Mobile Web Application

Chapter One: Introduction

React Native is a framework developed by Facebook in **2015** for creating native-style applications for Android & iOS under one common language, i.e. JavaScript. Initially, Facebook only developed React Native to support iOS. However, with its recent support of the Android operating system, the library can now render mobile UIs for both platforms.

What is React Native?

React Native is an open-source framework developed by Facebook that allows developers to build mobile applications using JavaScript and React. The key advantage of React Native is that it enables the development of apps for both iOS and Android platforms using a single codebase, which significantly reduces development time and effort.

Key Features of React Native

- **Cross-Platform Development:** Write once, run anywhere. React Native allows developers to create apps for both iOS and Android platforms using a shared codebase.
- **Native Performance:** React Native components are compiled into native code, which means that the apps deliver performance close to native apps.
- **Reusability:** React Native enables code reusability, which allows developers to use the same code for different platforms with minimal adjustments.
- **Hot Reloading:** This feature allows developers to instantly see the results of the latest change to the source code without rebuilding the app from scratch.
- **Component-Based Architecture:** Similar to React for the web, React Native uses a component-based architecture, making it easier to manage complex applications by breaking them down into smaller, reusable components.

How React Native Works?

React Native bridges the gap between JavaScript and native code. When you write code in React Native, it gets compiled into native components for iOS and Android. This means the app runs with the performance and look-and-feel of a native app.

Components of React Native

- **View:** This is the fundamental building block of UI in React Native. It maps to the native view on both iOS and Android.
- **Text:** Used for displaying text. It renders as native text elements on both platforms.
- **TextInput:** A component for text input fields.
- **ScrollView:** A component that provides a scrolling container.
- **StyleSheet:** A utility for defining styles. It works similarly to CSS.

React Native allows you to build cross-platform apps using JavaScript. It combines the best parts of native development with the React framework.

Why Learn React Native?

- React Native allows developers to create mobile apps using website technology. So, a developer who is handy in web development can easily develop a mobile app using React Native.
- React Native allows developers to build cross-platform apps that look and feel entirely Native since it uses JavaScript components that are both built on iOS and Android components.
- Since React Native uses JSX or TSX, a developer isn't required to learn complex languages. It needs Typescript to learn but is not complex language.
- React Native uses fundamental Android and iOS building blocks to compile Native apps for both platforms in JavaScript. This makes handling the code base easier.
- React Native allows you to build apps faster. Instead of recompiling, you can reload an app instantly.
- In React Native we can easily develop and test features by using various libraries and tools like Expo, ESLint, Jest, and Redux.
- On top of being responsive and providing an impressive user experience, React Native apps are faster and agile.
- React Native uses Native components which makes the rendering and execution of the app much faster

Prerequisites for Learning React Native

To begin with React-Native you should have a basic knowledge of the following technologies

- HTML, CSS and JavaScript
- ReactJS
- NodeJS Should be Installed in Your System

Installation

React Native uses Node.js, a JavaScript runtime, to build your JavaScript code. If you don't already have Node.js installed, it's time to get it!

Here we will use the **Expo CLI version** which will be much smoother to run your React Native applications. Follow the below steps one by one to setup your React native environment.

Step 1: Open your terminal and run the below command.

```
npm install -g expo-cli
```

Step 2: Now expo-cli is globally installed so you can create the project folder by running the below command.

```
expo init "projectName"
```

Step 3: Now go into the created folder and start the server by using the following command.

```
cd "projectName"
npm start web
```

Modern Approach

You can use **npx** to create and manage Expo projects without globally installing Expo CLI. This method is very convenient because **npx** allows you to run **Expo CLI** directly without the need for installation.

Step 1: Create the application using the following command.

```
npx create-expo-app my-new-project
```

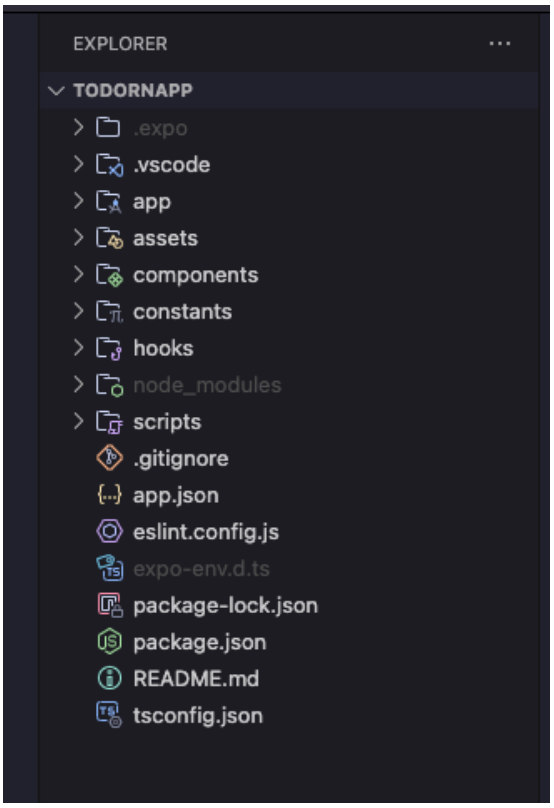
Step 2: Navigate to Your Project Folder

```
cd my-new-project
```

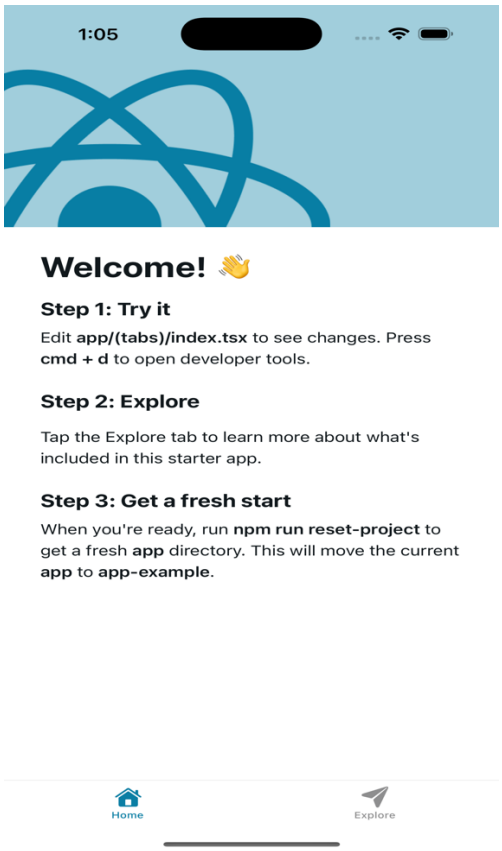
Step 3: Start the Development Server

The first app will look like

Project Structure:



Welcome page



Run this command

npm run reset-project (for typescript app)

After this command you will see index.tsx like this

Step 3: Run the app

To start the react-native program, execute this command in the terminal of the project folder.

npx expo start or npx expo or npx expo –tunnel(for diff network)

Then, the application will display a QR code.

- For Android users, download the "Expo Go" app from the Play Store. Open the app, and you will see a button labeled "Scan QR Code." Click that button and scan the QR code; it will automatically build the Android app on your device.
- For iOS users, simply scan the QR code using the Camera app.
- If you're using a web browser, it will provide a local host link that you can use as mentioned in below image.

Threads in React Native App

React Native uses multiple threads to manage the tasks involved in running an app. The main threads are:

UI Thread (Main Thread)

The **UI Thread** is responsible for rendering the user interface of the app. UI Thread, also known as **Main Thread**, is used for native android or iOS UI rendering. For example, in android this thread is used ensures that the app's interface is responsive and visually rendered.

JS Thread (JavaScript Thread)

JS thread or JavaScript thread is the thread where the logic will run. For e.g., this is the thread where the application's JavaScript code is executed, API calls are made, touch events are processed and many other. For performance, **React Native** ensures that the JS Thread sends updates to the UI Thread before the next frame rendering deadline (which is around 16.67ms for 60 frames per second on iOS).

- If the JS Thread performs complex computations that take **more than 16.67ms**, the UI can become sluggish or unresponsive. This is why the JS Thread must be optimized to avoid long delays in execution.

- Notably, some components like **NavigatorIOS** and **ScrollView** run completely on the UI Thread, which prevents them from being blocked by slow JS thread operations.

Native Modules Thread

The **Native Modules Thread** comes into play when the app needs to access platform-specific APIs (e.g., **camera, GPS)**. This thread allows React Native to call native functions from Android or iOS directly, bridging the gap between JavaScript and native code.

Render Thread (Android Only)

In Android (specifically for version L and later), the **Render Thread** is responsible for generating OpenGL commands used to draw the UI. The Render Thread enables the app to efficiently render graphics and images on the screen.

Process Involved in Working of React Native

React Native works by following a sequence of processes from app startup to rendering:

1) App Startup

On app startup, the main thread is used for executing and loading the JavaScript bundles. The JS bundles have the app's logic and UI components.

2) JavaScript Execution

After the successful loading of the JavaScript bundle, the **JS Thread** takes over the execution. This enables the JS Thread to execute the heavy computations without influencing the UI Thread, which keeps the user interface responsive.

3) Virtual DOM and Reconciliation

React Native employs Reconciliation to effectively render UI updates. The Reconciler then contrasts the current **virtual DOM** with the new virtual DOM ("**diffing**"), and where there are updates to be made, sends them off to the **Shadow Thread.**

4) Layout Calculation

The Shadow Thread determines the layout of the UI components and passes the layout parameters to the UI Thread. "**Shadow**" in this case is the virtual UI that is created during this calculation. The layout consists of position, size, and other characteristics of UI components.

5) Screen Rendering

As only the UI Thread is allowed to display UI on the screen, the layout data computed by the **Shadow Thread** is dispatched to the UI Thread. The UI Thread renders the final result on the screen and displays it to the user.

Parts of React Native

React Native can broadly be divided into three primary parts:

React Native - Native Side

The native side comprises the native modules and views that belong to the **Zombie** or **iOS** platform. It contains features such as the native UI and platform-specific APIs.

React Native - JS Side

The JS side is where the JavaScript code executes. This is the code you, as a developer, write using React and JavaScript. It handles app logic, state, events, and UI rendering.

React Native - Bridge

The Bridge provides a communication mechanism between the JS side and native side. Asynchronous communication is supported, meaning that JavaScript and native code will not block either thread. It enables sending updates, data, and events from JavaScript to native modules.

What do you understand by Virtual DOM ?

The Virtual DOM is an in-memory representation of the DOM. DOM refers to the Document Object Model that represents the content of XML or HTML documents as a tree structure so that the programs can be read, accessed and changed in the document structure, style, and content.

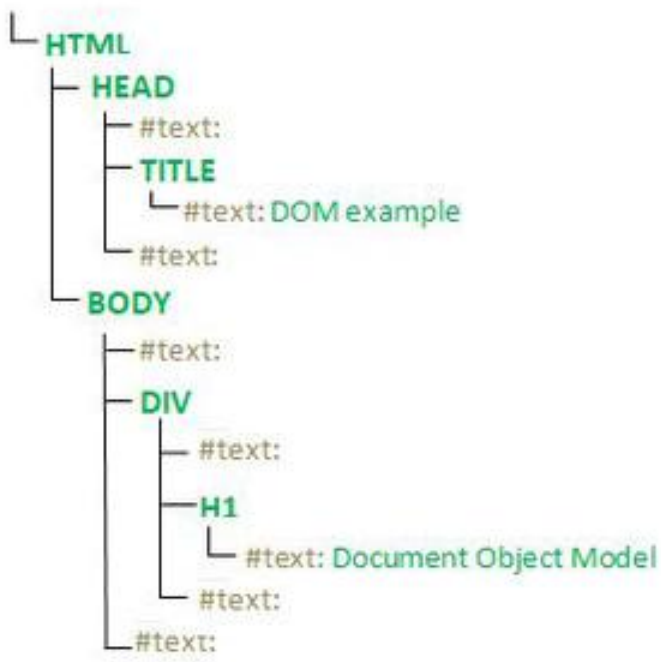
Prerequisites:

- DOM
- HTML
- React JS ReactDOM

Let's see how a document is parsed by a DOM. Consider the following sample HTML code.

```
<html>
<head>
  <title>DOM example</title>
<body>
  <div>
    <h1>Document Object Model</h1>
  </div>
</body>
</head>
</html>
```

The above code creates a tree structure as shown below:



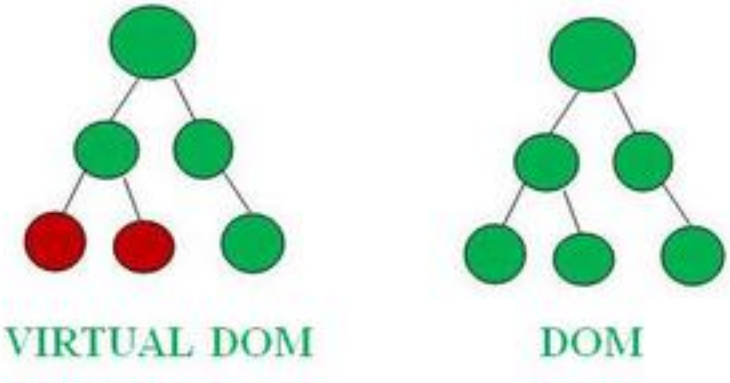
DOM Structure

Now we got a basic idea of what DOM is, If you are still unsure about it please learn here.

Virtual DOM:

The name itself says that it is a virtually created DOM. Virtual DOM is exactly like DOM and it has all the properties that DOM has. But the main difference is Whenever a code runs JavaScript Framework updates the whole DOM at once which gives a slow performance. whereas virtual DOM updates only the modified part of the DOM. Let's understand clearly:

When you run a code, the web page is divided into different modules. So, virtual DOM compares it with DOM and checks if there is any difference. If it finds a difference then DOM updates only the modified part and the other part remains the same.



As shown in the above image, virtual DOM is different from DOM, now DOM updates the child components which are different and the other remains exactly the same. This increases the performance.

Virtual DOM Key Concepts:

- Virtual DOM is the virtual representation of Real DOM
- React update the state changes in Virtual DOM first and then it syncs with Real DOM
- Virtual DOM is just like a blueprint of a machine, can do changes in the blueprint but those changes will not directly apply to the machine.
- Virtual DOM is a programming concept where a virtual representation of a UI is kept in memory synced with “Real DOM” by a library such as React DOM and this process is called reconciliation
- Virtual DOM makes the performance faster, not because the processing itself is done in less time. The reason is the amount of changed information – rather than wasting time on updating the entire page, you can dissect it into small elements and interactions

React JS vs React Native

Differences between React and React Native:

Category	React JS	React Native
Definition	A JavaScript library, widely used for developing the user interface.	A cross-platform mobile framework used for developing native mobile applications.
Platform	Since it is mainly used for web browsers, it can be easily executed on all platforms.	Since it is used for native applications, it takes a sufficient amount of developer effort to be customized and executed on all platforms.
User Interface	ReactJS renders HTML tags in its user interface. React components can include simple HTML tags.	React Native renders JSX in its user interface. React Native supports specific JSX tags that are used.
Styling	ReactJS uses Cascading Style Sheets (CSS).	React Native uses a Stylesheet object (JavaScript object).
Rendering	ReactJS uses VirtualDOM, a tool that allows for easy interaction with DOM elements.	React Native widely uses native APIs.

Category	React JS	React Native
Navigation	ReactJS uses the React router to allow users to visit different web pages.	React Native uses its built-in Navigator library to allow users to visit different screens.
External library support	ReactJS supports third-party packages but lacks native library support.	React Native lacks both native libraries and third-party packages.
Animation	Since ReactJS focuses on UI, it requires animations, which can be easily added using CSS.	To incorporate animations in React Native, it uses an animated API.
Security	It has comparatively higher security.	It has comparatively lower security.
Uses	It is widely used to develop a dynamic user interface for web applications.	It is used to develop true native mobile applications.
Applications	Facebook, Netflix, Medium, Udemy	Uber Eats, Tesla

React Native Debugging

Debugging is very important for building applications and removing errors. A good knowledge of debugging techniques allows for the faster and efficient development of software.

Here we are going to discuss a few debugging techniques in React Native. We will be using *expo-cli* to develop, run, and debug our applications, which is one of the simplest and fastest ways of building a React Native application.

The following are the debugging techniques generally used in React Native:

Logging

It is a very quick and easy technique to debug your application in the development phase. It is one of the easiest techniques to get an insight into the functioning of the application. To do logging, we simply use the console.log() statements to log the required information or indicators. However, we should always remember to remove these console.log() statements before we push our product into the development phase, as these statements will simply create an overhead there.

Chapter Two: React Native Components

Component

We know that React Applications are a collection of interactive components. With React Native, you can make components using either classes or functions. Originally, class components were the only components that could have state. But since the introduction of React Hooks API, you can add state and more to function components. The Components are the building blocks in React Native, which are similar to the container where all UI elements are gathered and help in rendering details in the foreground as a native UI on either Android or iOS devices. React comes with multiple built-in components such as Text, View, Image, ScrollView, TextInput, etc.

Importing Component

The world of JavaScript is always moving, and one of the latest ECMAScript versions now provides a more advanced module importing pattern. In the previous version, the developer had to use the following command.

```
module. exports = { // Define your exports here. };
const module = require('./file');
```

But now each module can have a default export or may export several named parameters, and if it is possible to export, it will surely be possible to import the same. Thus, with the recent ECMAScript version, every module may import the default export or several named parameters or even a valid combination.

Approach

React Native uses the same features as mentioned above, and you may treat each React Component as a module itself. Thus, it is possible to import React Native Components, and it is one of the basic operations to be performed. In React, we use the keyword **import** and **from** to import a particular module or a named parameter. Let's see the different ways we can use the import operation in a React Native Application.

Importing the default export

Each module in reacts native needs at least one default export. In order to import the default export from a file, we can use the location of the file and use the keyword import before it, or we could give a specific name i.e. COMP_NAME to the import which makes the syntax as the following.

```
import COMP_NAME from LOCATION
```

Importing named values

Every module can have no named parameters, and in case we need to import one we should use the syntax as follows.

```
import { COMP_NAME } from LOCATION
```

Similarly, for multiple imports, we can use a comma (**,**) separator to separate two-parameter names within the curly braces. As shown below.

```
import { COMP_NAME1, COMP_NAME2, ... , COMP_NAMEn } from LOCATION
```

Importing a combination of Default Exports and Named Values

The title makes it clear **that** what we need to see is the syntax of the same. In order to import a combination, we should use the following syntax.

```
import GIVEN_NAME, { PARA_NAME, ... } from ADDRESS
```

Example:

```tsx
// Importing components from react-native library.
import { Text, View } from "react-native";

export default function App() {
  return (
    // Using react-natives built in components.
    <View
      style={{
        flex: 0.5,
        justifyContent: "center",
        alignItems: "center",
        backgroundColor: "green",
      }}
    >
      <Text
        style={{
          color: "white",
        }}
      >
        Softi Academy
      </Text>
    </View>
  );
}
```

FlatList Component

FlatList is a React Native component that is a scrolling list that shows changing information while keeping the same look. It's great for long lists where the number of items can change. Instead of loading all items simultaneously, this component only shows what you can see on the screen. This makes it faster and gives users a better experience.

Syntax of FlatList:

```
<FlatList
      data={}
      renderItem={}
      keyExtractor={}
 />
```

FlatList Props:

Props	Type	Description
renderItem	function	Extracts an item from the data and displays it in the list.
data	ArrayLike	A list of items to render
ItemSeparatorComponent	component, function, element	Used to render in between each item, but not at the top or bottom. For example, a separate line between two items.
ListEmptyComponent	component, element	Used to render when the list is empty. For example, displaying "No Data Found" when the list is empty.
ListHeaderComponent	component, element	Used to render at the bottom of all the items. For example, "End of List"
ListFooterComponent	component, element	Used to render at the bottom of all the items. For example, "Heading of the List"
ListHeaderComponentStyle	ViewStyle	Used to Style ListHeaderComponent
ListFooterComponentStyle	ViewStyle	Used to Style ListFooterComponent

Props	Type	Description
columnWrapperStyle	ViewStyle	Used to customize the style for multi-item rows when numColumns > 1.
extraData	any	Used to re-render the list when the state changes.
getItemLayout	function	Used to optimize performance by providing item layout.
horizontal	boolean	Set to true for horizontal scrolling
initialNumToRender	number (default: 10)	Number of items to render initially
initialScrollIndex	number	Set the initial scroll index
inverted	boolean	Set to true for inverted scrolling
keyExtractor	function	Function to generate unique keys for items.
numColumns	number	Set the number of columns
onRefresh	function	Used to call for pull-to-refresh; requires refreshing prop.
refreshing	boolean	Set to true during refresh loading.

Props	Type	Description
removeClippedSubviews	boolean	Used to improve performance by removing invisible views (Android only).
onViewableItemsChanged	function	Used to call back when viewable items change.
viewabilityConfig	object	Used to configure for determining item viewability (e.g., visibility%, waitForInteraction).
viewabilityConfigCallbackPairs	array	It is an Array of ViewabilityConfig + callback pairs for tracking multiple configs.

FlatList Methods:

Method	Description
flashScrollIndicators()	Used to display the scroll indicators momentarily
getNativeScrollRef()	It provides a reference to the underlying scroll component
getScrollResponder()	It provides a handle to the underlying scroll responder
getScrollableNode()	It provides a handle to the underlying scroll node
scrollToEnd()	Scrolls to the end of the content

Method	Description
scrollToIndex()	Scrolls to the particular item index of the list