React Native

ANIL JOSEPH

Agenda

- Introduction to React Native
- Creating React Native Applications
- Understanding React Native Components
- Debugging
- Forms and User Inputs
- Animations
- React Redux
- Navigation
- Networking and WebSockets
- Component API's
- Native Modules
- Testing
- Publishing

Software

- Node.js and NPM
- JavaScript Editor(Any one)
  - **Visual Studio Code**
- React-Native CLI
  - npm install –g react-native-cli
- Android Studio
  - Requires JDK 1.8
- XCode(For Mac)
- Chrome

React Native

Framework to build ***native mobile applications*** using JavaScript and React.

Based on the Principle:

Learn Once, Write Anywhere

Mobile Applications

Native Mobile Applications

- Built with Native Android(Java) or iOS API(Swift or Objective C)
- Fast and Responsive
- Wider Functionality(Camera, Microphone, Gestures etc)
- Match UI/UX to platform conventions
- Multiple Code bases

Mobile Applications

Mobile Web Applications

- Built using JavaScript, HTML & CSS
- Loaded through the browser
- Responsive Web Applications
- Progressive
- One code base
- No Native Feel(UI/UX)

Mobile Applications

Hybrid Applications

- Installed like a Native App, however is a Web Application
- One Code base(Mostly)
- Access device features
- Easier to scale to different platforms
- Performance does not match Native Apps
- User Experience is different
- Tools
  - PhoneGap, Cordova, Ionic

Mobile Applications

Cross-Platform Applications

- Developed in an intermediate language like JavaScript
- User Interface is rendered using Native Controls
- Better UI/UX
- Access device features
- Multiple Code bases but ***code can be shared***
- Performance matches that of Native Applications
- Tools
  - **React Native**, Xamarin, NativeScript

React Native

Framework to build ***native mobile applications*** using JavaScript and React.

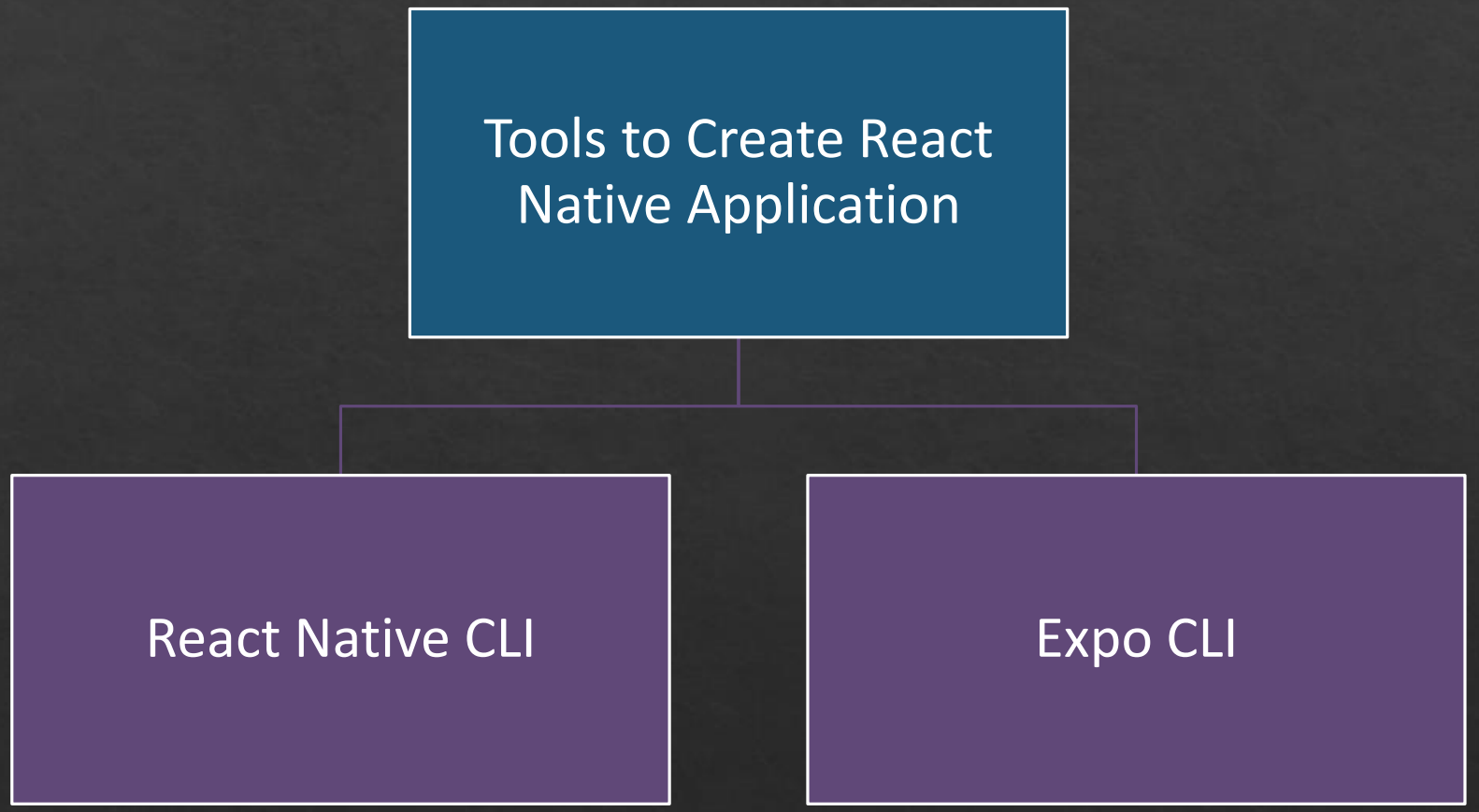
React Native uses the same fundamental UI building blocks as regular iOS and Android applications.

Better developer experience with reloading an Hot reloading

Use components written in Swift and Java with React Native

React Native was announced by Facebook in 2015

Getting Started



Getting Started-React Native CLI

1

Install NodeJs

- Runtime to run/execute JavaScript on the machine/server
- This installation also install npm(Node Package Manager)

2

Install "**React-Native CLI**"

- **Tool to create react native applications**
- **npm install -g react-native-cli**

3

Create Application/Project

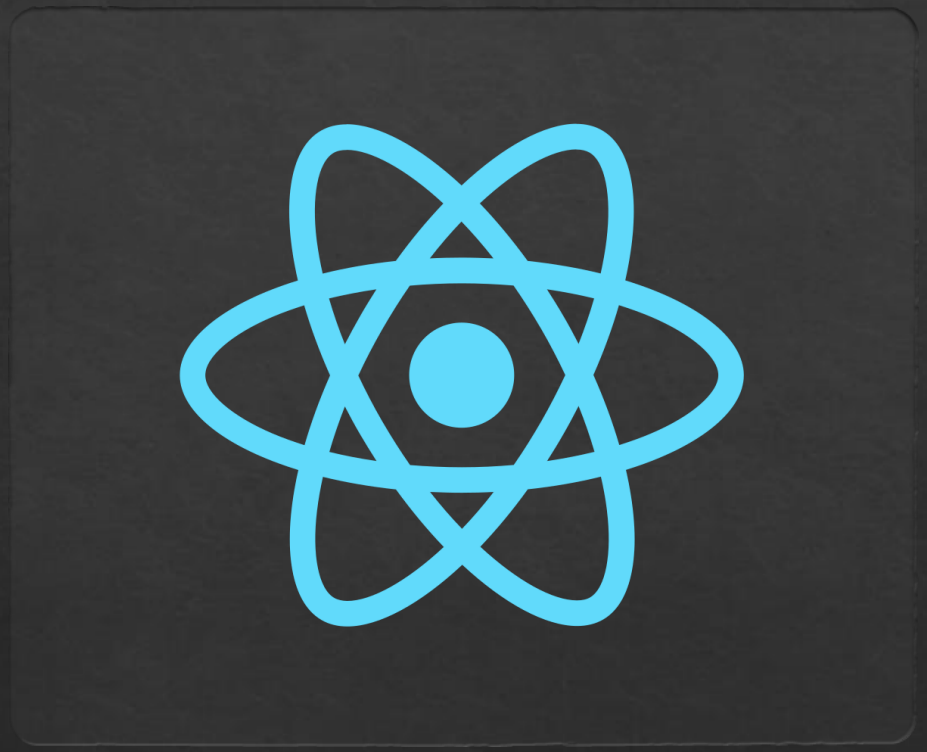
- **React-native init ReactNativeApp**

4

Start the Application

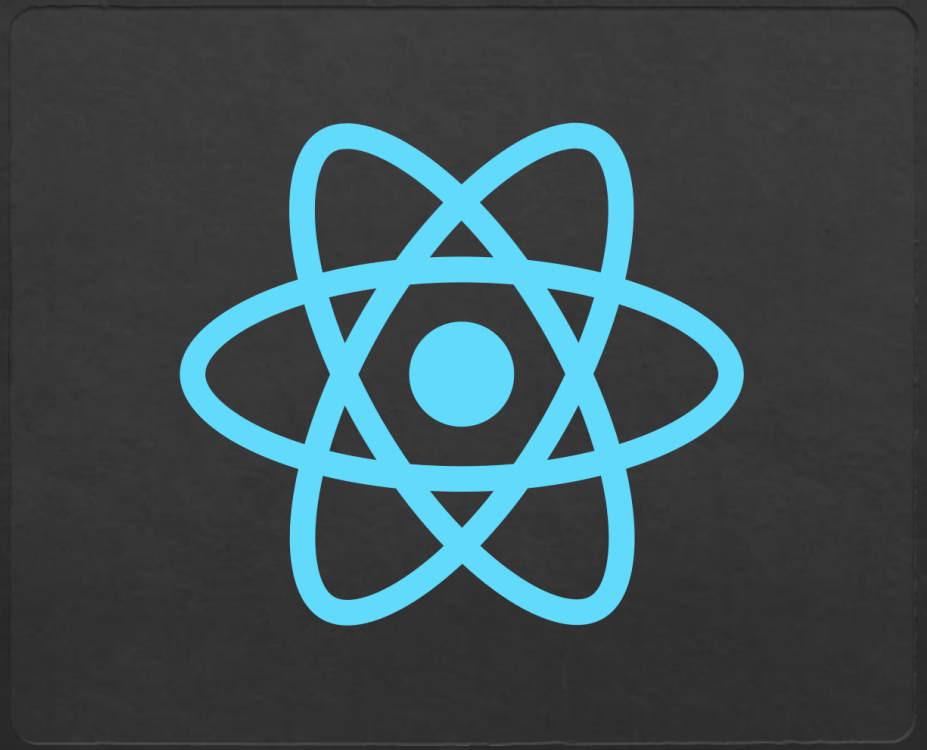
- **cd ReactNativeApp**
- **npm start**

What is React?



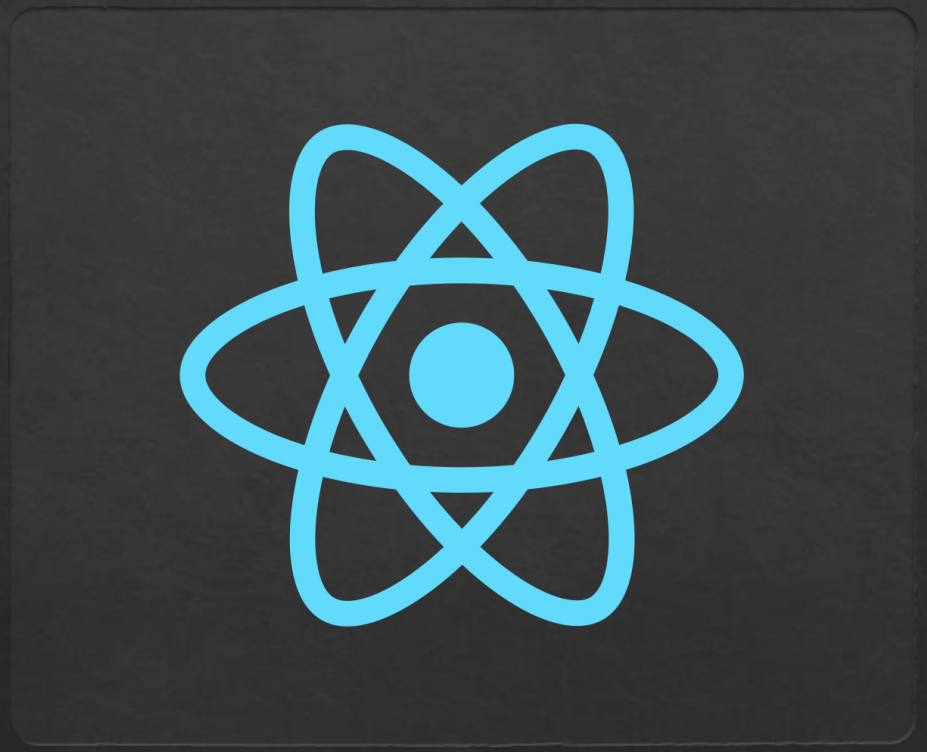
- React Native is built on Reacts
- React is a JavaScript library to build ***User Interfaces***.
- Used for building applications for the ***browser***.
- React allows to create custom HTML elements called ***components***.
- Components are used to build ***complex User Interface***
- Reacts promotes writing ***maintainable, manageable and reusable code***.

Why React?



- React handles *State*
  - UI State is difficult to handle with JavaScript
- React focuses on business logic
  - Focus on business logic
- React is *fast*.
  - Apps made in React can handle complex updates and still feel quick and responsive.

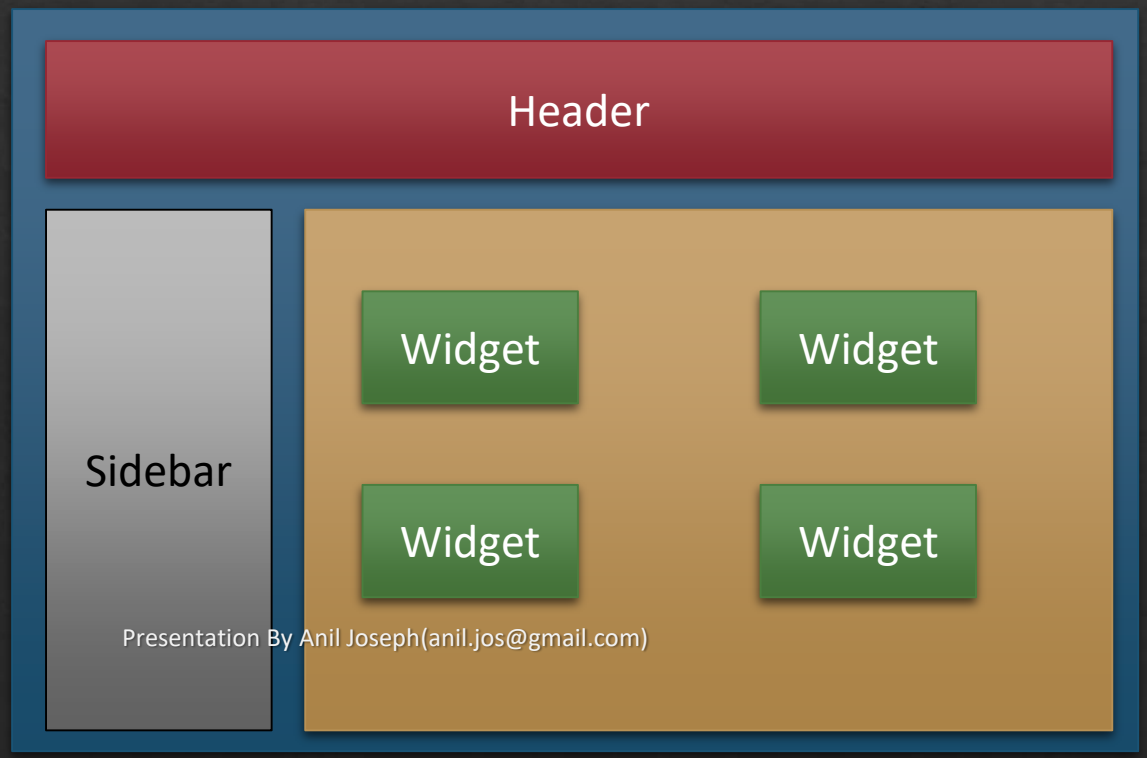
Why React?

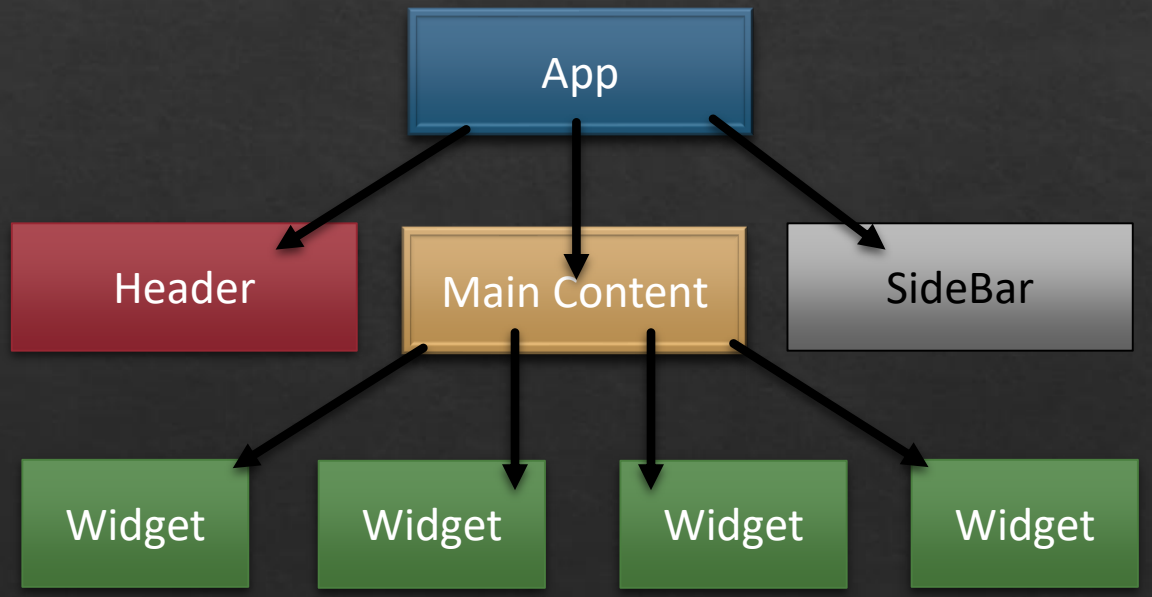


- React is *modular*
  - Instead of writing large, dense files of code, you can write many smaller, reusable files.
- React is *scalable*.
  - Large programs that display a lot of changing data are where React performs best.
- React has a Huge ecosystem, community

React Native Components

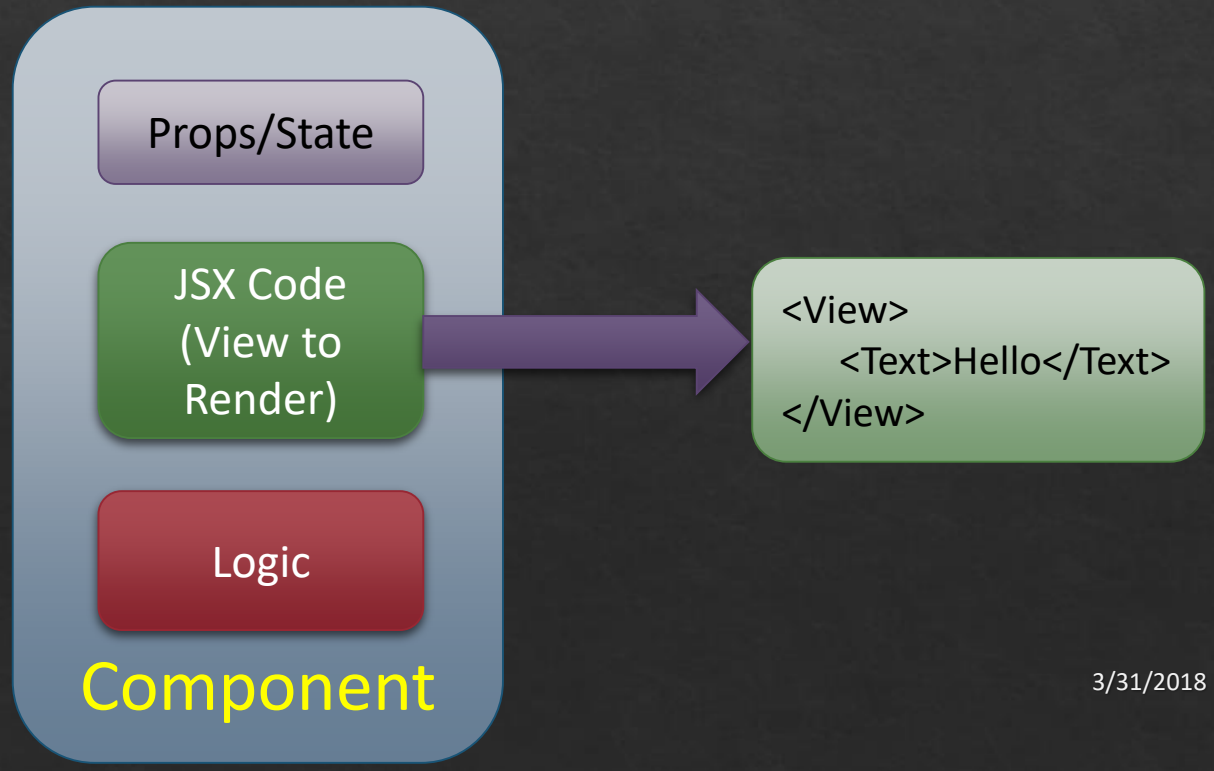
- Components are the **core building blocks of ReactNative applications**
- Creating a React Application is all about creating components.
- A React app can be depicted as a **component tree**





ReactNative Components

- The UI in a ReactNative Application is composed of ***components,*** the building blocks.
- Components are designed to be reusable.



Types of Components

Functional

- Presentational
- Stateless

Class Based

- Containers
- Stateful

JSX

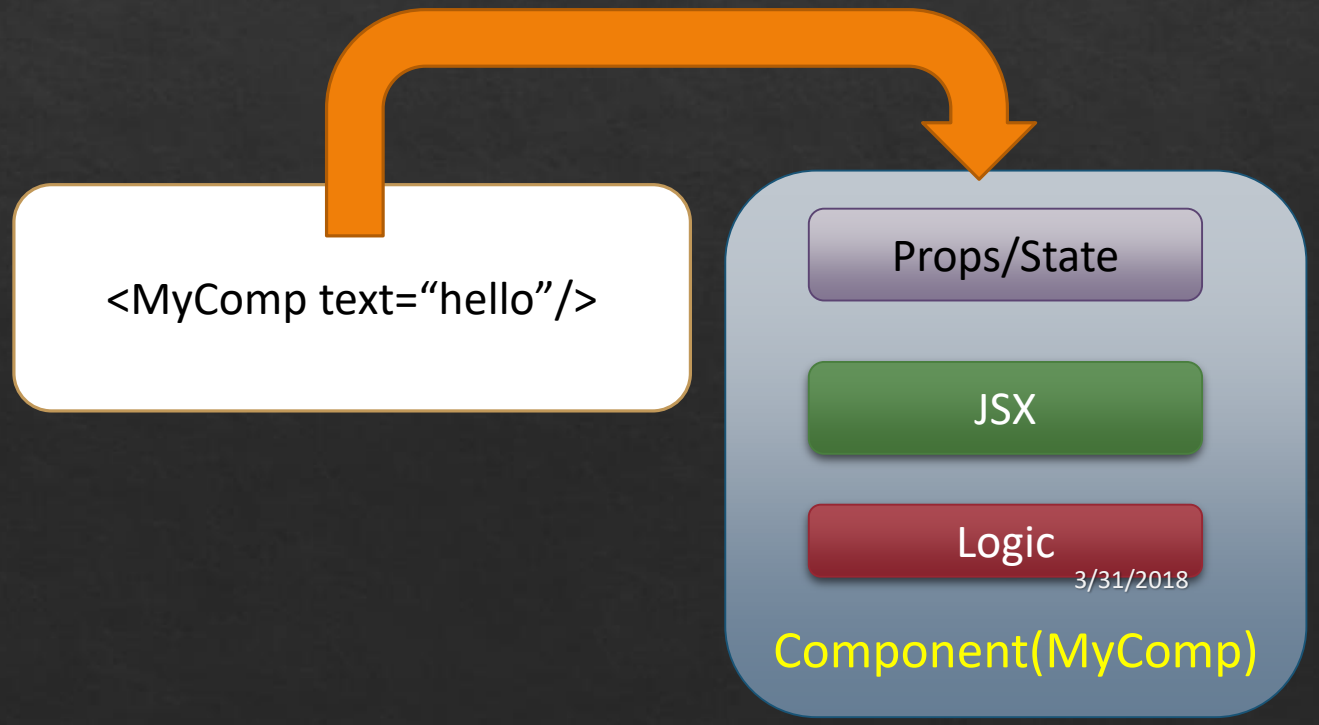
- *JSX* is a syntax extension for JavaScript.
- It was written to be used with React.
- JSX code looks a lot like Markup.
  - **It's actually JavaScript**
- A *JSX compiler* will translate any JSX into regular JavaScript.
- JSX elements are treated as JavaScript *expressions*.
  - They can go anywhere that JavaScript expressions can go.
- That means that a JSX element can be
  - Saved in a variable
  - Passed to a function
  - Stored in an object or array

Components: Dynamic Content

- Dynamic content is outputted in the JSX using an expression.
- The syntax
  - { expression }
- This can be any one line expression
- Complex functionalities can be done by calling functions
  - { invokeSomeMethod() }

Components: Properties(props)

- Most components can be customized with different parameters when they are created.
- These creation parameters are called ***"props"***.
- ***Props*** are used similar to HTML attributes.
- ***Changes to props will automatically re-render the component.***



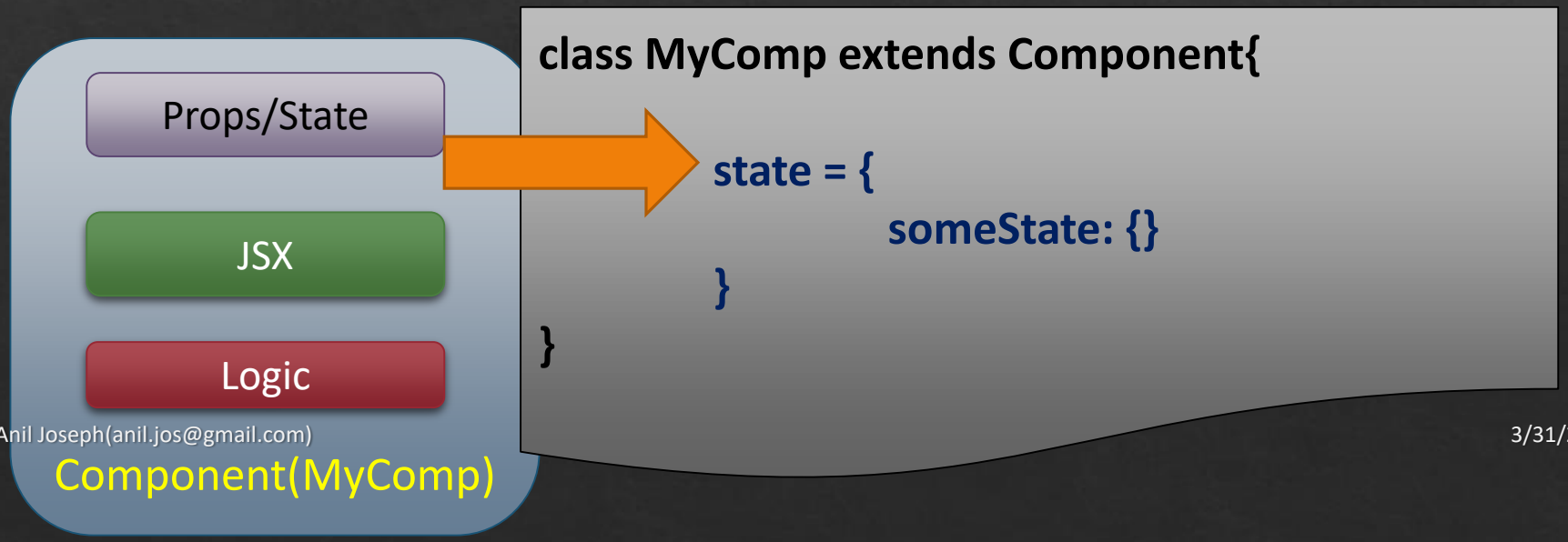
Props Types

Use Prop-Types for Type Checking the Component Properties

npm install prop-types

Component State

- State holds information about the component
- State is used when a component needs to keep track of information between renderings.
- State is created and initialized in the component itself.
- State is available only with Stateful components(Class Based).
- **Component state should be always updated using the setState method**
- **State updates trigger a rerender of the component.**



Props and State

"props" and "state" are CORE concepts of React.

Only changes in "props" and/ or "state" trigger React to re-render the components and potentially update the DOM in the browser

props allow you to pass data from a parent (wrapping) component to a child (embedded) component.

State is used to change the component from within.

Event Handling

- Handling events with React elements is very similar to handling events on DOM elements with some syntactic differences.
- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.
- Event handlers will be passed instances of ***SyntheticEvent***
  - A cross-browser wrapper around the browser's native event
- Details
  - https://reactjs.org/docs/events.html

React Native Components

React Native Components	• React Native provides a number of built-in components.
Basic Components	• View, Text, TextInput, Image, ScrollView, StyleSheet
User Interface	• Button, Picker, Slider, Switch
List Views	• FlatList, SectionList
iOS & Android Components	• Components specific to iOS and Android
Documentation	• View the official pgae for the full list • https://facebook.github.io/react-native/docs/getting-started

View

View is the most fundamental component for building a UI

View is a container

View supports layout with Flexbox

Supports Touch Handling

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

Flexbox

The View component can specifies the layout of its children using the Flexbox algorithm

Flexbox is designed to provide a consistent layout on different screen sizes.

Flexbox works the in React Native similar to CSS on the web

Flexbox properties

- flexDirection
- alignItems
- justifyContent

Components

Button

- A basic button component that should render nicely on any platform. Supports a minimal level of customization.

Text

- A React component for displaying text.
- Text supports nesting, styling, and touch handling.

TextInput

- A foundational component for inputting text into the app via a keyboard.
- Supports features like auto-correction, auto-capitalization, placeholder text, and different keyboard types

Image

- A React component for displaying different types of images

Components

KeyboardAvoidingView

- This component solves the problem of views that need to move out of the way of the virtual keyboard.
- It automatically adjusts either its position or bottom padding based on the position of the keyboard.

ScrollView

- Component that wraps platform ScrollView while providing integration with touch locking "responder" system

Modal

- The Modal component is a simple way to present content above an enclosing view.

Components

WebView

- Renders web content in a native view.

FlatList

- A performant interface for rendering simple, flat lists, supporting the most handy features
- Supports many features like Header, Footer, Pull to Refresh, Seperator, Scroll Loading

ActivityIndicator

- Displays a circular loading indicator.

Picker

- Renders the native picker component

RefreshControl

- This component is used inside a ScrollView or ListView to add pull to refresh functionality

Touch

- React Native provides components to handle all sorts of common gestures.
- For advanced gesture recognition there is a ***gesture responder system.***
- Components
  - Button
  - Touchables
  - ScrollView

Styles

- React Native uses JavaScript to define styles(Its doesn’t have a special language or syntax).
- Styles defined using JavaScript in React Native emulates CSS styles
- All of the core components of React Native accept a prop named style
- The style prop can be a plain old JavaScript object.
- Recommended to use the StyleSheet.create method to create the style
- StyleSheet.create validates and converts to native styles

Component Lifecycle

Component Lifecycle (Mount)

constructor

- Setup state
- Don’t Cause Side-Effects

componentWillMount

- Update State
- Don’t Cause Side-Effects
- Configuration

render

- Prepare and Struture JSX code

Render Child Components

ComponentDidMount

- Cause side effects
- **Initialize anything that relies on the DOM**

Component Lifecycle (Update)

componentWillReceiveProps(nextProps)

- Sync State to Props
- Don’t Cause Side-Effects

shouldComponentUpdate(nextProps,nextState)

- Decide whether to Continue or Not
- Don’t Cause Side-Effects

componentWillUpdate(nextProps, nextState)

- Sync State to Props
- Don’t Cause Side-Effects

render()

Update Child Component Props

componentDidUpdate()

- Cause Side-Effects
- Don’t Update State

Component Lifecycle (Unmount)

componentWillUnmount

- Remove Listeners
- Cancel Active network calls
- Invalidate Timers

Component API’s

- Alert
  - Launches an alert dialog with the specified title and message.
- AsyncStorage
  - A simple, unencrypted, asynchronous, persistent, key-value storage system that is global to the app.
  - It should be used instead of LocalStorage.
- Dimensions
  - Gets the dimensions of the screen and notifies changes to the screen
- Geolocation
  - The Geolocation API extends the Geolocation web spec.
- Linking
  - Linking gives you a general interface to interact with both incoming and outgoing app links.

Component API's

- NetInfo
  - NetInfo exposes info about online/offline status
- Keyboard
  - Keyboard module to control keyboard events.
- CameraRoll
  - CameraRoll provides access to the local camera roll or photo library.

Networking

- ReactNative does not have any API's for AJAX calls.
- We have to use an AJAX Library for server communications
- Popular Libraries
  - Axios
  - jQuery AJAX
  - Fetch API
- In a component AJAX calls to fetch data from the server should be made in the ***componentDidMount*** lifecycle method.

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';

Debugging

- Error Messages
  - Messages generated by React during development mode
- Browser Developer Tools
- React Tools
  - Tools for Chrome and Firefox
- Error Boundaries
  - Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree**
  - **Log errors**
  - **Display a fallback UI**

Immutability

- React components have the concept of *state*.
- Whenever state changes React re-renders the component.
- If state complicated (like with nested objects) it'd be hard to check whether your state changed or not.
  - Have to use *deep equality check*
- Immutability
  - *Whenever your object has to be mutated , don't mutate it*
  - *Create a copy*
- Immutability makes it easier to detect changes and update the UI

Virtual DOM

The Virtual DOM (VDOM)

- Where a “virtual”, representation of a UI is kept in memory
- This is synced with the “real” DOM.
- This process is called reconciliation.

Reconciliation

- When the render() function is called it creates a tree of React elements.
- On the next update render() will return a different tree
- React then figures out how to efficiently update the UI to match the most recent tree.
- Uses the Diff algorithm

Higher-order Components

- A Higher Order Component is just a React Component that wraps another one.
- Higher Order Components is a Pattern used extensively with React
- Uses
  - Code reuse, logic and bootstrap abstraction
  - Render Highjacking
  - State abstraction and manipulation
  - Props manipulation
- Its basically a functions that returns a class component with the Wrapped Component.

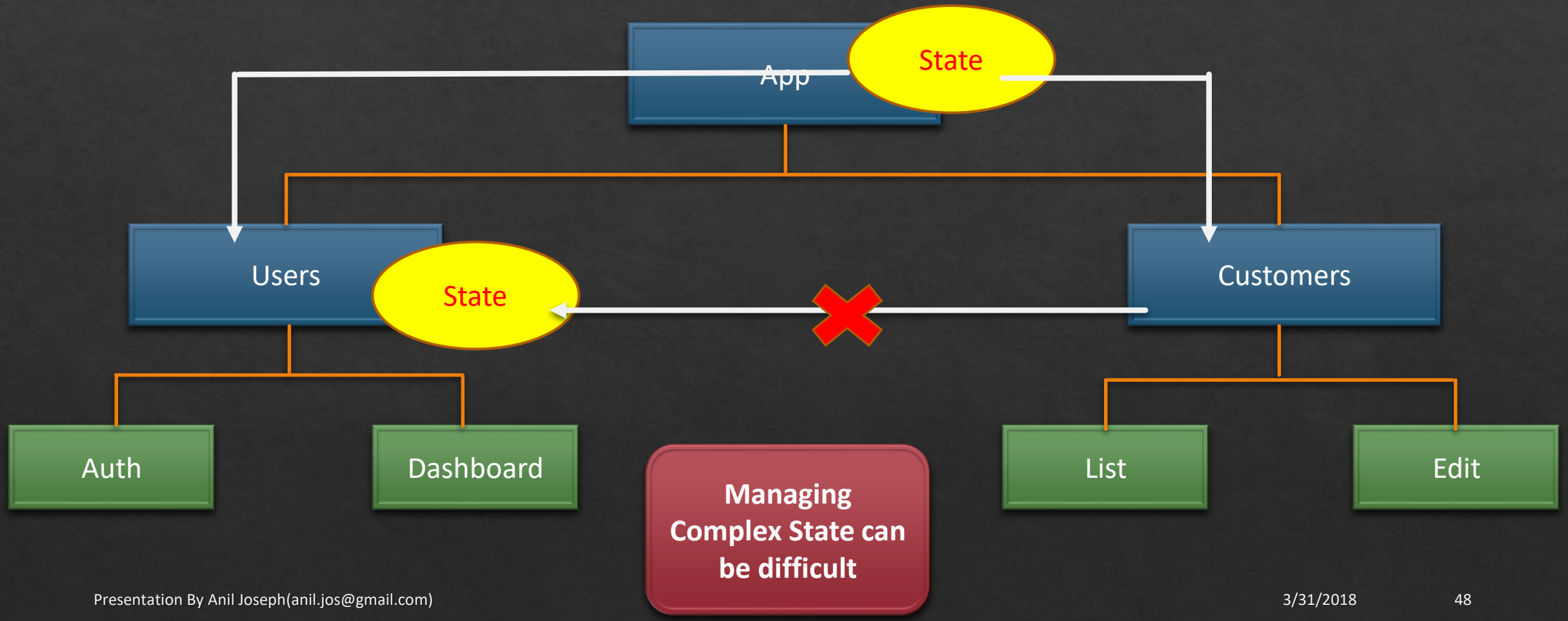
Redux

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

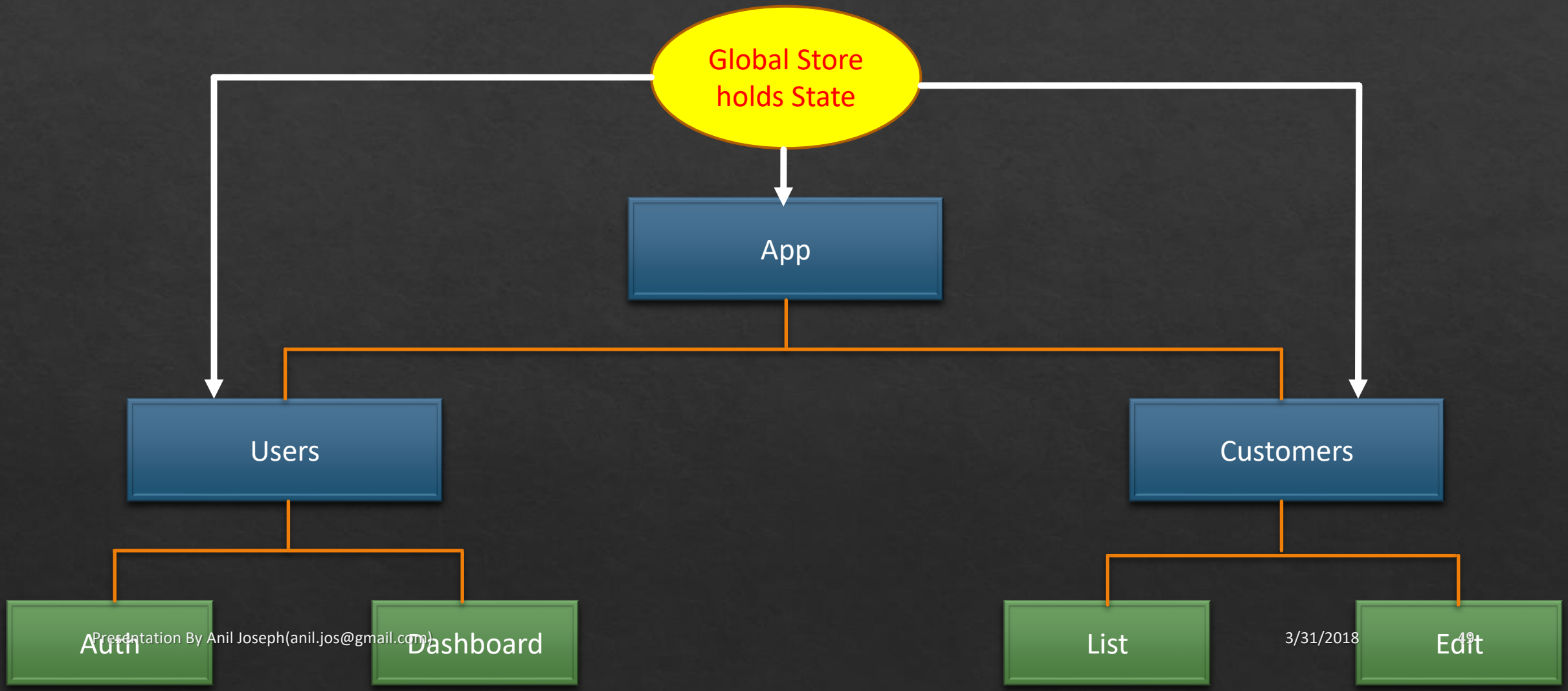
It is primarily used together with React or Angular for building user interfaces.

Redux was built on top of functional programming concepts.

Why Redux?


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

Redux State Management



Redux Parts

Store

- Store is the object that holds the application state
- The entire state is represented by a single store.

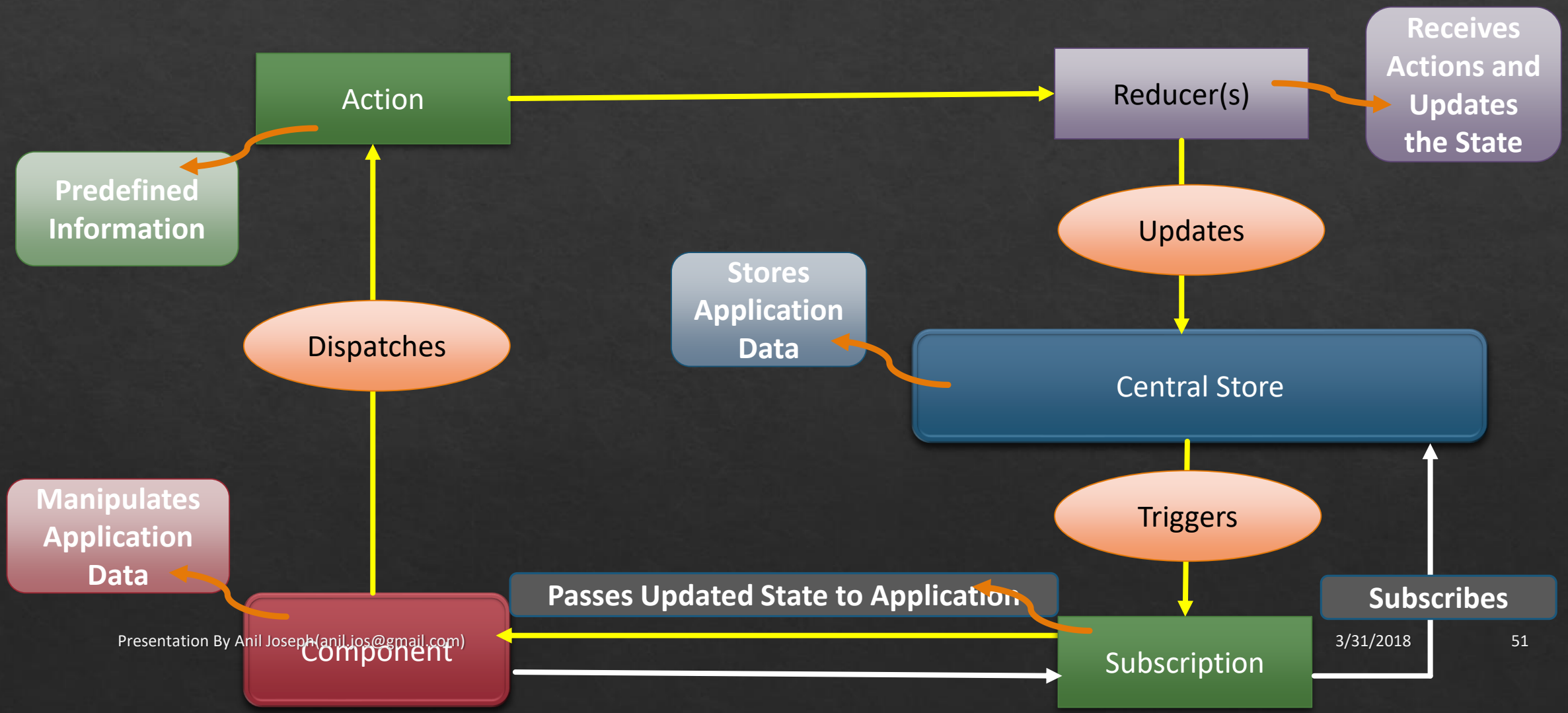
Actions

- Actions are events.
- Actions send data from the application (user interactions, internal events such as API calls, and form submissions) to the store.

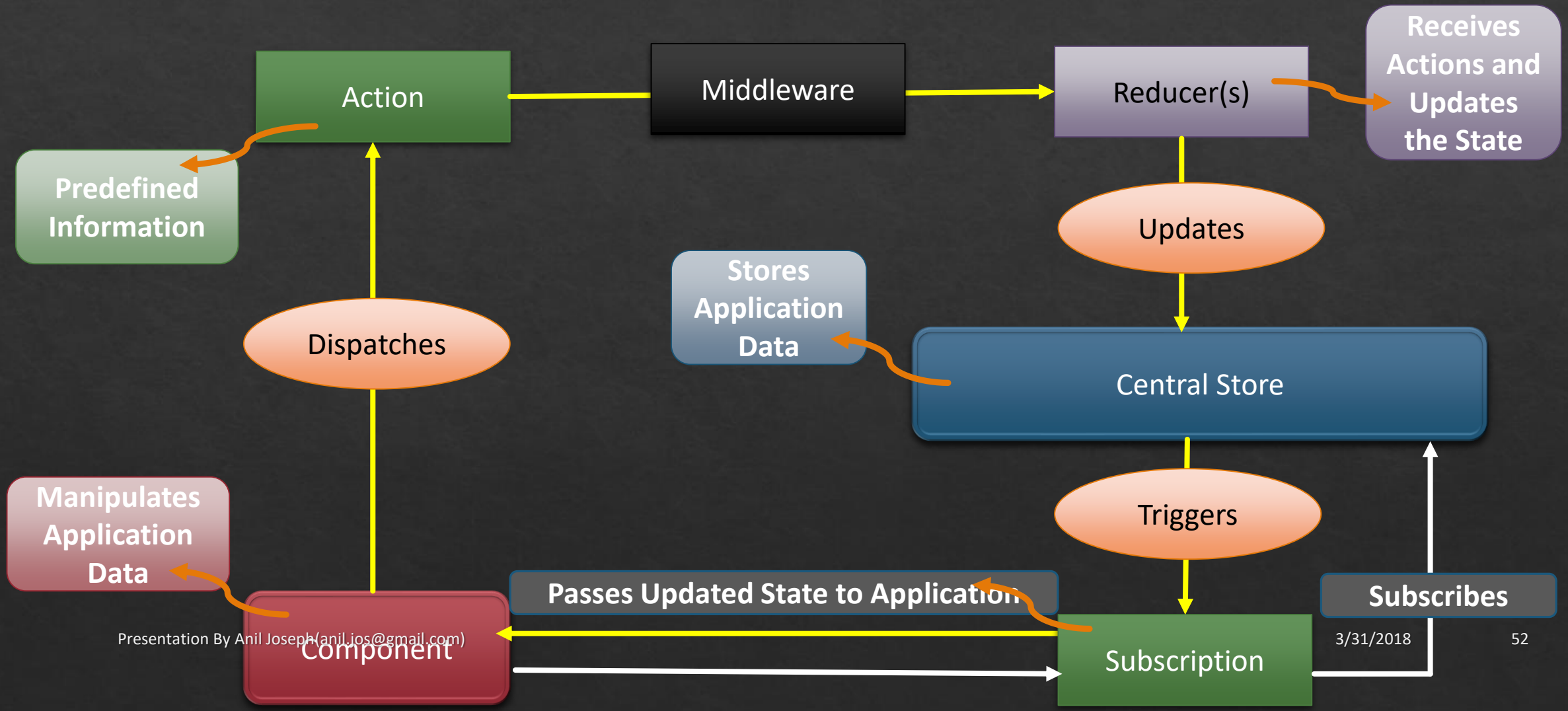
Reducers

- In Redux, reducers are functions (pure)
- Its takes the current state of the application and an action and then return a new state.

Redux Flow


Action
Reducer(s)
Receives Actions and Updates the State
Predefined Information
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes Updated State to Application
Subscribes
Component
Subscription

Redux Flow


Action
Middleware
Reducer(s)
Receives Actions and Updates the State
Predefined Information
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes Updated State to Application
Subscribes
Component
Subscription

Redux React

Redux react is a library that integrates Redux to a React Application

Comprises of Components & Functions

Installation

npm install react-redux

Types of State

Local UI State

- Show/Hide UI
- Handled By the Component

Persistent State

- Orders, Blogs
- Stored on Server, can be managed by Redux

Client State

- IsAuntenticated, Filter Information
- Managed by Redux

Navigation

- Navigation manages the presentation and transition of screens in an applications
- React Native recommends a number of libraries for Navigation
- Recommended Libraries
  - React Navigtion(https://facebook.github.io/react-native/docs/navigation#react-navigation)
  - NavigatorIOS(https://facebook.github.io/react-native/docs/navigation#navigatorios)
  - Native Navigation(http://airbnb.io/native-navigation/)
  - React Native Navigation(https://github.com/wix/react-native-navigation)

Platform Specific Code

- React Native provides two ways to easily organize your code and separate it by platform\
  - 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)

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

Resources

React: https://reactjs.org/

React Native: https://facebook.github.io/react-native/

React Native: http://www.reactnative.com/