React Native

Desenvolvimento de Software e Sistemas Móveis (DSSMV)

Licenciatura em Engenharia de Telecomunicações e Informática

LETI/ISEP

2025/26

Paulo Baltarejo Sousa

pbs@isep.ipp.pt

Instituto Superior de Engenharia do Porto

P.PORTO

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

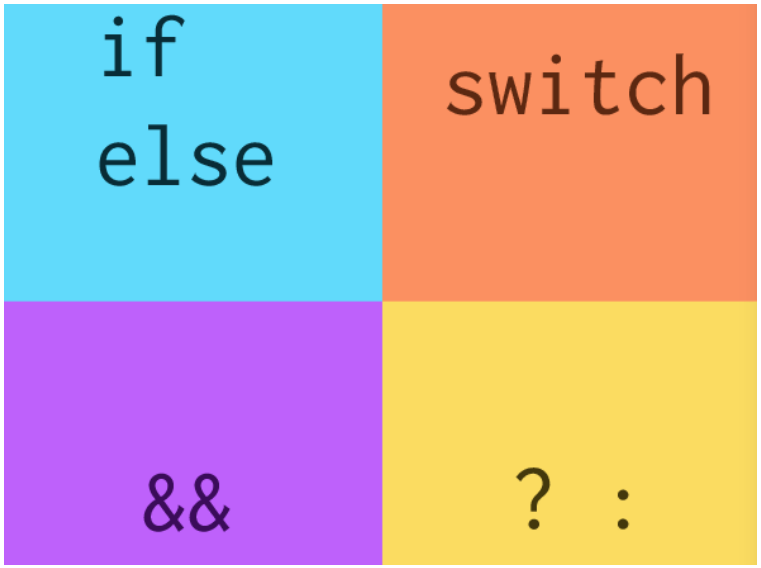
1. Conditional Rendering
2. Fetching data from REST API
3. Axios
4. Where to fetch in React's component tree?
5. Flux with Networking
6. Navigation
   - Stack
   - Tab
   - Drawer
7. How To Organize React Native Project
8. Bibliography

1 Conditional Rendering
2 Fetching data from REST API
3 Axios
4 Where to fetch in React's component tree?
5 Flux with Networking
6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project
8 Bibliography

What is?

- Conditional rendering is a term to describe the ability to render different user interface (UI) markup if a condition is true or false.
- In React, it allows us to render different elements or components based on a condition.



Using an `if...else` Statement

```
const [logged, setLogged] = useState(false);
...
let button;
if(logged == false){
  button = <LoggedInUser handleClick={handleClick}/>
}else{
  button = <LoggedOutUser handleClick={handleClick}/>
}
...
return (
<View style={{flex: 1, flexDirection: "column", alignItems:"center", justifyContent:"
      space-around", padding:6}}>
  <Text>
  This is a Demo showing several ways to implement Conditional Rendering in React.
  </Text>
  {button}
</View>
);
```

Using an `switch` Statement

```
  const [logged, setLogged] = useState(false);
  ...
const button = () => {
  switch(logged) {
    case true:
      return <LoggedInUser handleClick={handleClick}/>;
    case false:
      return <LoggedOutUser handleClick={handleClick}/>;
    default:
      return null;
  }
};
  ...
return (
  <View style={{flex: 1, flexDirection: "column", alignItems:"center", justifyContent:
        "space-around", padding:6}}>
    <Text>
    This is a Demo showing several ways to implement Conditional Rendering in React.
    </Text>
    {button()}
  </View>
);
```

Using Ternary Operators

```
const [logged, setLogged] = useState(false);
...
const button = logged == false ? <LoggedInUser handleClick={handleClick}/> : <
      LoggedOutUser handleClick={handleClick}/>;
  ...
return (
  <View style={{flex: 1, flexDirection: "column", alignItems:"center", justifyContent:
        "space-around", padding:6}}>
    <Text>
    This is a Demo showing several ways to implement Conditional Rendering in React.
    </Text>
    {button}
  </View>
);
```

Using Logical && (Short Circuit Evaluation)

```
const [logged, setLogged] = useState(false);
return (
  <View style={{flex: 1, flexDirection: "column", alignItems:"center", justifyContent:
        "space-around", padding:6}}>
    <Text>
    This is a Demo showing several ways to implement Conditional Rendering in React.
    </Text>
    {logged && <DisplayAnImage/>}
  </View>
);
```

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

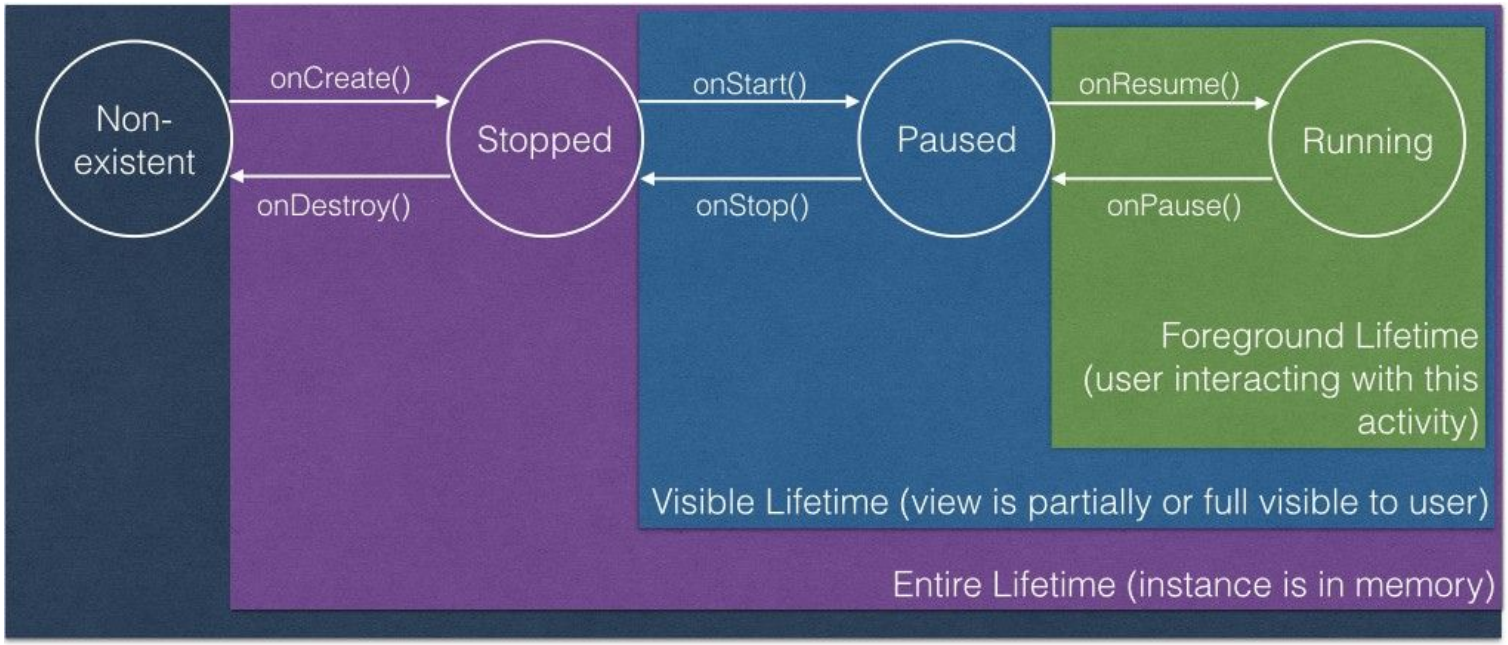
5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Lifecycle methods



- In functional components, the `useEffect` hook allows us to replace class component lifecycle code.

Lifecycle method to fetch data

- `componentDidMount` method is a good place to fetch data:
  - When this method runs, the component was already rendered once with the `render` method, but it would render again when the fetched data would be stored in the local `state` of the component with `setState`.
- React components are flexible and you can choose your own solution to fetch data.

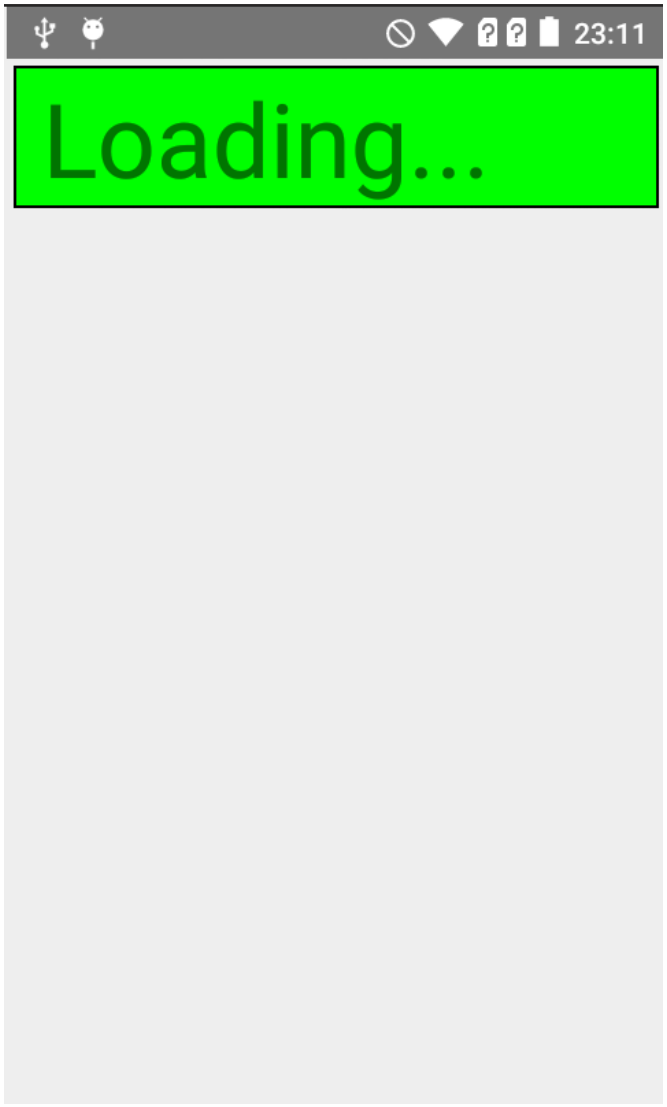
fetch

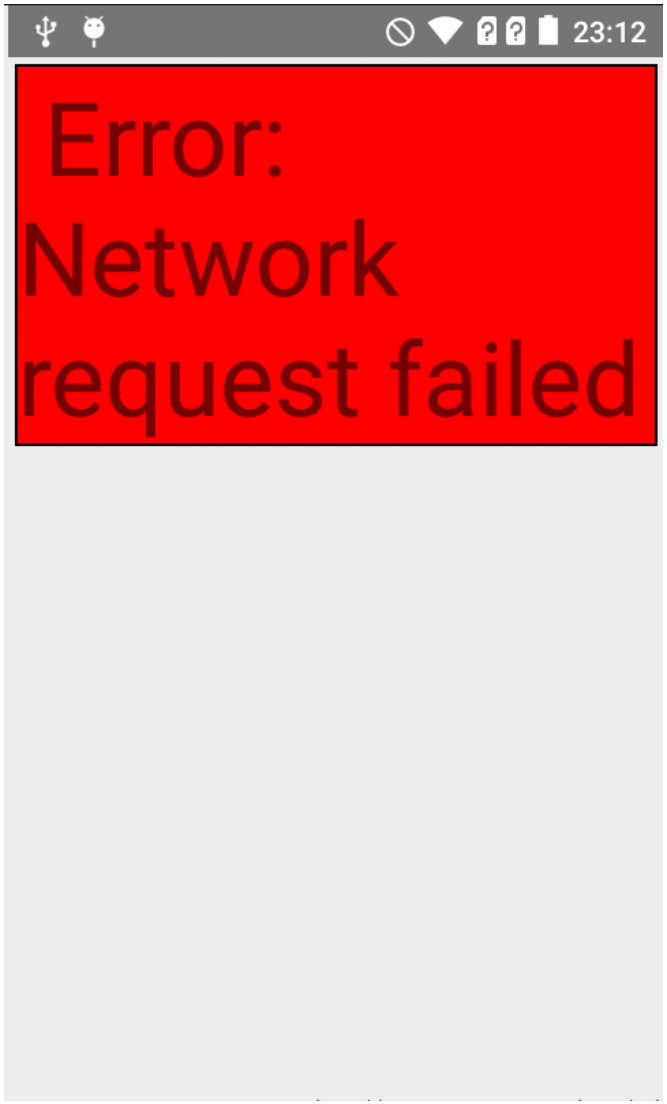
- The `fetch` method only has one mandatory argument, which is the URL of the resource you wish to fetch.

```
const initialData = {
  hits: [],
  query: "",
};
...
const [news, setNews] = useState(initialData);
...
fetch(API + DEFAULT_QUERY)
  .then(response => response.json())
  .then(json => setNews({...data, hits: json.hits, query: json.query}));
```

1. Invoke `fetch`
2. First `then`: When the promise is fulfilled, it returns a `Response` object, which exposes a `json` method.
   - The `json ()` method returns the response body as JSON and returns also a new promise, which means we need to chain on another `then`
3. Second `then`: triggered by the `response.json()` method, updates the `data`.

Loading... & Error





Loading (I)

- To display a loading spinner or similar we have to know the current state of data fetching.

```
const initialData = {
  loading:false,
  data: {
    hits: [],
    query: "",
  }
};
```

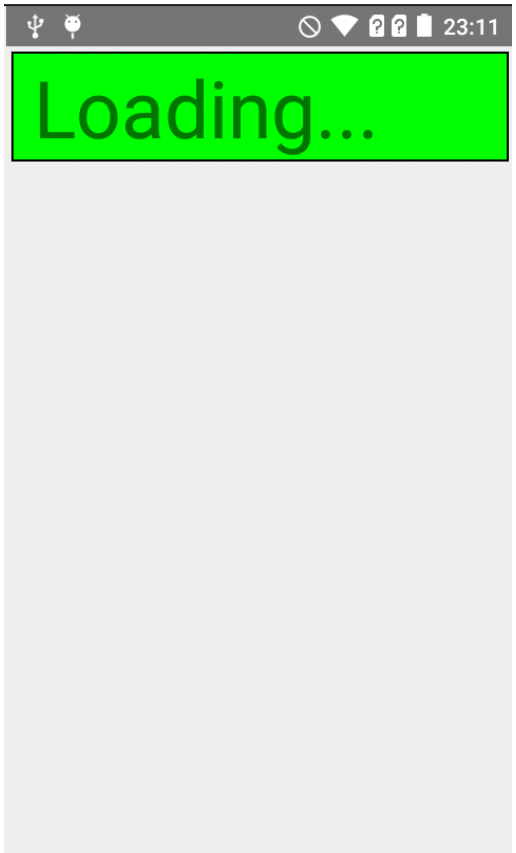
- Set `loading` based on the data fetching.

```
const [news, setNews] = useState(initialData);
const { loading, data } = news;
const { hits, query } = data;
const fetchNews = () =>{
  setNews({...news,
  loading: true,
  })
fetch(API + DEFAULT_QUERY)
  .then(response => response.json())
  .then(json => setNews({...news,
    loading: false,
    data:{
      hits:json.hits,
      query: json.query
    }
  }))
}
useEffect(() => {
  fetchNews();
  return ()=>{}
}, []);
```

Loading (II)

- Conditional rendering

```
if(loading == true){
  return(<View>
    <NewsLoading />
  </View>);
}else{
  return (<View>
    <NewsHeader query={query}/>
    <NewsList hits={hits}/>
  </View> );
}
```



Handling Error (I)

- Proper handling of errors should be considered in every project, since the server could be not responding (maintenance, hardware problems, ...) or the request is missing some parameters or ...

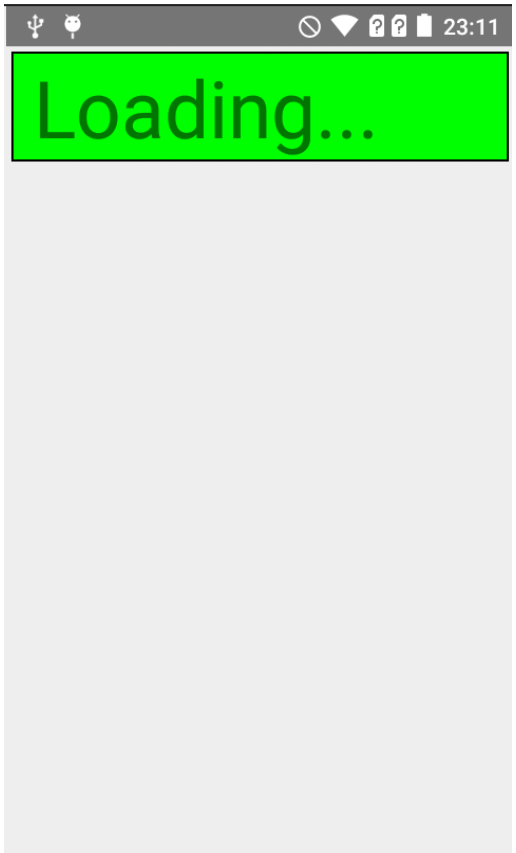
```
const initialData = {
  loading:false,
  error:null,
  data: {
    hits: [],
    query: "",
  }
};
```

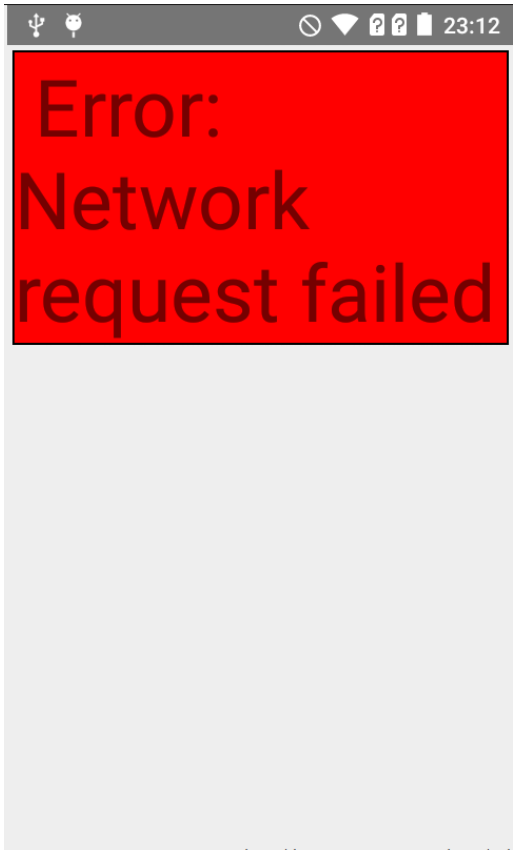
```
const fetchNews = () =>{
  setNews({...news,
    loading: true,
    error: null, })
  fetch(API + DEFAULT_QUERY)
  .then(response => response.json())
  .then(json => setNews({...news,
    loading: false,
    error: null,
    data:{
      hits:json.hits,
      query: json.query
    }}))
  .catch(error => setNews({...news,
    loading: false,
    error: error,
    data:{},}))
 }
...
```

Handling Error (II)

- Conditional rendering

```
if(loading == true){
  return(<View>
  <NewsLoading />
  </View>);
}else{
  if(error){
    return (<View>
    <NewsError message={error.
          message} />
    </View>);
  }else{
    return (<View>
    <NewsHeader query={query
          }/>
    <NewsList hits={hits}/>
    </View> );
  }
}
```



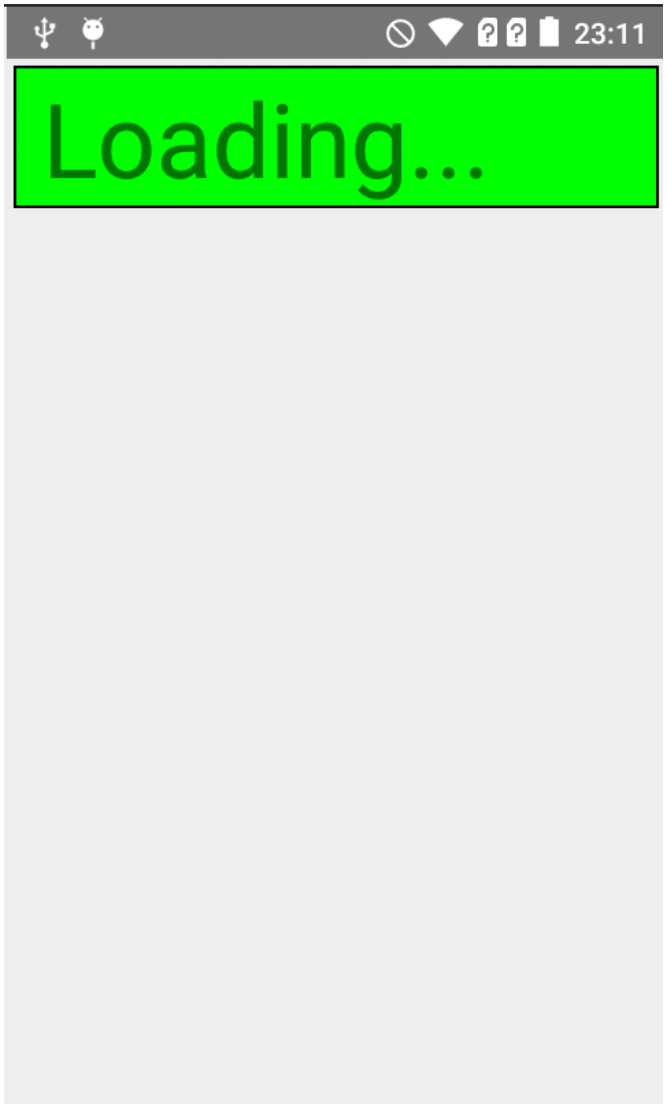


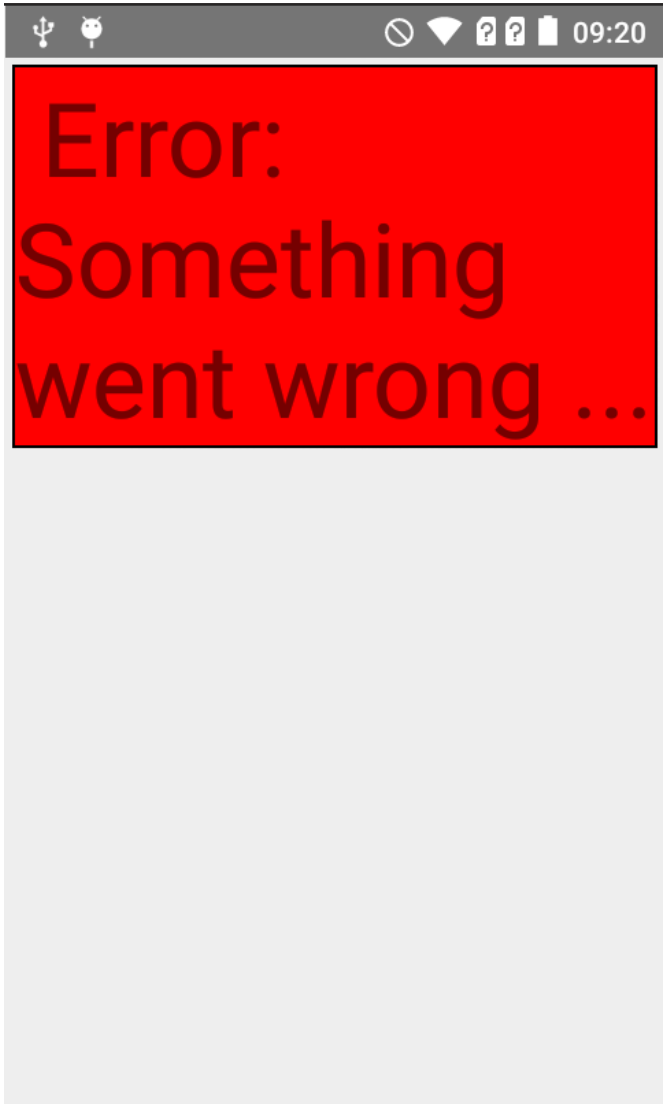
Handling Error (III)

- The `fetch` method will automatically throw an error for network errors but not for HTTP errors such as 4xx or 5xx responses.
- For HTTP errors we can check the `response.ok` property to see if the request failed and if it is the case you can force it to run into the `catch` block by throwing an error.

```
...
 fetch(API + DEFAULT_QUERY)
 .then(response => {
  if(response.ok) {
   return response.json();
  }else {
   throw new Error('Something went wrong ...');
  }
 })
 ...
 .catch(error => setNews({...news,
  loading: false,
  error: error,
  data:{},
 }))
```

Handling Error (IV)





Supplying request options

- The `fetch` method can optionally accept a second parameter, which is an option object that configures the request:

```
const request = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
};
fetch(URL, request)
  .then(...)
  .then (...)
  .catch(...);
```

Uploading JSON data

```
const data = {...}
const request = {
  method: 'POST',
  headers: {
  'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
};
fetch(URL, request)
  .then(...)
  .then (...)
  .catch(...);
```

async... await

- We can use the `async` keyword before a function name to wrap the return value of this function in a **Promise**.
- We can use the `await` keyword (in an `async` function) to wait for a promise to be resolved or rejected before continuing code execution in this block.

```
useEffect(async ()=>{
  setState({ isLoading: true, error:null });
  try{
    const response = await fetch(URL);
    const data = await response.json();
    setState({...state,hits: data.hits, isLoading:false, error:null });
  }catch(err){
    setState({...state, error:err, isLoading: false });
  }
  },[]);
```

- https://dmitripavlutin.com/javascript-fetch-async-await/

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Axios

- You can install `axios` in your project with `npm install axios` and then use it instead of the native `fetch` API in your project.

```
const fetchNews = () =>{
  ...
  axios.get(API + DEFAULT_QUERY)
  .then(result => {
    const json = result.data;
    setNews({
      ...news,
      loading: false,
      error: null,
      data: {
        hits: json.hits,
        query: json.query
      }
    })})
...
}
```

- Axios returns a JSON response.

Making Requests and Response

- Axios is a promise based HTTP client
- Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations
  - Axios making requests
    - `axios.get(url[, config])`
    - `axios.delete(url[, config])`
    - `axios.post(url[, data[, config]])`
    - `axios.put(url[, data[, config]])`
  - Axios Response object consists of:
    - `data` - the payload returned from the server
    - `status` - the HTTP code returned from the server
    - `statusText` - the HTTP status message returned by the server
    - `headers` - headers sent by server

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Criteria

- Imagine you already have a component tree that has several levels of components in its hierarchy.
- Fetching data.
  - Which level in your component hierarchy, to be more precise, which specific component, should fetch the data now?
  - Basically it depends on three criteria:
    - **Who is interested in this data?** The fetching component should be a common parent component for all these components.
    - **Where do you want to show a conditional loading indicator (e.g. loading spinner, progress bar) when the fetched data from the asynchronous request is pending?** The loading indicator could be shown in the common parent component from the first criteria. Then the common parent component would still be the component to fetch the data.
    - **Where do you want to show an optional error message when the request fails?** Here the same rules from the second criteria for the loading indicator apply.

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

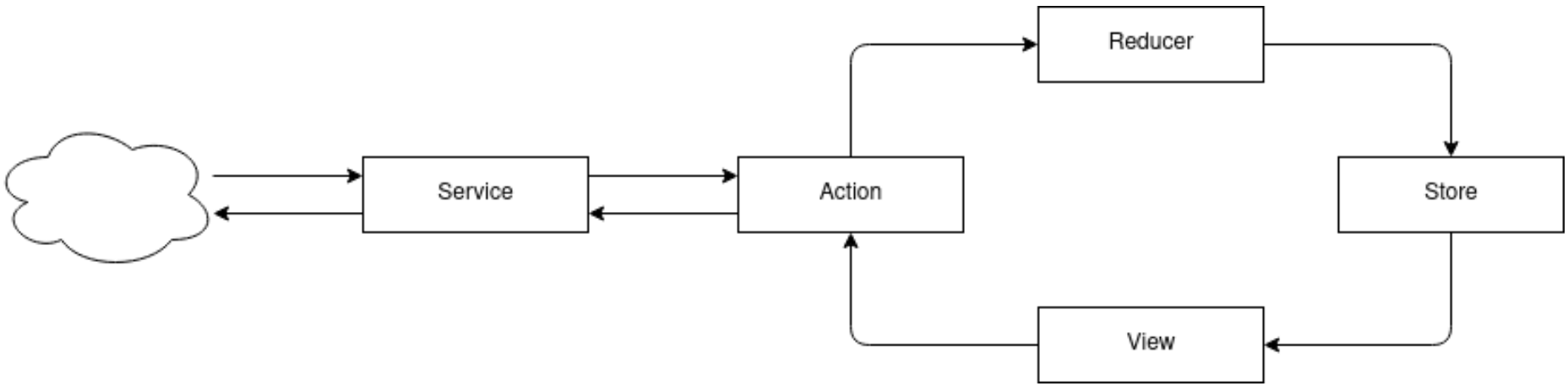
5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

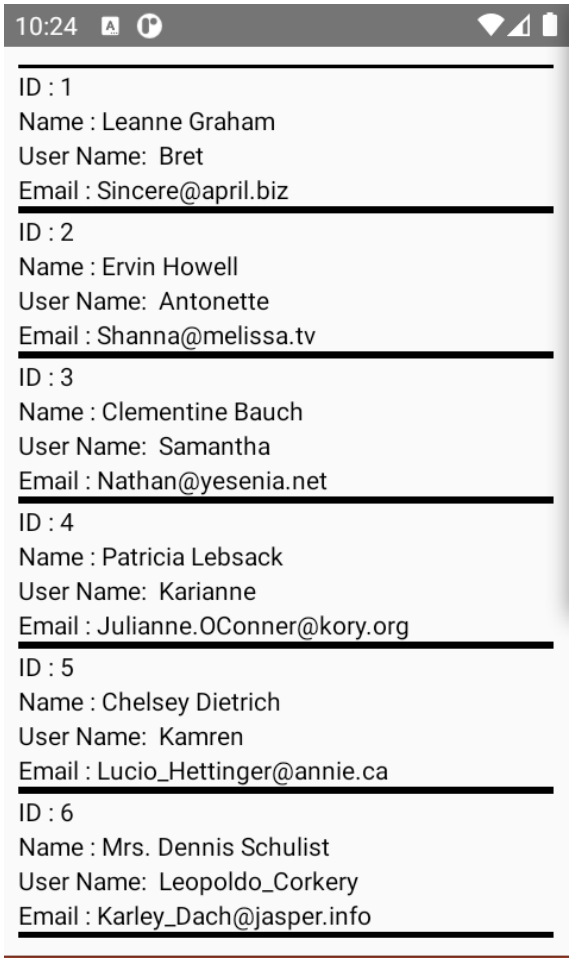
7 How To Organize React Native Project

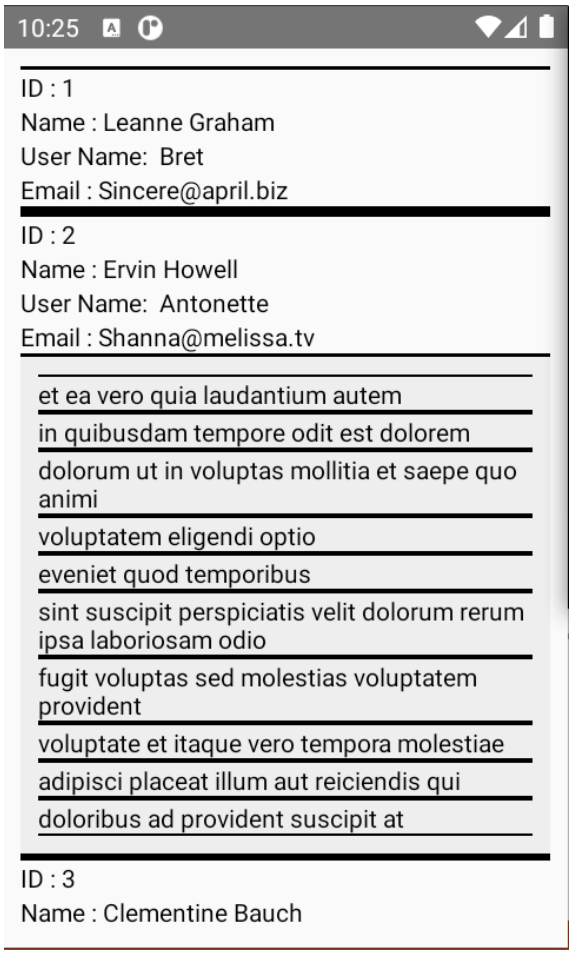
8 Bibliography

Flux Architecture

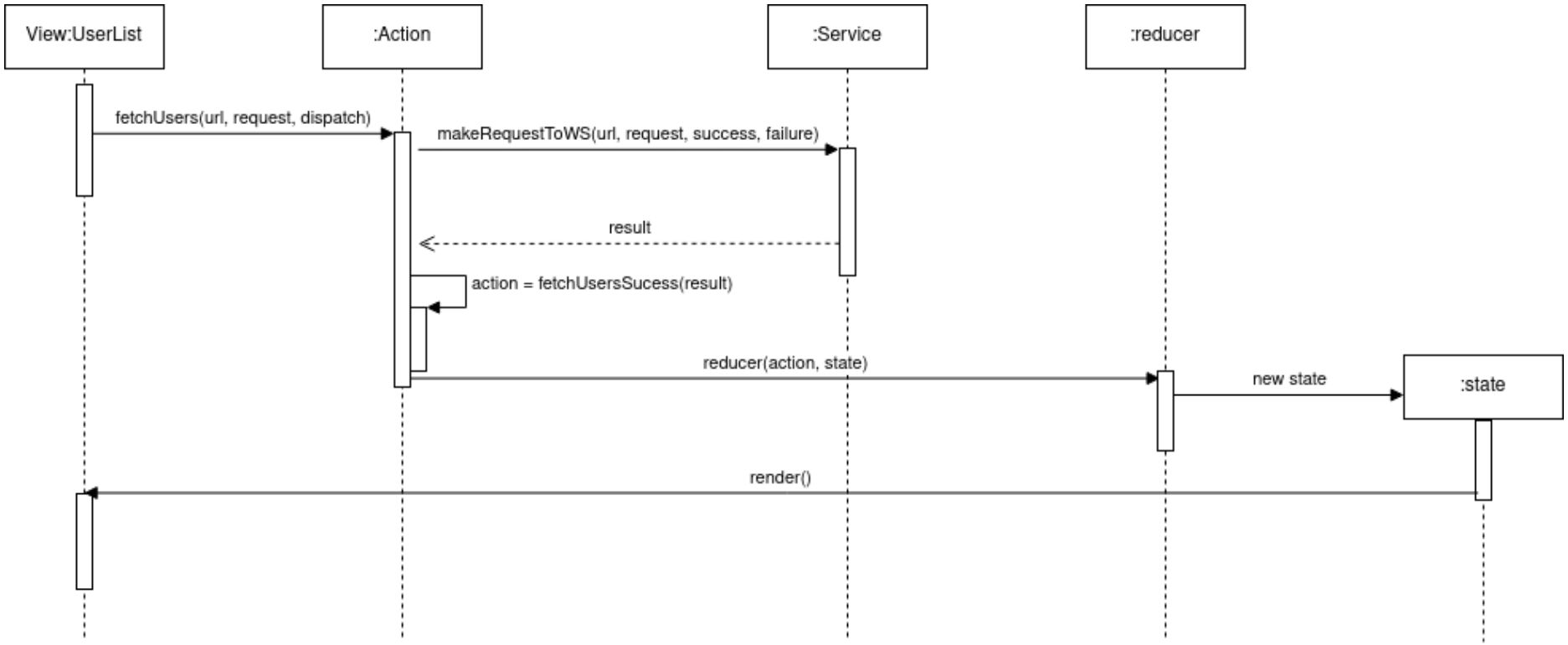


Output


10:24
ID : 1
Name : Leanne Graham
User Name: Bret
Email : Sincere@april.biz
ID : 2
Name : Ervin Howell
User Name: Antonette
Email : Shanna@melissa.tv
ID : 3
Name : Clementine Bauch
User Name: Samantha
Email : Nathan@yesenia.net
ID : 4
Name : Patricia Lebsack
User Name: Karianne
Email : Julianne.OConner@kory.org
ID : 5
Name : Chelsey Dietrich
User Name: Kamren
Email : Lucio_Hettinger@annie.ca
ID : 6
Name : Mrs. Dennis Schulist
User Name: Leopoldo_Corkery
Email : Karley_Dach@jasper.info


10:25
ID : 1
Name : Leanne Graham
User Name: Bret
Email : Sincere@april.biz
ID : 2
Name : Ervin Howell
User Name: Antonette
Email : Shanna@melissa.tv
et ea vero quia laudantium autem
in quibusdam tempore odit est dolorem
dolorum ut in voluptas mollitia et saepe quo animi
voluptatem eligendi optio
eveniet quod temporibus
sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio
fugit voluptas sed molestias voluptatem provident
voluptate et itaque vero tempora molestiae
adipisci placeat illum aut reiciendis qui
doloribus ad provident suscipit at
ID : 3
Name : Clementine Bauch

Fetch Users


View:UserList
:Action
:Service
:reducer
fetchUsers(url, request, dispatch)
makeRequestToWS(url, request, success, failure)
result
action = fetchUsersSucess(result)
reducer(action, state)
new state
:state
render()

UsersList

```
const UsersList = () => {
  const { state, dispatch } = useContext(AppContext);
  ...
  const fetchData = () =>{
    dispatch(fetchUsersStarted());
    const url = `${URL_API}/users`;
    const request = {};
    fetchUsers(url, request, dispatch);
  }
  useEffect(() => {
    fetchData();
  }, []);
  ...
}
```

Actions & Service

```
export function fetchUsers(url, request, dispatch) {
  //função ser executado em caso de sucesso
  const success = (result) => dispatch(fetchUsersSuccess(result));
  //função ser executado em caso de falha
  const failure = (error) => dispatch(fetchUsersFailure(error.message));
  makeRequestToWS(url, request, success, failure);
}
```

```
export function makeRequestToWS(url, request, success, failure) {
  fetch(url, request)
  .then(res => res.json())
  .then(result => success(result))
  .catch(error => failure(error.message))
  ;
}
```

Action & Reducer

```
export function fetchUsersSuccess(users) {
  return {
    type: FETCH_USERS_SUCCESS,
    payload: {
      data:
      [...users]
    }
  }
}
```

```
function reducer(state, action) {
  switch (action.type) {
    ...
    case FETCH_USERS_SUCCESS:
    return {
      ...state,
      users: {
        loading: false,
        error: null,
        data: [...action.payload.data]
      }
    }
  ...
```

UserList

```
const UsersList = () => {
...
  if (loading === true) {
    return (...);
  }
  else {
    if (error !== null) {
      return (...);
    } else {
      if (data.length > 0) {
        return (... );
      } else {
        return (...);
      }
    }
  }
}
```

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 **Navigation**
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Navigation

- React Navigation [1] is the standard routing solution for native apps
- React Navigation is made up of some core utilities and those are then used by navigators to create the navigation structure in your app.
- Required package:
  `npm install @react-navigation/native`
- Dependencies:
  `npm install react-native-screens react-native-safe-area-context`
- More at
  `https://reactnavigation.org/docs/getting-started`

[1] `https://reactnavigation.org/`

Navigation: Android (I)

- Configuring react-native-screens on Android
  - Edit `MainActivity.kt` file under `android/app/src/main/java/<yourpackagename>/`, and add the blue code to the body of `MainActivity` class:

```
...
import com.facebook.react.defaults.DefaultReactActivityDelegate
import android.os.Bundle
import com.swmansion.rnscreens.fragment.restoration.RNScreensFragmentFactory

class MainActivity : ReactActivity() {
...
  override fun createReactActivityDelegate(): ReactActivityDelegate =
  DefaultReactActivityDelegate(this, mainComponentName, fabricEnabled)

  override fun onCreate(savedInstanceState: Bundle?) {
    supportFragmentManager.fragmentFactory = RNScreensFragmentFactory()
    super.onCreate(savedInstanceState)
  }
}
```

Navigation: Android (II)

- Edit `AndroidManifest.xml` file under `android/app/src/main/`, and add the blue configuration option to the `<application>` tag:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application
  ...
  android:supportsRtl="true"
  android:enableOnBackInvokedCallback="false"
  >
    <activity android:name=".MainActivity"
    ...
    </activity>
  </application>
</manifest>
```

Navigation: iOS

- If you are on a Mac and developing for iOS, you need to install the pods (via Cocoapods) to complete the linking.
  - `npx pod-install ios`

NavigationContainer

- `NavigationContainer` is a component which manages the navigation tree and contains the navigation state.
- This component must wrap all navigators structure.

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

function App() {
  return (
    <NavigationContainer>
    {
    ...
    }
    </NavigationContainer>
  );
}

export default App;
```

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Stack (I)

- React Navigation's stack navigator provides a way for your app to transition between screens and manage navigation history.
- Required package:
  `npm install @react-navigation/native-stack`

Stack (II)

- `createNativeStackNavigator` is a function that returns an object containing 2 properties: `Screen` and `Navigator`
  - The `Navigator` should contain `Screen` elements as its children to define the configuration for routes.

```
...
import { createNativeStackNavigator } from '@react-navigation/stack';

import FirstPage from './FirstPage';
import SecondPage from './SecondPage';
function App() {
  const Stack = createStackNavigator();
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="FirstPage">
        <Stack.Screen
          name="FirstPage"
          component={FirstPage}/>
        <Stack.Screen
          name="SecondPage"
          component={SecondPage}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Stack (III)

- `navigation` prop that is passed down to every screen components in stack navigator.
- `navigate('SecondPage')`, the navigate function (on the `navigation` prop) specify the name of the route (the screen) to move to.

```
function FirstPage(props) {
  const { navigation } = props;
  return (
    <View>
      <Button
        title="Second Page"
        onPress={() => navigation.navigate('SecondPage')}/>
    </View >
  );
}
```

Stack (IV)

- Passing parameters

```
function FirstPage(props) {
  const { navigation } = props;
  return (
    <View>
      <Button
      title="Second Page"
      onPress={() => navigation.navigate('SecondPage',{data: 'some data'})}/>
    </View >
  );
}
```

- Read parameters

```
function SecondPage(props) {
  const { navigation, route } = props;
  const { data } = route.params;
  return (...);
}
```

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

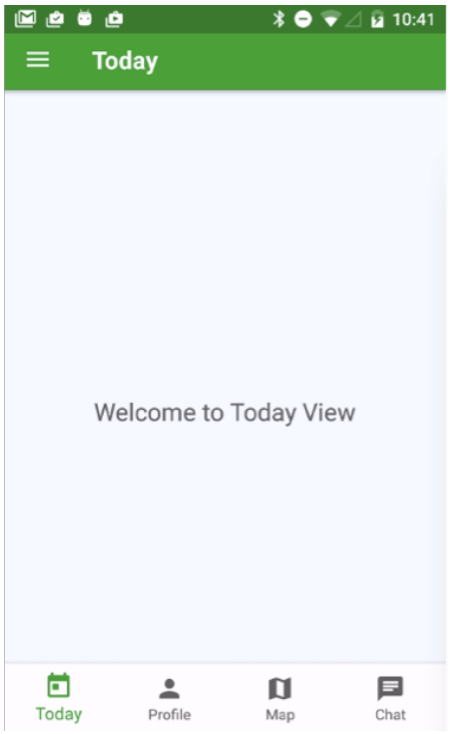
6 Navigation
- Stack
- Tab
- Drawer

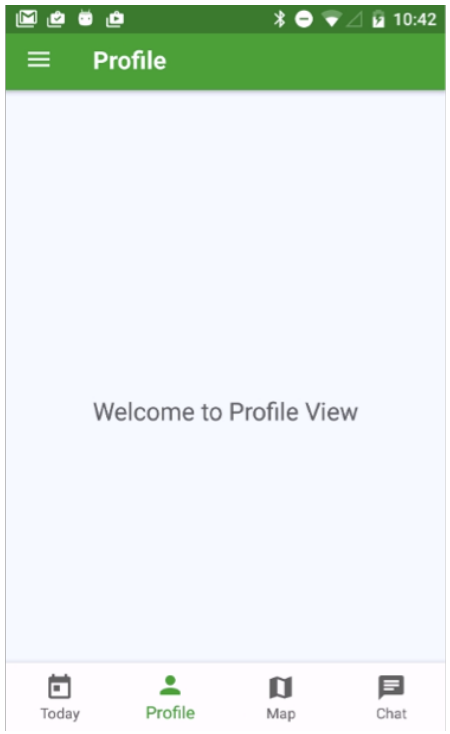
7 How To Organize React Native Project

8 Bibliography

Tab (I)

- React Navigation's tab navigator
- Required package:
  `npm install @react-navigation/bottom-tabs`





Tab (II)

```
  ...
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
const Tab = createBottomTabNavigator();
import FirstPage from './FirstPage';
import SecondPage from './SecondPage';
function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator >
        <Tab.Screen
          name="FirstPage"
          component={FirstPage}
        />
        <Tab.Screen
          name="SecondPage"
          component={SecondPage}
        />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

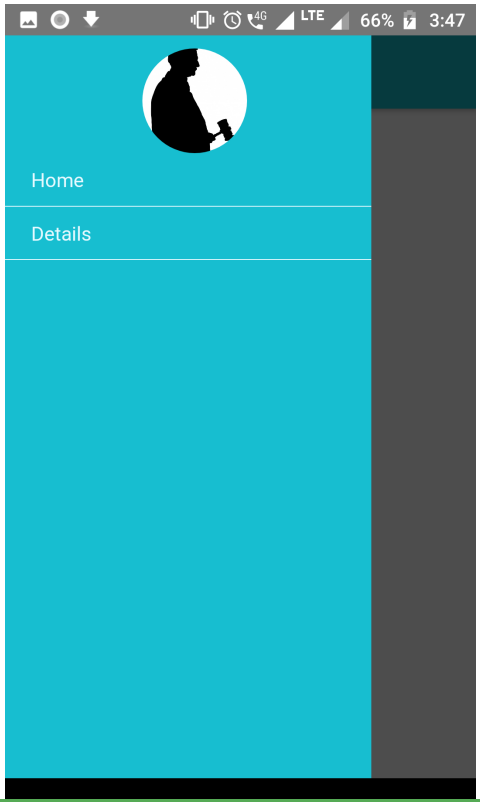
1 Conditional Rendering
2 Fetching data from REST API
3 Axios
4 Where to fetch in React's component tree?
5 Flux with Networking
6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project
8 Bibliography

Drawer (I)

- React Navigation's drawer navigator
- Required packages:

```
npm install @react-navigation/drawer
npm install react-native-gesture-handler
react-native-reanimated react-native-worklets
```



Drawer (II)

- Add worklets' babel plugin by updating `babel.config.js` file

```
module.exports = {
  presets: [
  'module:@react-native/babel-preset'
  ],
  plugins: [
  'react-native-worklets/plugin'
  ],
};
```

Drawer (III)

```
...
import 'react-native-gesture-handler';
import { createDrawerNavigator } from '@react-navigation/drawer';
const Drawer = createDrawerNavigator();

import FirstPage from './FirstPage';
import SecondPage from './SecondPage';

function App() {
return (
  <NavigationContainer>
  <Drawer.Navigator initialRouteName="FirstPage">
    <Drawer.Screen
      name="FirstPage"
      component={FirstPage}
    />
    <Drawer.Screen
      name="SecondPage"
      component={SecondPage}
    />
    </Drawer.Navigator>
  </NavigationContainer>
  );
}

export default App;
```

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Folders

- Each developer has it own way of organizing a project and there is no right or wrong way.
- A good approach for organizing you project could encompass the following folders:
  - `assets`
    - It is used to store images, logos, etc.
  - `context`
    - It includes reducers, actions, and store (context).
  - `components`
    - It includes all components.
  - `screens`
    - It includes all the screens that we can navigate to.
    - On each screen, we can use the components that we build on the components folder.
  - `services`
    - It includes all functions to fetch data from any API.
  - `navigation` (or `routes`)
    - It includes the routes structure of our application.
    - These routes will reflect the screens that we can navigate to.

1 Conditional Rendering

2 Fetching data from REST API

3 Axios

4 Where to fetch in React's component tree?

5 Flux with Networking

6 Navigation
- Stack
- Tab
- Drawer

7 How To Organize React Native Project

8 Bibliography

Resources

- David Flanagan, "JavaScript: The Definitive Guide", O'Reilly Media, Inc., 2020
- Adam Boduch and Roy Derks, "React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition",Packt Publishing, 2020
- Dan Ward, "React Native Cookbook Second Edition Step-by-step recipes for solving common React Native development problems", ,Packt Publishing, 2019
- `https://reactnative.dev/`
- `https://reactjs.org/`
- `https://reactnavigation.org/`