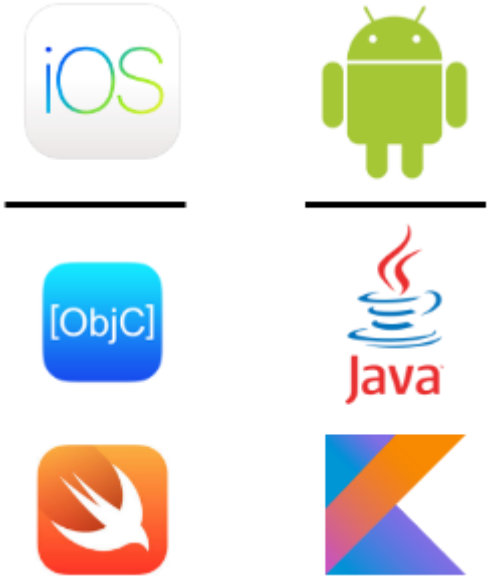
Introduction

1. Applications natives vs applications hybride cross-platforms

a. Applications Native

Les applications natives sont des applications spécifiquement développées pour un système d'exploitation. Les systèmes d'exploitation les plus connus pour le mobile sont iOS et Android. Si vous souhaitez développer une application native et compatible avec iOS et Android, il faudra développer deux applications complètement différentes : une première pour iOS, en langage Swift ou Objective-C, et une seconde pour Android, en langage Kotlin ou Java.



b. Applications hybride cross-platforms

Les applications cross-platforms, à l'inverse des applications natives, sont développées une seule et unique fois et sont compatibles sur iOS et Android. Le développement d'applications cross-platforms passe par des frameworks. Parmi les plus connus, on retrouve Ionic, PhoneGap, Xamarin et Titanium.

De gauche à droite : Ionic, PhoneGap, Xamarin et Titanium

Plus besoin d'apprendre un langage de programmation par plateforme, on développe notre application une fois, dans un langage, et le framework se charge de vous créer une application compatible iOS et Android. On passe de deux applications créées (une iOS, une Android) à une application d'où le gain de temps est énorme.

En réalité, non. Les applications cross-platforms sont réputées moins performantes et moins fluides que les applications natives. Souvent, les développeurs sont déçus ; cela a été mon cas pendant des années. Sur le papier, on vous promet une application compatible pour les deux plateformes iOS et Android. En réalité, il y a souvent des ajustements à faire pour chaque plateforme, ce qui rend votre projet de moins en moins clair et surtout de moins en moins unique (argument n° 1 des applications cross-platforms).

On reproche également aux applications cross-platforms d'avoir un rendu visuel plus proche du web que du mobile.

c. React Native

React Native se positionne dans la création d'applications cross-platforms, donc vous n'aurez qu'une application à développer et un seul langage à apprendre, du Javascript

Alors que certains frameworks cross-platforms se contentent d'utiliser des composants web (je pense notamment à Ionic), React Native utilise les composants mobiles natifs. S'il y a bien une information à retenir sur React Native, c'est celle-là.

Cela signifie que, lorsque vous définissez par exemple une vue en React Native, sur iOS votre application affiche une `UIView` et sur Android une `android.view.View`, deux composants natifs. Le fonctionnement est le même pour tous les types d'éléments graphiques : boutons, textes, listes, chargement, etc. ***React Native convertit tous vos éléments en leur équivalent natif***. C'est grâce à cette fonctionnalité que vos applications seront plus performantes, plus fluides et plus ressemblantes à une application mobile.

2. Views Android

a. Android Activity et layout :

Une activité, ou Activity, en anglais, est une brique fondamentale d'Android. C'est le point d'entrée de n'importe quelle application Android.

Une activité a pour rôle principal d'interagir avec l'utilisateur. C'est une classe Java ou Kotlin, qui hérite obligatoirement de la classe Android Activity ou AppCompatActivity.

Pour interagir avec l'utilisateur, il faut lui présenter des éléments graphiques et des éléments de contrôle ou widgets pour qu'il puisse s'amuser avec ses petits doigts. Ces widgets peuvent être des boutons, des zones de saisie ou des menus déroulants, par exemple.

Afin de déterminer quels éléments graphiques utiliser et comment les positionner à l'écran, nous utilisons un fichier layout. Un fichier layout est un fichier XML que l'activité va charger après avoir été instanciée. Ce fichier XML est toujours stocké dans le répertoire res/layout de votre projet.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</LinearLayout>

```

b. TextView

```xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:padding="8dp"
    android:text="Welcome! What's your name?" />
```

c. EditText

```xml
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:hint="Please type your name" />
```

d. Button

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_margin="16dp"
    android:text="Let's play" />
```

3. Components React

Le component est LE concept clé de React. React Native fournit un certain nombre de composants de base intégrés prêts à être utilisés dans votre application. Un component est un élément graphique affiché par l'interface utilisateur. Toutes les vues React sont composées entièrement de components.

- Composants de base (vue, texte, image, saisie de texte, scrollview, feuille de style)
- Interface utilisateur (bouton, changer)
- Vues de liste (flatlist, liste des sections)
- Spécifique à Android (backhandler, tiroirdispositionAndroid, autorisationandroid, toastandroid)
- Spécifique à iOS (feuille d'actionIOS)
- Autres (indicateur d'activité, alerte, animé, dimensions, clavierEviterVue, Mise en relation, Modal, PixelRation, RefreshControl, Barre d'état)

Les éléments graphiques, vues, boutons, textes, images, listes, etc., sont des components. Si vous créez une vue Home avec une image et un texte, votre vue Home est un component et elle possède un component image et un component texte.

a. Créez un component custom

Lorsque l'on crée des components custom en React Native, on a pour habitude de les placer dans un dossier Components de notre application et surtout de créer un fichier Javascript par component.

Pour créer un component custom, on a besoin d'importer la librairie React. Rappelez-vous le React.Component de notre fichier App.js.

```
1  // Components/Search.js
2  import React from 'react'
3
4  class Search extends React.Component {
5
6  }
```

Les components affichent des éléments graphiques à l'écran : des vues, des textes, des images, un profil, etc. En React, on dit qu'un component rend un élément graphique.

La classe React.Component implémente une méthode render. C'est cette méthode qui va définir ce que notre component va rendre à l'écran. Lorsque l'on crée un component custom, on doit obligatoirement réimplémenter cette méthode et retourner (return) les éléments graphiques, sans quoi notre component ne rendra rien et ne fonctionnera pas.

```
1  // Components/Search.js
2  import React from 'react'
3
4  class Search extends React.Component {
5      render() {
6          return (
7              // Ici on rend à l'écran les éléments graphiques de notre component custom Search
8          )
9      }
10 }
```

Avant de pouvoir utiliser les components React Native, il faut les importer de la librairie du même nom :

```
// Components/Search.js

import React from 'react'
import { View, TextInput, Button } from 'react-native'
```

Pour exemple, je veux que notre component Search affiche une **vue** avec un **champ de saisie de texte** et un **bouton**. Nous allons identifier les components React Native correspondants.

```
// Components/Search.js

import React from 'react'
import { View, TextInput, Button } from 'react-native'

class Search extends React.Component {
  render() {
    return (
      <View>
        <TextInput placeholder='Titre du film'/>
        <Button title='Rechercher' onPress={() => {}}/>
      </View>
    )
  }
}
```

b. App.js

Le point d'entrée de notre application (le main) est le fichier app.js dont le quel on integre notre premier component.

```
// App.js

import React from 'react'
import Search from './Components/Search'

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