MobApp – (Ré)introduction à React Native

Fabien Coelho, Laurent Daverio, Olivier Hermant

MINES PARIS | PSL | Centre de recherche en informatique

Plan

1. JSX
2. React Native
3. Placement des composants
4. Fonctions
5. Hooks
6. Requêtes à un serveur

1. JSX

JSX : JavaScript and XML

- 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 composants *natifs*) de l'appareil mobile
- créer ses **propres balises**
  - = développer ses **propres fonctions**
  - doivent retourner un rendu

2. React Native

L'app KivAppA

Mode d'emploi (cf. premier TP)

1. prendre son projet (`git`)
2. connecter la tablette/un téléphone android
3. démarrer le packager de React Native (`npm start`)
4. construire le projet et le lancer (taper "a")

Dans le fichier `App.js`:

- `imports`
- `export` du composant principal de la page,
- `KivAppA` : la **fonction de rendu**
  - retourne les **composants RN** qui doivent être affichés
  - balises XML, avec propriétés
- [DEMO]

Développer son composant

- donner un nom : `<Decembre>`
- l'appeler : RN s'occupe de tout, **sauf** de
- développer de la fonction de rendu éponyme :

```
const Decembre = (props) => {
  /* code : logique du composant Decembre */
  return (
    /* composants : rendu de Decembre */
  )
}
```

- **un seul argument** : `props`
  - objet avec des champs
  - correspondance avec les **attributs de la balise XML**
- NB :
  - entre { `accolades` } , le code JS,
  - entre { `accolades` } , dans du code JS, un objet en ligne,
  - conséquence : parfois deux paires d'accolades

3. Placement des composants

Placement des composants : flexboxes

- composant parent P (conteneur) avec style `flex : n`
- composant parent P avec style `flexDirection : "column"` (ou `"row"`, ou...)
- composants enfants $F_1, \cdots, F_k$: avec style `flex : n_i`
- place prise par F_i dans P :

$$\frac{n_i}{\sum_{j=1}^{k} n_j}$$

- `<Button>` à remplacer par `<Pressable>`

4. Fonctions

Les fonctions

- fonction = citoyen de 1^{e} classe = valeur comme une autre

```
(x,y) => { return x*y }
```

- fonction nommée ≡ affecter une fonction anonyme à une variable

```
const mult = (x,y) => { return x*y }
```

- on peut passer une fonction (anonyme ou non) en argument. Exemples:

```
<Button onPress={(evt) => {/* ... */}}
<DemiJournee choixMois={mois} changeMois={setMois}/>
```

5. Hooks

Ajouter un état à un composant : `useState`

Chaque composant `<Decembre>` a un compteur qui lui est propre.

- **hook** ($\approx$ fonctionnalité) particulier, le hook d'état
- ajoute **un état**

```
const [compteur, setCompteur] = useState(0)
```

  - accès en lecture par la variable `compteur`, valeur initiale `0`
  - accès en écriture par la fonction `setCompteur`
- différents `<Decembre>` : états indépendants 
- variables et fonctions d'état dans les `props`
  - partage (lecture et/ou écriture) avec sous-composants
  - **technique très utile : remonter l'état**
- exemple :
  - `<DemiJournee>` a accès en lecture/écriture à l'état "mois"
  - par `props.choixMois` et `props.changeMois`

6. Requêtes à un serveur

Requêtes en JS

- utilisation de `axios` (cf. TP HTML/JS)

  `axios({ /* paramètres dans un objet */ })`
- objet de paramètres

  `{method : ..., baseURL : ..., ...}`
  - `method` : la méthode HTTP (`"GET"`, `"POST"`, `"DELETE"`...)
  - `baseURL` : l'addresse du serveur (e.g. `"https://kiva.mobapp.minesparis.psl.eu/api"`)
  - `url` : la route de l'API (e.g. `"/store"`)
  - `headers` : un objet contenant les informations d'en-tête de la requête HTTP, en particulier l'authentification :

    `{Authorization : 'Basic ' + ...}`
  - paramètres supplémentaires (si besoin) :
    - `params` : dictionnaire de paramètres (GET)
    - `data` : idem (mais POST, PUT, PATCH...)
- requête $\Rightarrow$ attente réponse $\Rightarrow$ asynchrone $\Rightarrow$ callbacks
- `axios` retourne un objet : une promesse

Les promesses

- type d'objet "asynchrone"
- un jour,
  - soit est **tenue**,
  - soit est **rompue**.
- **à ce moment**, callback appelé
- **il faut enregistrer cette fonction** !
  - `.then` enregistre le callback de succès
  - `.catch` enregistre le callback d'erreur

Promesse tenue

- soit p la promesse créée et renvoyée par `axios`
- p appelle la fonction de callback enregistrée par `p.then((response) => { /* ... */ })`
- p fournit un objet de réponse
- dont les champs de contiennent :
  - `response.status` : code de retour (200, 201...)
  - `response.data` : contenu renvoyé par le serveur (déjà formatté = dé-JSON-ifié = sous forme d'objet)
  - ainsi que : en-tête HTTP de la réponse, etc (cf. doc axios)

Promesse rompue

- *p* crée un objet d'erreur
- appel le callback gestionnaire qui a été enregistré par `p.catch((error) => { /* ... */ })`
- on peut inspecter `error` dans le callback
- enregistrement **en dernier** dans la chaîne
- NB :
  - requête envoyée correctement, mais refusée par le serveur (400,401,404,...) = promesse tenue
  - présentation sciemment simplifiée

Que faire avec le contenu de la réponse

- bonne question !
- le plus souple : avoir un état
  - callback attaché à la requête : màj de l'état
  - état transmis (en lecture) à qui en a besoin

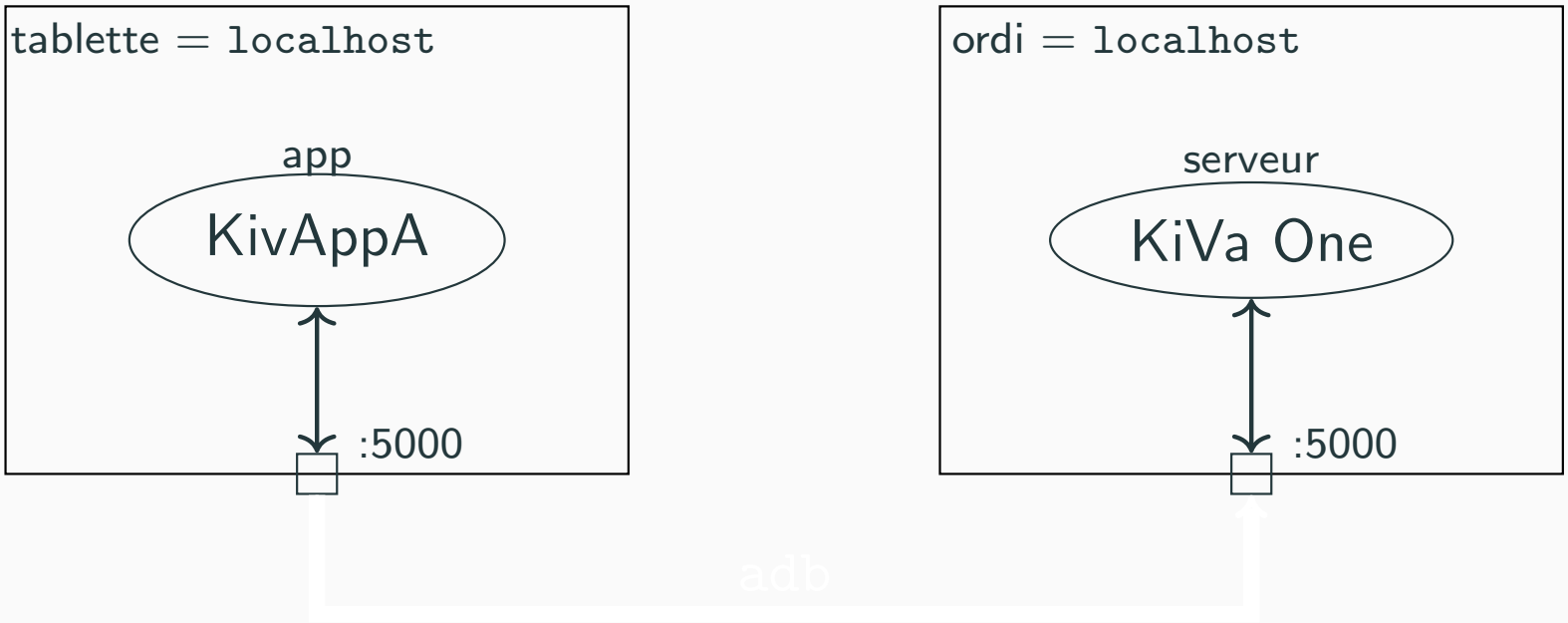
Démonstration

Analyse du point de départ du TP React Native.

- Composant `Decembre` :
  - fabriqué par nous
  - état
  - on peut l'utiliser plusieurs fois si on veut 
- Composant `DemiJournee` :
  - transmission de l'état du parent `KivAppA`
  - en lecture seule (pour l'instant)
- Composant `Janvier` :
  - fichier séparé
  - requête Axios avec `.then` et `.catch`
  - état `list` stocke la réponse
  - transmis en lecture seule (`props.display` et `<KeyValues display={list}>`)

Requête à un serveur "local"

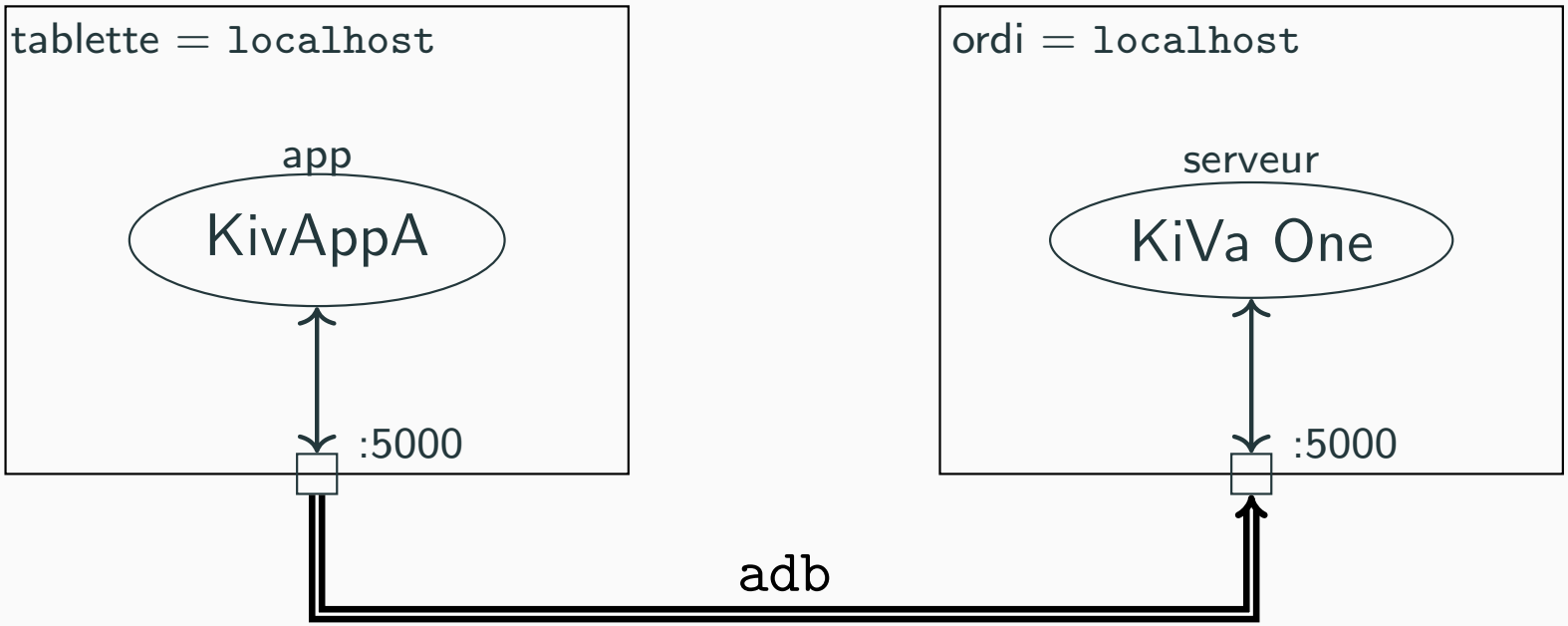
1. lancer le serveur local
2. **reverse port forwarding**



- tablette $\xrightarrow{\text{requête}}$ VM

Requête à un serveur "local"

1. lancer le serveur local
2. **reverse port forwarding**



- tablette $\xrightarrow{\text{requête}}$ VM
- demander à `adb` de faire le job

```
adb reverse tcp:5000 tcp:5000
```

3. configurer correctement l'URL dans `axios` et la lancer