

# “Advanced” React Native

Emily Tripoul

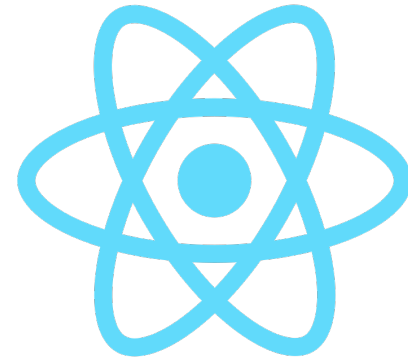
Meta!



# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases

# What is React Native



- React native is a front end Framework for mobile development
  - Front end = What the user see, interact with, and download&execute on their phone (or browser)
  - Back end = What runs on a distant server. The user does not have full power to interact with it. The interaction are limited to a specific interface (API Rest)
  - Framework = A library that controls how you write your code. You need to fit the rules of the library.

# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases

# NPM, Node, and the Javascript ecosystem

- Javascript runs in a browser
- Node.js allows running javascript outside of the browser
  - → Creating Javascript application
  - Similar to Java Virtual Machine
- A Node.js application or library is called a Package



# Node.js Quiz

- Which one can interact directly with the file system?
  - Node.js
  - Your Browser
  - Both
- Which one can act as a Server?
  - Node.js
  - Your Browser
  - Both
- Which one can act as a Client?
  - Node.js
  - Your Browser
  - Both

# Node.js Quiz

- Which one can interact directly with the file system?
  - Node.js → FileSystem API enables access to every file
  - Your Browser → LocalFileSystem API has limited access to only the current website data
  - Both
- Which one can act as a Server?
  - Node.js → Yes! Express is a Node.JS server similar to Flask
  - Your Browser → No
  - Both
- Which one can act as a Client?
  - Node.js → Yes
  - Your Browser → Yes
  - Both



# Npm & npx

- Npm is a package manager
- Npm allows downloading javascript code written by other developer
  - It's similar to `pip install` for python or `apt-get` on linux
- Npm also allows configuring the options to run the package
  - `npm run my-package`

```
{  
  "name": "Emily-secret-project",  
  "version": "1.0.0",  
  "scripts": { "my-package": "./node_modules/my-package/bin/" }  
}
```

- Npx is a shorthand for `npm run`
  - `npx my-package`

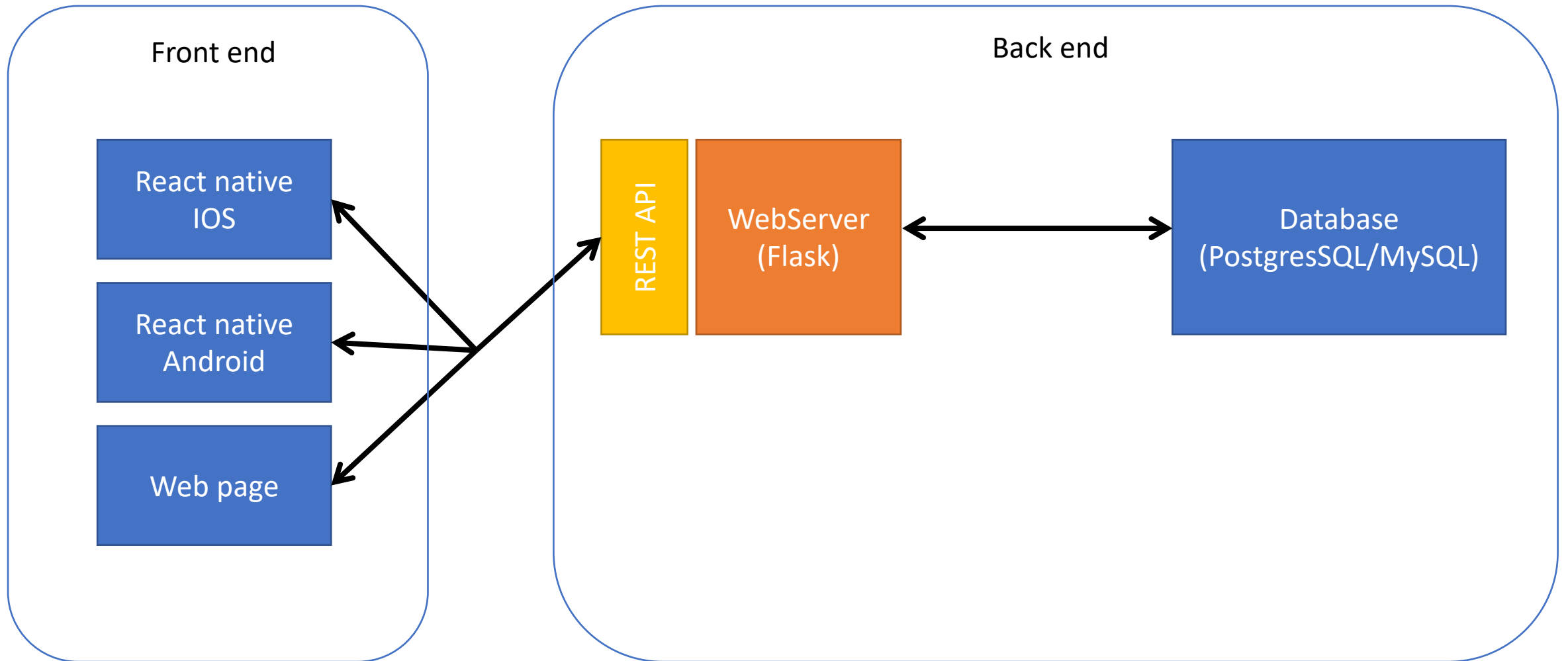
# Npm, npx, & react

- `npm install` → Install the packages required for react-native
  - React native server, React native compiler
- `npx react-native start` → This starts the react native compiler
- `npx react-native run-android` → Use the react native compiler, connect to the phone, and launch the android app

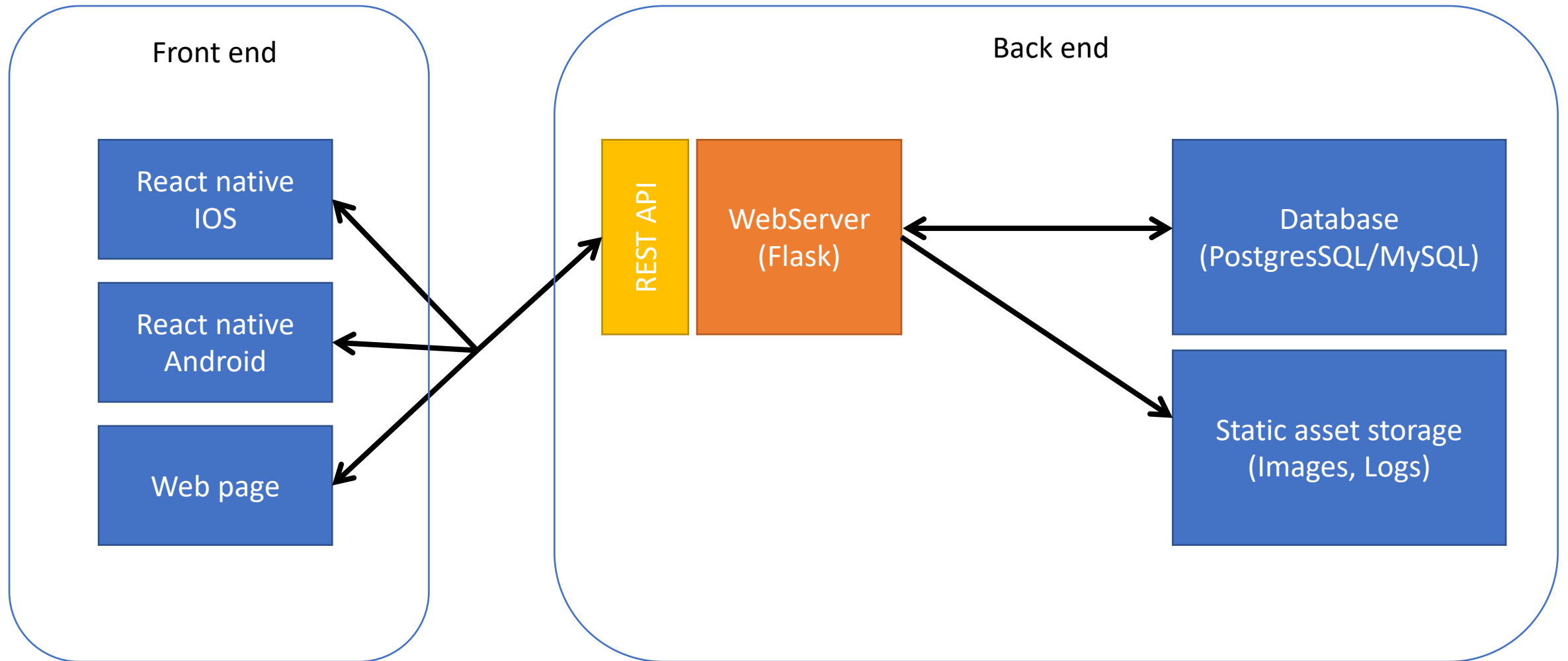
# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases

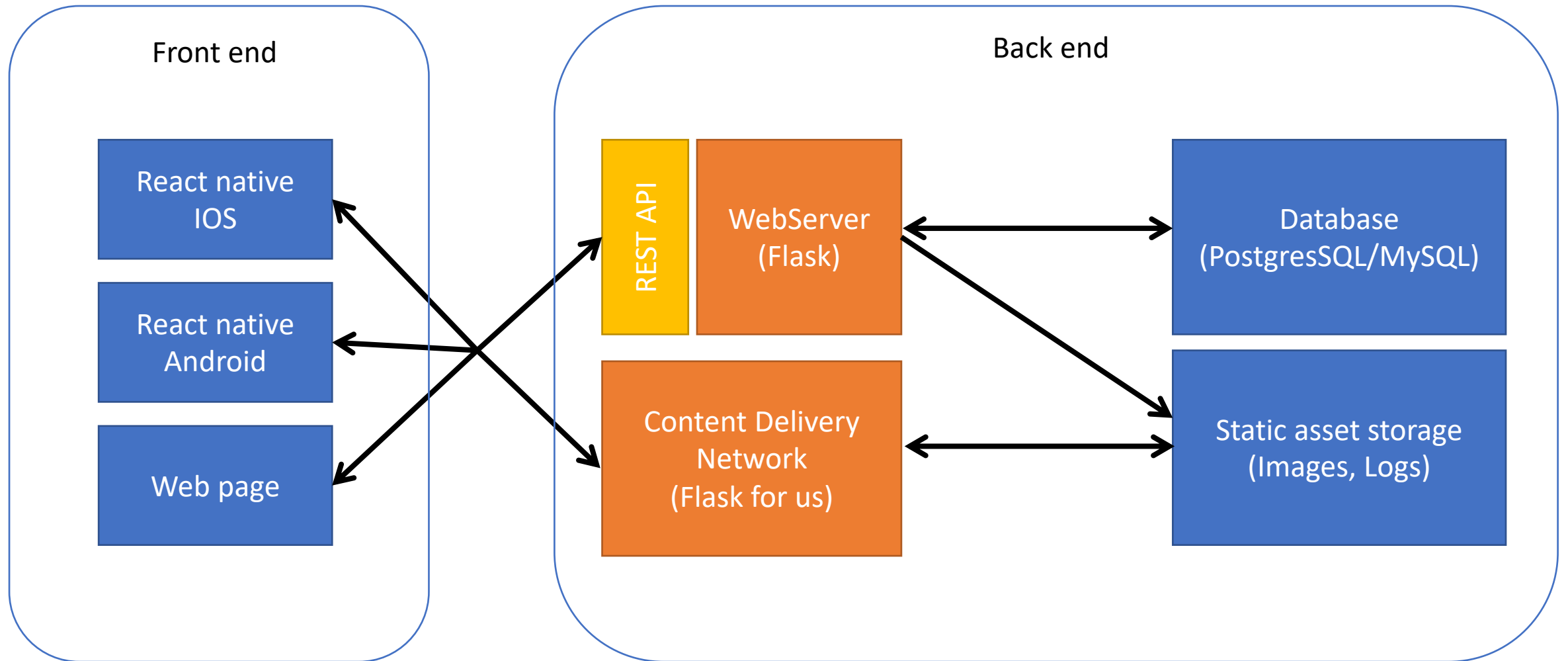
# Typical Architecture of a React Native application



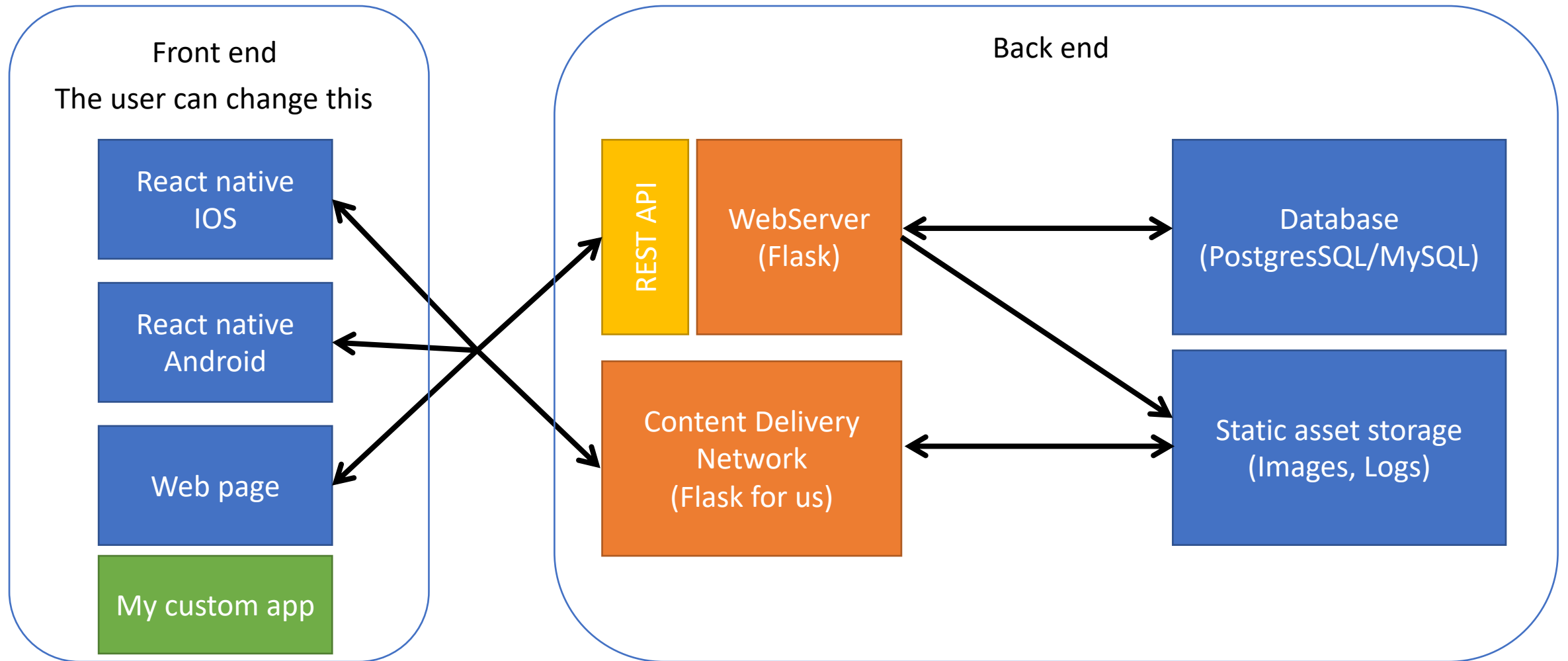
# Typical Architecture of a React Native application



# Typical Architecture of a React Native application



# Typical Architecture of a React Native application



# Where is the application logic?

## Front end (React Native)

- Display the login page for logged out users
- What are the action the user can take?

## Back end (Flask)

- Is the login correct?
- Can the user actually do them?



# Where is the application logic?

## Front end (React Native)

- Display the login page for logged out users
- What are the action the user can take?

## Back end (Flask)

- Is the login correct?
- Can the user actually do them?

## Conclusion

- The validation logic should always be on the server
- The application logic is often duplicated

Quick run into the TP

07:39 77%

Logged out

### Login

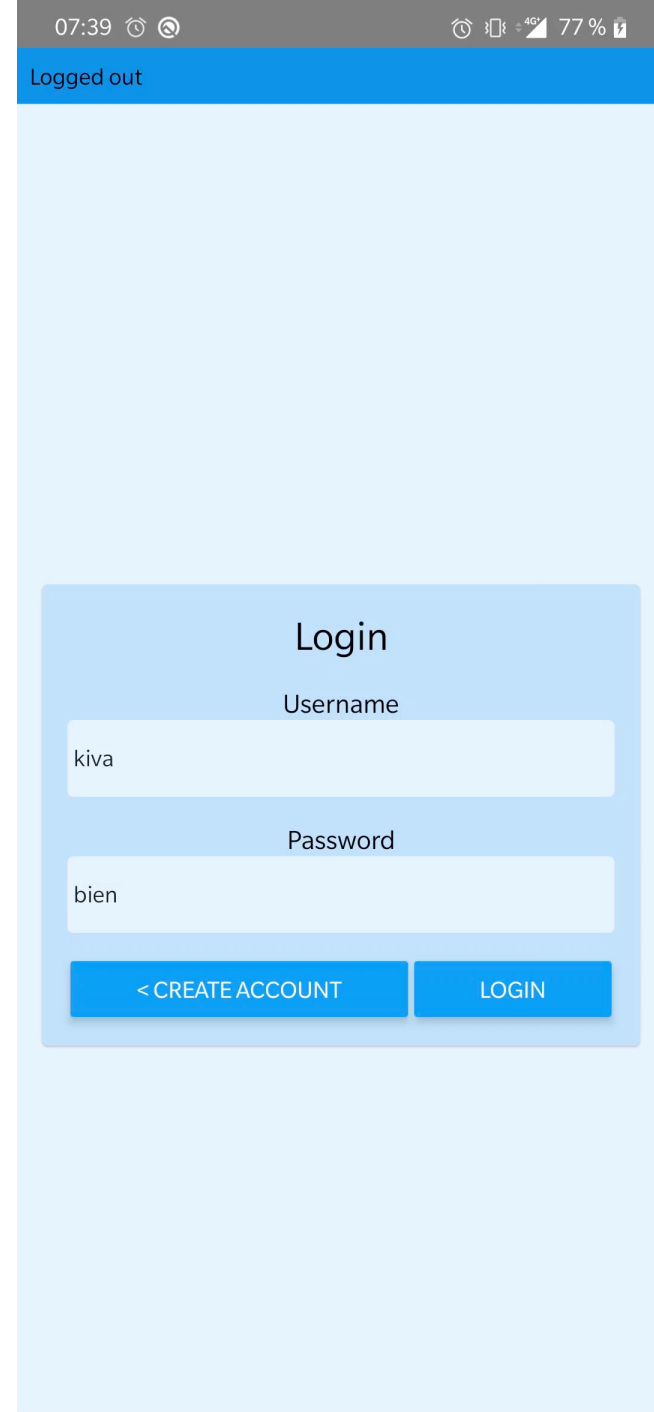
Username

kiva

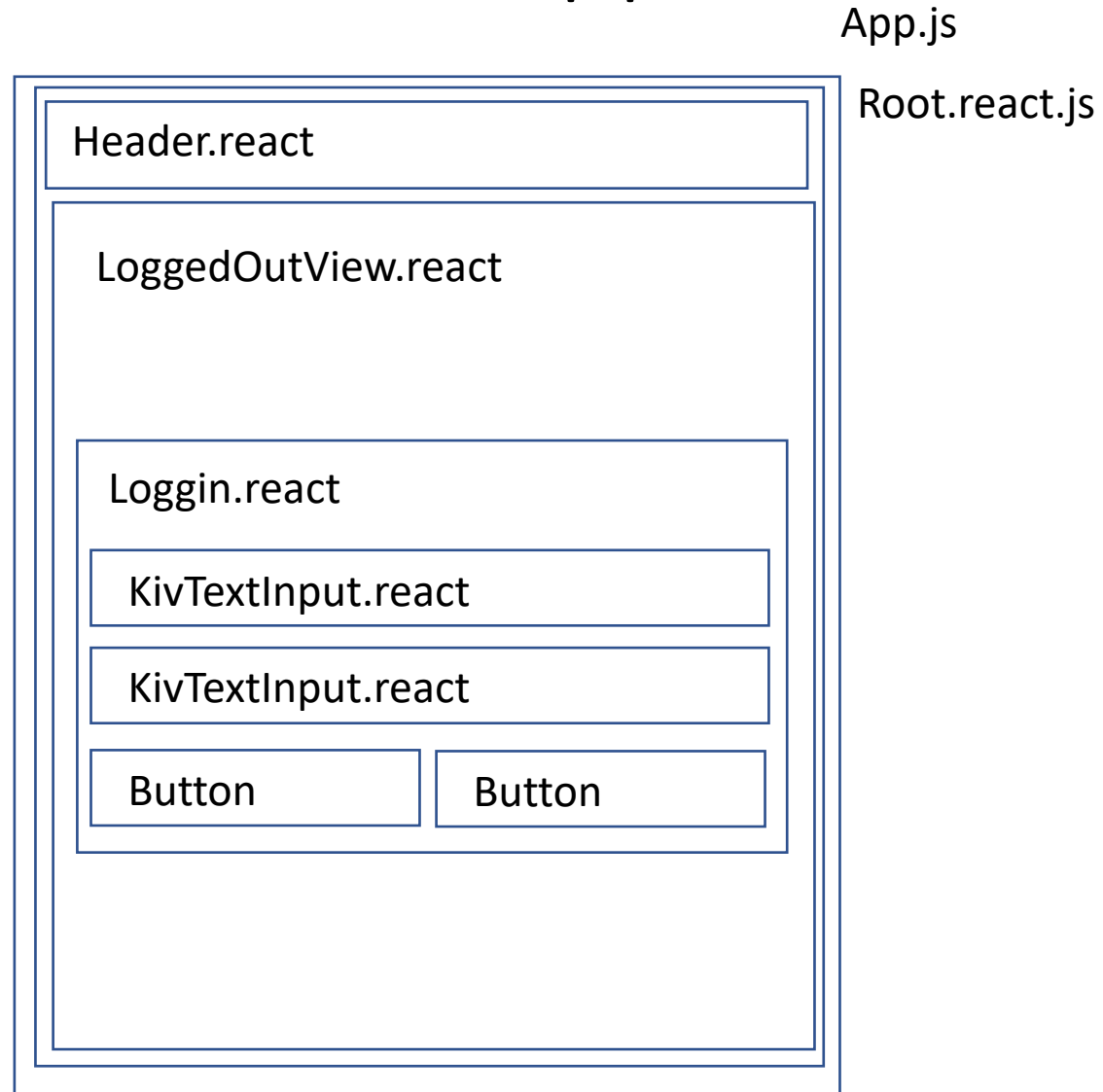
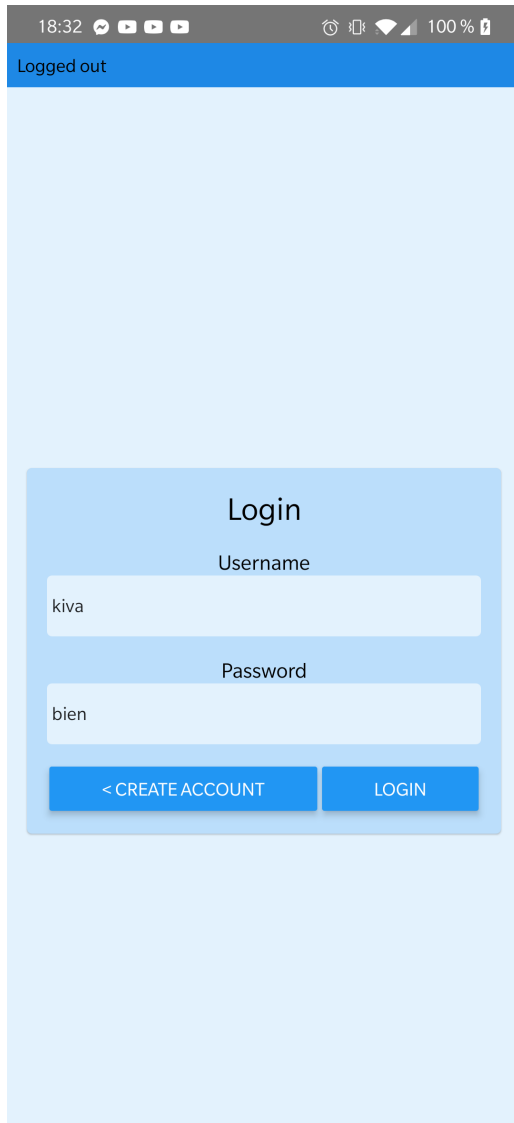
Password

bien

< CREATE ACCOUNT LOGIN

A mobile application login screen with a light blue background. At the top, there is a status bar showing the time 07:39 and battery level 77%. Below that is a blue header with the text "Logged out". The main content is a white rounded rectangle containing the "Login" form. The form has a title "Login", a "Username" field with the value "kiva", a "Password" field with the value "bien", and two buttons at the bottom: "< CREATE ACCOUNT" and "LOGIN".

# Structuring a React Native App



# React native app folder structure

- Root.react → Handles the main routing between logged in/ logged out view; Stores the credentials
- LoggedOut/
  - LoggedOutView.react → Handles the main routing between login & create account
  - Login.react
  - CreateAccount.react
- Common/
  - KivTextInput.react → Common component for the text input in react

# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases

# Lifecycle of a function component – Remember your last TP 😊

```
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 are the parameters that we pass to the component

Value is a state of UselessTextInput

setValue is a way to change the state of Value

Props.onChangeText is a props of MyAwesomeTextInput

onChangeText and value are props of TextInput

# Lifecycle of a function component – Remember your last TP 😊

```
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 are the parameters that we pass to the component

Value is a state of UselessTextInput

setValue is a way to change the state of Value

Props.onChangeText is a props of MyAwesomeTextInput

onChangeText and value are props of TextInput

We create a hierarchy of Components:

MyAwesomeTextInput contains a TextInput

# Lifecycle of a function component

UserStory : The User input their name in a Text Input

```
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} />);
}
```

1. The User Input their name in the TextInput



# Lifecycle of a function component

UserStory : The User input their name in a Text Input

```
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} />);
}
```

1. The User Input their name in the TextInput to 'emily'
2. [Native] The callback onChangeText is called with 'text=emily'

# Lifecycle of a function component

UserStory : The User input their name in a Text Input

```
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} />);
}
```

1. The User Input their name in the TextInput to 'emily'
2. [Native] The callback onChangeText is called with 'text=emily'
3. setValue(text); => We want to update value to 'Emily'  
This will happen on the next update of the react component

# Lifecycle of a function component

UserStory : The User input their name in a Text Input

```
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} />);
}
```

1. The User Input their name in the TextInput to 'emily'
  2. [Native] The callback onChangeText is called with 'text=emily'
  3. setValue(text); => We want to update value to 'Emily' This will happen on the next update of the react component
  4. Props.onChangeText() => We call another callback defined in the context where MyAwesomeTextInput is used.
- ➔ This is called bubbling up callbacks.

# Lifecycle of a function component

UserStory : The User input their name in a Text Input

```
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} />);
}
```

```
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} />);
}
```

Be careful of setValue / value usage!

# Lifecycle of a function component

```
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 }
```

UserStory : The User input their name & password

# Lifecycle of a function component

```
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 }
```

UserStory : The User input their name & password

Requirements:

- 2 MyAwesomeTextInput
- 1 Button
- 1 props function `onConnect` called when the user click the button

# Lifecycle of a function component

```
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 }
```

UserStory : The User input their name & password

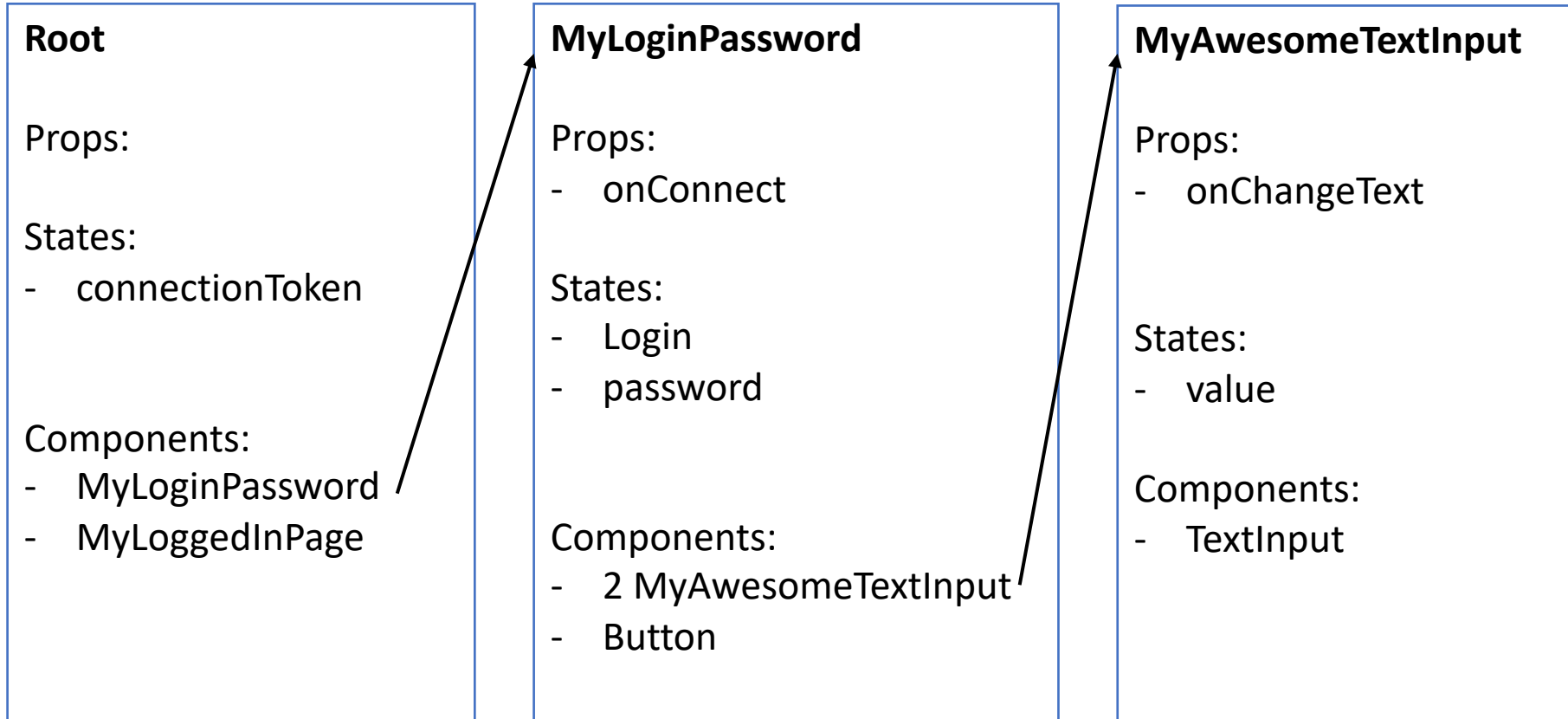
Requirements:

- 2 MyAwesomeTextInput
- 1 Button
- 1 props function `onConnect` called when the user click the button

What does `onConnect` do?

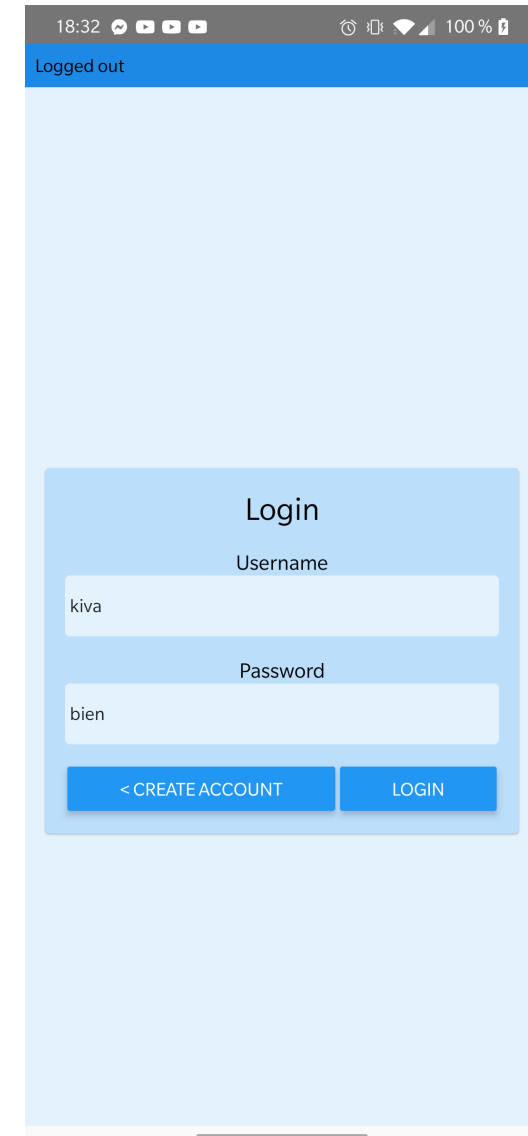
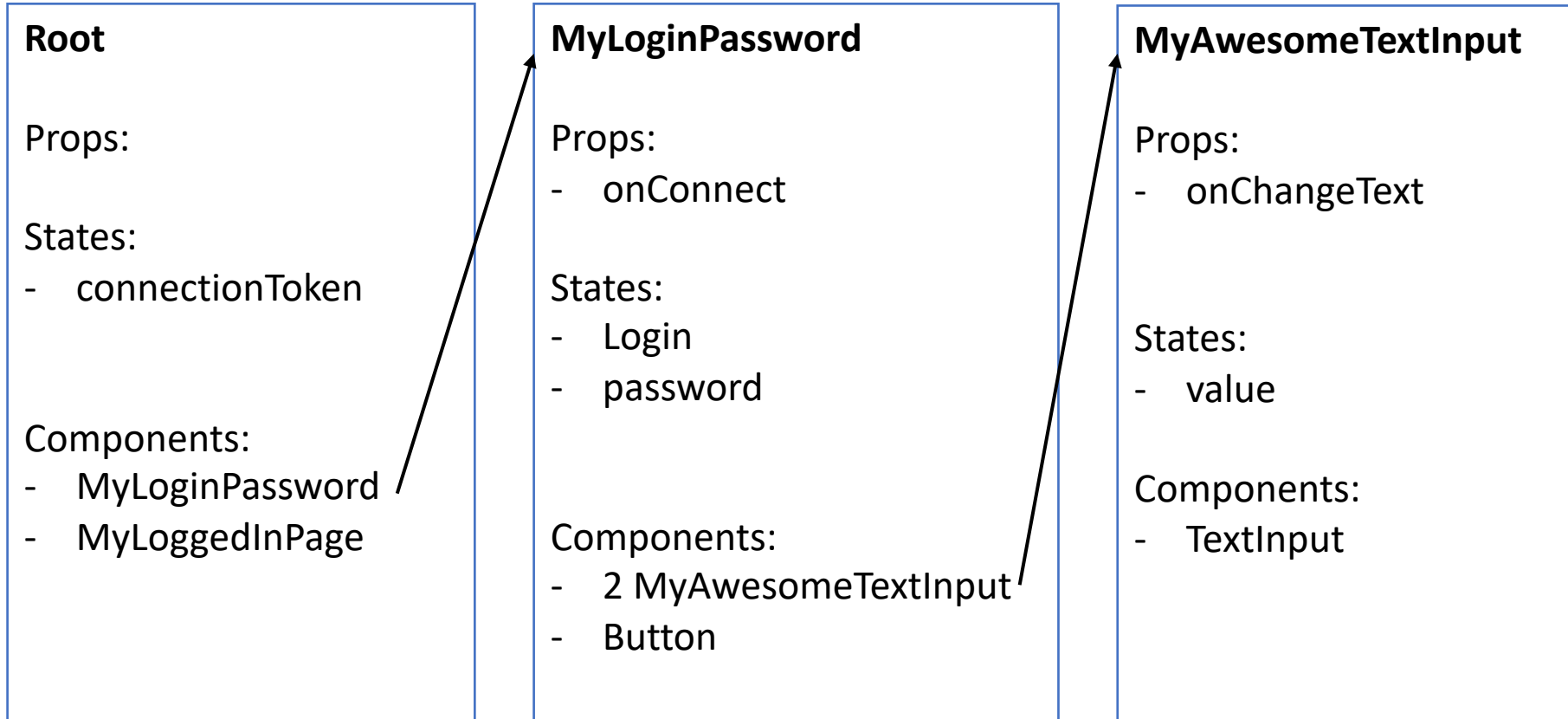
1. Call the server sendRequest to /api/login
2. Parse the response
3. Either
  - a. Store the token locally
  - b. Display an error message "Bad password"

# Composing our components





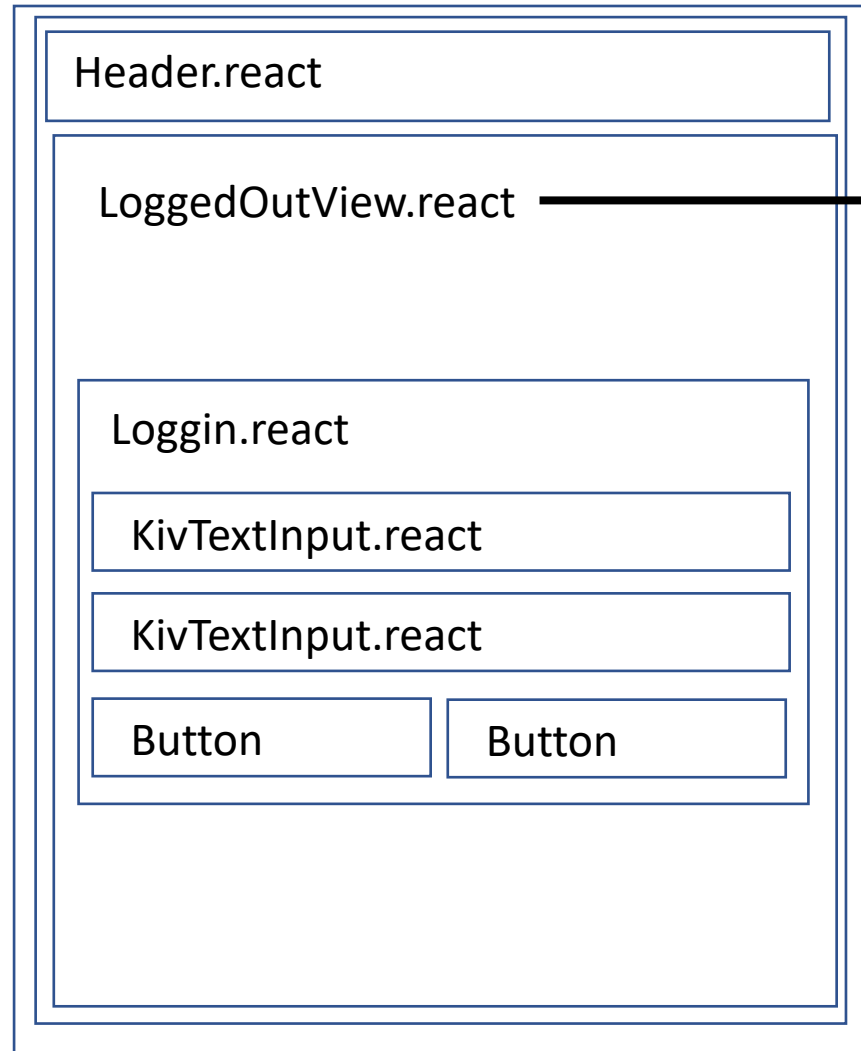
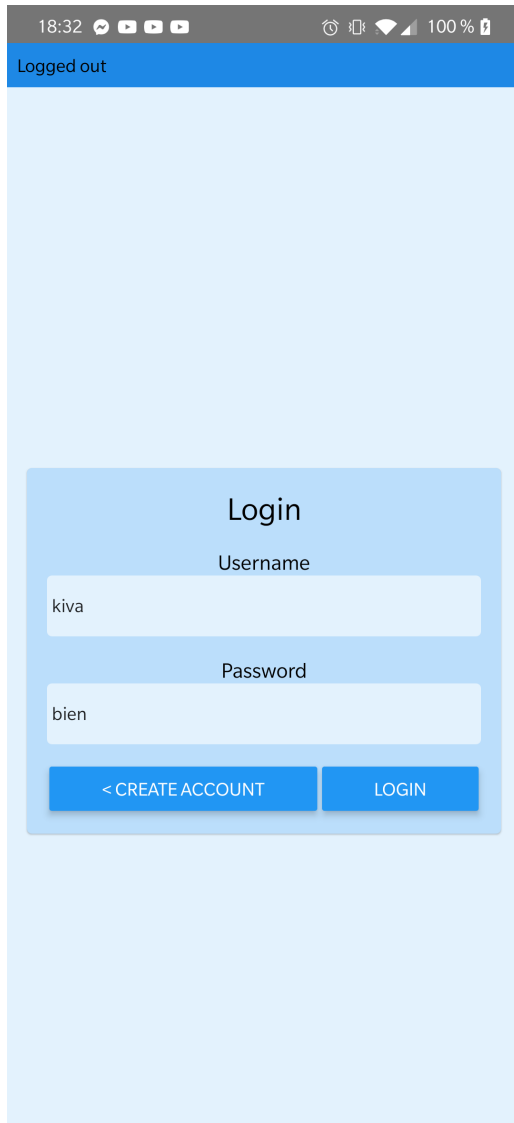
# Composing our components



# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases

# Handling Views



App.js

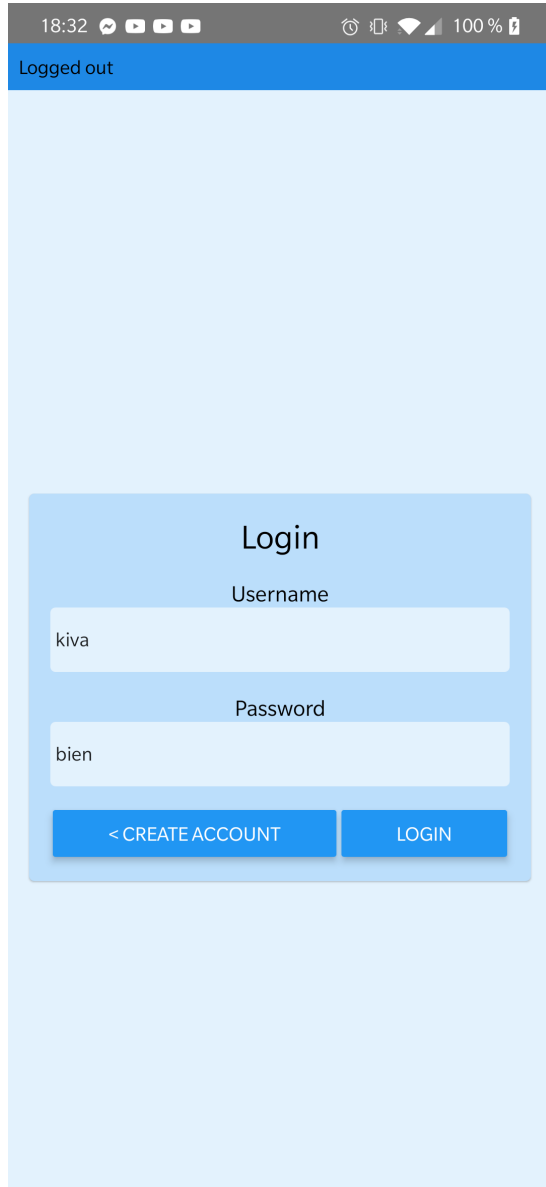
Root.react.js

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

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

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

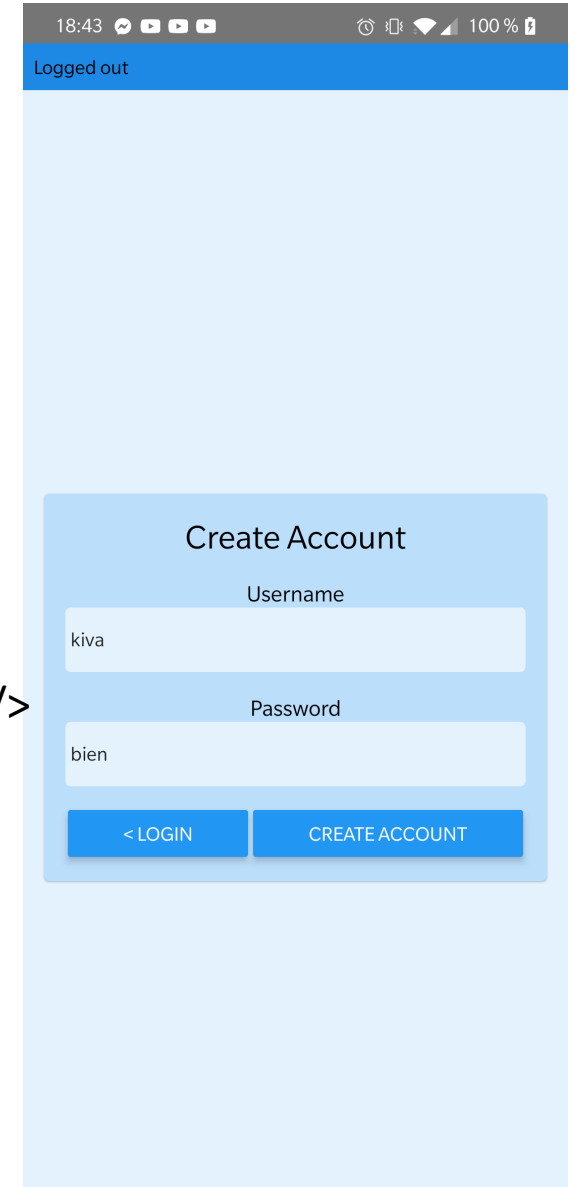
# Handling Views



`setTab('create_account')`



`setTab('login')`



The call is passed down to each button via “props”

Parent:

- `<Login onCreateAccount={() => setTab('create_account')} />`

Login.react:

- `<Button onPress={props.onCreateAccount} />`

# Breaking down a large file

```
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;
```

What to break down?

- Large files (>100 lines)
- Functions used everywhere (sendRequest)
- Components used multiple times (MyAwesomeInput)

Extreme:

1 file 1 react component

1 file 1 function

# Breaking down a large file

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

> const KivAppA = () => { ...
}

> function Hello(props){ ...
}

> function KeyValues(props){ ...
}

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 }
```

# How I break down my files

core/

  sendRequest.js

  MyAwesomeTextInput.react.js

loggedOut/

  MyLoginPage.react.js

loggedIn/

  MySuperCoolApp.react.js

Root.react.js => switch(tab)

App.js

# Plan

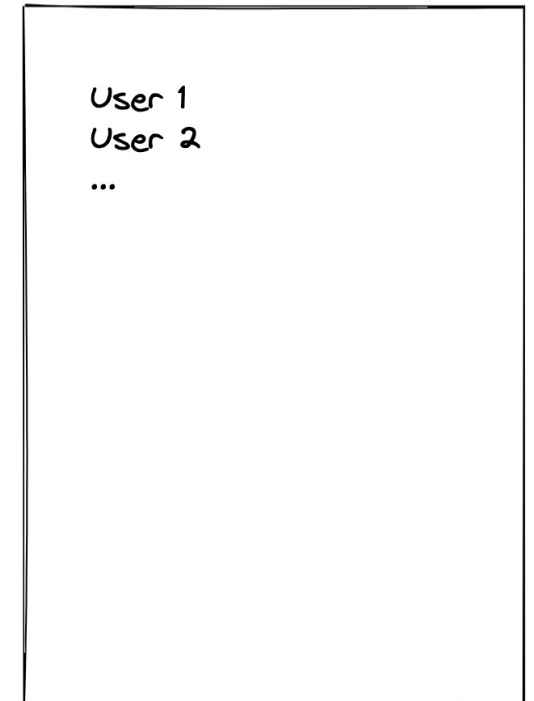
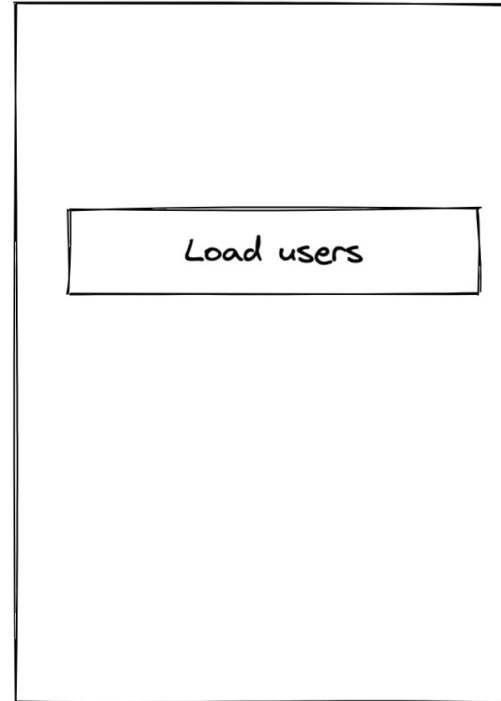
1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases



# Data Fetching on action

The user click a button

1. → Fetch data
2. Use the data fetched and update the component state
3. → re-render the component with the fetched data

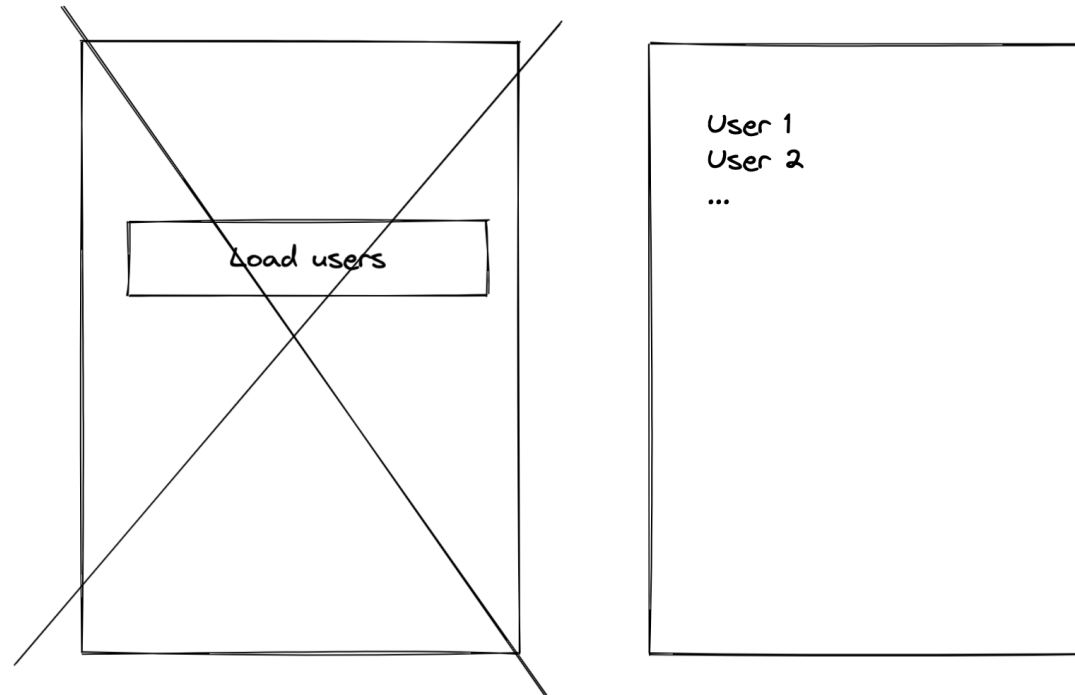


# Data Fetching on render

## Example: loading a list of users

Fetch the data during the rendering step?

This does not work as we don't know when to fetch the data: any state or props change can trigger a render



# Data Fetching on render

## Example: loading a list of users

Fetch the data during the rendering step?

This does not work as we don't know when to fetch the data: any state or props change can trigger a render

We need a way to specify when we want to execute an action during component rendering → **useEffect**

**useEffect** runs after the component rendering and only if specific variables are changed

# Data fetching on render

```
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 fetch the user on the server

getAllUsersRequest updates the state users

➔ We render the component again

useEffect ensures that we only fetch new data when 'filter' is updated

# Be careful of when data is getting fetched!

1. Render the main view
2. Fetch the users id
3. Render the list of user id (let's say 10 users)
4. Fetch each user data

# Be careful of when data is getting fetched!

1. Render the main view
2. Fetch the users id
3. Render the list of user id (let's say 10 users)
4. Fetch each user data

➔ Total 11 fetch! The app feels slow.

➔ Solution, fetch as much as you can, as soon as you can (On step 2 for example)

# Rappel Javascript

- User interface (UI) is highly asynchronous : We use callbacks & promise

```
Fetch( 'http://0.0.0.0/my_entrpoint',  
       {method:'GET'})
```

```
.then(response => response.json())
```

```
@app.route('/my_entrpoint', methods=['get'])  
def get_my_data():  
    return jsonify({'data':4})
```

# Rappel Javascript

- User interface (UI) is highly asynchronous : We use callbacks

```
Fetch( 'http://0.0.0.0/my_entrpoint',  
       {method:'GET' })
```

```
.then(response => response.json())
```

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

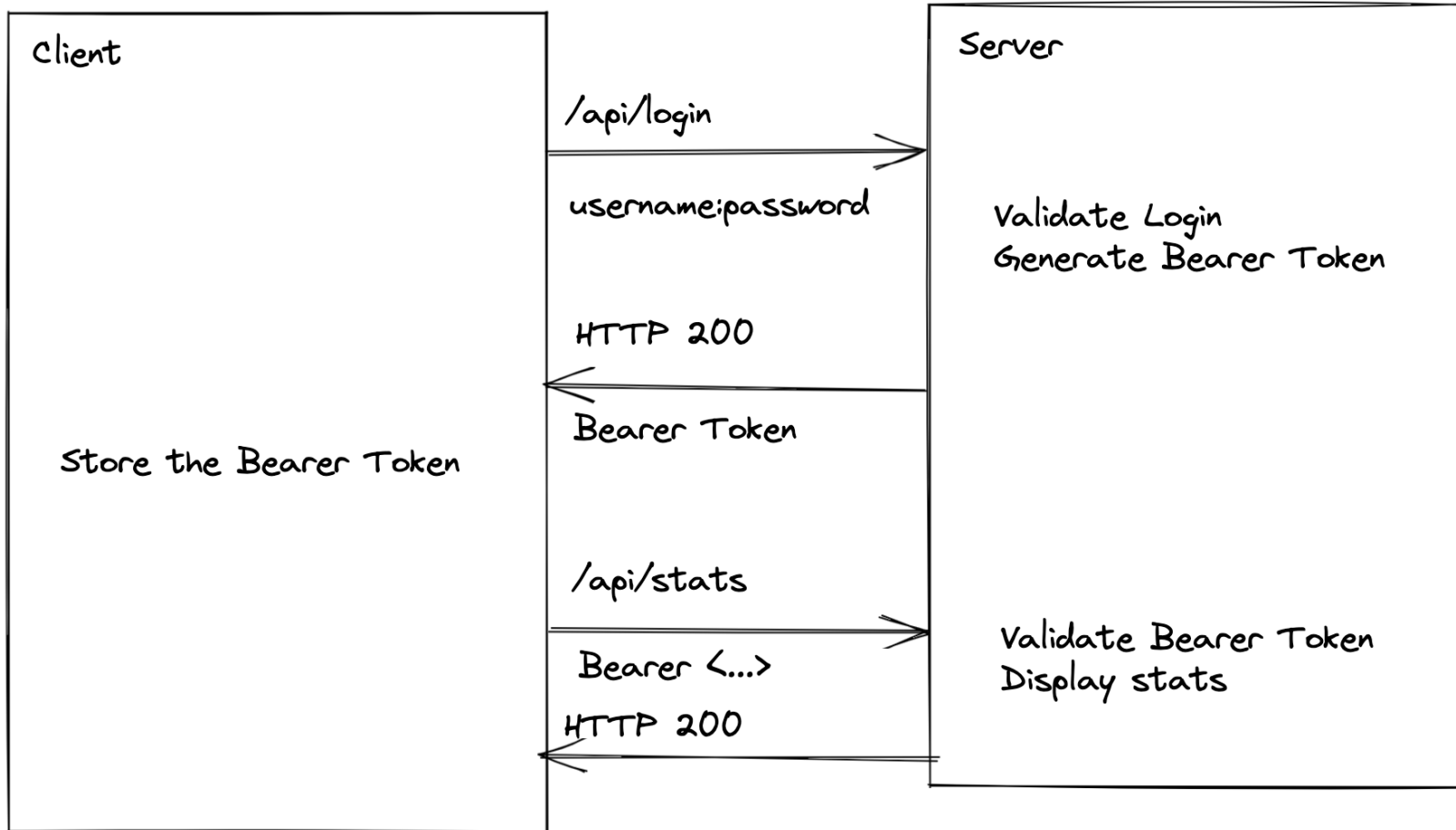
```
@app.route('/my_entrpoint', methods=['get'])  
def get_my_data():  
    return jsonify({'data':4})
```



# TP: Build an authentication system

- Use multiple panel → useState
- Load data on render → useEffect
- Store the login & password credentials in the right place →  
Application architecture

# TP: Build an authentication system



# Lifecycle of a function component

## Initialisation

1. Set the initial value of props
2. Set the initial values of useState

Props are the parameters that we pass to the component

# Lifecycle of a function component

## Initialisation

1. Set the initial value of props
2. Set the initial values of useState

```
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 }
```

Props are the parameters that we pass to the component

Value is a state of UserlessTextInput

setValue is a way to change the state of Value

onChangeText and value are props of TextInput

# Lifecycle of a function component

## Initialisation

1. Set the initial value of props
2. Set the initial values of useState

**Mounting** = storing the component in memory

1. Execute code before JSX
2. Render the component

# Lifecycle of a function component

## Initialisation

1. Set the initial value of props
2. Set the initial values of useState

## Mounting = storing the component in memory

1. Execute code before JSX
2. Render the component

## Updating

1. The internal state is updated
  1. Via props
  2. Via setState
2. Execute code before JSX
3. Render the component

# Lifecycle of a function component

## Initialisation

1. Set the initial value of props
2. Set the initial values of useState

## Mounting = storing the component in memory

1. Execute code before JSX
2. Render the component

## Updating

1. The internal state is updated
  1. Via props
  2. Via setState
2. Execute code before JSX
3. Render the component

## Unmounting = remove the component from memory

# Plan

1. Quid est React Native?
2. What is npm?
3. Architecture of a React Native App
4. LifeCycle of a react component
5. Practical use cases



# Life cycle of a useless input

```
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 }
```

# Life cycle of a useless input

```
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. Initialize

1. Value = `initial value`

# Life cycle of a useless input

```
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. Initialize

1. Value = `initial value`

## 2. Mount

1. Render the component  
TextInput with value  
`initial value`

# Life cycle of a useless input

```
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. Initialize
  - Value = `initial value`
2. Mount
  - Render the component
3. User interact with the input → onChangeText
  - Triggers the lambda callback
  - Calls setValue(text)
  - Triggers a new render with the updated value

# Life cycle of a useless input

```
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. Initialize
  - Value = 'Useless Placeholder'
2. Mount
  - Render the component
3. User interact with the input → onChangeText
  - Triggers an update with a new value={text}
4. The parent component stops loading the component => Unmount

# Handling error state

```
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 }
```

We use `isCorrect` to conditionally display the error message.

This can be used to validate data or do a specific action