Course Title: Android Programming (React Native II)

Course Code: BIT 6294

Target Audience: (300 Level Lincoln University (NSUK) Campus Students)

Credit: 4

Topic: State Management

Lecturer: Mr. Vincent

## Main Goal:

By the end of Week 7 – 8, students should be able to:

- Understand what state is and why it is needed
- Use useState to manage local components state
- Handle user input and forms in react native
- Share data globally using the Context API
- Understand the concept of Redux Toolkit
- Build functional Login and Signup forms using state

# WEEK 7: LOCAL AND GLOBAL STATE

## 1. Introduction to State Management

**State** represents data that changes over time in your application. In React Native, state determines what users see on their screens and how the app responds to user interactions.

## Types of State

- **Local State:** Data that belongs to a single component (form input, toggle switches)
- **Shared State:** Date needed by multiple components (user authentication status)
- **Global State:** Date accessible throughout the entire app (theme, user profile)

## Why State Management Matters

- Keeps UI in sync with data
- Enables computer communication
- Makes apps interactive and dynamic
- Improves code organization and maintainability

## 1. Local State with useState

Local state refers to data that is only relevant to a single components or a small, isolated part of the application. The useState Hook is the fundamental way to manage this data in functional components.

### 1. The useState Hook

- **Import:** you must import it from React: import React, {useState} from “react”;
- **Usage:** it returns an array with two elements:
  1. The current state value.
  2. A state setter function (used to update the state).

### 2. Syntax and example

```
import React, { useState } from "react";
import { Button, StyleSheet, Text } from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";

export default function Index() {
  //[currentState, stateSetter] = useState(initialState)
  const [count, setCount] = useState(0);
  return (
    <SafeAreaView style={styles.conatiner}>
      <Text style={styles.label}>Count: {count}</Text>
      <Button title="Increment" onPress={() => setCount(count + 1)} />
      <Button title="Decrement" onPress={() => setCount(count - 1)} />
      <Button title="Reset" onPress={() => setCount(0)} />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  conatiner: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    padding: 20,
  },
  label: {
    fontSize: 40,
    fontWeight: "bold",
    marginBottom: 20,
  },
});

```

**Exercise 1: Toggle Switch**

Create a React Native components with a toggle switch that controls notifications. Display text showing “Notifications: ON” Notifications: OFF” base on the switch state.

```
import { useState } from "react";
import { StyleSheet, Switch, Text } from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";

export default function Index() {
  //[currentState, stateSetter] = useState(initialState)
  const [isEnabled, setIsEnabled] = useState(false);
  return (
    <SafeAreaView style={styles.conatiner}>
      <Text style={styles.label}>Notification: {isEnabled ? "ON" : "OFF"}</Text>
      {/* <Button title="Toggler" onPress={() => setIsEnabled(!toggleSwitch)} />
*/}
      <Switch
        value={isEnabled}
        onValueChange={setIsEnabled}
        trackColor={{ false: "#767577", true: "#81b0ff" }}
        thumbColor={isEnabled ? "#f5dd4b" : "#f4f3f4"}
      />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  conatiner: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    padding: 20,
  },
  label: {
    fontSize: 40,
    fontWeight: "bold",
    marginBottom: 20,
  },
});
```

## Example 3: Managing Objects

```
import { useState } from "react";
import { StyleSheet, Text, TextInput } from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";

export default function Index() {
  //[currentState, stateSetter] = useState(initialState)
  const [user, setUser] = useState({ name: "", email: "", age: "" });

  const updateField = (field, value) => {
    setUser((prevUser) => ({
      ...prevUser,
      [field]: value,
    }));
  };

  return (
    <SafeAreaView style={styles.conatiner}>
      <TextInput
        style={styles.label}
        placeholder="Name"
        value={user.name}
        onChangeText={(text) => updateField("name", text)}
      />
      <TextInput
        style={styles.label}
        placeholder="Email"
        value={user.email}
        onChangeText={(text) => updateField("email", text)}
        keyboardType="email-address"
      />
      <TextInput
        style={styles.label}
        placeholder="Age"
        value={user.age}
        onChangeText={(text) => updateField("age", text)}
        keyboardType="numeric"
      />

      <Text style={styles.title}>
        Name: {user.name}
        {"\n"}
        Email:{user.email} {"\n"}
        Age:{user.age}
      </Text>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  conatiner: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    padding: 20,
  },
  label: {
    height: 40,
    borderWidth: 1,
    borderColor: "#ccc",
    width: "100%",
    paddingHorizontal: 10,
    marginBottom: 15,
    borderRadius: 5,
  },
  title: {
    fontSize: 16,
    lineHeight: "16",
    marginTop: 20,
  },
});
```

## Week 8: GLOBAL STATE MANAGEMENT

Global state is data that needs to be accessed by many different components throughout your application, regardless of their location in the component tree.

### 2. Context API