MobApp — React Native Avancé

Fabien Coelho, Laurent Daverio, Olivier Hermant

Plan

Node, NPM

Architecture d'un Application React Native

Cycle de vie d'un Composant React Native

Quand récupérer des données ?

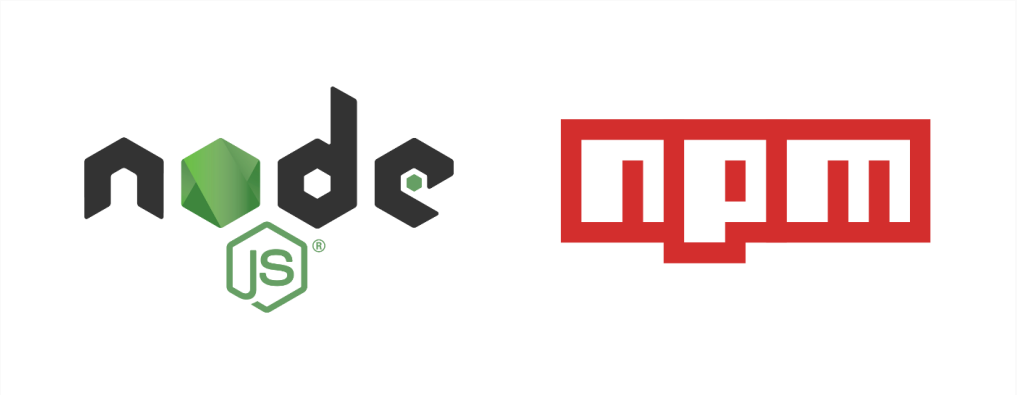
Quelques recettes JS(X)

TP : authentification

Node, NPM

NPM, Node et l'écosystème Javascript

- Javascript est un langage interprété par les navigateurs
- Node permet d'exécuter du javascript hors des navigateurs
- une application ou librairies node.js s'appelle un paquet (package)



Quizz Node.js

1. qui peut interagir avec le système de fichier ?
    - Node.js
    - un navigateur
    - les deux
2. qui peut faire office de serveur (backend) ?
    - Node.js
    - un navigateur
    - les deux
3. qui peut faire office de client (frontend) ?
    - Node.js
    - un navigateur
    - les deux

Quizz Node.js

1. qui peut interagir avec le système de fichier ?
    - Node.js : à travers l'API FileSystem
    - un navigateur : l'API LocalFileSystem offre un accès limité aux données locales (cache...)
    - **les deux**
2. qui peut faire office de serveur (backend) ?
    - **Node.js** : Express est un serveur Node.js qui ressemble à Flask
    - un navigateur : Non
    - les deux
3. qui peut faire office de client (frontend) ?
    - Node.js : oui
    - un navigateur : oui
    - **les deux**

Npm, npx, et react

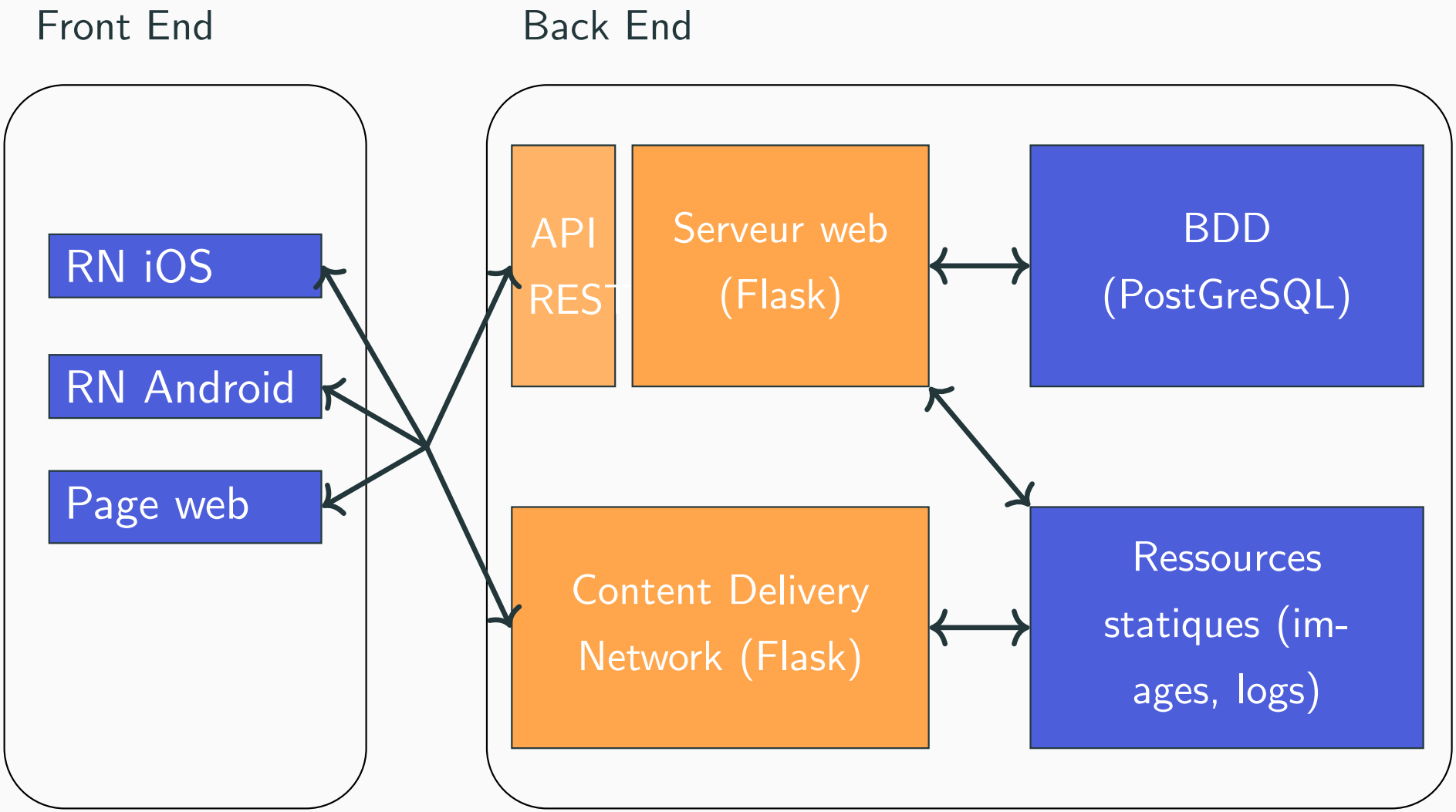
- npm est un gestionnaire de paquets (package manager)
  - similaire à `pip install` (python) ou `apt` (linux)
  - `npm install X` : installe le paquet `X` (màj de `package.json`)
  - `npm install` : installe les paquets listés dans `package.json`
  - autre gestionnaire pour Node.js: yarn
- npm permet de configurer des commandes (`package.json`)

```
"name": "KivAppA",
"version": "0.0.1",
"scripts": {
  "android": "react-native run-android",
  "ios": "react-native run-ios",
  "lint": "eslint .",
  "start": "react-native start",
  "test": "jest"
}
```

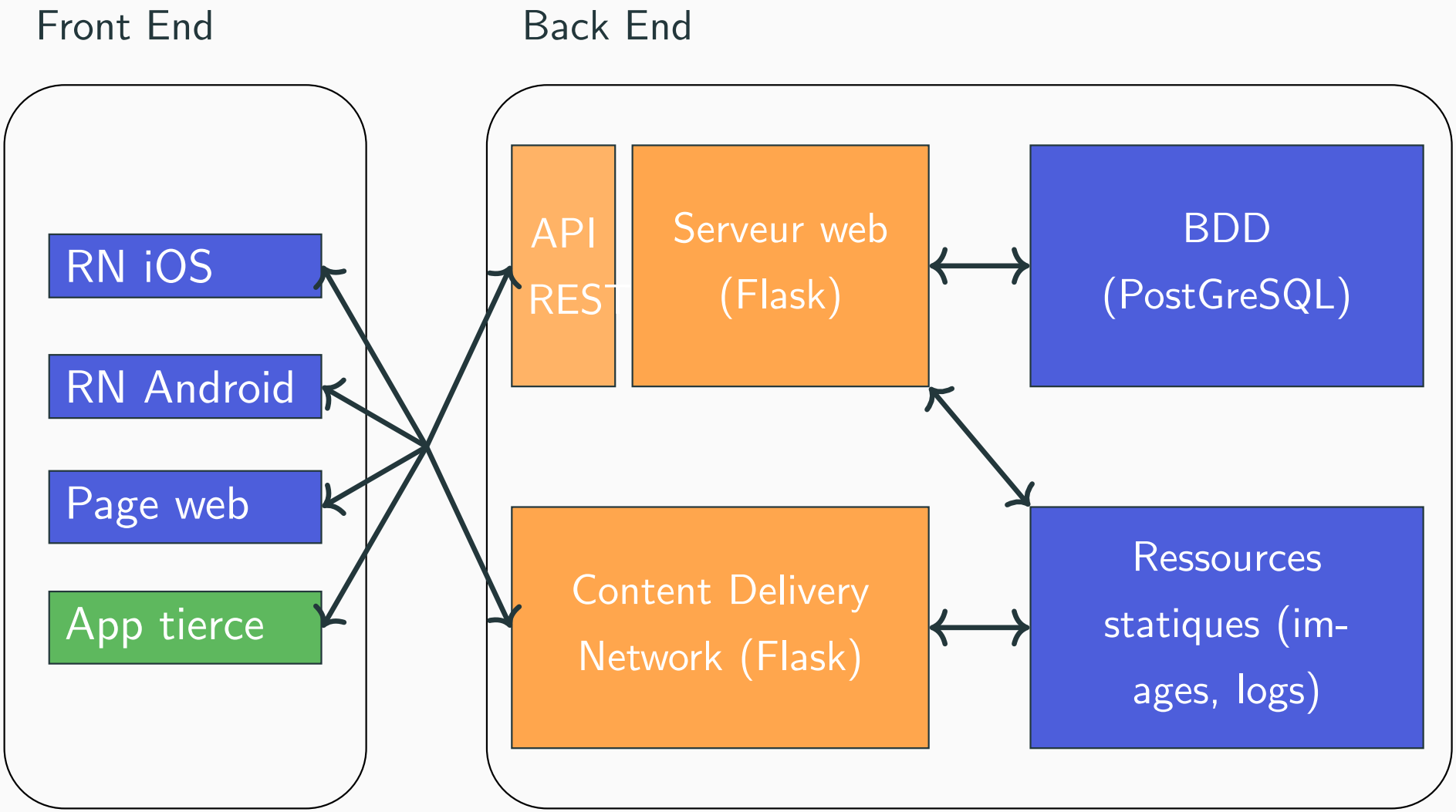
  - `npm run start` exécutera `npx react-native start`
  - raccourci `npm start`
  - `npx` : exécute des commandes

Architecture d'un Application React Native

Architecture typique d'une application



Architecture typique d'une application



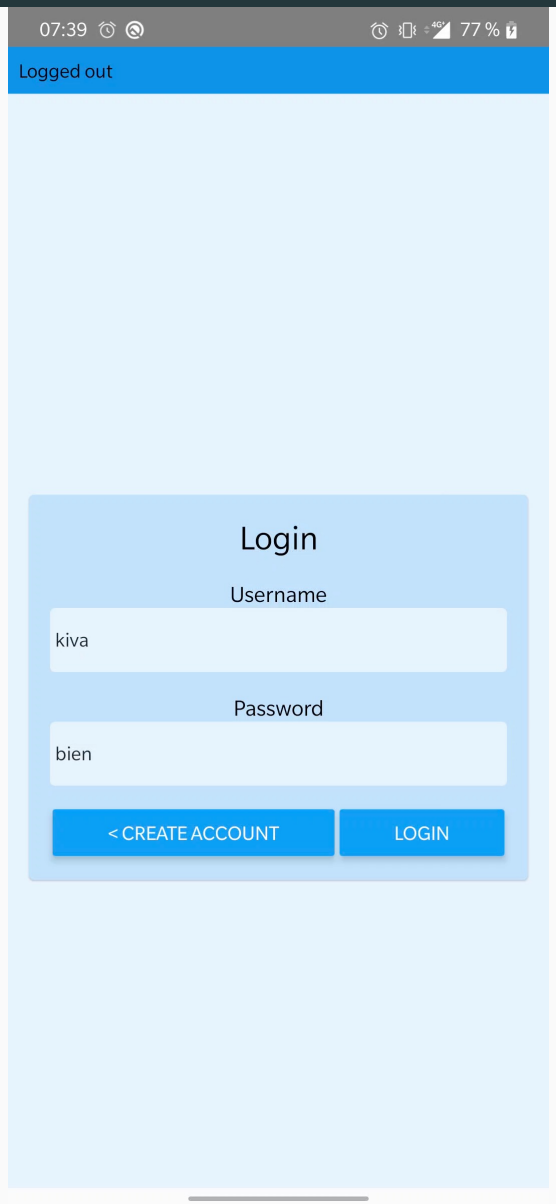
Où se situe la logique applicative ?

- Front end (React Native) :
  - page de login
  - autres actions possibles
- Back end (Flask) :
  - login/mdp : correct ?
  - l'utilisateur a-t-il le droit ?

Où se situe la logique applicative ?

- Front end (React Native) :
  - page de login
  - autres actions possibles
- Back end (Flask) :
  - login/mdp : correct ?
  - l'utilisateur a-t-il le droit ?
- conclusion :
  - logique de validation sur le serveur : obligatoire
  - logique applicative souvent dupliquée

Structurer une app React Native



App.jsx

Root.jsx

Header.jsx

LoggedOutView.jsx

Loggin.jsx

KivTextInput.jsx

KivTextInput.jsx

Bouton

Bouton

Structure des répertoires

- `Root.jsx` : pour le routage logué/délogué, stockage identifiants
- `loggedOut/` :
  - `LoggedOutView.jsx` : routage login/création
  - `Login.jsx`
  - `CreateAccount.jsx`
- `common/`
  - `KivTextInput.jsx` : composant pour l'input de texte

Structurer les composant React Native

```
import React, { useState } from 'react';
import { TextInput, View, Text } from 'react-native';

export default function MyAwesomeTextInput(props) {
    const [value, setValue] = useState('initial value');

    return (
        <TextInput
            onChangeText={text => {
                setValue(text);
                props.onChangeText(text);
            }}
            value={value} />);
}
```

- Usra entre son nom : `usra`

RN appel du callback avec `text='usra'`

- `setValue(text)` : la màj de `value` à `usra` se fera **à la prochaine mise à jour du composant**
- de plus, appel de la fonction définie dans le contexte “props” de `MyAwesomeTextInput`

Structurer les composant React Native

```jsx
import React, { useState } from 'react';
import { TextInput, View, Text } from 'react-native';

export default function MyAwesomeTextInput(props) {
    const [value, setValue] = useState('initial value')

    return (
        <TextInput
            onChangeText={text => {
                setValue(text);
                props.onChangeText(text);
            }}
            value={value} />);
}
```

```jsx
import React, { useState } from 'react';
import { TextInput, View, Text } from 'react-native';

export default function MyBADTextInput(props) {
    const [value, setValue] = useState('initial value');

    return (
        <TextInput
            onChangeText={text => {
                setValue(text);
    // Here Value is the old value. It has not been updated
    // yet to be `text`. It will only be update the next
    // time React Framework updates MyBADTextInput component
                props.onChangeText(value);
            }}
            value={value} />);
}
```

- attention à l'utilisation de `value/setValue`

Structurer les composant React Native

```
1  import React, { useState } from 'react';
2  import { TextInput, View, Text } from 'react-native';
3  import MyAwesomeTextInput from 'MyAwesomeTextInput.react';
4
5  /**
6   * @param props: {onConnect(login, password)}
7   */
8  export default function MyLoginPassword(props) {
9      const [login, setLogin] = useState();
10     const [password, setPassword] = useState();
11
12     return (
13         <View>
14             <Text value="Login" />
15             <MyAwesomeTextInput
16                 onChangeText={text => setLogin(text)}/>
17             <Text value="Password" />
18             <MyAwesomeTextInput
19                 onChangeText={text => setPassword(text)} />
20             <Button
21                 onPress={() => props.onConnect(login, password)}
22                 title="Connect" />
23         </View>);
24 }
```

"usra renseigne son login/mdp".

Exigences

- 2 `KivTextInput`
- 1 Bouton
- 1 appel de la fonction props `onConnect` au clic

Que fait `onConnect` :

1. requête à `/api/login`
2. récupérer la réponse et, soit
   - stocker localement le token
   - afficher un message "mauvais mdp"

Composer les composants

Root.jsx

Props :

States :

- connectionToken

Composants :

- Login
- MainPage

Login.jsx

Props :

- onConnect

States :

- login
- password

Composants :

- 2 KivTextInput
- Bouton

KivTextInput.jsx

Props :

- onChangeText

States :

- value

Composants :

- TextInput

Composer les composants

Root.jsx

Props :

States :

- connectionToken

Composants :

- Login
- MainPage

Login.jsx

Props :

- onConnect

States :

- login
- password

Composants :

- 2 KivTextInput
- Bouton

KivTextInput.j

Props :

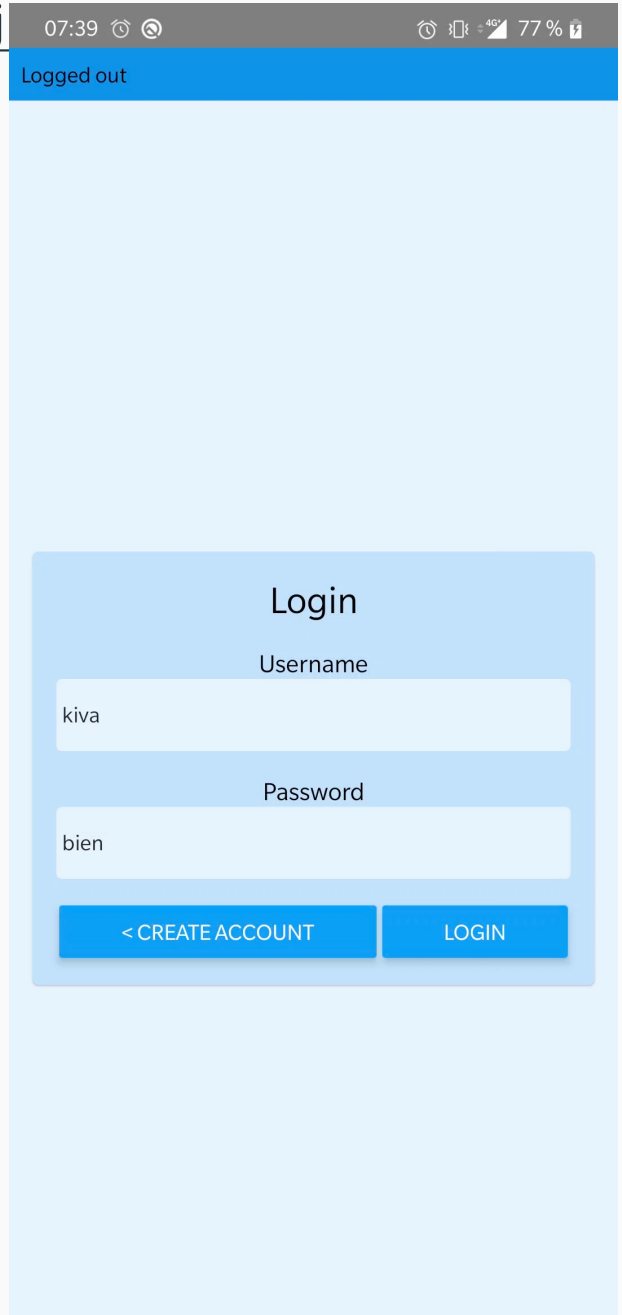
- onChangeText

States :

- value

Composants :

- TextInput

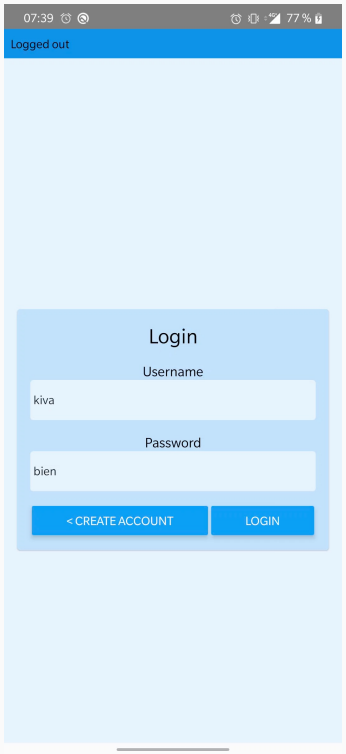


Gestion des messages d'erreur

```
1  import React, {useState} from 'react';
2  import {TextInput, View, Text} from 'react-native';
3
4  export default function UselessTextInput() {
5      const [value, setValue] = useState('initial value');
6
7      const isCorrect = value == 'correct value';
8
9      return (
10         <View>
11             {!isCorrect && <Text>Incorrect value!</Text>}
12             <TextInput
13                 onChangeText = {text => setValue(text)}
14                 value={value} />
15         </View>);
16 }
```

- `isCorrect` pour l'affichage conditionnel
- utile pour valider des données ou effectuer une action spécifique

Changer de vue



App.jsx

Root.jsx

Header.jsx

LoggedOutView.jsx

Loggin.jsx

KivTextInput.jsx

KivTextInput.jsx

Bouton

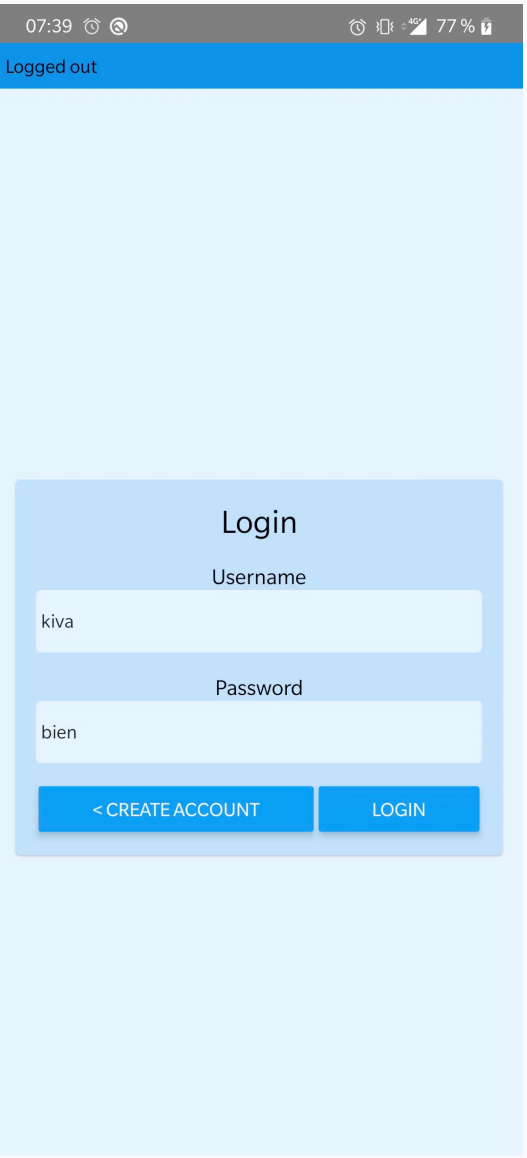
Bouton

```
[tab, setTab] = useState('login')

return (tab == 'login' ?
        <Login /> :
        <CreateAccount />)

if (tab == 'login') {
    return <Login />;
} else {
    return <CreateAccount />
}
```

Changer de vue



`setTab('login')`

`setTab('create')`

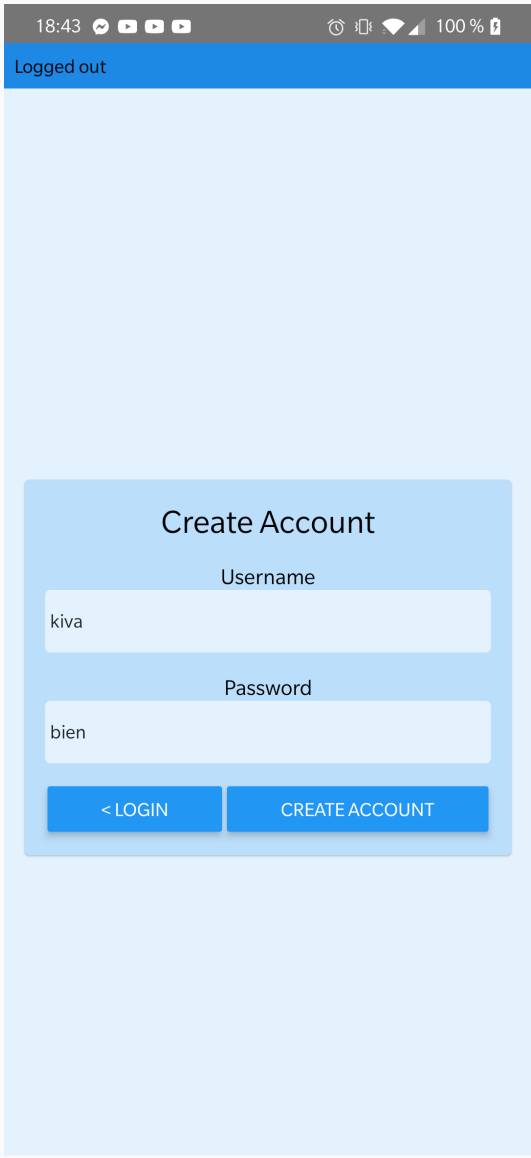
Appel à `setTab` via les `props`

- Parent :

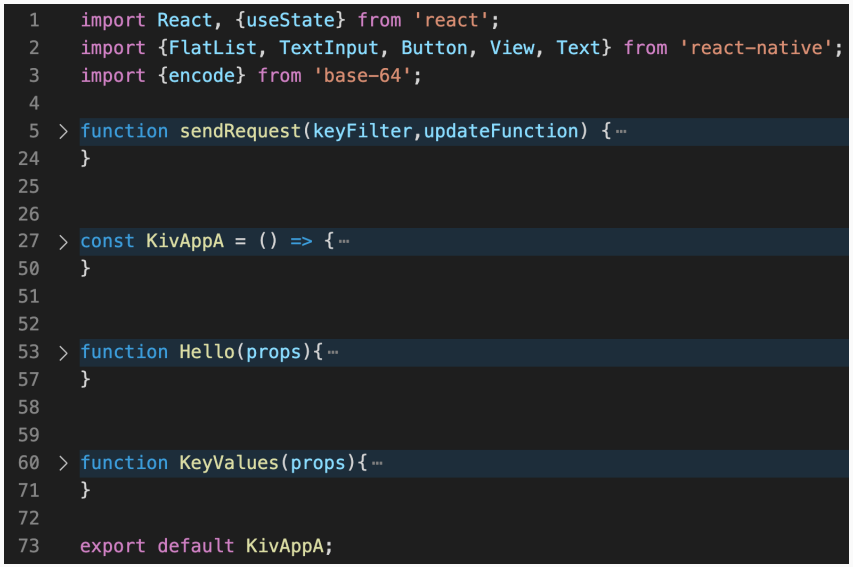
```
<Login onCreate={() =>
setTab('create')} />
```

- Enfant `Login.jsx` :

```
<Button onPress={props.onCreate}
/>
```



Diviser les fichiers



```
1   import React, {useState} from 'react';
2   import {FlatList, TextInput, Button, View, Text} from 'react-native';
3   import {encode} from 'base-64';
4
5   function sendRequest(keyFilter,updateFunction) {…
24  }
25
26
27  const KivAppA = () => {…
50  }
51
52
53  function Hello(props){…
57  }
58
59
60  function KeyValues(props){…
71  }
72
73  export default KivAppA;
```

Que séparer ?

- gros fichiers (¿ 100 loc)
- fonctions souvent utilisées (`sendRequest`)
- composants souvent utilisés (`KivTextInput`)

À l'extrême :

- 1 unique composant RN par fichier
- 1 unique fonction par fichier

Diviser les fichiers

```
1   import React, {useState} from 'react';
2   import {FlatList, TextInput, Button, View, Text} from 'react-native';
3
4   import {sendRequest} from 'sendRequest';
5
6 > const KivAppA = () => {…
29  }
30
31
32 > function Hello(props){…
36  }
37
38
39 > function KeyValues(props){…
50  }
51
52  export default KivAppA;
```

```
1   import {encode} from 'base-64';
2
3   export default function sendRequest(keyFilter,updateFunction) {
4     const route = '/store'
5     const url = new URL(route, 'https://kiva.mobapp.mines-paristech.fr/api')
6     url.searchParams.append('filter', keyFilter)
7     const hd = new Headers({'Authorization' : 'Basic ' + encode('kiva:bien') })
8     console.log('Requesting ' + url.toString())
9     fetch(url, {
10      method : 'GET',
11      headers : hd
12    }
13    ).then((response) => {
14      if (response.status == '200') {
15        return(response.json())
16      } else {
17        alert('Response code :' + response.status)
18      }
19    }).then(updateFunction).catch(
20      (e) => {alert('Something went wrong ' + e.message)}
21    )
22  }
```

Et il y a une erreur ici !

Organisation des répertoires

```
common/
    sendRequest.js
    consts.js
    KivTextInput.jsx
loggedOut/
    Login.jsx
    CreateUser.jsx
loggedIn/
    MainPage.jsx
Root.jsx
App.jsx
```

Cycle de vie d'un Composant React Native

Cycle de vie d'un composant

Initialisation

- props
- useState

```
import React, { useState } from 'react';
import { TextInput, View, Text } from 'react-native';

export default function MyAwesomeTextInput(props) {
    const [value, setValue] = useState('initial value');

    return (
        <TextInput
            onChangeText={text => {
                setValue(text);
                props.onChangeText(text);
            }}
            value={value} />);
}
```

`props` sont les paramètres que l'on passe au composant

`value` est un état (en lecture) de `MyAwesomeTextInput`, et `setValue` en est l'accesseur (en écriture)

`onChangeText` et `value` sont les `props` de `TextInput`.

Cycle de vie d'un composant

Initialisation

- props
- useState

Montage

chargement du composant en mémoire

1. exécution du code qui précède le JSX
2. rendu

Mise à jour

1. quand màj de l'état interne
   - via les `props`
   - via `setValue`
2. exécution du code qui précède le JSX
3. rendu du composant

Démontage

suppression du composant de la mémoire (quand le parent ne l'appelle plus)

Cycle de vie d'un composant

```
1   import React, {useState} from 'react';
2   import {TextInput, View, Text} from 'react-native';
3
4   export default function UselessTextInput() {
5       const [value, setValue] = useState('initial value');
6
7       return (
8           <TextInput
9               onChangeText = {text => setValue(text)}
10              value={value} />);
11  }
```

1. Initialisation : `value= 'initial value'`

Cycle de vie d'un composant

```
1   import React, {useState} from 'react';
2   import {TextInput, View, Text} from 'react-native';
3
4   export default function UselessTextInput() {
5       const [value, setValue] = useState('initial value');
6
7       return (
8           <TextInput
9               onChangeText = {text => setValue(text)}
10              value={value} />);
11  }
```

1. Initialisation : `value= 'initial value'`
2. montage : rendu du composant `TextInput` avec `value='initial value'`

Cycle de vie d'un composant

```
1   import React, {useState} from 'react';
2   import {TextInput, View, Text} from 'react-native';
3
4   export default function UselessTextInput() {
5       const [value, setValue] = useState('initial value');
6
7       return (
8           <TextInput
9               onChangeText = {text => setValue(text)}
10              value={value} />);
11  }
```

1. Initialisation : `value= 'initial value'`
2. montage : rendu du composant `TextInput` avec `value='initial value'`
3. interaction avec l'utilisateur :
   - `onChangeText`
   - déclenche le callback
   - appelle `setValue(text)`
   - déclenche un re-rendu avec `value` à jour

Cycle de vie d'un composant

```
1  import React, {useState} from 'react';
2  import {TextInput, View, Text} from 'react-native';
3
4  export default function UselessTextInput() {
5      const [value, setValue] = useState('initial value');
6
7      return (
8          <TextInput
9              onChangeText = {text => setValue(text)}
10             value={value} />);
11 }
```

1. Initialisation : `value= 'initial value'`
2. montage : rendu du composant `TextInput` avec `value='initial value'`
3. interaction avec l'utilisateur :
   - `onChangeText`
   - déclenche le callback
   - appelle `setValue(text)`
   - déclenche un re-rendu avec `value` à jour
4. le composant parent ne nous appelle plus : démontage

Quand récupérer des données ?

Récupérer des données : l'exemple d'une liste d'utilisateurs

- soumettre des requêtes lors du rendu ?
  - mauvaise idée car incontrôlé
  - tout changement d'état ou de props redéclenche les requêtes

Récupérer des données : l'exemple d'une liste d'utilisateurs

- soumettre des requêtes lors du rendu ?
  - mauvaise idée car incontrôlé
  - tout changement d'état ou de props redéclenche les requêtes
- besoin de spécifier **sous quelle conditions** effectuer une action lors d'un rendu de composant
- **useEffect** lie un callback à une variable

Récupérer des données : l'exemple d'une liste d'utilisateurs

- soumettre des requêtes lors du rendu ?
  - mauvaise idée car incontrôlé
  - tout changement d'état ou de props redéclenche les requêtes
- besoin de spécifier **sous quelle conditions** effectuer une action lors d'un rendu de composant
- **useEffect** lie un callback à une variable
- les effets/callbacks sont déclenchés :
  - après l'étape de rendu
  - et *seulement si* la variable spécifiée a été modifiée

Récupérer des données : l'exemple d'une liste d'utilisateurs

```
26  export default function AllUsers({ connectionToken }) {
27      const [users, setUsers] = useState(null);
28      const [isLoading, setIsLoading] = useState(false);
29      const getAllUsersRequest = () => {
30          setIsLoading(true);
31          sendRequest(new URL('/get_all', 'http://localhost:5000/'), 'GET', connectionToken)
32              .then((value) => {
33                  setIsLoading(false);
34                  setUsers(value.users);
35              });
36      }
37      useEffect(() => {
38          getAllUsersRequest();
39      }, [connectionToken]);
40      return (
41          <View>
42              {isLoading &&
43                  <ActivityIndicator size='large' animating={true} color='#FF0000' />}
44              {users != null ? <FlatList
45                  data={users}
46                  renderItem={({ item }) => <AllUsersItem item={item} key={item.id} />}
47                  keyExtractor={item => item.id}
48              /> : null}
49          </View>
50      );
51  }
```

- `getAllUsersRequest` récupère les données utilisateur sur le serveur
- puis met à jour l'état `users` : cela déclenche un nouveau rendu !

- **useEffect** assure que `getAllUsersRequest` n'est appelé **que** lorsque `connectionToken` change.
  - et pas quand `users` change
  - petite problème sinon...

Récupérer des données : l'exemple d'une liste d'utilisateurs

Qui est mieux ?

1. rendu de la vue principale
2. récupérer les id des utilisateurs
3. rendu de la liste utilisateurs
    - pour chacun : récupérer les données

1. rendu de la vue principale
2. récupérer les id **et données** des utilisateurs
3. rendu de la liste utilisateurs

Récupérer des données : l'exemple d'une liste d'utilisateurs

Qui est mieux ?

1. rendu de la vue principale
2. récupérer les id des utilisateurs
3. rendu de la liste utilisateurs
    - pour chacun : récupérer les données
4. N+1 requêtes : lent et désordonné

1. rendu de la vue principale
2. récupérer les id **et données** des utilisateurs
3. rendu de la liste utilisateurs
4. aller chercher les données le plus tôt possible

Rappel Javascript

```
axios({method:'GET',
   url : http://localhost:5000/all_users})
```

```
@app.get('/all_users',authorize='READ')
def get_data():
   return jsonify({'data':42}),200
```

```
.then(response = { doSomething(); })
```

Quelques recettes JS(X)

Syntaxe déstructurante et spread

```
let oh = {nom: 'hermant', prenom : 'olivier',
  labo : 'CRI', ecole : 'Mines'}
let {prenom, nom, ...rest} = moi
console.log(prenom); // olivier
console.log(nom); // hermant
console.log(rest); // {labo: 'CRI', ecole: 'Mines'}
let fc = {prenom: 'Fabien', nom: 'Coelho', ...rest}
fc =  {...oh, prenom: 'Fabien', nom: 'Coelho'}
```

- fonctionne avec les objets et les listes (arrays)
- cas d'usage typique:

  `function DemiJournee({choixMois,changeMois})`

  au lieu de

  `function DemiJournee(props)`

  puis `props.choixMois` et `props.changeMois`
- NB : syntaxe spread possible dans les composants (passage de plein de props inconnues). **Peu conseillé**.

Syntaxe abrégée des objets

```
let a = 1, b = 2
let o = {a : a, b : b}
let oo = {a, b}
```

Construit deux objets identiques

- attributs nommés a et b, de valeur respective 1 et 2

Composants avec enfants

Lorsque l'on écrit

```
<MonComposant>
<TextInput />
<Text>Ha</Text>
</MonComposant>
```

alors la fonction aura un "props" children:

```
function MonComposant({children}) {
  // ...
}
```

Nouveauté 2024 : le débogueur

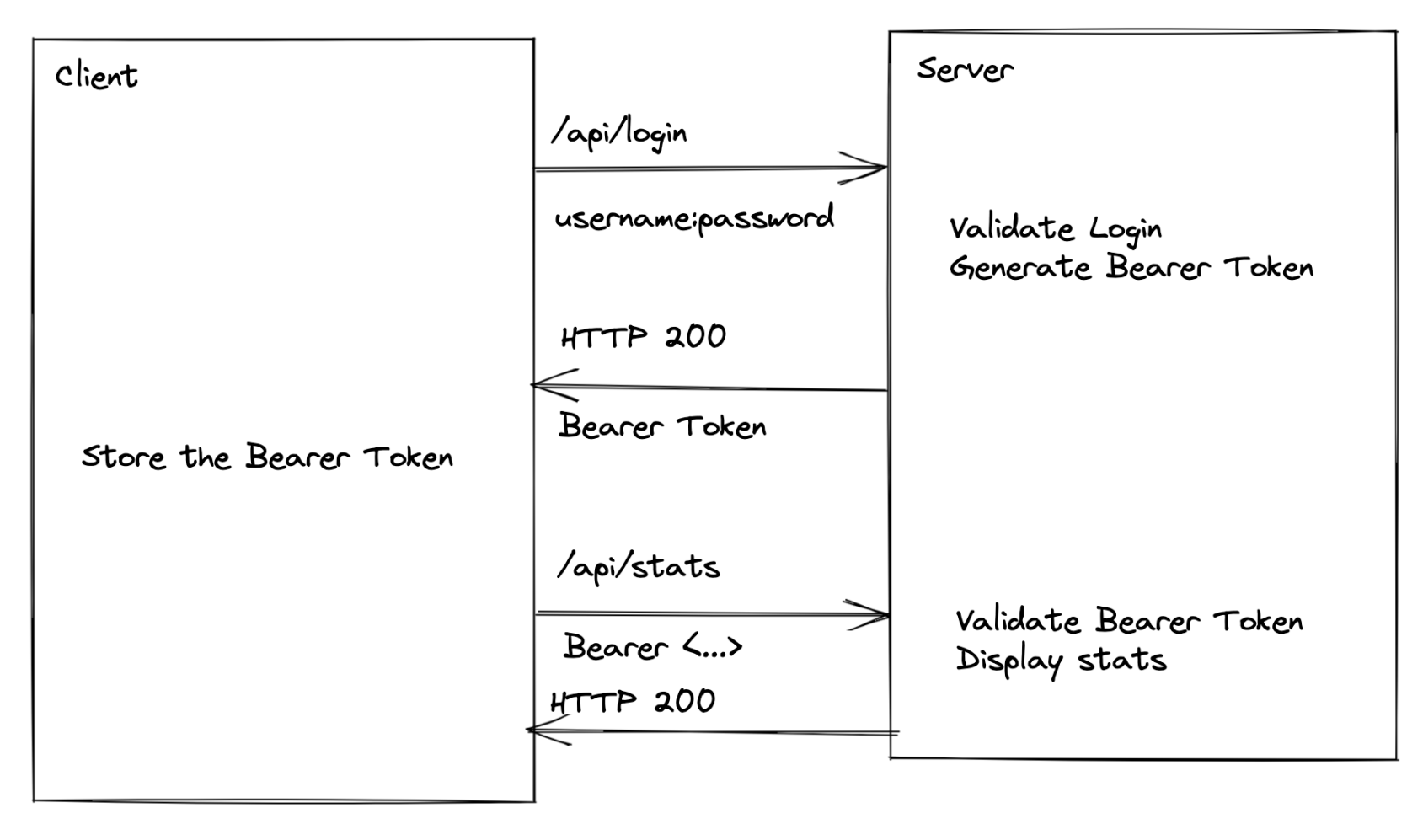
- dans le terminal Metro (`"npm start"`),
- taper `'j'`
- permet une exécution pas à pas, l'inspection des variables, objets, etc.
- voir `https://reactnative.dev/docs/react-native-devtools`

Plus de détails

- Voir la liste de recette JS, JSX et RN ici : https://www.mobapp.minesparis.psl.eu/faq/
- se renseigner sur **useContext**

TP : authentification

TP : authentification



TP : authentification

- différentes vues : **useState**
- chargement des données au moment du rendu : **useEffect**
- stockage login/mdp/token : architecture de l'application
- votre point de départ de projet