React Native

Dominique BESSON

SOMMAIRE

PRÉ-REQUIS

1

- ➔ NodeJs 8+ et npm à installer
- ➔ npm install -g expo-cli
- ➔ Expo Client (sur un Android ou un IOS)
- ➔ Un IDE (Visual Code avec l'extension "React Native Tools" par exemple)

2

- ➔ Python 2
- ➔ JDK 8+
- ➔ npm install -g react-native-cli
- ➔ Watchman pour Mac et Linux
- ➔ Android studio
  - ◆ SDK
  - ◆ NDK (Optionnel)
  - ◆ Android SDK Platform
  - ◆ Performance (Intel ® HAXM)
  - ◆ Android Virtual Device
- ➔ Configurer les variables d'environnement

https://facebook.github.io/react-native/docs/getting-started

PETITE HISTOIRE

Origines

- ➔ Framework de développement mobile Open-Source
- ➔ Prototypé lors d'un hackathon interne de Facebook (2013)
- ➔ V1 en 2015 lors du React.js Con.

Principe

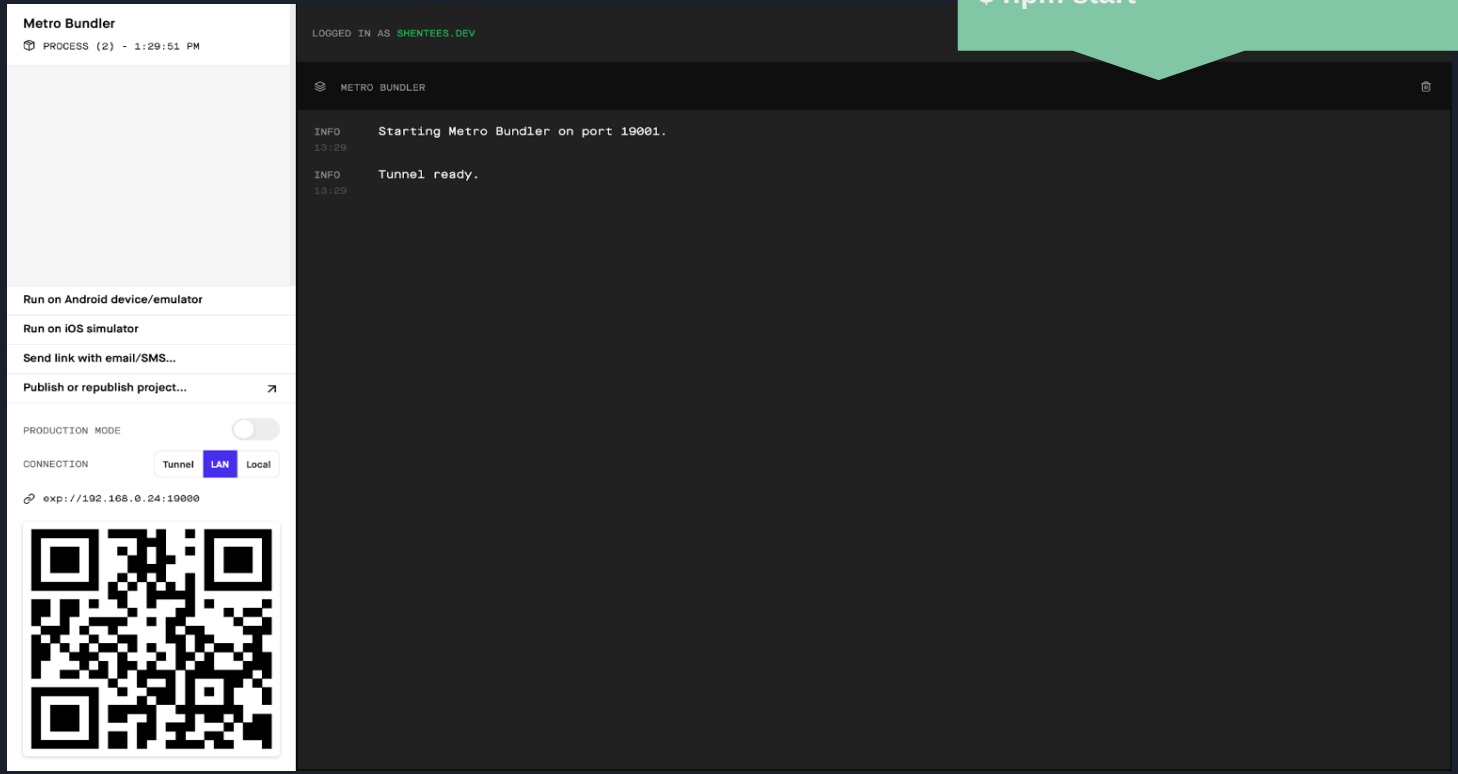
- ➔ Même principe que React (DOM Virtuel)
- ➔ Code JS exécuté dans le mobile via un thread JS
- ➔ Manipule des vues/objets natifs

Résultat

- ➔ Framework Cross-Platform
- ➔ Des performances comparables au natif
- ➔ Productivité augmenté, tests facilités, Simplicité
- ➔ Comparatif (Xamarin, IONIC)

EXPO CLIENT

```
$ npm i -g expo-cli
$ expo init hello-world
$ cd hello-world
$ npm start
```



HELLO WORLD

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

No Errors

Show Details

COMPONENT

- Élément rattaché à une/plusieurs vues native(s)
  - vues
- Certains sont spécifiques à des plateformes
- Les basics de react-native et les spécifiques de la communauté
- Import pour utilisation
- Manipulation à travers
  - State
  - Props
  - LifeCycle

State

- Représente l'état interne
- Est synchronisé avec l'état natif
- Manipuler seulement avec setState pour l'écriture

Props

- Représente l'état public, accessible à tous
- Est peuplé avec notamment les propriété lors de l'appel de la création
- Manipuler à peu près comme on veut
- Contient généralement les callbacks

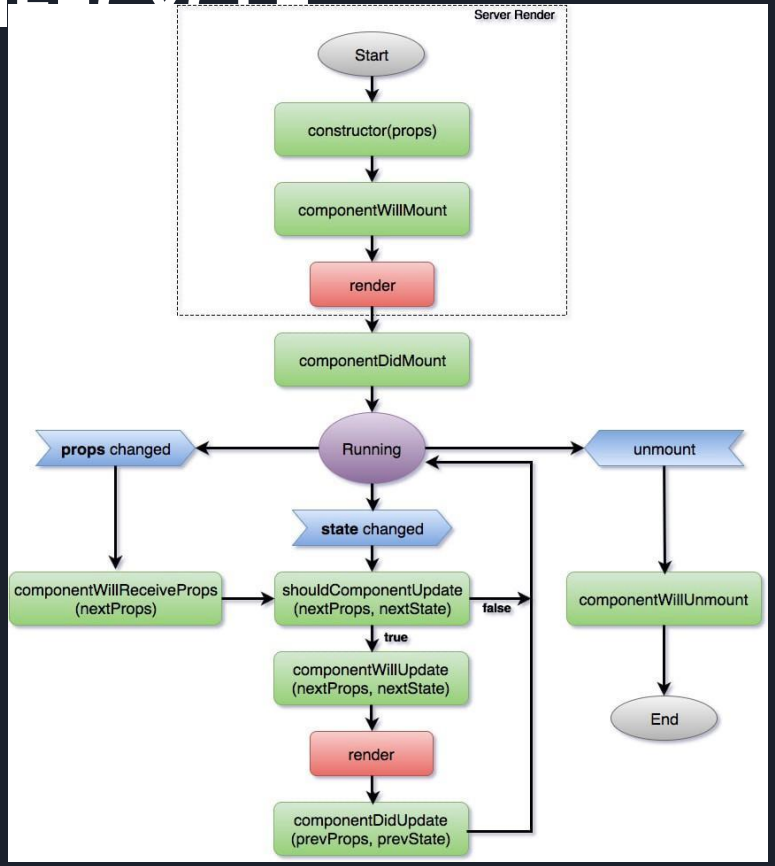
CALLBACK, PROPS, STATE

Callbacks

Fonctions appelées durant le cycle de vie d'un component pour tenir au courant de son état lors d'un événement.

```
export default class App extends React.Component {
  state = {
    clicked: false
  }

  render() {
    const text = this.state.clicked ? 'Cliqué !!!' : 'Pas cliqué'

    return (
      <View style={styles.container}>
        <Text style={styles.text} >{text}</Text>
        <Text style={styles.button1} style={styles.button1}
          onPress={() => {
            // Mon code réagissant à l'évènement
            this.setState({clicked: false})
          }}
        >Pas cliqué</Text>

        <Text style={styles.button2}
          onPress={() => this.setState({clicked: true})}>Cliqué</Text>
      </View>
    );
  }
}
```

COMPONENT - LIFECYCLE



STYLE

- Tous les components basiques ont une propriété “style”
- style créé avec l’objet “StyleSheet” du module ‘react-native’
  - StyleSheet.create()
- propriétés similaire au CSS

- Le positionnement est fait :
- en relatif par rapport au parent
- via l’algorithme FlexBox
  - flex: proportion prise sur l’espace disponible
  - flexDirection: indique l’axe de répartition des enfants sur l’axe primaire
  - justifyContent: indique la manière dont les enfants se répartissent cet axe
  - alignItems: indique la manière dont les enfants se répartissent sur l’axe secondaire

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: "row",
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  text:{
    backgroundColor: 'green',
  },
  button1:{
    backgroundColor: 'yellow',
  },
  button2:{
    backgroundColor: 'blue'
  }
});
```

PROMISE, ASYNC, AWAIT

```
longProcess(){
  return new Promise((resolve, fail) => {
    /*
    Grosse opération asynchrone qui va utiliser resolve ou fail
    */

    // sinon opération synchrone utilisant résolve ou fail
    if(true)
      resolve("réussi !");
    else
      fail("Raté :p");
  });
}

randomCallback(){
  this.longProcess()
    .then((result) => {
      // opération basée sur le résultat
    })
    .catch((err) => {
      // opération basée sur une erreur
    })
}
```

```
longProcess(){
  return new Promise((resolve, fail) => {
    /*
    Grosse opération asynchrone qui va utiliser resolve ou fail
    */

    // sinon opération synchrone utilisant résolve ou fail
    if(true)
      resolve("réussi !");
    else
      fail("Raté :p");
  });
}

async randomCallback(){
  try{
    const result = await this.longProcess()
  }catch(exeption){
    // Gérer le fail ici
  }
}
```

FETCH API

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue',
  }),
});
```

```
longProcess3(){
  return fetch('https://facebook.github.io/react-native/movies.json');
}

async randomCallback3(){
  try{
    const result = await this.longProcess3()
  }catch(exeption){
    // Gérer le fail ici
  }
}
componentDidMount(){
  this.randomCallback3()
}
```

REACT-NAVIGATION

- ➔ Gère la logique “Routing” de l’app
- ➔ **npm install --save react-navigation**

```
import {createStackNavigator, createAppContainer} from 'react-navigation';
import HomeScreen from './Home'
import InfoScreen from './Info'

const MainNavigator = createStackNavigator({
  Home: {screen: HomeScreen},
  Info: {screen: InfoScreen},
});

const App = createAppContainer(MainNavigator);

export default App;
```

```
render() {
  const text = this.state.clicked ? 'Cliqué !!!' : 'Pas cliqué'

  return (
    <View style={styles.container}>
      <Text style={styles.text} >{text}</Text>
      <Text style={styles.button1} style={styles.button1}
        onPress={() => {
          // Mon code réagissant à l'évènement
          this.setState({clicked: false})
        }}
      >Pas cliqué</Text>

      <Text style={styles.button2}
        onPress={() =>
          this.props.navigation.navigate('Info', {name: "lol"})
      }>Cliqué</Text>
    </View>
  );
```

REACT-NATIVE ELEMENTS

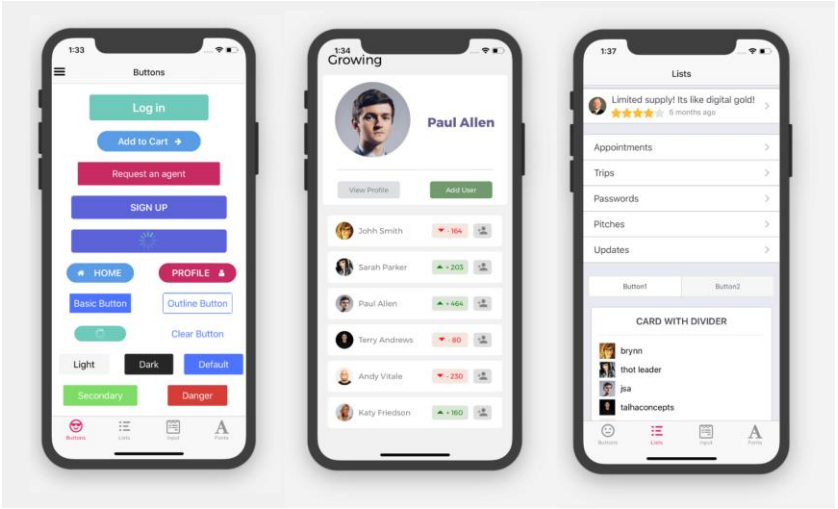
- UI Toolkit open source
- Ensemble de composants prêt à utiliser
- Large choix
- Grosse communauté
- Nécessite parfois un linking manuel

Bouton | Avatar | Icon

Header | SearchBar | Text

Et pleins d'autres encore

$ npm install --save react-native-elements



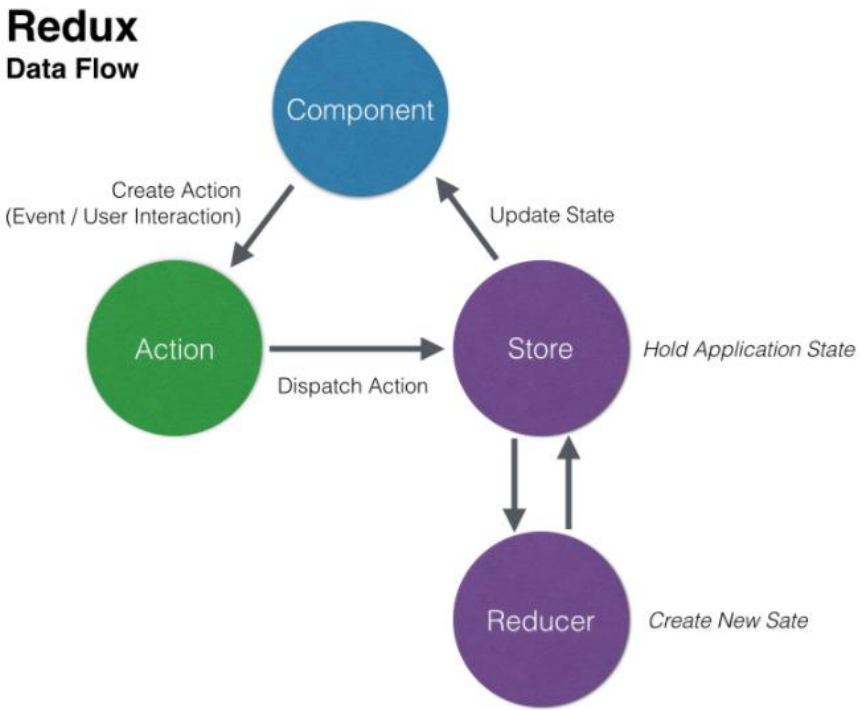
REDUX

- ➔ Librairie JS
- ➔ Gère un State global commun

Fonctionnement

- ➔ la vue lance une action pour changer l'état global
- ➔ le Store intercepte, passe aux différents reducers pour traiter l'action et modifier le state
- ➔ Les vues ayant un callback sur le state sont informées de la MAJ

$ npm install --save redux



REDUX

```
import { createStore } from 'redux'
import textReducer from './Reducer'

const store = createStore(textReducer)

export default class Init extends React.Component {

  componentDidMount(){
    this.props.store.subscribe(() => console.log(store.getState()))
  }

  render() {
    return (
      <RandomComponent />
    );
  }
}

<Init store={store} />
```

Action

```
const action = {
    type: 'TEXT',
    payload: this.text
}

this.props.store.dispatch(action)
```

Reducer

```
const textReducer = (state = {}, action) => {
  switch(action.type){
    case 'TEXT':
      return {...state, text: action.payload}
    default:
      return state
  }
}
```

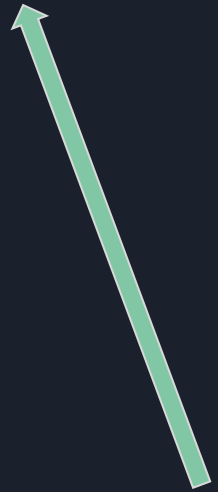
Le Store est à passer à chaque vue à connecter !

REACT-REDUX

$ npm install --save react-redux

```
import { connect } from 'react-redux'

function mapStateToProps(state){
    return {
        text: state.text
    }
}

export default connect(mapStateToProps)(App)
```

```
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import textReducer from './Reducer'
import App from './App'

const store = createStore(textReducer)

export default class Begin extends React.Component {
  render() {
    return (
      <Provider store={store}>
        <App />
      </Provider>
    );
  }
}
```



Action :

```
const action = {
    type: 'TEXT',
    payload: this.text
}

this.props.dispatch(action)
```

Reducer :

```
const textReducer = (state = {}, action) => {
  switch(action.type){
    case 'TEXT':
      return {...state, text: action.payload}
    default:
      return state
  }
}
```

ASYNCSTORAGE

- unencrypted, asynchronous, persistent, key-value storage system
- sur IOS
  - Enregistré via un dictionnaire
  - Dans un fichier séparé pour les enregistrements plus largeS
- sur Android
  - SQLite
  - RocksDB
- chaque méthode retourne une Promise

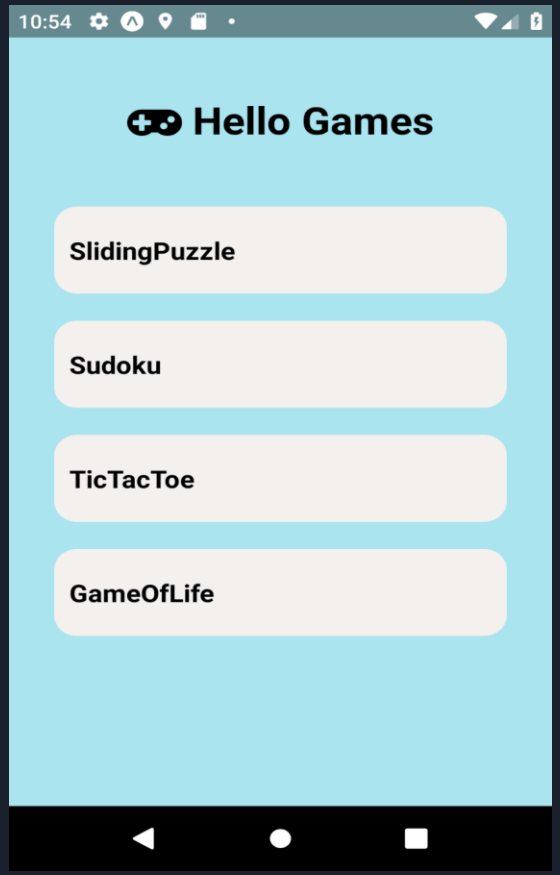
Import { AsyncStorage } from 'react-native'

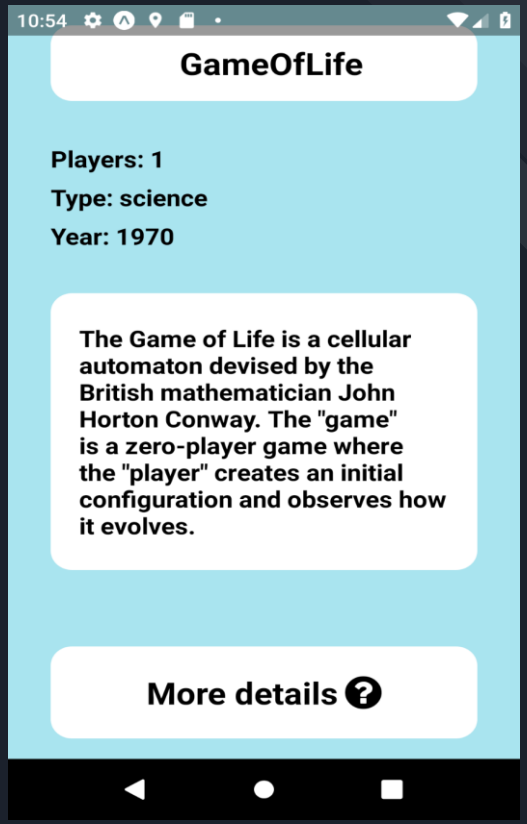
```
_storeData = async () => {
  try {
    await AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.');
  } catch (error) {
    // Error saving data
  }
};
```

```
_retrieveData = async () => {
  try {
    const value = await AsyncStorage.getItem('TASKS');
    if (value !== null) {
      // We have data!!
      console.log(value);
    }
  } catch (error) {
    // Error retrieving data
  }
};
```

PROJET

https://androidlessonsapi.herokuapp.com/api/help/


10:54
Hello Games
SlidingPuzzle
Sudoku
TicTacToe
GameOfLife


10:54
GameOfLife
Players: 1
Type: science
Year: 1970
The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway. The "game" is a zero-player game where the "player" creates an initial configuration and observes how it evolves.
More details


10:54
https://en.m.wikipedia.org/wi
WIKIPEDIA
Conway's Game of Life
"Game of Life" redirects here. For other uses, see Game of Life (disambiguation).
"Conway game" redirects here. For Conway's surreal number game theory, see Surreal number.
A single Gosper's glider gun creating "gliders"