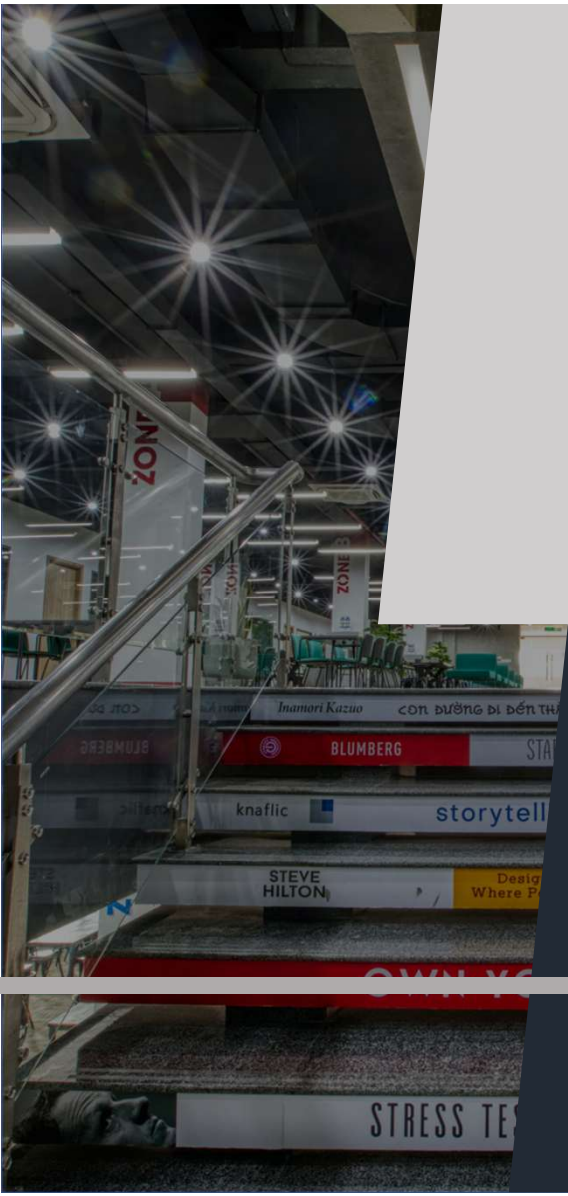


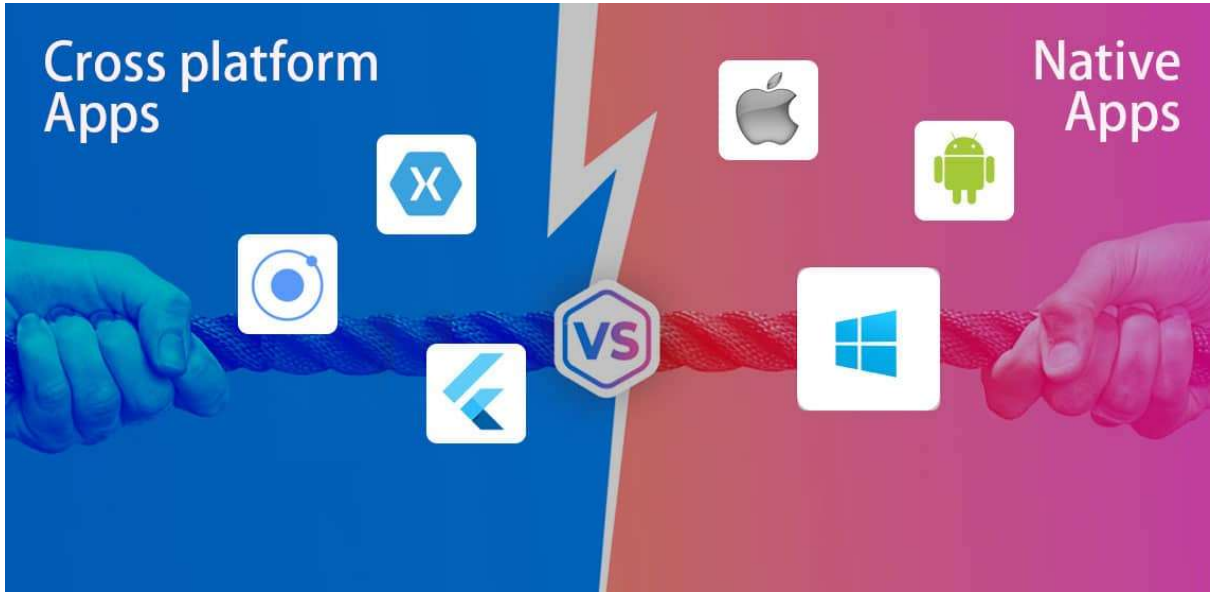
EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Mobile Programming

Chapter 1: INTRODUCTION TO MOBILE PROGRAMMING

I. Introduction

- In the world of mobile app development, we have two primary approaches: multi-Platform (Cross-Platform) and Native Development



Native Mobile Programming

- Native development involves creating separate versions of an app for each platform—iOS and Android.
- Advantages: Maximum Performance, Full Access to Platform Features, Native Look and Feel, Platform-Specific Optimizations
- Disadvantages: Higher Development Cost , Longer Development Time, Platform-Specific Codebase, Double Maintenance Effort

Multi-Platform (Cross-Platform) Mobile Programming

- Multi-platform development, also known as cross-platform development, enables you to write code once and deploy it across multiple platforms.
- React native
- Flutter
- Xamarin
- Kotlin
- Advantages: Cost and Time Efficiency, Shared Codebase, Easier Maintenance, Rapid Prototyping
- Disadvantages: Slightly Lower Performance, Limited Access to Platform-Specific Features, Potential Compatibility Issues

What is React Native?

- React Native is an open-source framework for building mobile apps.
- Developed by Facebook, it allows you to use React and JavaScript/Typescript to create native-quality mobile apps for iOS and Android.

Advantages of React Native

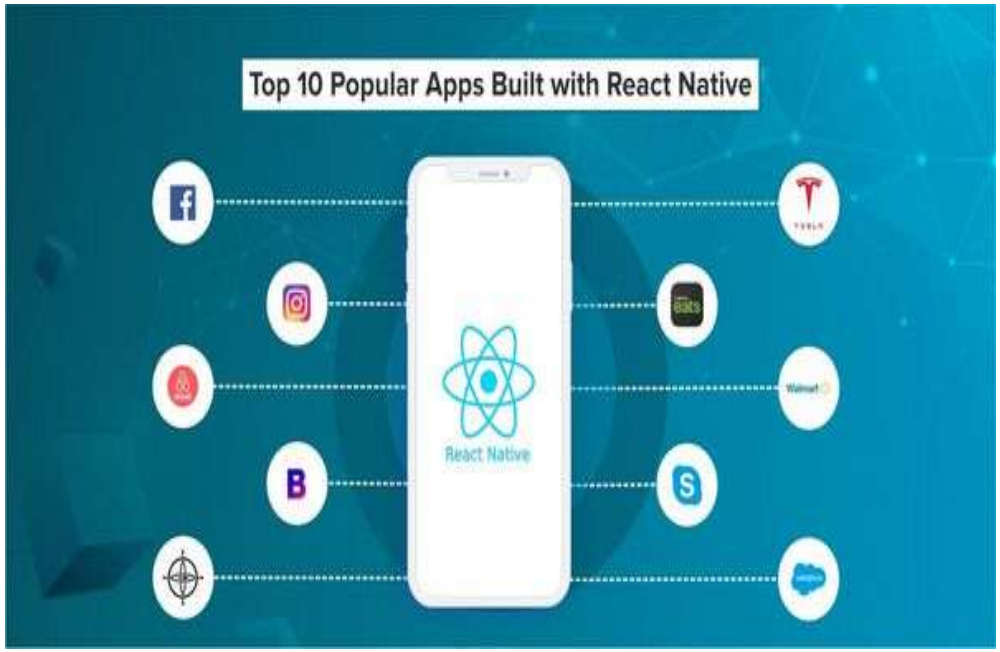
- Cross-Platform Development: Write code once, run it on both iOS and Android.
- Reusable Components: Components can be reused across platforms.
- Hot Reloading: Real-time code changes without app restart.
- Strong Community: Active community and third-party libraries.
- Performance: Near-native performance thanks to native modules.
- Faster Development: Streamlined development process.

Disadvantages of React Native

- Limited Access: Some native features may require native code.
- Debugging: Debugging can be complex compared to web development.
- Frequent Updates: Keeping up with platform updates can be challenging.
- Navigation: Complex navigation can be harder to implement.
- Third-Party Dependencies: Reliance on third-party libraries.

Use Cases

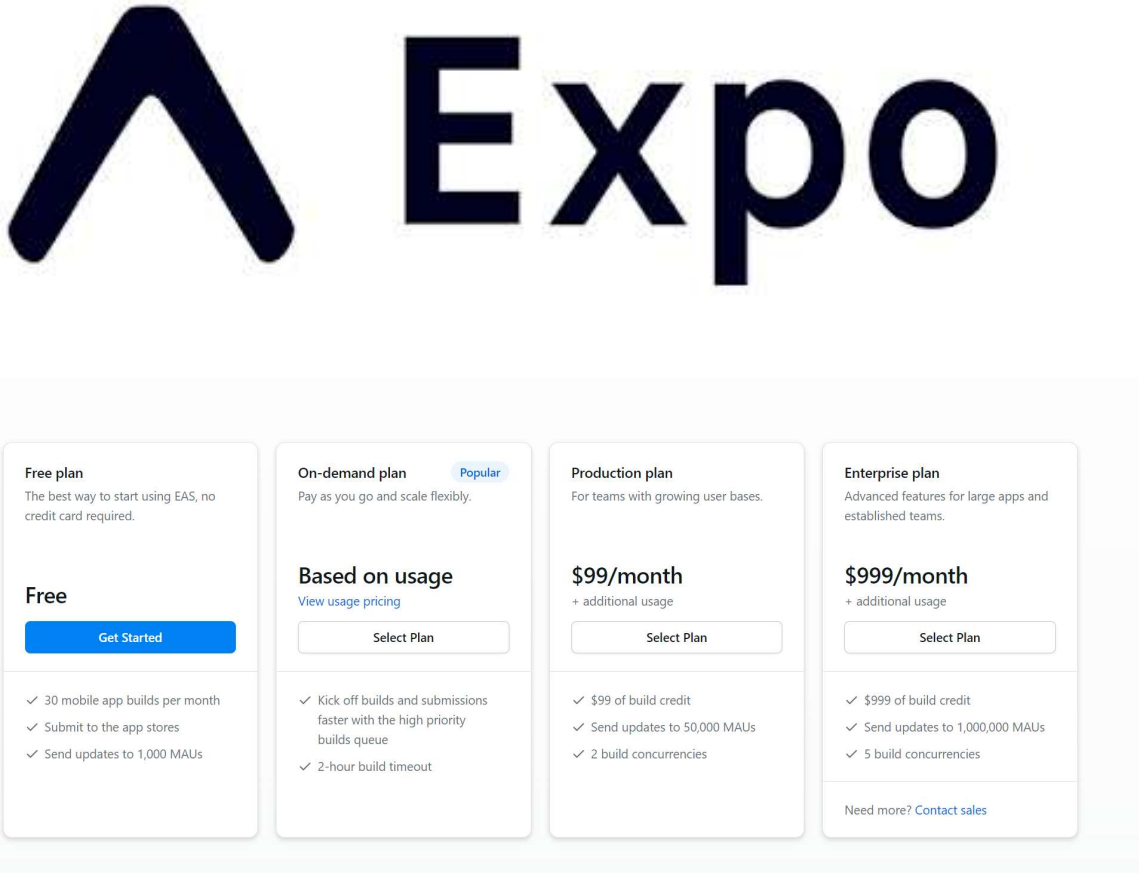
- Prototyping: Quickly develop and test app ideas.
- MVPs: Build Minimum Viable Products rapidly.
- Content-Driven Apps: News, blogs, social media.
- E-commerce: Shopping apps with dynamic content.
- Showcase examples like Facebook, Instagram, Airbnb, and Tesla using React Native for their apps.



II. Setting up the development environment

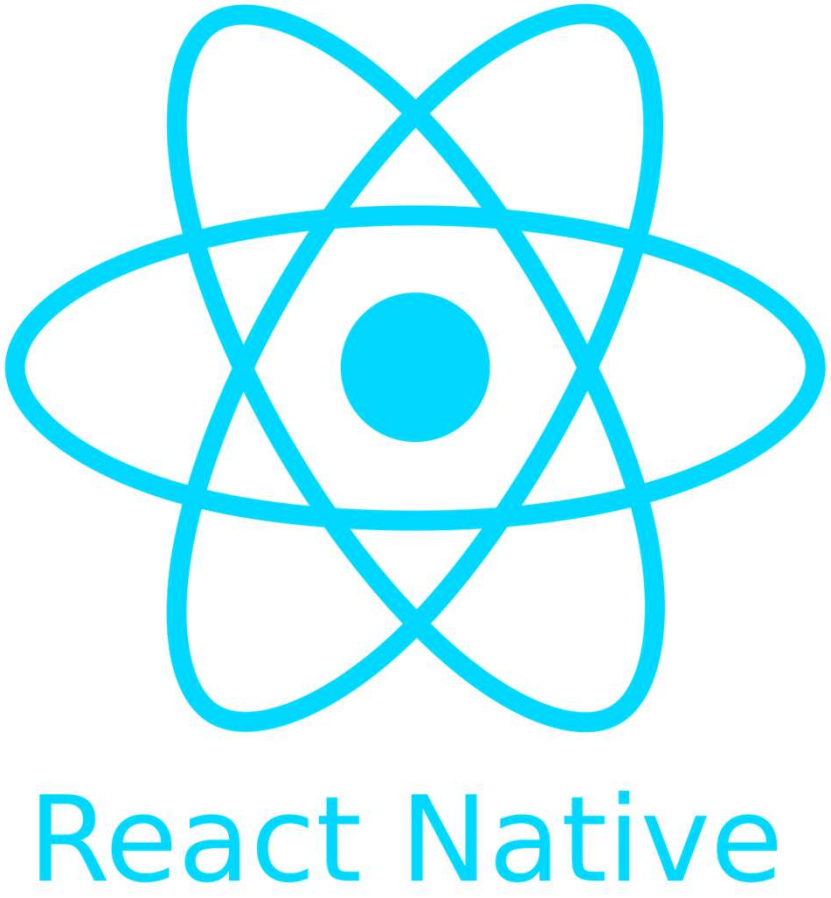
Expo CLI

- Expo CLI is a command-line tool provided by Expo(**https://expo.dev/),** a framework built around React Native. It simplifies the development process and provides additional features.
- Ease of Setup, Development Speed, Over-the-Air Updates
- Limitations: Expo CLI may limit access to some native modules and configurations



React Native CLI

- React Native CLI is the official command-line tool for React Native development. It provides core functionality for creating, managing, and building React Native apps.
- An open source tool
- Full Control
- Native Module Integration
- Access to Third-Party Libraries



Prerequisites

- Node.js (LTS version)
- npm (Node Package Manager)
- Yarn (optional but recommended)
- Xcode (for macOS users)
- Android Studio (for Android development)

Install Chocolatey, Node.Js, NpN, Yarn, Expo CLI, React Native CLI

- Download and install from: https://chocolatey.org/install
- Open an Administrator Command Prompt (right click Command Prompt and select "Run as Administrator")
- Install Node.js.
- *choco install -y nodejs-lts microsoft-openjdk17*
- Install Yarn (Optional)
  - *choco install yarn*
- Install Expo CLI (Optional): *npm install -g expo-cli*
- Install the React Native CLI globally: *npm install -g react-native-cli*

Install Android Studio (All Platforms)

- For Android development, install Android Studio from https://developer.android.com/studio."

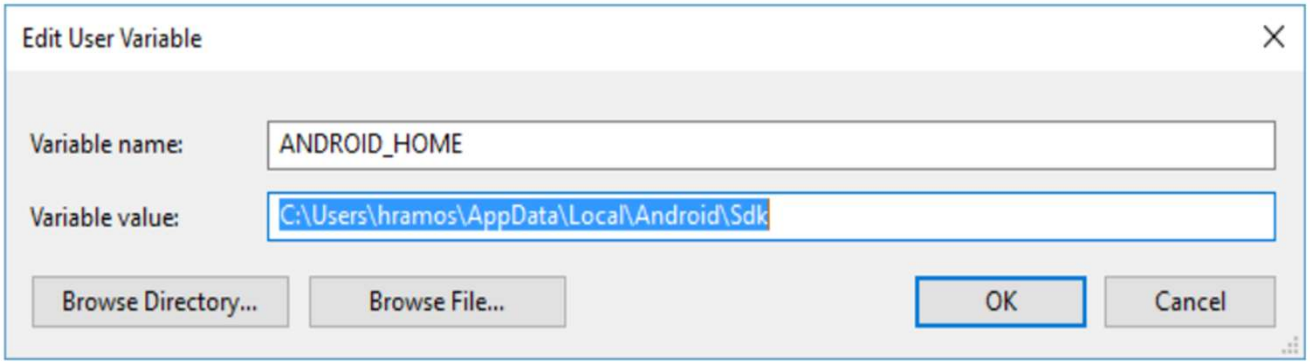
Select the "SDK Platforms" tab from within the SDK Manager, then check the box next to "Show Package Details" in the bottom right corner. Look for and expand the `Android 14 (UpsideDownCake)` entry, then make sure the following items are checked:

- `Android SDK Platform 34`
- `Intel x86 Atom_64 System Image` or `Google APIs Intel x86 Atom System Image`

Configure the ANDROID_HOME environment variable

The React Native tools require some environment variables to be set up in order to build apps with native code.

1. Open the **Windows Control Panel.**
2. Click on **User Accounts,** then click **User Accounts** again
3. Click on **Change my environment variables**
4. Click on **New...** to create a new `ANDROID_HOME` user variable that points to the path to your Android SDK:



III. Structure a program

```
mport React from 'react';
mport { Text, View } from 'react-nat

onst HelloWorld=()=>{
 return (
   <View style={{alignItems: 'center
   backgroundColor: '#F5FCFF', top:1
     <Text>Hello, World!</Text>
   </View>
 );
;

xport default HelloWorld;
```

Creating a Project with Expo CLI

npm Yarn

```
npx create-expo-app AwesomeProject

cd AwesomeProject
npx expo start
```

- Run the online application at https://snack.expo.dev/

Create a New React Native App with React native CLI

- Present the steps to create a new app:
- Let's create your first React Native app with the following command:

    npx react-native init MyFirstApp

Project Structure

- "Your project is structured like this:
  - `MyFirstApp/`
    - `android/`: Android-specific files
    - `ios/`: iOS-specific files
    - `node_modules/`: Third-party libraries
    - `App.js`: Main app component"

Open and Edit App.js/App.tsx file

demo2 > JS App.js > ...

```
1  import { StatusBar } from 'expo-status-bar';
2  import React from 'react';
3  import { StyleSheet, Text, View } from 'react-native';
4
5  export default function App() {
6    return (
7      <View style={styles.container}>
8        <Text>Open up App.js to start working on your app!</Text>
9        <StatusBar style="auto" />
10     </View>
11   );
12 }
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     backgroundColor: '#fff',
18     alignItems: 'center',
19     justifyContent: 'center',
20   },
21 });
22
```

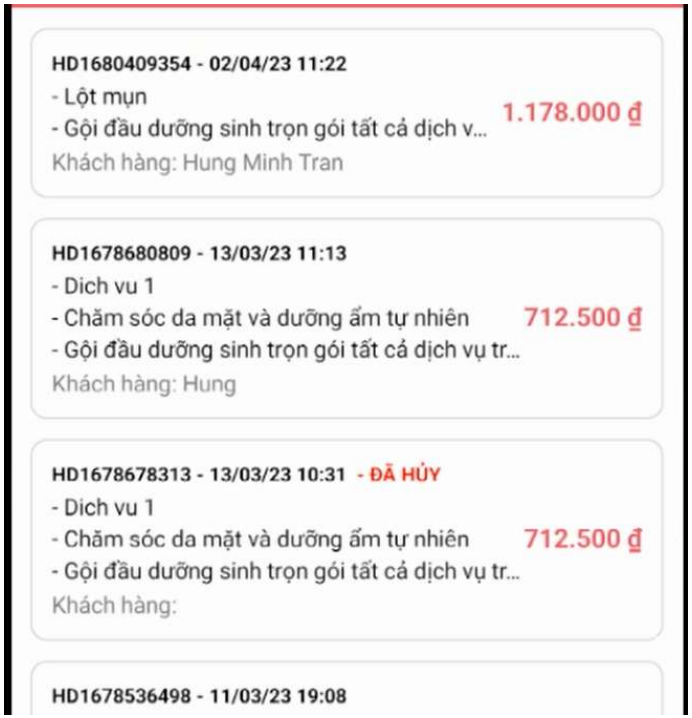
Allows import/export components (functions, classes, variables) from one file to another for use.

Run the App

- To run your app, use:

npx react-native run-android [--port xxxx] (for Android)

npx react-native run-ios (for iOS)"



Imports and Exports

- ES6 provides two ways to export multiple components from one file:
- **named export**, **default export**.
  - Named export: use export and {}

```
//functionsFile.js

//exporting a function
export function squareNumber(x) {
  return x * x;
}

//exporting a variable
export const pi = 3.14;

//Cách khác để export:

//exporting a function
function squareNumber(x) {
  return x * x;
}

//exporting a variable
const pi = 3.14;
export {squareNumber, pi} ;
```

Import with Named export

- Syntax: import { <Name 1> , <name 2>…} form “fileJS”;

```
//main.js
import {squareNumber, pi} from "functionsFile";
const radius = 7;
console.log("Area of a circle is", pi * squareNumber(7));

//Cách khác để import

import * as mathFuncs from "functionsFile";
console.log("Area of circle is ", mathFuncs.pi * mathFuncs.squareNumber(7));
```

Use * to import all

Default export

- Systax: import, export default …

```
//functionsFile.js

export default function(x) {
  return x * x ;
}
```

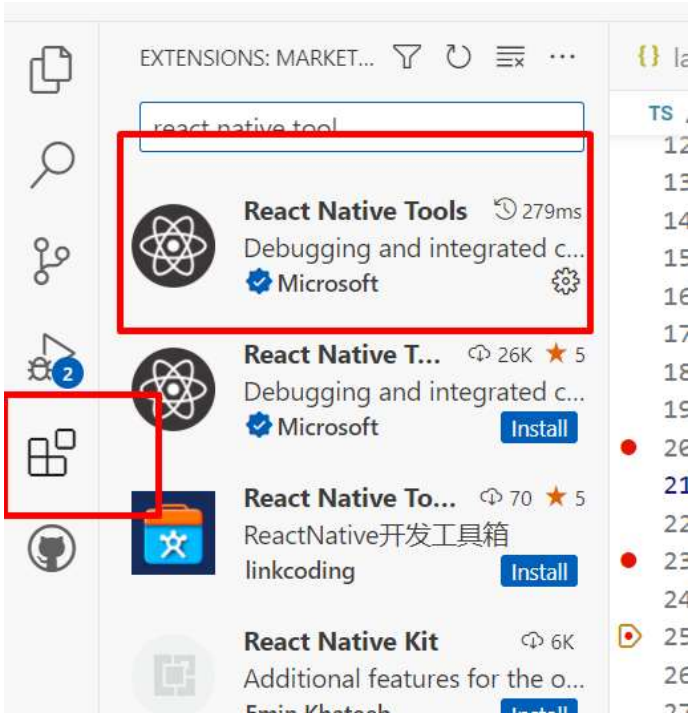
import vào 1 file khác

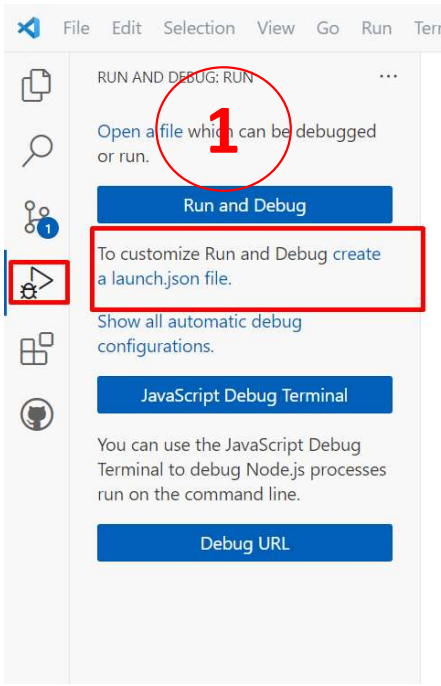
Not included in {}

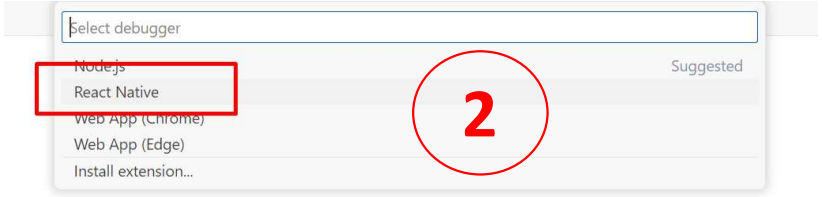
```
//main.js

import squareNumber from "functionsFile";
squareNumber(7);
```

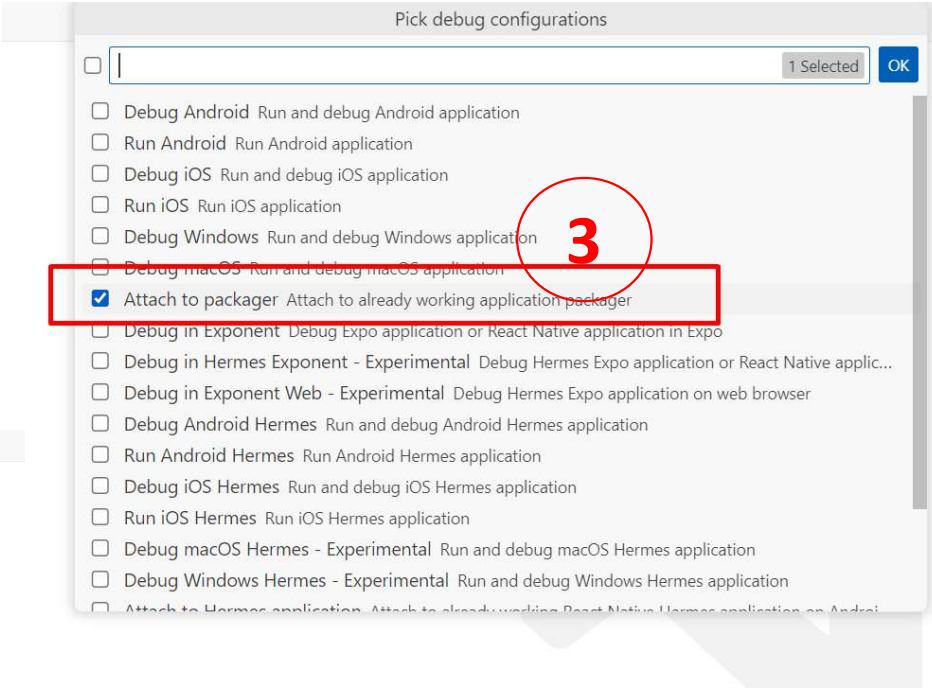
IV. Debugging

- Setup extension ‘React native tool’ on Visual studio Code




File Edit Selection View Go Run Terminal
RUN AND DEBUG: RUN
Open a file which can be debugged or run.
1
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL

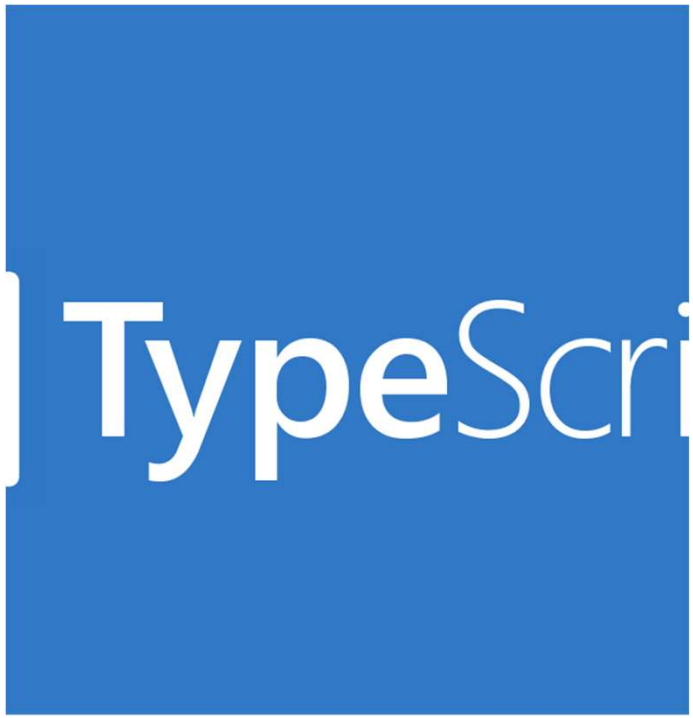

Select debugger
Node.js
Suggested
React Native
Web App (Chrome)
Web App (Edge)
Install extension...
2


Pick debug configurations
1 Selected
OK
Debug Android Run and debug Android application
Run Android Run Android application
Debug iOS Run and debug iOS application
Run iOS Run iOS application
Debug Windows Run and debug Windows application
3
Attach to packager Attach to already working application packager
Debug in Exponent Debug Expo application or React Native application in Expo
Debug in Hermes Exponent - Experimental Debug Hermes Expo application or React Native applic...
Debug in Exponent Web - Experimental Debug Hermes Expo application on web browser
Debug Android Hermes Run and debug Android Hermes application
Run Android Hermes Run Android Hermes application
Debug iOS Hermes Run and debug iOS Hermes application
Run iOS Hermes Run iOS Hermes application
Debug macOS Hermes - Experimental Run and debug macOS Hermes application
Debug Windows Hermes - Experimental Run and debug Windows Hermes application


EXPLORER
DEMOEXPOAPP
.expo
.vscode
.react
launch.json U
assets
node_modules
.gitignore
App.js M
app.json
babel.config.js
package-lock.json
package.json
launch.json U
.vscode > launch.json > ...
4
{
// Use IntelliSense to learn about possible attributes.
// Hover to view descriptions of existing attributes.
// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
"version": "0.2.0",
"configurations": [
{
"name": "Attach to packager",
"cwd": "${workspaceFolder}",
"type": "reactnative",
"request": "attach"
}
]
}

V. Typescript

- TypeScript is a statically typed superset of JavaScript that enhances code quality and developer productivity by adding strong typing to your code.



Why TypeScript in React Native?

- In React Native, TypeScript offers:
  - Better code quality
  - Enhanced developer tooling
  - Improved collaboration in teams
  - Reduced runtime errors
- TypeScript is used to:
  - Define component props and states
  - Annotate function parameters and return types
  - Type-check your code at compile-time

Basic Types

- number, string, boolean, array
- null, undefined, any, void“

- Example: let age: number = 30;
- let name: string = "John";
- let isCompleted: boolean = true;
- let numbers: number[] = [1, 2, 3, 4, 5];

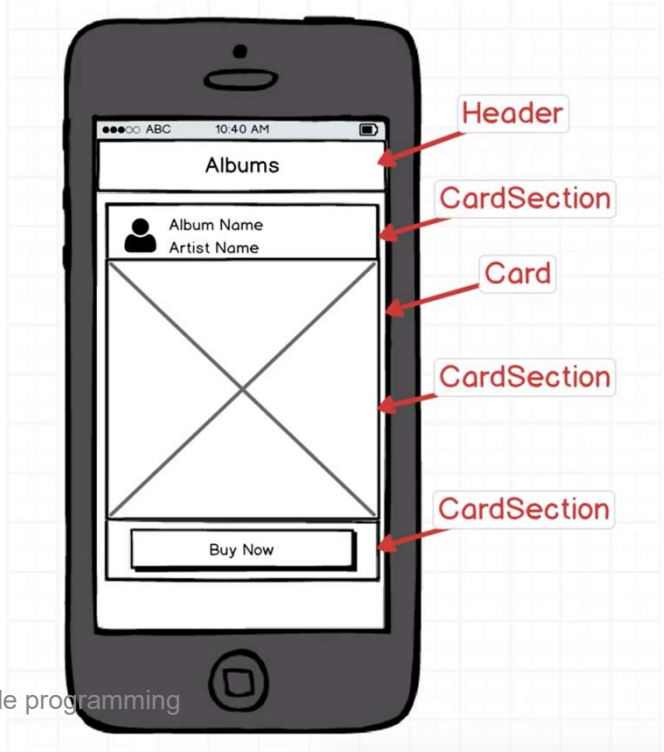
React Native Component, function with TypeScript

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

interface Props {
  name: string;
}

class Greeting extends Component<Props> {
  render() {
    return (
      <View>
        <Text>Hello, {this.props.name}!</Text>
      </View>
    );
  }
}
```

```tsx
function greet(name: string): string {
  return `Hello, ${name}!`;
}
```

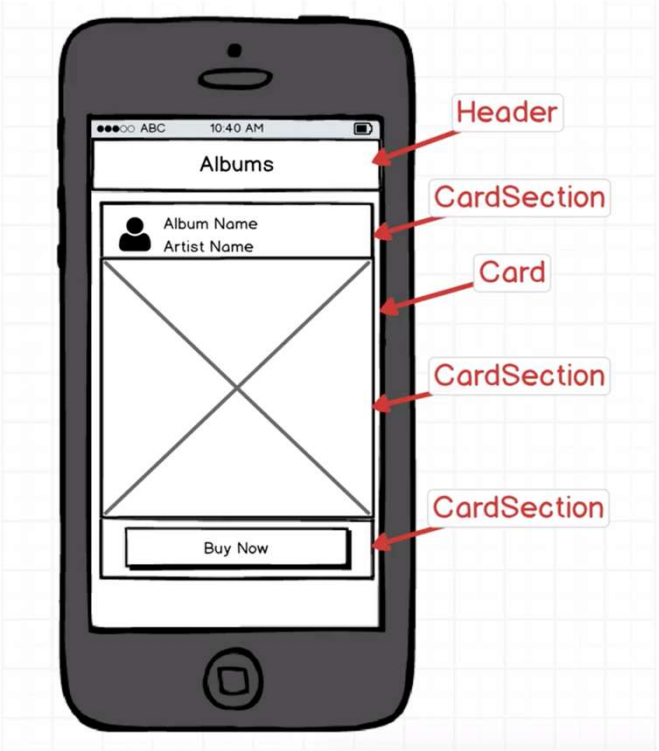
VI. Components

- Components are a fundamental concept of both React and React native. It is the breakdown of the application into small components that create their high reusability and scalability



Components

- In addition to the components we build and reuse, React native provides a lot of default components.
  - **Native Components**
  - **Custom Component**



Native Components

- React Native provides a set of core/native components that allow you to build mobile user interfaces using familiar concepts from web development.
- These components are part of the React Native framework and offer a bridge to native UI elements on both iOS and Android platforms
- Can use community-made Native Components: https://reactnative.directory/

IOS VIEW	WEB ANALOG	DESCRIPTION
`<UIView>`	A non-scrollling `<div>`	A container that suppo with flexbox, style, som handling, and accessib controls
`<UITextView>`	`<p>`	Displays, styles, and ne strings of text and eve touch events
`<UIImageView>`	`<img>`	Displays different type images
`<UIScrollView>`	`<div>`	A generic scrolling con that can contain multi components and views
`<UITextField>`	`<input type="text">`	Allows the user to ente

Native Components

- The following core component APIs Documents can be found: https://reactnative.dev/docs/components-and-apis

```
import { View, Text, Image, ScrollView, TextInput } from 'react

const App = () => {
  return (
    <ScrollView>
      <Text>Some text</Text>
      <View>
        <Text>Some more text</Text>
        <Image
          source={{
            uri: 'https://reactnative.dev/docs/assets/p_cat2.pn
          }}
          style={{ width: 200, height: 200 }}
        />
      </View>
      <TextInput
        style={{
          height: 40,
          borderColor: 'gray',
          borderWidth: 1
        }}
        defaultValue="You can type in me"
      />
    </ScrollView>
  );
```

Custom Component

- There are two main types of custom components in React Native:
  - Functional Components
  - Class Components

Custom Component

- Functional Components:

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

const HelloWorld = () => {
  return (
    <View>
      <Text>Hello, World!</Text>
    </View>
  );
};

export default HelloWorld;
```

```
import React from 'react';
import HelloWorld from './Helloworld'
export default App =()=>{

  return (
    <HelloWorld/>
  )
}
```

Custom Component

- Class Components

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

class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  handleIncrement = () => {
    this.setState((prevState) => ({
      count: prevState.count + 1,
    }));
  }

  render() {
    return (
      <View>
        <Text>Count: {this.state.count}</Text>
        <Button title="Increment" onPress={this.handleIncrement} />
      </View>
    );
  }
}
export default MyComponent;
```

```
import React from 'react';
import MyComponent from './MyComponent'
export default App =()=>{

  return (
    <MyComponent/>
  )
}
```

Props

- 1. Data in the React Component is managed by state and props.
- 2. The state can change while prop is immutable. This means that the state can update in the future while prop is impossible.
- 3. Props: allows changing the React Component through properties.

Multiple Props

```jsx
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;
```

Nhúng thuộc tính name của Cat vào JSX qua props, mặt đin tất cả component điều có props

Hello, I am Maru!
Hello, I am Jellylorum!
Hello, I am Spot!

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

const CatApp = () => {
  return (
    <View>
      <Image
        source={{uri: "https://reactnative.dev/docs/assets/p_cat1.png"}}
        style={{width: 200, height: 200}}
      />
      <Text>Hello, I am your cat!</Text>
    </View>
  );
}

export default CatApp;
```

Hello, I am your cat!

State

- Props are considered parameters passed to configure components to use and remain immutable.
- While State handles when data componets changes due to interaction from users
  - Use prop when rendering components.
  - Enable state to keep track of the component's data that wants to change over time.

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

const Counter = () => {
  const [count, setCount] = useState(0);

  const handleIncrement = () => {
    setCount(count + 1);
  };

  return (
    <View>
      <Text>Count {count}</Text>
      <Button title="Increment" onPress={handleIncrement} />
    </View>
  );
};
```

State & array

```
import { useState } from 'react';

let nextId = 0;

export default function List() {
  const [name, setName] = useState('');
  const [artists, setArtists] = useState([]);
```

```
  return (
    <>
      <h1>Inspiring sculptors:</h1>
      <input
        value={name}
        onChange={e => setName(e.target.value)}
      />
      <button onClick={() => {
        artists.push({
          id: nextId++,
          name: name,
        });
      }}>Add</button>
      <ul>
        {artists.map(artist => (
          <li key={artist.id}>{artist.name}</li>
        ))}
      </ul>
    </>
  );
}
```

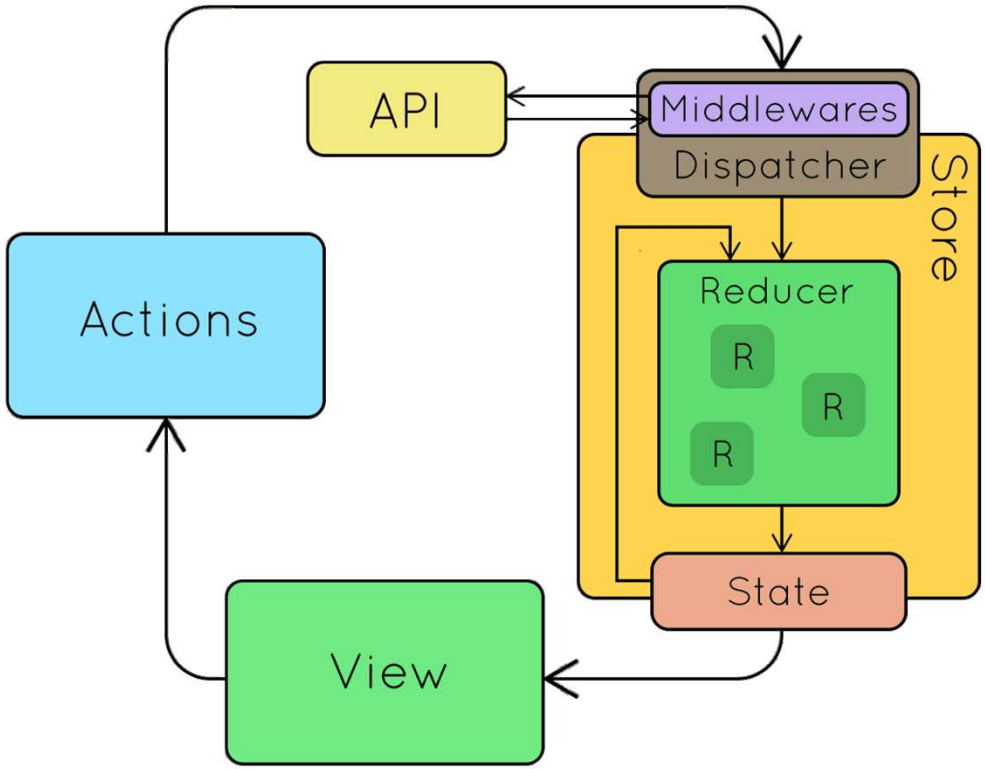
https://react.dev/learn/managing-state

State & array

- Example

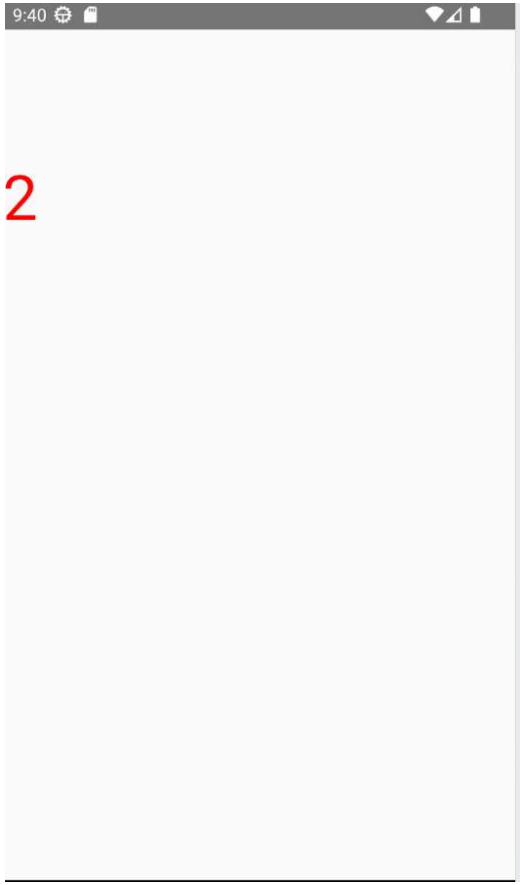
Redux

- Redux is a predictable state container for JavaScript apps. Redux has 4 main components:
- Actions
- Reducer
- Store
- View



Redux

```
1   import { createStore } from 'redux'
2   import { View, Text } from 'react-native';
3
4    // 1.
5   function counter(state = 0, action) {
6     switch (action.type) {
7       case 'INCREMENT':
8         return state + 1
9       case 'DECREMENT':
10        return state - 1
11      default:
12        return state
13    }
14  }
15  export default App = () => {
16  // 2.
17  let store = createStore(counter)
18
19  // 4.
20  store.dispatch({ type: 'INCREMENT' }) // in ra 1
21  store.dispatch({ type: 'INCREMENT' }) // in ra 2
22
23  return (
24    <View style = {{top:100}}><Text>{store.getState()}</Text></View>//3
25  );
26  }
```



https://snack.expo.dev/@taitv/demo-redux

Redux toolkit

- **useSelector:**
  - A part of the react-redux library.
  - Allowing you to access the state and perform component rendering based on that state

```
import { useSelector } from 'react-redux';
const myData = useSelector((state) => state.myReducer.myData);
```

- **useDispatch**
  - A part of the react-redux library
  - Allowing you to dispatch actions to modify the state stored in the Redux store

```
import { useDispatch } from 'react-redux';
const dispatch = useDispatch();
dispatch(increment());
```

Redux toolkit

- The **Redux Toolkit** package is intended to be the standard way to write Redux logic.
- Cài đặt: **npm install @reduxjs/toolkit**

- Configuring a Redux store is too complicated.
- I have to add a lot of packages to get Redux to do anything useful.
- Redux requires too much boilerplate code.

Redux toolkit

1. Create a Redux State Slice

```
import { createSlice, configureStore } from
'@reduxjs/toolkit’

const counterSlice = createSlice({
  name: 'counter',
  initialState: 1000,
  reducers: {
    increment: t => t += 1,
    decrement: d => d -= 1,
    increa10: e => e += 10,
  },

});
export const { increment, decrement, increa10 }
= counterSlice.actions
```

2. Create a Configure Store

```
const store = configureStore({
  reducer: {counter:counterSlice.reducer},

});
export default store;
```

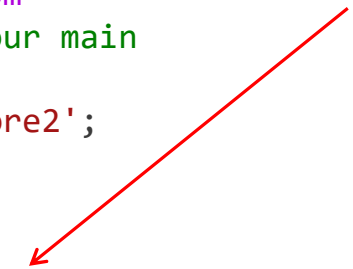
Redux toolkit

3. Create component

```
function MyComponent() {
  const counter = useSelector(state =>
state.counter);
  const dispatch = useDispatch();

  const increment2 = () => {
    dispatch{increment};
  };
  return (
        <View>
          <Text>Counter: {counter}</Text>
      <Button title='Increment'
onPress={increment2}></Button>
      </View>
  );
}
export default MyComponent;
```

4. Call the component into App.js

```
import React from 'react';
import { Provider } from 'react-
redux';
import RootComponent from
'./RootComponent'; // Your main
application component
import store from './store2';

function App() {
  return (
    <Provider store={store}>
    <RootComponent/>
    </Provider>
  );
}

export default App;
```



Publishing

- Step 1: On Windows keytool must be run from C:\Program Files\Java\jdkx.x.x_x\bin, as administrator. Open powershell.

Run keytool -genkeypair -v -storetype PKCS12 -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000

- Step 2: Setting up Gradle variables

Place the my-upload-key.keystore file under the android/app directory in your project folder

Edit the file ~/.gradle/gradle.properties or android/gradle.properties, and add the following (replace ***** with the correct keystore password, alias and key password)

MYAPP_UPLOAD_STORE_FILE=my-upload-key.keystore
MYAPP_UPLOAD_KEY_ALIAS=my-key-alias
*MYAPP_UPLOAD_STORE_PASSWORD=*****

*MYAPP_UPLOAD_KEY_PASSWORD=*****

Publishing

- Step 3: Edit the file android/app/build.gradle in your project folder, and add the signing config

```
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {
                storeFile file(MYAPP_UPLOAD_STORE_FILE)
                storePassword MYAPP_UPLOAD_STORE_PASSWORD
                keyAlias MYAPP_UPLOAD_KEY_ALIAS
                keyPassword MYAPP_UPLOAD_KEY_PASSWORD
            }
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
```

Publishing

- Step 4.1 : Generating the release AAB

  npx react-native build-android --mode=release

  Optional: divided by CPU type

```
android {
    splits {
        abi {
          reset()
           enable true
          universalApk false
           include "armeabi-v7a", "arm64-v8a", "x86", "x86_64"
        }
    }
}
```

- Step 4.2: Build React native App to apk file

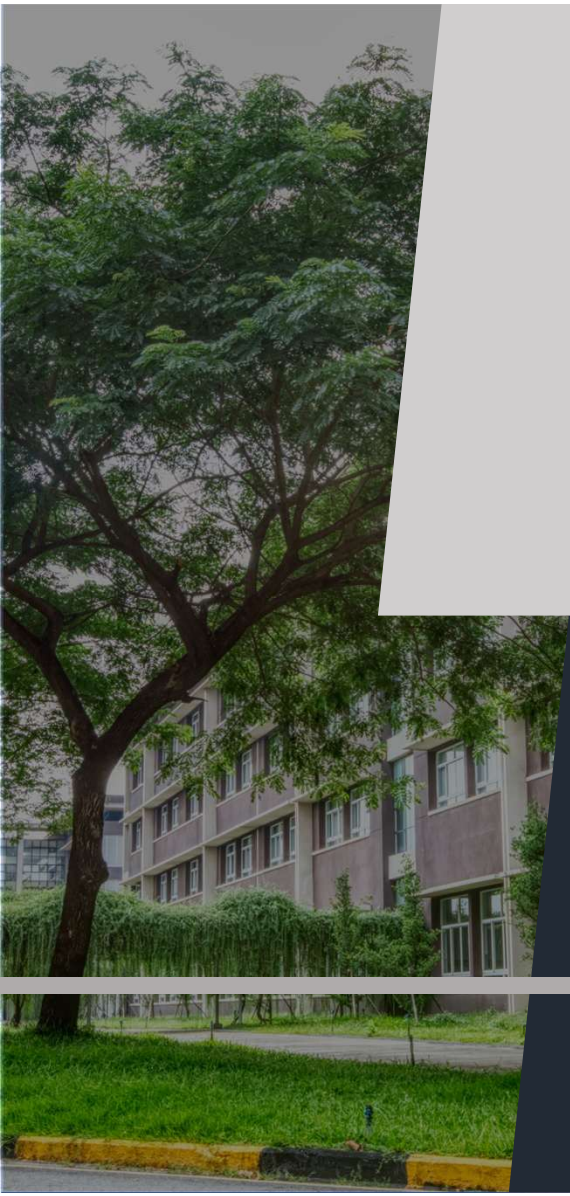
  ./gradlew assembleRelease

Exercises

1. Install the environment and create your first project with Expo CLI and React Native CLI

2. Create a component that solves quadratic equations.

3. Enter the numerator and denominator of the fraction. Conveniently shorten and print to the screen.



EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Q&A