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

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

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

Outline

1 React Native

2 Setup

3 Let’s start

4 Components

5 Lifecycle

6 Bibliography

React Native

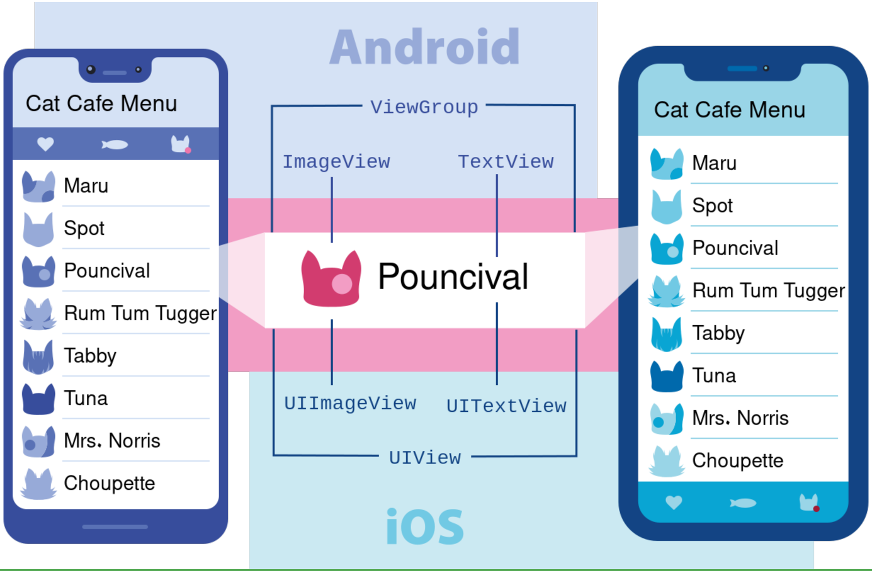
Overview

- Facebook created React Native to build its mobile applications.
- **React Native is an open source framework for building Android and iOS applications** using React and the app platform's native capabilities.
- With React Native, you use **JavaScript** to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components.

Views (I)

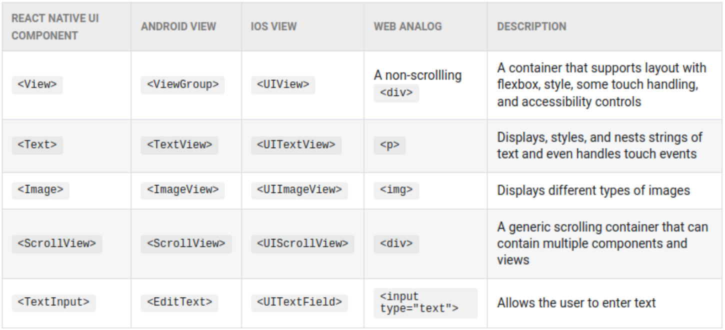
- In Android and iOS development, a `View` is the basic building block of UI:
  - A small rectangular element on the screen which can be used to display text, images, or respond to user input.
- Even the smallest visual elements of an app, like a line of text or a button, are kinds of views.
- Some kinds of views can contain other views.

Views (II)


Android
ViewGroup
ImageView
TextView
Pouncival
UIImageView
UITextView
UIView
iOS
Cat Cafe Menu
Maru
Spot
Pouncival
Rum Tum Tugger
Tabby
Tuna
Mrs. Norris
Choupette

Native Components

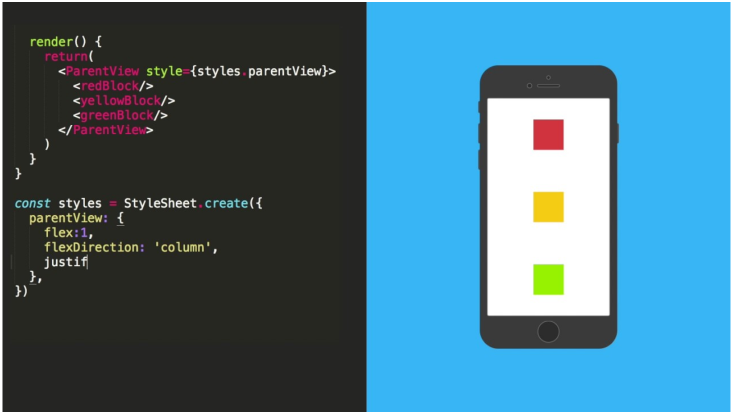
- In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C.
- With React Native, you can invoke these views with JavaScript using React components.
- At runtime, React Native creates the corresponding Android and iOS views for those components.



REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrollling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Layouts [1]

- FlexBox



```
render() {
  return(
    <ParentView style={styles.parentView}>
      <redBlock/>
      <yellowBlock/>
      <greenBlock/>
    </ParentView>
  )
 }
}

const styles = StyleSheet.create({
  parentView: {
    flex:1,
    flexDirection: 'column',
    justif
  },
})
```

[1] https://reactnative.dev/docs/flexbox

Setup

Links

- Setting up the development environment
  - `https://reactnative.dev/docs/environment-setup`
- Running On Device
  - `https://reactnative.dev/docs/running-on-device`

Let’s start

Create React Native App

- React Native has a built-in command line interface, which you can use to generate a new project.
  - `npm` - Manages Node packages.
  - `npx` - A tool for executing Node packages.
  - `nvm` - Node Version Manager
- Create an app

```
npx @react-native-community/cli@latest init
FirstApp
cd FirstApp
```

- Set the emulator (Fixing errors)

```
npx react-native doctor
```

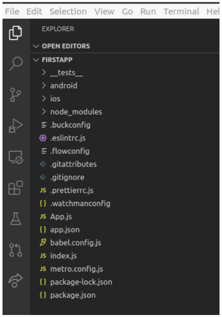
- Launch Metro

```
npm start
```

- Launch app in Android device (in another terminal)

```
npm run android
```

Create React App: Structure (I)



```
> __tests__
> android
> ios
> node_modules
.buckconfig
.eslintrc.js
.flowconfig
.gitattributes
.gitignore
.prettierrc.js
.watchmanconfig
App.js
app.json
babel.config.js
index.js
metro.config.js
package-lock.json
package.json
```

- Folders:
  - `node_modules`: Javascript library
  - `android`: Android project.
  - `ios`: iOS project.
  - `__tests__`: Tests.
- JavaScript files:
  - `App.js`: The main component.
  - `index.js`: The main file of our application where the components are registered.
- Configuration files
  - `package.json` and `package-lock.json` : Node.js configuration files.
  - `.gitignore` and `.gitattributes`: for git.

Create React App: Structure (II)

- Configuration files (cont.)
  - `.prettierrc.js`: Code formatter.
  - `babel.config.js`: The configuration file for Babel (A compiler and transpiler for JavaScript)
  - `metro.config.js`: The configuration file for Metro, a JavaScript bundler for React Native,
  - `app.json`: it is used to configure many things such as app name, icon, splash screen, deep linking scheme, API keys to use for some services and so on.
  - `watchmanconfig`: The configuration file for Watchman, a file watch service. This service watches files and records when they change. It can also trigger actions (such as rebuilding assets) when matching files change.
  - `.eslintrc.js`: The configuration file for ESLint, a JavaScript and JSX linter (a tool for code quality).
  - `tsconfig.json` file specifies Typescript project configuration.
  - `Gemfile` is a file that is created to describe the `gem` dependencies required to run a Ruby program.

Entry point

- index.js

```
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

  - `AppRegistry` is the JS entry point to running all React Native apps.
  - App root components should register themselves with `AppRegistry.registerComponent`
  - The native system can load the bundle for the app and then actually run the app when it’s ready by invoking `AppRegistry.runApplication`.

Components

App Component

```
import React, {Component} from 'react';
import {View,Text} from 'react-native';
class App extends Component {
 constructor(props) {
  super(props);
  this.state = {
  title: "DSSMV React-native example"
  };
 }
 render() {
  const title = this.state.title;
  return (
   <View>
    <Text>{title}</Text>
   </View>
  );
 }
}
export default App;
```

Class

```
import React, {Component} from 'react';

class App extends Component {
  ...
}
export default App;
```

- The `App` class extends from `Component`.
- The `Component` has all the functionalities that a component in React needs to have.

constructor, props & state

```
class App extends Component {
 constructor(props) {
  super(props);
  this.state = {
   title: "DSSMV React-native example"
  };
 }
 ...
 const title = this.state.title;
}
```

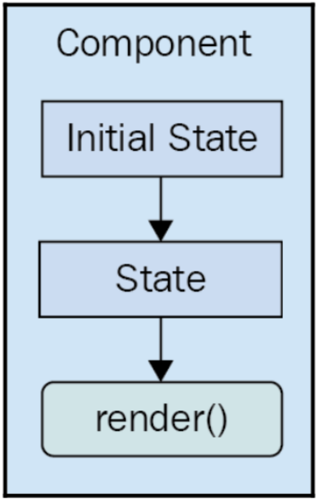
- The `constructor` is called only once when the component initializes.
  - The `props` stands for properties
    - `props` get passed to the component (similar to function parameters, i.e., declared within a function).
  - The `state`
    - `state` is managed within the component (similar to variables declared within a function).
  - `props` and `state` are both plain **JavaScript objects**

render (I)

```
class App extends Component {
...
render() {
  const title = this.state.title;
  return (
    <View>
      <Text>{title}</Text>
    </View>
    )
  }
}
```

- The `render` method
  - It defines the output of a React Component.
  - It should examine `props` and `state`
  - Whatever it returns is rendered as a React element
    - Inside of the `return` statement, the JSX code must be wrapped by { and }.

`render` (II)



Counter component

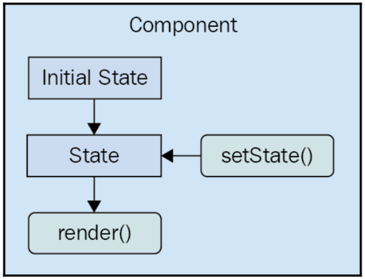
```
import React, { Component } from 'react';
import { View, Text, Button } from 'react-native';
class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  handleClick = () => {
    let val = this.state.count;
    val = val + 1;
    this.setState({ count: val })
  }
  render() {
    const count = this.state.count;
    return (
      <View >
        <Button onPress={this.handleClick} title="Click here" />
        <Text>{count}</Text>
      </View>
    );
  }
}
export default Counter;
```

setState (I)

```
class Counter extends Component {
...
 handleClick = () => {
   let val = this.state.count;
   val = val + 1;
   this.setState({count: val })
 }
 render() {
   const count = this.state.count;
   return (
     <View >
       <Button onPress={this.handleClick} title="Click here" />
       <Text>{count}</Text>
     </View>
 );
 }
}
```

- `setState()` method will always lead to a re-render
  - The method `render` will be invoked

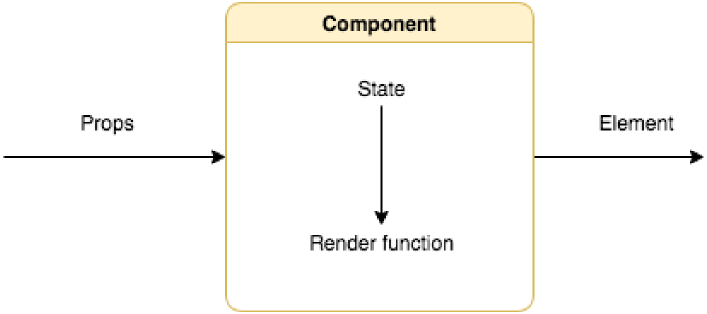
`setState` (II)



- This is called a **change in state**, and whenever you tell a React component to change its state, the component will automatically re-render itself, calling `render`
- The state of a component is something that either the **component itself can set**, or **other pieces of code, outside of the component**.

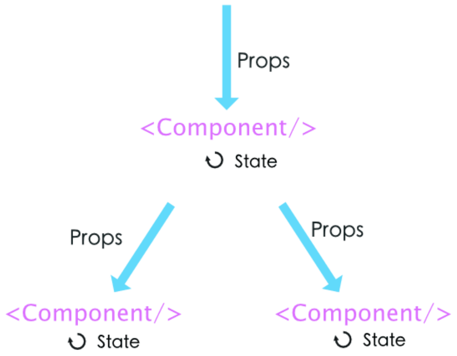
props

- Properties (`props`) are like state data that gets passed into components.
- `props` are like arguments to a function.



`props` *vs* `state`

- `state`
  - Should have an initial value.
  - Read and write (mutable)
- `props`
  - Passed from a parent component.
  - Read-only (unmutable)



App component: props for passing data to child (I)

```
import React, { Component } from 'react';
import PeopleList from './PeopleList';
const data = [
  {name: 'Charlie',job: 'Janitor'},
  {name: 'Mac', job: 'Bouncer'},
  {name: 'Dee', job: 'Aspring actress'},
  {name: 'Dennis',job: 'Bartender'}
  ];
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      people: data,
    };
  }
  render() {
    const {people} = this.state;
    return (
      <View>
        <PeopleList listOfItems={people} />
      </View>
    );
  }
}
```

PeopleList component: props for passing data to child (II)

```
import React, { Component } from 'react';
import { FlatList, Text } from 'react-native';
import PeopleListItem from './PeopleListItem';

class PeopleList extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    const people = this.props.listOfItems;
    return (
      <FlatList
        data={people}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
        <PeopleListItem
          id={item.id}
          name={item.name}
          job={item.job}
        />
      )}
      />
    );
  }
}
export default PeopleList;
```

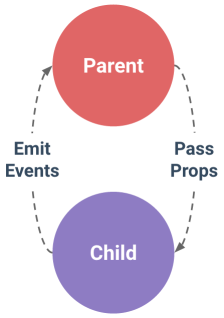
PeopleListItem component: props for passing data to child (III)

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

class PeopleListItem extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    const {name, job} = this.props;
    return (
      <View>
        <Text>{name}, {job}</Text>
      </View>
    );
  }
}
export default PeopleListItem;
```

Child passing data to Parent

- Passing the data from the child to parent component is a bit trickier (emit events):
  - Create a **callback function in the parent** component.
    - This callback function will get the data from the child component.
  - Pass the **callback function in the parent as a prop** to the child component.
  - The **child component calls the parent callback function** using props.



App component: Child passing data to Parent (I)

```
  ...
class App extends Component {
  ...
  deletePerson = (name) => {
    const { people } = this.state;
    let newList = [...people];
    newList = newList.filter((item) => (item.name.trim() !== name.trim()));
    this.setState({people: newList,})
  }
  render() {
    const {people} = this.state;
    return (
    <View>
      ...
      <PeopleList listOfItems={people} deleteItem={this.deletePerson} />
    </View>
    );
  }
}
export default App;
```

PeopleList component: Child passing data to Parent (II)

```
...
class PeopleList extends Component {
  ...
  handleClick = (name) => {
    this.props.deleteItem(name);
  }
  render() {
    const people = this.props.listOfItems;
    return (
      <FlatList
      ...
      renderItem={({ item }) => (
      <PeopleListItem
        id={item.id}
        name={item.name}
        job={item.job}
        handleClick={this.handleClick}
      />)
      }
    />
    );
  }
}
export default PeopleList;
```

`PeopleListItem` component: Child passing data to Parent (III)

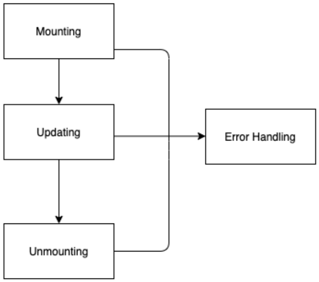
```
import React, { Component } from 'react';
import { View, Text, Button } from 'react-native';

class PeopleListItem extends Component {
  ...
  handleClick = (name) => {
    this.props.handleClick(name);
  }
  render() {
    const { name, job } = this.props;
    return (
      <View>
        <Text>{name}, {job}</Text>
        <Button onPress={() => this.handleClick(name)} title="Delete" />
      </View >
    );
  }
}
export default PeopleListItem;
```

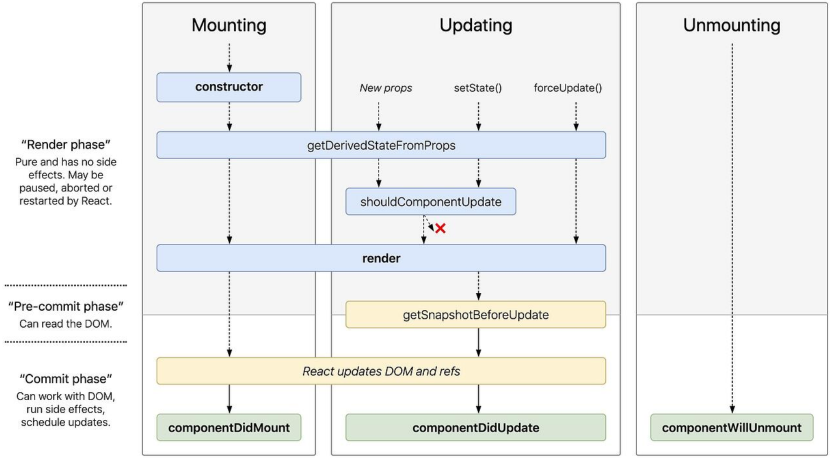
Lifecycle

Lifecycle Phases

- A component's life cycle in React Native can be divided into 4 phases:
  - **Mounting**: Component instance is created and inserted into the UI.
  - **Updating**: Component is said to be born and it start growing by receiving new updates.
  - **Unmounting**: Component gets removed from actual UI.
  - **Error Handling**: It is called when any error occurs while rendering the component.



Methods



Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-commit phase"
Can read the DOM.
"Commit phase"
Can work with DOM, run side effects, schedule updates.

Methods (I)

- `constructor()`
  - Initializing local state by assigning an object to `this.state`.
  - No UI rendering is done.
  - It receives props as an argument.
    - `setState()` method must be invoked in the `constructor()`.
- `render()`
  - It tells what to display on the screen.
  - It is pure function which means it does not modify the `state`.
- `ComponentDidMount()`
  - This method gets called when react native component has finished the rendering part.
  - It is good place to load data from remote endpoint and update the `state` with the result.

Methods (II)

- `componentDidUpdate()`
  - It is invoked immediately after re-rendering of the component gets completed.
- `componentWillUnmount()`
  - It gets invoked when component is removed from the UI.
  - We can perform clean up tasks in this method like invalidating timers, cancelling ongoing network request.

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/`