authReducer,
  data: dataReducer,
});

export const store = createStore(rootReducer);
```

Partager des données dans l'application avec Redux

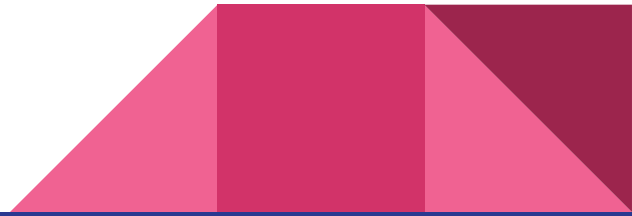
- Les states redux permettent de gérer des données qui vont servir à plusieurs endroits dans l'application (authentification, données utilisateur, ...)
- Il est possible de rendre des données de redux persistantes en combinant redux et AsyncStorage avec le module redux-persist -> https://github.com/rt2zz/redux-persist
- Certains développeurs utilisent une architecture basée sur Redux -> il existe des templates pour ça



Exercice 1 - Rendre les données de la TodoList persistantes

A partir de la TodoList du cours précédent :

- Les données de la TodoList sont stockées dans une clé de l'AsyncStorage ('todoList' par exemple)
- Quand les données de la TodoList changent, écraser les données de l'AsyncStorage avec son nouveau contenu
- A l'ouverture de la page, charger les données de la TodoList depuis l'AsyncStorage



Exercice 2 - Gérer les données de la TodoList avec Redux

A partir de la TodoList de l'exercice précédent :

- Le state de la TodoList doit être géré par Redux
- Faire un reducer qui gère :
  - Une action "addTodo" qui permet d'ajouter une Todo
  - Une action "removeTodo" qui permet de supprimer une Todo
- Bonus : Gérer la persistance des données avec redux-persist