

# React Native Fundamentals

## **Full Stack Bootcamp (Day 2)**

# Agenda

- Day 1
  - ES6+
- **Day 2**
  - **React Native**
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

# What's React Native?

- React Native is a version of React that creates native based information
- React Native (also known as RN) is a popular JavaScript-based mobile app framework that allows you to build natively-rendered mobile apps for iOS and Android. The framework lets you create an application for various platforms by using the same codebase.
- Today React Native is available today to generate web, Android and iOS
- Works with NodeJS

# Who use React Native?

- Facebook
- Instagram
- Walmart
- Airbnb
- Tesla
- Adidas
- Discord
- Bloomberg

# How does React works

- JavaScript is bundled with an interpret called Babel
- Uses separate threads for Javascript, UI and layout
- Threads communicate asynchronously using a native bridge

# Setting up the environment

- Go to <https://nodejs.org/en/download/> and download and install it in your computer

Latest LTS Version: **16.15.1** (includes npm 8.11.0)

Download the Node.js source code or a pre-built installer for your platform

**LTS**  
Recommended For Most Users

**Current**  
Latest Features

**Windows Installer**  
node-v16.15.1-x64.msi

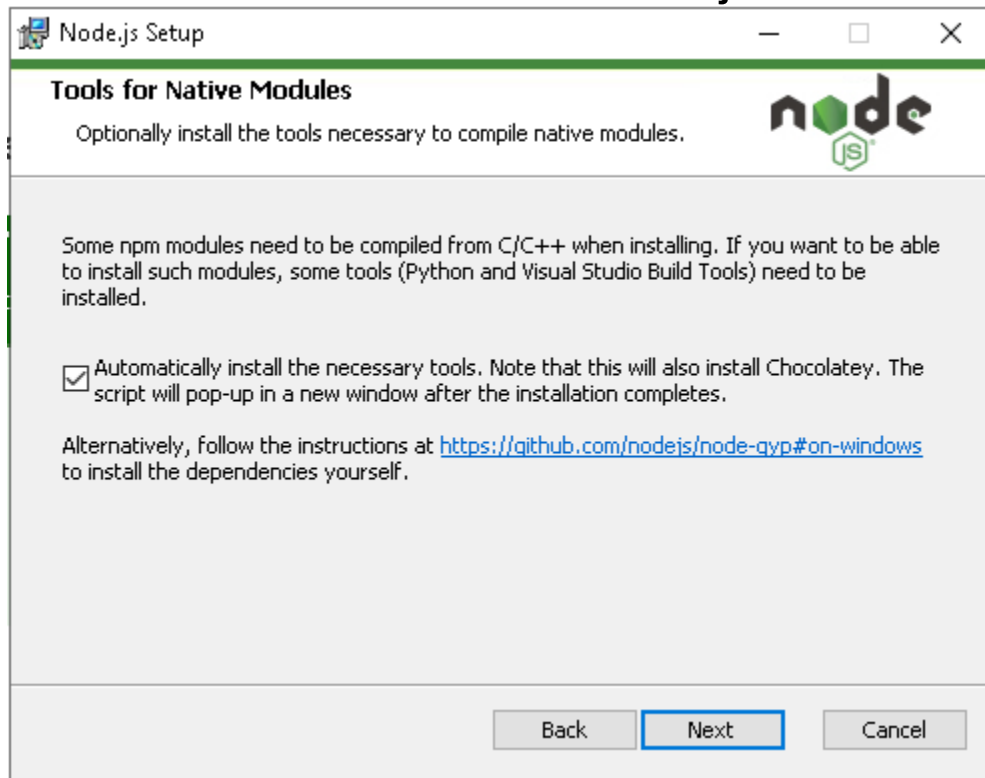
**macOS Installer**  
node-v16.15.1.pkg

**Source Code**  
node-v16.15.1.tar.gz

<b>Windows Installer (.msi)</b>	32-bit	64-bit
<b>Windows Binary (.zip)</b>	32-bit	64-bit

# Setting up the environment

Be sure to select the checkbox to install nodejs tools.





# Setting up the environment

- Install expo app if you intent to test your app on Android/iOS
- use npm to install the expo CLI command line utility:

```
npm install -g expo-cli
expo init AwesomeProject #select tabs?

cd AwesomeProject
npm start # you can also use: expo start
```

C:\> npm install expo-cli

or on the command line via: npm <command> --key=value

More configuration info: npm help config

Configuration fields: npm help 7 config

npm@8.11.0 C:\Program Files\nodejs\node\_modules\npm

C:\Users\charly>npm install -g expo-cli

npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

npm WARN deprecated source-map-url@0.4.1: See <https://github.com/lydell/source-map-url#deprecated>

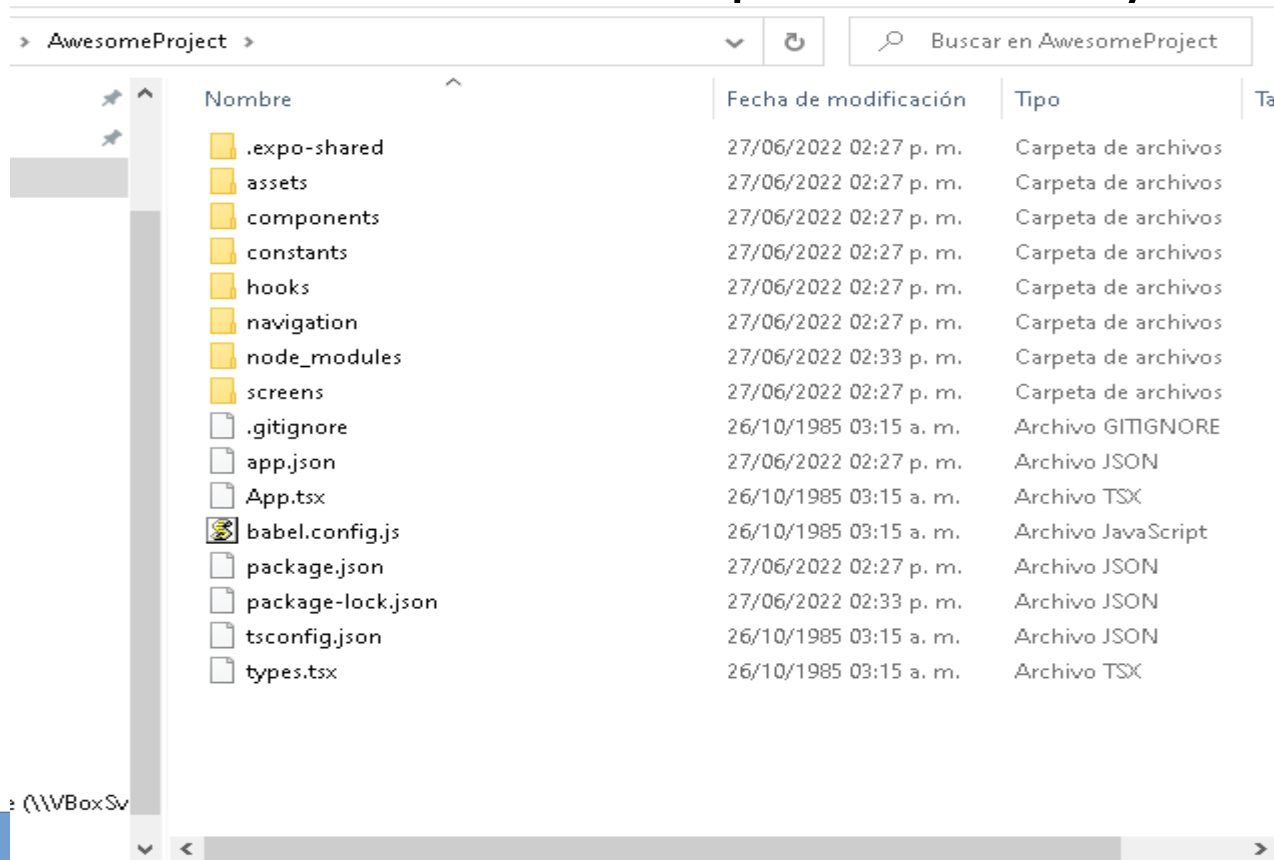
npm WARN deprecated urix@0.1.0: Please see <https://github.com/lydell/urix#deprecated>

npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Object)



# The app

- Look out on the AwesomeProject directory



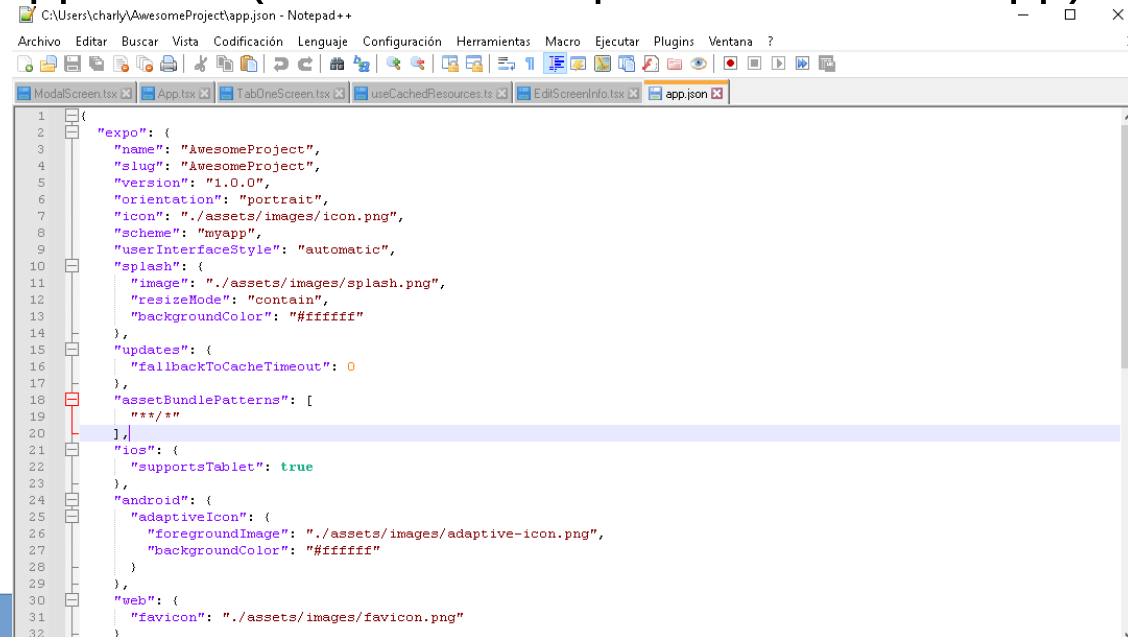
Nombre	Fecha de modificación	Tipo
.expo-shared	27/06/2022 02:27 p. m.	Carpeta de archivos
assets	27/06/2022 02:27 p. m.	Carpeta de archivos
components	27/06/2022 02:27 p. m.	Carpeta de archivos
constants	27/06/2022 02:27 p. m.	Carpeta de archivos
hooks	27/06/2022 02:27 p. m.	Carpeta de archivos
navigation	27/06/2022 02:27 p. m.	Carpeta de archivos
node_modules	27/06/2022 02:33 p. m.	Carpeta de archivos
screens	27/06/2022 02:27 p. m.	Carpeta de archivos
.gitignore	26/10/1985 03:15 a. m.	Archivo GITIGNORE
app.json	27/06/2022 02:27 p. m.	Archivo JSON
App.tsx	26/10/1985 03:15 a. m.	Archivo TSX
babel.config.js	26/10/1985 03:15 a. m.	Archivo JavaScript
package.json	27/06/2022 02:27 p. m.	Archivo JSON
package-lock.json	27/06/2022 02:33 p. m.	Archivo JSON
tsconfig.json	26/10/1985 03:15 a. m.	Archivo JSON
types.tsx	26/10/1985 03:15 a. m.	Archivo TSX

# Excercise 1

Starting the trip

# App.json

- It's the application configuration file
- You set things like: app version, icon, orientation of the app (default, portrait or landscape), splash image among others
- It's loaded at the time the app starts (the react interpreter loads the app)



```
1 {
2   "expo": {
3     "name": "AwesomeProject",
4     "slug": "AwesomeProject",
5     "version": "1.0.0",
6     "orientation": "portrait",
7     "icon": "./assets/images/icon.png",
8     "scheme": "myapp",
9     "userInterfaceStyle": "automatic",
10    "splash": {
11      "image": "./assets/images/splash.png",
12      "resizeMode": "contain",
13      "backgroundColor": "#ffffff"
14    },
15    "updates": {
16      "fallbackToCacheTimeout": 0
17    },
18    "assetBundlePatterns": [
19      "**/*"
20    ],
21    "ios": {
22      "supportsTablet": true
23    },
24    "android": {
25      "adaptiveIcon": {
26        "foregroundImage": "./assets/images/adaptive-icon.png",
27        "backgroundColor": "#ffffff"
28      }
29    },
30    "web": {
31      "favicon": "./assets/images/favicon.png"
32    }
33  }
```

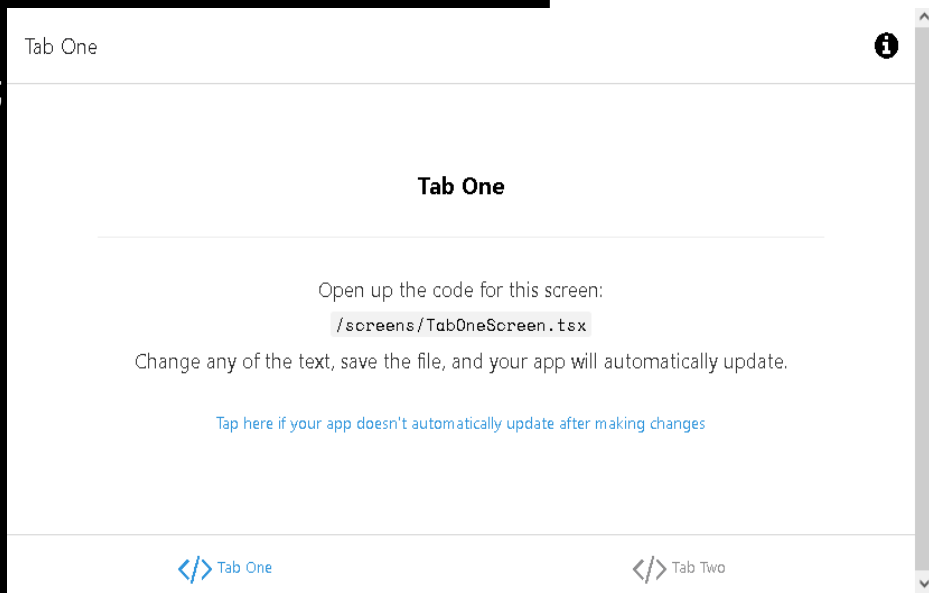
# screens

```
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider } from 'react-native-safe-area-context';

import useCachedResources from './hooks/useCachedResources';
import useColorScheme from './hooks/useColorScheme';
import Navigation from './navigation';

export default function App() {
  const isLoadingComplete = useCachedResources();
  const colorScheme = useColorScheme();

  if (!isLoadingComplete) {
    return null;
  } else {
    return (
      <SafeAreaProvider>
        <Navigation colorScheme={colorScheme} />
        <StatusBar />
      </SafeAreaProvider>
    );
  }
}
```



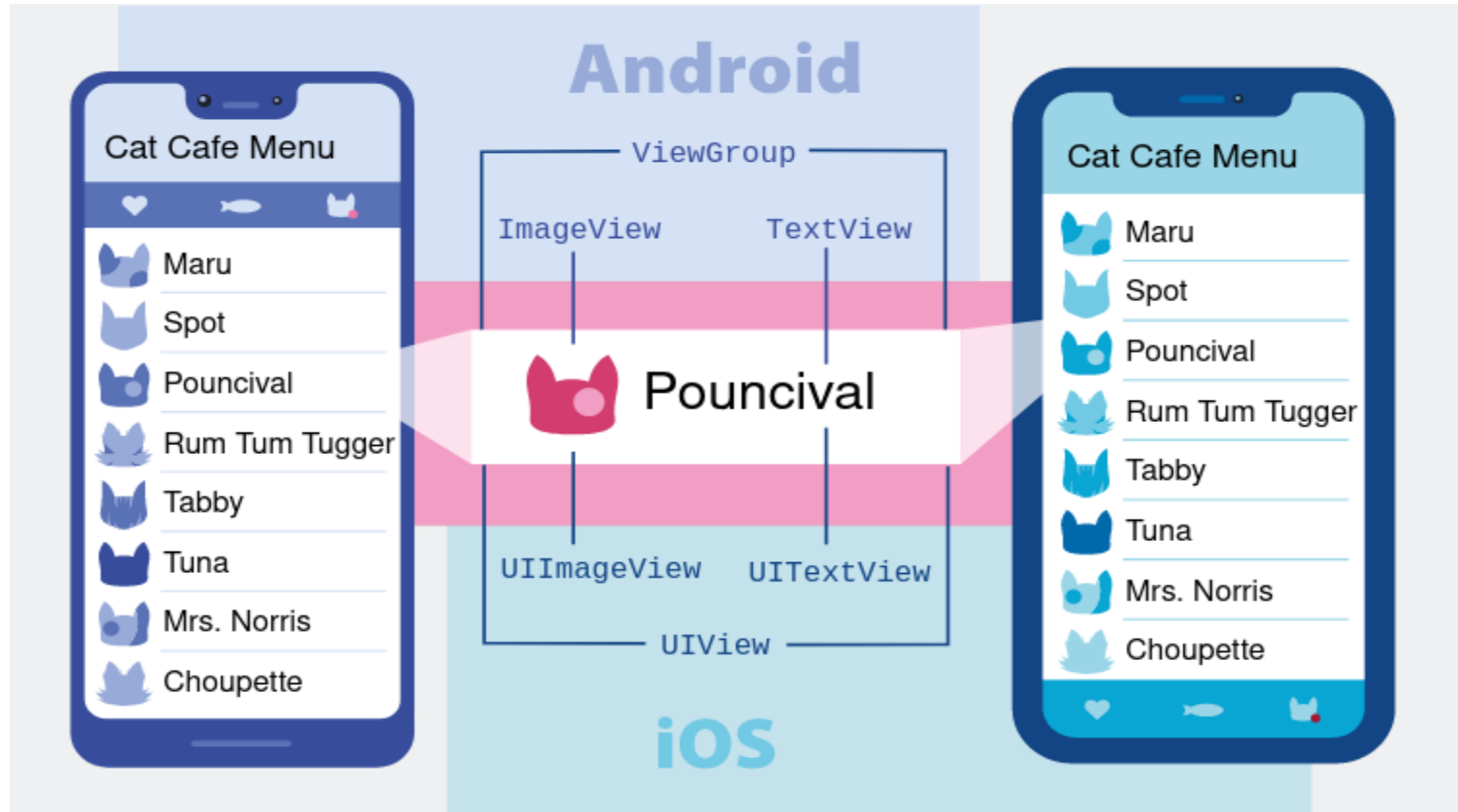
# Screens tab1

```
import { StyleSheet } from 'react-native';
import EditScreenInfo from '../components/EditScreenInfo';
import { Text, View } from '../components/Themed';
import { RootTabScreenProps } from '../types';
export default function TabOneScreen({ navigation }:
RootTabScreenProps<'TabOne'>) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Tab One</Text>
      <View style={styles.separator} lightColor="#eee"
darkColor="rgba(255,255,255,0.1)" />
      <EditScreenInfo path="/screens/TabOneScreen.tsx" />
    </View>
  );
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  // ...
});
```

# Core components

React offers some out of the box components such as:

- Button
- Views
- Text
- Image
- TextInput
- ScrollView
- StyleSheet
- Switch
- FlatList
- SectionList
- Modal



# Hello World

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

const App = () => {
  return (
    <ScrollView>
      <Text>Some text</Text>
      <View>
        <Text>Hello World</Text>
        <Image
          source={{
            uri:
'/assets/p_cat2.png',
```

```
      }}
      style={{ width: 200, height:
200 }}
    </View>
    <TextInput
      style={{
        height: 40,
        borderColor: 'gray',
        borderWidth: 1
      }}
      defaultValue="write here"
    </ScrollView>
```



# Components, Hello World component

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

const Cat = () => {
  return (
    <Text>Hello World!</Text>
  );
}

export default Cat;
```

# JSX

JSX is a combination of JavaScript and XML sublanguage that emulates HTML in several ways, their tags are used to place components along the GUI generated on React GUI

```
<CustomButton color="red" shadowSize={2}>  
  My Personal button  
</CustomButton>
```

Is translated into:

```
React.createElement(  
  CustomButton,  
  {color: 'red', shadowSize: 2},  
  'My Personal button'  
)
```

# Props

Properties can be defined for the component and used on their own tags, for example:

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

```
const Cat = (props) => {
  return (
    <View>
      <Text>Hello, I am
{props.name}</Text>
    </View>
  );
}
```

```
const Cafe = () => {
  return (
    <View>
      <Cat name="Maru" />
```

```
        <Cat
name="Jellylorum" />
```

```
        <Cat name="Spot" />
      </View>
    );
  }
}
```

```
export default Cafe;
```

# Text Input

**TextInput** and **Text** are 2 great components to show how interaction is achieved on React

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

const HamburgerEater = () => {
  const [text, setText] =
useState('');
  return (
    <View style={{padding: 10}}>
      <TextInput
        style={{height: 40}}
        placeholder="Type here to
eat!"
        onChangeText={newText =>
setText(newText)}
```

```
        defaultValue={text}
      />
      <Text style={{padding: 10,
fontSize: 42}}>
        {text.split(' ').map((word)
=> word && ' ').join(' ')}
      </Text>
    </View>
  );
}

export default HamburgerEater;
```

# Stylesheets

React allows to create an Stylesheet object, the main reason to use this is the validation of the properties names that React makes.

```
export default function App(){
  return(
    <SafeAreaView style={[styles.container, areaViewStyle]}>
      <Button title= "Press me"
        onPress={() => console.log("pressed");}>
    </SafeAreaView>
  );
  const areaViewStyle = {backgroundColor: "green"};
  const styles = StyleSheet.create({
    container: {
      flex: 1,
      backgroundColor: "#fff",
      justifyContent: "center",
      alignItems: "center",
    },
  });
}
```

# Exercise 2

TextInput for everybody

# Platform object

In order to create specific OS behaviour you can use the platform object in order to take action with the information of the object.

```
import { ..., Platform } from 'react-native';
const styles=StyleSheet.create({
  container:{
    flex:1,
    backgroundColor:"#fff",
    paddingTop: Platform.OS === "android" ? 20:0,
    alignItems:"center",
  },
});
```



# Exercise 2

TextInput for everybody

# Image

To put an Image you can use the Image and Image background objects

```
return (  
  <ImageBackground source={...} style={{width: '100%', height: '100%'}}>  
    <Image source={require('./my-icon.png')} />  
    <Image  
      source={{  
        uri: 'https://reactjs.org/logo-og.png',  
        cache: 'only-if-cached'  
      }}  
      style={{ width: 400, height: 400 }}  
    />  
    <Image source={{ uri: 'something.jpg' }} />  
  
  </ImageBackground>  
);
```

# Exercise 3 and 4

Images and searches.

# Layouts - Introduction

Layouts are the way the information is going to be presented on the final screen.

In order to do that we will talk about:

- Dimensions
- Device orientation
- Flexbox
- Absolute and Relative position

# Dimensions

Inside of our styles we can set height and width sizes as well as other related properties such as padding.

Those properties are on DIPS (Density Independent Pixels)

```
import { ..., View } from 'react-native';
export default function App()
  Console.log(Dimensions.get("screen"));
  return (
    <SafeAreaView style={styles.container}>{const styles=StyleSheet.create({
      container:{
        flex:1,
        backgroundColor:"#fff",
        paddingTop: Platform.OS === "android" ? 20:0,
        alignItems:"center",
      },
    });
  );
```

# Orientation changes

Since Dimensions object doesn't reload the properties of width and height we must use an external library, in order to do that we use npm to install our libraries:

```
npm i @react-native-community/hooks
```

```
import {useDimensions} from "@react-native-community/hooks"
export default function App()
  Console.log(useDimensions);
  return (
    <SafeAreaView style={styles.container}>{const styles=StyleSheet.create({
      container:{
        flex:1,
        backgroundColor:"#fff",
        paddingTop: Platform.OS === "android" ? 20:0,
        alignItems:"center",
      },
    });
  );
```

# Flexbox

If you don't want to be calculating all the widths and heights then you can use Flex, imagine that you have a view and inside of it you have several components, with flex you will be given each component a weight on how much space it will take.

```
<SafeAreaView style={styles.container}>
  <View style={styles.container}, {{backgroundColor="dodgerblue"}}/>
  <View style={styles.container}, {{backgroundColor="red"}}/>
  <View style={styles.container}, {{flex:3,
backgroundColor="orange"}}/>
</SafeAreaView>
{const styles=StyleSheet.create({
  container:{
    flex:1, backgroundColor:"#fff",    paddingTop: Platform.OS ==
"android" ? 20:0,alignItems:"center",
  },
});
```



# Flexbox

When working with flex we usually don't want to have all "stacked" (one component below the other), that's why flex provides a way to design the direction it will take.

```
<SafeAreaView style={styles.container}, {{flexDirection="column",
justifyContent="space-around"}} alignItems="center">
  <View style={styles.container}, {{backgroundColor="dodgerblue"}}/>
  <View style={styles.container}, {{backgroundColor="red"}}/>
  <View style={styles.container}, {{flex:3,
backgroundColor="orange"}}/>
</SafeAreaView>
{const styles=StyleSheet.create({
  container:{
    flex:1, backgroundColor:"#fff",    paddingTop: Platform.OS ==
"android" ? 20:0,alignItems:"center",
  },
});
```

# Exercise 5

Squering the screen.

# Absolute and relative positioning

On certain applications we want to put the components exactly on a specific position on the screen.

```
<SafeAreaView style={styles.container}, {{flexDirection="column",
justifyContent="space-around"}} alignItems="center">
  <View style={styles.container}, {{backgroundColor="dodgerblue", right:20}}/>
  <View style={styles.container}, {{backgroundColor="red", bottom:20
position:"relative"}}/>
    <View style={styles.container}, {{top:30, backgroundColor="orange", position:
"absolute"}}/>
    <View style={styles.container}, {{top:30, backgroundColor="green"}}/>
</SafeAreaView>
{const styles=StyleSheet.create({
  container:{
    flex:1, backgroundColor:"#fff", paddingTop: Platform.OS ≡ "android" ? 20:0,
    alignItems:"center",
  },
});
```

# Interaction

Objects you can interact to:

- Button
- TouchableHighlight
- TouchableNativeFeedback
- TouchableOpacity
- TouchableWithoutFeedback

# Interaction (cont..)

## Example

Importing  
thecomponents

```
import React, { Component } from 'react';  
import { Platform, StyleSheet, Text, TouchableHighlight, TouchableOpacity,  
TouchableNativeFeedback, TouchableWithoutFeedback, View } from 'react-  
native';
```

```
export default class Touchables extends Component {  
  _onPressButton() {  
    alert('You tapped the button!')  
  }  
  
  _onLongPressButton() {  
    alert('You long-pressed the button!')  
  }  
}
```

Methods for  
interaction

# Interaction (cont..)

## Example

Iconnecting the  
events

```
render() {  
  return (  
    <View style={styles.container}>  
      <TouchableHighlight onPress={this._onPressButton} underlayColor="white">  
        <View style={styles.button}>  
          <Text style={styles.buttonText}>TouchableHighlight</Text>  
        </View>  
      </TouchableHighlight>  
      <TouchableOpacity onPress={this._onPressButton}>  
        <View style={styles.button}>  
          <Text style={styles.buttonText}>TouchableOpacity</Text>  
        </View>  
      </TouchableOpacity>  
      <TouchableNativeFeedback  
        onPress={this._onPressButton}  
        background={Platform.OS === 'android' ?  
TouchableNativeFeedback.SelectableBackground() : ''}>  
        <View style={styles.button}>  
          <Text style={styles.buttonText}>TouchableNativeFeedback {Platform.OS !==  
'android' ? '(Android only)' : ''}</Text>  
        </View>  
      </TouchableNativeFeedback>  
    </View>  
  )  
}
```

# Data with FlatList

Imagine you have a set of items of color with a colorName and color value properties:

```
const [selectedColors, setSelectColors]=useState([]);
Const handleValueChange =useCallback((value, color) => {
    if(value === true){
        setSelectedColors(colors => [... colors, color]);
    }else{
        setSelectedColors(colors => colors.filter(selectedColor
=> color.coloname !== selectedColor.colorName, ),,));}},[]);
<View style={styles.container}>
  <Text style={styles.name}> Name of the Pallete </Text>
  <FlatList
    data={colors}
    KeyExtractor = {item =>item.colorName}
    RenderItem =(({item}) => (
      <View styles = {styles.color}>
        <Text> {item.colorName} </Text>
        <Switch value={!!selectedColor.find(color => color.coloname ===
item.colorName)} onChange={selected=>{handleValueChange(selected, item)}}/>
      </View>
    ))
  />
```



# Exercise 6

Countries.

# Navigating between screens

We usually create several screen in our applications, for example: The login window and our overview screen.

```
npm install @react-navigation/native @react-navigation/native-stack
```

```
expo install react-native-screens react-native-safe-area-context
```

```
cd ios  
pod install  
cd ..
```

**Just for iOS,  
beware you  
need  
cocoapods**

# Navigating between screens

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

const Stack = createNativeStackNavigator();

const MyStack = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

# Navigating between screens

```
const HomeScreen = ({ navigation }) => {  
  return (  
    <Button  
      title="Go to Jane's profile"  
      onPress={() =>  
        navigation.navigate('Profile', { name: 'Jane' })  
      }  
    />  
  );  
};  
  
const ProfileScreen = ({ navigation, route }) => {  
  return <Text>This is {route.params.name}'s profile</Text>;  
};
```

# Navigating between screens

react-native-screens package requires one additional configuration step to properly work on Android devices. Edit MainActivity.java file which is located in android/app/src/main/java/<your package name>/MainActivity.java.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(null);  
}
```

and make sure to add an import statement at the top of this file:

```
import android.os.Bundle;
```

# Navigating between screens

When creating your screen you should have a navigator object as parameter, this object is the one which will allow to change from one screen to another.

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}
```

# Nested Navigator

For more complex environments you can nest navigators in order to change just a part of the screen

```
function Home() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Feed" component={Feed} />
      <Tab.Screen name="Messages" component={Messages} />
    </Tab.Navigator>
  );
}

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={Home}
          options={{ headerShown: false }}
        />
        <Stack.Screen name="Profile" component={Profile} />
        <Stack.Screen name="Settings" component={Settings} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

# Exercise 7

Navigating.



# Eager and lazy loading

When we use the import directive we use eager loading, that is all the components imported are loaded at the time they are called, but if we want that to change in order to prevent network traffic or delays when loading we can use an import function creating a lazy load

```
import("./math").then(math => {  
  console.log(math.add(16, 26));  
});  
  
const OtherComponent = React.lazy(() => import('./OtherComponent'))
```

# ErrorHandling using error boundaries

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

```
class ErrorBoundary extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { hasError: false };  
  }  
  static getDerivedStateFromError(error) {  
    // Update state so the next render will show the fallback UI.  
    return { hasError: true };  
  }  
  componentDidCatch(error, errorInfo) {  
    // You can also log the error to an error reporting service  
    logErrorToMyService(error, errorInfo);  
  }  
  render() {  
    if (this.state.hasError) {  
      // You can render any custom fallback UI  
      return <h1>Something went wrong.</h1>;  
    }  
    return this.props.children;  
  }  
}
```

Then you can use this as a wrapper component:

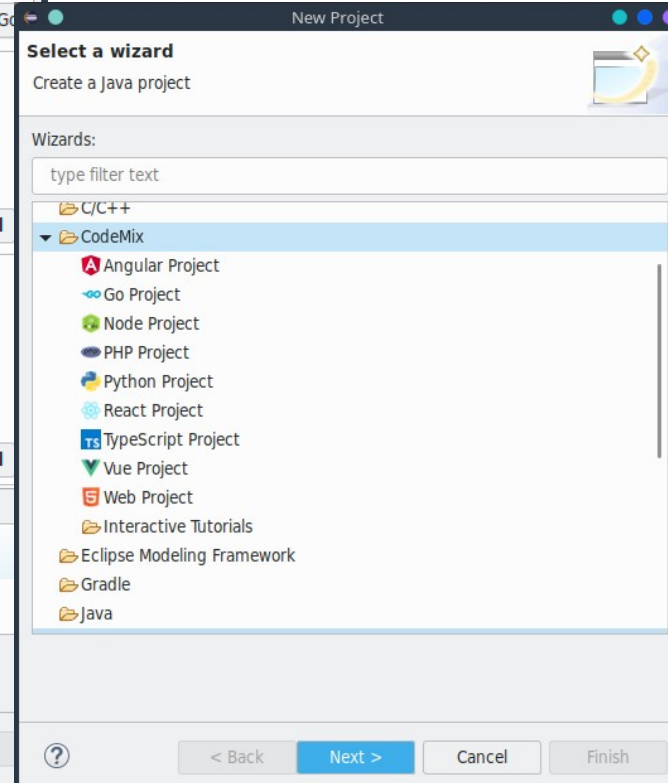
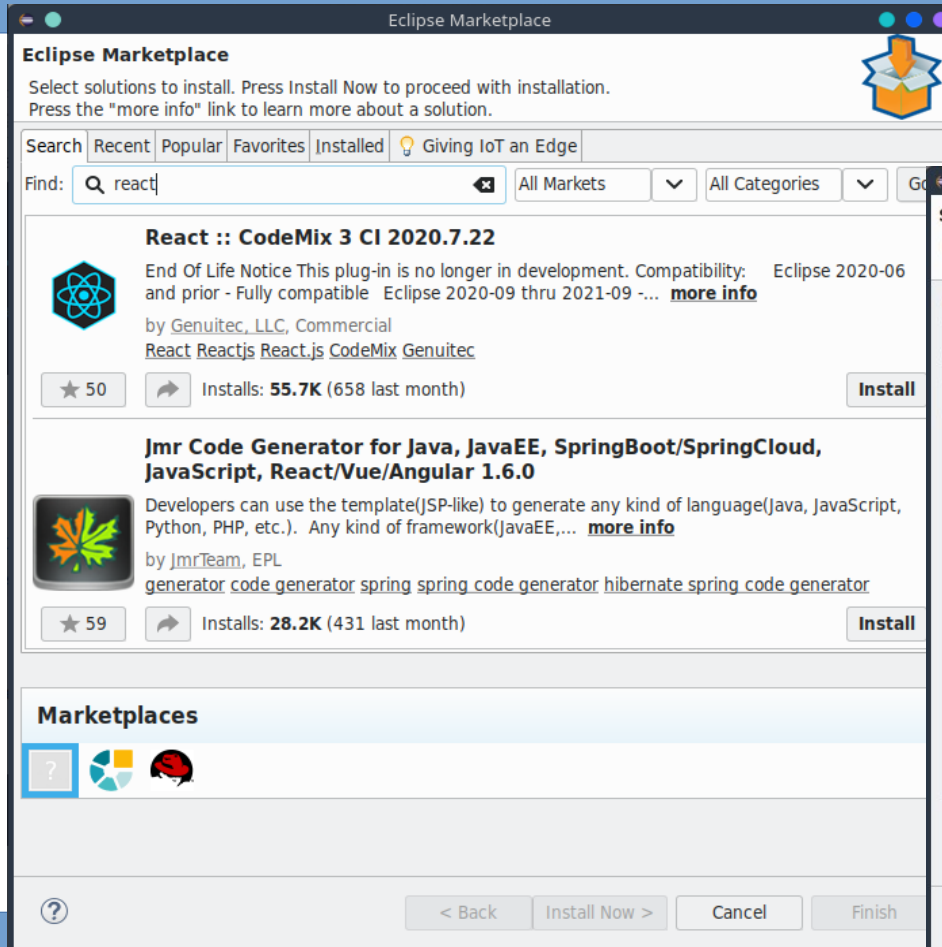
```
<ErrorBoundary>  
  <MyWidget />  
</ErrorBoundary>
```

# Exercise 8

All together.

# Integrating

You can use some React plugins with your favorite IDE, for example with Eclipse you can use CodeMix from the Eclipse Market Place.



# Integrating Android

- Create an /android subfolder inside of your project structure.
- Create a package.json file in the root directory

```
{
  "name": "MyReactApp",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "yarn react-native start"
  }
}
```

```
// greeting.js
export let message = 'Hi';

export function setMessage(msg) {
  message = msg;
}
```

```
// app.js
import {message, setMessage} from
'./greeting.js';
console.log(message); // 'Hi'
setMessage('Hello');
console.log(message); // 'Hello'
```

# What's next?

- WebWorkers
- IndexedDB
- Performance checkers (Audits, Redux, React devtools)
- PWA

# Day 2 summary

**React Introduction** – What's React, what's the difference with React Native.

**JSX**– What's JSX and how to work with it

**Installation**– How to setup our environment.

**Components**– What's a component and how to create one.

**Core components**– What are the core components of React.

**Dimensions and device orientation**– Learn how to use the Dimension, useDimensions and other objects to see what's the screen size and accommodate components.

**Flexbox**– learn how and when to use Flex.

**Interaction** – How to have components that interact with the users.

**Screens**– Learn how to have several screens.

**Navigator**– Learn how Navigator to change and communicate among screens

**Lazy loading** – Learn how to import components in a lazy way in order to have a better performance at loading

**Error handling** – How to use Error boundaries to manage error when using components

**Integration with android**– How to use plugins to use Eclipse or Android Studio in order integrate React