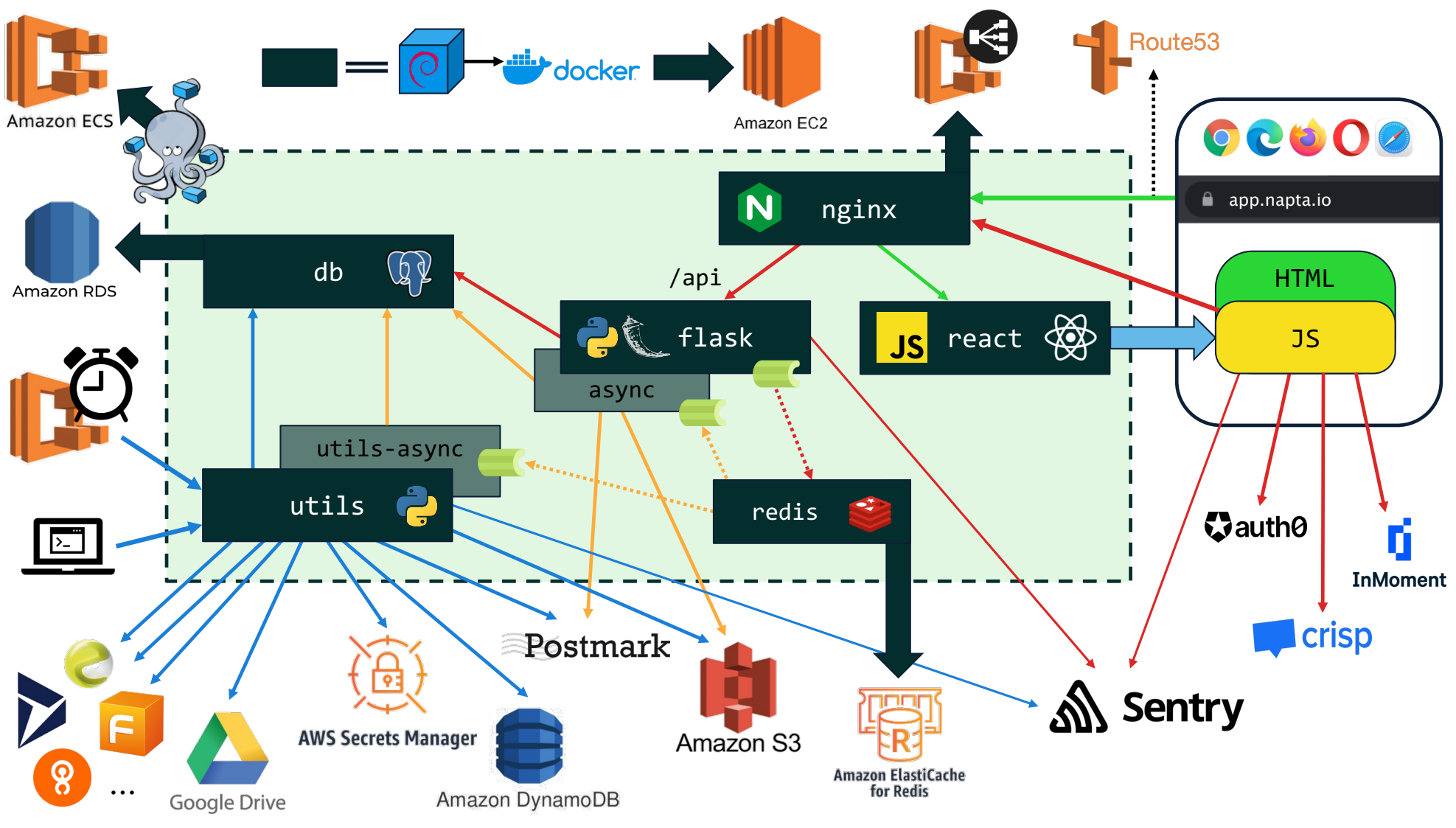
MobApp – Introduction à React Native

Fabien Coelho, Laurent Daverio, Olivier Hermant

Plan

1. JS : programmation asynchrone
2. React Native
3. JSX
4. Git
5. Hooks
6. Année prochaine
7. Travaux pratiques
8. Requêtes HTTP


Amazon ECS
docker
Amazon EC2
Route53
app.napta.io
nginx
db
Amazon RDS
/api
flask
async
react
HTML
JS
utils-async
utils
redis
auth0
InMoment
crisp
Postmark
AWS Secrets Manager
Amazon S3
Amazon ElastiCache for Redis
Sentry
Google Drive
Amazon DynamoDB

1. JS : programmation asynchrone

Manipuler les fonctions

- fonction = citoyen de 1^{e} classe = variable comme une autre
  - comme un entier, un tableau, une chaîne de caractères, etc

Manipuler les fonctions

- fonction = citoyen de 1^{e} classe = variable comme une autre
  - comme un entier, un tableau, une chaîne de caractères, etc
- **Question**. Qu'est-ce qui est une fonction ?

1. `alert`
2. `alert('Hello, World !')`
3. $\sin(\pi)$
4. sin
5. $x \mapsto \sin(x)$
6. `(x) => { alert(x) }`

Manipuler les fonctions

- fonction = citoyen de 1^{e} classe = variable comme une autre
  - comme un entier, un tableau, une chaîne de caractères, etc
- **Question**. Qu'est-ce qui est une fonction ?

1. `alert`
2.
3.
4. sin
5. $x \mapsto \sin(x)$
6. `(x) => { alert(x) }`

Manipuler les fonctions

- fonction = citoyen de 1^{e} classe = variable comme une autre
  - comme un entier, un tableau, une chaîne de caractères, etc
- **Question**. Qu'est-ce qui est une fonction ?

1.	`alert`	2.	
3.		4.	sin
5.	$x \mapsto \sin(x)$	6.	`(x) => { alert(x) }`

- **fonction anonyme**
  - deux syntaxes

```
function (x,y) { /* du code */ }
   (x,y) => { /* du code */ }
```

  - et, sans les accolades

```
                (x,y) => 3
équivaut à      (x,y) => {return 3}
```

- exemples :

```
const f = function (x,y) { /* du code */ }
const f = (x,y,z) => { /* de l'autre code */ }
```

JS et asynchrone

- `window.onload = f` : exécute**ra** (futur) `f` à la fin du chargement de la page
  - crucial si `f` doit accéder à des éléments de la page
  - `f` est une fonction acceptant **UN** argument
- `window.onlad = f(arg)` est "évidemment faux" : pourquoi ?
- [page_dyn.html] du cours html/css/js

JS et asynchrone

- `window.onload = f` : exécute**ra** (futur) `f` à la fin du chargement de la page
  - crucial si `f` doit accéder à des éléments de la page
  - `f` est une fonction acceptant **UN** argument
- `window.onlad = f(arg)` est "évidemment faux" : pourquoi ?
- [page_dyn.html] du cours html/css/js
- questions ?

2. React Native

Pourquoi React Native ?

- Cahier des charges : dev **mobile**, front-end 2025++

Pourquoi React Native ?

- Cahier des charges : dev **mobile**, front-end 2025++
- choix "naturels" : iOS $\Rightarrow$ Swift, Android $\Rightarrow$ Kotlin, Java.
  - [-] spécifique à **une** plateforme
  - [-] développement plus complexe
  - [+] contrôle total

Pourquoi React Native ?

- Cahier des charges : dev **mobile**, front-end 2025++
- choix "naturels" : iOS ⇒ Swift, Android ⇒ Kotlin, Java.
  - [-] spécifique à **une** plateforme
  - [-] développement plus complexe
  - [+] contrôle total
- React Native
  - [+-] modèle de programmation = logique & style
  - [+] plus facile d'accès
  - [+] programmation Web react similaire
  - [-] liberté, contrôle
  - [-] couches empilées ⇒ +ressources (cf. VM, vrai autres choix)
  - [+] bien sur un CV (césure)
  - [+-] *à la mode*
  - [+] support, documentation, etc.

Pourquoi React Native ?

- Cahier des charges : dev **mobile**, front-end 2025++
- choix "naturels" : iOS ⇒ Swift, Android ⇒ Kotlin, Java.
  - [−] spécifique à **une** plateforme
  - [−] développement plus complexe
  - [+] contrôle total
- React Native
  - [+−] modèle de programmation = logique & style
  - [+] plus facile d'accès
  - [+] programmation Web react similaire
  - [−] liberté, contrôle
  - [−] couches empilées ⇒ +ressources (cf. VM, vrai autres choix)
  - [+] bien sur un CV (césure)
  - [+−] *à la mode*
  - [+] support, documentation, etc.
- le front-end évolue **très** rapidement. Concurrence mobile ?
  - Flutter, Ionic, votre techno préférée
  - ce qui est déjà obsolète ou que nous ne connaissons pas

3. JSX

JSX : JavaScript and XML

- tout est dans le titre : javascript, balises XML au milieu
- JS : pour la logique
- balises XML : pas ~~du HTML~~ (mais y ressemble)
  - composants React Native
  - servent au rendu
  - liées aux composants graphiques et *natifs* de l'appareil mobile
- possibilité de créer ses **propres balises**
  - en développant ses **propres fonctions**
  - qui doivent retourner un rendu

Premier projet en React-Native

- stack technique (*pour Android*) : installée TP prise en main

Premier projet en React-Native

- stack technique (*pour Android*) : installée TP prise en main
- créer un projet, git clone proprement : cf. TP
- Deux styles de programmation en React Native
  - ~~avec des classes~~ ($<$ 2019)
  - avec des fonctions (notre choix)

Premier projet en React-Native

- stack technique (*pour Android*) : installée TP prise en main
- créer un projet, git clone proprement : cf. TP
- Deux styles de programmation en React Native
  - ~~avec des classes~~ ($<$ 2019)
  - avec des fonctions (notre choix)
- demo time !

```
git clone https://gitlab.cri.minesparis.psl.eu/hermant/kivappa
```

npm

- `Node.js` : plateforme logicielle en JS
- `npm` : gestionnaire de paquets (**n**ode **p**ackage **m***anager*)
  - installation de librairies/paquets développés pour `Node.js`
  - et les paquets requis par ces paquets... (*dépendances*).
  - ces dépendances sont (automatiquement) listées dans les fichiers `package.json` et `package-lock.json`
  - ne pas éditer ces fichiers à la main
- `npm` servira aussi à lancer le (packager de) projet

Hello World

Mode d'emploi :

1. fork, cloner et installer son projet (ou le créer)
2. connecter la tablette
3. démarrer le packager de React Native (`npm start`)
4. construire le projet et le lancer (taper "a")

Tout se passe dans le fichier `App.js` :

- `imports`
- `export` du composant principal de la page
- `KivAppA` : la fonction de rendu
  - retourne les composants RN qui seront affichés
  - balises XML, avec propriétés
- tag [2024-01-hello]

Programmer son propre composant

- écrire une fonction `Hello` qui retourne le composant `<Text>`
- appel de `Hello` par le composant homonyme `<Hello>`
- passage d'argument (le texte à afficher) à `Hello` comme attribut de la balise XML
- `Hello` possède **un unique** paramètre, `props`
- objet qui contient tous les paramètres
- tag [2024-03]

NB :

- entre { *accolades* } , le code JS,
- entre { *accolades* } *dans du code JS*, un objet "en ligne",
- conséquence : parfois deux paires d'accolades

4. Git

Git et GitLab

User Story

En tant que développeur, je travaille sur un projet RN, et je le partage avec mon groupe (et les profs).

- gitlab.cri.minesparis.psl.eu
- contiendra votre projet
- utilise le système de versionnement git
  - principe local–distant
- illustration :

  https://illustrated-git.readthedocs.io/en/latest/#working-with-remote-repositories

Git aujourd'hui

- un "repository" pour votre TP front-end, cloné d'un point de départ fourni par nous
- vous êtes seul contributeur (plus simple)
- processus et commandes :
  1. développer une fonctionnalité
  2. `git add`, puis `git commit` (local), puis `git push` ($\rightarrow$ distant)
- possibilité de collaborer avec soi-même

User Story 2

En tant que développeur, je développe sur 3 ordinateurs différents.

- `git pull` pour màj du repository local par rapport au distant.

5. Hooks

Ajouter un bouton : le hook d'état

User Story 3

En tant qu'utilisateur, je clique, une variable s'incrémente et s'affiche.

- composant `<Button>`, attribut RN `onPress`
- tag [2024-05]
- solution "naïve" (variable + incrément `onPress`) échoue
- `onPress` demande un `callback` (fonction asynchrone)
  - solution semi-naïve échoue : modifications ignorées par le modèle RN

Ajouter un bouton : le hook d'état

User Story 3

En tant qu'utilisateur, je clique, une variable s'incrémente et s'affiche.

- composant `<Button>`, attribut RN `onPress`
- tag [2024-05]
- solution "naïve" (variable + incrément `onPress`) échoue
- `onPress` demande un `callback` (fonction asynchrone)
  - solution semi-naïve échoue : modifications ignorées par le modèle RN
- utiliser un hook (≈ feature RN) d'état
- changer la valeur avec `setCompteur` : re-rendu automatisé
  - merci le framework RN (cf. année prochaine) et les hook d'état

Ajouter un bouton : le hook d'état

User Story 3

En tant qu'utilisateur, je clique, une variable s'incrémente et s'affiche.

- composant `<Button>`, attribut RN `onPress`
- tag [2024-05]
- solution "naïve" (variable + incrément `onPress`) échoue
- `onPress` demande un `callback` (fonction asynchrone)
  - solution semi-naïve échoue : modifications ignorées par le modèle RN
- utiliser un hook ($\approx$ feature RN) d'état
- changer la valeur avec `setCompteur` : re-rendu automatisé
  - merci le framework RN (cf. année prochaine) et les hook d'état
- on peut passer les fonctions de modification en tant qu'attribut/propriété : les fonctions sont des citoyens de 1^{e} classe (tag [2023-04])

6. Année prochaine

Cours Front-End en 2025

- 06/01/2025 et 10/01/2025 : récapitulatif RN, requêtes HTTP avec axios, gestion de la disposition (formulaires propres), hooks d'état et effets
- 15/01/2025
  - modèle RN expliqué,
  - authentification/login.
- autres recettes : à la demande.

Cours Front-End en 2025

- 06/01/2025 et 10/01/2025 : récapitulatif RN, requêtes HTTP avec axios, gestion de la disposition (formulaires propres), hooks d'état et effets
- 15/01/2025
  - modèle RN expliqué,
  - authentification/login.
- autres recettes : à la demande.
- conséquence : le TP et RN seront légèrement "magiques" aujourd'hui

7. Travaux pratiques

8. Requêtes HTTP

Requêtes en React Native

- utiliser `fetch` ou `axios`
- états pour passer les informations pertinentes
- callbacks pour la mise à jour
- et des jolis composants RN
- DEMO ([branche `devel`])
- revue de code