Workshop

Titre: *Créer votre première application mobile avec React Native*
Durée: *1H30*

Introduction

React Native est un framework créé par Facebook à la suite d'un Hackaton en 2015. Il est basé sur **React, une librairie Javascript** développée deux ans auparavant par un ingénieur Facebook.

Pendant ce workshop, découvrez les bases de la programmation mobile avec React Native. Le but de cette activité est de concevoir un **générateur de mot de passe**. Les participants verront les points suivant :

- Appréhender les stratégies de développement d'une application mobile (oral)
    - Pourquoi avoir une application mobile en plus d'un site web ?
    - Quelle est la différence entre **native** et **hybride** ?

- Initialisation d'une application React

- Créer sa première **vue responsive**

- Génération de mot de passe en utilisant une **API**

Les pré-requis

Pour développer avec React Native, vous allez avoir besoin de Node.js, ainsi que son gestionnaire de paquets **NPM** (Node Package Manager, qui permet d'installer des outils de développement).

Une fois Node installé, il faut installer **Expo**. Expo est un outil qui permet entre autres de créer des applications React Native.

```
npm install -g expo-cli
```

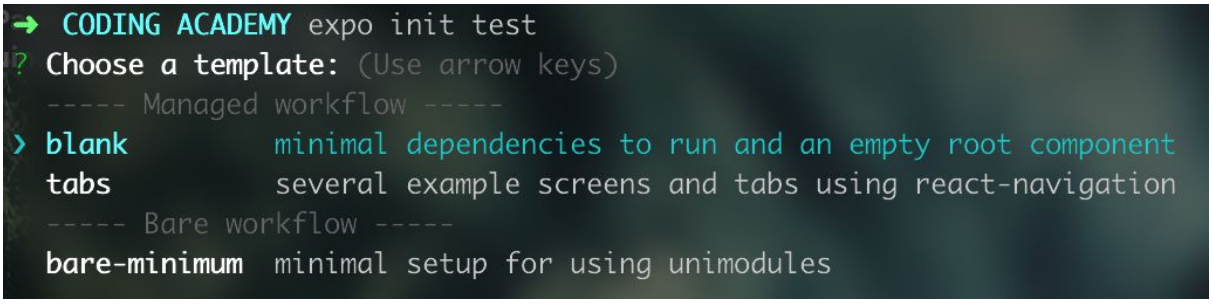
En plus d'être un outil pour créer des applications React Native, Expo nous offre aussi une application hôte qui va nous permettre d'**afficher en temps réel le rendu de l'application**.
Si vous avez un smartphone, vous pouvez télécharger l'application Expo sur iOS / Android. Sinon vous pouvez soit télécharger un simulateur de téléphone sur votre ordinateur ou alors demander un rendu avec le smartphone d'un responsable.

Initialisation du projet

Nous allons créer notre projet **genPass** qui je le rappelle est une application mobile qui génère un mot de passe. Pour cela, nous allons utiliser *expo* :

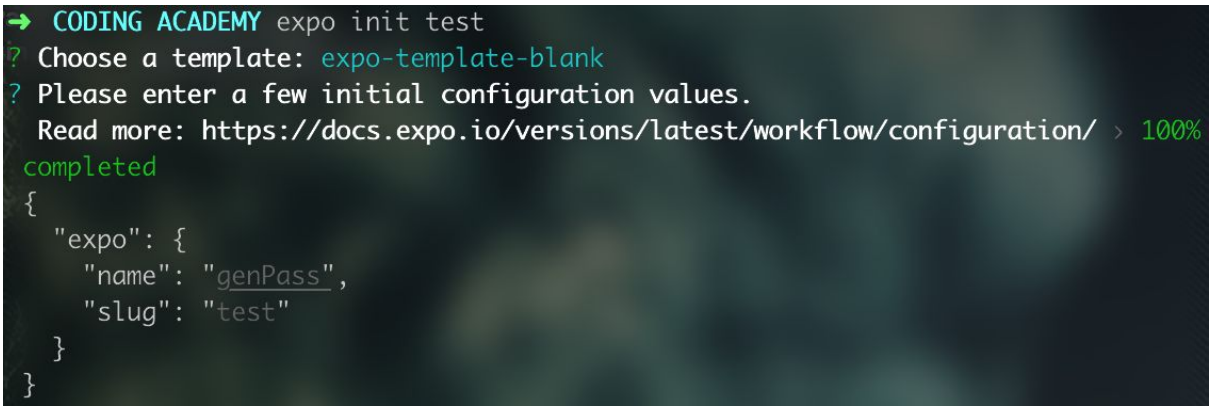
```
expo init genPass
```

Pendant l'exécution de cette commande, expo vous demande quelques informations sur votre projet.
Expo vous demande dans un premier temps avec quelle base vous souhaitez commencer



On va partir d'un projet vierge, sélectionnez donc **'blank'**.

Expo vous demande ensuite de donner un nom à votre projet, dans notre cas **genPass**



Votre projet et maintenant créé !
Je vous invite à aller dans le dossier genPass, et à faire la commande

```
npm start
```

npm start va lancer votre application par l'intermédiaire de **Expo**. Il va ensuite vous générer un **QR Code** que vous pouvez scanner avec votre smartphone. La méthode diffère en fonction de votre appareil (iOS ou Android)

Si vous avez un iPhone :

- Ouvrir l'application Appareil Photo
- Scanner le QR Code généré dans votre terminal, vous êtes redirigé vers l'application Expo

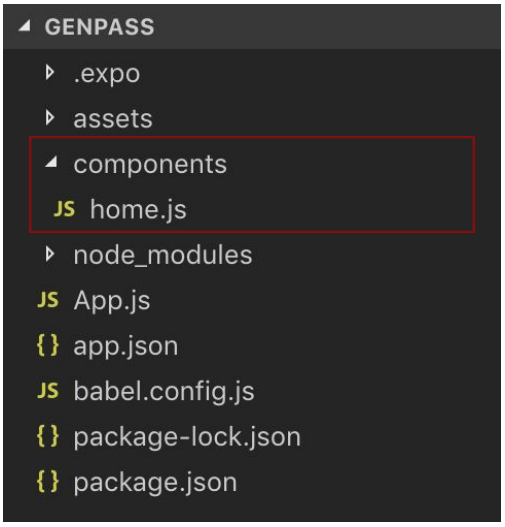
Si vous avez un Android :

- Ouvrir Expo
- Cliquer sur Scan QR Code
- Cibler le QR Code

Créer un component

React Native est basé sur un système de **component**. Un component est un **élément graphique affiché par l'interface utilisateur**. Toutes les vues (pages) React sont entièrement composées de components. Cette notion de component est très importante. Elle permet de découper nos interfaces utilisateur en **pièces indépendantes** et **réutilisables**, et de penser à chaque pièce séparément.

Nous allons créer un dossier *components* à la racine de notre projet et y créer un fichier *home.js*, comme ceci




```
import React from 'react';

export default class Home extends React.Component {

  constructor(props) {
    super(props);
  }

}
```

La classe *Home* **hérite** de *React.Component*, c'est à dire que toutes les variables et fonctions de *React.Component* sont **accessibles** depuis votre classe Home. On dit que Home est **enfant** de React.Component.

Comme son nom l'indique, le **constructor** permet d'instancier un objet de la classe, autrement dit, on crée un "*Home*". La variable ***props*** est un paramètre du constructor de *React.Component* appelé avec la fonction ***super***.

Copier la fonction *render* du fichier *app.js* et mettez la dans votre classe Home.
Faites de même pour le style, penser aux imports…
Voici ce que vous devriez avoir :

```
// App.js

import React from "react"
import { View, Text, StyleSheet } from 'react-native'

export default class Home extends React.Component {
   constructor(props) {
       super(props);
   }

   render() {
       return (
           <View style={styles.container}>
               <Text>Open up App.js to start working on your app!</Text>
           </View>
       )
   }
}

const styles = StyleSheet.create({
   container: {
       flex: 1,
       alignItems: 'center',
       justifyContent: 'center',
   }
});
```

La fonction ***render*** permet d'envoyer la vue à l'application et de l'afficher. Le contenu qui est dans le *return* de la fonction *render* est appelé "**Langage de balisage**" utilisé pour React (**JSX**). Cette syntaxe se rapproche plus communément du web (*HTML*).

Nous avons aussi créé une variable constante appelée *styles,* qui va contenir l'ensemble de nos feuilles de style. On peut remarquer que c'est similaire au ***css*** mais ça reste du javascript malgré tout. Cette variable est ensuite utilisée dans la ***propriété*** '*style'* de la balise *View* :

```
<View style={styles.container}></View>
```

Maintenant qu'on a créé notre component et qu'on a isolé notre élément graphique, on peut dès à présent appeler notre component Home depuis App.js

```
import React from 'react';
import Home from './components/home';

export default class App extends React.Component {
 render() {
   return (
     <Home/>
   );
 }
};
```

Recharger l'application. Visuellement rien n'a changé, mais maintenant la vue est divisée en component. On peut avoir autant de component que l'on souhaite dans une seule vue.

Personnaliser sa vue

Nous allons maintenant personnaliser notre vue en modifiant le texte ainsi que son style. Ajouter deux nouveaux styles pour le titre :

```
const styles = StyleSheet.create({
   container: {
       flex: 1, // S'adapte à l'écran de l'utilisateur
       alignItems: 'center', // Centre le contenu
       justifyContent: 'center' // Position des éléments par rapport au flex
   },
   text: {
       fontSize: 24, // Taille de la police
       alignItems: 'center' // Centrer
   },
   title: {
       marginTop: 25 // Positionnement vertical
   }
}
```

On ne va pas rentrer dans les détails des propriétés css de cet élément dans ce workshop, mais cela permet de centrer à la fois le contenu de notre page ainsi que centrer les textes et de positionner le titre.

Remplacer la ligne qui contient la balise *Text* avec ceci :

```
<Text style={[styles.text, styles.title]}>Génération de mot de passe</Text>
```

Nous allons à présent créer un bouton "*Générer*" qui aura pour effet de générer un mot de passe. Pour cela, nous allons utiliser un nouvel élement, le ***TouchableOpacity***, c'est l'équivalent d'un bouton avec une certaine animation lors du clique.

Nous allons dans un premier temps l'importer :

```
import { View, Text, StyleSheet, TouchableOpacity } from 'react-native'
```

Et mettre une balise <*TouchableOpacity*> avec un texte à l'intérieur du bouton :

```
<TouchableOpacity onPress={this.getPassword}>
  <Text style={styles.button}>GÉNÉRER</Text>
</TouchableOpacity>
```

On peut voir la propriété ***onPress*** qui a pour valeur "*{this.getPassword}*". L'action du clique sur le bouton, aussi appelé ***événement***, exécute automatiquement la fonction *getPassword* de la classe Home. Nous créerons cette fonction plus tard.

Voici le style de cet élément :

```
const styles = StyleSheet.create({
   container: {
       flex: 1,
       alignItems: 'center',
       justifyContent: 'center'
   },
   text: {
       fontSize: 24,
       alignItems: 'center'
   },
   title: {
       marginTop: 25
   },
   button: {
       backgroundColor: '#147efb', // Couleur de fond (bleu)
       borderWidth: 1, // Épaisseur du contour
       borderColor: 'white', // Couleur du contour
       borderRadius: 12, // Puissance de l'arrondi sur les bords
       color: 'white', // Couleur du texte
       fontSize: 22, // Taille de police
       fontWeight: 'bold', // Epaisseur du texte (gras)
       marginTop: 45, // Positionnement vertical
       padding: 12, // Positionnement à l'intérieur du bouton
       overflow: 'hidden', // Cacher le superflux du bouton
       textAlign: 'center' // Texte centré
   }
});
```

Générer le mot de passe

Il nous manque un dernier élément graphique à ajouter : le champ de texte pour le mot de passe généré.

Je vous invite à créer un champ de texte en dessous de la *TouchableOpacity* qui prendra 2 styles différents - https://facebook.github.io/react-native/docs/components-and-apis

```
const styles = StyleSheet.create({
   // Style existant
   text: {
       fontSize: 24,
       alignItems: 'center'
   },
   // Style à créer
   password: {
       marginTop: 15,
       alignSelf: 'center'
   },
   ...
});
```

Il manque seulement le contenu de cet élément *Text*, qui sera le mot de passe généré.

Pour générer le mot de passe, nous allons faire appel à une API* (cf fin de sujet). Cette partie est un peu plus complexe, on ne va donc pas s'attarder dessus. Voici la fonction dans la classe Home qui vous renverra un mot de passe :

```
getPassword = async () => {
  let url = "https://makemeapassword.ligos.net/api/v1/alphanumeric/json";
  fetch(url).then(response => response.json()).then((data) => {
    if (data) {
      console.log(data.pwd[0]);
      this.setState({
        password: data.pws[0]
      });
    } else {
      this.setState({
        password: "API surchargé, veuillez réessayer plus tard"
      });
    }
  });
}
```

Le *console.log(data.pwd[0])* permet d'afficher le mot de passe dans la **console**.

Afficher le mot de passe

Nous avons vu dans la partie précédente comment récupérer un mot de passe. Mais comment allons nous l'afficher au moment où on le reçoit.
Autrement dit, comment allons nous dire à l'application de recharger le component pour remplacer le texte vide en dessous du bouton par le mot de passe ?

Pour cela, nous allons utiliser ce qu'on appelle un ***state***. Nous n'allons pas rentrer dans les détails, mais pour faire simple, c'est une variable accessible comme n'importe quelle autre variable de la classe, mise à part qu'elle est **unique** et **propre à l'élément parent**. Rappelez-vous, l'élément parent est le *React.Component* !

Cette variable qui est en réalité un objet (on peut mettre autant de valeurs qu'on veut à l'intérieur) est spéciale puisqu'à chaque **changement** de cette variable, la vue de votre **component** se **rafraîchit/recharge**.

Dans un premier temps nous allons initialiser la variable state avec un mot de passe vide dans le constructeur de la classe *Home*

```
// Home.js

constructor(props) {
  super(props);

  this.state = {
    password: "" // Password vide
  }

}
```

Le ***this*** fait référence à la classe actuelle.
Remarquer que si vous n'appelez pas le constructeur du parent (la fonction *super*), une erreur sera levée en vous disant que state n'est pas reconnu.

Si vous regardez dans la fonction ***getPassword****,* lorsqu'on reçoit le mot de passe, on fait appelle à la fonction ***setState*** qui permet de récupérer les modifications de vos données et indique à React que le component a besoin d'être rechargé avec ces nouvelles données.

Il nous reste plus qu'à afficher la valeur de ***password*** qui se trouve dans l'objet ***state*** dans la balise ***Text***.

```
<Text style={[styles.text,styles.password]}>{this.state.password}</Text>
```

Pour afficher la valeur d'une variable au sein de notre vue, on va utiliser un affichage dynamique. Pour cela on utilise les "**{ }**" et mettre notre variable qu'on souhaite afficher au milieu "**{this.state.password}**".

Au lancement de l'application le mot de passe s'affiche bel et bien mais vu qu'on l'initialise dans le constructeur et qu'il est **vide**, il n'affiche rien. Au moment où l'on clique sur le bouton "Générer", on fait appelle à l'**API** pour récupérer un mot de passe et le mettre dans notre ***password*** de l'objet ***state***. Le component se recharge automatiquement grâce à la fonction ***setState***, le mot de passe étant maintenant rempli, il s'affiche en dessous.

Voici ce que vous devriez avoir dans le fichier Home.js

```
import React from "react"
import { View, Text, StyleSheet, TouchableOpacity } from 'react-native'

export default class Home extends React.Component {
   constructor(props) {
       super(props); // Constructeur du parent

       // Initialisation de state avec password vide
       this.state = {
           password: ""
       }
   }

   // Fonction asynchrone pour récupérer un mot de passe via une API
   getPassword = async () => {
       let url = "https://makemeapassword.ligos.net/api/v1/alphanumeric/json";
       fetch(url).then(response => response.json()).then((data) => {
           if (data) {
               this.setState({
                   password: data.pws[0]
               });
           } else {
               this.setState({
                   password: "API surchargé, veuillez réessayer plus tard"
               });
           }
       });
   }

   render() {
       return (
           <View style={styles.container}>
               <Text style={[styles.text, styles.title]}>Génération de mot de
passe</Text>
```

```
                <TouchableOpacity onPress={this.getPassword}>
                    <Text style={styles.button}>GENERER</Text>
                </TouchableOpacity>

                <Text style={[styles.text, styles.password]}>{this.state.password}</Text>
            </View>
        )
    }
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
    },
    text: {
        fontSize: 24,
        alignItems: 'center'
    },
    title: {
        marginTop: 25,
    },
    password: {
        marginTop: 15,
        alignSelf: 'center'
    },
    button: {
        backgroundColor: '#147efb',
        borderWidth: 1,
        borderRadius: 12,
        borderColor: 'white',
        color: 'white',
        fontSize: 22,
        fontWeight: 'bold',
        marginTop: 45,
        padding: 12,
        overflow: 'hidden',
        textAlign: 'center',
    }
});
```

Pour aller plus loin…

Pour améliorer l'application vous pouvez par exemple :

- Afficher une nouvelle vue au moment de la génération du mot de passe
- Concevoir une navigation entre vos vues
- Implémenter des paramètres pour le mot de passe (hexa, alphanumeric, pin)
- Mettre un "load spinner" pour éviter d'appuyer plusieurs fois sur le bouton "Générer" et indiquer à l'utilisateur que le mot de passe est en train de se générer
- Faire vos propres styles

Voici un exemple du projet amélioré
https://github.com/Saloukai/genPass

* API

Une API (interface de programmation) est comme une prise de courant. On s’y connecte pour en récupérer des données.