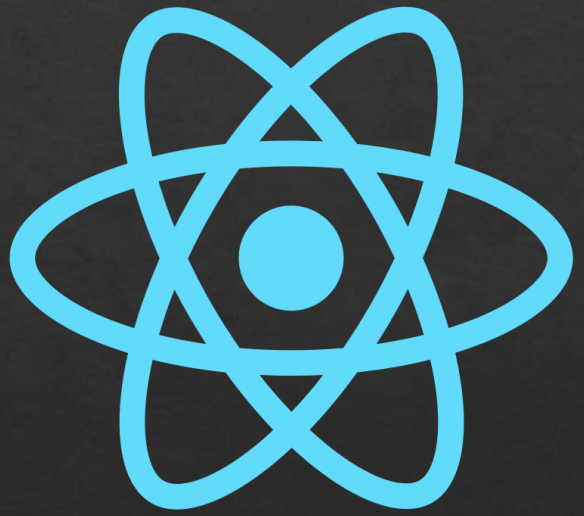
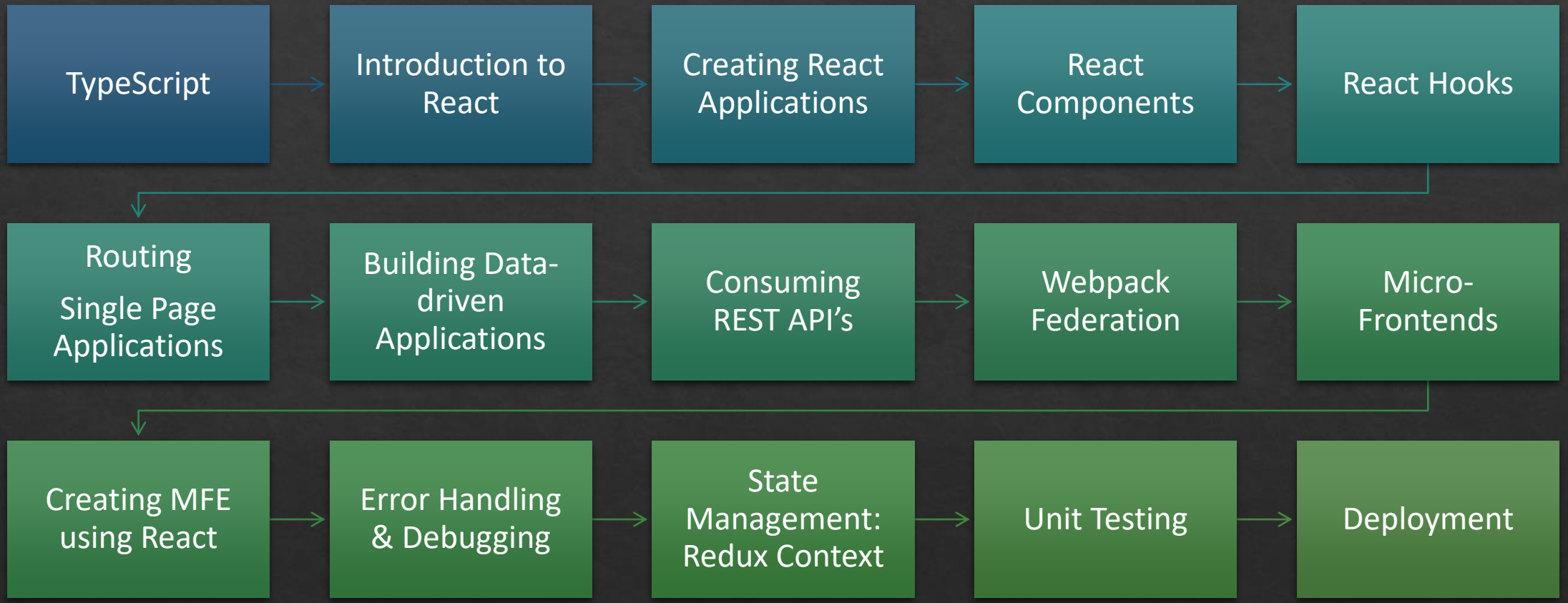
React & Micro-Frontends



ANIL JOSEPH

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

Agenda



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

Anil Joseph

Introduction

- Over 20 years of experience in the industry
- Technologies
  - C, C++
  - Java, Enterprise Java
  - .NET & .NET Core
  - **UI Technologies: React, Angular, jQuery, ExtJs**
  - Mobile: Native Android, React Native
- Worked on numerous projects
- Conducting trainings for corporates (700+)

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

Software

Node.js & NPM

HTML, CSS, JavaScript, TypeScript Editor

(Visual Studio Code)

Browsers(Chromium)

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

JavaScript

An interpreted language

Dynamic

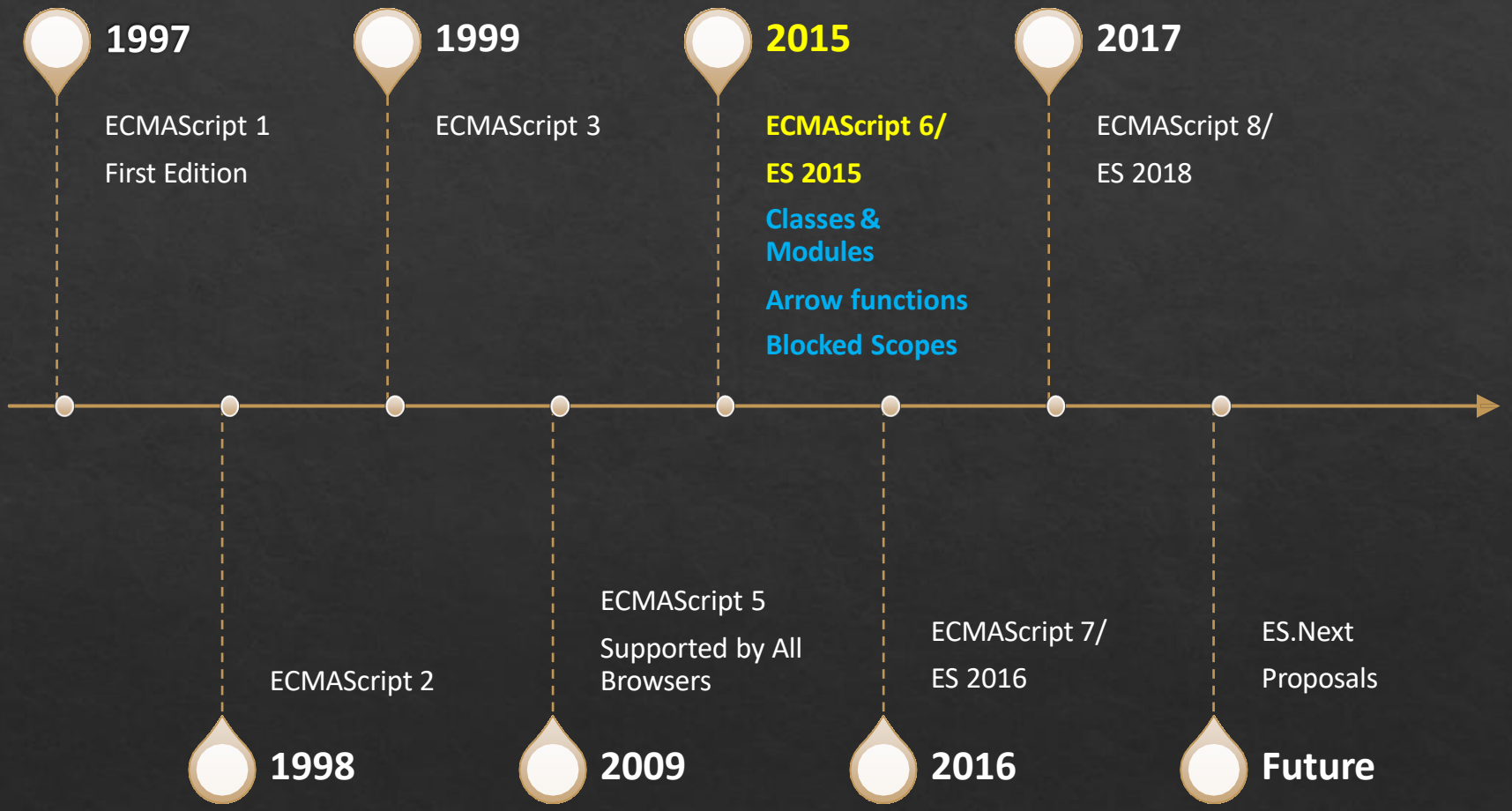
Object-oriented

Supports Functional style of programming

Available on the browsers and Node.js

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

ECMAScript Versions



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

TypeScript

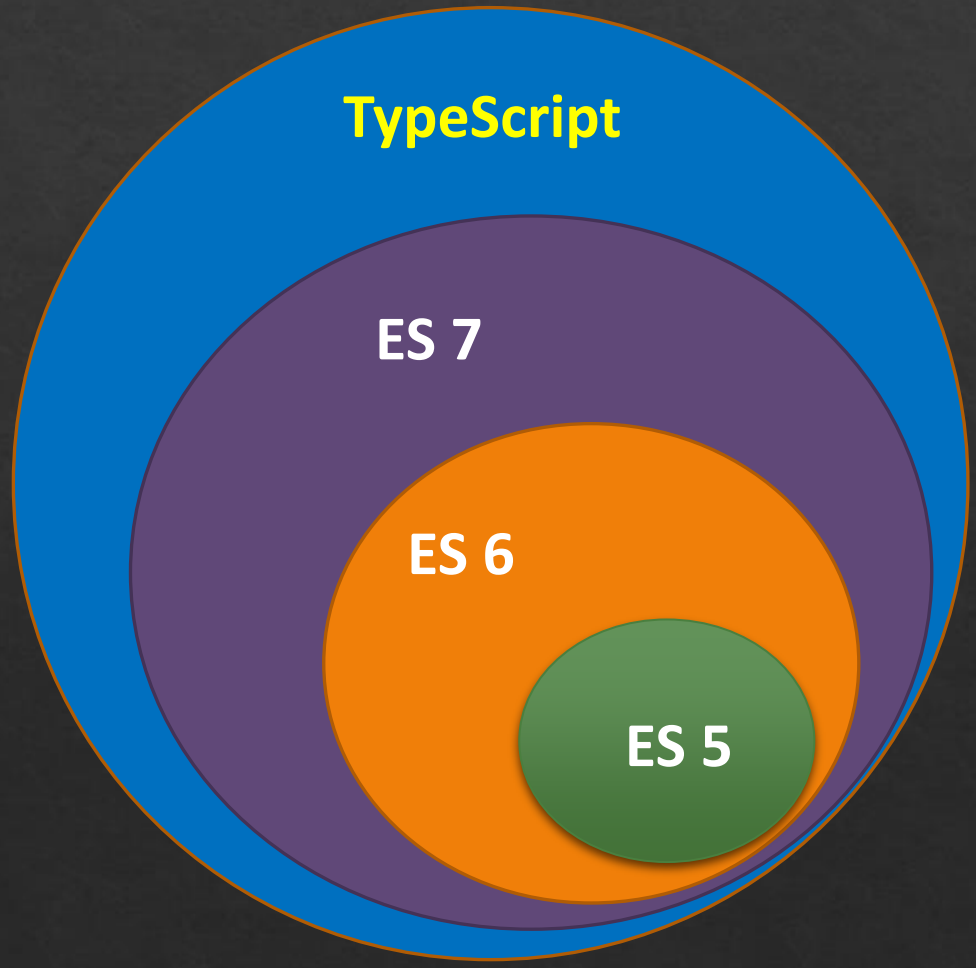
TypeScript is programming language developed and maintained by Microsoft.

TypeScript is a typed superset of JavaScript.

Transcompiles to JavaScript.

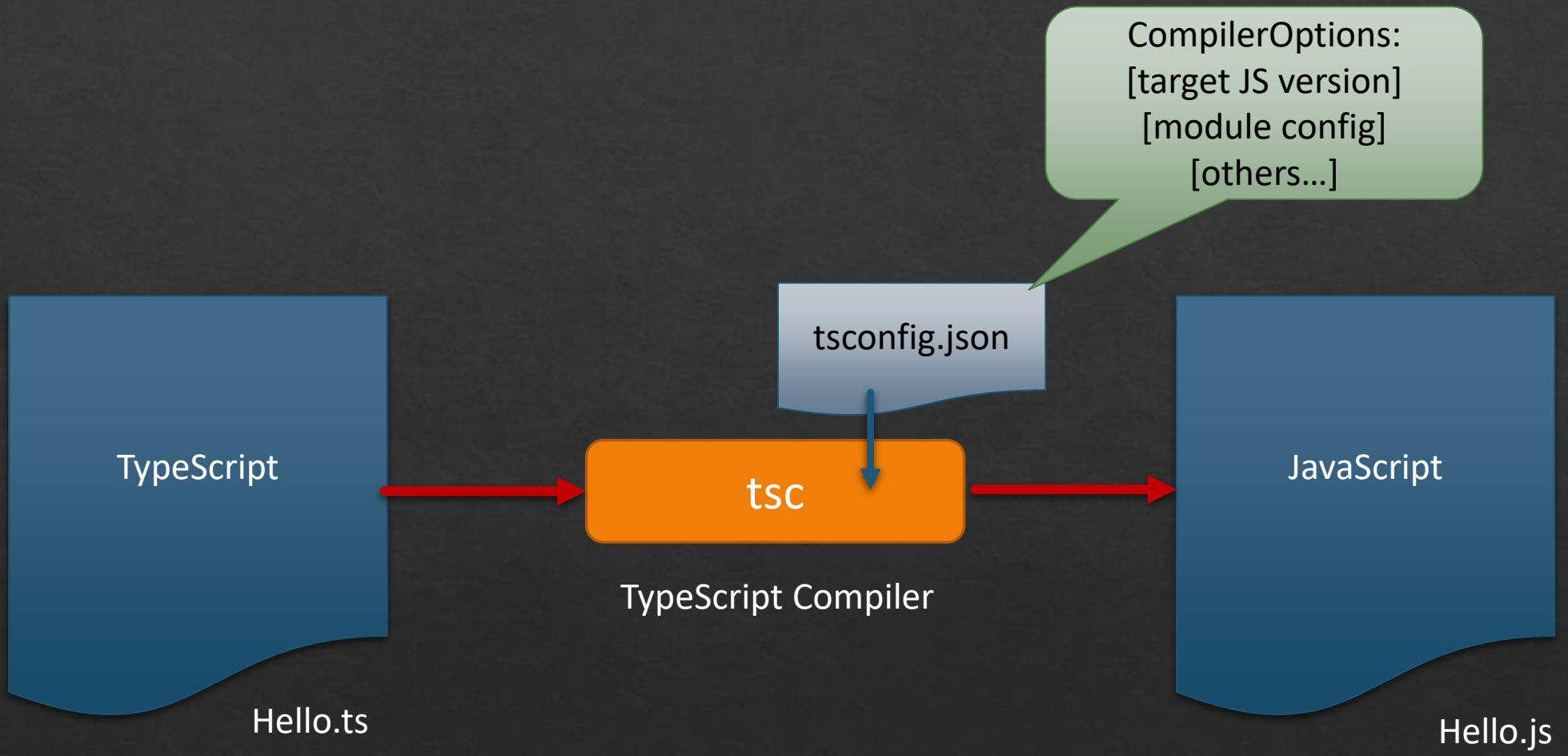
Designed for development of large applications.

Open Source.



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

TypeScript



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

TypeScript Features

- Type Annotations
- Compile-Time Type Checking
- Type Inference
- Interfaces
- Classes and Inheritance
- Namespaces and Modules
- Generics
- Decorators
- Arrow Functions

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

TypeScript Installation

- Install NodeJs and NPM
- Run the command **npm install -g typescript**

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

TypeScript Types

Boolean

- **let** isAvailable:boolean = false

Number

- **let** age: number = 16;

String

- **let** name: string = “Anil";

Array

- **let** list: number[] = [1, 2, 3];
- **let** list: Array<number> = [1, 2, 3];

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

TypeScript Types

Enum

- **enum** Color {Red, Green, Blue}
- **let** c: Color = Color.Green;

Any

- **let** x: any = 4;
- x = “hello”

void

```
function foo(): void {
  console.log(“foo");
}
```

null and undefined

- var x: string = null;
- var y:string= undefined

Defining New Types

- Type alias
  - Used to give a type a new name. It's like creating a shorthand for a more complex type.
- Interfaces
  - Used to define the shape of an object or the signature of a function. They are more extensible than type aliases because they can be reopened to add new properties.
- Classes
  - Define both the shape and the behavior of objects. A class can implement interfaces.
- Enums
  - Allow you to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases.

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

Type aliases

- A type alias is declared using the ***type*** keyword followed by an identifier and a type annotation. Once defined, the alias can be used anywhere a type can be used.
- Flexibility: Type aliases can represent primitive types, unions, intersections, and any other valid TypeScript types.
- Readability: Using type aliases can make complex type definitions easier to work with and understand.

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

Interfaces

- Interfaces are a powerful way of defining contracts.
- Example

```
interface Vehicle{

    name: string;
    speed: number;
    gear?: number;

    applyBrakes(decrement: number): void;
}
```

- Interfaces can extend interfaces
  - (keyword extends)
- Classes implements interfaces
  - (keyword implements)

Classes

- Access Modifiers
  - public, private, protected
- Constructors
- Properties
- Static Members
- Inheritance
- Abstract classes and methods

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

Arrow Functions

Represents a function expression

An arrow function expression has a shorter syntax than a function expression.

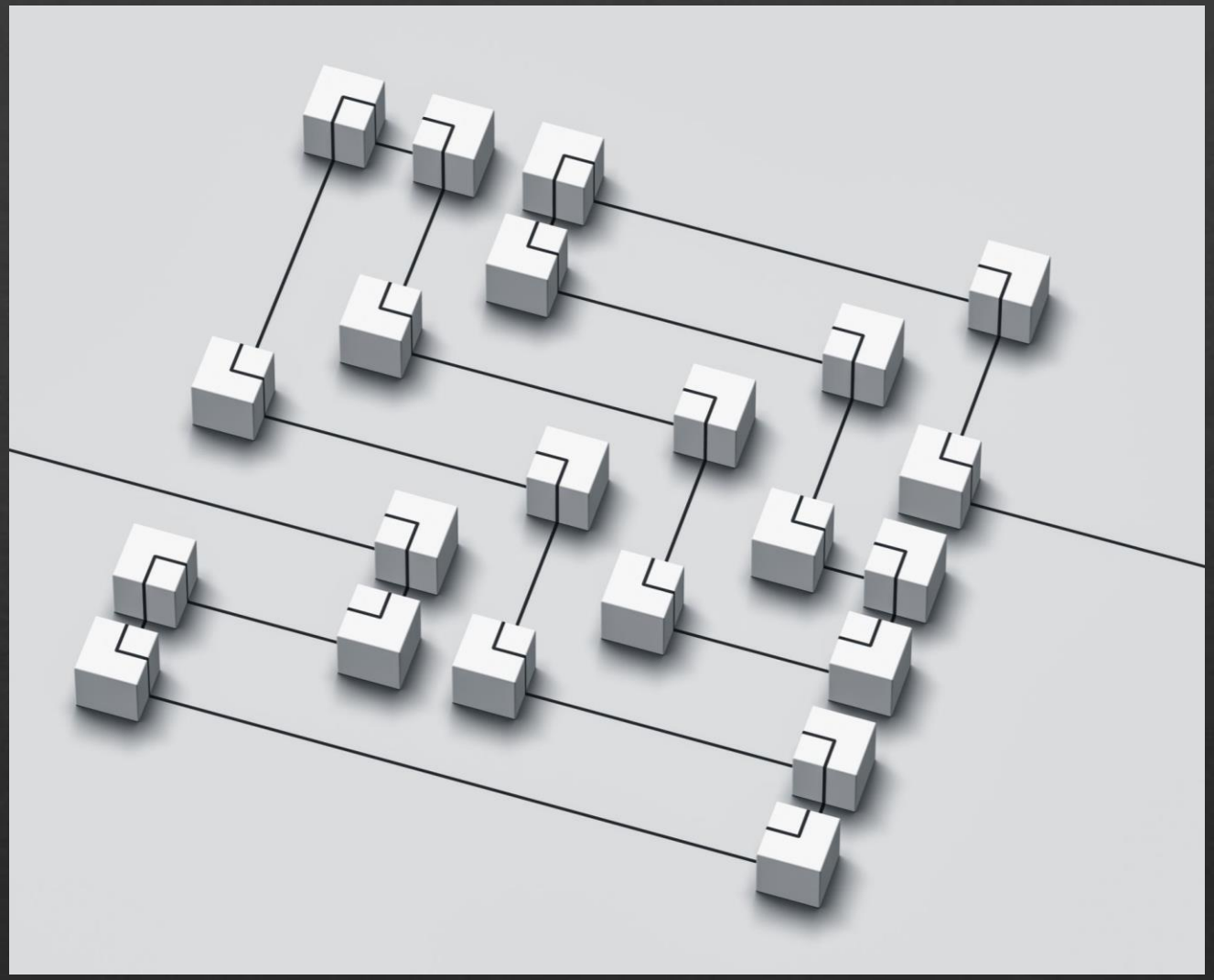
They do not receive the implicit arguments “this” and “arguments”.

Used widely for asynchronous and functional programming

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

TypeScript Modules

- Starting with ECMAScript 2015(ES6), JavaScript has the concept of modules.
- Modules have a scope of their own
- In the module system every JS file is a module and all declarations in the file is scoped to that module.
- The same concept is shared in TypeScript



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

Modules

- Use the import and export statements.

```
let foo = function(){
   //some code
}


export default foo;
```

one.js

```
import foo from ‘./one’;

foo();
```

two.js

```
import bar from ‘./one’;

bar();
```

three.js

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

Modules

```
export let foo = function(){
   //some code
}

 export let bar = function(){
    //some code
 }
```

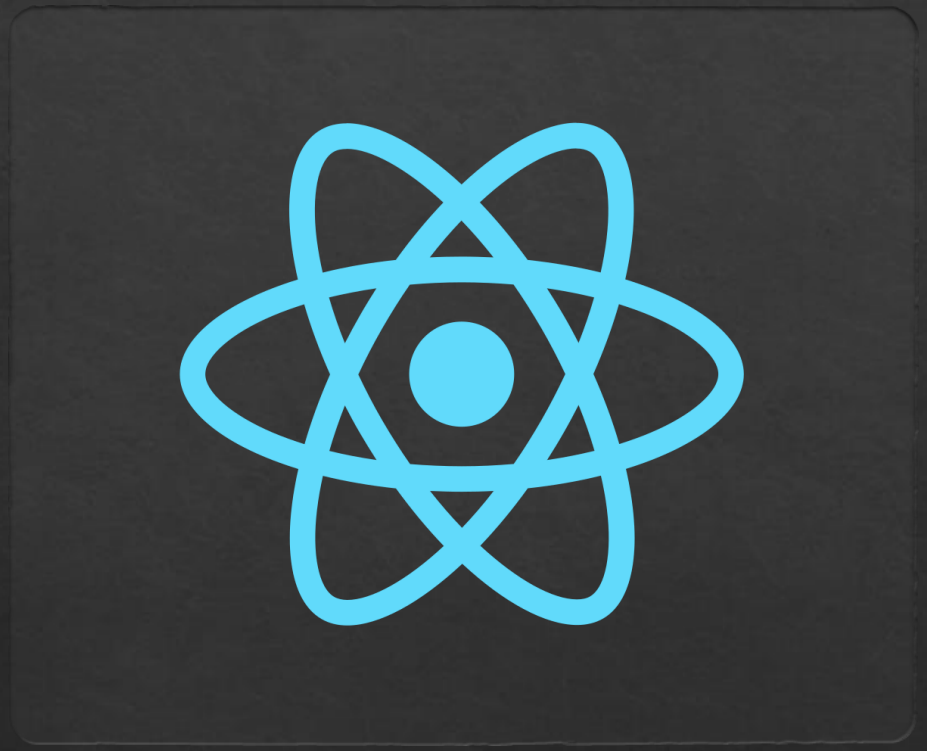
one.js

```
import {foo, bar} from './one';

foo();
bar();
```

two.js

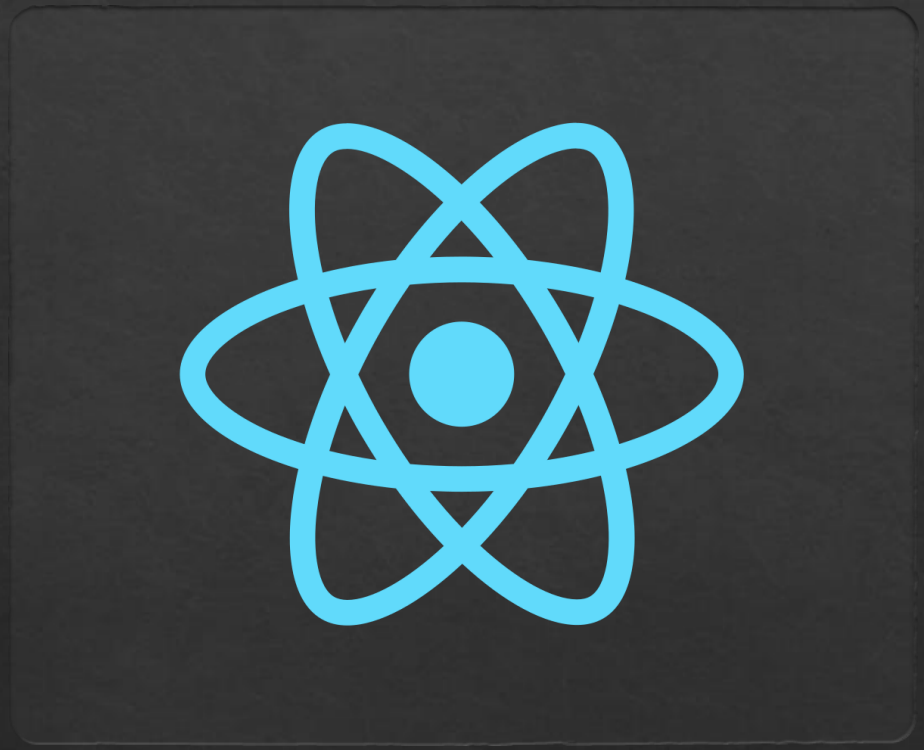
What is React?



- A JavaScript library to build ***User Interfaces***.
- 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***.

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

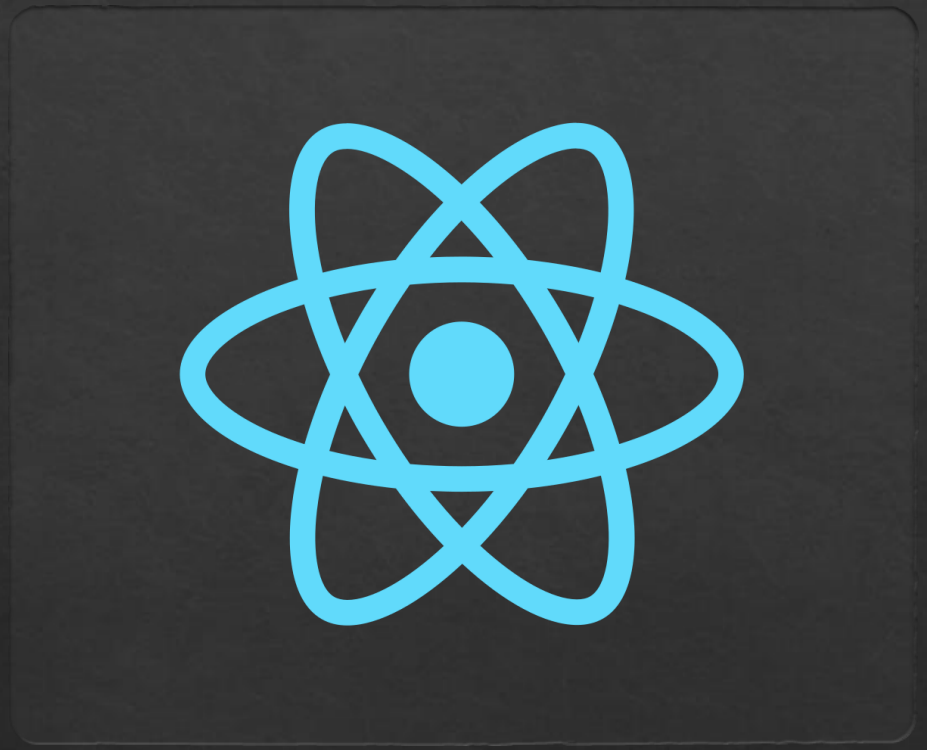
Why React?



- 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.

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

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

History

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

Getting Started

Two React Libraries

- React
- ReactDOM

JSX

- A JavaScript extension syntax allowing quoting of HTML

Babel

- The compiler for writing next generation JavaScript.
- Compiles JSX to JavaScript

Creating a React Project

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

Creating a React Project

- Create and configure the project manually.
- Requires knowledge of multiple tools and their configuration
- Need to keep up with new versions and technologies
- Highly Customizable

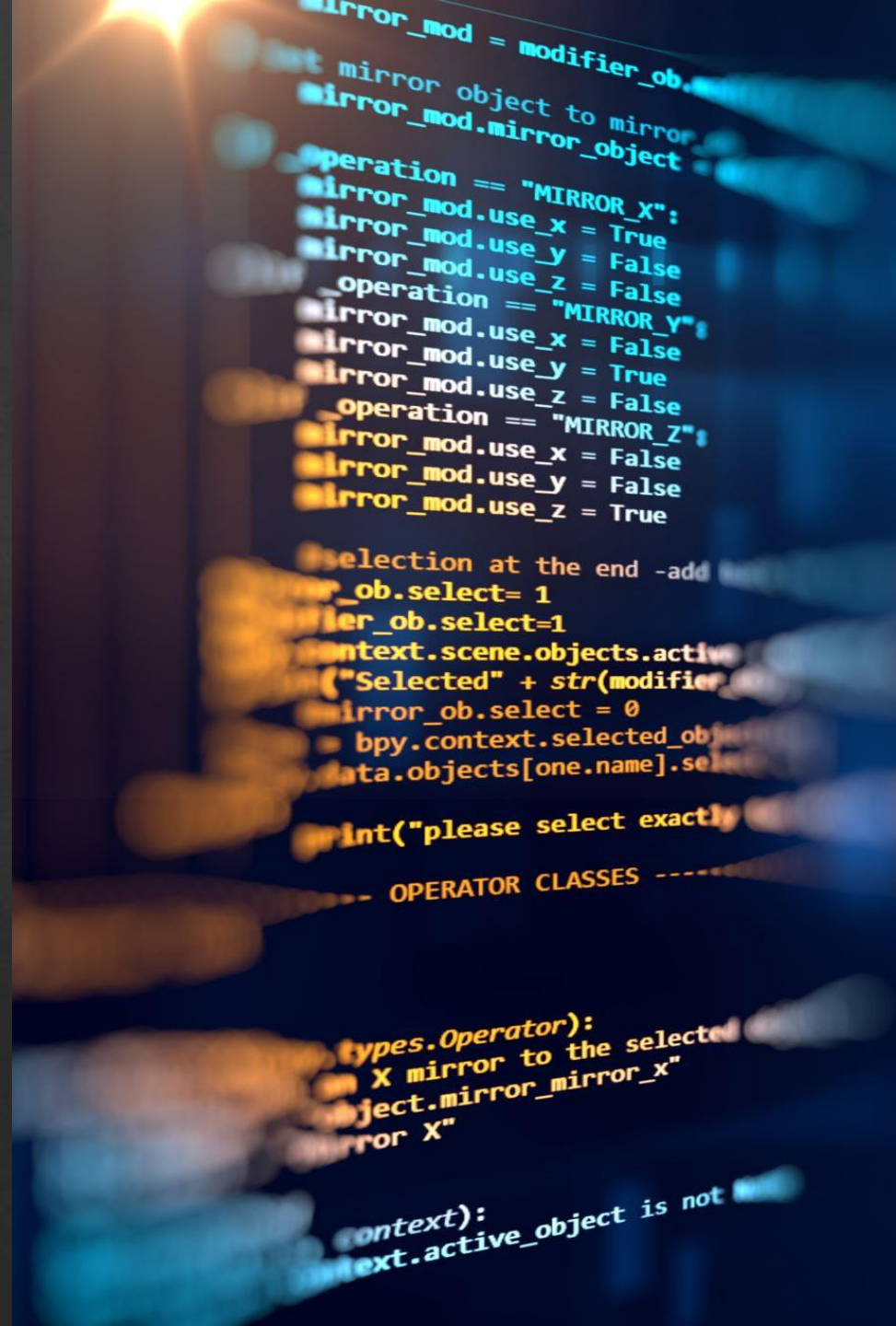
- Use a toolchain
- Easy to start with(Zero configuration)
- Customizations will be challenging

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

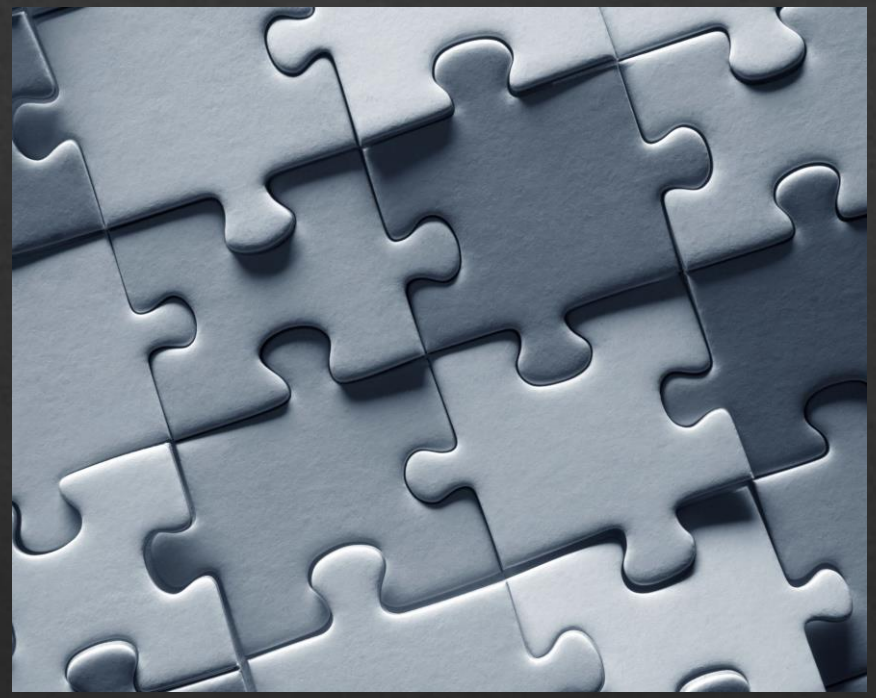
Tool Chains

- A toolchain is a set of programming tools that is used to perform a complex software development task or to create a software product.(Wiki)
- Using a toolchain provides a better developer experience
- Advantages of Tool Chains
  - Scaling to many files and components.
  - Using third-party libraries from npm.
  - Detecting common mistakes early.
  - Live-editing CSS and JS in development.
  - Optimizing the output for production.

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



Create React App



- Create React App is an officially supported way to ***create single-page React applications***.
- It offers a modern build setup with no configuration.
- Sets up the development environment to use the latest JavaScript features.
- Optimizes the application for production.
- Integrates the tools: NPM, Babel, Webpack, Webpack development server

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

Create Project

1

Install NodeJs

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

2

Install Toolchain

- **Install create-react-app**
- **npm install create-react-app -g**

3

Create React Application

- **create-react-app the-react-app**

4

Start the Application

- **cd the-react-app**
- **npm start**

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

Create Project

1

Install NodeJs

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

2

Create React Application

- **npx create-react-app the-react-app**

3

Start the Application

- **cd the-react-app**
- **npm start**

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

Create Project-Typescript

Install NodeJs

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

Create React Application

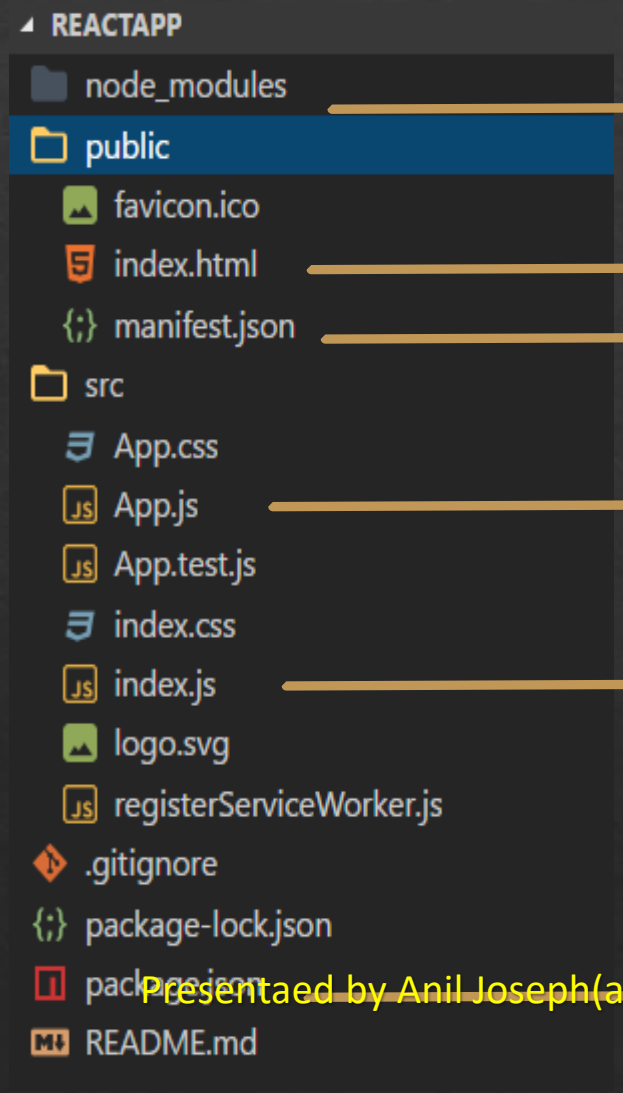
- **npx create-react-app the-react-app –template typescript**

Start the Application

- **cd the-react-app**
- **npm start**

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

Project Structure



Folder with all downloaded dependencies

The page displayed in the browser

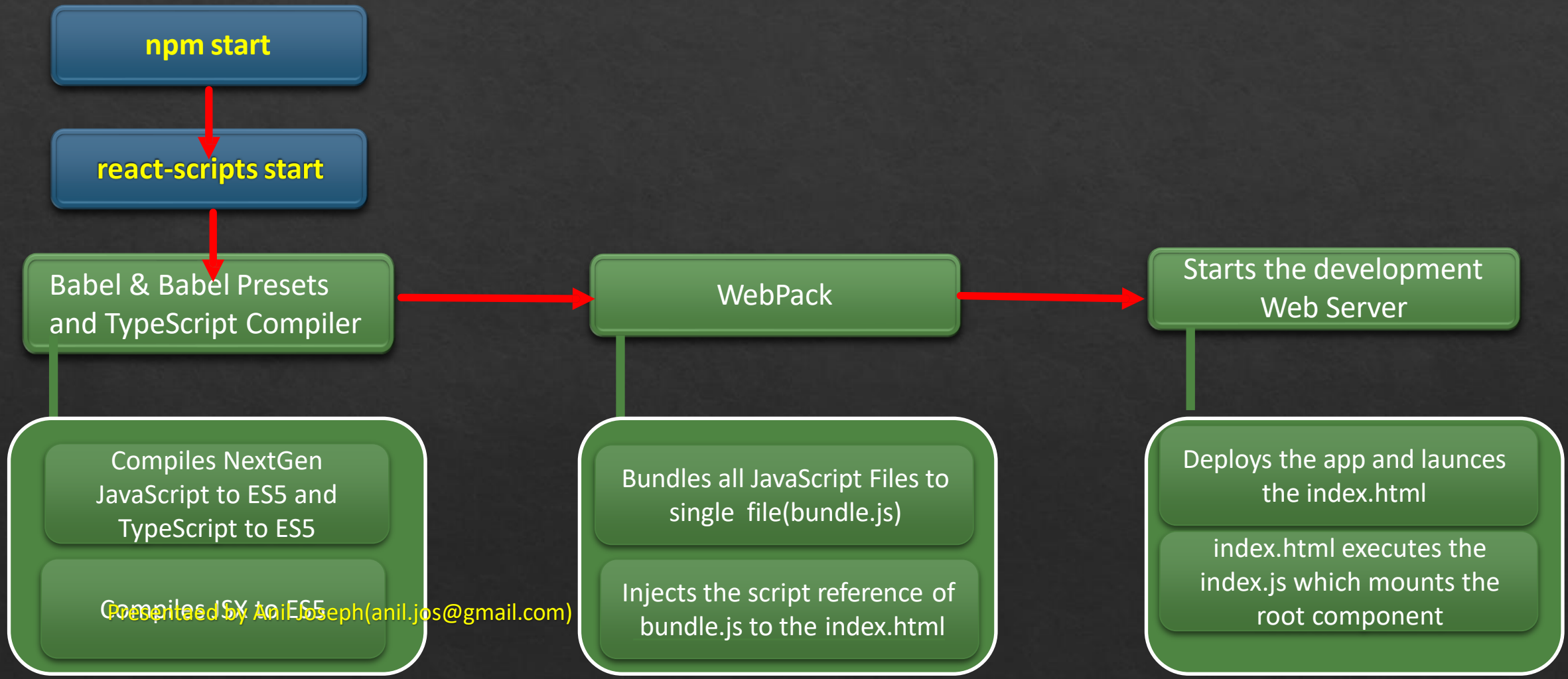
The Web Application Manifest (for installable web applications)

The root component

Initial Script executed on page load. Renders the root component

NPM configuration file, contains the dependencies and scripts

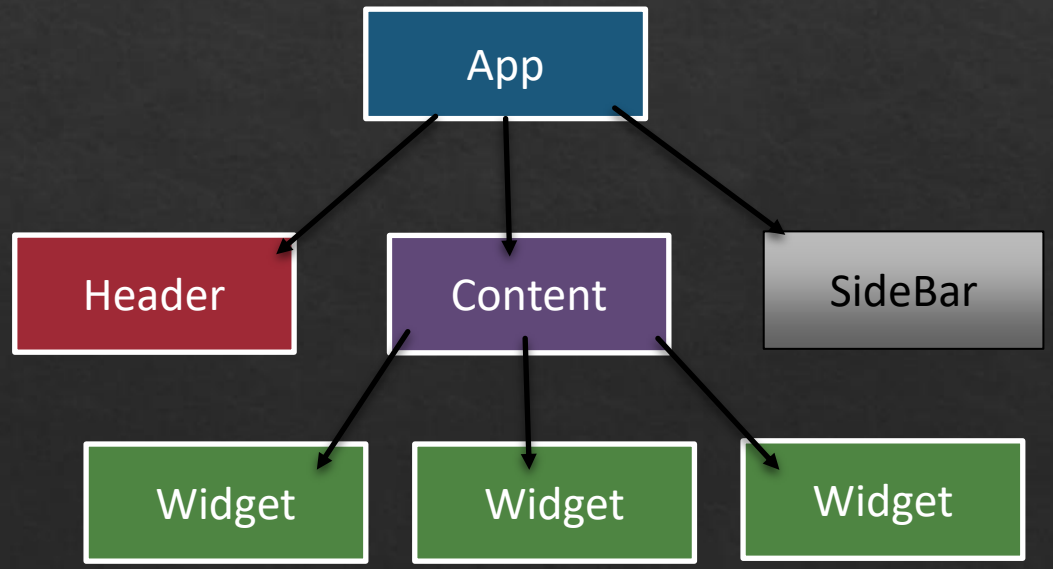
Project Execution



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

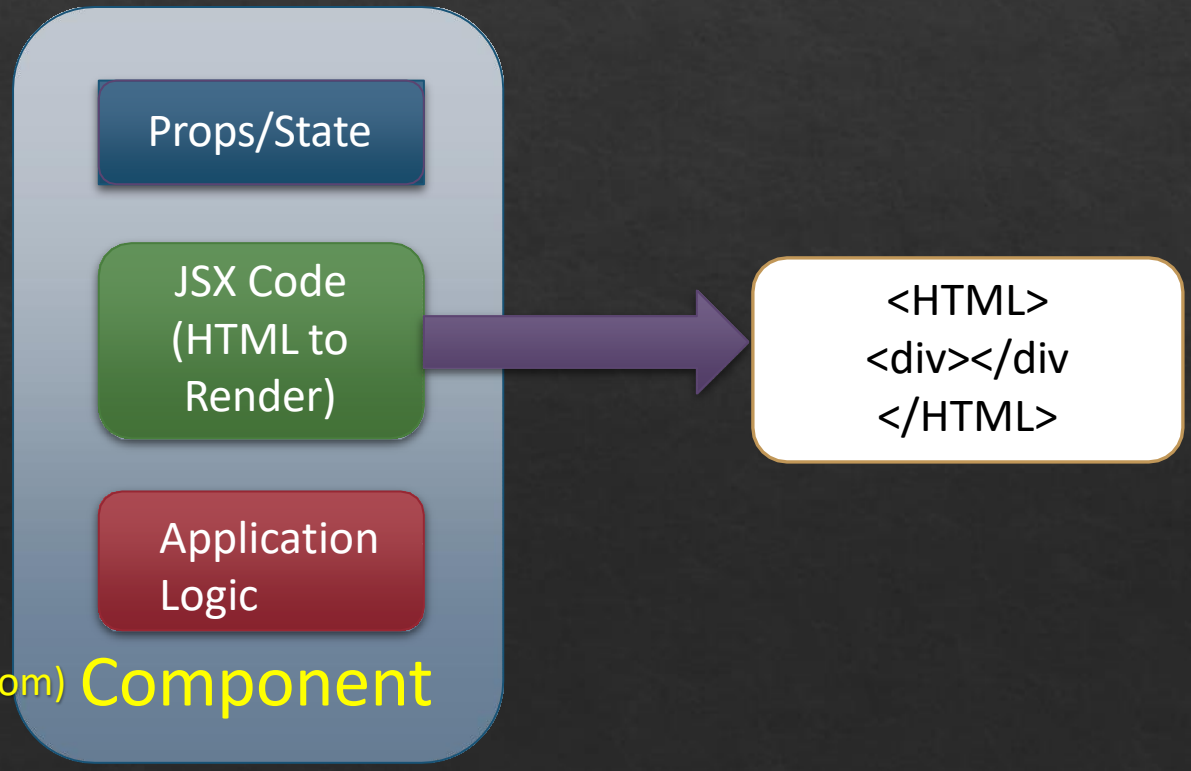
React Components

- Components are the core building blocks in React
- Creating a React applications is all about designing and implementing components
- A React application can be depicted as a component tree.



React Components

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



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

Types of Components

Functional

- Presentational
- Stateless (till React 16.8)
- Stateful (using React Hooks)

Class based

- Containers
- Stateful

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

JSX

- *JSX* is a syntax extension for JavaScript.
- It was written to be used with React.
- JSX code looks a lot like HTML.
  - **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() }

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

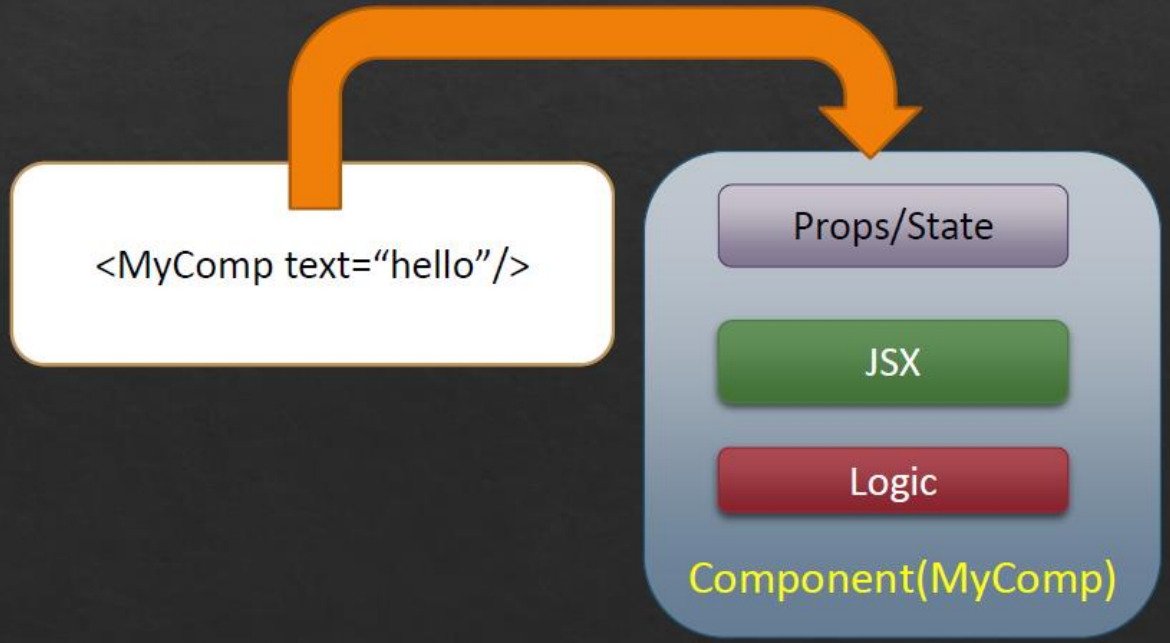
Components: Properties(props)

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.



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

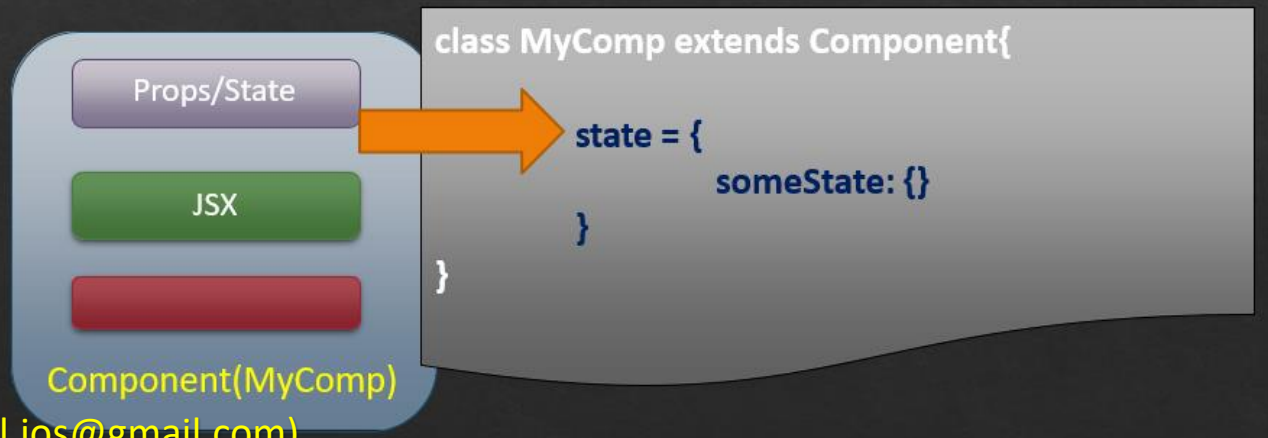
Component State

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 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.

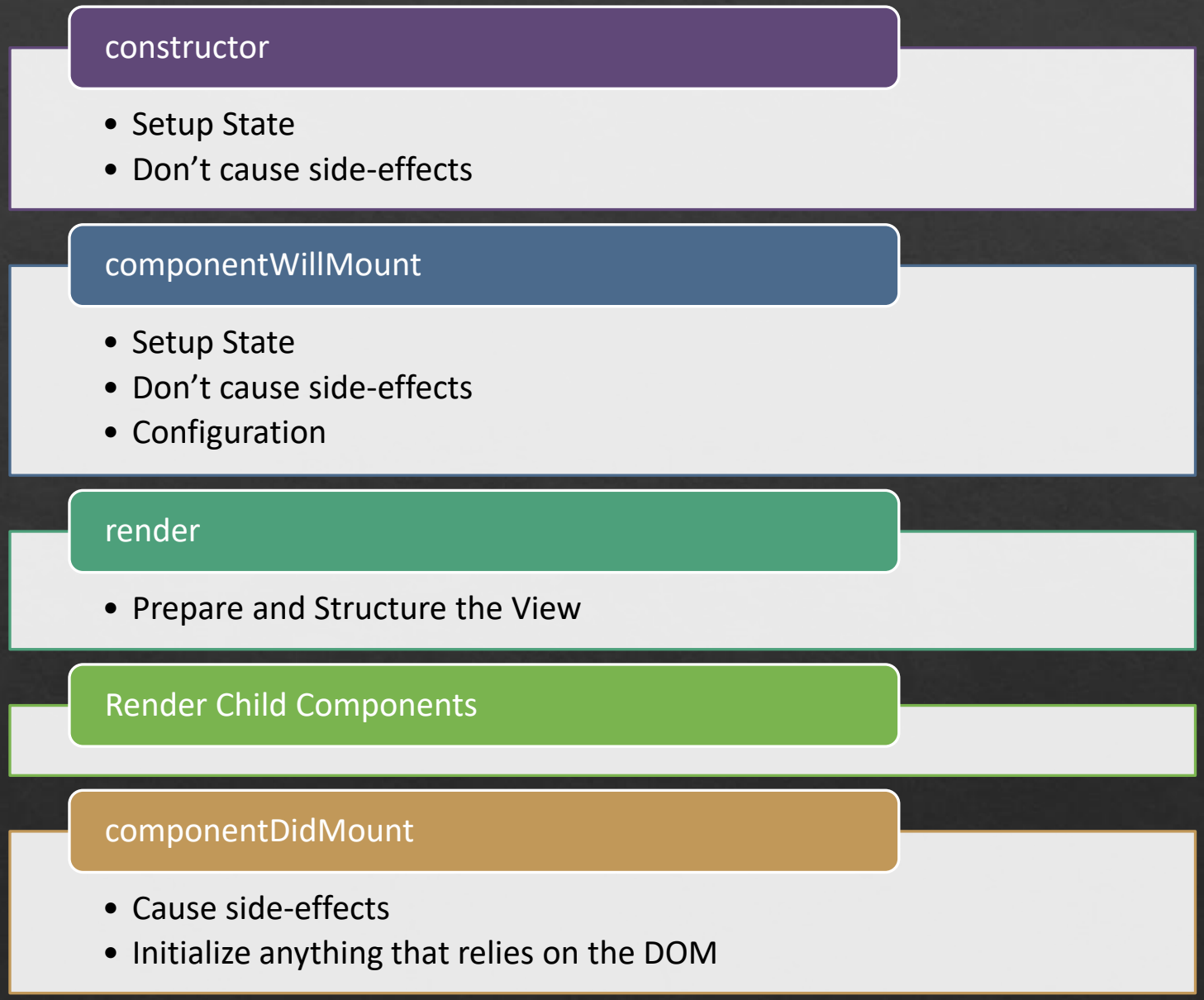
State is used to change the component from within.

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

Component Lifecycle

Component Lifecycle(Mount)



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

Component Lifecycle(Update)

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

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/Render Child Components

componentDidUpdate

- Cause Side-Effects
- Don’t update state

Component Lifecycle(Unmount)

- componentWillUnmount
  - Remove Listeners
  - Cancel Active network calls
  - Invalidate Timers

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

AJAX and APIs

- React 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';

AJAX and APIs

- React 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';

React Hooks

React hooks was introduced in version 16.8

Many React features like state, lifecycle hooks etc. were available only with class-based components prior to 16.8.

Hooks let us use state and other React features in a functional component.

React team recommends the functional components over class-based since they can be highly optimized by the tools.

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

React Hooks

State Hooks(useState)

- Equivalent of state in class-based components

Effect Hooks(useEffect)

- Used to perform side-effects in a function
- component
- Equivalent to lifecycle hooks in class-based components

Context Hooks(useContext)

- Used to access the React Context;

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

React Hooks

Callback Hooks(useCallback)

- Returns a memoized callback.
- Used to optimize the components

Memo Hooks(useMemo)

- Returns a memoized value.
- Used to optimize the components

Ref Hooks(useRef)

- Returns a mutable ref object

Custom Hooks

- A functions
- Uses Other Hooks

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

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**

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

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.

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

Single Page Applications

A single-page application is an application that works inside a browser and does not require page reloading during use.

SPA is fast, as most resources (HTML+CSS+Scripts) are only loaded once throughout the lifespan of application. Only data is transmitted back and forth.

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

Routing

- Routers allow to navigate between the different views in the application using JavaScript on the client-side.
- Navigations are updated to the browser history which allows to go back and forward
- Usage of path and search parameters are allowed
- React does not provide any API for routing.
- Many other Libraries available which integrates with React
  - React-router(The de-facto standard)
  - Director
  - Aviator
  - Finch

React Router

- React Router is a collection of **navigational components** that compose declaratively with your application.
- Installation
  - npm install react-router-dom

React Router Components

<BrowserRouter>

- A <Router> that uses the HTML5 history to keep your UI in sync with the URL.

<HashRouter>

- A <Router> that uses the hash portion of the to keep your UI in sync with the URL.

<Route>

- The Route component is perhaps the most important component in React Router

<Link>

- Provides declarative, accessible navigation around your application.

<NavLink>

- A special version of the <Link> that will add styling attributes to the rendered element when it matches the current URL.

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

State Management

React Context

- Available from React 16.3

React Redux

- Library to manage state

Mob

- Library to manage state

RxJs

- Reactive Programming using Observables

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

Types of State

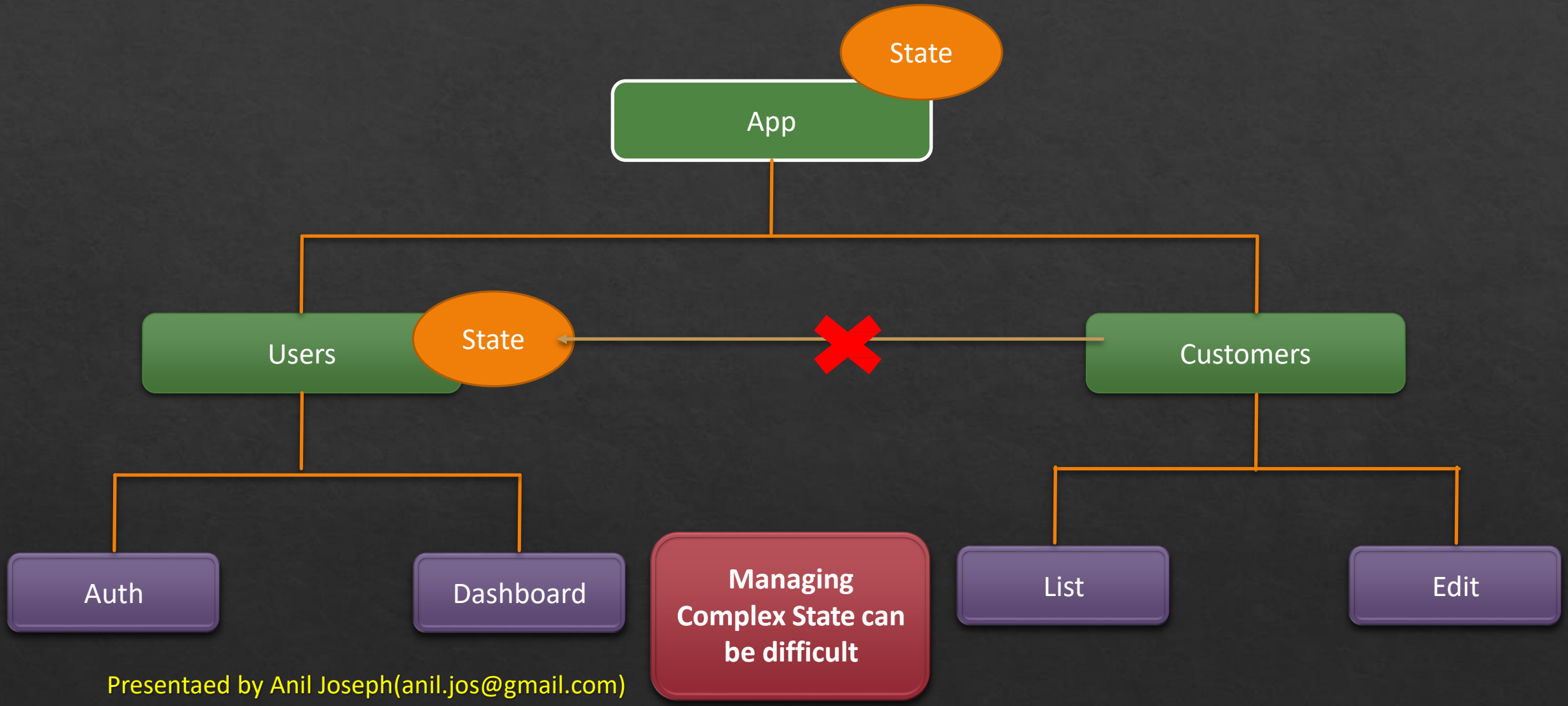
Redux

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.

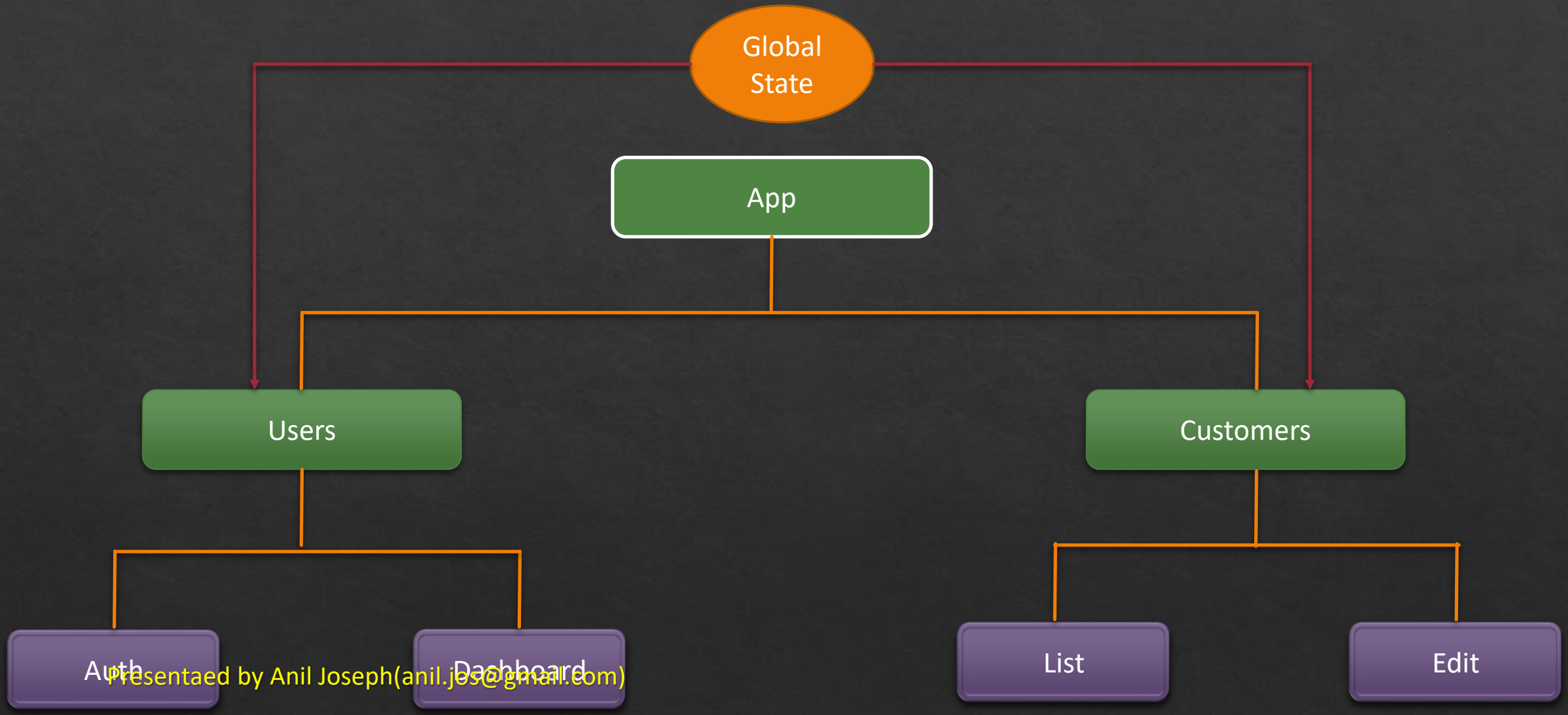
Redux was built on top of functional programming concepts.

Why Redux?

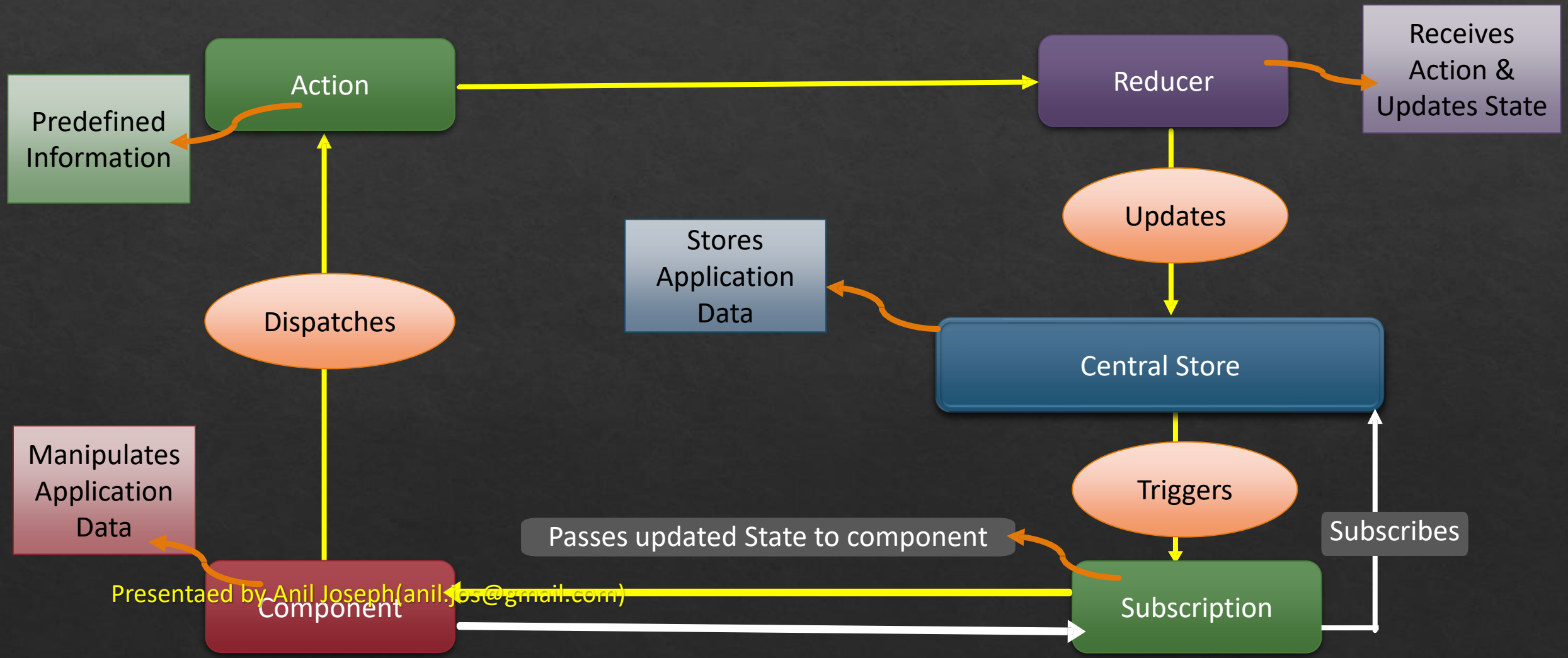


Presentaed by Anil Joseph(anil.jos@gmail.com)

Why Redux?

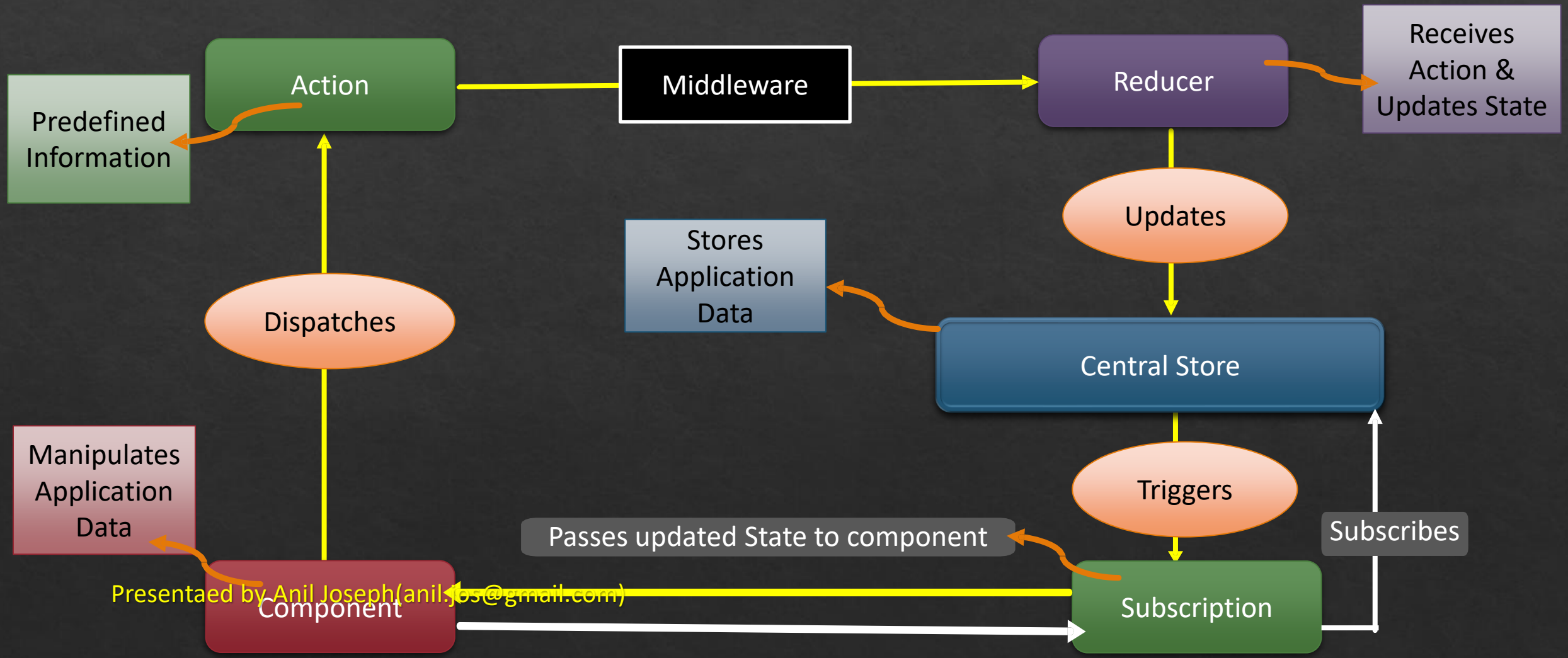


Redux Flow



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

Redux Flow



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

Redux Middleware

- Middleware in Redux provides an extension point between dispatching an action and the moment it reaches the reducer.
- Enables side effects, such as async operations, logging, and routing.
- Enhances functionality without modifying the core Redux workflow.

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

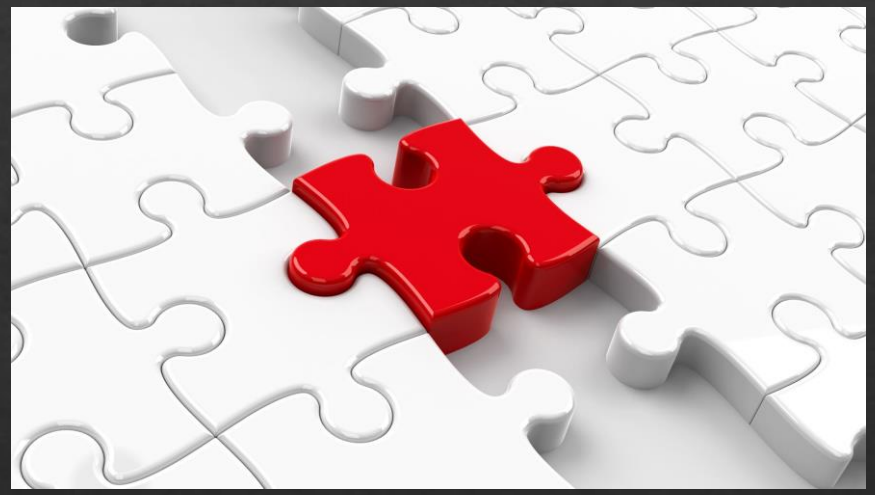
Redux Middleware

- Redux Thunk: Allows action creators to return a function instead of an action, enabling asynchronous logic.
- Redux Saga: Manages side effects with generator functions, providing more powerful async capabilities.
- Redux Logger: Logs actions and state changes for debugging purposes.

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

React Redux

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



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

Redux Toolkit

- An official, opinionated, batteries-included toolset for efficient Redux development.
- Simplifies the process of writing Redux logic and reducing boilerplate code.
- Benefits
  - Reduced Boilerplate:
    - Provides a set of tools and functions that encapsulate common Redux patterns.
  - Improved Code Readability
    - Cleaner and more maintainable code.
  - Optimized Performance
    - Built-in support for creating efficient Redux logic.

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

Error Boundaries

- Error boundaries are React components that catch JavaScript errors anywhere in their child component tree
  - Used to log errors
  - Display a fallback UI
- Error boundaries do not catch errors for:
  - Event handlers
  - Asynchronous code
  - Server side rendering
  - Errors thrown in the error boundary itself
- A class component becomes an error boundary if it defines(either one)
  - componentDidCatch
  - static getDerivedStateFromError()

React Context

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- Example of props to be passed down
  - Theme
  - Locale
  - Authenticated user
- API
  - React.createContext

Error Boundaries

- Error boundaries are React components that catch JavaScript errors anywhere in their child component tree
  - Used to log errors
  - Display a fallback UI
- Error boundaries do not catch errors for:
  - Event handlers
  - Asynchronous code
  - Server side rendering
  - Errors thrown in the error boundary itself
- A class component becomes an error boundary if it defines(either one)
  - componentDidCatch
  - static getDerivedStateFromError()

Code Splitting

- Code splitting allows you to split your app into separate bundles which your users can progressively load.
- API
  - React.lazy
  - Suspense

Micro-Frontends

- Micro-frontends are an architectural style where a frontend application is decomposed into smaller, individual, and loosely coupled pieces, each owned by different teams.
- These pieces, or micro-frontends, can be developed, tested, and deployed independently, allowing for greater flexibility and scalability.
- Each micro-frontend is responsible for rendering a specific part of the user interface
- Each micro-frontend can be built using different technologies or frameworks, making the overall application more modular and easier to maintain.
- The approach extends the principles of microservices to the frontend, promoting better separation of concerns and enabling teams to work more autonomously.

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

Benefits of MFE

- Independent Deployments:
  - Each micro-frontend can be deployed independently, allowing for faster and more frequent releases without affecting the entire application.
- Improved Scalability:
  - Micro-frontends enable scaling individual parts of the application independently based on the traffic and load they handle.
- Technology Agnostic:
  - Different micro-frontends can be built using different technologies, enabling teams to choose the best tools and frameworks for their specific needs.

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

Benefits of MFE

- Better Team Collaboration:
  - Teams can work autonomously on their respective micro-frontends, reducing dependencies and improving productivity.
- Reduced Complexity:
  - By breaking down a large monolithic frontend into smaller pieces, the overall complexity of the application is reduced, making it easier to maintain and enhance.
- Enhanced User Experience:
  - Allows for incremental updates and A/B testing, leading to a more responsive and user-centric application.

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

Technologies to Create Micro-Frontends (MFE)

- Webpack Module Federation:
  - Enables multiple independently built and deployed bundles to form a single application.
  - Facilitates sharing of code and dependencies between different micro-frontends.
- Single SPA (Single Single Page Application):
  - A framework for building micro-frontends, allowing multiple frameworks to coexist in a single app.
  - Provides lifecycle hooks to load and unload micro-frontends.
- Piral
- Luigi

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

Webpack Module Federation

- Webpack Module Federation is a feature introduced in Webpack 5
- Enables the sharing of modules between separate builds.
- Allows multiple independently built and deployed applications to dynamically share code and dependencies, forming a single cohesive application.

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

Key Features of Module Federation

- Dynamic Remotes:
  - Allows applications to load remote modules dynamically at runtime, rather than statically at build time.
- Shared Modules:
  - Facilitates sharing of common dependencies (e.g., React, lodash) between different micro-frontends, reducing duplication and ensuring consistency.
- Independent Deployment:
  - Enables each micro-frontend to be developed, tested, and deployed independently, making the development process more agile.

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

Key Features of Module Federation

- Version Management:
  - Handles different versions of shared libraries gracefully, allowing for compatibility between various parts of the application.
- Reduced Bundle Size:
  - By sharing common dependencies, Webpack Module Federation helps in reducing the overall bundle size, leading to faster load times.
- Seamless Integration:
  - Integrates seamlessly with existing Webpack configurations, making it easy to adopt without significant changes to the build setup.

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

remoteEntry.js

- The file acts as the manifest that allows a host application to dynamically load and use modules exposed by a remote micro-frontend (MFE).
- Key Features
  - Metadata Provider: Contains metadata about the remote micro-frontend, including the modules it exposes, their locations, and the dependencies required.
  - Dynamic Loading: When the host application requests a module from the remote, the remoteEntry.js file provides the necessary information to locate and load that module at runtime.
  - Shared Dependencies: Includes information about shared dependencies, ensuring that both the host and remote use the same instance of shared libraries (e.g., React)

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

remoteEntry.js

- When you build your remote MFE, Webpack generates the remoteEntry.js file based on the configuration specified in the ModuleFederationPlugin.
- The file contains references to all the exposed modules and the shared dependencies declared in the remote's Webpack configuration.
- When the host application starts, it fetches the remoteEntry.js file from the remote's server.
- This can be done dynamically at runtime, allowing the host to use the latest version of the remote without needing to rebuild.
- The host uses the metadata in remoteEntry.js to resolve and load the exposed modules dynamically.

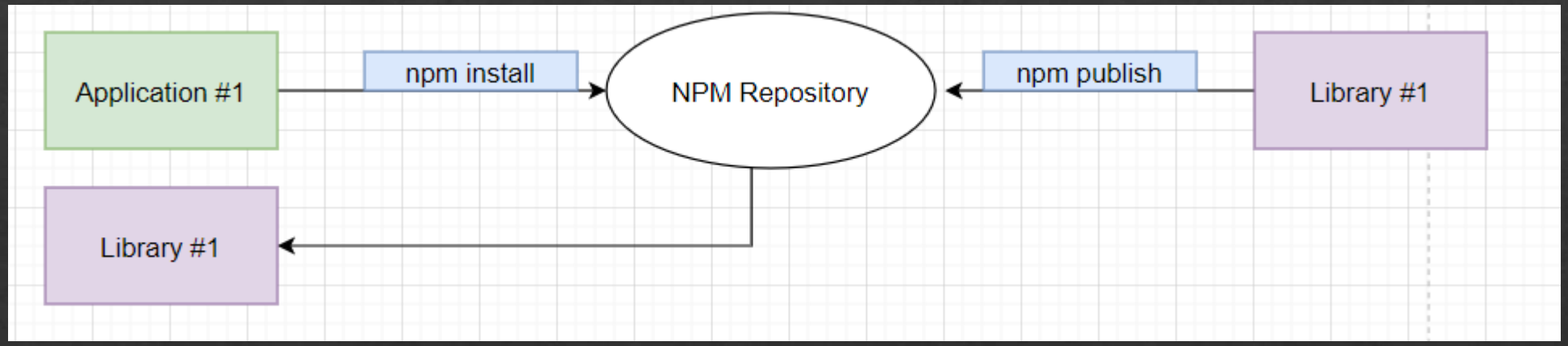
Presentaed by Anil Joseph(anil.jos@gmail.com)

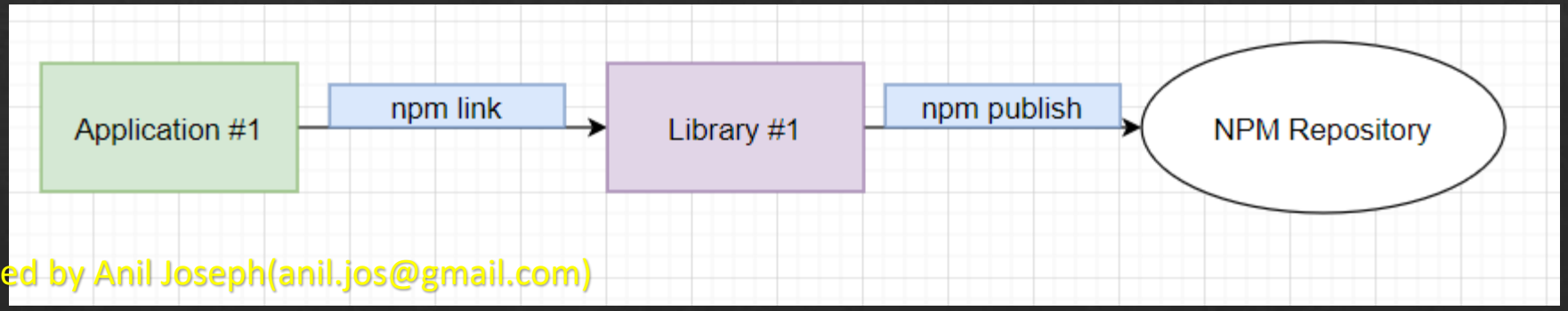
Design Patterns

- Higher Order Components
- Render Prop Pattern
- Provider
- Compound Component

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

React Library





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

American Disability Act(ADA) compliance

- Semantic HTML
  - Use <header>, <nav>, <main>, <section>, and <footer> for layout.
  - Use <button>, <input>, <label>, and <form> for interactive elements.
- Use ARIA attributes to enhance accessibility where native HTML elements fall short:
  - aria-label: Provides an accessible name for elements.
  - aria-hidden: Hides elements from screen readers.
  - role: Defines the role of an element.
- Keyboard Navigation
  - Use tabIndex to control the tab order.
  - Ensure focus states are visible and intuitive.
  - Manage focus using focus() and blur() methods when necessary.

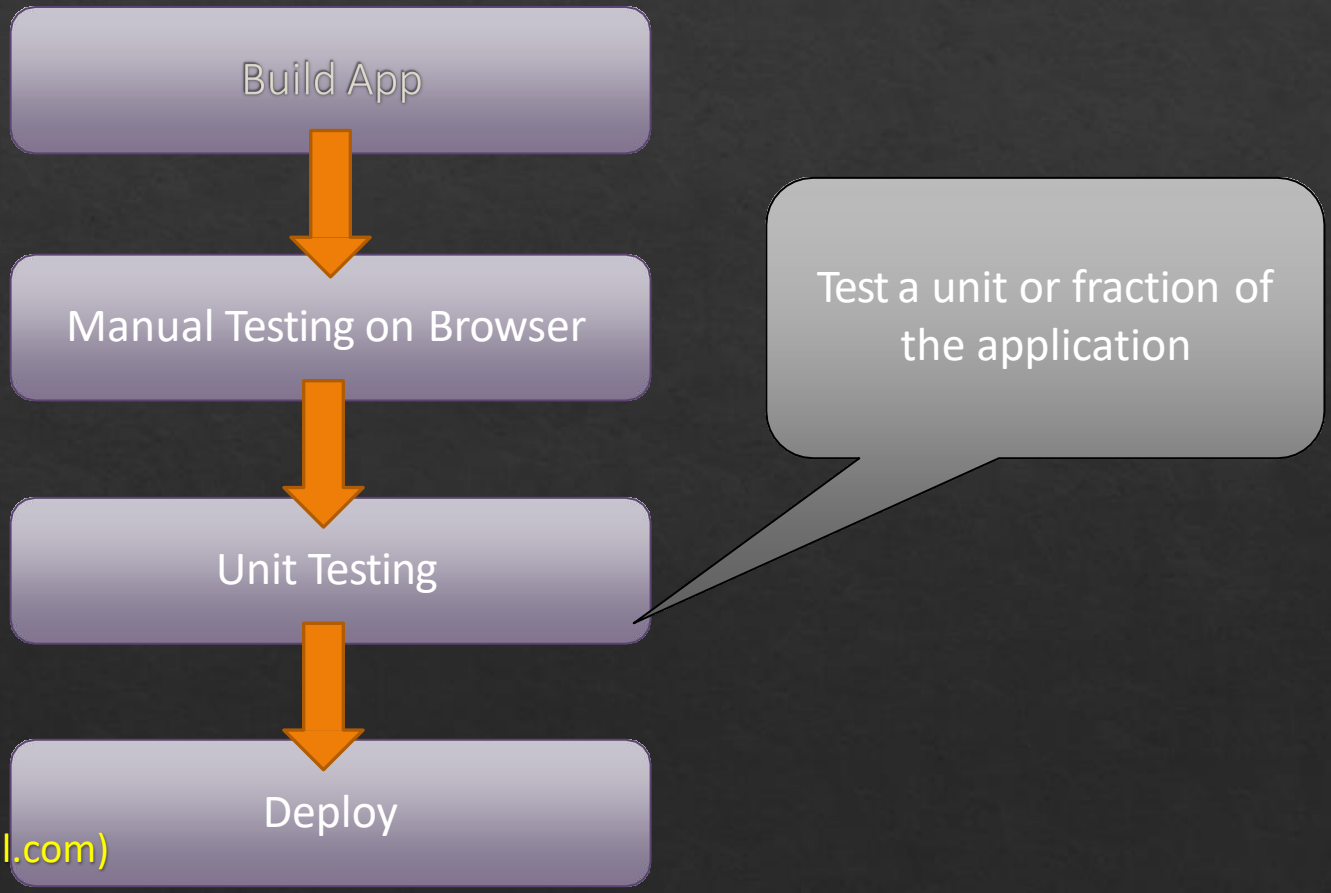
Presentaed by Anil Joseph(anil.jos@gmail.com)

American Disability Act(ADA) compliance

- Alt Text for Images
  - Provide meaningful alt attributes for images:
  -
- Form Accessibility
  - Use <label> for each input field.
  - Provide helpful error messages and guidance.
- Accessibility Libraries
  - React ARIA
- Testing with screen readers

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

Testing


Build App
Manual Testing on Browser
Unit Testing
Deploy
Test a unit or fraction of the application

Testing Tools

Testing API
&
Test Runner

Write the Unit Test.
Executes the Unit Tests.

Jest
(Built over Jasmine)

Test Utilities

Simulate the React App

react-test-renderer

enzyme

testing-library/react

Jest

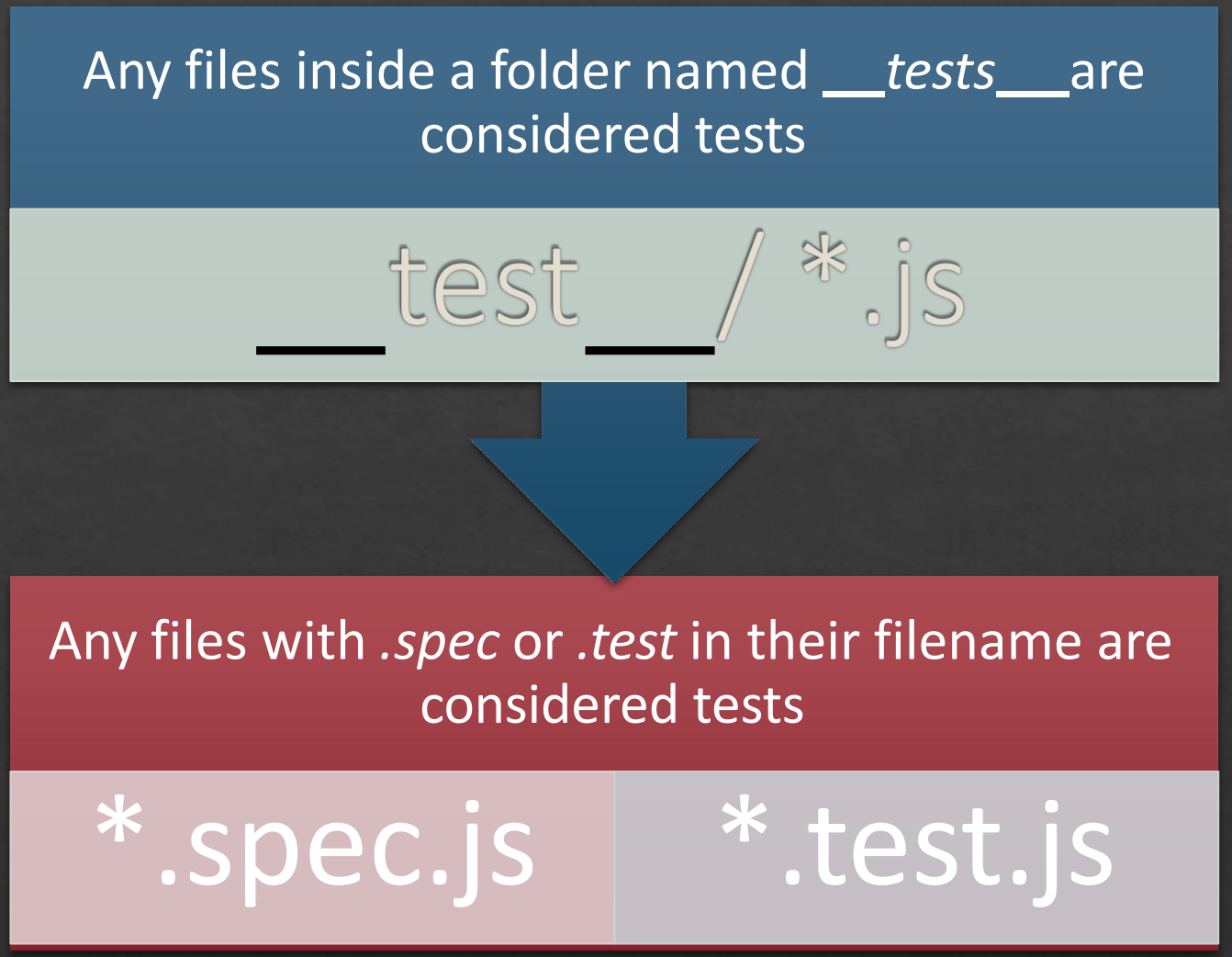
A testing library and test runner with handy features

Created by the members of the React team and the recommended tool for unit testing react

Built on top of Jasmine/Mocha

Additional features like mocking and snapshot testing

Jest: Identifying Test files



Jest Global Methods

it

- Method which you pass a function to, that function is executed as block of tests by the test runner.
- Alias name: test

describe

- An optional method for grouping any number of *it* or *test* statements
- Alias name: suite

Setup & Teardown Global functions

beforeEach	BeforeEach runs a block of code before each test
afterEach	Runs a block of code after each test
beforeAll	BeforeAll runs code just once, before the first test
afterAll	Runs a block of code after the last test)

What to Test?

Test Isolated Components

Test Conditional Outputs

Don’t Test Library Functions

Don’t Test Complex Connection

End to End Test

End-to-end testing is a methodology used to test whether the flow of an application is performing as designed from start to finish.

The purpose of carrying out end-to-end tests is to identify system dependencies and to ensure that the right information is passed between various system components and systems.s

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

E2E Tools

Cypress

Pupeeter

Selenium Webdriver

Nightwatch.js

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

Thank You

ANIL JOSEPH

anil.jos@gmail.com