Lecture 02

Introduction to React Native Programming

2.0 Introduction

React Native represents the modern paradigm shift from **imperative programming** toward **component-based software architecture** and **declarative programming**.
Instead of describing *how* the system works, developers define *what* the UI should look like.

React Native extends the core principles of **React** into the mobile ecosystem, enabling **dynamic UI behavior** that responds automatically to data changes.

Key Theoretical Foundations

React Native integrates multiple disciplines of software design:

- **Software Engineering:** Modular, maintainable systems
- **Object-oriented Design:** Encapsulation and responsibility separation
- **Functional Paradigm:** UI as a pure function of data

These principles converge to form a system where logic, data, and presentation coexist harmoniously.

2.1 Environment Setup Overview

1. Install Node.js (LTS)

- Download from https://nodejs.org
- Choose the LTS version (e.g., 18 or 20)
- Verify installation:

```
node -v
npm -v
```

1. Use the Local Expo CLI

Global `expo-cli` is deprecated

- Remove legacy global CLI (if installed):

```
npm uninstall -g expo-cli
```

2. Use the Local Expo CLI

- Install expo

```
npm install expo-cli
```

- Use Local CLI via npx:

```
npx expo <command>
```

3. Create a New Expo Project

- Run:

```
npx create-expo-app myApp --template
```

- select ">Blank (minimal ....)

4. If you want to run on web

Install this package

```
npx expo install react-dom react-native-web
```

5. Run the Project

Start Metro Bundler:

```
npx expo start
```

- A QR Code will appear for mobile preview

6. Install Expo Go on Mobile

- **Android**
- Download from Google Play Store
- **iOS**
- Download from App Store

Open Expo Go → Scan the QR Code to launch the app

7. Set Up Android Emulator (Optional)

1. Install Android Studio
2. Open **SDK Manager** → Install Android SDK
3. Open **AVD Manager** → Create a Virtual Device
4. Start the Emulator → Run:

```
npx expo start
```

Select "Run on Android device/emulator"

8. Validate the Environment

Use:

```
npx expo-doctor
```

To ensure all dependencies and configurations are correct

Essential Tools Summary

- Node.js LTS
- npm
- Local Expo CLI (npx expo)
- Expo Go
- Android Studio (optional)

2.2 React Component: The Building Block of UI

Definition and Role

A **Component** is a reusable, self-contained unit that defines both the structure and logic of a UI segment.
It can be compared to a “conceptual module” — a small, independent part of a larger digital system.

In React:

> **UI = f(Data)**
>
> The UI is a function of data.

Example: A Basic Component

```
function Welcome() {
  return (
    <Text>Welcome to our App!</Text>
  );
}
```

Analysis:

- A **Stateless Functional Component**
- Always returns the same output for the same input
- Demonstrates *functional purity* and *predictable behavior*

Component Composition

```
function App() {
  return (
    <View>
      <Welcome />
      <Text>Let’s get started!</Text>
    </View>
  );
}
```

Composition allows smaller components to combine hierarchically —
a key concept in **Component-based Architecture** and **Separation of Concerns**.

2.2.3 Key Characteristics of Components

1. **Encapsulation** – Self-contained logic and data
2. **Reusability** – Designed for reuse with different inputs
3. **Composition** – Components can form complex UI hierarchies
4. **Maintainability** – Easy to update without side effects

Example: Encapsulation

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <View>
      <Text>Count: {count}</Text>
      <Button title="Add" onPress={() => setCount(count + 1)} />
    </View>
  );
}
```

Encapsulation hides the internal logic (`useState`)

→ external users don’t need to know how it works, only how to use it.

Example: Reusability

```
function Welcome(props) {
  return <Text>Hello {props.name}</Text>;
}

<View>
  <Welcome name="Student" />
  <Welcome name="Instructor" />
</View>
```

Encourages **Parametric Polymorphism** – behavior changes according to input.

2.3 JSX Syntax: The Language of UI Logic

Concept

JSX (JavaScript XML) allows developers to describe UI structure within JavaScript — blending logic and visual structure into one declarative syntax.

It reflects **Declarative Programming**:

> “Tell the system *what to show*, not *how to draw it*.”

Structure of JSX

```
function Greeting() {
  const user = "Somchai";
  return (
    <View>
      <Text>Hello {user}</Text>
    </View>
  );
}
```

Theory:

- `<View>` and `<Text>` map to native components (UIView, ViewGroup)
- `{}` embeds JavaScript expressions
- JSX is compiled by **Babel** into a **Virtual UI Tree**

Benefits of JSX

Concept	Description
Readability	Code resembles actual UI layout
Logic Integration	Logic and structure coexist seamlessly
Declarative Thinking	Focus on results, not procedural steps

Declarative Conditional Example

```
<View>
  {isLoggedIn ? <Text>Welcome Back!</Text> : <Text>Please Sign In</Text>}
</View>
```

React automatically re-renders the UI based on the `isLoggedIn` state — no manual DOM manipulation required.

2.4 Props & State: Data Mechanisms in React

Props (Properties)

Props are *inputs* from parent components that configure child behavior.

They establish **unidirectional data flow** — data moves **top-down** only.

```
function Welcome(props) {
  return <Text>Hello {props.name}</Text>;
}

function App() {
  return <Welcome name="Student" />;
}
```

Theory:

- `App` → parent, passes `name`
- `Welcome` → child, displays based on props
- Reflects **Functional Programming:** same input → same output
- Promotes **Immutability** – children cannot modify received data

State

State represents the internal, changeable data of a component.

When it changes, React **re-renders** automatically.

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <View>
      <Text>Count: {count}</Text>
      <Button title="Add" onPress={() => setCount(count + 1)} />
    </View>
  );
}
```

Concept:

State drives UI.

This is the essence of **Reactive Programming** — data controls the interface.

Props vs State

Aspect	Props	State
Source	External	Internal
Changeable	Immutable	Mutable
Purpose	Configure component	Manage dynamic data
UI Update	Does not trigger render	Triggers re-render

2.5 Working with Expo

What is Expo?

Expo is a **runtime environment** that simplifies running and testing React Native apps — no need for complex Android/iOS SDK setup.

In theory, Expo acts as an **Abstraction Layer**
between **JavaScript logic** and **native operating systems**.

Expo Workflow

Step	Conceptual Meaning
1. Create project	Initialize basic structure via Expo CLI
2. Run on Expo Go	Simulate app on real device instantly
3. JS ↔️ Native Bridge	Expo translates JavaScript commands to native functions
4. Build App	Generate `.apk` or `.ipa` automatically

Project Structure

```
/MyApp
├── App.js
├── package.json
├── assets/
└── components/
```

- **App.js** – Entry point / Root Component
- **package.json** – Metadata and dependencies
- **assets/** – Images, sounds, media
- **components/** – Custom UI elements

Reflects **Modular Design** and **Scalability** principles.

Example: Running with Expo

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

export default function App() {
  return (
    <View>
      <Text>Hello React Native!</Text>
    </View>
  );
}
```

Expo compiles and runs this instantly on **Expo Go**, bridging JavaScript logic and native APIs.

2.6 Summary of Core Concepts

React Native embodies the harmony between theory and practical implementation.

Theoretical Concept	Manifestation in Code
Component-based Architecture	UI divided into manageable units
Declarative Programming	JSX describes desired UI
State-driven Rendering	Automatic UI updates with data changes
Unidirectional Data Flow	Props flow top-down
Abstraction Layer	Expo bridges JS and native systems

Key Takeaways

- React Native = theory *made tangible*
- Merges **Software Architecture** and **Reactive Design**
- Demonstrates how principles like **Abstraction**, **Encapsulation**, and **Data Flow** create real, maintainable software systems

> “React Native transforms software theory into interactive reality.”

End of Lecture

Thank you!