

ORSYS
formation

Construire des apps native
pour les périphériques mobiles



React-native



Sommaire générale

- **1) Introduction** - Le développement natif, l'hétérogénéité et la fragmentation. - Les principaux outils cross-platform. - Positionnement de React Native par rapport aux solutions existantes.
- **2) React Native** - Rappels sur ES6/S201x. Notion de transpileur. - React et le superset de JavaScript JSX. - React Native : principes clés, fonctionnement général. - Installation et configuration de React Native. - Outils de développement et de debug. Travaux pratiques Installation et configuration de React Native. Maîtriser le debugger.
- **3) Architecture d'application** - Configurer un composant : state & props. - React Native et MVC. - Le pattern Flux, une alternative au MVC. - L'arrivée de Redux : le store, le reducer, les actions. Travaux pratiques Initialisation d'un projet React Native. Configuration d'un composant.
- **4) Construire son interface** - Les composants de base (View, Text et Image) et leurs cycles de vie. - Les événements Touch, la ListView et la ScrollView. - Organiser le layout de l'application. Mise en page avec Flexbox. - Styler les composants. - Les différentes solutions de navigation entre les pages. - Composants avancés. - Utiliser des composants de la communauté. - Ajouter des animations et des transitions. Travaux pratiques Développer une première application.



Sommaire générale

- **5) Les formulaires et la gestion des données** - Les principaux composants de formulaires. - La validation de la saisie et la gestion des erreurs. - Redux-form et alternatives. * Récupération des données : xmlhttprequest et fetch. - Le stockage local. - La gestion offline. Travaux pratiques Développer un formulaire, valider les données et assurer la persistance des données.
- **6) Interagir avec le terminal** - Les principales API natives de React Native. - Utiliser les plugins Cordova/PhoneGap. - Développer un module natif. Travaux pratiques Implémentation des principales API natives.
- **7) Usages avancés** - Best Practices et erreurs fréquentes. - Tests unitaires et fonctionnels. - Publier l'application. - Mises à jour Over The Air. - Frameworks et outils complémentaires. Travaux pratiques Tester et déployer des applications React Native.



présentation logistique

ORSYS & VOUS



Bienvenue chez Orsys

Présentation



- Indépendant
 - Prestataire d'Orsys

- Animateur
 - Monsieur DESORBAIX Alexandre

- Développeur
 - software, client lourd ,serveur
 - » c, c++, c#, java, ...
 - web, client léger
 - » html, php, sql, js, css, angular, react, ...
- blogger sous Wordpress



Bienvenue chez Orsys
logistique



- Nombre de jours :
 - 3 jours
- Horaires
 - Début : 9h
 - Fin : 17h30 approx.
- Pause
 - 10h30 approx. Matin
 - 12h30 approx. Midi
 - 15h30 approx. Après-midi



INTRODUCTION



Sommaire

❑ Dans cette partie, Introduction, nous aborderons :

- ❑ Le développement natif, l'hétérogénéité et la fragmentation.
- ❑ Les principaux outils cross-platform.
- ❑ Positionnement de React Native par rapport aux solutions existantes.



Développement natif

- Apple

- Dev initialement prévu

- sous XCODE (osx only)

- En objectif-C

- issue de small talk et C

- En C/C++

- Puis swift



Environ 20% du marché des mobiles

- Abonnement obligatoire pour

- Initialement pour tester les app sur des périphériques physiques

- Requis pour poster une app sur le store apple

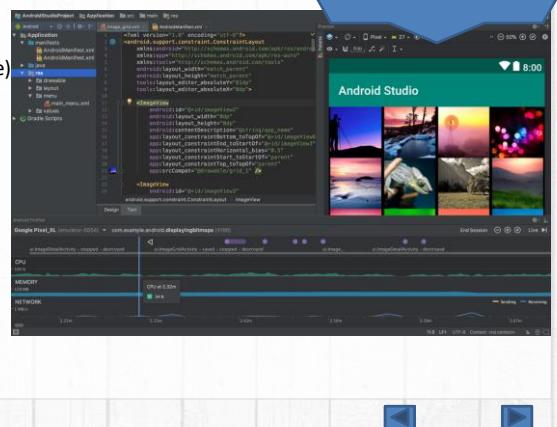
- Contrôle de politique de l'application



Développement natif

- Android
 - Dev Initialement prévu
 - Sur de multiples outils avec les outils Android Tools
 - Eclipse, NetBeans, puis dernièrement Android studio
 - En java
 - Puis **kotlin** pour des raisons de droits(Sun → oracle)
 - Cf session Sun → oracle
 - En java native (interaction java C/C++)
 - En C/C++
- Aucun abonnement
 - pour tester sur periph. Physique
 - poster une appli sur les stores Android
 - Google Play store, Samsung store, Huawei store,...

Environ 80% du marché des mobiles



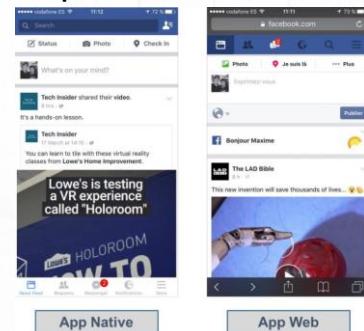
Développement natif

- Le développement natif est par définition un développement limité à une plateforme spécifiquement ciblée.
 - Le natif permet des performances accrues pour les applications
 - Permet l'usage de toutes les fonctionnalités matérielles
- En Natif pur dev Android ≠ dev iOS
- Des plateformes permettent de composer du code commun pour ces plateformes et compiler (ou assembler)
 - des apps 100% natives
 - Des apps hybrides
 - Des apps web



Développement web

- La solution la plus simple et moins couteuse pour le dev multiplateforme
 - S'appuie sur le html 5 / css / javascript
 - Limitation de la compatibilité matériels des périphériques (GPS, micro, sensors, ...)
 - Performances limitées du au javascript interprété.
 - Possibilité d'encapsuler une appli web dans un « icone installable » sur les périphériques physique
 - Facultatif, consultation web possible par navigateur



Développement hybride

- Entre le dev natif et le dev web
 - Développement facilité et plus rapide que du natif
 - Souvent cross plateforme
 - Du code commun
 - Et du code natif
 - Moins performant et stable que le dev natif
 - Plus performant que le dev web
 - Meilleur accès aux périphériques
 - Limitation du cout
 - Développement et maintient d'une seule app pour toutes les plateformes



Tour d'horizon des Framework hybrides/natif

- Les Grands Framework

- Ionic

- Code html5/javascript ou typescript
 - Plugins phonegap/Cordova pour les accès natifs au périphérique
 - ios & Android
 - Développement hybride



- Xamarin

- Code C#
 - Ecriture dans Visual studio
 - Portage de toutes les fonctionnalités natives en C#
 - Impossible d'utilisé des lib native ios / Android



Tour d'horizon

- Flutter (open sources)
 - Code *google dart*
 - Code partagé dart + code natif (C/java)
 - Possibilité de s'appuyer sur des lib natives
 - Contrôles fournis + nombreuses librairies tierces
 - Développement sous vs code, Android studio, ...



- React Native (open sources)
 - Développer et maintenu par Facebook
 - Code javascript / typescript basé sur les concepts React web
 - Contrôles natifs
 - App native après compilation
 - Permet l'écriture de certains composant en Swift, Objective-C ou Java
 - Vue native s'appuie sur react native



Les performances et le partage de code

- Ionic
 - Compatible Android 4.4+, ios 8+, Windows 10
 - Partage jusqu'à 98% du code**
- Xamarin
 - Compatible Android 4.0.3+, ios 8+
 - Partage jusqu'à 96% du code
- Flutter
 - Compatible Android, ios
 - Partage entre 50 et 90% du code
- React native
 - Compatible Android 4.1+, ios 8+, Windows 10, osx
 - Partage jusqu'à 90% du code



❑1) Introduction

- ✓ Le développement natif, l'hétérogénéité et la fragmentation.
- ✓ Les principaux outils cross-platform.
- ✓ Positionnement de React Native par rapport aux solutions existantes.



RAPPELS JS & TS



La gestion de imports

- Avec import et **export / default / import {} from**
- Syntaxe :
 - pour être importer, un contenu doit être exporter

```
export class ExportedThing{};  
export default class DefaultExported{...;}
```

- Puis importer a la demande

```
import {ExportedThing} from './rep/fichier';  
import DefaultExported from './rep/fichier';
```



Renommer un import avec as

Import { exportedVar as myNewVarName } from './fichier'

Il existe d'autres patterns pour la construction de module
C'est autres moteur sont pris en charge par web pack

ex: pattern amd et la fonction **require('...');**

Import & export Component

- Importer des composants
 - Js
 - Import DESCRIPTEUR from './folder/file'
 - CSS ,...
 - Import './fichier.css'
- Export par default
 - Obligatoire pour react
 - Export default MonComposant



Bonnes pratiques

- Les script chargées en derniers
 - <script src=« ... »></script></body>
- Mode strict
 - Permet d'exprimer des erreurs parfois sourde ou juste des mauvaises pratiques
- L'usage de ' ... ' pour les chaînes de caractères dans le js
 - Var maChaine ='Demat breizh';
- Tous les blocs possèdent des { ... }
 - If(...){ ... }
- On limite l'usage des *eval*()
- On privilégie l'opérateur === ou !==
 - a === b
- On met les constantes en premier pour les comparaisons
 - if(undefined === maValue)



Es7 2016 / Es6 2015

- ES 5 est succéder par différentes versions
 - Apportant de nouvelles fonctionnalités
 - Es6 (ou es2015)
 - puis es7
 - ,...
- Les navigateurs supportent pas forcément les versions supérieures
 - Babylon.js convertit le code js dans différentes versions cibles



Nouveautés syntaxiques es6 :

- L'usage du mot clef class pour la définition d'objets
 - Mot clef private, public protected
- Les Arrow function
 - Fonction préservant le contexte (this) de déclaration lors de l'exécution
 - » En remplacement du .bind()
 - (arg)=>{ } eq. (function(arg){...}).bind(this)
- Les promise
 - Un promesse d'exécution grâce à then(()=>{})
- La nouvelle portée de scope : le module



Nouveautés syntaxiques es6 :

- Les template strings

- Construction de chaine finale

- `${chaines[0]}${varI}.`

- \${ expression } pour délimiter les évaluations de variables
 - Les évaluations peuvent s'imbriquer

- La chaine est définie avec accent grave et non des apostrophes

- ` une littérale `
 - ' ~~pas un littérale~~ '
 - " ~~pas encore une littérale~~ "

- Peut effectuer des choix, ternaires, ...



Doc mdn des templates string : appelé des « *littérales et gabarits* » :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Litt%C3%A9raux_gabarits

Syntaxe es6

- Speard operator

- Permet de remplir un `array` ou `Iterable<T>` à avec les contenu d'un tableau b facilement

```
let tab1=[elem2,eleme3];
let tab2=[elem1, ...tab1]; //tab2 ➔ [elem1,elem2,elem3]
```

- On pourrait imaginer la même chose pour des objets, ...

- TypeScript l'a fait



C'est l'histoire d'un mec ...

- Jean-Pierre se dit que sa chance vas tourner
 - Ce jour d'avril Il y avais une course de chevaux qui allais démarrer
 - Il a donc fait un pari sur un cheval
 - Il attend devant la course qui débute
 - Son cheval est placé
 - Il remporte sa mise



Ici une notion d'asynchronicité aurais pu être mis en place

Jp aurais pu attendre boire un petit café tranquillement ou tout simplement revenir demain

Peut être que tout sa vie reposais sur ce parie

C'est l'histoire d'un mec ...

- Jean-Pierre lorsque il boit son café au bar joue souvent aux jeux d'argent.

- Plutôt chanceux, ce jour d'avril il se dit que c'est aujourd'hui

- Il gratte, il perd ..

- Il fait un loto

- Une course de chevaux, allais démarrer

- Il a donc fait un pari sur un cheval
 - Qui à un look de gagnant (cf. photo 1).
 - Il attend devant la course qui débute
 - Son cheval est placé
 - Mais il a coché sans le vouloir le » 8 petit ange des prés (cf. photo 2) → dernier
 - Enfin quand il arrivera, si il arrive

photo 1



photo 2



Ici une notion d' asynchronicité aurais pu être mis en place

Jp aurais pu attendre boire un petit café tranquillement ou tout simplement revenir demain

Peut être que tout sa vie reposais sur ce parie, il a donc tout perdu

Tout le reste de sa vie dépendais de ce paris il a donc attendu car il ne pouvais rien faire d'autre tant qu'il ne savais pas le déroulement de la suite des choses

C'est l'histoire d'un mec

- Jp sais qu'il a tout joué au tiercé du coup il rentre pu un sous en poche
- A 22h ce jour d'avril, il entend le tirage du loto

- Souvenez vous, jp à joué avant le tierce :
 - Il avais fais sa grille

3 – 8 – 21 – 35 – 49

- Enregister son ticket contre un reçu de jeu
 - Il a donc prévu après le tirage de vérifier si il a gagner
 - » Il à déjà prévu que si il gagnait, il irais encaisser ses gains
 - et que c'est un gros lot, il déménagerais à Mulhouse
 - » Il a aussi prévu que si il gagnait rien, il jetterais son ticket
- Vue qu'il étais d'humeur gagnante il a jouer et fait d'autre chose
 - C'est la notification de l'appli qui lui à rappeler que le tirage venait d'être fait

- Jp 42 ans en ce jour d'avril avais tous les bon numéro, dans l'ordre du loto.

Chanceux le type



technique

- Le jeux a gratter n'avais aucune notion d'async
- La course de chevaux na pas été utiliser sous forme d'async
- Notion de promise es6 et **await** function es7/ts
- Le loto quand à lui :
 - Jouer une fonction **async** qui **await** le tirage du soir
 - Une fois que l'action de jouer est fini j' exécute une tache
 - En sachant à l'avance quelle prendrais du temps et que j'aurais bien d'autres taches à effectuer d'ici le résultat de la fonction appeler.
 - Je prévois directement, au moment de l'appel de la fonction, le **retour d'une promise**.
 - Promesse à laquelle je peux définir quoi faire quand l'exécution sera effectuer
 - avec **.then(retour=>{ ... })**;



Async / await / Promise

```
• Fichier.js
async function doSomething2() {
  console.log(await doSomethingAsync())
}

function doSomethingAsync() {
  return new Promise((resolve) => {
    setTimeout(() => resolve('I did'), 3000);
  });
}

async function doSomething() {
  console.log(await doSomethingAsync())
  console.log(await doSomething2())
}

console.log('Before');
doSomething();
console.log('After');
```

- Pour être attendu un fonction :

- Doit soit être déclarer **async**

- Comporter un(des **await**)

- Doit retourner une promise

- Une fonction **async** s'utilise soit

- » Notion d'attente explicite, préfixer par **await**

- » Pas d'attente explicite, pas de préfixe à l'appel



Les décorateurs

- Un wrapper pour les *High Order*
 - Le décorateur est abstrait, il n'est que syntaxe pour appeler une fonction
 - Créer un objet a de type A par une fonction B vivant dans la peau de B
- Disponibles de façon expérimentale depuis ES7
 - Intégré à type script
- Permet la création de décorateurs personnalisés
 - Définition d'une fonction a utiliser pour le décorateur



Fonction encore expérimentale

Les wrapper High order peuvent être définis par une simple fonction.

Le décorateur apporte surtout de la lisibilité et de la maintenabilité du code en le rendant plus simple à lire et à écrire.

Il a pour seul rôle de substituer un `@quelqueChose()` à une fonction pour rendre plus abstrait et donc plus simple certains fonctionnements

Type script

- Du javascript en mieux
 - Un langage écrit par Microsoft
 - Sortie en version 0.8 en Octobre 2012
 - Dernière version 3.8.3 du 28 février 2020
- Proche du **js** et du **c#**, **java**
 - Cocréer par le principal inventeur du c# & concepteur du .NET
 - Interface, module, class, héritage, ...
- Un vrai langage Objets avec typage fort, intégration forte dans l'ide et dans node.js
- Un outils de trans-compilation
 - Du **ts** qui vers du **js** grâce à une commande « **tsc** »
- Un outil pour le **ts**
 - Un Linter « **tsLint** » permet le **marquage d'erreurs** et la **standardisation de l'écriture** du code



Type script

- Du javascript en mieux
 - Un langage écrit par Microsoft
 - Sortie en version 0.8 en Octobre 2012
 - Dernière version 3.8.3 du 28 février 2020
- Proche du **js** et du **c#**, **java**
 - Cocréer par le principal inventeur du c# & concepteur du .NET
 - Interface, module, class, héritage, ...
- Un vrai langage Objets avec typage fort, intégration forte dans l'ide et dans node.js
- Un outils de trans-compilation
 - Du **ts** qui vers du **js** grâce à une commande « **tsc** »
- Un outil pour le **ts**
 - Un Linter « **tsLint** » permet le **marquage d'erreurs** et la **standardisation de l'écriture** du code



Typage

- L'usage de let et const :
 - let : Son positionnement en mémoire peut changer d'objet mais rester du même type toute sa durée de vie.
 - const : le contenu référencé sera toujours le même, il ne pourra pas être réaffecter à un autre objet.
 - L'usage de const est bonne pratique dans maximum de cas, le linter peut vous le rappeler ...

- La portée de la variable est uniquement le bloc dans lequel elle est déclaré.

- (a: A) => { let b = new A(); console.log(`a:\${a} & b:\${b} existe que dans cette fonction`); }
 - If(...) { const a = new A(); console.log(`a:\${a} existe que dans cette fonction`); }

- Typage basiques :

```
let varName: Type;  
let nombre: number;  
    let chaine: string;
```

- Le Duck Typing

```
let departement = 56; //attention  
let fullName = `Sarah Vigote`;
```

- Rendre une variable facultative

```
let departement?: string; //attention aux accès
```



Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

Typage arrays et autre generiques

- **Les génériques**

- Fonction nécessitant un type de quelle doit manager
- Exemple d'un objet Array qui limite le type de contenu a des string
 - » Les fonctions de l'array vont definir de facon generique que si que des string sont stocker il peut renvoyer que des string

» <T> pour dire un type generique pas connus lors de l'ecriture d'une fonction

- et dependra du type fournit lors de l'usage a chaque appel d'usage

```
let names: Array<string>;
```

- Le typage des *array* possède plusieurs syntaxes:

- Il possède aussi les syntaxes suivante en plus de la syntaxe générifique

```
let names string[];
```

```
let names: [string];
```



Typage

- L'usage de let et const
- Typage basiques de variable:

```
let varName: Type;  
let nombre: number;  
let chaîne: string;
```

nom est un paramètre typés obligatoire

```
function hello(  
    nom: string,  
    age: number = 25,  
    isMale?: boolean  
): string  
{return `Demat ${nom} ${age}`;}
```

age est un paramètre typés obligatoire
Mais avec une valeur par défaut

isMale est un paramètre facultatif
sans valeur par défaut

La fonction doit renvoyer une string

Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

Class

- Les classes
 - permettent le définition d'un objet
 - Des champs ou propriétés
 - Des méthodes du cycle de vie de l'objet
 - **constructor**,...
 - Des méthodes et des fonctions
 - Il peut avoir des champs qui lui appartient qu'à lui
 - Notion de **private**, **protected**
 - Il peut avoir des champs/fonctions disponibles pour interagir avec lui
 - Notion de **public**



Génèrera une fonction pour l'es5 représentant l'objet transformé **sans class**

C'est l'histoire d'un mec...

- C'est l'histoire d'un mec, de 56 ans habitant à mulhouse prénommé jean-pierre



- Il acheté une machine à café, le type ...
 - Elle à un bouton marche / arrêt
 - Un bac d'eau à remplir d'un litre
 - Et du café en poudre à mettre dans le filtre à café

- Analysons la conception
de l'objet



C'est l'histoire d'un mec...

- Sans intérêt pour nous

C'est l'histoire d'un mec, de 56 ans habitant à Mulhouse prénomme jean-pierre



- Identifions une class machine à café

- Des fonctions d' interaction **public**
 - Elle à une fonction marche / arrêt
 - Un champs définissant le volume d'eau du bac
 - Elle possède un champ de type café
- Des fonctions et champs qui lui sont propre
 - Température de l'eau
 - Activation de la résistance chauffante
 - Activation de la pompe
 - Eclairage du bouton lors du branchement

- Une cafetière c'est donc un objet!!!

Et jean-pierre aussi
Heu l'autre!!



Class machine à café

- Machine à café

- Des champs privés

- Des champs public

- Un constructeur

- Des fonctions interne privée

- Des interactions extérieur, public
 - This, context de l'objet dans l'objet

```
class MachineCafe{  
    private temperature: number;  
    private temperatureMax=95;  
    private isPumpStarted: boolean;  
    private isHeaterStarted: boolean;  
  
    public isOnButton: boolean;  
    public cafe?: Cafe;  
    public niveauEau: number;  
  
    constructor()  
    {  
        this.temperature=0;... /*  
        this.niveauEau=this.senseWaterLevel();  
    }  
  
    private senseWaterLevel(){}
    private startPump(){}
    private startHeater(){}
  
  
    public makeCafe()  
    {  
        this.startHeater();  
        this.startPump();  
    }  
};
```

C'est l'histoire d'un mec...

- Jean-pierre est pas adroit il met du café en poudre partout tous les matins



- Il acheté une machine à café,
 - **oui encore**, système **café à dosettes** cette fois
 - elle **fait tout pareil**, mais plus pratique
 - Elle à un bouton marche / arrêt
 - Un bac d'eau à remplir d'un litre
 - Et un objet **filtre** qui contient **café** est redéfini différemment
 - Il accepte désormais du **café en dosette**
- Alors une machine à dosette c'est une machine qui fais aussi du café.

Comme l'autre!!!



On parle ici de l'extension d'une class
qui reprend tout ce que fais un objet pour faire un nouvelle objet a partir du premier.

Il peut redéfinir des fonctions ou méthodes existante chez le parent et en apporter d'autres.

Ici notre cafetier posséde les memes fonctions : faire le café par exemple

Mais le fonctionnement n'est pas profilement le meme.

Le café cette fois , toujours présent mais un nouveau conteneur doit etre insérer dans la machine , ... plus le café directement

Class machine à café

- Machine à café à dosettes
 - Ses champs privés
 - Ceux du parent étant eux aussi **private** il sont **innaccessible**
 - » Notion d'héritage
 - » Notion de **protected**
 - Des champs / fonctions public, protected
 - Permettant l'héritage aux futurs extensions
 - **Redéfinition** de la manière de faire le café
 - » Notions d'**override**
 - Des champs / fonction privés qui lui sont totalement personnel
 - Un constructeur
 - Appel du constructeur parent
 - » Notion de **super()**
 - Des fonctions interne protégé pour permettre aux futurs génération de pleinement hérité de leur ainés

```
class CoffeMachinPad extends CoffeMachin{  
    public padHolder: PadHolder;  
  
    constructor()  
    {  
        super();  
    }  
  
    public insertDose(pad:CoffePad){ ... }  
  
    public makeCofe(){  
        this.coffee= this.padHolder.coffee;  
        this.startHeater(97);  
        this.startPump(255);  
    }  
};
```

C'est l'histoire d'un mec...

- Jean-Pierre est devenu un expert du café
 - Il a essayé une 20aine de systèmes différents.
- Il à finit par déduire que
 - Faire du café **c'est toujours** :
 - Mettre du café/ eau
 - Démarrer
 - Collecter boisson chaude
 - abstraction de faire le café
 - Les machines implémentent les fonctions de faire du café commun et possède des champs communs
 - Il peuvent grâce à **implements** bénéficier de l'abstraction d'autres fonctionnalités, ex : mousseur à lait



On parle d'abstraction.

Dans l'abstraction on définit (ou uniquement le besoin de de déclarations), le contenu commun a implémenter chez l'enfant

L'abstraction est plus a voir comme la définition d'un *Object* générique plutôt *sous forme de besoins conceptuel*

Contrairement à **extends**, **implements** se limite à forcer l' éxistance de fonctions ou champs pour chaque implem.

C'est l'histoire d'un mec...

- Jean-Pierre est devenu un expert du café
 - Il a essayé une 20aine de cafés différant.
- Il à finit par déduire que
 - Le café est toujours caractérisé de la même manière mais nécessite pas un besoin d'objet définit Seul le fait qu'il contienne tout me suffit
 - Il utilise donc une interface qui définit juste le contenu décrit
 - Arome,
 - Provenance,
 - ... mais **aucune fonctionnalité** a proprement parler
 - Il formalise juste qu'est ce qu'on peu appeler du café et qui le caractérise



C'est l'histoire d'un mec...

- Jean-pierre est devenu un expert du café
 - Il a essayé une 100aine de cafés diffèrent.
- Il à finit par déduire que
 - Il a créer une classification pour les différents champs plutôt qu'un simple nombre
 - Il utilise donc une **enum** qui définit juste une catégorisations
 - Arome,
 - » corsé, doux
 - Provenance,
 - » Pérou, Mexique, ...
 - ... mais **aucune fonctionnalité** à proprement parler
 - Il formalise et catégorise pour une valeur



On parle ici d'**enumeration**,

Chaque ensemble est definissable comme un type a part en tierre valant toujour un des champs disponible pour l'**enumeration**



La gestion de imports

- Avec import et **export / default / import {} from**
- Syntaxe :
 - pour être importer, un contenu doit être exporter

```
export class ExportedThing{};  
export default class DefaultExported{...;}
```

- Puis importer a la demande

```
import {ExportedThing} from './rep/fichier';  
import DefaultExported from './rep/fichier';
```



Renomer un import avec as

Import { exportedVar as myNewVarName } fromm './fichier'

Il existe d'autres paterns pour la construction de module
Cest autres moteur sont pris en charge par web pack

ex: pattern amd et la fonction **require('...');**

tsc

- La phase de transpilation (ex: tsc) vérifie l'intégrité de forme des variables
 - Un fichier tsconfig.json qui spécifie :
 - Le type de sortie
 - Les fonctionnalités à prendre en charge ,...
 - Il se génère aisément grâce à **tsc --init**
 - Attention cette phase de compilation garanti uniquement l'intégrité des types jusqu'à la transpilation
 - » Le code sera **exécuter** avec une version **js** générée



Doc tsc : <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>

Commande pour initialiser le fichier tsconfig.json:

tsc –init

Générer dans le tsconfig

```
"target": "es5",          /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', or 'ESNEXT'. */  
"module": "commonjs",    /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020', or 'ESNext'. */
```

Tscconfig.json

- Exemple de fichier tsconfig.json générée par tsc -init

```
{  
  "compilerOptions": {  
    /* Basic Options */  
    // "incremental": true, /* Enable incremental compilation */  
    "target": "es5", /* Specify ECMAScript target version: 'ES3' (default) */  
    "module": "commonjs", /* module code generation: 'none','commonjs','amd' */  
    // "lib": [], /* Specify library files to be included */  
    // "allowJs": true, /* Allow Javascript files to be included */  
    // "checkJs": true, /* Report errors in .js files. */  
    // "outFile": "./", /* Concatenate and emit output to single file. */  
    // "outDir": "./", /* Redirect output structure to the directory. */  
    // "rootDir": "./", /* Specify the root directory of input files. */  
    // "removeComments": true, /* Do not emit comments to output. */  
    // "noEmit": true, /* Do not emit outputs. */  
    // "isolatedModules": true, /* Transpile each file as a separate module */  
    /* Strict Type-Checking Options */  
    "strict": true, /* Enable all strict type-checking options. */  
    /* Experimental Options */  
    // "experimentalDecorators": true,/* Enables support for ES7 decorators. */  
    // "emitDecoratorMetadata": true, /* Enables support for emitting type of  
    // decorators. */  
  }  
}
```

Doc : https://www.typescriptlang.org/docs/handbook/tsconfig_json.html

- Une série de paramètres

- La cible visée

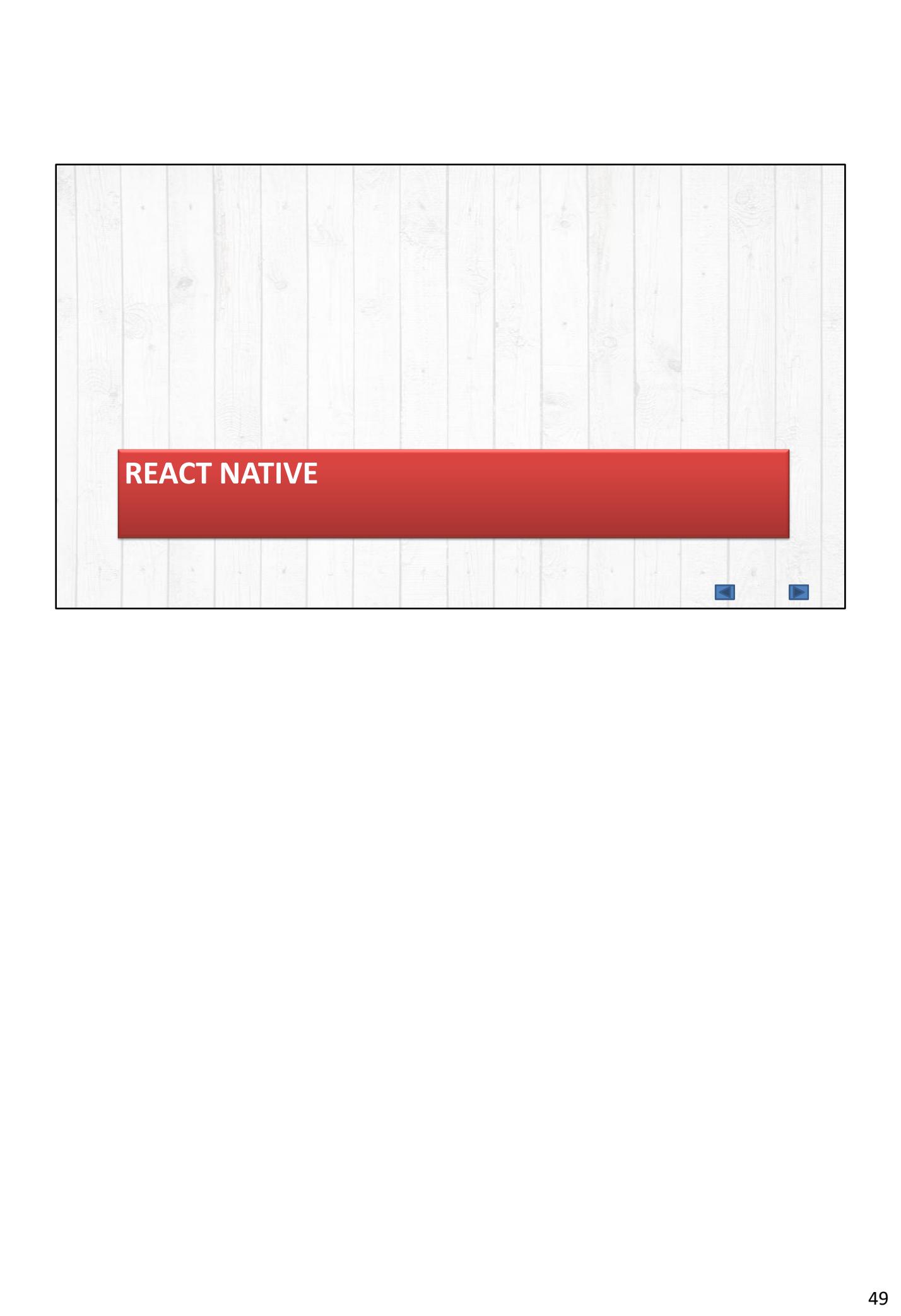
- La gestion des modules

- Les chemins entrés / sorties

- ...

- Les paramètres pour une grande partie peuvent être **set** dès la ligne de commande tsc





REACT NATIVE



❑ 2) React Native

- ❑ Rappels sur ES6/S201x. Notion de transpileur.
- ❑ React et le superset de JavaScript JSX.
- ❑ React Native : principes clés, fonctionnement général.
- ❑ Installation et configuration de React Native.
- ❑ Outils de développement et de debug.

- ❑ Travaux pratiques Installation et configuration de React Native. Maîtriser le debugger.



Editeurs avancés pour le html/css/php/mysql/...

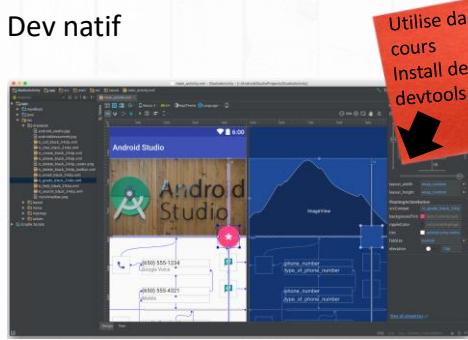
VS Code

- Gratuit, multi langues, plugins & snippets, liaison debugger
- support GIT
- Intégration react native + Android Tools

Android studio

- Gratuit, multi langues, plugins & snippets, liaison debugger,
- Pre-Install de Android DevTools, VM
- Dev natif

Utilise dans le cours
Install des devtools



Configuration : <https://riptutorial.com/fr/react-native>

Développement sur périphérique physique

- Android dev Tools (ou ADT) doit être ajouté au PATH de la machine
 - Android studio le fait pour nous* (sous Windows)
- Branchez votre périphérique (téléphone, tablette)
 - Activer les options de développement
 - » Cliquez 7 fois sur le numéro de version des paramètres
 - Activer le debug USB
 - » Un certificat sera à valider à chaque connexion de debug USB
- Redémarrer ADT en mode USB pour activer le debug USB sur l'ordinateur

Commande :

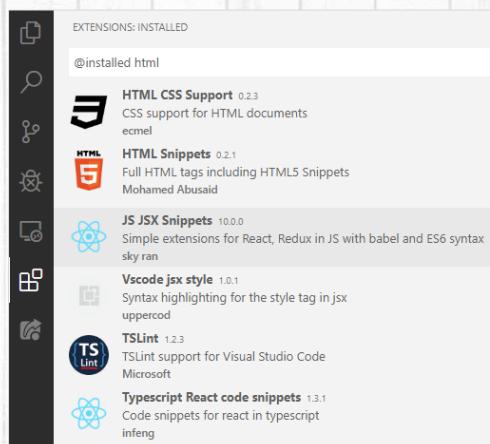
```
adt usb
```



Liste des drivers wiko : <https://rootmydevice.com/usb-drivers/download-wiko-usb-drivers/>

Vs code les plugins

- Css3 support
 - Ajout de compléction
- Html support
 - Ajout de complétion html
- Js jsx snippets
 - Snippets
 - eslint
- Typescript react code snippets
 - Snippets
 - TSLint



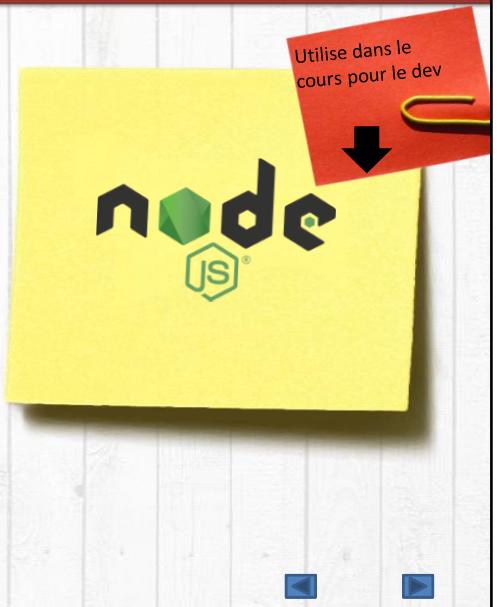
IDE pour dev IOS

- Malgré le partage de code pour toutes les plateformes
 - Pour IOS seul xcode permet d'assembler une appli apple
 - Xcode ne fonctionne que sur Mac osx et sur ipad
 - Le code peut être développer sur d'autres ide



Node.js & npm

- Node
 - Exécution de code JS côté serveur
 - Application
 - Scripting, ...
 - Lot d'applications existant
 - En ligne de commande
 - Permet de faire fonctionner les outils nécessaires au développement écrit en JS/TS, compilation et test de l'application



Configuration Node.js

- Gestionnaire de dépendances

- npm

- Gestionnaire de dépendances principale de node.js
 - Permet de récupérer les outils nécessaires aux différentes tâches



- Yarn

- Gestionnaire de dépendances secondaire pour node
 - Comme npm mais fourni par facebook
 - Souvent utilisé pour react



Yarn vs npm : <https://blog.soshace.com/yarn-package-manager-in-2019-should-we-keep-on-comparing-yarn-with-npm/>



npm

- Commandes npm

- Initialisation d'un répertoire de projet
 - Créer un fichier minimal package.json
 - » **npm init -y**
 - Préconfiguré node avec le répertoire .git (si .git présent)
 - -y : initialisation « quiet »
- Installation des paquets
 - » **npm install -g packetname packet2@version**
 - -g : installation pour l'utilisateur courant,
 - sans -g installation pour le projet, dans le répertoire node_modules
 - --save permet l'ajout du paquet aux dépendances du projet
- Npm start vs npm run monscript
 - Dans package.json, section scripts
 - Script name spécifiques nécessitant pas 'run' ex npm test, npm install, ...



Package node

- React native propose sous node une commande cli de scaffolding de projet

- La structure du projet
- La configuration de la plateforme
- L'assemblage et l'envoie à android-tools pour le portage vers un périphérique virtuelle ou physique

npx react-native init nom-du-projet

- Il résulte un projet nodejs avec un fichier package.json offrant différents scripts d'exécution ainsi que la définition des dépendances du projet
 - » Babel, ts, eslint et surtout **react** la base de react-native



La commande npx permet l'exécution d'un paquet pas encore installer sans l'installer sur la machine.

Il est possible d'utilisé :

npm install -g react-native pour l'installer
globalement sur la machine

puis

react-native init nom-du-projet pour
initialisé le projet

Npx permet donc en une seul commande d'effectuer ces 2 taches pour initialiser le projet

Serveur GIT

- Plateforme de disposition de code versionné
 - Repository
 - Gestion des commit distant
 - Travail d'équipe
- Plateformes gratuites
 - Github, GitLab
 - Hébergé ou officiel
 - 1 repo / account
 - Altasian Bitbuckets *(Sélectionné pour le cours)
 - Multi-repository
 - Free (max 1Go/repo) ou payed account
 - Officiel uniquement



Git - bitbuckets

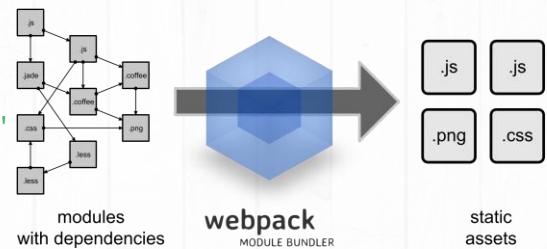
- Crédation d'un compte gratuit
 - <https://bitbucket.org/account/signup/>
 - <https://bitbucket.org/account/signin/>
- Crédation 1ere repository
 - Edit, commit, approuve du readme.md
 - Clone local de la repository
- Ouverture du répertoire sous Visual Studio Code
 - Edition du readme.md sou vscode
 - 1^{er} commit & push
- Approuve commit



Pour assurer la construction

- **Webpack**

- L'assembleur de module
 - Ex Avec common.js → require("")
 - avec `import bar from './bar'`
- Angular/cli nous fourni un environnement webpack préétabli



- **Babel.js**

- Se dit :le « compilateur » de js
 - Il compile le code et peut le rendre dans des Versions ES antérieurs
 - Transpile aussi le ts en js

BABEL

Web pack : <https://webpack.js.org/>

Babel : <https://babeljs.io/>

❑ Dans cette partie, React Native, nous avons abordé :

- ✓ Rappels sur ES6/S201x.
- ✓ Bonnes pratiques ES
- ✓ Concept async et les promises
- ✓ Notion de transpileur et typescript
- ✓ Typage TS
- ✓ Interface TS



React-native c'est avant tout du REACT

REACT



Une librairie

- React c'est une **librairie js/ts**, elle permet
 - De créer des applications clientes web
 - Créer de petit composant à assembler entre eux
 - Assembler de plus gros composants constituer de plus petit composants
 - Faire interagir ces gros composants entre eux
 - Créer une app grâce aux composants

Et c'est
tout !!!!



Une librairie

- React c'est une librairie js, qui permet aussi :
 - La mise en œuvre de tests
 - Reconfiguration de la plateforme d'automatisation des tests
 - La prise en charge de js et ts (webpack/babel)
- La librairie contient aussi une série de script serveur de construction d'app
 - Une commande **node.js** pour construire le squelette d'un app.



Les composants

- L'application est découpée en hiérarchie de composant
 - Un composant est une balise utilisée dans l'app
 - Un composant est composé
 - Un **model**
 - Des **données** la composant
 - Des actions possibles
 - Sa représentation ou **render**
 - Des **composants enfants**
 - Du style
 - Il possède des valeurs d'entrée
 - Props
 - Il **peut posséder un état** qui lui est propre
 - state



Props

- On appelle ***props*** les valeurs fournies aux composants
 - Chaque attribut présent dans une balise de composant
 - Disponible dans le composant par un objet de valeur communément appelé ***props***
 - Les ***props*** sont **en lecture seul**
 - Lorsque le parent d'un composant met à jour les ***props*** d'un enfant, le composant enfant effectuera une nouvelle fois le ***render*** du composant
 - Mettra à jour les ***props*** de ses enfants lors du ***render***



Component concepts

- Un component
 - Reçoit des **props** (en lecture seul)
 - **Commence toujours par une majuscule**
 - Pour dissocier les Composant et les html
 - Gère son cycle de vie
 - Si statefull ou hooks
 - Cf. cycle de vie
 - Il existe 2 types de component
 - Les **function**, composants simple exprimant des valeurs (**stateless** ou avec hooks)
 - Les objets **class**, composants plus complexe et plus lourd, gérant plus d'interactions (**statefull**)



State et data dynamique du component

- Le state est un conteneur pour les variables du composant
- Il est initialisé avec un état initial
 - Mickael beau jeune homme afro américain
- Il est modifier par le composant ou autre,
 - Mickael à subit des changements de son state
 - On parle de l'état de Mickael à cette instant
 - Son render a été affecté
 - Mickael à re subit un changement de son state
 - On parle du nouvel état issue de l'état initial et ayant subit déjà un changement



Jsx subset

- Js + XML dans le même fichier

Permet le retour de contenu de composants

– Blocs XML Commencent et finissent par ()

– Dans ces blocs xml,

- les commentaires s'écrivent :
– {/*comm.*/}

- Les sections en js dans un partie html de jsx
– { monarray.map(e=>{<balise/>}) }



```
function App(props) {
  return (
    <div className="App" >
      {/* parcour d'array de logos */}
      {
        props.logos.map(
          e=>{<img src={logo}>/>})
      }
    </div>
  );
}
export default App;
```



Jsx subset

- Js + XML dans le même fichier
 - Attributs xhtml disponibles par leurs noms JS
 - Ex :
 - » Id →
 - » style → style
 - » class → className
 - L'attribut **style** est un objet en js et reçoit donc un objet de styles css
 - Ex : style={{textAlign:'center', fontWeight : '900'}}}



Usage des composants

- Les composants sont utilisables sous forme de balises

Composant.jsx

```
Import MonComponent from './moncomponent/moncomponent'  
function(props){  
    return (<MonComponent valeur1={Objet/funct} valeur2="valeur"/>);  
}
```

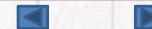


Nom des composants avec Majuscules
React.Component != html

Les **balise** portent le **nom** du **composant** qui doit être insérer

Il reçoit ses **props** sous forme d'**attributs** de balises
Chaque attribut créera un champs dans les props
Peut recevoir des valeur, des objets des fonctions, ...

Il doit être **importer / exporter correctement** et connu lors de la présence de la balise



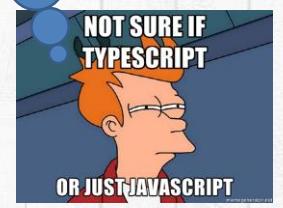
Tout le reste n'est que JS

- High Order Component

- Approche HighOrderComponent

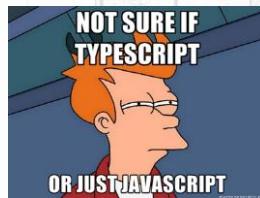
- On parle de wrapper de component
 - Un composant gérant un enfant, connu ou passé en children, et de datas,
 - » en régulant le partage fils<-> parent<->app(niveau supérieur)

- Approche store, redux, mobx, ...



React assemble, rien d'autre

- React ne permet que la confection rapide de l'app
 - Assembler avec des component
- Qui s'occupera du reste ?
 - l'écosystème de react avec les nombreuses librairies tierces
 - Avec des appels de fonctions js native



❑ Dans cette partie, React Native, nous avons abordé :

- ✓ Le rôle du moteur React
- ✓ Le superset de JavaScript JSX.
- ✓ Concept de composant
- ✓ La logique props & état
- ✓ High Order Component



3) ARCHITECTURE D'APPLICATION



Sommaire

- ❑ Dans cette partie, Architecture d'application, nous aborderons :
 - ❑ Configurer un composant : state & props & hooks.
 - ❑ React Native et MVC.
 - ❑ Binding des Inputs
 - ❑ Travaux pratiques Initialisation d'un projet React Native.
Configuration d'un composant.



Composant React

- 2 possibilité pour la création de composants
 - Class
 - **Historiquement** pour les statefull
 - Usage déprécié au profit des *function* avec useState/useEffect
 - Function
 - **Historiquement** uniquement stateless
 - Statefull depuis react 16.8 avec useState & useEffect

- ClassCmp.jsx

```
Class ClassCMP extends React.Component{
  constructor(props){
    super(props);
    this.state={};
  }
  render(){
    return <composant/>
  }
}
Export default ClassCMP
```

- FunctionCmp.jsx

```
Const FunctionCmp=(props)=>{
  const [state,setState]=useState({})
  useEffect(()=>{ },[ ])
  return <Composant/>
}
Export default FunctionCmp
```

PropsType

- Contrôle du type de contenu pour les props :
 - `propTypes`
 - `isRequired`
 - Sur tous les types
 - **Rend obligatoire, si pas présent = facultatif**
 - Types existant :
 - `string`
 - `number`
 - `object`
 - `Array`
 - `Bool`
 - `Func`, ...

Liste exhaustive :
<https://fr.reactjs.org/docs/typechecking-with-proptypes.html#prop-types>

Composant.js
`import PropTypes from 'prop-types';`

class Greeting extends React.Component { ... }

`Greeting.propTypes = {`
 name: PropTypes.string
};



defaultPropsType

- Contrôle du contenu des props :

- Positionner des valeurs si elles ne sont pas fournies dans les props lors de la création du composant <MonCmp .../>
- Même fonctionnement que propsType mais avec des valeurs, au lieu des types

📄 **Composant.js**

```
Greeting.defaultProps = {  
    name: 'nom inconnu'  
};
```



State et data dynamique du component

- Le state est un conteneur pour les variables du composant
- Il est initialisé avec un état initial
 - Le papillon commence sa vie en chenille
- Il est modifié par le composant ou autre,
 - La chenille subit des changements de son state
 - On parle de l'état du papillon à cette instant
 - Son render a été affecté
 - La chenille subit un changement de son state
 - On parle du nouvel état issu de l'état initial et ayant subit déjà un changement



Attention ici on parle de state muté pour certains usages on devra créer une nouvelle copie de la chenille à chaque état pour préserver l'ancienne état de la chenille on parle de state immutable

Class et state (*à éviter*)

- Le state doit être initialiser dès la création

- Dans le constructeur :

 └ Cmp.jsx

```
Class Cmp extends React.Component{
  constructor(props){
    super(props);
    this.state={  contenu de départ  };
  }
}
```

A moins d'être sur d'avoir
à le faire et que c'est
**vraiment la seule et
unique solution possible**

- Le state doit jamais être accédé en écriture direct

- Lors d'écriture sur le state du composant utiliser this.setState({})

- Gestion async du changement

- Déclenchement des fonction de cycle de vie (*shouldComponentUpdate*)

- Déclenchement du **render()** en fin de modification du state

- Cascade de mise à jour des props des enfants

<https://fr.reactjs.org/docs/react-component.html#shouldcomponentupdate>

Le cycle de vie

- Tout au long de la vie du component des fonctions vont être exécutées en fonction de l'état et de son contenu
- Doc :
<https://fr.reactjs.org/docs/react-component.html#the-component-lifecycle>

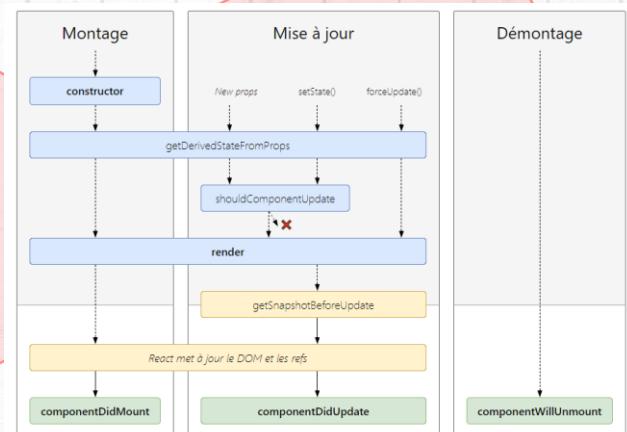


Diagram : <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

hooks

- La gestion d'état est historiquement limitée à la **class**
 - Un seul objet d'état pour tout l'objet
 - Une seule fonction de mise à jour du state **setState()**
- Les **hooks** permettent de **définir**
 - **plusieurs state** avec chacun leur fonctions de mise à jour
 - Depuis la version 16.8 de react
- Les **hooks d 'effet** sont des hooks avec la capacité d'avoir des fonctions de cycle de vie



Les **hooks** sont souvent utilisé pour les états **interne** d'un composant

Doc :

Hooks : <https://fr.reactjs.org/docs/hooks-intro.html>

Effet Hooks : <https://fr.reactjs.org/docs/hooks-effect.html>

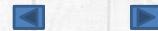
```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);
  // Similaire à componentDidMount et componentDidUpdate : useEffect(() => {
  // Met à jour le titre du document via l'API du navigateur document.title = `Vous avez
  cliqué ${count} fois`;
  });
  return ( <div> <p>Vous avez cliqué {count} fois</p> <button onClick={() =>
  setCount(count + 1)}> Cliquez ici </button> </div> );
}
```

Function & useState

- Les hooks existent seulement sur les composants *function*
- Comme pour les class les hooks peuvent réexécuter le rendu d'un composant en cas de changement de référence de valeur
- Les valeurs d'état peuvent être observées pour le cycle de vie
- Pour obtenir une valeur étatique

```
const [etat, setterEtat]=useState(valeurInitial)
```

- Etat représente la valeur de l'état **en lecture seule**
- SetterEtat permet une mise à jour **async** de la ref. de valeur d'état
 - Le setter nécessite une nouvelle référence de valeur complète (reconstituer l'objet si valeur objet)



Function & cycle de vie

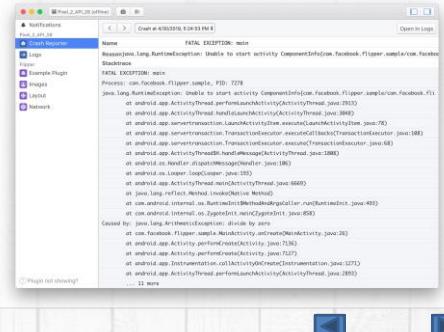
- Contrairement au class une seul fonction pour didMount, didUpdate, willUnmount
- Le montage et l'update exécute la fonction fournis
- Exemple de useEffect :

```
useEffect(()=>{  
    //instructions  
    return ()=>{ /*willUnmount function*/ }  
},[valeur1,...])
```
- La fonction rentrée (*facultative*) sera exécutée au démontage du composant
- Plusieurs configuration possible grâce au tableau de valeur en 2eme param
 - Aucun tableau : exécution à chaque nouveau rendu (équiv. componentDidMount + componentDidUpdate)
 - Tableau vide : exécution mount uniquement (équiv. componentDidMount)
 - Tableau avec au moins une référence : exécution à chaque changement d'une référence du tableau (mount + update d'une valeur du tableau)



Debug React native

- Avant la v0.69
 - Depuis <http://localhost:8081>
 - Accès à la console js et breakpoint
 - Déprécié depuis la 0.62 tend à être supprimé au profit de flipper
- Depuis la v0.69
 - Grâce à flipper,
 - npm install --save-dev react-native-flipper**
 - Accès à plein d'outils : Layout Inspector, Network, Databases, Images, Shared Preferences, Crash Reporter, React DevTools, Metro Logs
 - Possibilité d'ajout de plugins pour plus d'outils
 - Exemple : react-navigation, ...



<https://fbflipper.com/docs/features/react-native/>

Flipper peut être mis en œuvre avant la 0.69 avec installation et configuration manuelle

Install watchman : <https://facebook.github.io/watchman/> (PS >choco install watchman)

Activé et installer les plugins, certains nécessitent l'install global du composant pour son intégration dans flipper (react-devtools)

Redux

- Redux manage un store pour stocker un état pour une app ou un groupe de composants
 - Il nécessite une fonction qui dispatch et exécute les tâches possibles sur le store
 - Prendre produit, retour produit, payer produit, ...
 - L'ensemble agit sur l'état du magasin
 - Les tâches sont affectées à un reducer
 - Son rôle est de
 - » modifier l'état en **un nouvel état** issue du précédent
 - » le **rendre l'état** ou une valeur de l'état
 - Il **agit** en fonction d'**actions** qui lui sont soumis
- React-redux est une intégration plus simplifiée des stores dans react



State & Store

- 2 constructions possible :
 - *State locaux distribués par props*
 - Impose un/des **composants** pour **maintient du state**
 - Gestion de **propagation** de l'**état niveau** d'arborescence **par niveau** d'arborescence
 - **Composants dépendants** de la fourniture des données d'état depuis le parent
 - Impose des **rechargements complet d'arborescence** enfant (a partir du composant maintenant le state) à chaque changement
 - **Corruption** possible des états par les composants (`useState`)
 - *State partagé par un store*
 - Les composants sont **autonome** pour l'accès aux états
 - Le **store** manage et **contrôle** les capacités de **modifications**
 - **Alerte** les **composants connectés** des modifications
 - Permet le **rendu uniquement** des **composants connectés** aux données
 - uniquement **si le changement doit les affecter**

- L'état local a toujours son usage
 - Lorsque les informations stocké sont propre au composant et sans besoin de partage
- Le store ne stocke des datas sérialisables
- NB : il est préférable de concevoir le magasin et a structuration de data en amont



Redux devtools

- Inspecteur redux
 - Liste les actions
 - leurs contenus
 - Les changements affectés par l'action
- Permet un dispatch à la volée
- Import/export de scenario d'action au store

The screenshot shows the Redux DevTools extension integrated into a browser window. The top navigation bar includes tabs for 'Actions' (which is selected), 'Settings', 'Console', 'Sources', 'Performance insights', 'Réseau', 'Redux', and 'React App'. Below the navigation is a toolbar with icons for 'Reset', 'Revert', 'Sweep', 'Commit', and others.

The main area displays a list of actions on the left and their details on the right. The actions listed are:

- @@INIT 4:11:19.46
- INIT +00:00.00
- UPDATE_MEME +00:01.01
- UPDATE_MEME +00:00.01
- INIT_VALUES +00:00.02
- UPDATE_MEME +00:00.00
- UPDATE_MEME +00:03.69

A specific action, 'UPDATE_MEME (6)', is expanded to show its payload:

```
1 {  
2   type: ''  
3 }
```

Below the list, there are tabs for 'Action', 'Action', 'State', 'Diff', 'Trace', and 'Test'. The 'Action' tab is active. On the right, there are buttons for 'Tree', 'Chart', and 'Raw'. At the bottom, there's a 'Custom action' section with a code editor containing the same JSON object, and buttons for 'Dispatch' and other actions like 'Log monitor', 'Chart', and 'RTK Query'.

redux

- Pour gérer une store
 - Un reducer
 - Reçoit :
 - une action
 - L'ancien state
 - avec un state initial (valeur par def.)
 - *la nouvelle valeur à manager dans le state
 - Retourne un **nouvel état** assemblé issue de l'ancien (**non muté**)
 - Les créateurs d'action
 - Renvoie l'objet d'action à dispatcher
 - Un store
 - Le store sera l'interface des composants avec le reducer
 - Il permet d'abonner des fonction aux changements du state du magasin
 - Il **dispatch** les appels au reducer avec le renvoi de l'ancien state qu'il manage
 - » Seul l'action sera à fournir
 - Crédit du store & affectation du reducer
 - Abonnement
 - Lecture de l'état du store
 - Usage du store
- Exemple

```
function counter(state = 0, action) {
  switch (action.type) {
    case 'INCREMENT':
      return state + action.payload
    case 'DECREMENT':
      return state - action.payload
    default:
      return state
  }
}

export const increm=(val)=>{
  return {
    type: 'INCREMENT', payload:val
  }
}

export let store = createStore(counter)

store.subscribe(() =>
  console.log(store.getState())
)

store.dispatch({ type: 'INCREMENT' })
store.dispatch(increm(1))
```

*facultatif

createStore vs configureStore

- createStore est déprécié, privilégié configureStore plus complet
 - Cette fonction est par redux toolkit
 - Gestion en *slice* des *reducers*
 - Rendu des actions de la *slice*
 - maSlice.actions renvoie une fonction *dispatchable* au store pour chaque cas du *reducer*
 - Rendu du *reducer* complet de la *slice*
 - maSlice.reducer renvoie le reducer de la slice
 - Permet la « mutabilité » des valeurs dans les reducers
 - Mutabilité d'un brouillon pour génération d'un nouveau state non muté



Création de slice/store avec redux toolkit

- `createSlice`
 - Name : nom de la tranche
 - `initialState`: état initial de la tranche
 - `Reducers`: fonctions de réduction
 - Recevant le state
 - Mutant le state avec la nouvelle valeur
- `maSlice.actions` : rendu des fonctions *dispatchables* d'actions
- Déclaration du store avec `configureStore`
 - Rendu du reducer par `maSlice.reducer`
- Dispatch d'une action

```
import { createSlice, configureStore } from '@reduxjs/toolkit'

const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {
    incremented: state => {
      state.value += 1
    },
    decremented: state => {
      state.value -= 1
    }
  }
})

export const { incremented, decremented } = counterSlice.actions

const store = configureStore({
  reducer: counterSlice.reducer
})

store.dispatch(decremented())
```

React-redux

- Il permet l'implémentation facilitée de redux dans les composants react
 - Il fournit un composant pour produire un contexte *store* pour tous ces enfants et toutes les fonctions de react-redux

```
<Provider store={storeVar}>
    <LesComposantsDeLApp/>
</Provider>
```

- Il existe deux façons de connecter un composant au store



React-redux connect()()

ancienne bonne pratique

- Connect

- Grâce à deux fonctions et un HOC, il permet de projeter dans les props d'un composant des valeurs issues du store ainsi que des dispatch spécifiques

- AVANTAGES :

- Garder le composant classique et en créer en parallèle une version connectée utilisable dans les deux cas
- Dissocier du composant son rôle et son accès aux valeurs
- Faciliter la projection des props dans le composant

```
const MemeSVGViewer:React.FC<IMemeViewerProps>=
(props)=>{return <div onClick={props.click}>...</div>}

function mapStateToProps(state, ownProps) {
  return {
    ...ownProps,
    meme: state.current,
    image: state.ressources.image,
  };
}

function mapDispatchToProps(dispatch:Function){
  return{
    click:()=>{dispatch({action:'VIEWER_CLICKED'})},
  };
}

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(MemeSVGViewer)
```

React-redux by hooks

- useDispatch
 - Fonction renvoyant la fonction dispatch du store
 - Fournie par

```
<provider store={store}>
  <App/>
</provider>
```
- useSelector
 - Fonction Hook de sélection de contenu du store
 - Fonction qui reçoit le state du store
 - Fournie par *provider*
 - Retourne la sélection voulue provenant du state du store
- Connecter un component :

```
import UnconnectedCmp from '../UnconnectedCmp/UnconnectedCmp'
export default const ConnectedCmp=(props)=>{
  dispatch=useDispatch()
  data=useSelector(s=>s.data)
  return <UnconnectedCmp onaction={(param)=>dispatch({type:'ACT', payload:param})} content={data}/>
}
```



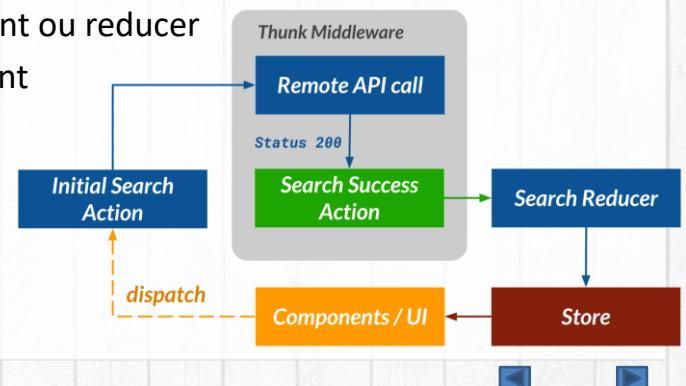
Redux-middleware

- Le middleware est un intermédiaire d'actions entre le dispatch et l'exécution du réducteur
- Comment il modifie le comportement du dispatch ?
 - S'interpose avant l'appel du *reducer*
 - Effectue des tâches *sync* ou *async*
 - Peut redéclencher un dispatch
 - Renvoie l'action interceptée au prochain middleware jusqu'au *reducer*



Redux thunk

- Le dispatch parcourt donc le/les middleware .
- Le middleware prend la décision de
 - Faire passer l'action au suivant ou reducer
 - Ou de la traiter préalablement
 - Puis renvoyer une action une fois la tâche terminée



Redux-thunk syntaxe

- **asyncThunk pour redux**

```
createAsyncThunk('typeName', await Promise)
```

- Fabrique une action *dispatchable suffixer par l'état de la promise*
 - **typeName/fulfilled**
 - **typeName/rejected**
 - **typeName/pending**
- **extraReducer du configureStore**
 - Pour les *async thunk*, il est nécessaire que le traitement des cas soit construit par un builder hors du reducer synchrone
 - » **addCase**
 - *typeName/rejected, typeName/fulfilled, ...*
 - » **addMatcher**
 - **/pending, typeName/*, ...*
 - » **addDefault**

- **Dispatcher l'asyncthunk**

- *Store.dispatch(getMessages(data))*

- messagesSlice.ts
 - Dispatchable async func

```
export const getMessages = createAsyncThunk(
  "messages/hydrate",
  async () => {
    const pr = await fetch("http://.../msgs")
    return await pr.json()
  }
);

const messageSlice = createSlice({
  name: "messages",
  initialState,
  reducers: {sync reducer content},
  extraReducers: (builder) => {
    builder.addCase(getMessages.pending, (state) => {
      state.status = "pending";
    });
    builder.addCase(
      getMessages.fulfilled,(state, action) => {
        state.status = "idle";
        state.value = action.payload;
      }
    );
    builder.addCase(getMessages.rejected, (state) => {
      state.status = "failed";
    });
  },
});
```

❑ Dans cette partie, Architecture d'application, nous aborderons :

- ✓ Configurer un composant : state & props.
- ✓ React Native et MVC.

- ✓ L'arrivée de Redux : le store, le reducer, les actions.
- ✓ Mise en place de redux avec `@reduxjs/toolkit`
- ✓ L'implementation react-redux
- ✓ Appel async pour redux & `asyncThunk`

- ✓ Travaux pratiques Initialisation d'un projet React Native.
Configuration d'un composant.



CONSTRUIRE SON INTERFACE



Sommaire

- ❑ Dans cette partie, Construire son interface, nous aborderons :
 - ❑ Les composants de base (View, Text et Image) et leurs cycles de vie.
 - ❑ Les événements Touch, la ListView et la ScrollView.
 - ❑ Organiser le layout de l'application. Mise en page avec Flexbox.
 - ❑ Styler les composants.
 - ❑ Les différentes solutions de navigation entre les pages.
 - ❑ Composants avancés.
 - ❑ Utiliser des composants de la communauté.
 - ❑ Ajouter des animations et des transitions.
 - ❑ Travaux pratiques Développer une première application.



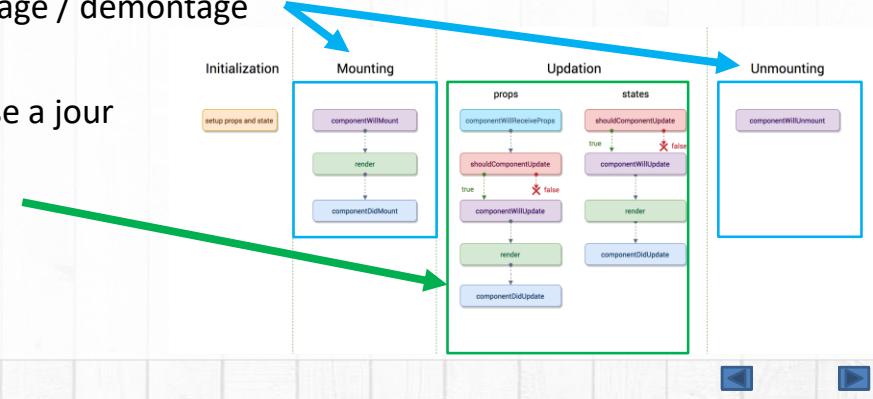
Cycle de vie des Composants

- Le cycle de vie permet le déclenchement de méthodes à des instant définit de la vie d'un composant

– Lors du montage / démontage

– Lors de la mise a jour

- Des props
- Du state



Liste des composants *built-in*

Les composants built-in sont

- les composants fournis nativement par React-native
 - Ils sont fortement compatibles pour tous les os
 - Une série iOS et une Android, non cross-plateforme
- Ils acceptent certaines propriétés de styles standard à RN.
 - Un warning sera affiché à chaque style incompatible
- Ils possèdent une série de props pour gérer les événements

<https://reactnative.dev/docs/components-and-apis#basic-components>

• Composants basiques

- » View, ScrollView
- » Text, TextInput
- » Image
- » StyleSheet

• Composants de formulaire

- » Button, Switch

• Composants de listes

- » ListView, SectionView

• Composants divers

- ActivityIndicator, Alert, Modal
- StatusBar



Composant View

- Container de composants simple
 - Ressemble fortement à <div> en html
- Style limité : [doc](#)
 - border-*
 - elevation
 - Opacity
- *Flex-able*
 - *flex-direction : row ou column*
 - *flex*



Text

Le composant **Text** est

un conteneur d'affichage de contenu textuelle, du contenu enfant

Les sauts de ligne, se font grâce aux échappements Unicode ('\n')

Il est styleable

Il support les events

<https://reactnative.dev/docs/text>

- Style limité à : [DOC](#)

- Gestion de fonts

- color, fontSize, fontFamily,

- Alignment

- textAlign, TextAlignmentVertical, writingDirection, ...

The screenshot shows a code editor window titled "Nested Text Example" with an "Expo" logo. The code uses StyleSheet.create to define two styles: "baseText" (bold) and "innerText" (red). It then creates a component "BoldAndBeautiful" that returns a nested element where the inner text is styled with both bold and red colors. A preview window below shows the resulting text "I am bold and red". At the bottom, there are tabs for "Preview" (selected), "My Device", "iOS", "Android", and "Web".

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

const BoldAndBeautiful = () => {
  return (
    <Text style={styles.baseText}>
      I am bold
      <Text style={styles.innerText}> and red</Text>
    </Text>
  );
};

const styles = StyleSheet.create({
  baseText: {
    fontWeight: 'bold'
  },
  innerText: {
    color: 'red'
  }
});

export default BoldAndBeautiful;
```

TextInput

Le composant **TextInput** est

un conteneur de saisie de texte, il peut être multilignes

Les sauts de ligne, se font grâce aux échappements Unicode {'\n'}

Il est style-able

Il support les events

<https://reactnative.dev/docs/text>

- Style limité à : [DOC](#)

– La issue de <Text>

- Props

– valeur

- Value, secureTextEntry,

```
TextInput ⓘ ⓘ
import React, { Component } from 'react';
import { TextInput } from 'react-native';

const UselessTextInput = () => {
  const [value, onChangeText] = React.useState('Useless Placeholder');

  return (
    <TextInput
      style={{ height: 40, borderColor: 'gray', borderWidth: 1 }}
      onChange={text => onChangeText(text)}
      value={value}
    />
  );
}

export default UselessTextInput;
```

Image

Le composant **Image** est

un conteneur d'affichage d'image

La props source reçoit un objet avec le contenu de l'image (base64) ou l'url de l'image

Il est styleable

Il support les events

<https://reactnative.dev/docs/image>

- **Style limité à :** [DOC](#)

- Couleur

- tintColor, overlayColor

- Border

- Size

- resizeMode, overflow

- **Props source:**

- Contenu packagé dans l'app:

- source={require('@expo/chemin/fichier.png')}

- Contenu url:

- source={{uri: 'https://tiny_logo.png', }}

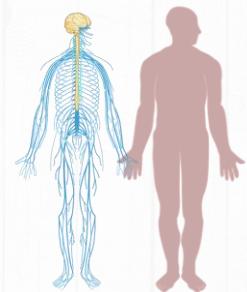
- Contenu base64 :

- source={{uri:'data:image/png;base64,...'}}



Composant avec styles

- Contrairement à react, RN** décrit le style grâce à un objet js de style et non en css pur.
 - Ressemblant à la transposition JS du css
 - Possédant des champs composé du style pour cette ensemble
 - Chaque built-in composants acceptent la *props style*
- La fonction de l'objet **stylesheet** fournit par react-native permet la création de cette objet grâce à la fonction **create**



RN** : React-native

Documentation : <https://reactnative.dev/docs/0.59/style>

Composant avec styles

- Contrairement à react, RN** décrit le style grâce à un objet js de style et non en css pur.
 - Ressemblant à la transposition JS du css
 - Possédant des champs composé du style pour cette ensemble
 - Chaque built-in composants acceptent la *props style*
- La fonction de l'objet **stylesheet** fournit par react-native permet la création de cette objet grâce à la fonction *create*

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

const styles = StyleSheet.create({
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});

export default class LotsOfStyles extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>just red</Text>
        <Text style={styles.bigBlue}>just bigBlue</Text>
        <Text style={[styles.bigBlue, styles.red]}>bigBlue,
        then red</Text>
        <Text style={[styles.red, styles.bigBlue]}>red, then
        bigBlue</Text>
      </View>
    );
  }
}
```

RN** : React-native

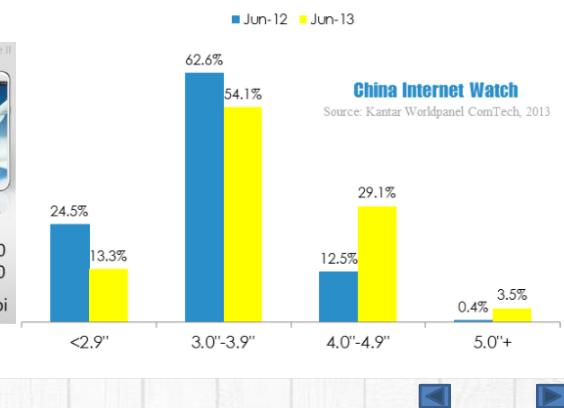
Documentation : <https://reactnative.dev/docs/0.59/style>

Dimensions & ScreenSize & style

- Une API native provide les size, ratio, des écrans
- Les % sont accepter en unités de dimensions, marges, ...

Galaxy S 4	HTC One	iPhone 5	Optimus G Pro	BlackBerry Z10	Xperia Z	Nexus 4	Galaxy Note II
5" x 1920 x 1080 441ppi	4.7" x 1920 x 1080 468ppi	4" x 1136 x 640 326ppi	5.5" x 1920 x 1080 401ppi	4.2" x 1280 x 768 355ppi	5" x 1920 x 1080 441ppi	4.7" x 1280 x 768 318ppi	5.5" x 1280 x 720 267ppi

Chinese Smartphone Screen Size



Layout & flexbox

- LE flexbox dispose des composants
 - En ligne
 - flexDirection : row
 - flexDirection : reverse-row
 - En colonne
 - flexDirection : column
 - flexDirection : reverse-column
- Les dimension sont fournies aux composants
 - Par : width & height
- Propriété de flexibilité :
 - flex-grow, flex-shrink et flex-basis ou la super propriété : **flex**
- **Générateur de code pour layouts :**
<https://yogalayout.com/>



Gestion de la navigation

- La gestion de navigation est fournit par un module react & RN

- Module react react-navigation

```
npm install @react-navigation/native @react-navigation/stack
```

- Implémentation React-native

```
npm install react-native-reanimated react-native-gesture-handler react-native-screens  
react-native-safe-area-context @react-native-community/masked-view
```

Documentation : <https://reactnative.dev/docs/navigation>



Navigation avec @react-navigation/native installation

- Installation
 - npm install @react-navigation/native
 - npm install react-native-screens react-native-safe-area-context
 - Ios :
npx pod-install ios
 - Android : dans /android/app/src/main/java/<your package name>/MainActivity.kt
- Relancer la compilation (yarn android / ios)

```
import android.os.Bundle
class MainActivity : ReactActivity() {
    /**
     * creator for React-native navigation
     */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(null
    }
    //...
}
```



React-native screen + react-native-safe-area-cintext by expo :npx expo install react-native-screens react-native-safe-area-context

Component StackNavigator

- Pour la navigation il est nécessaire de:
 - Dans l'ensemble de l'app
 - Créer un NavigationContainer
 - Dans le navigationContainer
 - Un stackNavigator composé de stackScreen définissant toutes les fenêtres navigable

```
import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

const App : React.JSX.Element = props =>
<NavigationContainer>
  <Stack.Navigator initialRouteName="Home">
    <Stack.Screen name="Home" component={Main} />
    <Stack.Screen name="Start rent" options={{title: 'hello'}}>
      {props => <StartRent {...props} />}
    </Stack.Screen>
  </Stack.Navigator>
</NavigationContainer>

export default App;
```

Navigation

- Pour la navigation il est nécessaire de:
 - Dans l'ensemble de l'app
 - Créer un NavigationContainer
 - Dans le navigationContainer
 - Un stackNavigator composé de stackScreen définissant toutes les fenêtres navigable
- Pour naviguer un simple appel à
navigation.navigate

```
import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
const Stack = createStackNavigator();
const MyStack = () => { return (
<NavigationContainer>
  <Stack.Navigator>
    <Stack.Screen name="Home" component={HomeScreen} options={{ title: 'Welcome' }} />
    <Stack.Screen name="Profile" component={ProfileScreen} />
  </Stack.Navigator>
)</NavigationContainer> ); };


```

- Dans les composants navigable :

```
const HomeScreen = ({ navigation }) => {
  return (
    <Button title="Go to Jane's profile" onPress={() =>
      .navigation.navigate('Profile', { name: 'Jane' })
    } /> );
};
const ProfileScreen = () => { return
<Text>This is Jane's profile</Text>; };
```



Gestion bouton matériel android

- <https://reactnative.dev/docs/backhandler>
- <https://newbedev.com/react-native-device-back-button-handling>

Toast android

- <https://reactnative.dev/docs/toastandroid>

Plateform management

- <https://reactnative.dev/docs/platform>

vibrate

- <https://reactnative.dev/docs/vibration>

share

- <https://reactnative.dev/docs/share>

Composants tiers

- RN s'appuie sur des bibliothèque écrite par la communauté officielle :

- @react-native-community :
 - <https://www.npmjs.com/search?q=%40react-native-community&page=0&perPage=50>
 - Netinfo : pour les états réseaux
 - Datetimepicker : champs de saisie pour les date
 - Geolocation
 - Clipboard
 - ImagePicker
 - ,...



Composants et lib de UI

- React native elements

- <https://www.npmjs.com/package/react-native-elements>



- Native-base
 - <https://www.npmjs.com/package/native-base>
- D'autres composants ou librairies:
 - <https://github.com/jondot/awesome-react-native#ui>



React-native-elements

- Une librairie de composants UI
 - Complete
 - Badges, avatar, list, slider, rating, pricing, overlay, socialIcon, ...
 - Simple
 - Yarn add react-native-elements ou npm i -s react-native-elements
 - Expo install react-native-elements



Conclusion

□ Dans cette partie, Construire son interface, nous aborderons :

- ✓ Les composants de base (View, Text et Image) et leurs cycles de vie.
Les événements Touch, la ListView et la ScrollView.
- ✓ Organiser le layout de l'application. Mise en page avec Flexbox.
- ✓ Styler les composants.
- ✓ Les différentes solutions de navigation entre les pages.
- ✓ Composants avancés.
- ✓ Utiliser des composants de la communauté.
Ajouter des animations et des transitions.
- ✓ Travaux pratiques Développer une première application.



5) LES FORMULAIRES ET LA GESTION DES DONNÉES



❑ 5) Les formulaires et la gestion des données

- ❑ Les principaux composants de formulaires.
- ❑ La validation de la saisie et la gestion des erreurs.
- ❑ Redux-form et alternatives.
- ❑ Récupération des données : xmlhttprequest et fetch.
- ❑ Le stockage local.
- ❑ La gestion offline.

- ❑ Travaux pratiques Développer un formulaire, valider les données et assurer la persistance des données.



Principaux composants de formulaire

- Les composants de formulaire sont
 - TextInput :
 - Champs de saisie textuel (mono ou multilignes)
 - Switch
 - Slider à états modifiable (on / off)
 - Button
 - Bouton d'action (pour le formulaire et l'appli en générale)
- Vous pouvez bien sur utilisé d'autres composants créer ou ajouter a votre projet pour cette usage
- Cf. section built-in Components



React-redux-form

- react-redux-form est une implémentation de redux-form
- Permet de connecter un formulaire au store redux
 - Propose de nouveaux components
 - Form → base du formulaire (equiv. A form en html)
 - Field → un champs de saisie supportant un input enfant, possède une connexion à un model
- Documentation : <https://redux-form.com/>



tcomb

- Une alternative pour les formulaires
 - <https://github.com/gcanti/tcomb-form-native>
- Non disponible sur la dernière version de react-native
- Fournit un formulaire automatique en fonction d'un structure de données





AsyncStorage

- Stockage persistant et synchronisable
 - AsyncStorage est déprécié :
 - En remplacement :
 - @react-native-community/async-storage
 - » Doc : <https://react-native-community.github.io/async-storage/>
 - Un couple clef / valeur
 - Synchro grâce aux états de réseaux
 - Fetch lors de la réception réseaux activé
 - Pour de plus larges stockage
 - Stockage sqlite :
<https://www.npmjs.com/package/react-native-sqlite-storage?activeTab=readme>

- Exemple d'async storage :

```
import AsyncStorage from '@react-native-community/async-storage';
// Enregistrement
const storeData = async (value) => {
  try {
    await AsyncStorage.setItem('@storage_Key', value)
  } catch (e) { /* saving error */ }
}
// Lecture
const getData = async () => {
  try {
    return value =
    await AsyncStorage.getItem('@storage_Key')
  } catch(e) { /* error reading value */ }
}
```

async-storage supporté par :

- iOS
- Android
- Web
- MacOS
- Windows

Il est possible de stocker une chaîne JSON strigifier

```
const jsonValue = JSON.stringify(value)
await AsyncStorage.setItem('@storage_Key', jsonValue)
```

Api info de connexion réseau : [@react-native-community/netinfo](#)

Synchro web service : <https://stackoverflow.com/questions/55700655/store-data-offline-and-sync-once-online-using-react-native-and-redux-store>

5) Les formulaires et la gestion des données

- ✓ Les principaux composants de formulaires.
- ✓ La validation de la saisie et la gestion des erreurs.
- ✓ Redux-form et alternatives.
- ✓ Récupération des données : xmlhttprequest et fetch.
- ✓ Le stockage local.
- ✓ La gestion offline.

Travaux pratiques Développer un formulaire, valider les données et assurer la persistance des données.



INTERAGIR AVEC LE TERMINAL



Sommaire

❑ Dans cette partie nous aborderons

- ❑ Interagir avec le terminal
- ❑ Les principales API natives de React Native.
- ❑ Utiliser les plugins Cordova/PhoneGap.
- ❑ Développer un module natif.



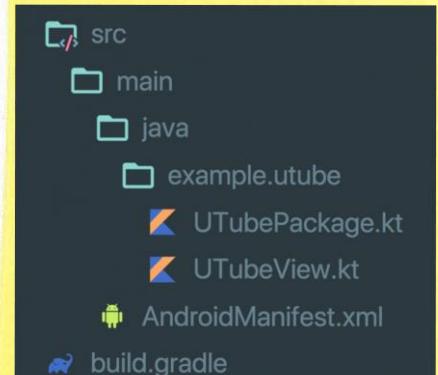
Cordova to RN

- Sur 100% des plugins cordova :
 - 70% des plugin existent déjà en plugins RN ou en fonctions natives
 - 15% sont les plugin de cordova Core
 - 15% : homologue RN considérer dangereux
- Plugin import Cordova :
 - <https://github.com/axemclion/react-native-cordova-plugin>



Des composants natif sous react-native

- Il est possible de créer des composant natif
- Exportation et usage du composant dans react comme un composant react
 - Chaque plateforme à son code de composant natif



Créer un composant kotlin : <https://callstack.com/blog/writing-a-native-module-for-react-native-using-kotlin/>

Créer un composant swift : <https://teabreak.e-spres-oh.com/swift-in-react-native-the-ultimate-guide-part-2-ui-components-907767123d9e>

Interaction de la console

- Rn propose des interactions :
 - `console.log()`
 - Permet de sortir vers la console pc
 - Des modules existent pour utiliser une console dans l'app
 - Le react devtool
 - Propose la représentation de la window rendu (rendered) avec
 - ses states
 - ses props
 - Sa structure enfant



API et composants RN

- RN propose une série de composants et d'API
 - Les composants basique
 - Les composants de formulaire
 - Composants de listes
 - Une série ANDROID
 - Une série ios
 - Autres :
 - Modal
 - Dimensions
 - Outils du clavier
 - ...
 - Liste exhaustive : <https://reactnative.dev/docs/components-and-apis>



Conclusion

□ Dans cette section nous avons aborder :

- ✓ Interagir avec le terminal
- ✓ Les principales API natives de React Native.
- ✓ Utiliser les plugins Cordova/PhoneGap.
- ✓ Développer un module natif.



USAGES AVANCÉS



Sommaire

- ❑ Best Practices et erreurs fréquentes.
- ❑ Tests unitaires et fonctionnels.
- ❑ Publier l'application.
- ❑ Mises à jour Over The Air.
- ❑ Frameworks et outils complémentaires



Publication

- La publication nécessite la publication de l'application à expo
 - Pour tous les os
 - Définition du app.json
- Publication grâce à la commande expo
 - expo publish
 - » Commande proche de npm pour la publication

```
• App.json
{
  "expo": {
    "name": "App Name",
    "icon": "./path/to/your/app-
icon.png",
    "version": "1.0.0",
    "slug": "app-slug",
    "sdkVersion": "XX.0.0",
    "ios": {
      "bundleIdentifier":
        "com.companyname.appname"
    },
    "android": {
      "package":
        "com.companyname.appname"
    }
  }
}
```

Publication expo

- Builder l'app native
 - Commande expo
 - Expo build:android
 - Expo build:ios
 - Stocker les archives de version générer
 - Nécessaire pour l'update de l'app à expo
 - Sauver le *keystore android*
 - expo fetch:android:keystore



Publication google Play

- Pre requis :
 - Une App Builder
 - Un ID unique représentant l'app
 - Un fichier apk 100MO l'APK, fichier d'extension Si +
 - compte google play
 - <https://play.google.com/apps/publish/signup/>
- Create an app
 - Saisir les infos de l'app.
 - App release
 - Content rating
 - Gestion du prix de l'app
 - Dans app releases, publish ...



Nécessite apk signé: <https://developer.android.com/studio/publish/app-signing>

Publier une app android :

<https://developer.android.com/studio/publish/app-signing>

Publier tuto : <https://instabug.com/blog/how-to-submit-your-app-to-the-google-play-store/>

Etapes proche pour ios : <https://instabug.com/blog/how-to-submit-app-to-app-store/>

Mises à jour Over The Air

- Mise a jour OTA
 - Mise a jour automatique des app clients
 - <https://docs.expo.io/guides/configuring-ota-updates/>
 - Automatique
 - Manuel
 - désactiver



Conclusion

- ❑ Best Practices et erreurs fréquentes.
- ❑ Tests unitaires et fonctionnels.
 - ✓ Publier l'application.
 - ✓ Mises à jour Over The Air.
- ❑ Frameworks et outils complémentaires



ANNEXE TECHNIQUE

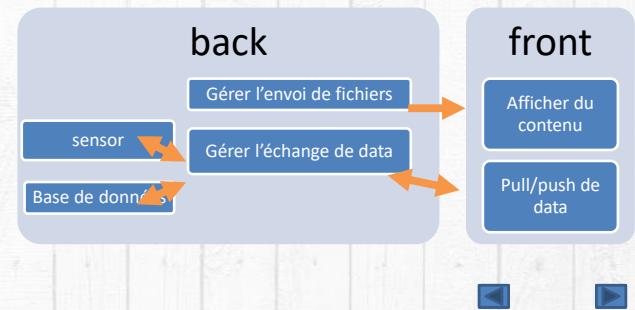


Qui fais quoi ?

- Qui à besoin de quoi ?
 - Le client, Un navigateur
 - Charge le site et met en action le JS
 - Il géra grâce au js les interactions IHM
 - Chargerà une version du site html/css/js
 - Et les librairies js associées
 - Il géra les échanges http
 - Accès à des données par Webservice (REST, json)

– Le serveur

- Il Répondra aux requêtes http pour accéder aux fichiers du site
- Il répondra aux requêtes liées aux données (webservice tel que REST)
- Il géra des échanges internes pour l'accès au sgbdr, aux sensors, ...



Un web service

- **Web Services :**

- client,
 - celui qui accède
- fournisseur,
 - celui qui diffuse
- annuaire de services
 - celui qui recense ex UDDI pour les WSDL *(SOAP)
- intégration d'applications
 - Des API

- **Métaphore :**

le serveur du restaurant et sa "carte du menu", on se sers pas directement en cuisine



Un bistro service ?

- Un web service c'est comme Un bistrot
 - Il est de bon ton d'adopter le protocole adéquat
 - Bonjour
 - Un serveur prend ma commande
 - Une bière pression
 - L'hydromel du patron est une bouteille Qui ne peut être servis aux clients
 - Le serveur me sers ma commande si cela est possible.
 - Je consomme ce qui m'a été servi dans
 - le verre adéquat
 - le volume parfait



Un bistro service?

- Dans certaines enseignes il est possible de savoir ce qui est possible de commander
- Il est même expliquer que
 - Sur présentation d'un coupon Promo une réduction spéciale sera faite

APÉRITIFS

	plat du jour 13,50 euros
MARIE	
Poêlée fermière rôti à l'huile noisette	
MERCREDI	
Bouillabaisse de rosé et riz Pilaf	
JELUCH	
Steak de bœuf braisé au vin rouge	
VENREDI	
Filet d'ailette de bœuf et pomme de terre sautée au beurre et huile d'olive et ciboulette	
Samedi	
Steak de veau et pomme de terre sautée à l'huile et ciboulette	

PLAT DU JOUR 13,50 EUROS

à la carte : 15,00 €

MARIE

Poêlée fermière rôti à l'huile noisette

MERCREDI

Bouillabaisse de rosé et riz Pilaf

JELUCH

Steak de bœuf braisé au vin rouge

VENREDI

Filet d'ailette de bœuf et pomme de terre sautée au beurre et huile d'olive et ciboulette

Samedi

Steak de veau et pomme de terre sautée à l'huile et ciboulette

PLAT LYONNAIS 4€ ET 25 CL

à la carte : 10,00 €

MARIE

Steak Lyonnais Gribouille 75g et 80g

à la carte : 10,00 €

mousse Jambon Gribouille

à la carte : 10,00 €

LES EAUX MINÉRALES

à la carte : 1,00 €

PERIGUEUX

eaux minérales 0,5 des 25cl

à la carte : 1,00 €

BIÈRE PRESSION

à la carte : 2,00 €

HEINEKEN 0,0

HEINE

Les services Web

- Définition

Un service Web est un programme informatique inter-opérable permettant la communication et l'échange de données sans intervention humaine et en temps réel (Wikipedia). Basé sur des standards et protocoles ouverts (cf. XML)

- Dans la pratique

- Un service Web est une fonctionnalité (ou un ensemble de fonctionnalités)
- Mise à disposition via une interface (ressource) si possible unique
- Un service peut fournir :
 - du contenu (éventuellement dynamique) mis à disposition (ex: liste de produits)
 - une interface d'enregistrement (ajout, suppression, modification dans une base de données)
 - des fonctionnalités métier (ex: calculs de TVA par pays, des taxes d'une fiche de paye...)

- Les formes du service Web

- SOAP (WS-*), REST (Restful), appel RPC (version plus ancienne)

Précisions : Les services web sont le résultat de la mise à disposition d'un service développé par A sur Internet. Il est ainsi possible de les utiliser pour le développement d'un site. Ils permettent par exemple d'inclure un encart avec la météo sur sa page d'accueil sans avoir à le faire soi-même. Il faudra simplement intégrer ce service à sa page. Il est possible d'utiliser les services Web comme des briques de son site.

REST (*Representational State Transfer*) : Est une architecture logicielle, utilisée sur le Web pour décharger le serveur qui n'a pas besoin de gérer l'état des transactions (c'est le client qui sauvegarde toutes les variables utiles aux traitements métiers). Dans l'imagination des adeptes du Web agile, REST est une alternative à SOAP plus efficace, en fait les deux techniques ne sont pas incompatibles. On peut faire des échanges SOAP selon une architecture REST.

RPC (*Remote Procedure Call*) : c'est un appel de procédure distante = qui s'exécute dans un autre espace d'adresse (ordinateur local ou distant), c'est à dire qu'on demande (call) à un ordinateur distant (Remote) de faire une opération (procédure) pour nous et de nous renvoyer éventuellement son résultat (« tout c'est bien passé ». données en retour...).

Un bistrot sur le web?

- Le bar symbolise:
 - Réserve
 - les données, sgbdr, sensor, ...
 - Serveur du bar
 - comprendre une demande
 - Vérifier que le protocole & la demande sont bien formulés
 - Servir, traiter, assembler en fonction de la demande
 - Le client, nous humain,
 - machine consommant ce qui est servie en fonction du context
 - Quand à la carte
 - On parle ici de couche de **définition** de ce qui est **mis à disposition** ainsi que le **formalisme** pour le demander



REST

- REpresentational State Tranfert
 - Transfert de représentation d'état
 - CRUD
 - Permet l'échange de données
 - Couple ressource / uri
 - Appelé *endpoint*
 - S'appuie fortement sur le protocole HTTP
 - Transfert de ressource xml ou json (json est privilégié pour le js)



REST

- Accès CRUD
 - **Create**
 - **Read**
 - **Update**
 - **Delete**
- Accès à une ressource par HTTP
 - Méthodes HTTP
 - POST (**create**)
 - GET (**read**)
 - PUT (**update**)
 - DELETE (**delete**)
 - Capacités
 - Filtrage
 - champs
 - id
 - Positionnement
 - Relationnel *
 - Erreurs HTTP
 - 2xx OK
 - 4xx Erreur accès
 - 5xx Erreur Serveur



<https://firebase.google.com/docs/reference/rest/database>

<https://rnfirebase.io/> (implem react-native)

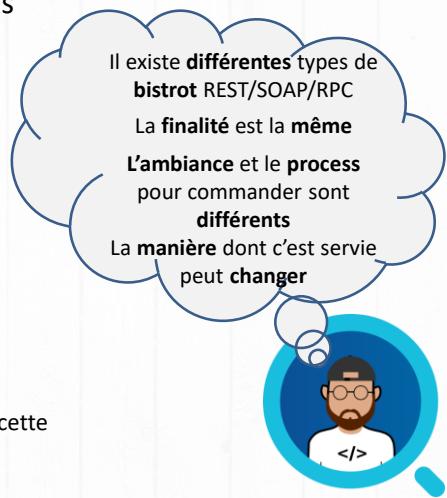
React-Native et localhost

- Le terminal physique possède sa propre adresse réseau
 - Accéder à un service réseau
 - Accéder en VM sur réseau host
 - Emulation wifi en NAT sur localhost de l'host
 - » 10.0.0.2 (gateway du réseau wifi)
 - Accéder en terminal physique a l'ip de dev
 - » Par wifi sur le même réseaux
 - Broadcast de ports
 - vers le "localhost" du terminal
 - » Command : abd reverse tcp:*portPC* tcp:*portTerminal*



Un bistrot sur le web !

- On parle **ici** de deux type de webservices
 - SOAP
 - REST
- La réserve
 - Votre partie applicatif qui construit les résultats
 - Accède au serveurs internes
 - Le moteur du web service
 - Effectue les contrôles
- Le serveur du bar
 - Le logiciel serveur qui reçoit les requêtes
 - Redirige les requêtes vers la réserve
- La carte
 - On parle **seulement** ici du **SOAP** qui permet cette logique de définition
 - WSDL (WebServiceDefinitionLayer)



Tout le reste n'est que JS

- React ne permet que la confection rapide de l'app
 - Assembler avec des component
- Qui s'occupera du **rest** ?
 - Les appels http
 - Le cycle de vie
 - Le composant est déjà monté
 - **componentDidMount**
 - Initial fetch for a component and son
 - await / async (es7/ts)
 - **async componentDidMount**

Est-ce que je lui dit
qu'il y a
fetch() & then()?



l'intégration de rest WS en js

- Pour faciliter l'accès aux données nous privilégierons le rest
 - Les échanges XML sont plus durs à mettre en œuvre et demande plus de ressources que je le JSON
- En js Historiquement les appelles http(sync et async) étaient fait par xhr (ou XMLHttpRequest)
 - **Xhr**
 - Async grâce à onreadystatechange
 - Prise en charge : fortement compatible
 - **Fetch** Une nouvelle approche est disponible dès es6
 - Approche async promise avec then()
 - Prise en charge : moins compatible



Examinons fetch

- Method GET par défaut
 - 1^{er} then : pour convertir un stream de réponse
→ réponse lisible
 - Différents formats
 - Conversion json ->objet
 - 2eme then : traitement après conversion
 - Un objet de configuration de la demande peut être fourni en 2eme arg de fetch
 - `fetch('https://www.reddit.com/r/javascript/.json', { method: 'post', body: JSON.stringify(opts) }).then(...).then(...)`

- Exemple GET :

```
const myImage =  
document.querySelector('img');  
fetch('flowers.jpg')  
.then(function(response) {  
    return response.blob();  
})  
.then(function(myBlob) {  
    const objectURL =  
URL.createObjectURL(myBlob);  
myImage.src = objectURL;  
});
```

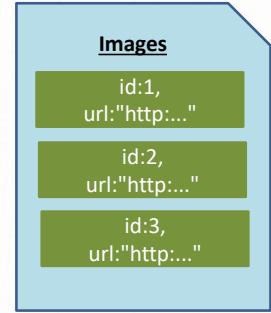


Utiliser fetch :

https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

TP rappels

- Exécuter un serveur json-serveur
 - Sur le flux en commentaire
 - <http://mon-json-server/images> et <http://mon-json-server/images/1>
- Faites un **objet** permettant,
 - d'effectuer des appels Rest pour
 - GET une ressource images sur le json-server
 - GET sur un ressource générique sur le json-server
 - Exécuter une call une fois la réponse reçue
 - Console log de l'objet reçue
 - Prévoir aussi les fonctions pour POST, PUT, PATCH, DELETE



Fichier json:

```
{  
    "images": [  
        {"id": 1, "url": "img/plonge.jpg"},  
        {"id": 2, "url": "img/kid1.jpg"},  
        {"id": 3, "url": "img/futurama1.jpg"},  
        {"id": 4, "url": "img/fry1.jpg"},  
        {"id": 5, "url": "url/farnsworth.jpg"},  
        {"id": 6, "url": "img/trololo.jpg"}  
],  
    "memes": [  
        {"id": 1, "image_id": 1, "text": [  
            {"  
                "x": 0,  

```

}

React-native-windows

- Compilation d'une app pour windows
 - <https://microsoft.github.io/react-native-windows/docs/getting-started>



Conclusion finale

- Remerciements et encouragements

Pendant cette formation, nous espérons :

- que vous avez appris de nombreuses choses théoriques et pratiques
- que vous y avez trouvé plus que ce que vous étiez venu chercher
- et que vous avez passé un bon moment avec votre professeur

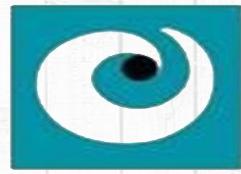
Merci d'avoir fait confiance à ORSYS, à bientôt pour d'autres formations !



Bibliographie

- Livres de référence
 - «Type script Notions fondamentales » édition eni
 - « Concevez des applications mobiles avec React Native » édition eyrolles
- Sites de référence
 - <http://www.w3.org/standards> (site du W3C -)
 - <http://www.w3schools.com/programming> (Web)
 - <http://www.oreillynet.com/développeurs> (forum de développeurs)
- <http://devdocs.io>





ORSYS
formation

A BIENTÔT...

TP



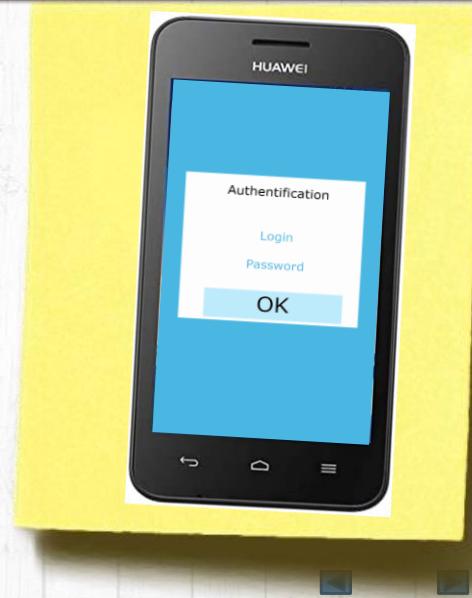
SplashScreen

- Page simple d'affichage
 - D'une image
 - D'un texte
- Contenu statique
- Aucune action



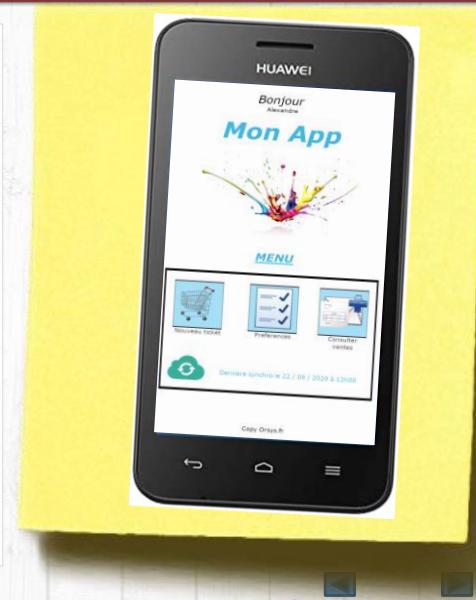
Login

- Composant *function*
 - Etat des input par hooks
- Page de formulaire
 - TextInput
 - Button
- Contenu statique
 - Styles des composants
- Actions
 - Debounce & setState des inputs
 - Alerter le parent une fois le boutton OK pressé



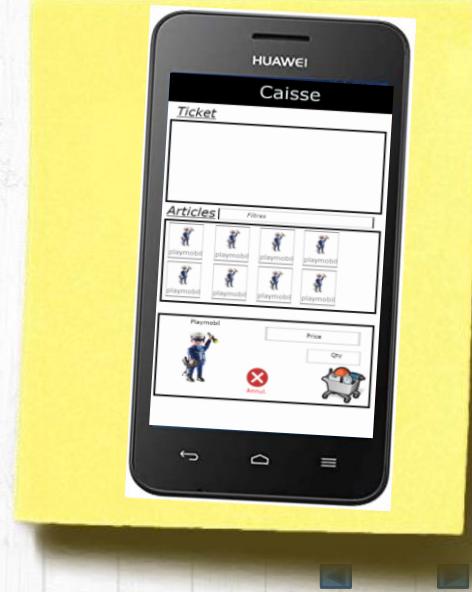
Home

- Composant Home de display de buttons
- Page de D'accès aux fenêtres
- Contenu statique
 - Styles des composants
 - Composant MyButton générique
- Contenu dynamique
 - Dernière date de synchro
- Actions
 - Gere le retour a la page home pour les enfants
 - Force la synchronisation



Caisse

- Composant Caisse
 - Composant ticket de caisse
 - Composant liste d'articles & article de liste
 - Composant d'éditeur de ligne
- Contenu dynamique
 - Ticket de caisse en cours
 - Liste d'article filtrés et sélectionnable
 - Affichage de l'article sélectionné
- Actions
 - Modif. des valeurs des inputs
 - Ajout au ticket en cours
 - Encaissement et enregistrement du ticket
 - Retour accueil



Option: sélection de ligne pour édition

Tickets de Ventes

- Composant Tickets
 - Composant ticket de caisse
 - Filtre de recherche
- Contenu dynamique
 - Ticket de caisse en cours sélectionné
 - Liste des tickets filtrés
- Actions
 - Sélection d'un ticket



Liste des actions nécessaires

- Login
 - Connexion
 - Contrôle et validation du login
 - » Navigation vers home et maintient du login
- Home
 - Navigation vers les pages
 - Synchronisation des sauvegardes locales
 - Déconnexion
 - Navigation vers LoginScreen



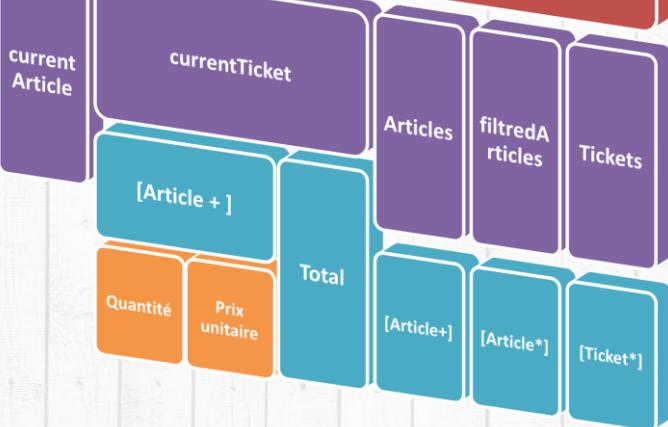
Liste des actions nécessaires

- Articles (liste filtrée)
 - Filtrer liste
 - Sélection article
- Caisse
 - Sélection article
 - Ajouter une vente
 - Désélection article
 - Supprimer article
 - Validation panier
 - Confirmation, puis navigation home
- Tickets
 - Filtrer liste de tickets
 - Vendeur / date
 - Sélection d'un ticket



Structure des données

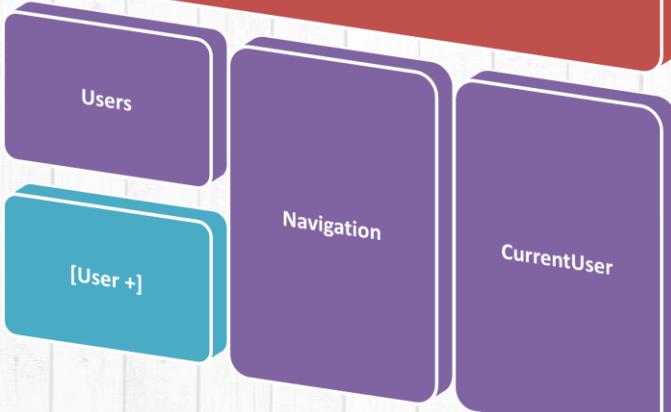
Store



- **CurrentArticle**
 - l'article sélectionné
- **CurrentTicket**
 - le ticket sélectionné ou en cours de vente
- **Articles**
 - Une liste global d'article
- **filteredArticles**
 - Liste d'articles filtrés
- **Tickets**
 - Liste de tous les ticket achevé

Structure des données

Store



- **CurrentArticle**
 - l'article sélectionné
- **CurrentTicket**
 - le ticket sélectionné ou en cours de vente
- **Articles**
 - Une liste global d'article
- **filteredArticles**
 - Liste d'articles filtrés
- **Tickets**
 - Liste de tous les ticket achevé

Structure des données

Article

titre

Description

BasePrix

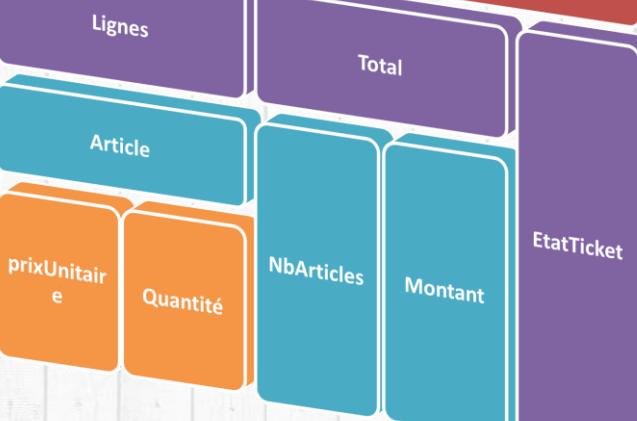
ImageSouc
re

stock

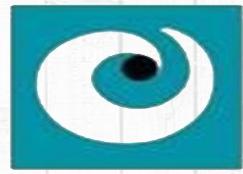
- Article
 - Titre : titre court du produit
 - Description : description longue, (facultatif)
 - BasePrix : prix de vente officiel
 - ImageSource : liens vers l'image
 - Stock : état actuel du stock, facultatif

Structure des données

Ticket

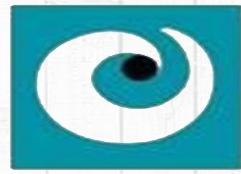


- Ticket
 - Articles : liste des produits
 - Article : Article standards
 - » Pour chaque ligne : article + complement
 - Total : section précalculé de valeurs
 - EtatTicket : etat de la vente
 - Encours
 - Achevé



ORSYS
formation





ORSYS
formation

