Lecture 02

React Native (part 1)

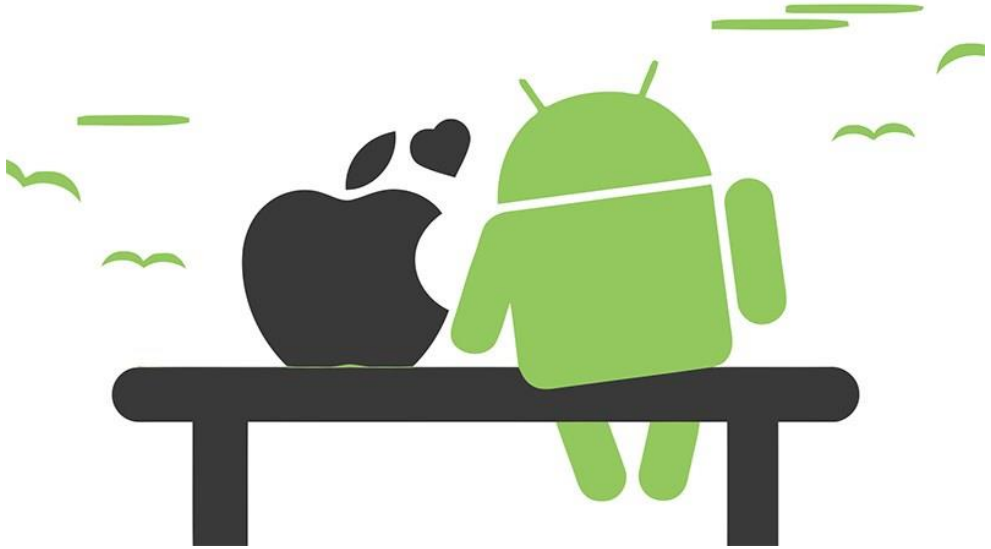
Contents

- Overview of mobile application development.
- Introduction to React Native and comparison with other technologies.
- Setting up the development environment
  (Node.js, Expo CLI, React Native CLI).
- Creating the first React Native app with Expo.
- Core components
- Styling
- Simple event handling
  - onChangeText
  - onPress

What is Mobile Application Development?

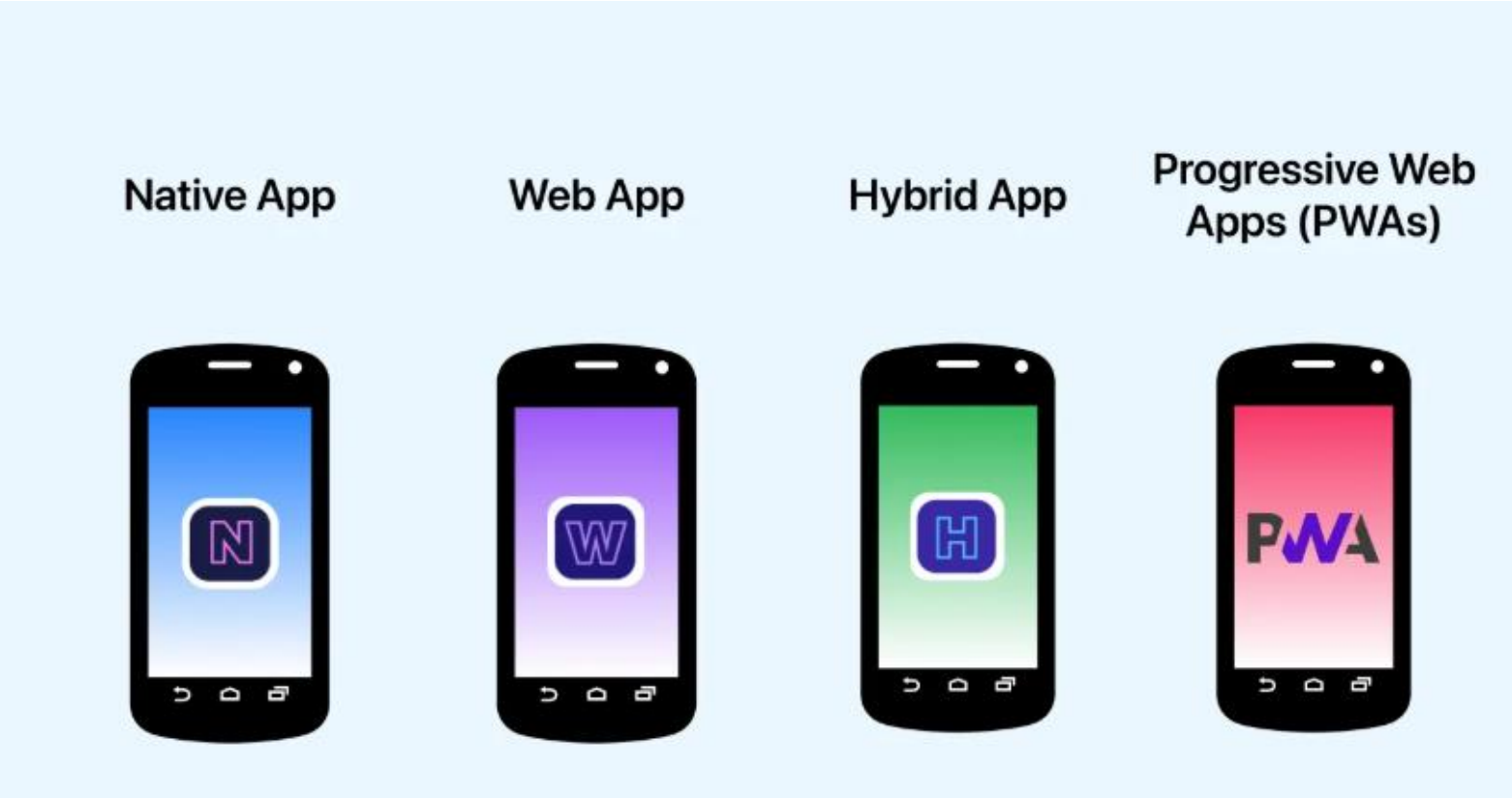
- Mobile application development is the process of creating software applications that run on mobile devices like smartphones and tablets.

- **Platforms** :The two main platforms for mobile apps are **Android** and **iOS**.
  - **Android** apps are typically developed using **Java** or **Kotlin**, while **iOS** apps are developed using **Swift** or **Objective-C**



Types of Apps

- Different types of mobile apps offer unique features and functionality. Understanding these types can help developers choose the right approach for developing a mobile app.



Types of Apps

Native Apps	Cross-Platform Apps	Hybrid Apps	Progressive Web Apps (PWAs)
Built specifically for one platform (either Android or iOS) using platform-specific languages and tools.	Developed to work on multiple platforms using a single codebase, often with frameworks like React Native or Flutter.	Combine elements of both native and web apps, using technologies like HTML5, CSS, and JavaScript.	Web applications that provide a mobile app-like experience through a web browser

Types of Apps

Native Apps	Cross-Platform Apps	Hybrid Apps	Progressive Web Apps (PWAs)
Pros: • Best runtime performance • Direct access to device APIs **Cons:** • Higher costs when building and maintaining your app • Multiple code-bases for each platform	**Pros:** • Single code base for multiple platforms • Easy to build and maintain your app **Cons:** • Dependent on bridges and libraries for native device features • Performance limitations due to bridging	**Pros:** • Shared code base between web and mobile apps • Using web development skillset for building mobile apps **Cons:** • Lower performance compared to native apps • Limited support for native device features	**Pros:** • Same app is available both for web and mobile • No installation required, accessible through a URL **Cons:** • Limited support for native device features • App capabilities depend on the browser in use

What is React Native?

React.js

A JavaScript library for building user interfaces

Typically used for web development

It's actually react-dom that adds web support

React is platform-agnostic

+

React Native

A collection of "special" React Component

Components are compiled to native UI

Native platform APIs exposed to JS

React Native is like react dom; Connect React to a specific platform

→

Real Native Mobile Apps

IOS

Android

Why React Native?

- Faster development speed
- Highly reusable code and easy to find developers that can use React
- Apps run across multiple platforms
- Possible to ship over the air updates bypassing App Store / Play Store

Expo CLI vs React Native CLI

Expo CLI and React Native CLI are two different tools used for developing React Native applications, each with its own strengths and use cases.

Expo CLI (“Expo”) → You can switch (“eject”) any time → React Native CLI

Expo CLI (“Expo”)	React Native CLI
Third-party service (free!)	By the React Native team & community
"Managed app development"	Bare-bone development
Very convenient, less friction	Fewer convenience features
Can leave the ecosystem any time	Can leave the ecosystem any time

Start a new React Native project with Expo

- Expo is a production-grade React Native Framework. Expo provides developer tooling that makes developing apps easier, such as file-based routing, a standard library of native modules, and much more.
- Expo's Framework is free and open source, with an active community on GitHub and Discord. The Expo team works in close collaboration with the React Native team at Meta to bring the latest React Native features to the Expo SDK.
- The team at Expo also provides Expo Application Services (EAS), an optional set of services that complements Expo, the Framework, in each step of the development process.
- To create a new Expo project, run the following in your terminal:

```
npx create-expo-app@latest <app-name>
```

How to create?

Step 1: Make sure that your computer has already installed Nodejs

If not, you can search for nodejs on Google and download the newest version (now is v22.13.0) [Node.js — Run JavaScript Everywhere]

How to create?

Step 2: Open the command line and run:

```
npx create-expo-app@latest <app-name>
```

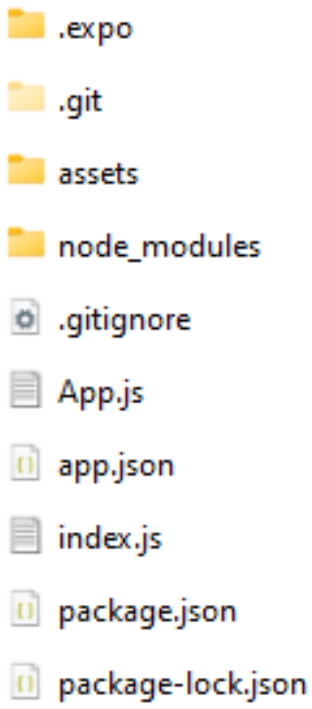
```
C:\Users\Vinh\Documents\Code>npx create-expo-app@latest
Creating an Expo project using the default template.

To choose from all available templates pass in the --template arg:
  $ npx create-expo-app --template

To choose from all available examples pass in the --example arg:
  $ npx create-expo-app --example

√ What is your app named? ... my-app
- Locating project files.
```

Successfully Installation

```
✅ Your project is ready!

To run your project, navigate to the directory and run one of the following npm commands.

- cd my-app
- npm run android
- npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development with
out a Mac
- npm run web
```

What’s inside a React Native project?

.gitignore:
  .expo
  .vscode
node_modules:
  #Holds 3rd party packages
assets:
  #Holds images used inside app
package.json, package-lock.json:
  #Holds dependencies, script commands, etc.
app.json:
  #Configuration settings
App.js:
  #The real code



Run on Website

Step 1: Install dependencies

```
npx expo install
react-dom react-
native-web
@expo/metro-runtime
```

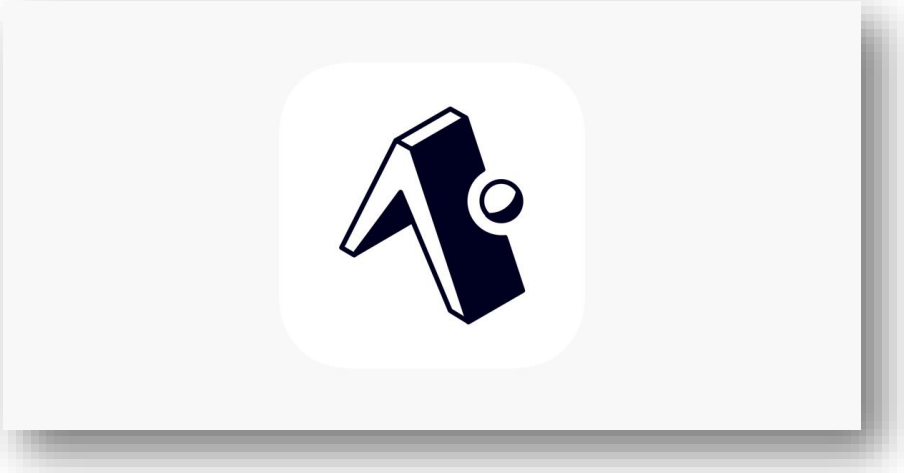
Step 2: Run

```
npm run web
```

```
› Metro waiting on exp://192.168.10.101:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Web is waiting on http://localhost:8081

› Using Expo Go
› Press s │ switch to development build

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu
› shift+m │ more tools
› Press o │ open project code in your editor

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
Web Bundled 12615ms index.js (163 modules)
```

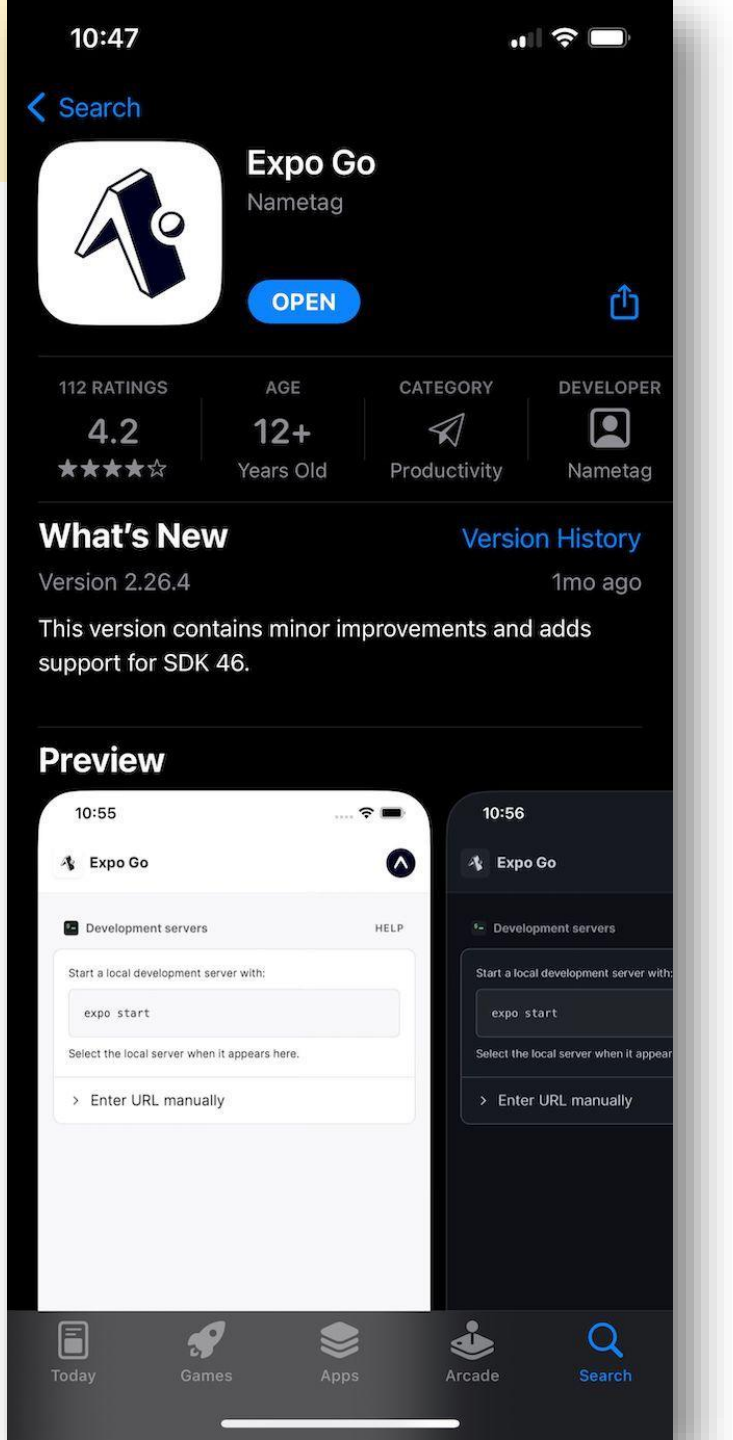
Expo Go

- Provided by Expo Technologies, Inc.
- Free, open-source sandbox for learning and experimenting with React Native on mobile devices



Run on Expo Go

- Download from App Store (iOS) or Play Store (Android)



Run on Expo Go

- To run your app on Expo Go:
  - Open terminal
  - Type “**npx expo start**”

- Expo Developing Server will be started as a QR code generated to be scanned.

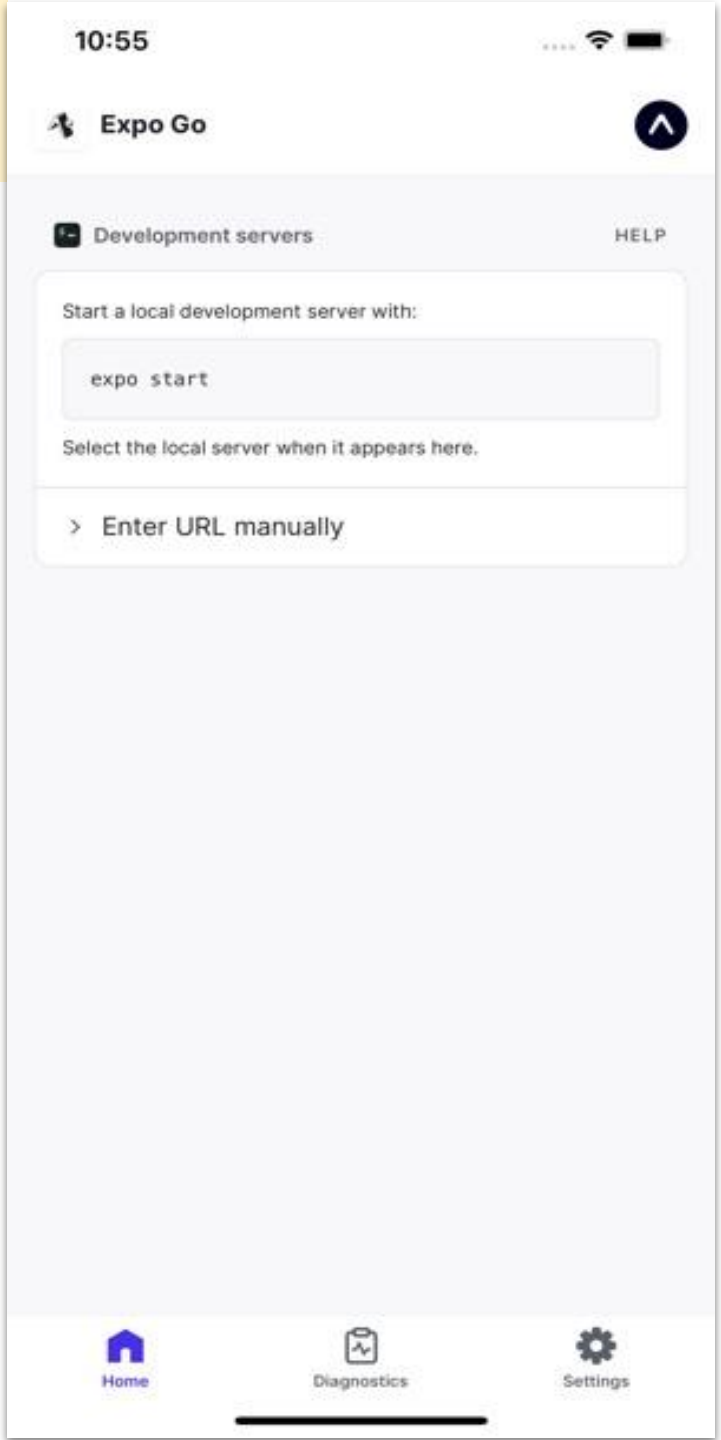
```
PS F:\Coding_Projects\Python\Python scripts\MPR> cd AwesomeProject
>>
PS F:\Coding_Projects\Python\Python scripts\MPR\AwesomeProject> npx expo start
Starting project at F:\Coding_Projects\Python\Python scripts\MPR\AwesomeProject
Starting Metro Bundler
```


```
› Metro waiting on exp://192.[redacted]
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Using Expo Go
› Press s | switch to development build

› Press a | open Android
› Press w | open web

› Press j | open debugger
› Press r | reload app
› Press m | toggle menu
› Press o | open project code in your editor

› Press ? | show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
```

Run on Expo Go

- Using mobile devices to scan QR code
  - On Android: click “Scan QR code”
  - On iOS: Camera > Open the link

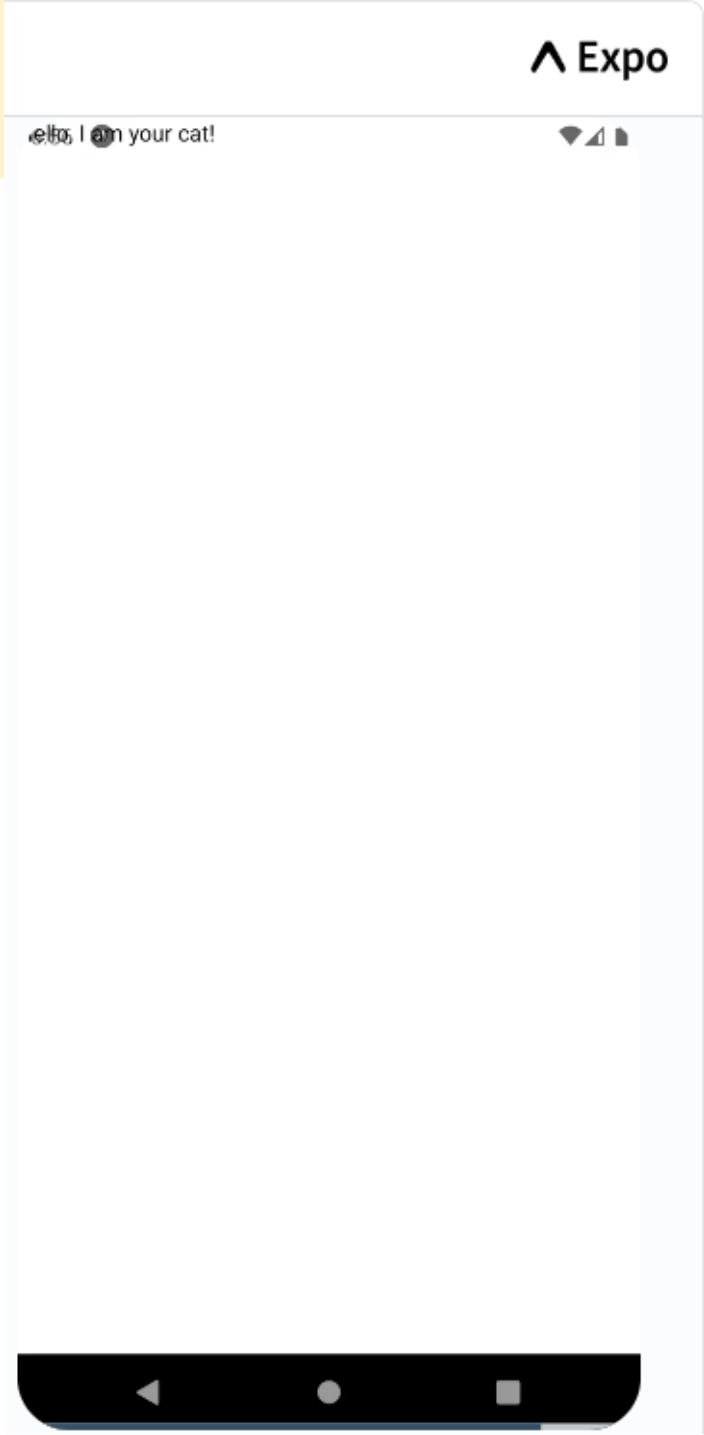
```
Logs for your project will appear below. Press Ctrl+C to exit.
Android Bundling complete 1713ms (E:\sevagoth\node_modules\expo\AppEntry.js)
```



A look under the hood

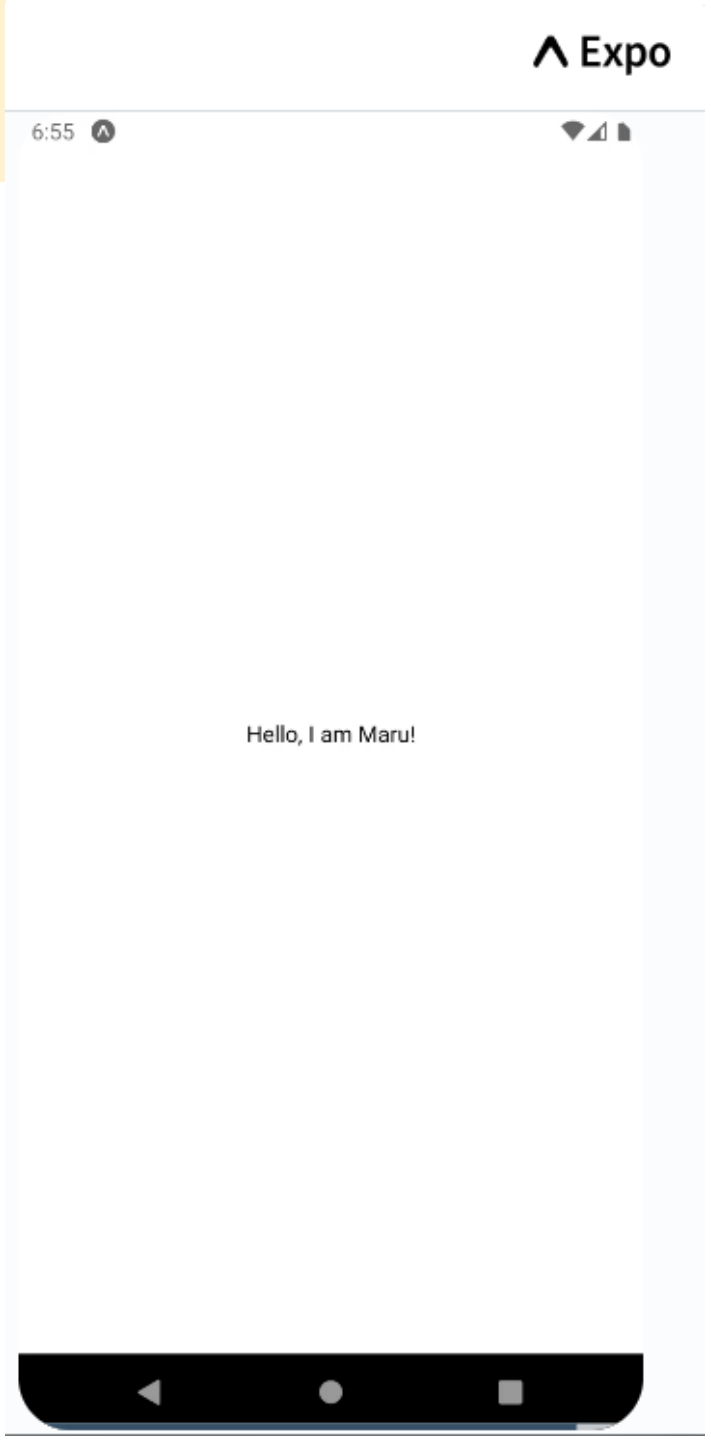
```
import React from 'react';
import {Text} from 'react-native';

const Cat = () => {
  return <Text>Hello, I am your cat!</Text>;
};

export default Cat;
```

Only JSX and Components are compiled, not JavaScript logic.



A look under the hood

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

const Cat = () => {
  const name = 'Maru';
  return (
    <View style={styles.container}>
      <Text>Hello, I am {name}!</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});
export default Cat;
```



Let’s analyze the HelloWorld app!

- There’s a function component called `App`
  - This component acts as the **root** component *by default*.

```
export default function App() {
  return (
    <View style={styles.container}>
      <Text>Hello, world!</Text>
      <StatusBar style="auto" />
    </View>
  );
}
```

- We can see other components: `View`, `Text`, `StatusBar`
- Also note the `style` attribute attached to the `View` component

Native components

- In React Native, you can’t use HTML tags in JSX code.
  - Unlike React where your application runs on the browser, React Native application runs on mobile devices.
  - There’s no DOM to work with.
- Mobile UI elements are “exposed” as React Native components.
  - Think of how you can read user inputs in Java because the system’s keyboard is exposed as the `System.in` object.

React Native components are compiled...

- React Native's built-in components are compiled into actual iOS or Android code.

Web Browser (react-dom)	Native Component (Android)	Native Component (iOS)	React Native JSX
`<div>`	`android.View`	`UIView`	`<View>`
`<input>`	`EditText`	`UITextField`	`<TextInput>`
...	...	...	...

React Native components documentation

https://reactnative.dev/docs/components-and-apis

Core Components and APIs

React Native provides a number of built-in Core Components ready for you to use in your app. You can find them all in the left sidebar (or menu above, if you are on a narrow screen). If you're not sure where to get started, take a look at the following categories:

- Basic Components
- User Interface
- List Views
- Android-specific
- iOS-specific
- Others

You're not limited to the components and APIs bundled with React Native. React Native has a community of thousands of developers. If you're looking for a library that does

Building app UI by combining Core components

"Core" Components
(Built Into React Native)

"Translation" to native UI widgets is provided by React Native

```
<View />
<Text />
<Button />
<TextInput />
<Image />
<... />
```

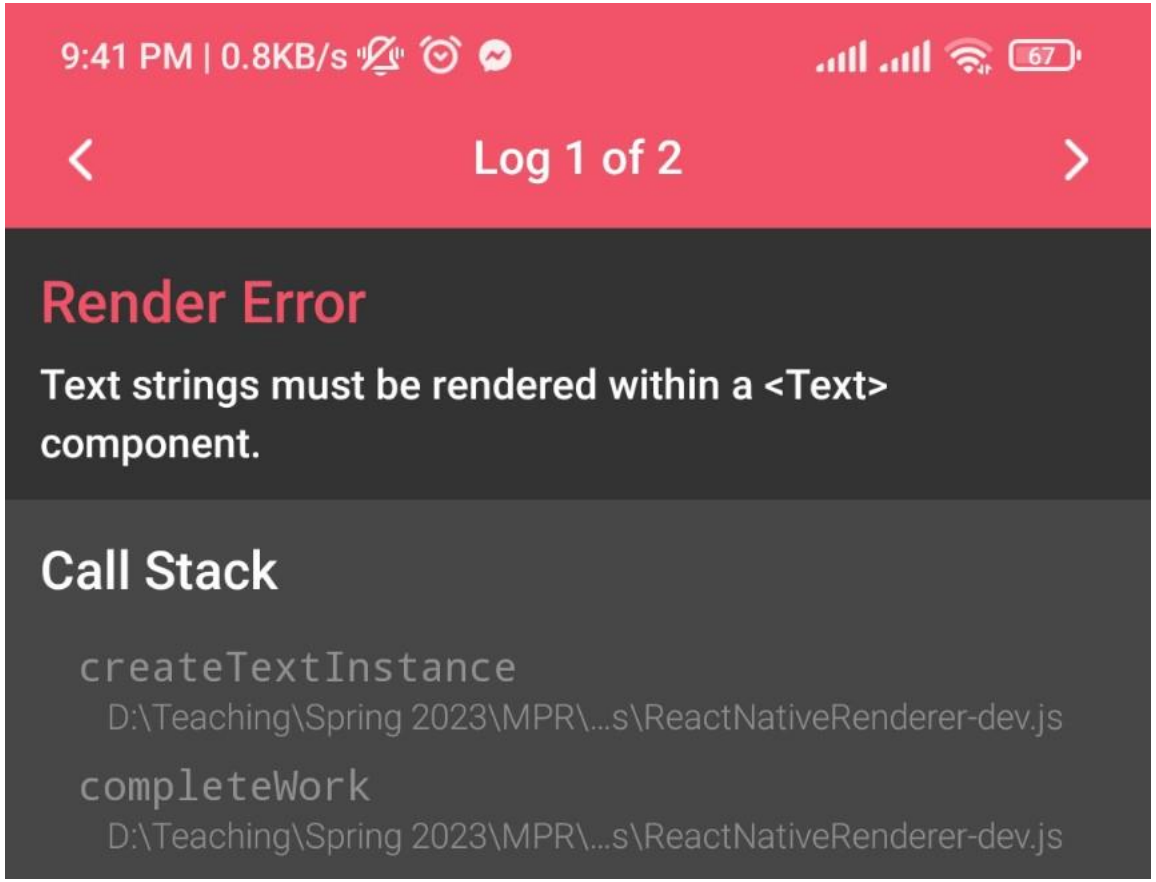
Your UI & Custom Components

Combination of "Core" components & other built-in components

```
const MyTitle = props => {
  return (
    <View>
      <Text>{props.title}</Text>
    </View>
  );
};
```

React Native is different from HTML

```
export default function App() {
  return (
    <View style={styles.container}>
      Hello, world!
    </View>
  );
}
```



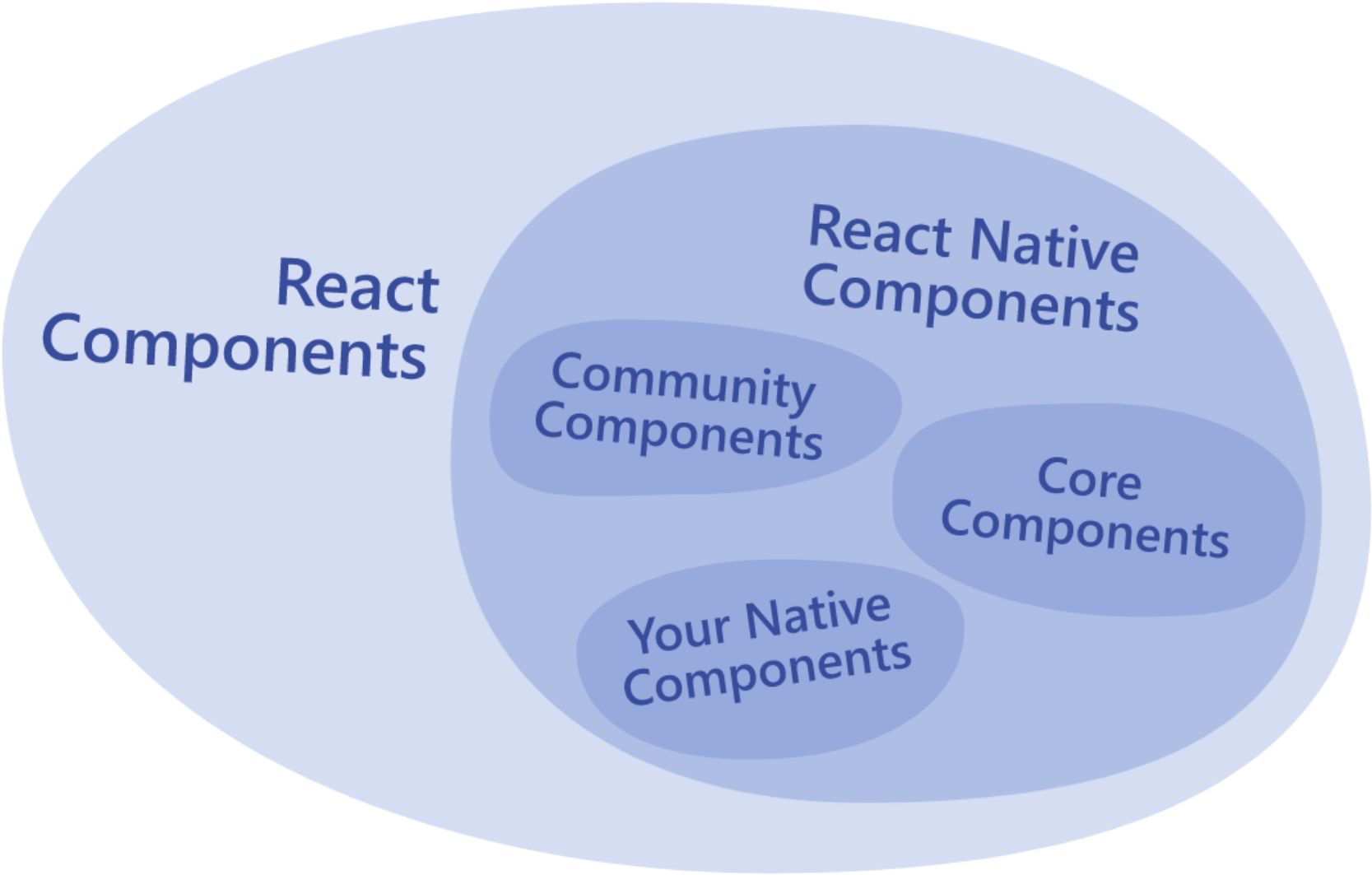
- Each core component has a different role.

  E.g. Only `Text` component can hold text strings. `View` is used as container for laying out other components.

Some important React Native 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

The Components landscape



React Native styling

There Is No CSS!

Inline Styles ⟷ **StyleSheet Objects**

Written in JavaScript
(i.e. in the JavaScript code files, next to the component code)

Based on CSS syntax, but only a **subset** of properties & features is supported!

React Native inline styling

- A core component receives a styling object via the `style` attribute.
  - Different properties require different types of values (string, integer...)
  - This styling is close to CSS but not exactly CSS.
  - You'll get VSCode suggestions for property names.

```
    <View style={styles.container}>
      <View>
        <Text>Another piece of text!</Text>
      </View>
      <Text style={{mar}}>Hello World!</Text>
      <Button title='Ta  margin?          (property) FlexStyle.margin?:
    </View>              marginBottom?
  );                     marginEnd?
}                        marginHorizontal?
                         marginLeft?
```

```
<View style={styles.container}>
  <Text style={{
    margin: 15,
    borderWidth: 2,
    borderColor: 'red'
  }}>Hello, world!</Text>
</View>
```

React Native StyleSheet object

- Allows style re-use & better code readability

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  myTextStyle: {
    margin: 15,
    borderWidth: 2,
    borderColor: 'red'
  }
});
```

```
<View style={styles.container}>
  <Text style={styles.myTextStyle}>
    Hello, world!
  </Text>
</View>
```

Additional reading about styling

- The official styling documentation
  - https://reactnative.dev/docs/style
- An article about different ways of presenting colors
  - https://reactnative.dev/docs/colors
- Each core component's documentation lists all supported style properties. Example:
  - https://reactnative.dev/docs/view#style

The `TextInput` component

- A basic component for inputting text into the app via a keyboard
- Important props (attributes):
  - `value`: set the predefined value for the text field
  - `placeholder`: displays a placeholder text when the input is empty
  - `onChangeText`: set the function to call every time the text value changes
  - `editable`: make the input editable or not (default: `true`)
  - Some other common props: `maxLength`, `multiline`, `numberOfLines`, `onSubmitEditing`, `onFocus`

The `Button` component

- A basic button that should render nicely on any platform.
  - Supports a minimal level of customization.
  - If more customization is needed, `Pressable` can be used instead.
- Important props (attributes):
  - `title`: the actual display text of the button
  - `onPress`: the event handling attribute
  - `color`: set the theme color for the button
    (renders differently on iOS and Android)
  - Other props: `disabled, accessibilityLabel, accessibilityActions..`

Layout with FlexBox

Layouts are (typically) created with Flexbox

Very similar to browser CSS flexbox!

Elements are positioned inside of containers

Positioning is controlled via style settings applied to the element container

Layout with FlexBox

```
flex: 1,
flexDirection: 'column',
justifyContent: 'flex-start',
alignItems: 'flex-start'
```

The element (container) should expand to occupy available space

Cross Axis

Main Axis

`flexDirection` controls the orientations of "Main Axis" and "Cross Axis"

i.e. if "Main Axis" is "top to bottom" or "left to right"