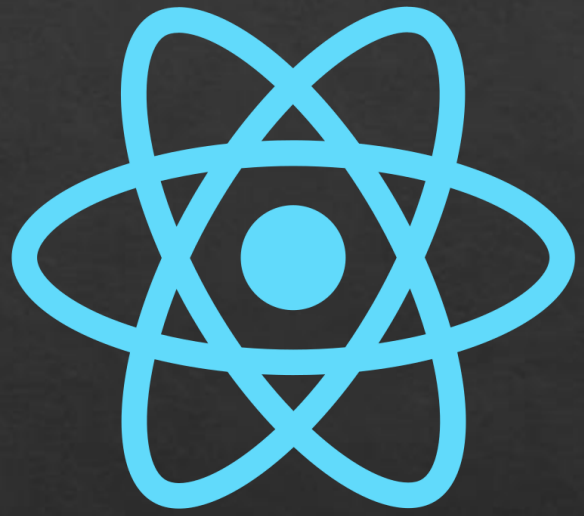
React Native

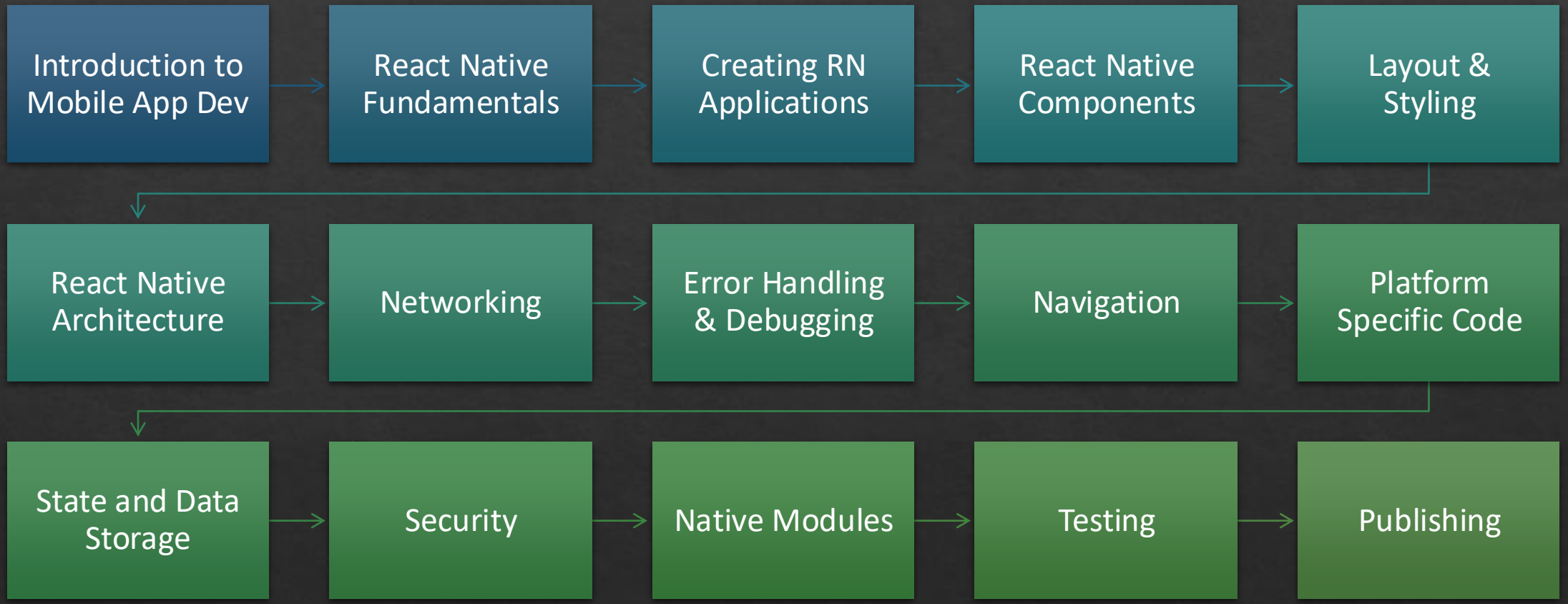


ANIL JOSEPH

WA: 9833169784

Email: anil.jos@gmail.com

Agenda



Presentaed by Anil Joseph(anil.jos@gmail.com)

Introduction

Anil Joseph

- 20+ years of experience in the industry
- Technologies
  - C, C++
  - Java, Enterprise Java
  - .Net, .Net Core
  - JavaScript, Node.js
  - React & Angular
  - Mobile: React Native, Native Android, Xamarin, MAUI
  - GenAI and Agentic AI
- Worked on numerous projects
- Conducted trainings for Corporates

Software

Node.js & NPM

Visual Studio Code

Android Studio
&
Emulator

Xcode & Simulator

Browsers(Chromium)

Prerequisites

JavaScript

TypeScript

React

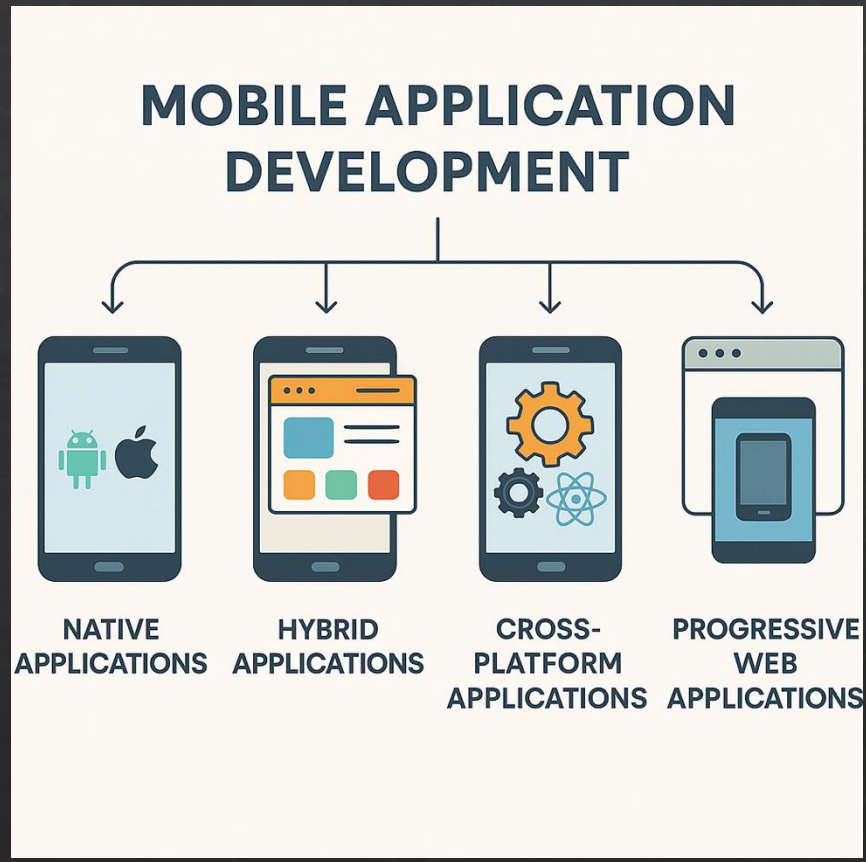
React Native

Learn Once, write anywhere

Presentaed by Anil Joseph(anil.jos@gmail.com)

Mobile Application Development

- Native Applications
- Hybrid Applications
- Cross-Platform Applications
- Progressive Web Application



Presentaed by Anil Joseph(anil.jos@gmail.com)

Native Applications

- Built using platform-specific languages like iOS: Swift / Objective-C,Android: Kotlin / Java
- Best performance, full access to device hardware
- Usually the most expensive and time-consuming to build
- Ideal for high-end apps: gaming, AR/VR, camera-heavy apps

Presentaed by Anil Joseph(anil.jos@gmail.com)

Hybrid Applications

- Built using web technologies (HTML, CSS, JavaScript) and wrapped in a native shell
- Frameworks: Ionic, Cordova, Capacitor
- Faster to build than pure native
- Performance is decent, but not as smooth as fully native apps
- Good for simple to medium-complexity apps

Cross-Platform Applications

- One codebase → runs on both iOS and Android
- Frameworks: React Native, Flutter, Xamarin, .NET MAUI
- Almost-native performance
- Saves time and cost while still giving access to device APIs
- Great for most business apps, dashboards, banking apps, etc.

Presentaed by Anil Joseph(anil.jos@gmail.com)

Progressive Web Apps (PWAs)

- Web apps that feel like mobile apps
- Installed from the browser, not the App Store
- Very fast to develop and deploy
- Limited access to deeper device hardware
- Good for content-driven apps, e-commerce, forms, internal tools

React Native

- A JavaScript framework that lets you build mobile apps for iOS and Android using a single codebase.
- Uses native components under the hood, so apps feel fast and smooth.
- Built and maintained by Meta (Facebook) with a huge open-source community.
- Built on top of React.
- Based on the principle of “**Learn Once, Write Anywhere”**

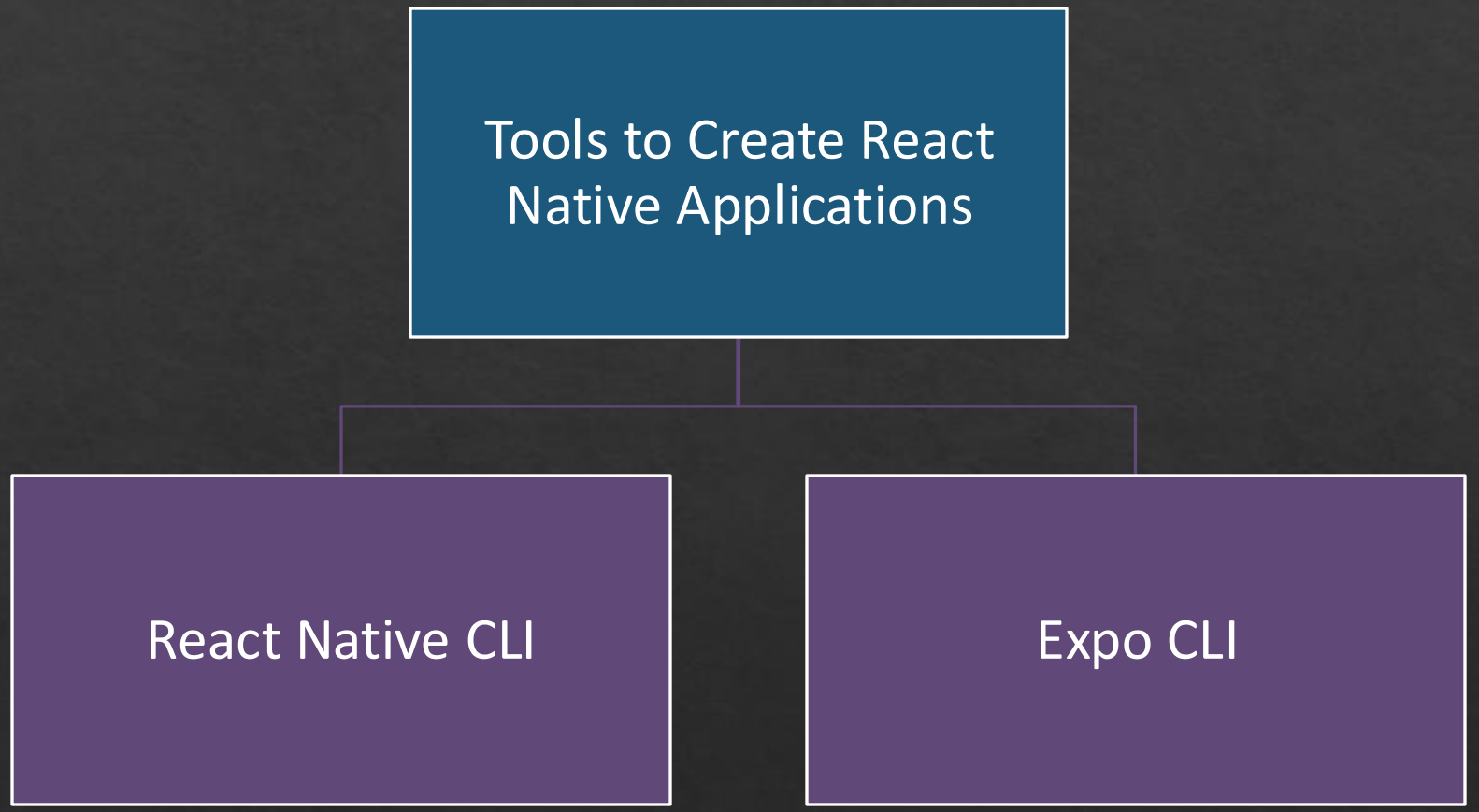
React Native vs. Native Apps

Factor	React Native	Native (iOS + Android)
Codebase	Single codebase for both platforms	Separate apps for iOS & Android
Development Speed	Faster — shared UI & logic	Slower — everything built twice
Performance	Very good, near-native	Best-in-class performance
Access to Device APIs	Good, sometimes requires native modules	Full access out of the box
Developer Skills	JavaScript + React	Swift/Objective-C (iOS), Kotlin/Java (Android)
Ecosystem & Libraries	Large community + plugins, sometimes unstable	Official, stable, platform-specific
Maintenance Cost	Lower	Higher
Best For	Business apps, dashboards, fintech apps, content apps	High-end games, AR/VR, animation-heavy apps

History

- Born inside Facebook (Meta) — originally created for their internal mobile apps.
- First public release: 2015, introduced at Facebook's F8 conference.
- Quickly gained popularity because it allowed JavaScript developers to build real native apps.
- 2016–2018: Huge ecosystem growth — community libraries, UI kits, navigation systems.
- 2020 onwards: Focus shifted to major improvements like New Architecture, TurboModules, Fabric Renderer, and better performance.
- Today (2025), RN is one of the most widely used cross-platform frameworks across startups and enterprises.

Getting Started



React Native Community CLI

- Command line tools that help you build apps with react-native.
- Replaces the old “react-native-cli”.

Getting Started: React Native CLI

1

Install NodeJs

Install Android Studio

Install XCode

2

Create React Native Application

- **npx @react-native-community/cli@latest init AwesomeProject**

3

Start the Application

- **cd AwesomeProject**
- **npm start**
- **npm run android**
- **npm run ios**

React Native Framework

- React Native allows developers to use the **React programming paradigm** to ship applications to native targets.
- The React Native provides the **core APIs** and **functionalities** for application development
- Creating production ready applications require more tools and libraries like
  - Tools for publishing the app to a store
  - Libraries for Navigation, Icon etc.
- A React Native Framework provides the tools and libraries.
- Expo is a Open Source React Native Framework.

Expo

- Expo is a **framework and a set of tools** that sits on top of React Native to make mobile development easier.
- It handles a lot of the **native setup** for you — so you can focus on writing JavaScript/TypeScript.
- Expo SDK, a powerful SDK: camera, sensors, notifications, secure storage, location, permissions… all without writing native code.
- Works on all platforms: iOS, Android, and Web (with Expo Web).
- You can run your app instantly on your phone using **Expo Go**, without compiling native code.

Getting Started: Expo CLI

1

Install NodeJs

Install Android Studio(optional)

Install Xcode(optional)

2

Create React Native Application

- **npx create-expo-app@latest**
- **Use –template option to choose a template**

3

Start the Application

- cd app-name
- npm start
- npm start

Expo Builds

- Managed Workflows
  - Write JavaScript/TypeScript → Expo takes care of all the native stuff.
  - No need of XCode or Android Studio(unless you want to use the Emulator or Simulator)
- Development Builds
  - More powerful and more flexible.
  - Can include Native modules
  - Integrate deeper native libraries (e.g., biometrics, encryption, payment SDKs)
- Start with a managed workflow and move to a development build

Expo Application

- app.json
  - The main configuration file for an Expo project.
  - Controls how your app looks, behaves, and builds on iOS, Android, and the web.
  - Expo reads this file every time you run or build the app.
  - Configure :
    - App name & icon, Splash scree, App version, Permissions etc.
- index.js
  - Start script of the application.
  - Registers the root component to the AppRegistry.
- AppRegistry
  - React Native’s entry point that tells the native OS which component should start the app.

React Native Application Code – Big Picture

- A React Native app is still **just a React app**
  - Components, props, state, hooks – all the same ideas
- The main difference is:
  - UI is not rendered to the **DOM** (,)
  - It is rendered to **native UI components** (<View>, <Text>, etc.)
- Styling is done via **JavaScript objects**, not CSS files
- Navigation, gestures, animations, platform APIs are all handled by libraries

A Simple React Native Screen (Expo App)

```jsx
import React from "react";
import { View, Text, Button, StyleSheet } from "react-native";

export default function App() {
  const [count, setCount] = React.useState(0);

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to React Native 🚀</Text>
      <Text>You clicked {count} times</Text>
      <Button title="Click me" onPress={() => setCount(count + 1)} />
    </View>
  );
}

const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" },
  title: { fontSize: 22, fontWeight: "600", marginBottom: 10 },
});
```

React (Web) vs React Native – Mental Mapping

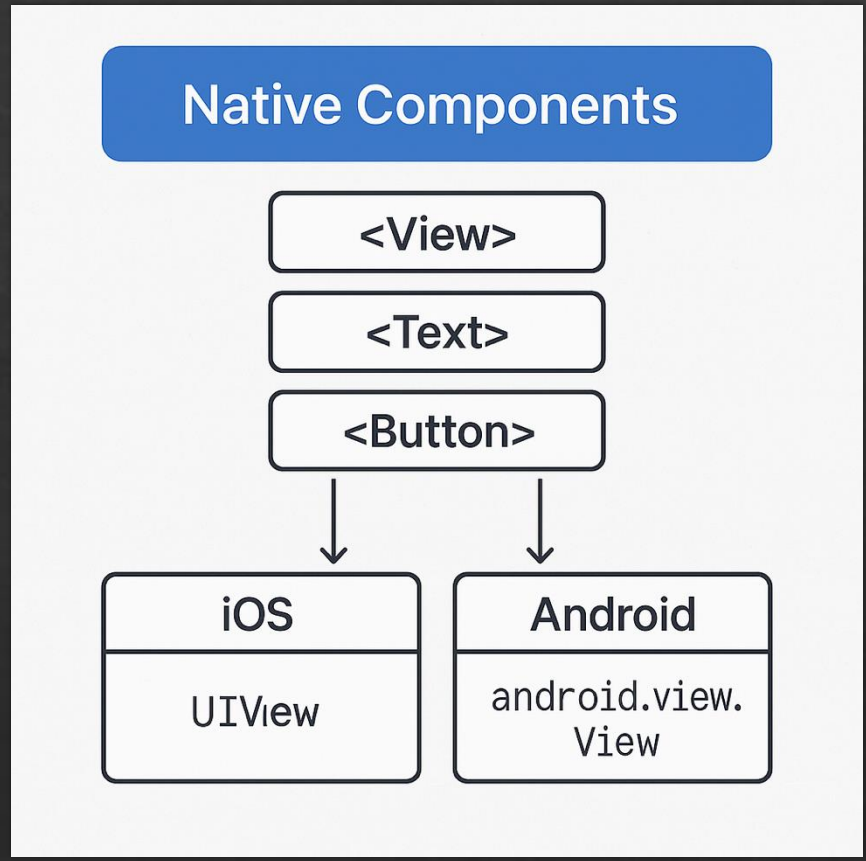
- **JSX Elements**
  - Web:
    - , , <p>, <img>, <button>
  - React Native:
    - <View>, <Text>, <Image>, <Pressable>, <TouchableOpacity>, <ScrollView>
- **Styling**
  - Web:
    - CSS files, CSS Modules, styled-components, Tailwind, className
  - React Native:
    - StyleSheet.create({ ... }) or inline JS objects
    - No CSS selectors, no id, no .class

React (Web) vs React Native – Mental Mapping

- **Platform**
  - Web:: Renders to DOM in the browser
  - React Native: Renders to **native views** via the RN bridge / new architecture
    - Your JS runs in a JS engine, UI is 100% native
- Hooks / State / Props
  - Same concepts as React web:
    - useState, useEffect, useContext, useMemo, useCallback, custom hooks
  - React developers can basically bring over their state management patterns unchanged:
    - Redux, Zustand, MobX, React Query, etc.

Native Components

- In Android, views are written in Kotlin/Java; in iOS, they're written in Swift/Objective-C.
- With React Native, you use JavaScript + React components to invoke those native views.
- At runtime, React Native creates the actual Android/iOS views behind the scenes.
- Since RN components use the same underlying native UI elements, apps look, feel, and perform like truly native apps.
- These platform-backed elements are called Native Components.



Core Components

- React Native provides a set of built-in, ready-to-use Native called Core Components.
- React Native has many Core Components for everything from controls to activity indicators.

Core Components

Basic Components

- View, Text, TextInput, ScrollView, Image, Pressable, etc

User Interface

- Button, Switch

List View

- FlatList, SectionList

Android

- BackHandler, ToastAndroid etc.

iOS

- ActionSheetIOS

Others

- Alert, Dimensions etc.

View

View is the most fundamental component for building a UI

View is a container

View supports layout with Flexbox

Supports Touch Events

Its maps directly to the native equivalent: UIView or android.view

Flexbox

- A layout system that helps you arrange elements on the screen
- Controls how items align, space themselves, and wrap
- Default layout mechanism in React Native.
- Flexbox
  - flex – how much space an element should take
  - flexDirection – row or column layout
  - justifyContent – vertical alignment (in a column)
  - alignItems – horizontal alignment
  - flexWrap – wrap items to next line

Navigation

- Just like on the web, apps need to move between screens
- React Native uses navigation libraries to manage screens, routes, stacks, tabs, and drawers
- The most popular library is React Navigation
- Expo provides Expo Navigation based on React Navigation

React Navigation: Native Stack Navigation

- Stack Navigator provides a way for your app to transition between screens where each new screen is placed on top of a stack.
- This navigator uses the native APIs UINavigationController on iOS and Fragment on Android
- Supports a Static and Dynamic Configuration

React Navigation: Bottom Tabs

- A simple tab bar on the bottom of the screen that lets you switch between different routes.
- Routes are lazily initialized -- their screen components are not mounted until they are first focused.

Vector Icons

- Icons created using **mathematical paths**, not pixels
- Libraries like **react-native-vector-icons**, **Expo Vector Icons provide these icons**
- **Scalable, Lightweight, Consistent Look**
- **Large Icon set**
- **Easy to use**

ScrollView

- A container that lets content scroll
- Renders all its children at once
- Best for small, static, or short content
- Simple and easy to use

Touchable & Pressable

- Components used to make things touchable / clickable in mobile apps.
- They can wrap any element (text, image, view) and give it tap, press, and feedback behavior.
- Touchable
  - TouchableOpacity → reduces opacity on press
  - TouchableHighlight → adds a highlight background
  - TouchableWithoutFeedback → no visual feedback
  - TouchableNativeFeedback (Android only)
- Pressable

Pressable

- Most flexible touch component
- Full control over:
  - onPress, onLongPress, onPressIn, onPressOut, disabled states
- Style can change based on press state

Virtualized Lists

- Lists that **render only the items currently visible on the screen.**
- Instead of drawing *all* items at once, they load/unload items as you scroll
- Makes long lists fast + memory-efficient
- Provided by FlatList, SectionList, and VirtualizedList
- Advantages
  - High performance, Low memory usage, Easy to Use

FlatList: Optimizations

- Stable keyExtractor (no index keys)
- Memoized row components (React.memo)
- useCallback for renderItem & callbacks
- getItemLayout for fixed-height rows
- Tune initialNumToRender, maxToRenderPerBatch, windowSize
- Use pagination with onEndReached
- Avoid nested virtualized lists

SectionList

- A **virtualized list** like FlatList, but with **grouped sections**
- Each section has:A header (e.g. “A”, “B”, “Fruits”, “Veggies”)An array of data itemsPerfect for grouped / categorized lists

Navigation Patterns

- Stack Navigation
  - Screens on top of each other (like browser history)
  - Perfect for “go to details” flows
- Tab Navigation
  - Bottom or Top tabs to switch between major sections
  - Ideal for social apps, shopping apps, dashboards
- Drawer Navigation
  - Slide-in menu from left side
  - Good for apps with many sections or settings

Networking

- Applications need to communicate to the backend services.
- React Native supports HTTP and WebSocket protocols based communications
- HTTP calls
  - React Native's JavaScript supports both fetch API's and the XMLHttpRequest object.
  - XMLHttpRequest based API like axios, firsbee etc. can be used.
- WebSockets
  - React Native also supports WebSockets, a protocol which provides full-duplex communication channels over a single TCP connection.

axios

- Promise based HTTP client for the browser and node.js
- Installation
  - npm install axios
- Features
  - Make http requests
  - Supports the Promise API
  - Intercept request and response
  - Transform request and response data
  - Cancel requests
  - Automatic transforms for JSON data
  - Client side support for protecting against XSRF

axios methods

axios.request({config})

axios.get(url, {config})

axios.post(url,{data}, {config})

axios.delete(url, {config})

axios.put(url,{data}, {config})

axios global defaults

Base URL

- axios.defaults.baseURL = 'https://abc.com’;

Headers

- axios.defaults.headers.common['Authentication'] = AUTH_TOKEN;

Headers for specific methods

- axios.defaults.headers.post['Content-Type'] = 'application/json';

Navigations

- Navigation is provided by a community solution.
- React Navigation is a standalone library that allows developers to set up the screens of an app with navigation.
- Supports various types of navigation
  - Standard Stack Navigation
  - Drawer Navigation
  - Tab Navigation
- https://reactnavigation.org/

List Views

- React Native provides a suite of components for presenting lists of data.
- The two main components are
  - FlatList
  - SectionList

FlatList

- A performant interface for rendering basic, flat lists, supporting the most handy features
- Fully cross-platform.
- Optional horizontal mode.
- Configurable viewability callbacks.
- Header support.
- Footer support.
- Separator support.
- Pull to Refresh.
- Scroll loading.
- ScrollToIndex support.
- Multiple column support.

Section List

- A performant interface for rendering sectioned lists, supporting the most handy features
- Fully cross-platform.
- Configurable viewability callbacks.
- List header support.
- List footer support.
- Item separator support.
- Section header support.
- Section separator support.
- Heterogeneous data and item rendering support.
- Pull to Refresh.
- Scroll loading.

Component API's

- Alert
  - Launches the Alert dialog with the specified title, message and actions.
- Dimensions
  - Gets the dimensions(height & width) of the screen, window and notifies changes to the dimensions.
- Linking
  - Linking provides an interface to interact with incoming and outgoing messages.
- Keyboard
  - Keyboard module to control keyboard events.

Community Packages

- AsyncStorage
- NetInfo
- Geolocation
- Icons
- Camera
- Share

Platform Independent Code

- React Natives Provides 2 ways to write platform-independent code
  - Using the Platform Module
  - Using Platform specific file extensions

Platform Module

- Platform Module provides API's to detect the platform in which the app is running.
- Use the detection logic to write platform specific code.
- Platform.OS can be used to detect the platform
  - will be ***ios*** when running on iOS and ***android*** when running on Android.
- Platform.select is a method to get platform specific values
- Platform.Version can be used to detect the version of the android or ios platform

Platform-specific extensions

- To implement platform specific code you can write your code in different files for android and ios
  - Example: DefualtButton.ios.js & DefualtButton.android.js
- React Native will detect when a file has a .ios. or .android. extension and load the relevant platform file when required from other components.
- Platform specific extension can be used to start Android and iOS applications with different startup code(example with index.ios.js and index.android.js)

Data Storage Options

- In Memory
  - Redux and React Context
- Persistent
  - Async Storage, SQL-lite Database
- Secure Storage
  - iOS - Keychain Services
  - Android - Secure Shared Preferences
- File System
  - Sandboxed locations: Documents and cache
- Database Services
  - Firebase

Async Storage

- The most common storage for small amounts of data
- Works like localStorage in web
- Stores strings only(Key-Value pairs)
- Good for settings, flags, tokens (non-sensitive), preferences

Secure Storage

- A secure, encrypted key–value store used to save **sensitive data** on the device.
- Use the Expo SecureStore library
- Uses iOS Keychain
- Uses Android Keystore + EncryptedSharedPreferences
- Store sensitive information like: JWT Access/Refresh Token, passwords, API Keys,

Secure Storage Advantages

- iOS (Keychain)
  - AES-256 hardware-backed encryption
  - Keys stored in Secure Enclave (when available)
  - Protected by device passcode/FaceID/TouchID
- Android (Keystore)
  - AES-256-GCM encryption
  - Keys generated & stored in Trusted Execution Environment
  - Data stored in EncryptedSharedPreferences

SQL Lite

- SQLite is a **lightweight, embedded relational database** stored directly on the device.
- It supports SQL queries, tables, indexes, joins
- Libraires
  - **expo-sqlite** (Expo)
  - **react-native-sqlite-storage**
- Ideal when your app needs **structured, persistent, queryable data**.
- Use-cases
  - Offline-first applications
  - Local caching

File System

- React Native doesn’t give you direct access to the OS file system.
- Instead, you use a library:
  - Expo apps → expo-file-system
  - RN CLI apps → react-native-fs (or also expo-file-system if using Expo modules)
- Provides access the app sandboxed folders
  - documentDirectory – persistent, for your app’s files
  - cacheDirectory – temporary, OS/app may clear it
- Store PDF’s, Images etc.

React Native Architecture

- React Native has evolved over time and the architecture has changed for better performance.
- The new architecture was introduced in 2018 and by 2022(RN version 0.68) was available as on opt-in feature. By 2024(RN version 0.73) it’s the default architecture
- Features of the New Architecture
  - JSI(JavaScript Interface)
  - Fabric Renderer
  - TurboModules
  - Unified C++ core
  - Codegen (Automatic Native Code Generation)
  - Improved Multithreading

JSI (JavaScript Interface)

- JSI is a **lightweight, high-performance C++ interface** that lets JavaScript call native code **directly**, without using the old asynchronous Bridge.
- A new low-level connector that replaces the Bridge, enabling React Native's New Architecture to be fast, modern, and concurrent.
- Enables **synchronous**, low-latency communication between JS and Native.
- No JSON Serialization, event queues, bridge batching.
- Faster, Lower memory,

Fabric

- A modern, concurrent, and performant UI rendering system.
- Key Features
  - Concurrent React support → non-blocking UI updates.
  - Improved layout and diffing, using C++ core.
  - Better memory management.
  - Synchronous rendering where needed (critical UI updates).
  - Partial tree updates instead of re-rendering the entire tree.
  - Better support for large lists.

Shadow Tree

- The **Shadow Tree** is React Native Fabric's **in-memory representation** of your UI, similar in concept to the Virtual DOM.
- It holds **Shadow Nodes** for every component and is responsible for **layout, diffing, and computing native view mutations** before committing them to the screen.

Shadow Tree

Old Architecture

No Shadow Tree—UIManager processed JSON command batches from JS

Updates sent through async **Bridge**, serialized as JSON

Layout partly JS-driven, partly native

Whole view updates often required

Dependent on JS thread responsiveness

New Architecture (Fabric)

Maintains a Shadow Tree and computes minimal native mutations

No Bridge, no JSON—**direct C++ calls** via Fabric

Full layout in C++, always consistent

Partial tree updates, only changed nodes mutated

Many UI updates happen **independent of JS**

Yoga

- Yoga is a **cross-platform layout engine**, written in **C++**, that implements **Flexbox** to calculate size and position for all React Native components.
- It ensures that layout behaves the same way on iOS, Android, and Web, regardless of platform differences.
- Features
  - Implements Flexbox
  - Cross-platform + Deterministic
  - Used by Fabric (and used to be used by UIManager)
  - Fast and Lightweight(C++)

Threads in React Native

- JS Thread
  - Runs JavaScript (React logic, state updates, effects, reconciler).
- Native Main Thread (UI Thread)
  - Renders the actual UI (UIKit on iOS, Views on Android).
- Fabric “Shadow Tree / Layout” Thread (C++ Core)
  - Hosts the Shadow Tree (virtual UI tree).
  - Runs Yoga layout in C++.
  - Performs diffing, committing, and generating native mutations.
- JSI Runtime Thread(s)
  - Handles low-level JS <-> C++ bindings.

Concurrent React support

- Fabric fully supports Concurrent React, allowing rendering work to be paused, resumed, or reprioritized without freezing the UI.
- This keeps interactions smooth even during expensive state updates or long renders.

Synchronous Rendering (in Fabric)

- n the New Architecture, certain UI updates—especially those involving **direct user interaction**—can be applied **immediately on the native thread**, without waiting for the JS thread, React reconciliation, or asynchronous scheduling.

JSI: Code

```
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <Button
      title={`Count: ${count}`}
      onPress={() => setCount(c => c + 1)}
    />
  );
}
```

Old Architecture

🔁 1. User taps the Button (Native → JS)

1. User taps the native button.
2. Native layer (Android `View`, iOS `UIView`) sends a **touch event**.
3. Event is **enqueued** in a **native → JS event queue**.
4. Bridge picks up the event and sends it to the **JS thread** asynchronously.

> If the **JS thread is busy**, the event waits in the queue → tap feels delayed.

Old Architecture

🧠 2. JS Handles Event, Updates State

5. JS thread receives the event and runs `onPress`.
6. `setCount(c => c + 1)` schedules a **state update**.
7. React reconciler runs:
    - Builds a new React tree
    - Diffs it with the previous one
    - Figures out: `title` prop on `<Button>` changed from `Count: 0` → `Count: 1`.

All of this happens on the **single JS thread**.

Old Architecture

📦 3. JS → Native UI Updates via Bridge (Serialization + Batching)

8. React Native creates a set of **UI update commands**, e.g.:

   > "Update view #42: set `title = "Count: 1"`"

9. These commands are:
   - **Serialized to JSON** (or similar format)
   - **Batched** together
   - Pushed onto the **Bridge's JS → Native message queue**
10. On the native side, the Bridge:
    - Reads the batch from the queue
    - **Deserializes** JSON back into native commands
    - Passes them to **UIManager**
    - UIManager mutates the native views (changes the button label).

New Architecture

🔁 1. User taps the Button (Native → JS via JSI)

1. User taps the native button.
2. Native view triggers an event.
3. Event is delivered to JS via **JSI**, not the Bridge:
    - No JSON
    - No separate "Bridge thread"
    - Just a call into the **JS runtime** (with scheduling/priorities).

> Still asynchronous in practice (scheduled work), but with **far less overhead** and **more control** (priorities, concurrent rendering).

New Architecture

🧠 2. JS Handles Event, Updates State (Concurrent React)

4. JS runtime runs `onPress`.
5. `setCount(c => c + 1)` schedules a **state update**, potentially as:
   - **Urgent update** (e.g., click)
   - or **Transition** (via `startTransition`) as low-priority
6. React reconciler (Concurrent React) runs:
   - Builds the next tree
   - Diffs it with the previous tree
   - Decides the Button's `title` changed.

New Architecture

🌳 3. Fabric Commit: Shadow Tree → Native Views (No Serialization)

7. Instead of building a JSON command list for UIManager, Fabric:
   - Updates the **Shadow Tree** (C++ in-memory structure of the UI).
   - Runs **layout** using Yoga in C++.
   - Computes the **minimal set of mutations** needed for native views.
8. On commit, Fabric sends **structured mutations** directly to the native UI layer:
   - No JSON serialization
   - No batched string payload over a Bridge
   - No separate Bridge queue object → it's **C++ code calling native APIs**.
9. Native UI is updated: button text becomes `"Count: 1"`.

Turbo Modules

- **Next-generation native modules** in React Native that replace the old “Native Modules” system.
- They use **JSI (JavaScript Interface)** for direct JS ↔ native calls, removing the Bridge and enabling faster, type-safe, lazy-loaded native functionality.
- Direct Calls via JSI (No Bridge)
- Lazy Loading
- Codegen for Type Safety
- Multithreaded + High Performance
- Can be developed in multiple languages: **Swift, Kotlin, Java, Objective-C, or C++**.

Expo Modules

- Expo Modules are a **modern, developer-friendly system** for creating native functionality in React Native
- Developed using **Swift, Kotlin, and TypeScript**, without dealing with complex C++ or low-level JSI bindings.
- They provide a clean API layer on top of iOS and Android native code while remaining fully compatible with the **React Native New Architecture**.

Expo Modules Advantages

- Add native features quickly with minimal boilerplate
- Access OS APIs (sensors, hardware features, device info)
- Avoid writing C++ or JSI code manually
- Maintain compatibility with Expo toolchains

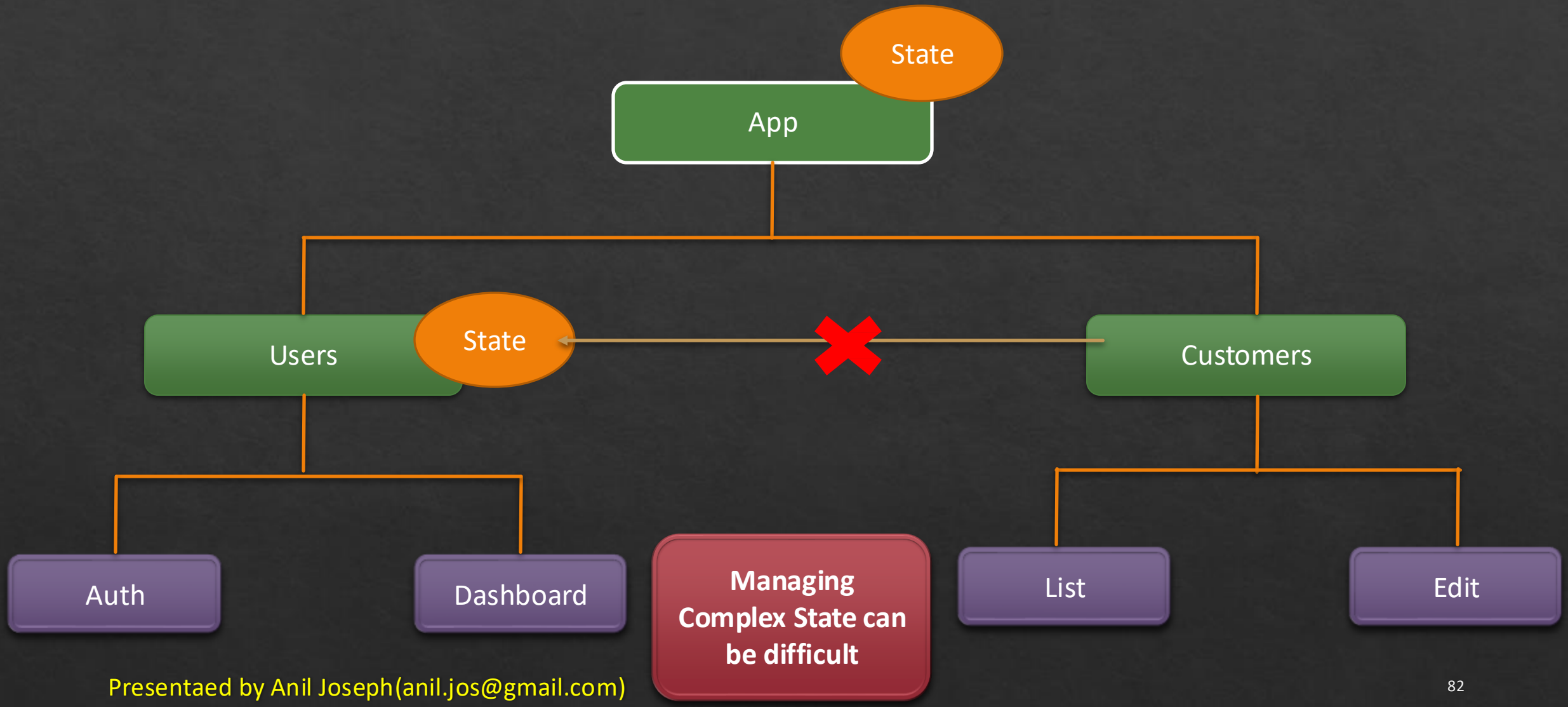
Redux

Redux is an open-source JavaScript library designed for managing application state.

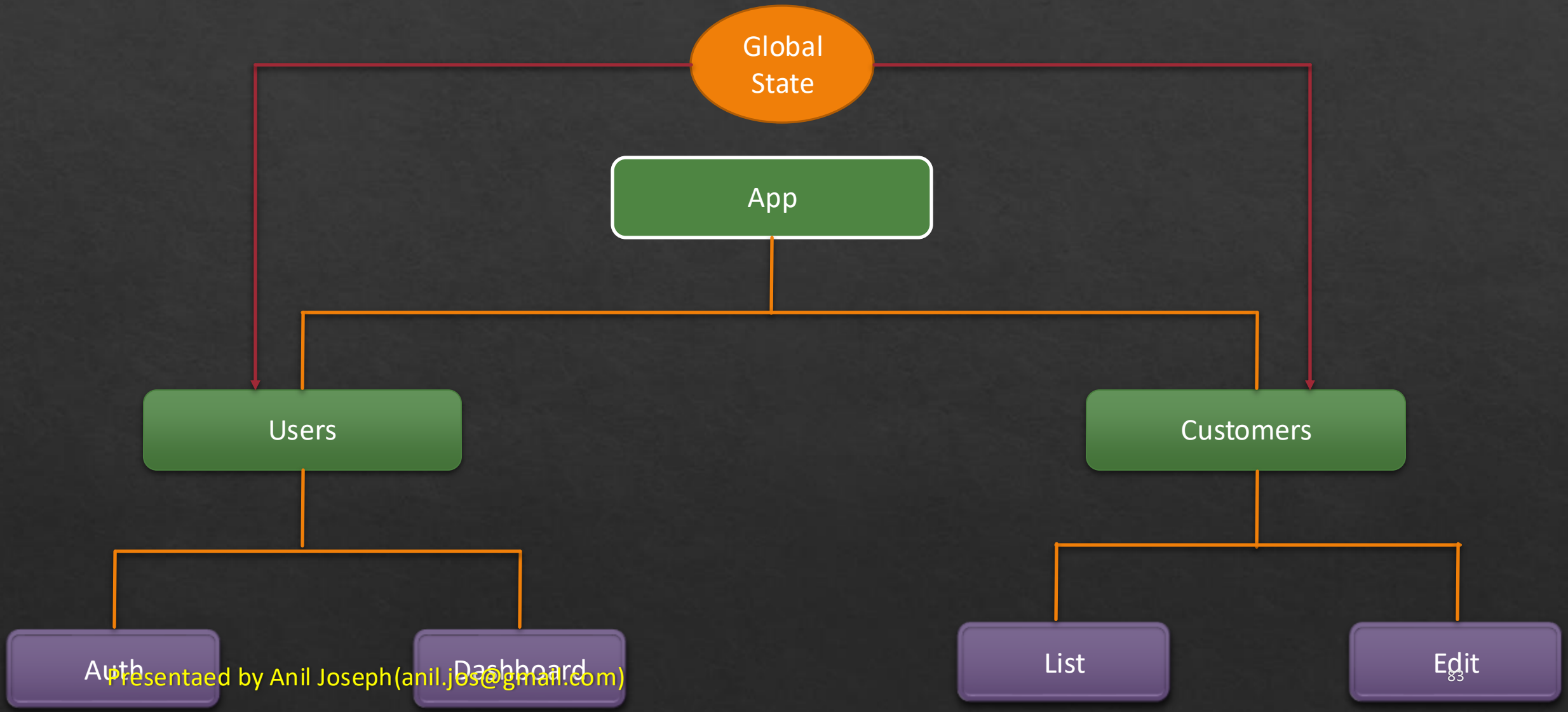
It is primarily used together with React or Angular for building user interfaces.

Redux was built on top of functional programming concepts.

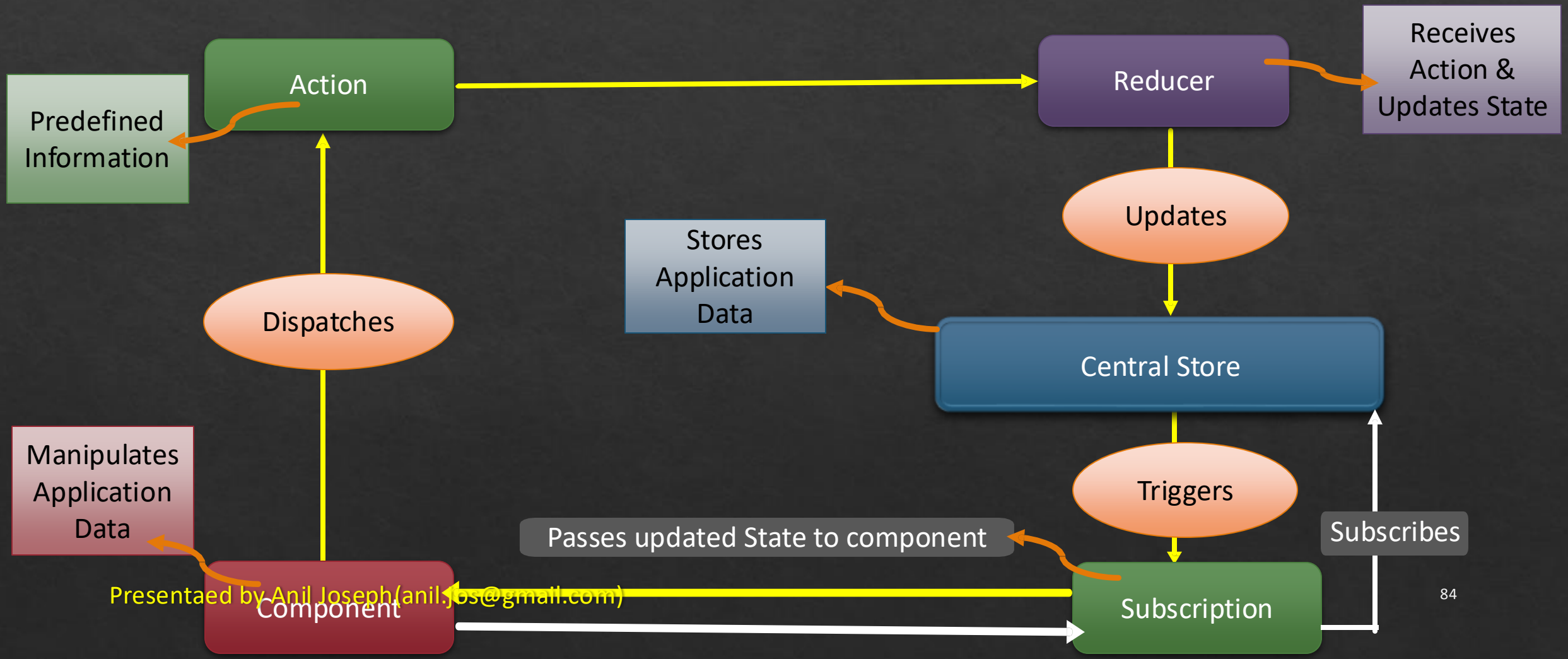
Why Redux?


State
App
Users
State
Customers
Auth
Dashboard
Managing Complex State can be difficult
List
Edit

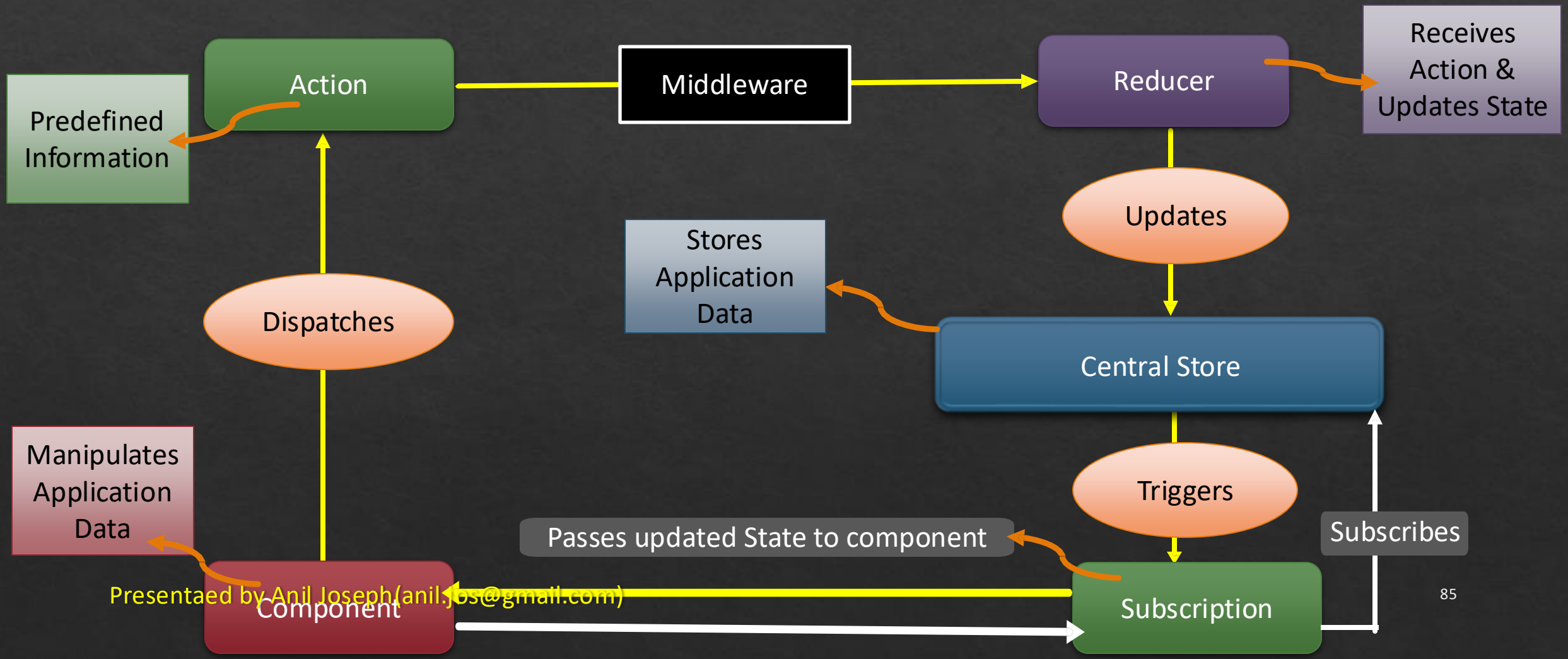
Why Redux?



Redux Flow

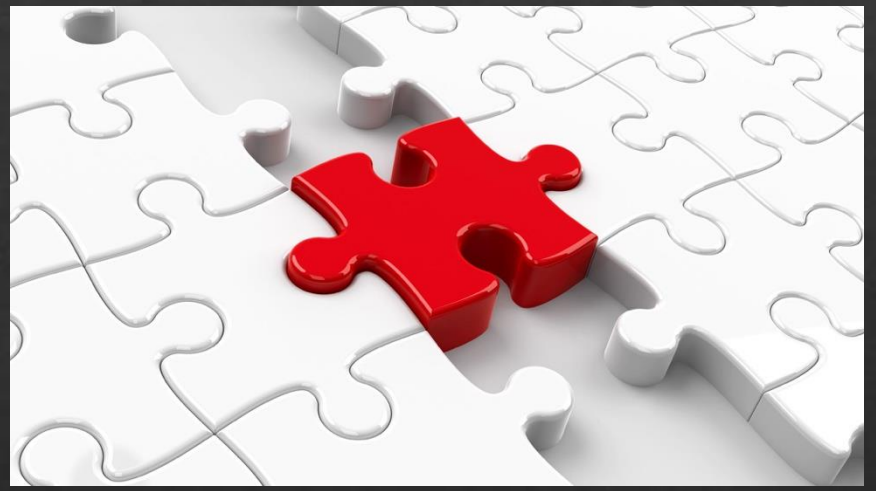

Action
Predefined Information
Reducer
Receives Action & Updates State
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes updated State to component
Subscribes
Component
Subscription

Redux Flow


Action
Middleware
Reducer
Receives Action & Updates State
Predefined Information
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes updated State to component
Subscribes
Component
Subscription

React Redux

- Redux react is a library that integrates Redux to a React Application
- Comprises of Components & Functions
- Installation
  - npm install react-redux



Redux Toolkit

- An opinionated toolset for efficient Redux development.
- Includes utilities to simplify common use cases like store setup, creating reducers, immutable update logic, and more.
- Provides good defaults for store setup out of the box, and includes the most commonly used Redux addons built-in.
- Takes inspiration from libraries like Immer and Autodux to let you write "mutative" immutable update logic, and even create entire "slices" of state automatically.
- Lets you focus on the core logic your app needs, so you can do more work with less code.

Animations

- React Native provides two animation systems
  - Animated: for granular and interactive control of specific values
  - LayoutAnimation: for animated global layout transactions.
- Animated
  - Component Types: View, Text, Image, and ScrollView
  - Custom Animated Components: Animated.createAnimatedComponent()
  - Configuring:
    - Animated.decay(): starts with an initial velocity and gradually slows to a stop.
    - Animated.spring(): provides a simple spring physics model.
    - Animated.timing(): animates a value over time using easing functions.
  - Native Driver: Using the native driver, we send everything about the animation to native before starting the animation.
    - You can use the native driver by specifying useNativeDriver: true in your animation configuration

Publishing Applications

- Android application are published to the PlayStore using the Play Store Console
- IOS app are published to the AppStore using the App Store Connect.
- React Native apps can be published in **two main ways**, depending on whether you're using **Expo** or a **Bare (CLI) app**.

App Build Formats

- APK — Android Package
  - Traditional Android app install file
  - Used for local installation and testing
- AAB — Android App Bundle
  - Modern publishing format required by **Google Play Store**
  - Contains all app resources; Play Store generates optimized APKs
  - Reduces app size for end-users
- IPA — iOS App Store Package
  - The packaged binary for iOS apps
  - Can only be installed on real devices if signed properly

Expo publish

- Expo provides a cloud-based build + publish service called **EAS (Expo Application Services)**.
- No need for Xcode or Android Studio
- Expo builds the APK/AAB/IPA on the cloud
- Easiest publishing method for React Native
- Advantages
  - No native setup needed
  - Handles signing, certificates, device configs
  - Reliable + beginner friendly
  - OTA updates supported (Expo Updates)

Publishing a Bare React Native App (CLI)

- Apps created using react native community cli or Expo apps that have been prebuild(**ejected)**.
- Build apps using **Android Studio** (Gradle) and **Xcode**
- More control over native code

Publish Configuration

- Permissions
  - Android: AndroidManifest.xml
  - iOS: info.plist
- App Name and Identifier
  - Set a visible AppName, App Version and unique App identifier
- Icons and Splash Screens

Unit Testing

- Jest (Default Test Runner)
  - Comes preconfigured with React Native
  - Fast, supports mocks, snapshots, timers, spies
  - Great for testing JS logic, components, and utils
- Testing Library for React Native(@testing-library/react-native)
  - Testing library built on top of React Testing Library
  - Helps test UI **the way a user interacts**
  - Focuses on: Text Visibility, Button Press(Touch), Input changes

End-to-end (E2E)

- End-to-end (E2E) tests verify your app **on a real device or simulator**
- **Tools**
  - Detox: Built specifically for **React Native**
  - Appium (Cross-Platform Automation)
  - Maestro (New, Lightweight E2E Tool)
  - BrowserStack: Cloud testing on **hundreds of real devices**

Micro-Frontends

- An architecture where **different parts of the UI are built and deployed independently**
- Each team owns a **feature module** (e.g., Payments, Profile, Chat)

Micro-Frontends Implementations

- Module-based Bundles
  - Each feature is compiled into a **separate JS bundle**
  - Loaded into the main app via:
    - Metro + custom bundler config
    - Re.Pack (Webpack-based bundling)
- Navigation-based Isolation
  - Each micro app exposes:
    - A **root screen**
    - A **navigation stack**
  - The shell app registers and mounts them dynamically

Micro Applications

- Self-contained **mini apps** inside a larger React Native app
- Each micro app has:
  - Its **own screens & navigation**
  - Its **own business logic**
  - Its **own state management**
  - Its **own module boundaries**
- Its multiple **sub-apps** inside a super-app (Amazon, Paytm).

Thank You

ANIL JOSEPH

anil.jos@gmail.com