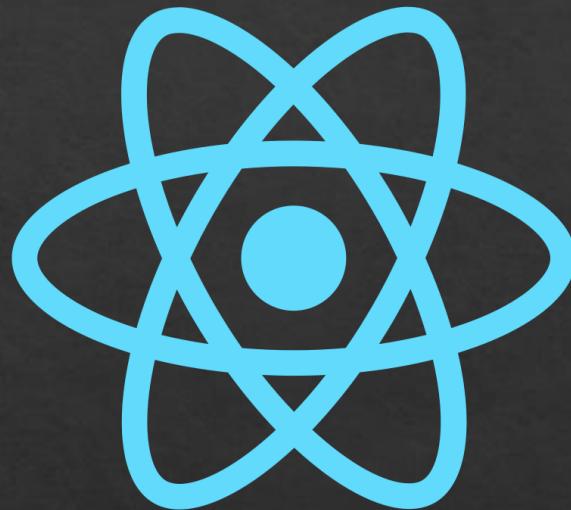


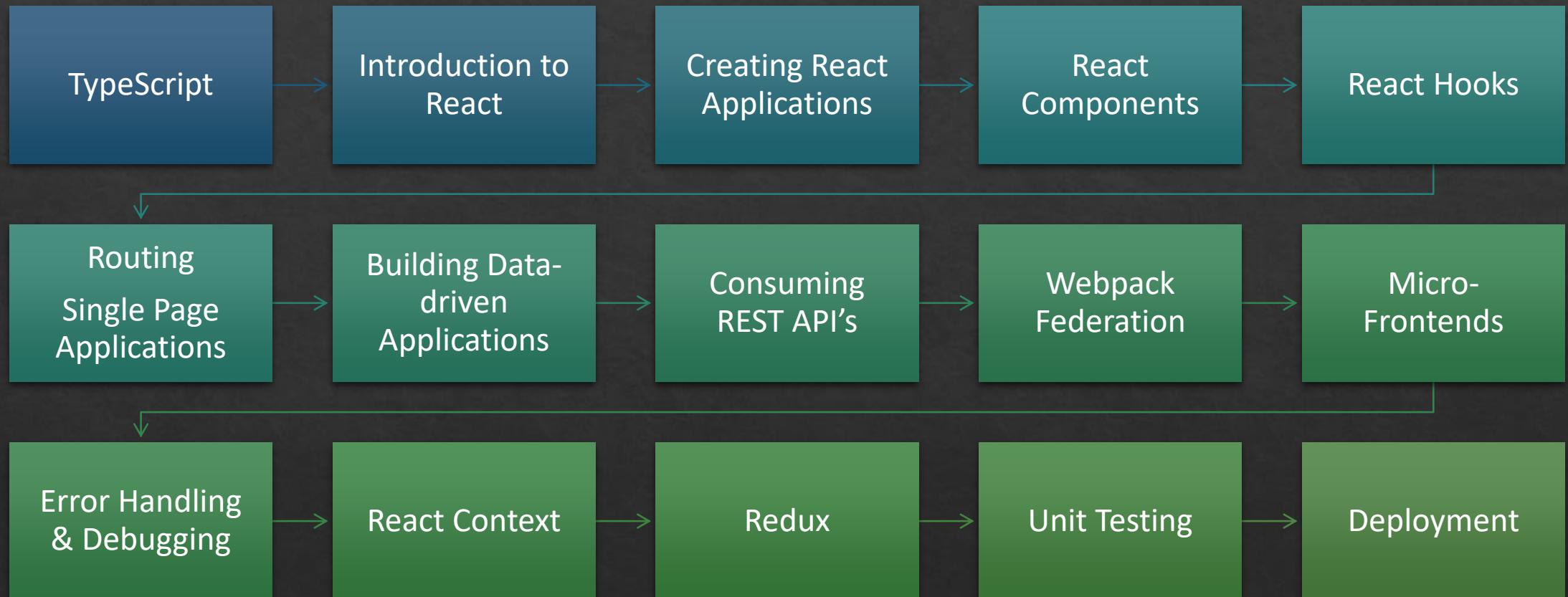
React



ANIL JOSEPH

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

Agenda



Presented 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+)

Software

Node.js & NPM

HTML, CSS, JavaScript,
TypeScript Editor
(Visual Studio Code)

Browsers(Chromium)

JavaScript

An interpreted language

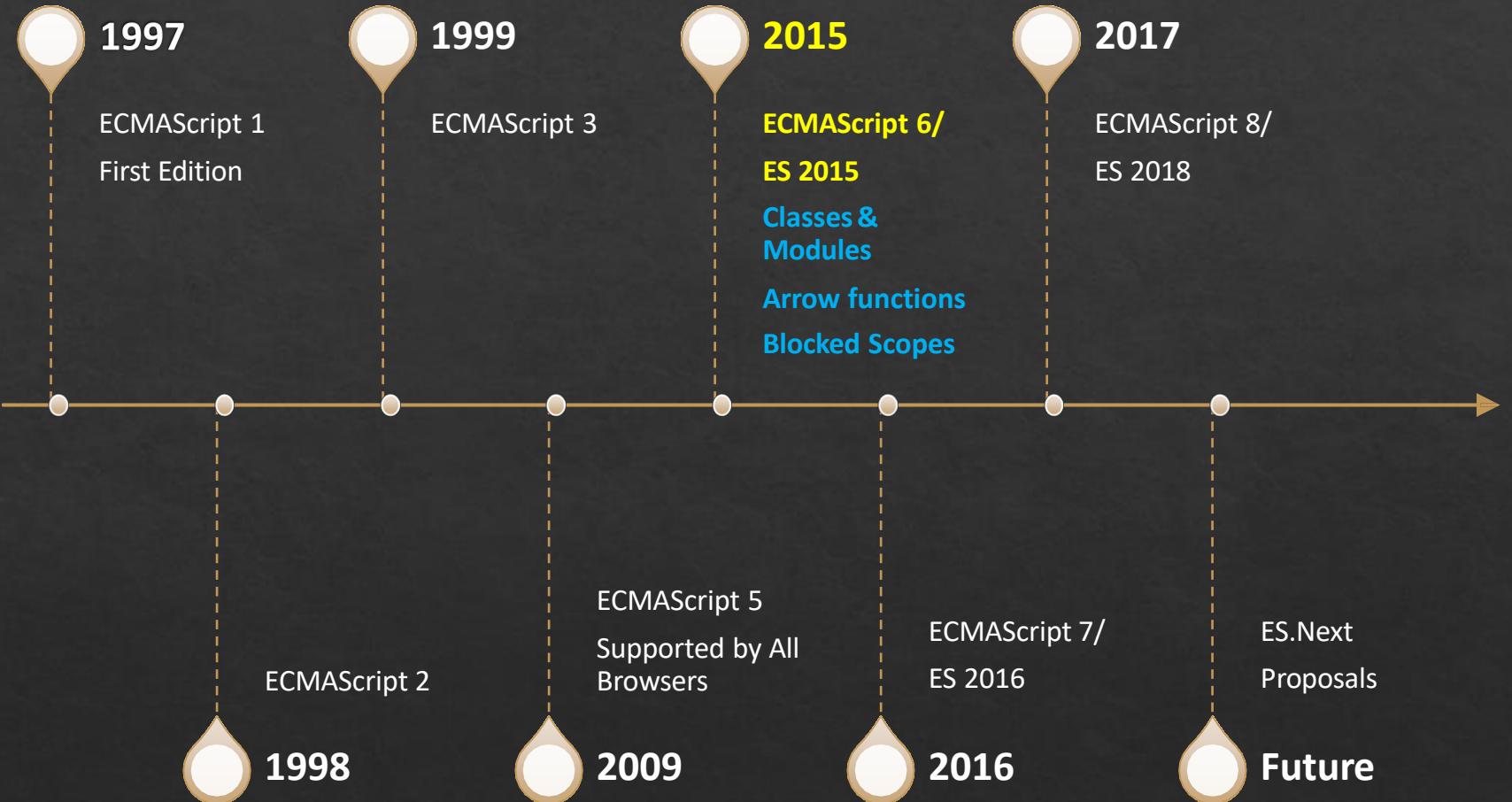
Dynamic

Object-oriented

Supports Functional style of programming

Available on the browsers and Node.js

ECMAScript Versions



Primitive Data Types

Number

- Floating Points, Decimals and Integer

String

- Sequence of characters

Boolean

- Logical: true/false

Undefined

- Not Initialized

Null

- Absence of an object value

Symbol

- A unique identifier

Non-Primitive Data Types

Object

- Collection of key-value pairs

Function

- A type of object that can be invoked(called)

Array

- A ordered collection of values

Date

- Represents Dates and Times

Regular
Expression

- Represents regular expressions.

Scopes

- ❖ Scope of a variable determines where it can be accessed and modified. There are several types of scope:
- ❖ Global
 - ❖ Variables declared outside of any function or block have global scope. They can be accessed from anywhere in the code.
- ❖ Function
 - ❖ Variables declared inside a function using var are scoped to that function. They cannot be accessed outside the function.
- ❖ Block(ES6)
 - ❖ Variables declared inside a block (i.e., a pair of curly braces {}) using let or const are scoped to that block. They cannot be accessed outside the block.

Scopes

- ❖ Lexical
 - ❖ JavaScript uses lexical scoping, which means that the scope of a variable is determined by its location within the source code, and nested functions have access to variables declared in their outer scope.
- ❖ Module(ES6)
 - ❖ When using ES6 modules, variables declared inside a module are scoped to that module and are not accessible outside it unless explicitly exported.

Hoisting

- ❖ Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase, before the code is executed.
 - ❖ This means that you can use variables and functions before they are declared in the code,
 - ❖ Depending on whether you use var, let, const, or function declarations, the behavior changes.
-
- ❖ Variables declared with var are hoisted to the top of their function or global scope and are initialized with undefined.
 - ❖ Variables declared with let and const are hoisted to the top of their block scope, but are not initialized.

Functions

- ❖ Functions are one of the fundamental building blocks in JavaScript.
- ❖ They allow you to encapsulate code into reusable blocks that can be called and executed from different parts of your program.
- ❖ Functions can take input, process it, and return output.

Functions

- ❖ Function declaration
 - ❖ A function declaration defines a named function. It is hoisted, meaning it can be called before it is defined in the code.
- ❖ Function Expressions
 - ❖ A function expression defines a function inside an expression and can be named or anonymous. It is not hoisted, so it cannot be called before it is defined.
- ❖ Arrow functions(ES6)
 - ❖ Arrow functions are a shorter syntax for function expressions and do not have their own this, arguments keywords. They are always anonymous.

Functions as objects

- ❖ First-Class Objects:
 - ❖ Functions can be assigned to variables, passed as arguments, and returned from other functions.
- ❖ Properties and Methods:
 - ❖ Functions can have properties and methods, such as call, apply, and bind.
- ❖ Higher-Order Functions:
 - ❖ Functions that take other functions as arguments or return functions.
- ❖ Function Properties:
 - ❖ Functions have properties like length and name.
- ❖ Constructors:
 - ❖ Functions can act as constructors when used with the new keyword.

“this” keyword

- ❖ The “**this**” keyword in JavaScript behaves differently depending on the context in which it is used.
- ❖ It essentially refers to the object that is executing the current function.
- ❖ Understanding how this works in different contexts is crucial for writing effective JavaScript code.

Contexts of “this”

- ❖ Global Context: this refers to the global object (window in browsers).
- ❖ Function Context: this refers to the global object in non-strict mode and undefined in strict mode.
- ❖ Method Context: this refers to the object the method belongs to.
- ❖ Constructor Context: this refers to the newly created instance.
- ❖ Arrow Function Context: this is inherited from the surrounding lexical context.
- ❖ Event Handler Context: this refers to the element that received the event.
- ❖ Explicit Setting: Use call, apply, or bind to explicitly set this.

Closures

- ❖ A closure is a fundamental concept in JavaScript that allows functions to "remember" the environment in which they were created.
- ❖ Closures enable functions to access variables from an outer function even after the outer function has returned.
- ❖ This is possible because the inner function maintains a reference to its lexical environment.
- ❖ **A function that retains access to its lexical scope, even when that function is executed outside its lexical scope.**

Closure

```
function outerFunction() {  
    let outerVariable = 'I am outside!';  
  
    function innerFunction() {  
        console.log(outerVariable); // Accesses outerVariable from the outer scope  
    }  
  
    return innerFunction;  
}  
  
const closure = outerFunction();  
closure(); // Output: I am outside!
```

Objects

- ❖ Objects are fundamental building blocks for representing real-world entities, data structures, and more complex constructs.
- ❖ An object is a collection of key-value pairs where each key is a string (also called a property) and each value can be any data type, including another object, function, array, etc.

Creating Objects

- ❖ Object Literals:
 - ❖ Define objects directly with {}.
- ❖ Constructor Functions:
 - ❖ Use functions with new to create objects.
- ❖ ES6 Classes:
 - ❖ Provide a clear syntax for creating and managing objects.
- ❖ Object.create Method:
 - ❖ Create objects with a specific prototype.

Execution Context

Execution context (EC) is defined as the environment in which JavaScript code is executed.

Execution context in JavaScript are of two types.

- **Global execution context**
- **Functional execution context**

Global Execution Context

Create Phase

Global Object, this
Scope: variables and functions
Scope Chain
Code

Execute Phase

Executes Code

Execution Context

```
function foo(){  
    var x = 20;  
    bar();  
}
```

```
function bar(){  
    var x = 30  
}
```

```
var x = 10;  
foo();
```

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

bar Execution Context(Create and Execute)

x = 30

foo Execution Context(Create and Execute)

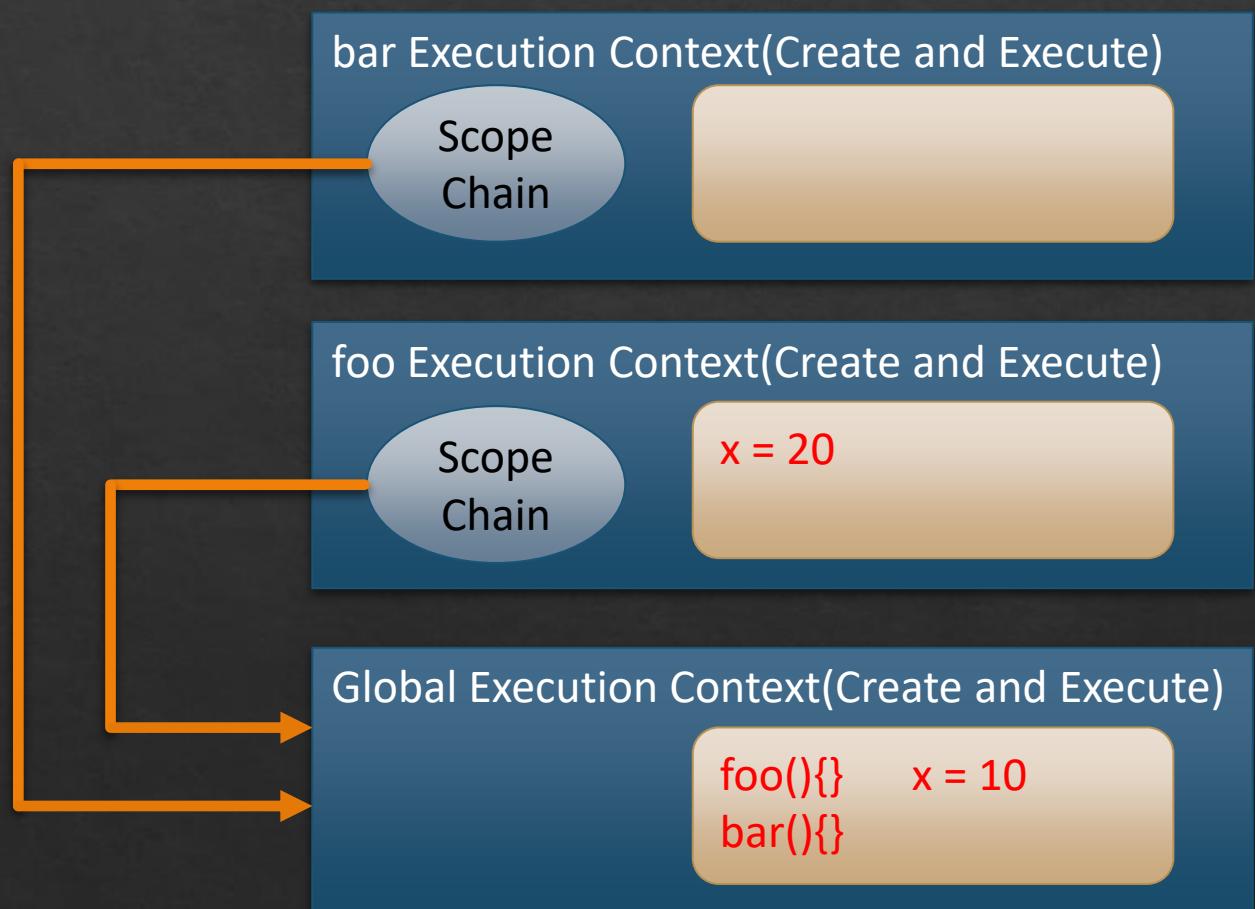
x = 20

Global Execution Context(Create and Execute)

foo(){} x = 10
bar(){}
Scope

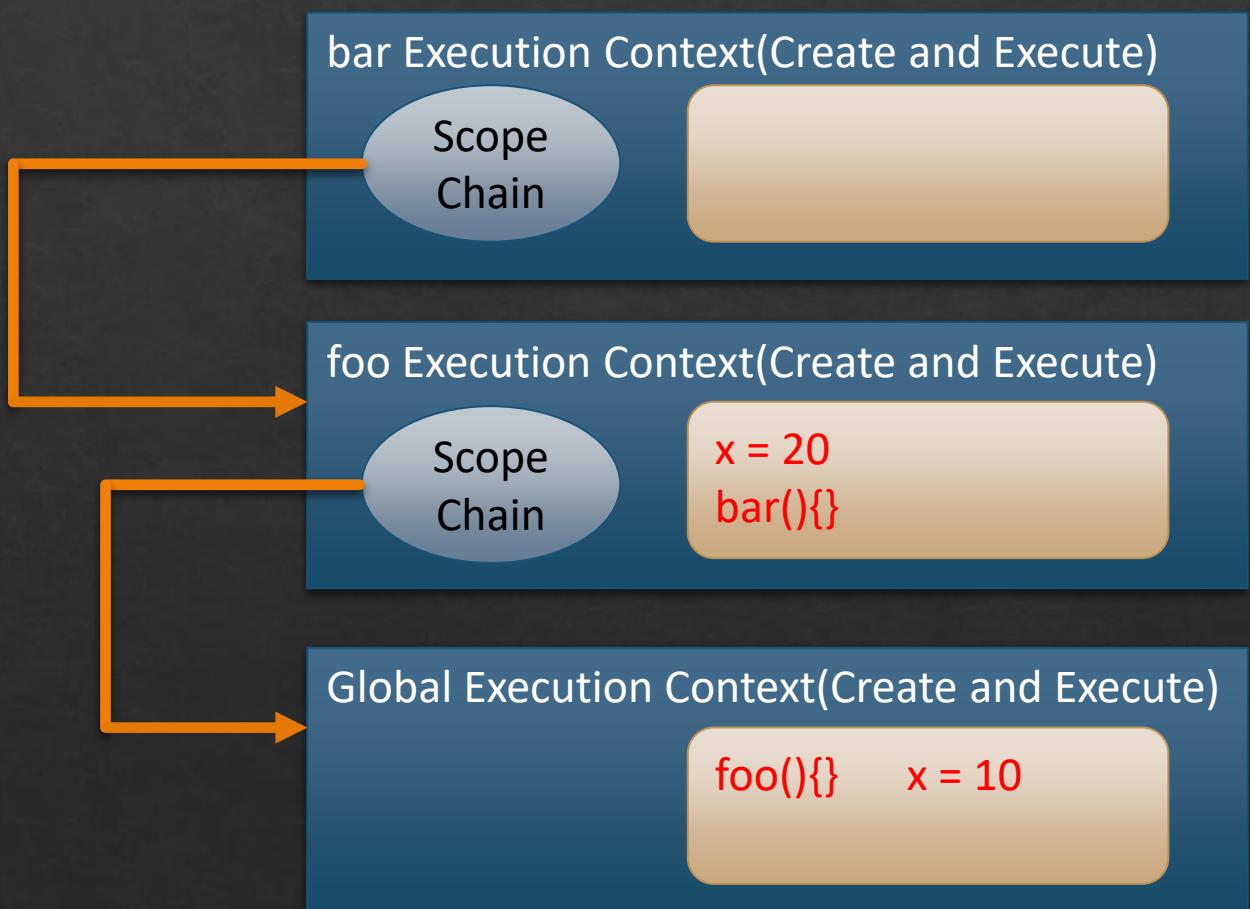
Execution Context

```
function foo(){  
    var x = 20;  
    bar();  
}  
  
function bar(){  
    console.log(x);  
}  
  
var x = 10;  
foo();
```



Execution Context

```
function foo(){  
    function bar(){  
        console.log(x);  
    }  
  
    var x = 20;  
    bar();  
}  
  
var x = 10;  
foo();
```



window object

- ❖ The **window** object represents the browser's window or tab containing a web page.
- ❖ Global Scope: Variables and functions declared globally become properties and methods of the window object.
- ❖ Timing Events: Methods for scheduling code execution after a delay or at regular intervals.
- ❖ Dialog Boxes: Methods for displaying alert, confirm, and prompt dialogs.
- ❖ Navigation: Properties and methods for interacting with the browser's history and location.
- ❖ Event Handling: Properties and methods for handling events.

document object

- ❖ The ***document*** object represents the HTML document loaded in the browser.
- ❖ It is part of the DOM and provides methods and properties for accessing and manipulating the content and structure of the web page.
- ❖ Accessing Elements: Methods for selecting elements in the DOM.
- ❖ Creating Elements: Methods for creating new elements.
- ❖ Event Handling: Methods for attaching event listeners to elements.
- ❖ Document Information: Properties for retrieving information about the document.
- ❖ Form Handling: Methods for working with forms and form elements.

Document: Access elements

```
var elementById = document.getElementById('myElement');

var elementsByClassName = document.getElementsByClassName('myClass');

var elementsByTagName = document.getElementsByTagName('p');

var elementByQuerySelector = document.querySelector('.myClass');

var elementsByQuerySelectorAll = document.querySelectorAll('p.myClass');
```

ES6 New Syntax

Block Scoping

Arrow
Functions

Rest and
Spread

Object Literals
Extensions

Template
Literals

for .. of loop

Destructuring

Modules

Block Scoped Constructs

- ❖ **Scope** determines the accessibility (visibility) of variables.
- ❖ ES5 supports two scopes
 - ❖ Global
 - ❖ Functional
- ❖ ES 6 introduces the block scoped constructs
 - ❖ **let**
 - ❖ **const**

Arrow Function

Represents a function expression

An arrow function 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

Spread and REST Operators

- ❖ **Spread syntax** allows
 - ❖ An iterable to be expanded in places where zero or more arguments are expected
 - ❖ Example: An array to be converted to a list of arguments of a method.
- ❖ **Rest parameter** syntax allows
 - ❖ To represent an indefinite number of arguments as an array.

for... of

New for.. Of
loop

Iterate over
iteratables

Example

- `for(var item of names){ }`

Template Literals

Allows
interpolation on a
String template.

Defined inside the
back ticks (` `)

Example

- `Number: \${number}`

Allows
Expressions

Destructuring

Destructuring is a convenient way of extracting multiple values from data stored in (possibly nested) objects and Arrays.

Example:

- var arr = [2300, 3400, 6000]
- var [a, b, c] = arr;

Example

- var employee = {id:1, name: “Anil”, location:“Mumbai”}
- var {id, name, location} = employee;

ES6 Classes

- ❖ ES 6 supports class based object-oriented programming
- ❖ Classes in ES6
 - ❖ Constructor
 - ❖ Properties
 - ❖ Methods
 - ❖ Static methods
 - ❖ Inheritance
- ❖ Classes are just syntactic sugar of constructor functions.

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

Modules

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

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

two.js

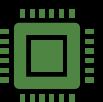
Node.js



A server-side JavaScript platform



Created by Ryan Dahl in 2009



Provides an easy way to build scalable network applications



Built on top of Google V8 JavaScript Engine and libuv

NPM(Node Package Manager)

- ❖ NPM is a package manager for the JavaScript programming language.
- ❖ Allows users to consume and distribute JavaScript modules that are available on the registry
- ❖ The default package manager for Node.
- ❖ Comprises of
 - ❖ Command line client, also called npm
 - ❖ An online database called the npm registry.
 - ❖ Public
 - ❖ Paid-for private packages
 - ❖ NPM website
- ❖ The package manager and the registry are managed by npm, Inc.

NPM Packages

- ❖ Packages are reusable code published in the npm registry.
- ❖ A directory with one or more files a metadata file called package.json.
- ❖ A package is a building block that solves a specific problem.
- ❖ An application generally depends on many packages.
- ❖ Packages can be used on
 - ❖ Server side. Example: Express, Request
 - ❖ Client Side. Example Angular, React
 - ❖ Command based. Typescript, Angular CLI, JSLint

NPM Commands

- ❖ NPM is installed with Node
- ❖ Check the version
 - ❖ `npm -v`
- ❖ Update npm
 - ❖ `npm install npm@latest -g`
- ❖ Find the path to npm's directory:
 - ❖ `npm config get prefix`
- ❖ Help
 - ❖ `npm -h`
 - ❖ `npm [command] -h`
 - ❖ `npm help [command]`
 - ❖ `npm help-search [key]`

package.json

- ❖ A metadata file for the project
- ❖ Used to track dependencies
- ❖ Create Scripts
- ❖ Command to create package.json
 - ❖ npm init
- ❖ Setting defaults
 - ❖ npm set init-author-name 'Anil Joseph'
 - ❖ npm get init-author-name
 - ❖ npm config delete init-author-name

Installing Packages

- ❖ Install to a project
 - ❖ npm install express
- ❖ Install and save to dependencies
 - ❖ npm install angular --save
- ❖ Install and save to development dependencies
 - ❖ npm install express --save-dev
- ❖ Install globally
 - ❖ npm install gulp -g

Install Packages with version

- ❖ Specific version
 - ❖ `npm install underscore@1.8.2`
- ❖ Latest Version
 - ❖ `npm install underscore@1.8.x`
 - ❖ `npm install underscore@1.8`
 - ❖ `npm install underscore@1.8.2`
 - ❖ `npm install underscore@1.x`
 - ❖ `npm install underscore@1`
 - ❖ `npm install underscore`
- ❖ Other Options
 - ❖ `npm install underscore@">=1.4.x < 1.6.x"`

Listing & Removing Package

- ❖ npm list
 - ❖ List all packages
 - ❖ Shows all dependencies of packages
- ❖ npm list --depth 1
 - ❖ List packages show dependency depth to 1 level
- ❖ npm list --global true
 - ❖ List global packages
- ❖ npm list --global true --depth 0
 - ❖ List global packages show dependency depth to 0 level

Listing & Removing Package

- ❖ `npm list --dev true`
 - ❖ List development dependiecies
- ❖ `npm list --prod true`
 - ❖ List Production dependencies
- ❖ `npm ls`
 - ❖ ls as shortcut
- ❖ `npm uninstall underscore/ npm rm underscore/ npm un underscore`
 - ❖ Removes/Uninstall a package
- ❖ `npm uninstall underscore --save`
 - ❖ Uninstalls and updates the package.json
- ❖ `npm uninstall underscore -g`
 - ❖ Unstall global package

NPM Package vs Module

- ❖ Module
 - ❖ Single JavaScript file with some reasonable functionality
- ❖ Package
 - ❖ A directory with one or more modules
 - ❖ package.json file contains metadata about the package

Challenges with JavaScript

- ❖ Inconsistent browser implementations to access the DOM, XML parsing etc.
- ❖ JavaScript supports ***global variables***, which could cause conflicts when using multiple libraries.
- ❖ Creations of ***namespaces*** is not straight forward.
- ❖ Implementation of ***object-oriented code*** differs from conventional object-oriented languages.
- ❖ Some language features are complex to understand

TypeScript

TypeScript is programming language developed and maintained by Microsoft.

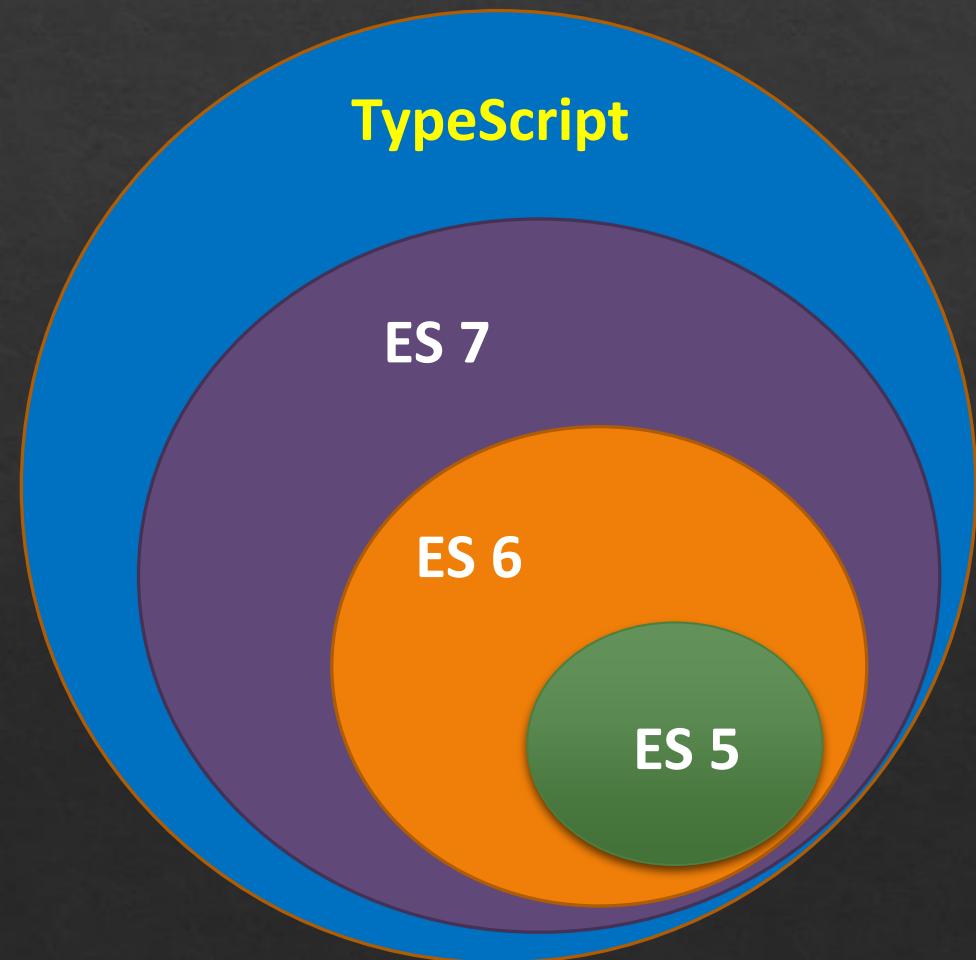
TypeScript is a typed superset of JavaScript.

Transcompiles to JavaScript.

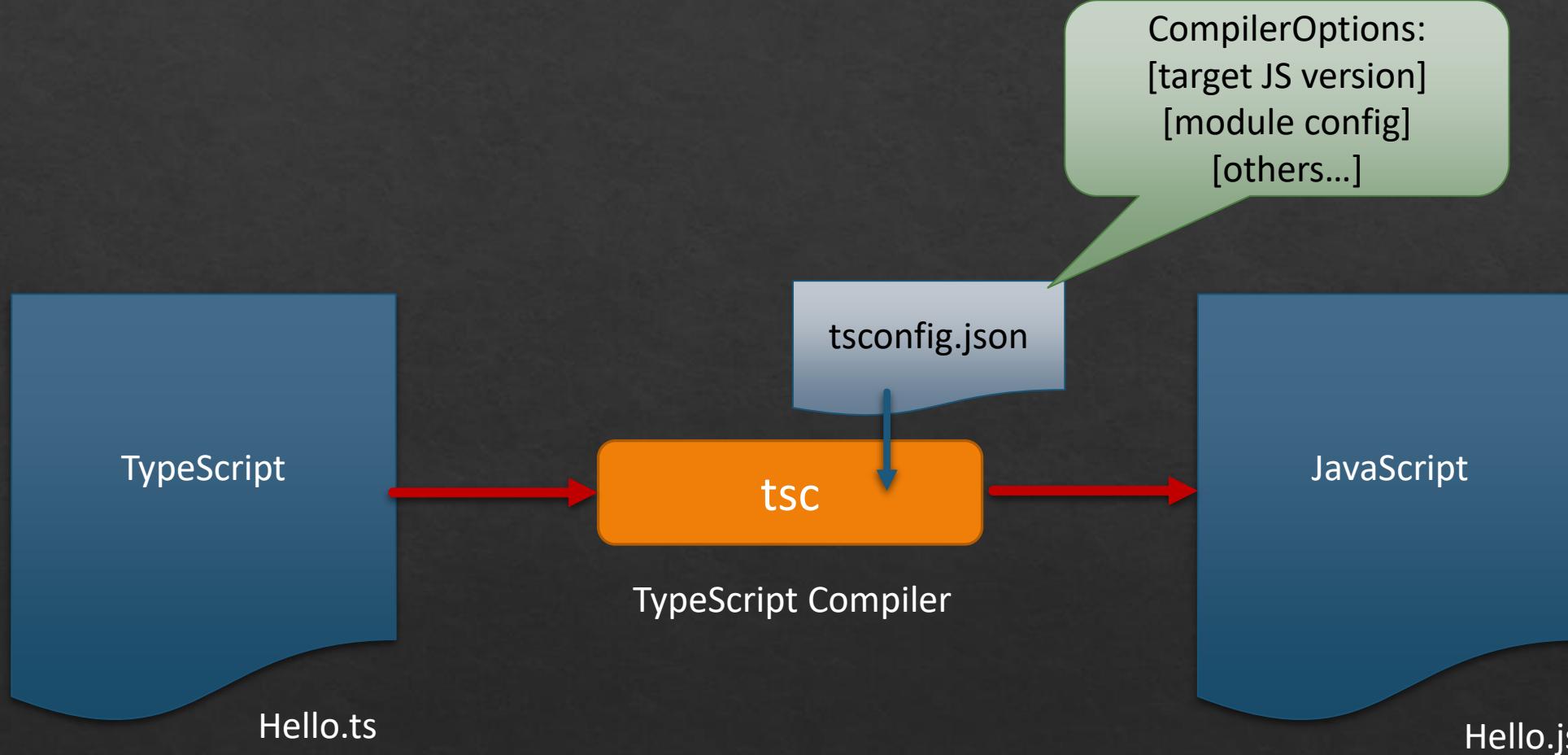
Designed for development of large applications.

Open Source.

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



TypeScript



TypeScript Features

Type Annotations

Compile-Time Type Checking

Type Inference

Interfaces

Classes and Inheritance

Namespaces and Modules

Generics

Decorators

Arrow Functions

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

TypeScript Installation

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

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];

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

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

- 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.

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.

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

Arrow Functions

Represents a function expression

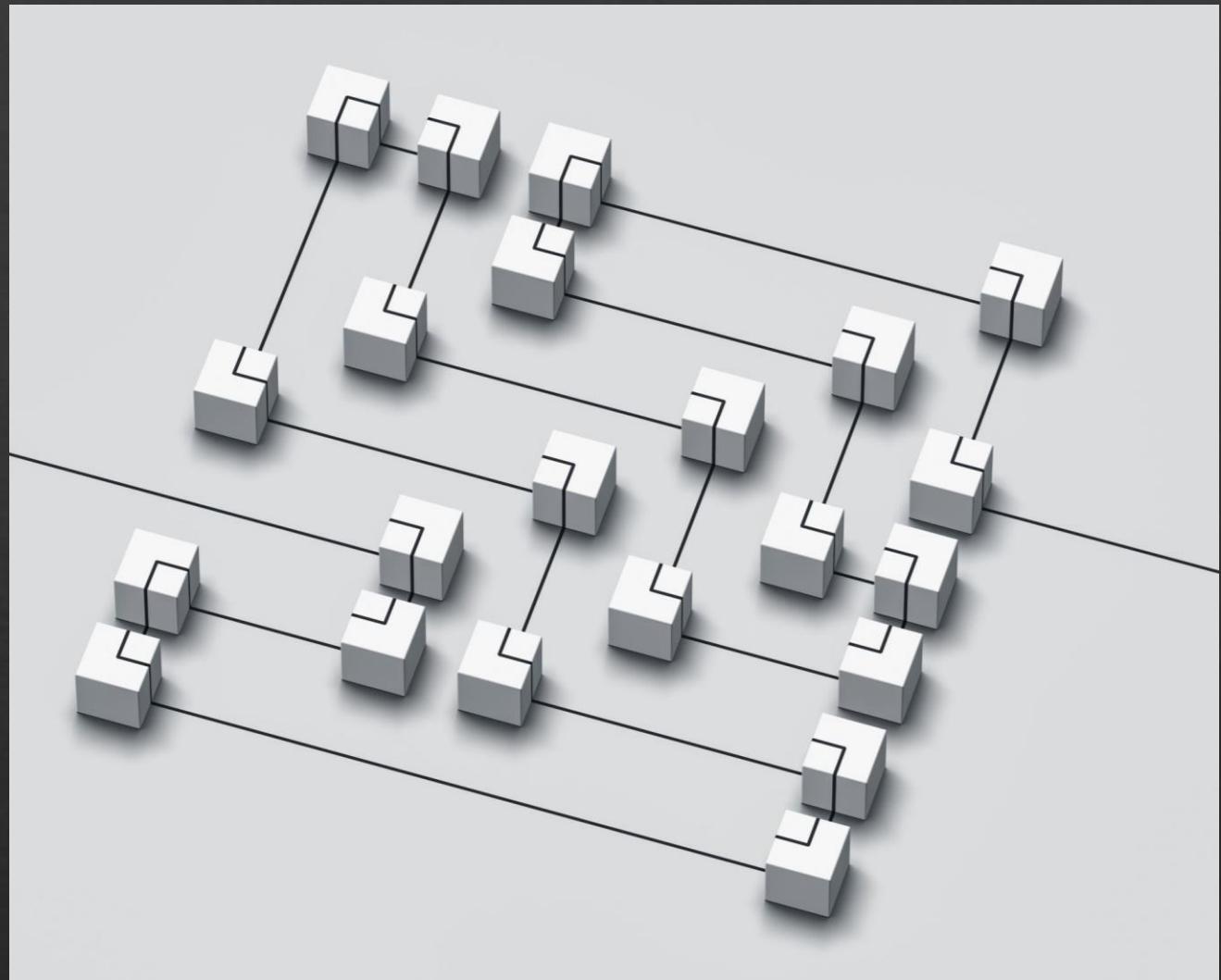
An arrow function expression has a shorter syntax than a function expression.

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

Used widely for asynchronous and functional programming

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



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

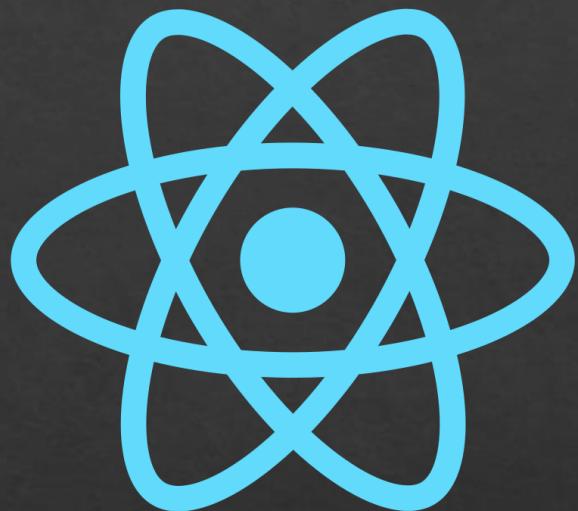
Modules

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

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

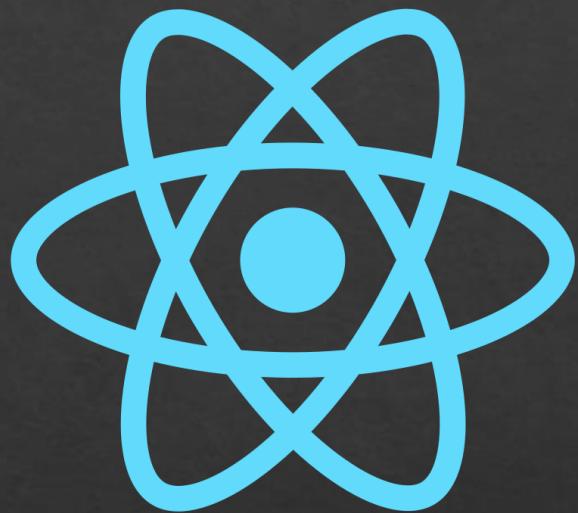
two.js

What is React?



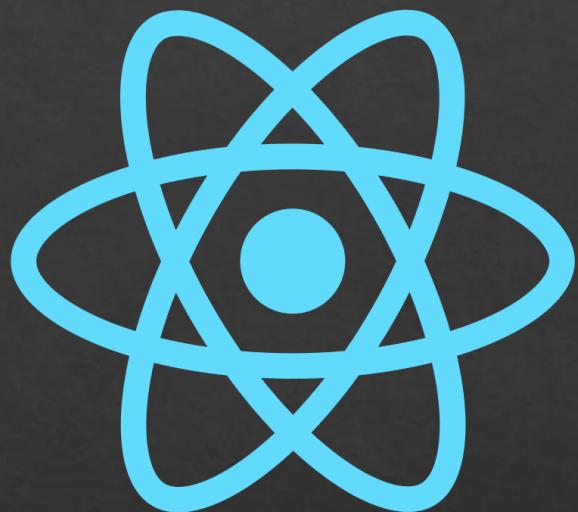
- ❖ A JavaScript library to build *User Interfaces*.
- ❖ Building applications for the *browser*.
- ❖ React allows to create custom HTML elements called *components*.
- ❖ Components are used to build *complex User Interface*
- ❖ Reacts promotes writing *maintainable, manageable and reusable code*.

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

History

React was created by **Jordan Walke**, a software engineer at Facebook.

It was first deployed on **Facebook's** newsfeed in 2011

Later on **Instagram** in 2012.

It was open-sourced in 2013.

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)

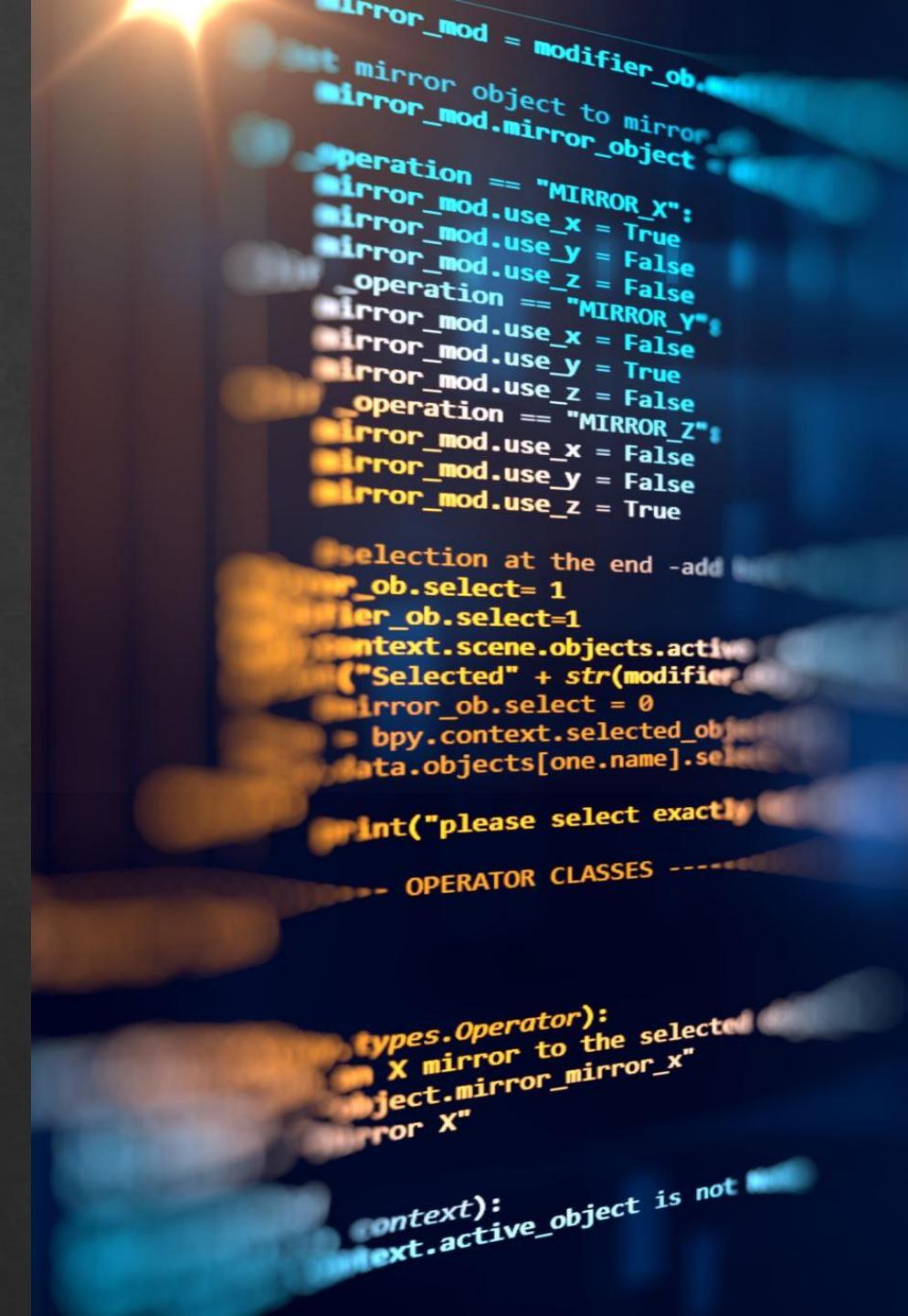
Creating a React Project

- ❖ 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

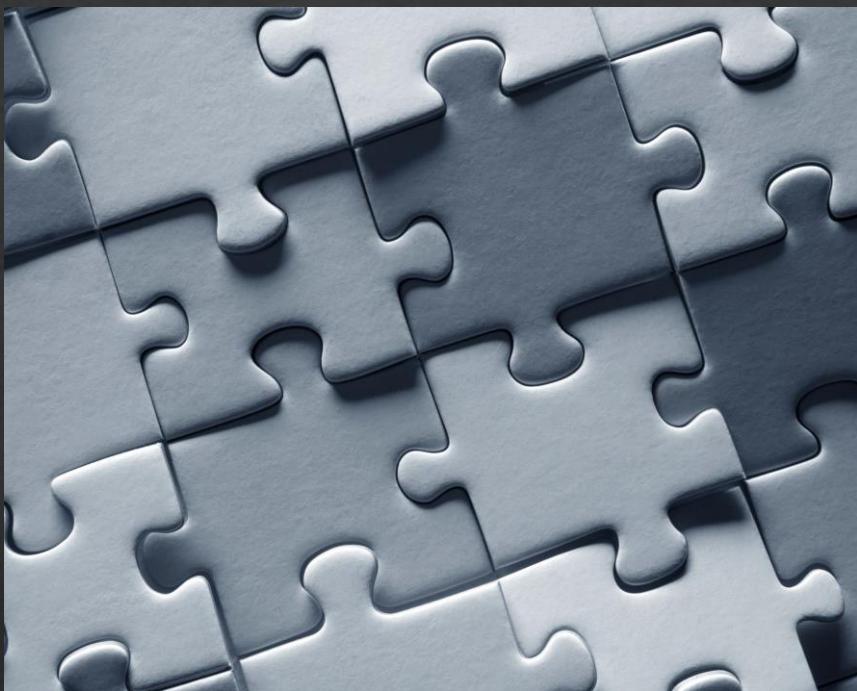
Tool Chains

- ❖ 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

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**

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**

Create Project-Typescript

1

Install NodeJs

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

2

Create React Application

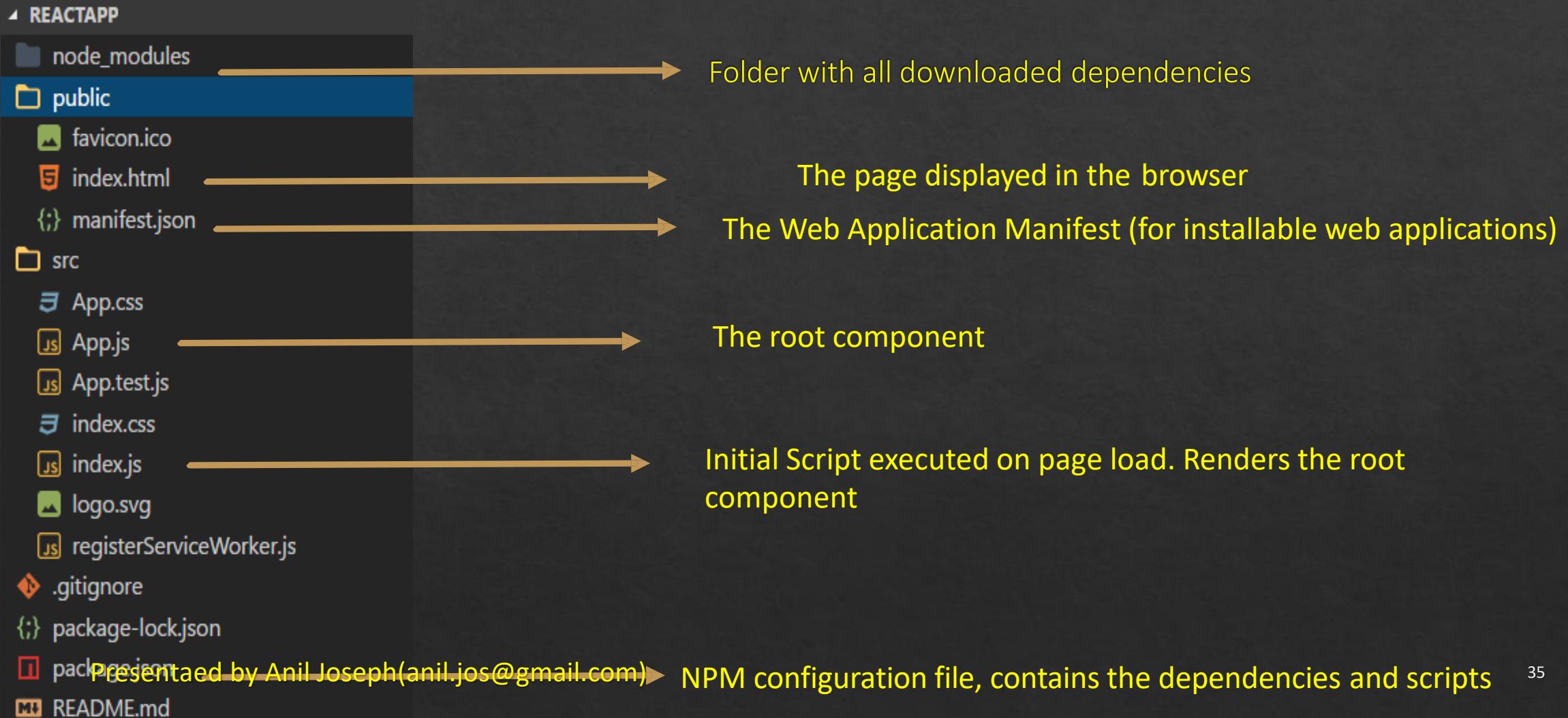
- `npx create-react-app the-react-app --template typescript`

3

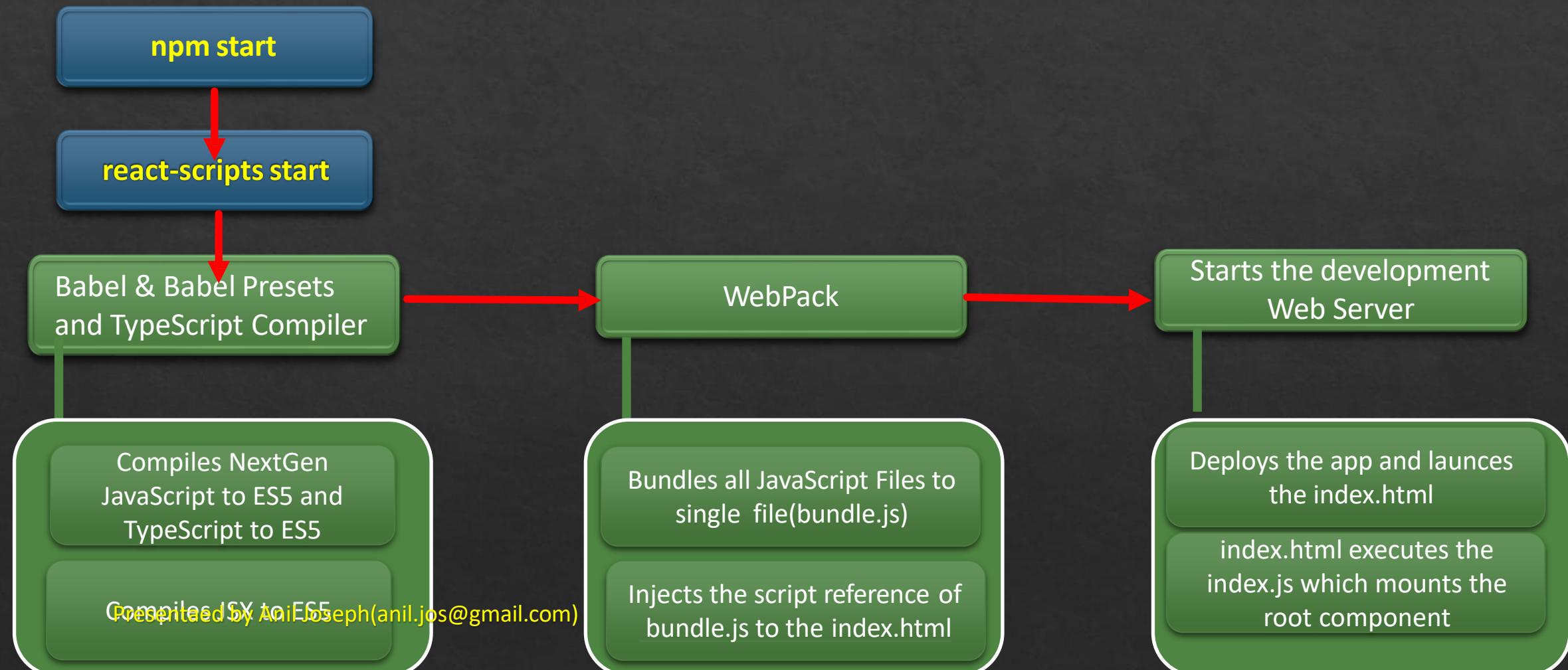
Start the Application

- `cd the-react-app`
- `npm start`

Project Structure

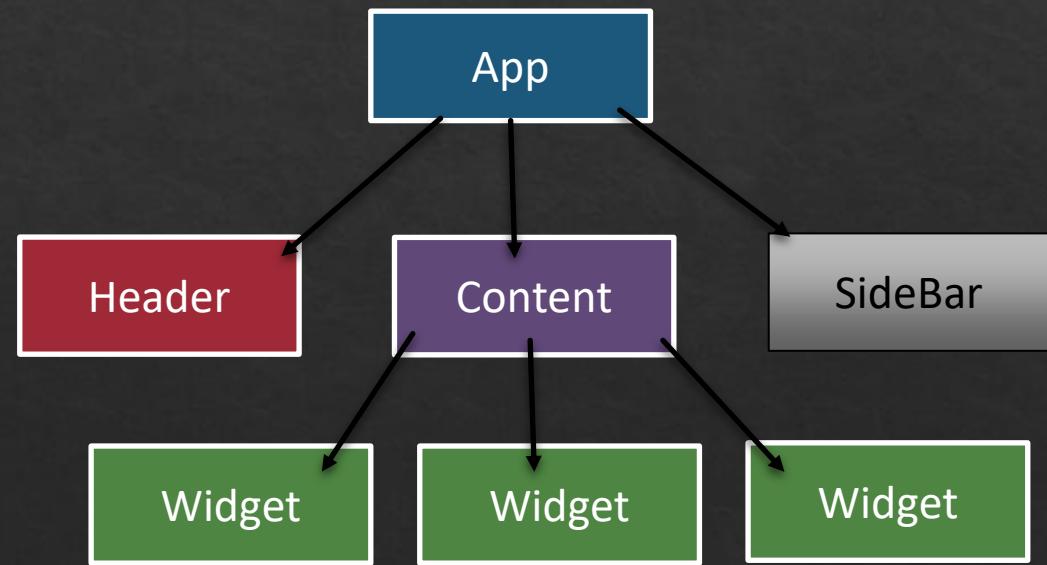
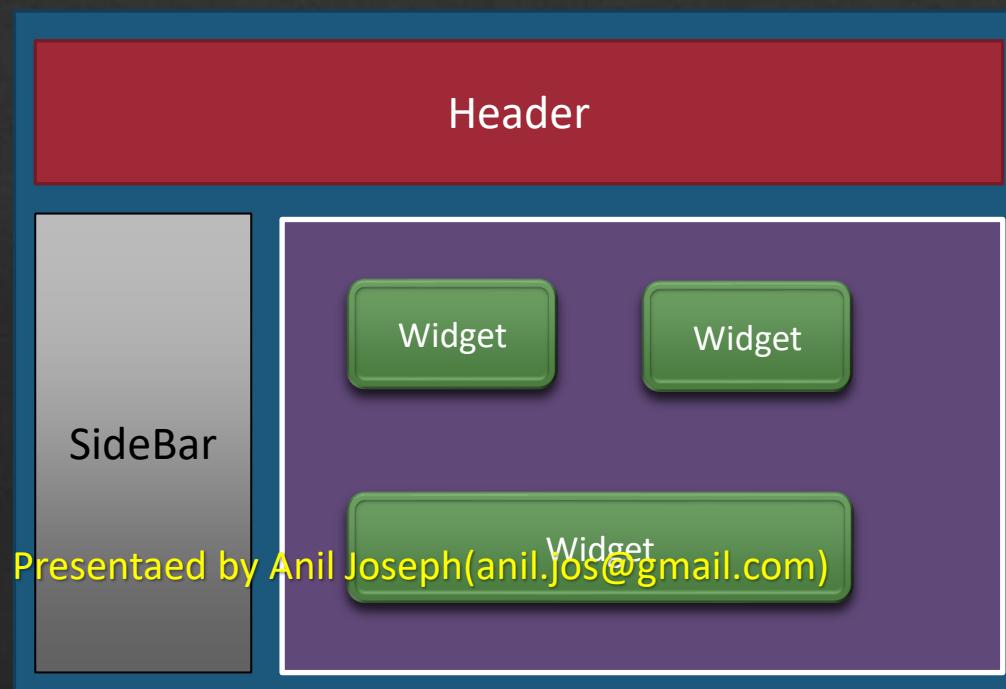


Project Execution



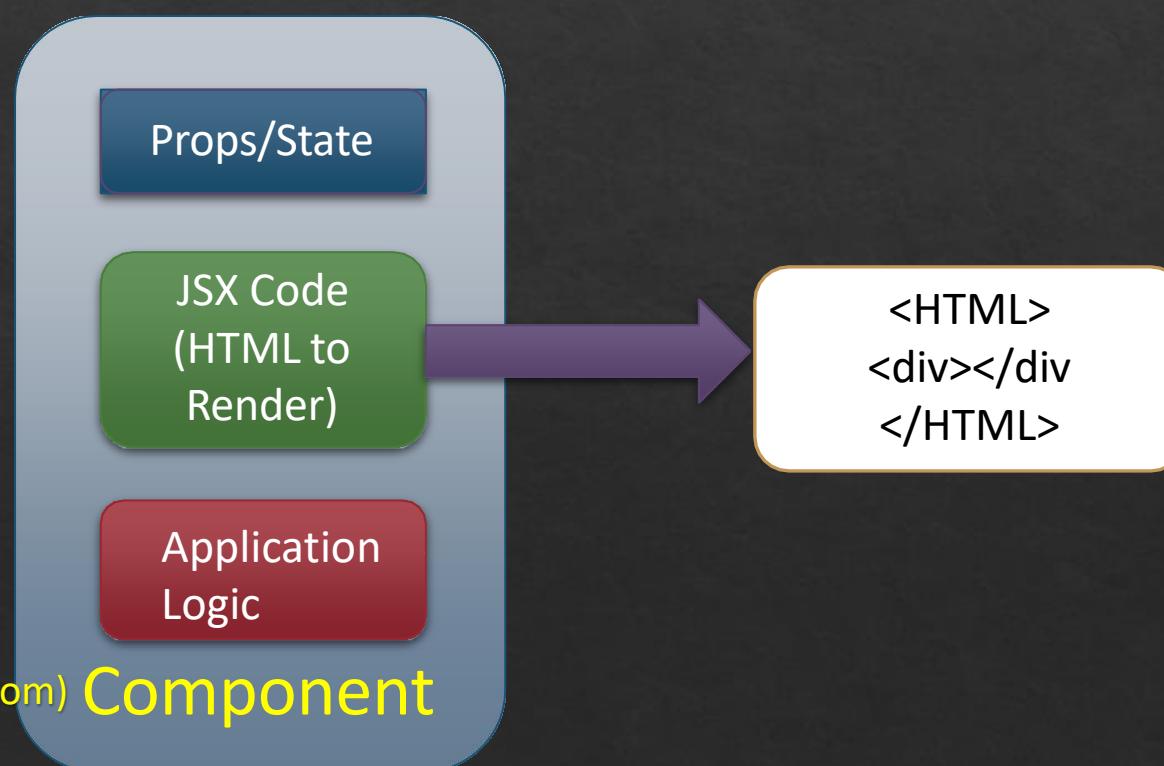
React Components

- ❖ 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



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

Types of Components

Functional

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

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 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() }`

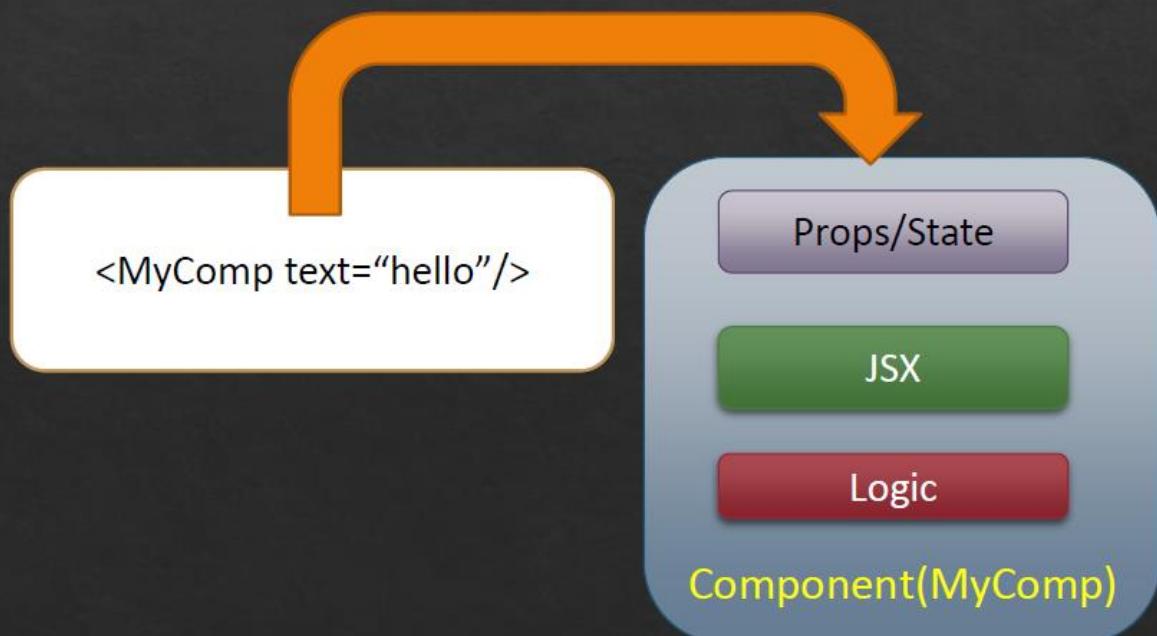
Components: Properties(props)

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.



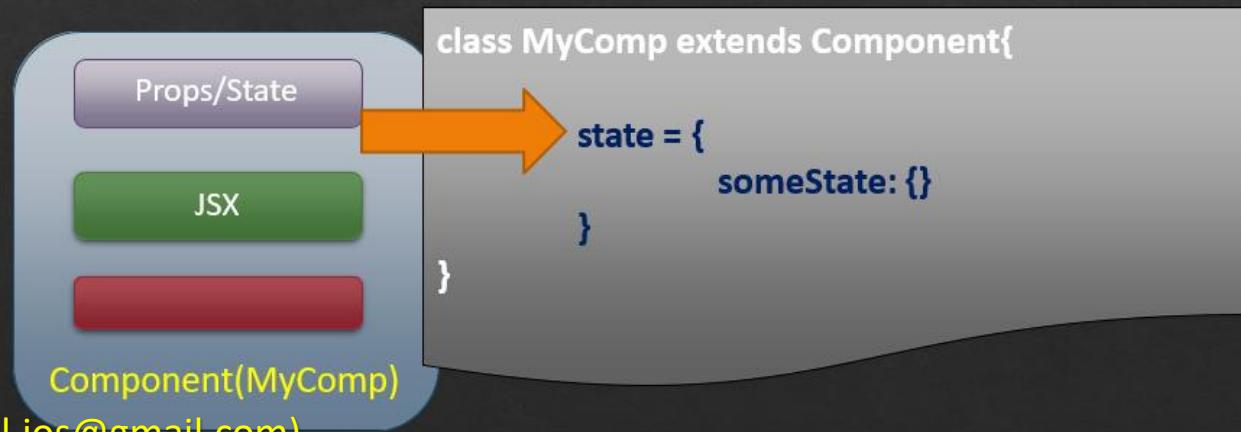
Component State

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

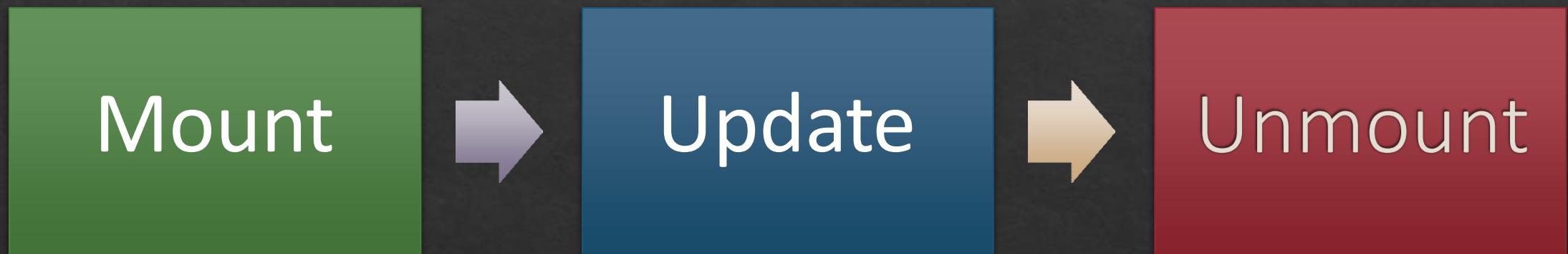
props allow you to pass data from a parent (wrapping) component to a child (embedded) component.

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)

constructor

- Setup State
- Don't cause side-effects

componentWillMount

- Setup State
- Don't cause side-effects
- Configuration

render

- Prepare and Structure the View

Render Child Components

componentDidMount

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

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

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.

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;

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

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**

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.

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.

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

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

<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.

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

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 or React Context

Client State

- IsAuthenticated, Filter Information
- Managed by Redux or React Context

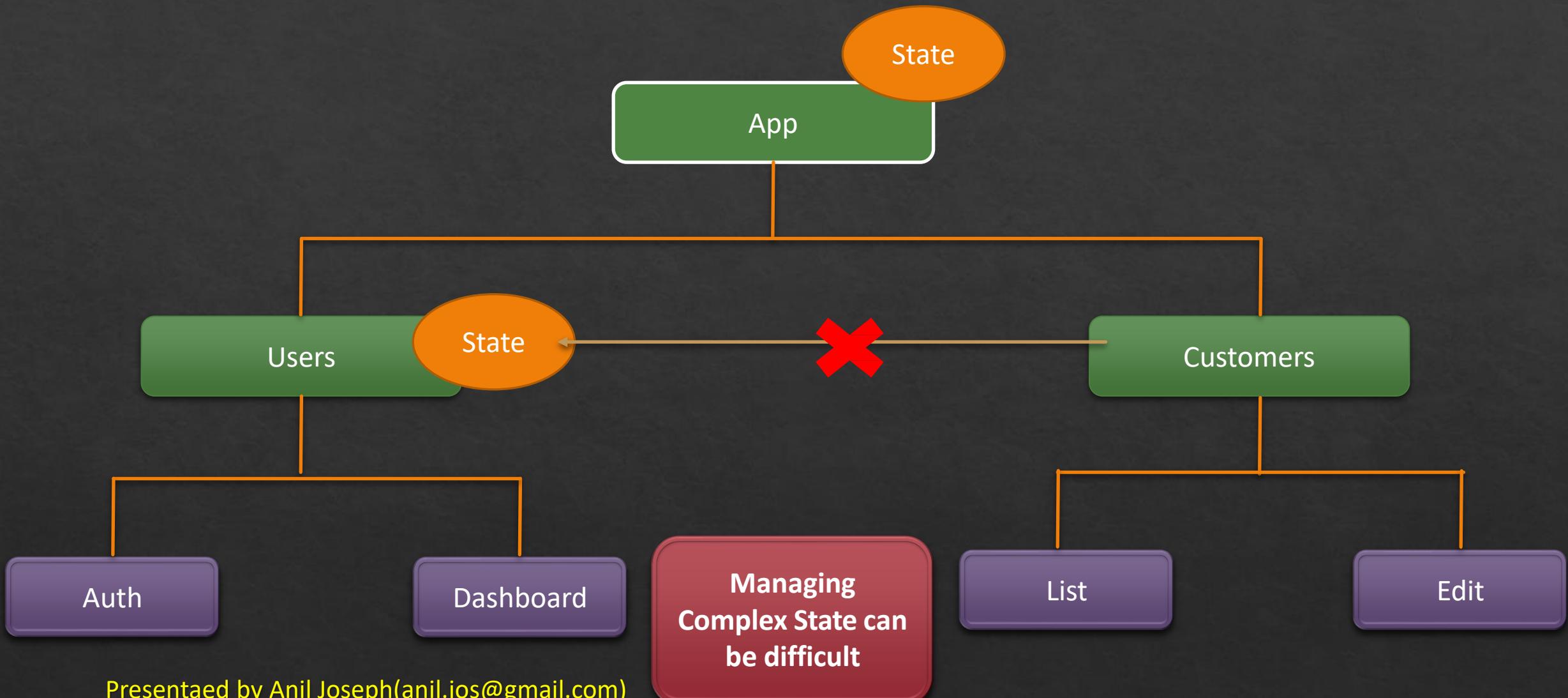
Redux

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

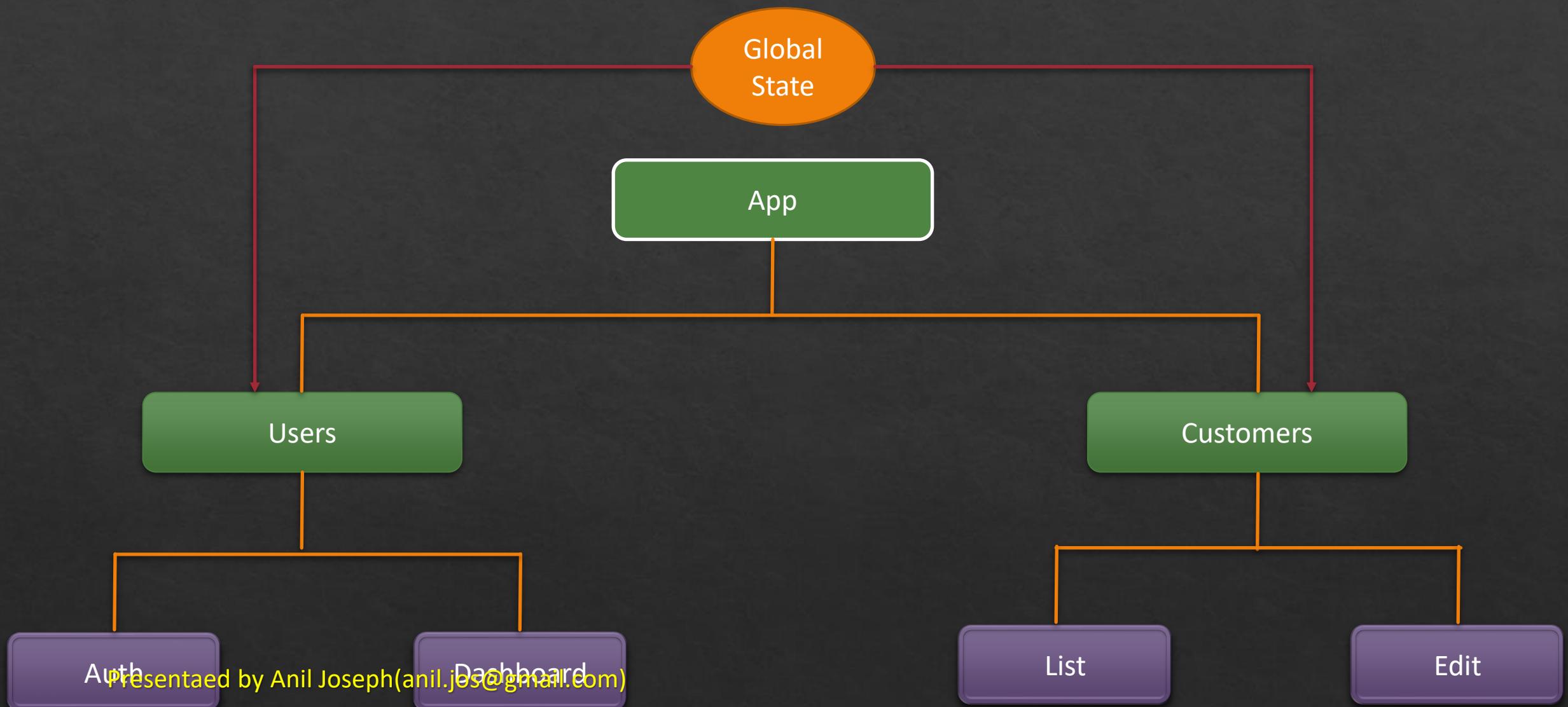
It is primarily used together with React or Angular for building user interfaces.

Redux was built on top of functional programming concepts.

Why Redux?



Why Redux?



Auth

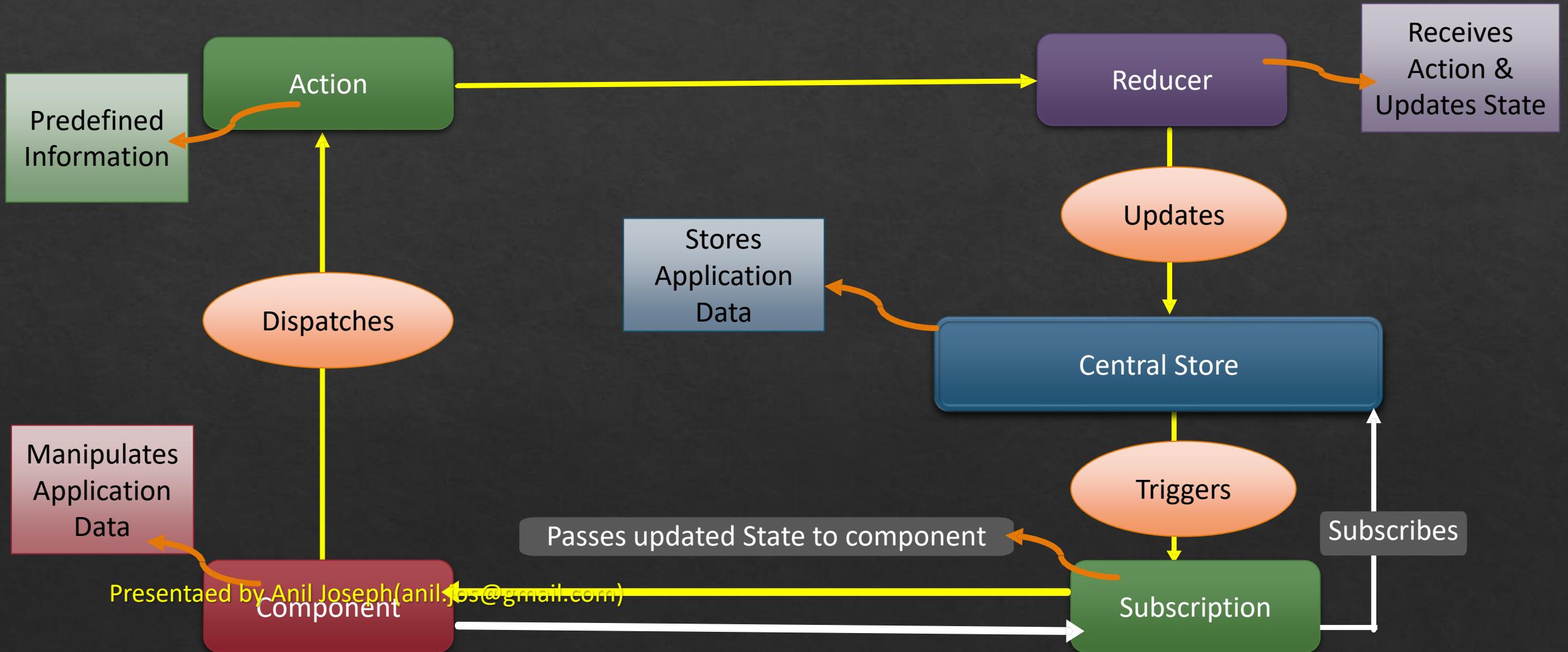
Presented by Anil Joseph(anil.jos@gmail.com)

Dashboard

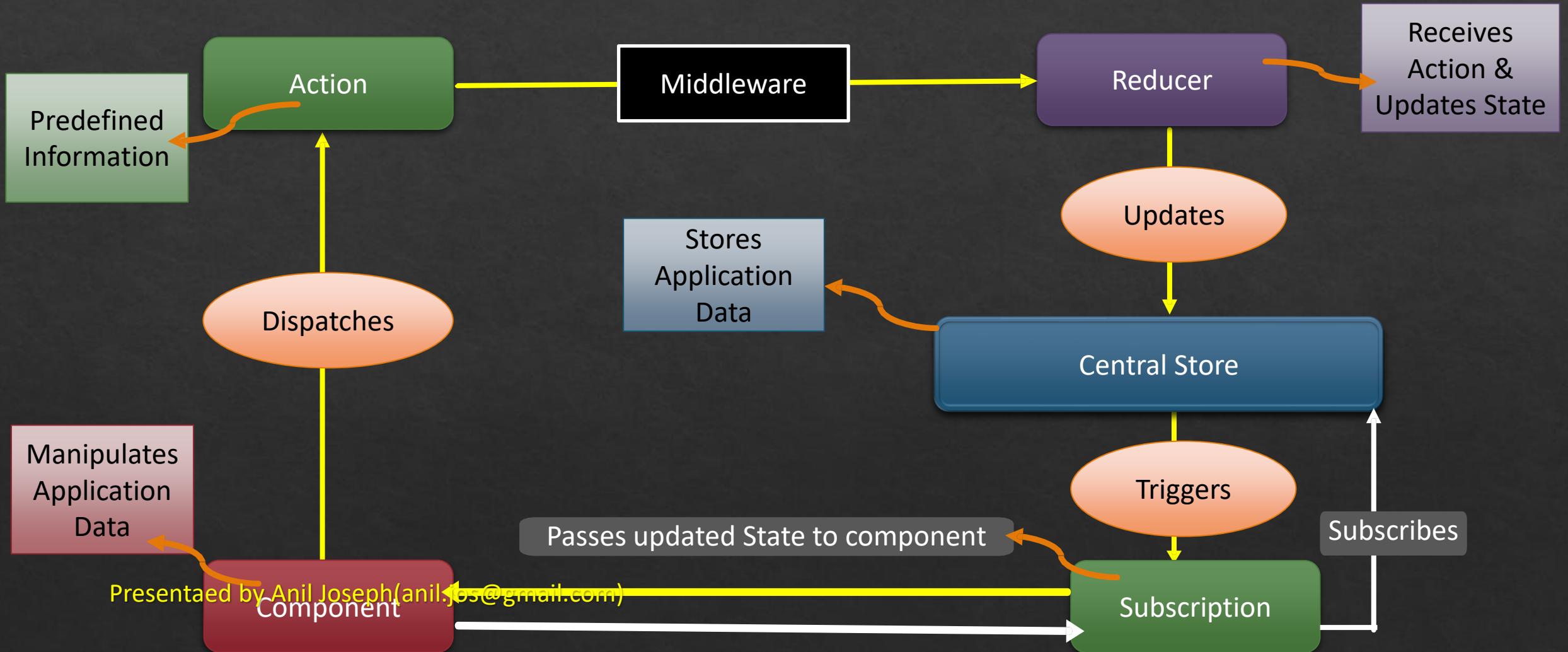
List

Edit

Redux Flow



Redux Flow



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 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.

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.

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.

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.

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

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.

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.

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.

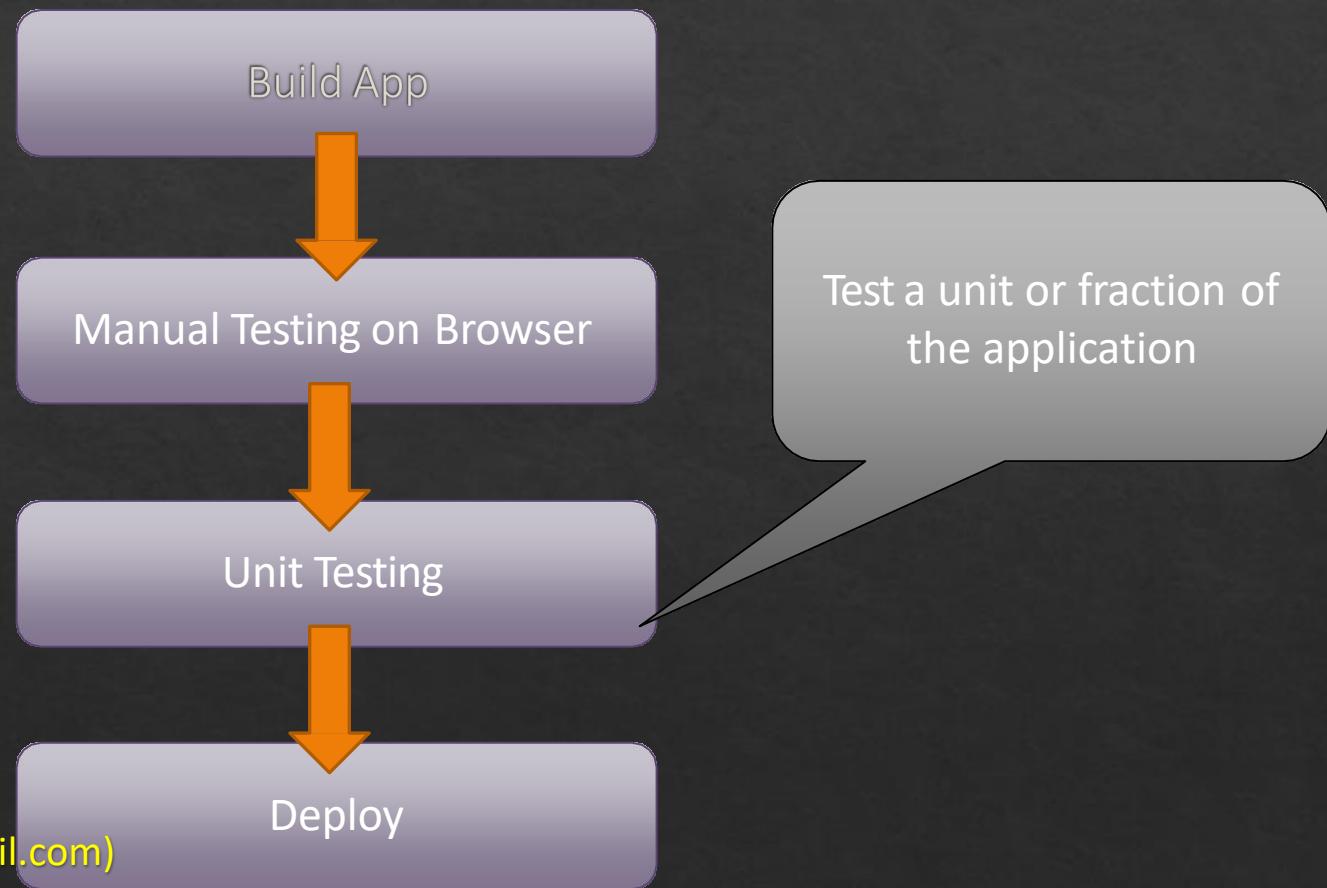
American Disability Act(ADA) compliance

- ❖ 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.

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

Testing



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

Any files inside a folder named `_tests_` are considered tests

`_test_ / *.js`



Any files with `.spec` or `.test` in their filename are considered tests

`*.spec.js`

`*.test.js`

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

Installation

```
npm install enzyme react-test-renderer  
enzyme-adapter-react-16
```

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

E2E Tools



Cypress



Puppeteer



Selenium Webdriver



Nightwatch.js

Thank You

ANIL JOSEPH

anil.jos@gmail.com

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