React Native

Tech Talk Team B

Mohammed Alnasser
Reshmasai Malleedi
Madison McFadden
Rebecca Rozansky

Table of Contents

I. Introduction

II. React Native Fundamentals

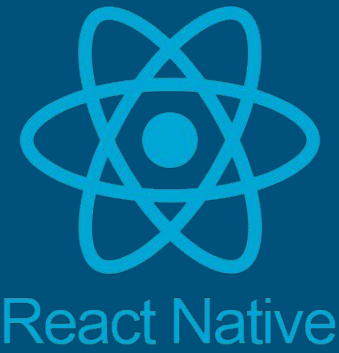
III. Getting Started

IV. Walkthrough Example

Introduction

What is React Native?

- A JavaScript framework for building mobile applications that can be deployed on both iOS and Android platforms
- It is based on React, a popular JavaScript library for building user interfaces on the web
- React Native uses a single codebase to build mobile apps, which means that developers can write code once and deploy it on both iOS and Android platforms



Why React Native?

- React Native provides a set of pre-built components that developers can use to build user interfaces, and it uses a virtual DOM (Document Object Model) to manage the state of the app
  - React Native apps can be developed using a declarative programming approach, which allows developers to focus on what the app should do rather than how it should do it
- React Native also allows developers to use native code where needed
  - Developers can leverage the performance and features of native platforms when necessary
- React Native supports hot reloading
  - Developers can see the changes they make to the code in real-time, without having to rebuild the app from scratch.

Popular Apps built using React Native

- Facebook
- Discord
- Pinterest
- Airbnb
- UberEats
- Instagram
- Wix
- Soundcloud
- Skype

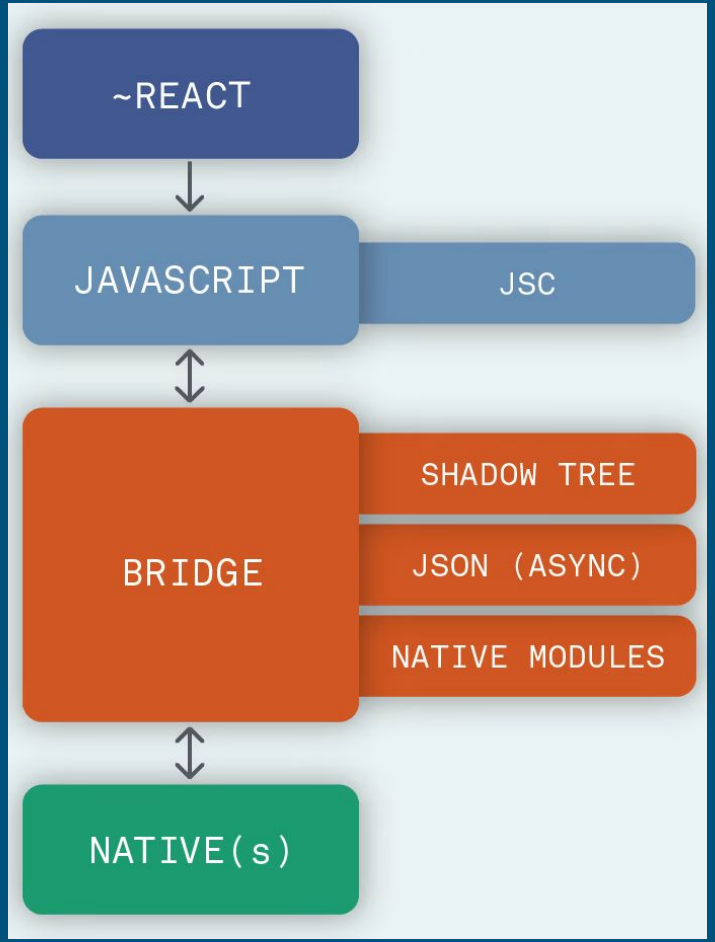
React Native Fundamentals

React Native Architecture

JavaScript Code
- responsible for managing the user interface, application logic, and data flow

Native Code
- responsible for handling platform-specific tasks such as rendering UI components, accessing device features like camera and GPS, and managing memory.

React Native uses a bridge to communicate between the JavaScript code and the native code, which allows them to interact with each other.
- The bridge passes messages between the two parts of the app, which enables the JavaScript code to call native modules and the native code to call JavaScript functions. Also uses a virtual DOM (Document Object Model) to manage the state of the app



Features

- React Native uses a component-based architecture, which means that the user interface of the app is built using reusable components.
  - Components are like building blocks that can be combined to create complex UI elements.
  - Components can also have their own state and lifecycle methods, which allows them to respond to user interactions and changes in the state of the app.
- React Native uses a style system that is similar to CSS, which allows developers to apply styles to components using a set of style properties.
  - The style system includes support for flexbox layout, which makes it easy to create responsive layouts that work across different screen sizes and orientations.
- To handle user input and application logic, React Native uses a combination of event handling and state management.
  - Events are triggered by user interactions, such as a button press or a swipe gesture, and they can be used to update the state of the app. State is a representation of the data that the app needs to display, and it can be updated in response to user interactions or other events.

Native Components

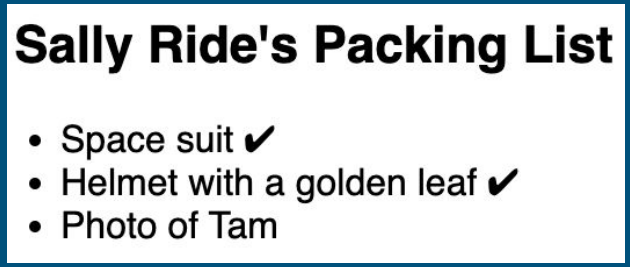
- For view development, React Native invokes views using JS and React components
- At runtime, React Native creates the corresponding Android and iOS views for those components.
    - Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. We call these platform-backed components Native Components.
- React Native comes with a set of essential, ready-to-use Native Components you can use to start building your app today called Core Components.

Core Components

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<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

Getting Started with React Native

Components



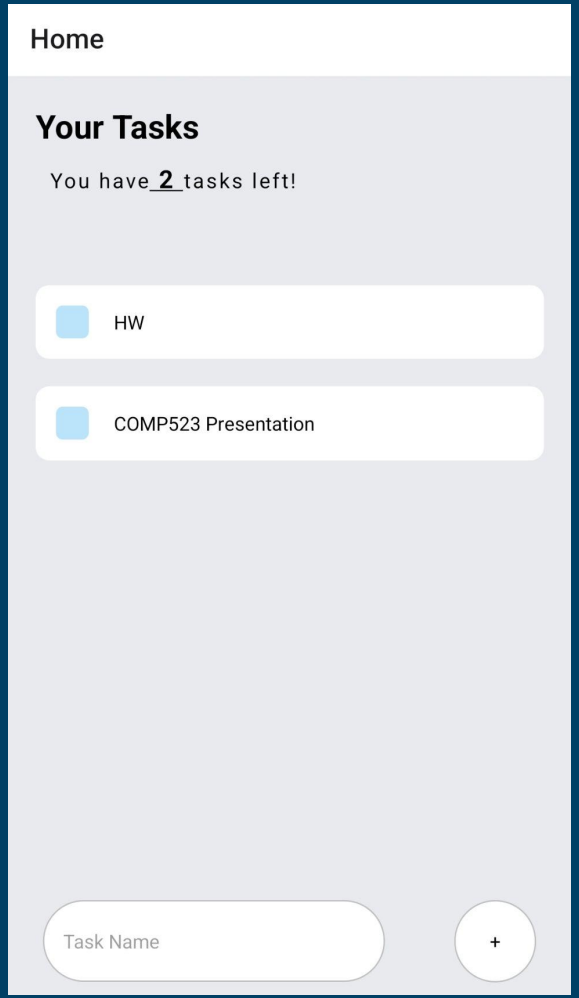
- What are they?
  - Components are like building blocks that can be combined to create complex UI elements
- Reusability via props
  - Props is short for “properties”. Props let you customize React components.
- Conditional Rendering
- State
  - state is like a component’s personal data storage. State is useful for handling data that changes over time or that comes from user interaction - gives components memory
- Stylesheets
  - In React Native, styling is done using a style sheet system that is similar to CSS but uses a JavaScript syntax. The style sheet system is designed to be fast and efficient, with styles being preprocessed and then applied directly to native components.
- Navigation
  - Navigating between screens in a React Native app, including Stack Navigation, Tab Navigation, and Drawer Navigation.

Running React Native Apps

- Expo App + NPM
- Running simulator via expo start
- iOS:
  - xCode
  - Create a project
  - Run the simulator
- Android:
  - Android Studio Emulator
  - Create a virtual device
  - Link the location to Expo

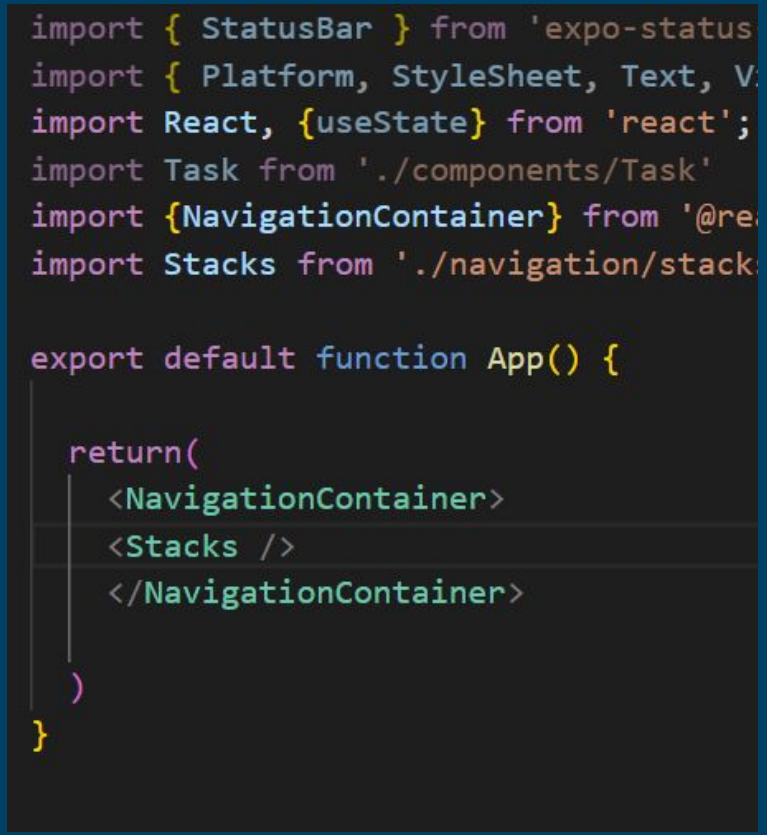
Todo List Tutorial

https://github.com/rebeccarozansky/react-native-tutorial
https://github.com/rebeccarozansky/react-native-tutorial



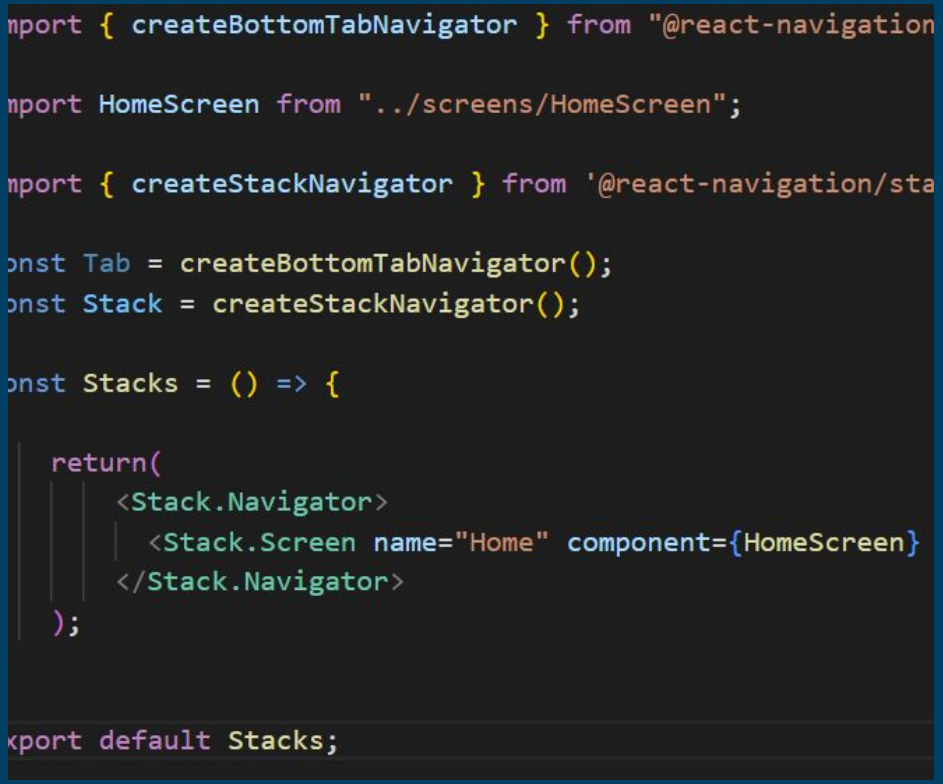
App.js

The main file where the code executes from



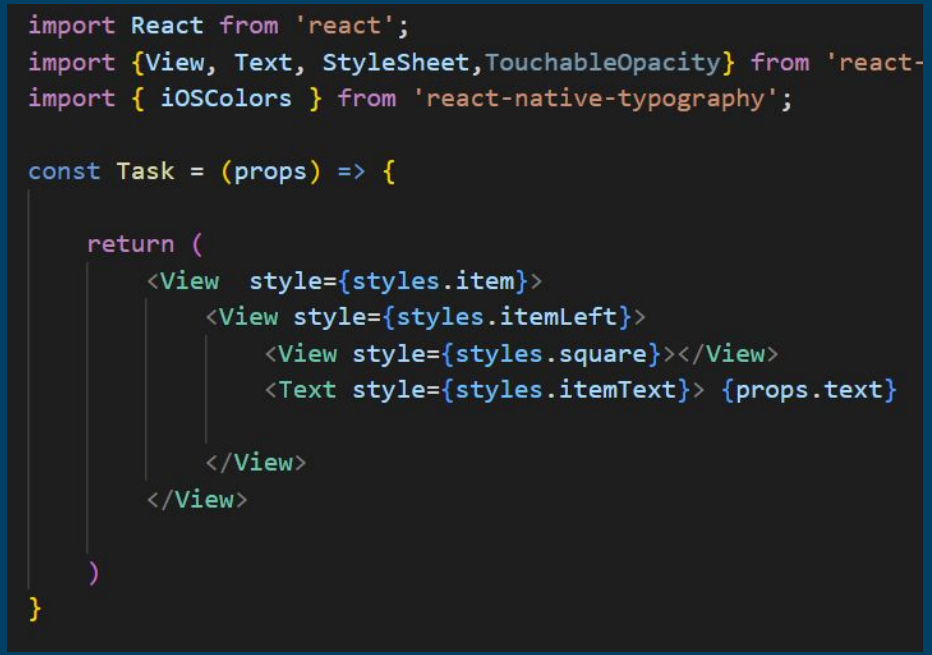
Navigation

Allows for navigation between screens of the app



Task.js

The task that is displayed in the to-do list.



```
import React from 'react';
import {View, Text, StyleSheet,TouchableOpacity} from 'react-
import { iOSColors } from 'react-native-typography';

const Task = (props) => {

    return (
        <View  style={styles.item}>
            <View style={styles.itemLeft}>
                <View style={styles.square}></View>
                <Text style={styles.itemText}> {props.text}

            </View>
        </View>

    )
}
```

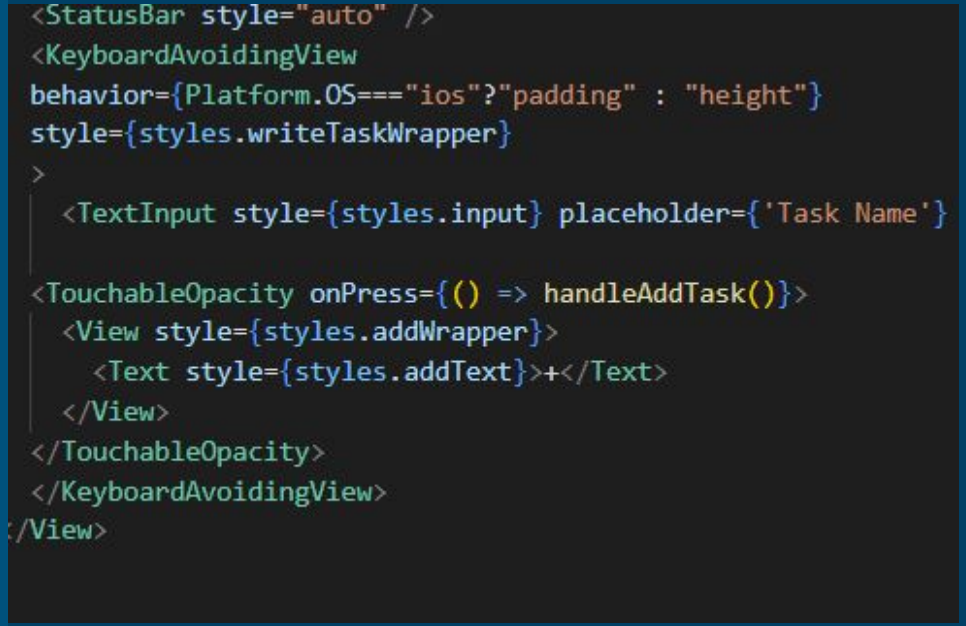
HomeScreen.js

Uses .map to display all tasks that exist

```
<View style={styles.container}>

  <View style={styles.tasksWrapper}>
  <Text style={styles.sectionTitle}>Your Tasks </Text>

  <Text style={styles.itemDescription}>You have
    <Text style={styles.itemDescriptionStrong}> {taskItems.length} </Text>
    tasks left! {"\n\n"}
  </Text>
  <ScrollView contentContainerStyle={{ flexGrow: 1 }} style={styles.items}>
    {
      taskItems.map((item,index)=>{
        return (
        <TouchableOpacity key={index} onLongPress={() => completeTask(index)}>
                    <Task text={item['task']} />
        </TouchableOpacity>
      )})

    }
```

HomeScreen.js

The functionality for adding a new task



HomeScreen.js

The states and functions that keep the application updated

```
const [task, setTask] = useState();
const [taskItems, setTaskItems] = useState([]);

const handleAddTask = () => {
    Keyboard.dismiss();
    var temp = {'task':task}
    var itemsCopy2 = [...taskItems,temp]
    setTaskItems(itemsCopy2)
    setTask(null);
}

const completeTask = (index) => {
    let itemsCopy = [...taskItems];
    itemsCopy.splice(index, 1);
    setTaskItems(itemsCopy);
}
```

Testing and Debugging

- Testing and debugging in React Native are similar to testing and debugging in other JavaScript-based frameworks
- Common Approaches:
  - Unit testing: involves testing individual components and functions in isolation to ensure they work as expected. Jest is a popular testing framework used for unit testing in React Native.
  - Integration testing: involves testing how different components work together. Enzyme and Detox are popular frameworks used for integration testing in React Native.
  - Console logging: You can use console.log statements to output values and see how your code is executing - simple yet effective

Questions?