To create a React Native app on a Windows PC for an Android device the **React Native CLI** (for full native control):-

Setup on windows pc:-

1. Install Core Dependencies

Regardless of the method, you must install these tools on your Windows PC:

- Node.js: Download and install the **LTS version** from the Node.js official site. Verify with `node -v` in your terminal.
- **JDK 17**: React Native currently recommends **JDK 17** for the best compatibility. After installing, set the `JAVA_HOME` environment variable to your JDK path (e.g., `C:\Program Files\Java\jdk-17.x.x`).

NOTE YOU CAN USE ANDROID STUDIO JBR ALOS IN JAVA_HOME :-

1. **Create/Edit JAVA_HOME**:

1. Under **System variables**, click **New** (or **Edit** if it already exists).
2. **Variable name**: `JAVA_HOME`
3. **Variable value**: `C:\Program Files\Android\Android Studio\jbr`

NOTE:-

Using `C:\Program Files\Android\Android Studio\jbr` is an excellent choice. This is the **JetBrains Runtime**, a customized OpenJDK 17 that comes bundled with Android Studio.

- **Android Studio**: Download it from the Android Studio website. During setup, ensure **Android SDK**, **Android SDK Platform**, and **Android Virtual Device** are checked.

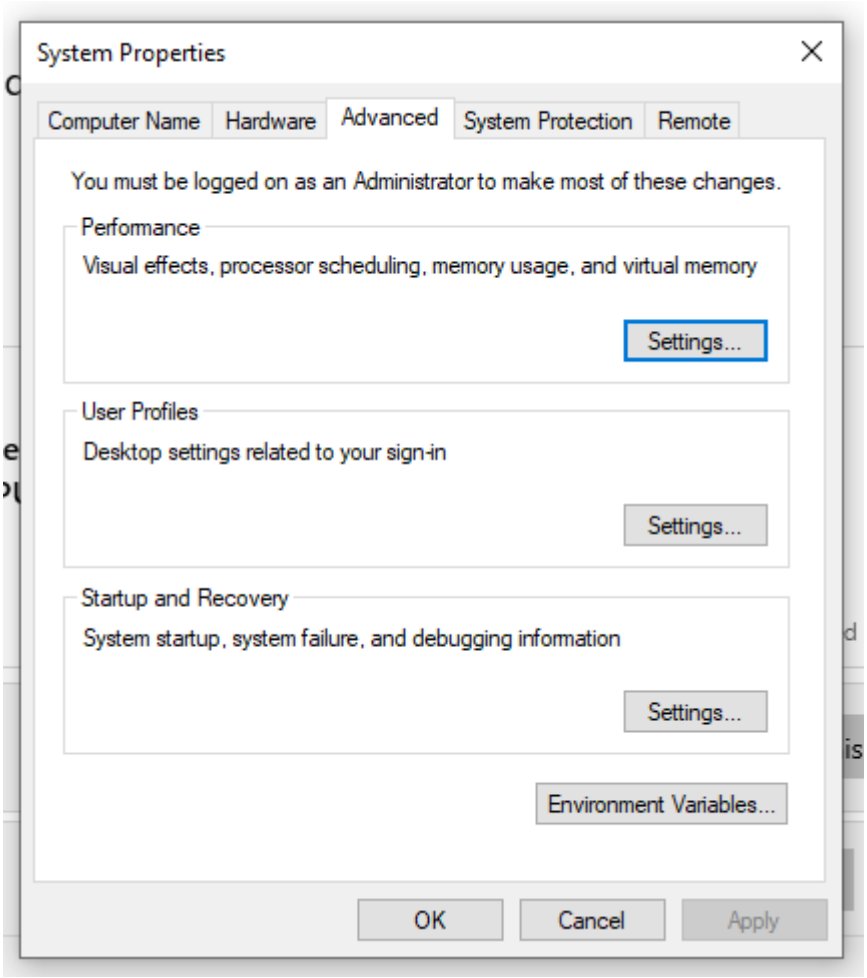
2. Configure Android Environment Variables

To allow React Native to communicate with Android tools, you must set these in Windows System Properties:

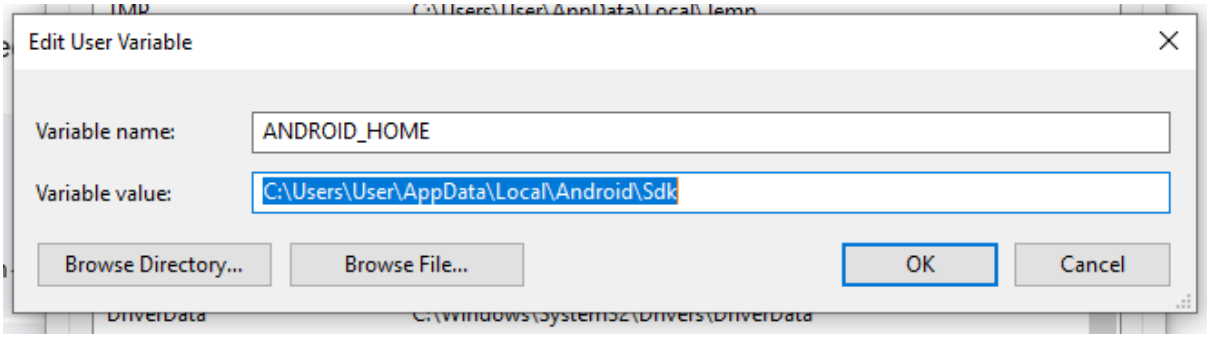
1. **ANDROID_HOME**: Create a new user variable pointing to your SDK path (typically `%LOCALAPPDATA%\Android\Sdk`).
2. **Path**: Edit your system `Path` variable and add these four entries:

1. `%ANDROID_HOME%\platform-tools`
2. `%ANDROID_HOME%\emulator`
3. `%ANDROID_HOME%\tools`

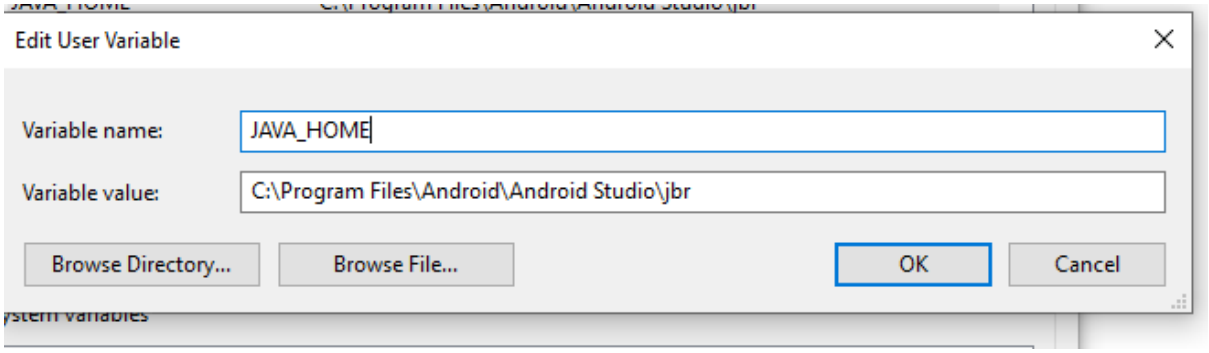
AS SHOWN STEP BY STEP :-



Add ANDROID_HOME

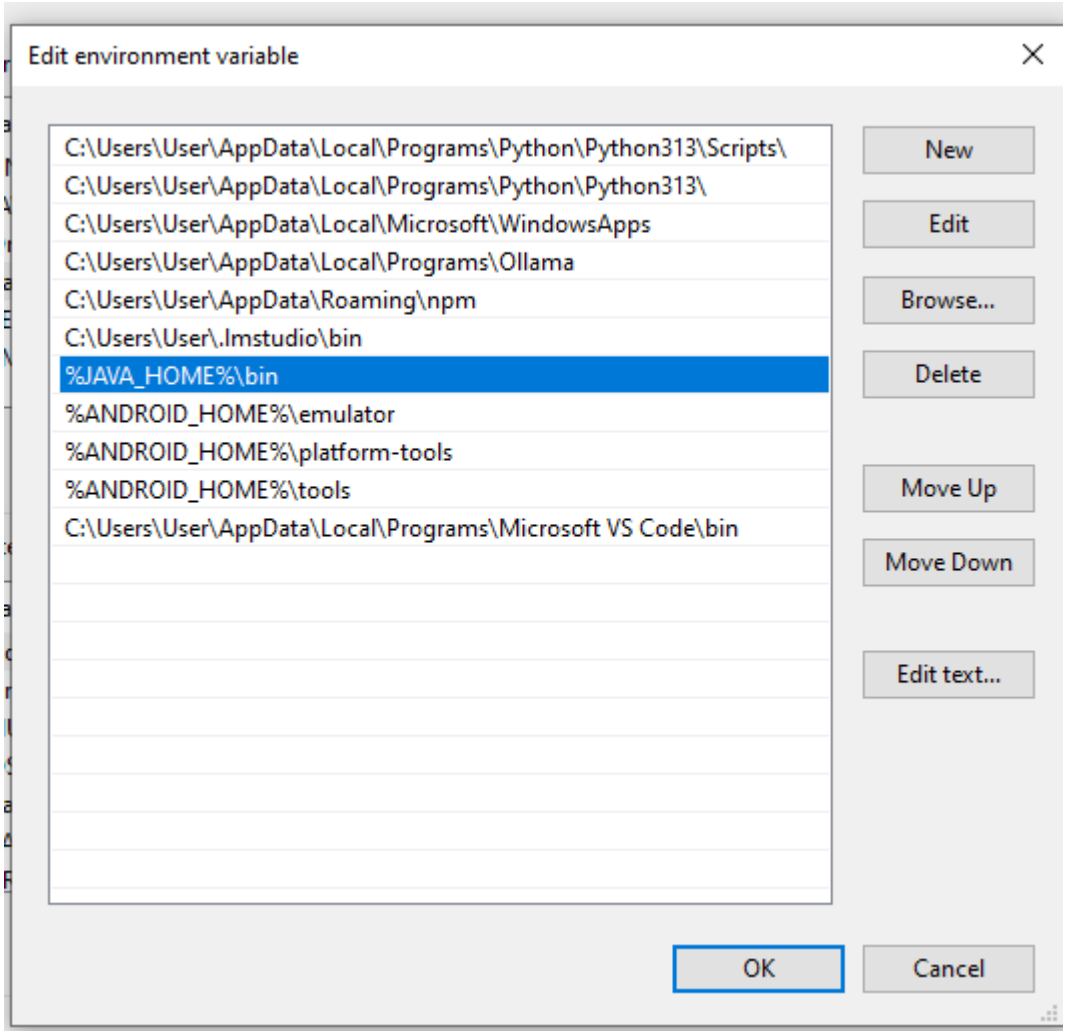


Add JAVA_HOME



AND NOW CLICK ON PATH AND CLICK ON ADD NEW

AND ADD FOLLOWING



Get your Project started, Run this Command Instead:-

```
npx @react-native-community/cli@latest init MyTutor
```

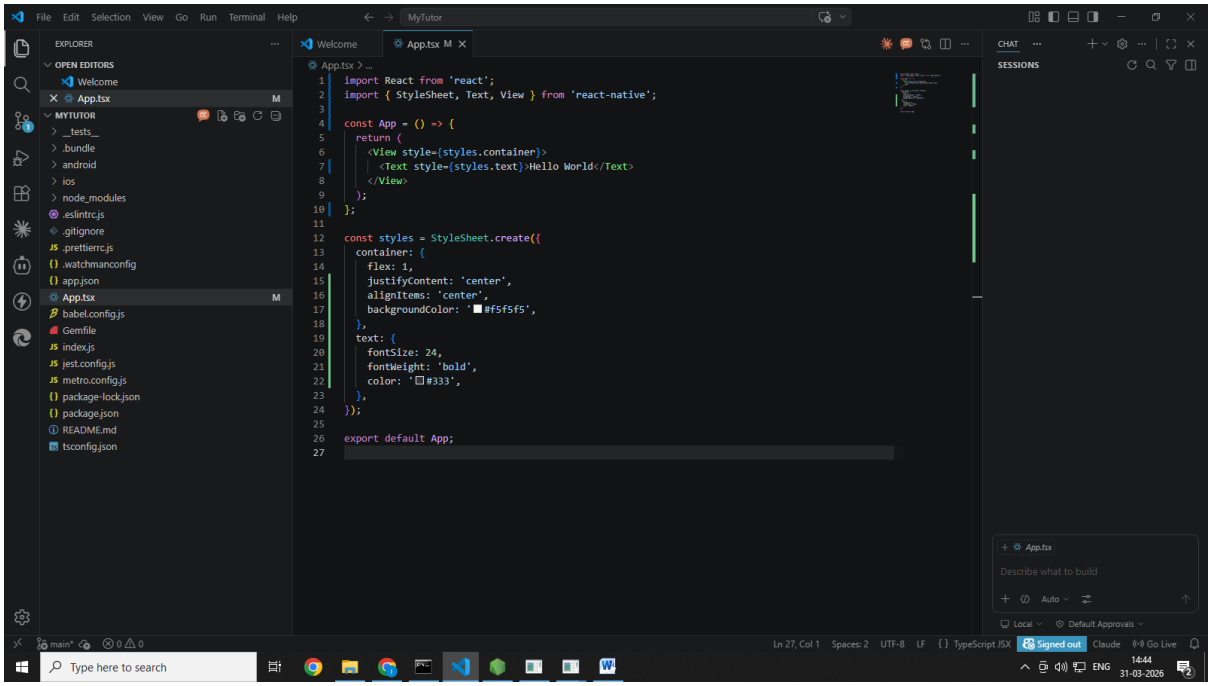
```
F:\react-native-warm-up>npx @react-native-community/cli@latest init MyTutor
Need to install the following packages:
@react-native-community/cli@20.1.3
Ok to proceed? (y)
```

Ok to proceed? Then press y (yes)

Once the process finishes, navigate into your folder:-

```
cd MyTutor
```

and open your project folder in vs code:-
as shown below.


File Edit Selection View Go Run Terminal Help
MyTutor
EXPLORER
OPEN EDITORS
Welcome
App.tsx M
MYTUTOR
__tests__
.bundle
android
ios
node_modules
.eslintrc.js
.gitignore
.prettierrc.js
.watchmanconfig
app.json
App.tsx M
babel.config.js
Gemfile
index.js
jest.config.js
metro.config.js
package-lock.json
package.json
README.md
tsconfig.json
Welcome
App.tsx M
App.tsx > ...
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
const App = () => {
return (
<View style={styles.container}>
<Text style={styles.text}>Hello World</Text>
</View>
);
};
const styles = StyleSheet.create({
container: {
flex: 1,
justifyContent: 'center',
alignItems: 'center',
backgroundColor: '#f5f5f5',
},
text: {
fontSize: 24,
fontWeight: 'bold',
color: '#333',
},
});
export default App;
CHAT
SESSIONS
App.tsx
Describe what to build
Auto
Local
Default Approvals
main*
Ln 27, Col 1
Spaces: 2
UTF-8
LF
TypeScript JSX
Signed out
Claude
Go Live
Type here to search
ENG
14:44
31-03-2026

And now open your App.tsx file and write following code to display hello world.

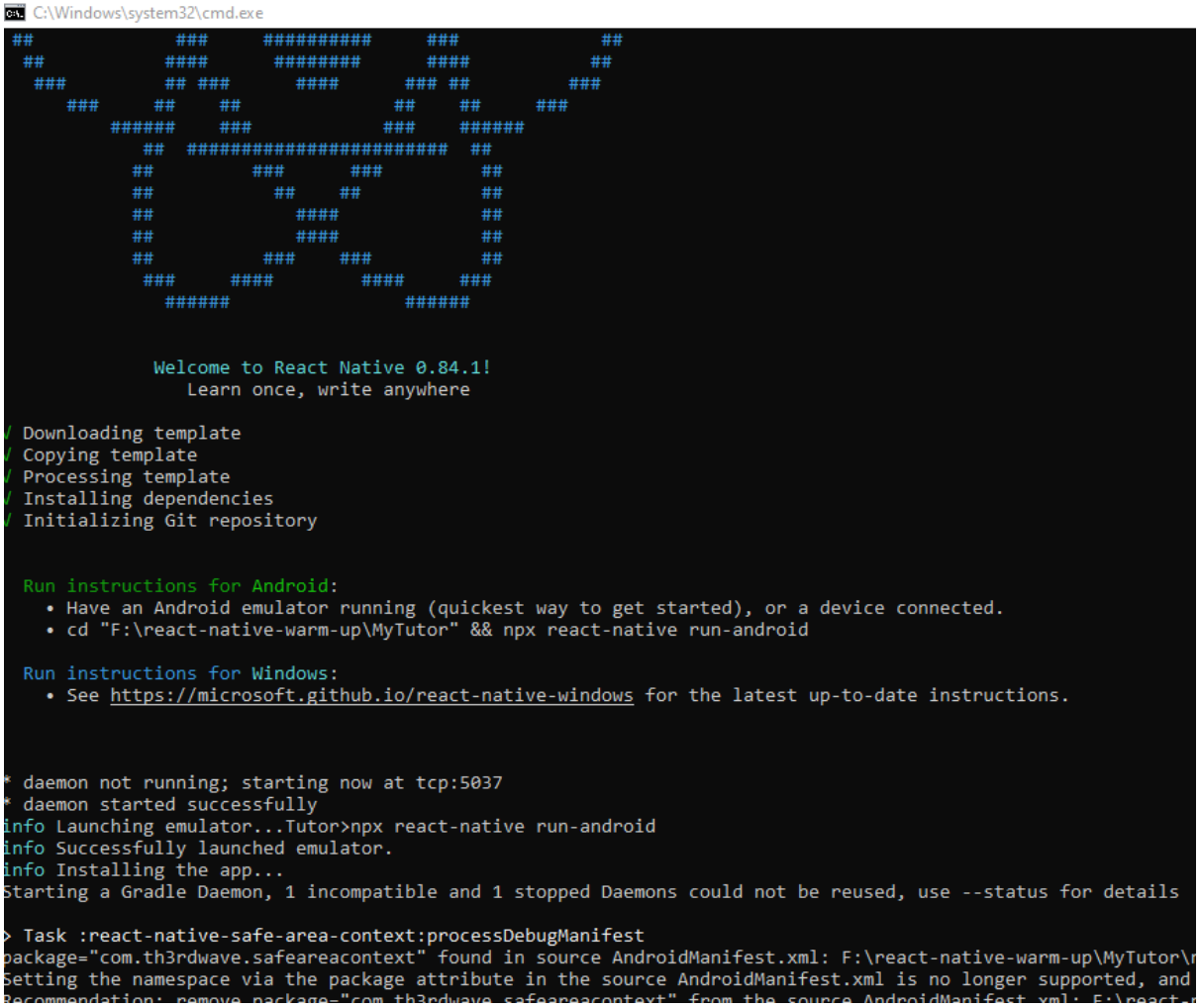
```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Hello World</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f5f5f5',
  },
  text: {
    fontSize: 24,
    fontWeight: 'bold',
    color: '#333',
  },
});

export default App;
```

Now run project:-

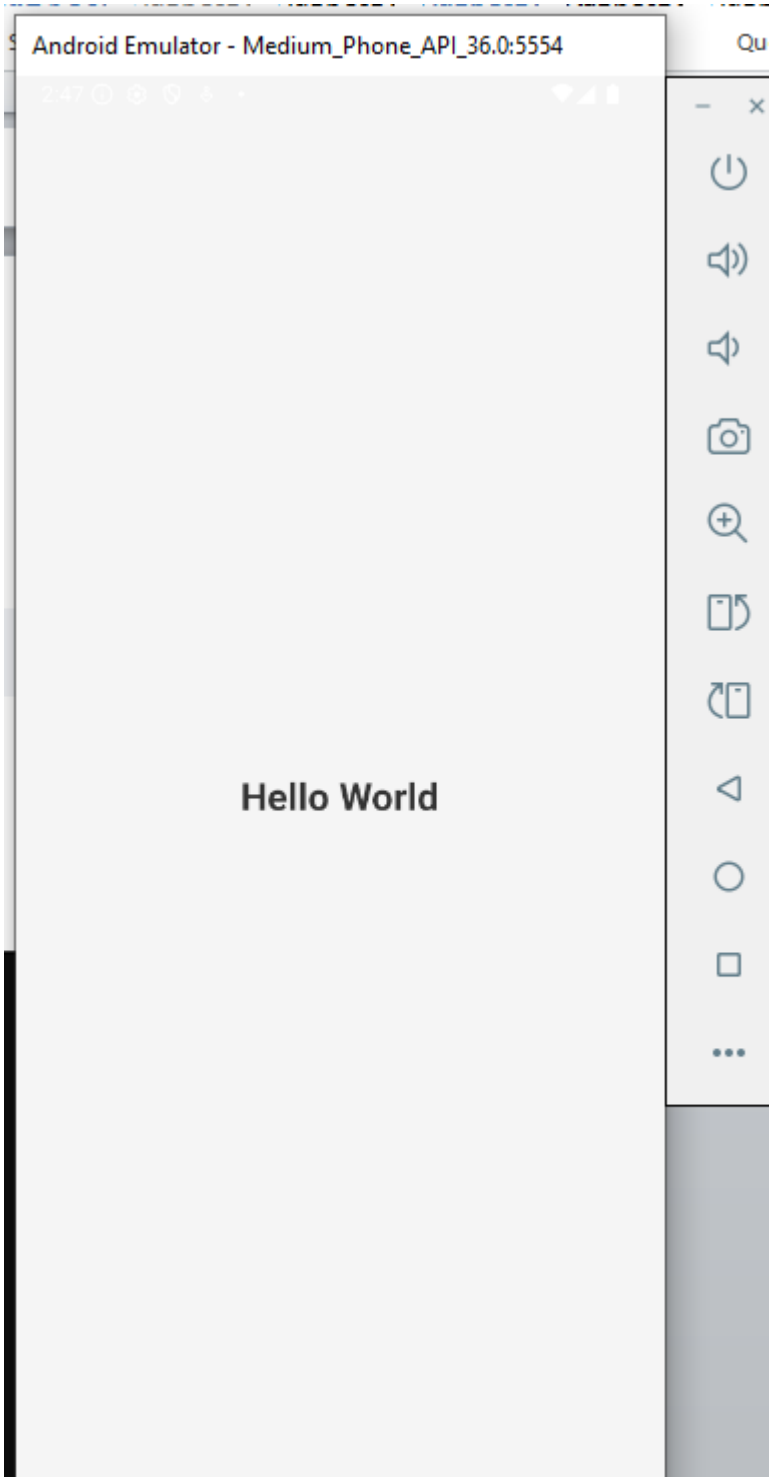
Run the App

Navigate into your project folder and start the build process:

```
cd MyTutor
npx react-native run-android
```



Oputput:-


Android Emulator - Medium_Phone_API_36.0:5554
Qu
Hello World

Now you can test Usestate concept in this :-

Simple design:-

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

const App = () => {
  // 1. State hook to track the number
  const [count, setCount] = useState<number>(0);

  // 2. Function to Increase
  const increment = () => {
    setCount(count + 1);
  };

  // 3. Function to Decrease
  const decrement = () => {
    // Optional logic: prevent count from going below 0
    if (count > 0) {
      setCount(count - 1);
    }
  };

  return (
    <View style={styles.center}>
      <Text style={styles.counterText}>{count}</Text>

      {/* 4. Two buttons using the functions */}
      <View style={styles.buttonContainer}>
        <Button title="Decrease" onPress={decrement} color="red" />
        <View style={{ width: 20 }} /> {/* Simple spacer */}
        <Button title="Increase" onPress={increment} color="green" />
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  center: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff'
```

```
  },
  counterText: {
    fontSize: 60,
    fontWeight: 'bold',
    marginBottom: 20
  },
  buttonContainer: {
    flexDirection: 'row', // Aligns buttons side-by-side
    alignItems: 'center'
  }
});

export default App;
```

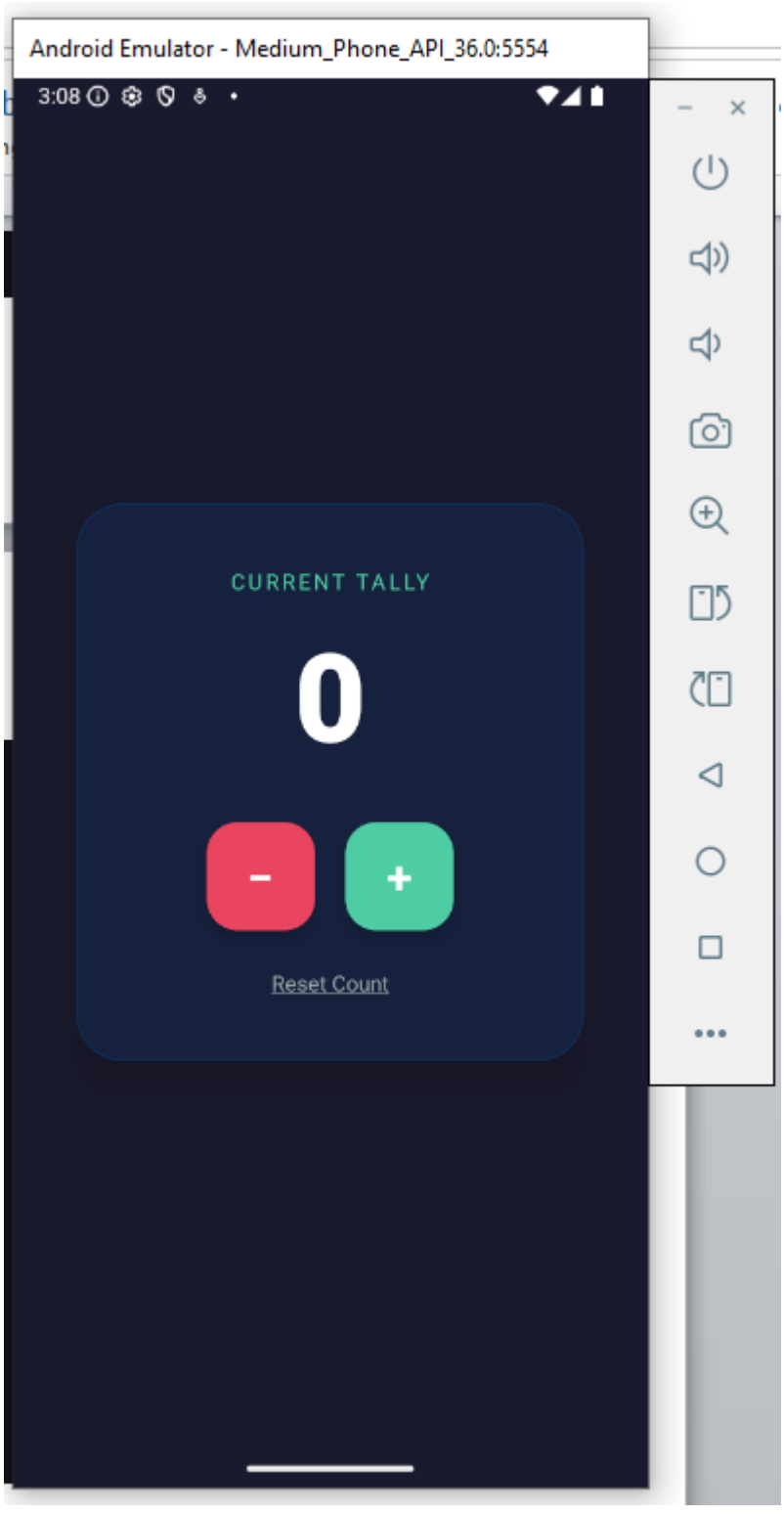
Advance design:-

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

const App = () => {
  const [count, setCount] = useState<number>(0);

  return (
    <View style={styles.container}>
      {/* Set status bar color to match background */}
      <StatusBar barStyle="light-content" backgroundColor="#1a1a2e" />

      <View style={styles.card}>
        <Text style={styles.label}>Current Tally</Text>
        <Text style={styles.counterText}>{count}</Text>

        <View style={styles.buttonRow}>
          {/* Custom Minus Button */}
          <TouchableOpacity
            style={[styles.button, styles.minusBtn]}
            onPress={() => count > 0 && setCount(count - 1)}
            activeOpacity={0.7}
          >
            <Text style={styles.buttonText}>-</Text>
          </TouchableOpacity>

          {/* Custom Plus Button */}
          <TouchableOpacity
            style={[styles.button, styles.plusBtn]}
            onPress={() => setCount(count + 1)}
```

```
            activeOpacity={0.7}
          >
            <Text style={styles.buttonText}>+</Text>
          </TouchableOpacity>
        </View>

        {/* Reset Link */}
        <TouchableOpacity onPress={() => setCount(0)}>
          <Text style={styles.resetText}>Reset Count</Text>
        </TouchableOpacity>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#1a1a2e', // Deep dark blue
    justifyContent: 'center',
    alignItems: 'center',
  },
  card: {
    backgroundColor: '#16213e',
    padding: 40,
    borderRadius: 30,
    alignItems: 'center',
    width: '80%',
    elevation: 20, // Android shadow
    shadowColor: '#000', // iOS shadow
    shadowOffset: { width: 0, height: 10 },
    shadowOpacity: 0.5,
    shadowRadius: 10,
    borderWidth: 1,
    borderColor: '#0f3460',
  },
  label: {
    color: '#4ecca3',
    fontSize: 14,
    textTransform: 'uppercase',
    letterSpacing: 2,
    marginBottom: 10,
  },
  counterText: {
    fontSize: 80,
    fontWeight: '900',
    color: '#fff',
    marginBottom: 30,
```

```
  },
  buttonRow: {
    flexDirection: 'row',
    gap: 20, // Modern way to add spacing
  },
  button: {
    width: 70,
    height: 70,
    borderRadius: 20,
    justifyContent: 'center',
    alignItems: 'center',
    elevation: 5,
  },
  minusBtn: {
    backgroundColor: '#e94560',
  },
  plusBtn: {
    backgroundColor: '#4ecca3',
  },
  buttonText: {
    fontSize: 30,
    color: '#fff',
    fontWeight: 'bold',
  },
  resetText: {
    marginTop: 25,
    color: '#95a5a6',
    textDecorationLine: 'underline',
  },
});

export default App;
```

output:-


Android Emulator - Medium_Phone_API_36.0:5554
3:08
CURRENT TALLY
0
-
+
Reset Count

1. Master the "Big Three" Components

In React Native, you don't use HTML tags. You must learn these equivalents:

- **`<View>`**: The `<div>` (Layout container).
- **`<Text>`**: The `<span>` or `<p>` (All text must go here).
- **`<Image>`**: The `<img>` tag.
- **`<ScrollView>`**: Essential because screens don't scroll automatically in mobile apps.

1. The `<View>` Component (The Container)

Web Equivalent: `<div>` or `<section>`

The `View` is the most fundamental component. It is used for layout, grouping other components, and styling (flexbox).

- **Rule:** You cannot put raw text directly inside a `<View>`.
- **Default Layout:** Unlike web `divs` (which stack vertically by default), `View` uses **Flexbox** where the default direction is **Column**.

Example:

tsx

```
<View style={{ flexDirection: 'row', padding: 20, backgroundColor: 'skyblue' }}>
  <View style={{ width: 50, height: 50, backgroundColor: 'red' }} />
  <View style={{ width: 50, height: 50, backgroundColor: 'blue' }} />
</View>
```

Use code with caution.

---

2. The `<Text>` Component (The Content)

Web Equivalent: `<span>`, `<p>`, `<h1>` to `<h6>`

In React Native, **all strings must be wrapped in `<Text>`**. If you try to put text inside a `<View>` without this, the app will crash.

- **Nesting:** You can nest `<Text>` inside `<Text>` to inherit styles.
- **No Cascading:** Styles don't automatically "trickle down" from a `View` to a `Text` like they do in CSS. You must style the Text component directly.

Example:

tsx

```tsx
<Text style={{ fontSize: 20, color: 'black' }}>
  This is normal text.
  <Text style={{ fontWeight: 'bold', color: 'red' }}> This part is bold and red.</Text>
</Text>
```

Use code with caution.

---

3. The `<Image>` Component (The Visuals)

Web Equivalent: `<img>`

Used to display local images or images from the web.

- **Local Images:** Use `require('./path/to/img.png')`.
- **Network Images:** Use an object with a `uri`. **Note:** Network images **must** have a defined `width` and `height` or they won't show up.

Example:

tsx

```tsx
<View>
  {/* Local Image */}
  <Image
    source={require('./assets/logo.png')}
    style={{ width: 100, height: 100 }}
  />

  {/* Web Image */}
  <Image
    source={{ uri: 'https://reactnative.dev' }}
    style={{ width: 50, height: 50, borderRadius: 25 }}
  />
</View>
```

Use code with caution.

---

4. The `<ScrollView>` (The Scroller)

Web Equivalent: `overflow-y: scroll` (but automatic on web)

In mobile, if your content is longer than the screen, it stays cut off. You **must** wrap it in a `ScrollView` to enable scrolling.

- **Rule:** Use this for small amounts of content (like a settings page or a long form).
- **Performance:** For very long lists (100+ items), use `FlatList` instead of `ScrollView` for better speed.

Example:

tsx

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

const LongPage = () => {
  return (
    <ScrollView style={{ flex: 1 }}>
      <View style={{ height: 1000, backgroundColor: '#eee' }}>
        <Text>Scroll down to see more!</Text>
        {/* Imagine lots of content here */}
        <Text style={{ marginTop: 900 }}>You reached the bottom!</Text>
      </View>
    </ScrollView>
  );
};
```

Summary Table for Quick Reference

Feature	React Native Component	Web (HTML) Equivalent	Must-Know Tip
Container	`<View>`	`<div>`	Uses Flexbox (Column) by default.
Text	`<Text>`	`<p>`, `<span>`	Essential; app crashes without it.
Images	`<Image>`	`<img>`	Web images need fixed width/height.
Scrolling	`<ScrollView>`	Browser default	Screens do NOT scroll by themselves.

2. Understand TypeScript Props

Since you are in `.tsx`, you define what data a component can receive. This prevents bugs before you even run the app.

Try this exercise: Create a folder named `components` and a file `Greeting.tsx`:

tsx

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

// Define the "Shape" of the data this component needs
interface GreetingProps {
  name: string;
  age?: number; // The '?' means age is optional
}

const Greeting = ({ name, age }: GreetingProps) => {
  return (
    <Text>Hello {name}! {age ? `You are ${age} years old.` : ''}</Text>
  );
};

export default Greeting;
```

To use your custom `Greeting` component in your `App.tsx` file, you need to **import** it and then place it inside a `<View>` container (since React Native requires a root wrapper for multiple elements).

Here is exactly how your `App.tsx` should look:

tsx

```tsx
import React from 'react';
import { StyleSheet, View } from 'react-native';
// 1. Import your custom component
import Greeting from './Greeting';

const App = () => {
  return (
```

```
    <View style={styles.container}>
      {/* 2. Use it like a custom HTML tag */}
      {/* Pass the 'name' and 'age' as props */}
      <Greeting name="John" age={25} />

      {/* Since 'age' is optional (?), you can leave it out too */}
      <Greeting name="Sarah" />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff',
  },
});

export default App;
```

Important Tips for `.tsx`:-

1. **File Location:** Make sure `Greeting.tsx` and `App.tsx` are in the same folder. If `Greeting.tsx` is in a folder named `components`, change the import to `./components/Greeting`.
2. **Type Checking:** Try changing `age={25}` to `age="25"` (a string). You will see a **red squiggly line** in VS Code—that's TypeScript protecting you from passing the wrong data type!
3. **Self-Closing:** Notice the tag ends with `/>`. Since we aren't putting anything *inside* the Greeting tag, we close it immediately.

3. Learn Interactive State (`useState`)

Mobile apps are all about interaction. Use the `useState` hook to change things on the screen.

Exercise: Add a counter to your `App.tsx`:

tsx

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

const App = () => {
  // TypeScript infers 'count' is a number because 0 is a number
  const [count, setCount] = useState(0);

  return (
    <View style={styles.center}>
      <Text style={{ fontSize: 40 }}>{count}</Text>
      <Button title="Click Me" onPress={() => setCount(count + 1)} />
    </View>
  );
};

const styles = StyleSheet.create({
  center: { flex: 1, justifyContent: 'center', alignItems: 'center' }
});

export default App;
```

4. Recommended Free Resources

- **Official Docs (The Gold Standard):** Go to reactnative.dev. It has an interactive simulator where you can test code in the browser.
- **React Navigation:** This is the most important library to learn next. It’s how you move from "Screen A" to "Screen B."

- **Native Stack:** Look up "React Navigation Stack" to learn how mobile headers and back buttons work.