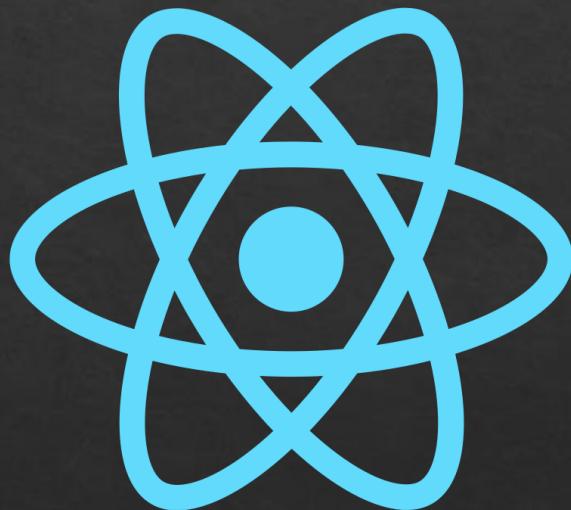
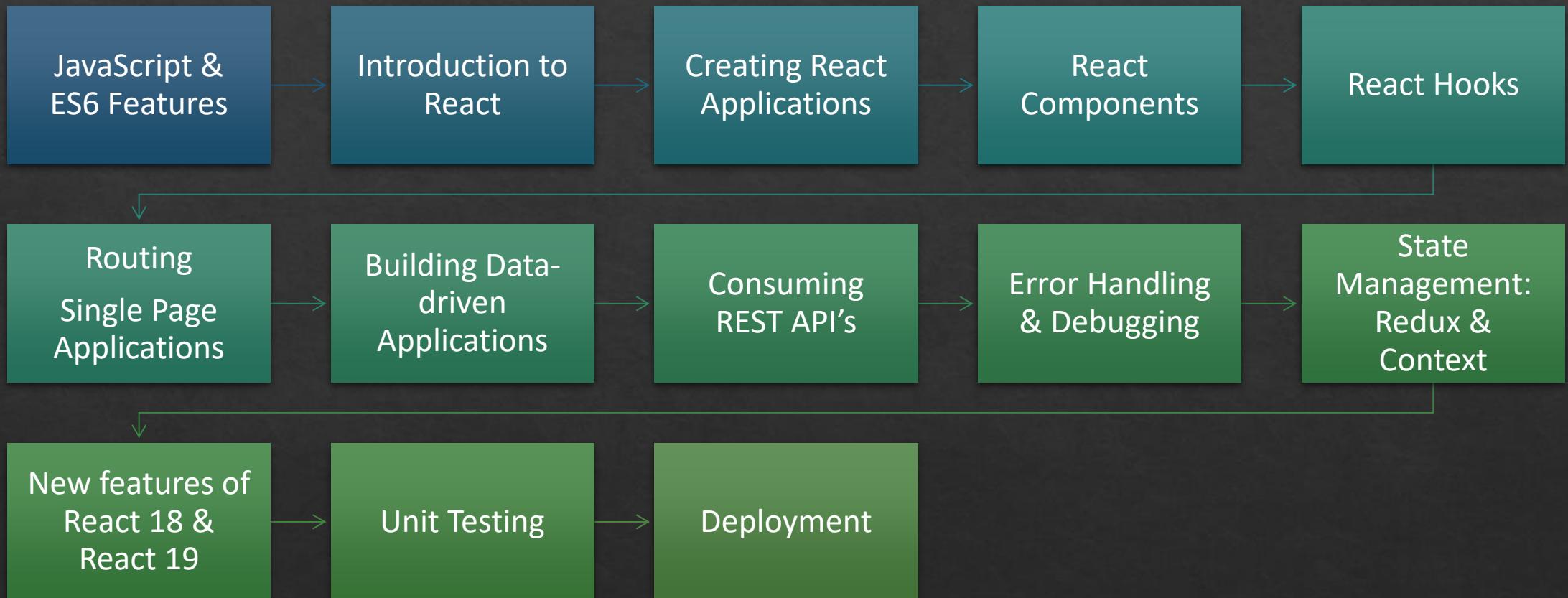


# React



ANIL JOSEPH

# Agenda



# 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)**

Postman

Browsers(Chromium)

# JavaScript

---

An interpreted language

---

Dynamic

---

Object-oriented

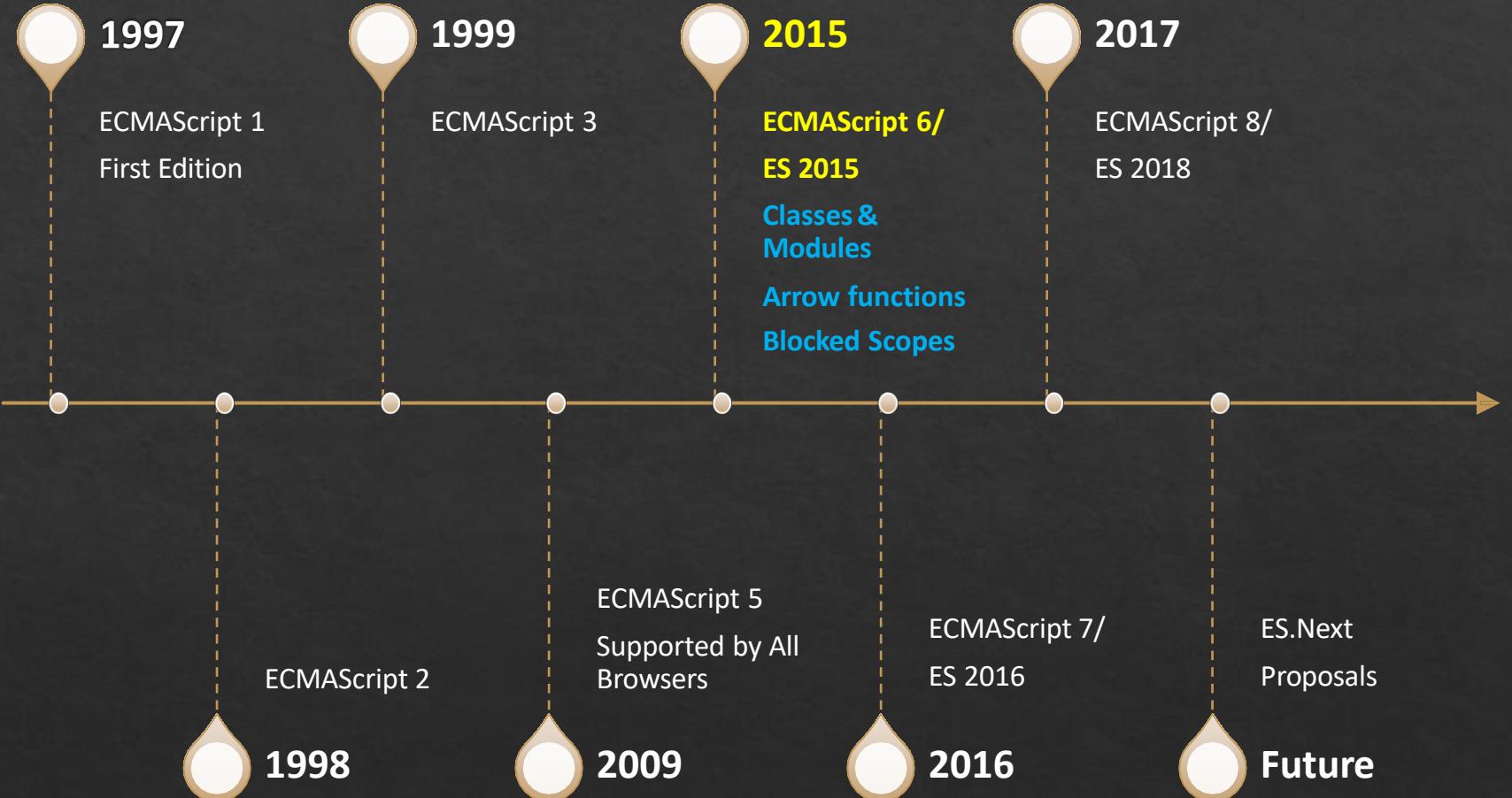
---

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”

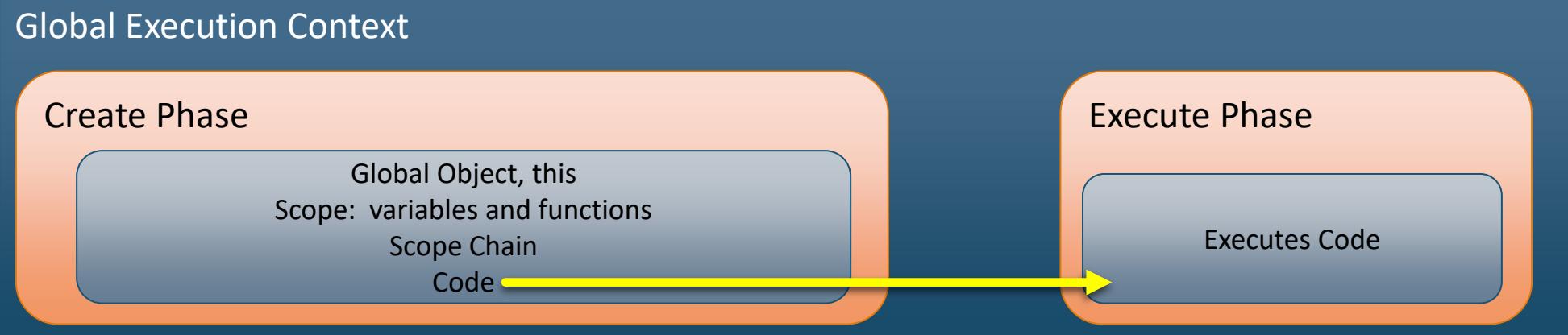
- ❖ Global Context: this refers to the global object (window in browsers).
- ❖ Function Context: this refers to the global object in non-strict mode and undefined in strict mode.
- ❖ Method Context: this refers to the object the method belongs to.
- ❖ Constructor Context: this refers to the newly created 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.

# 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**



# Execution Context

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

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

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

bar Execution Context(Create and Execute)

x = 30

foo Execution Context(Create and Execute)

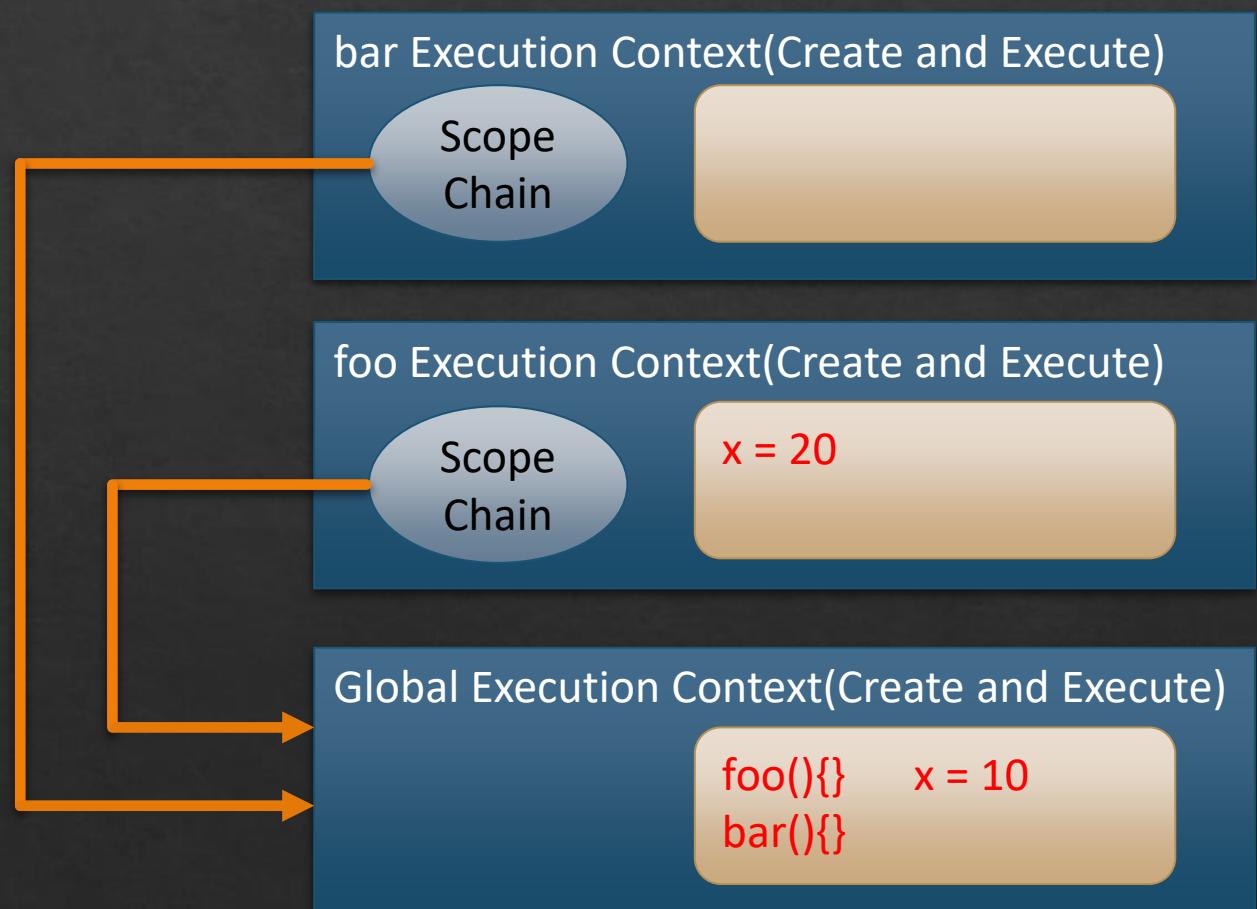
x = 20

Global Execution Context(Create and Execute)

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

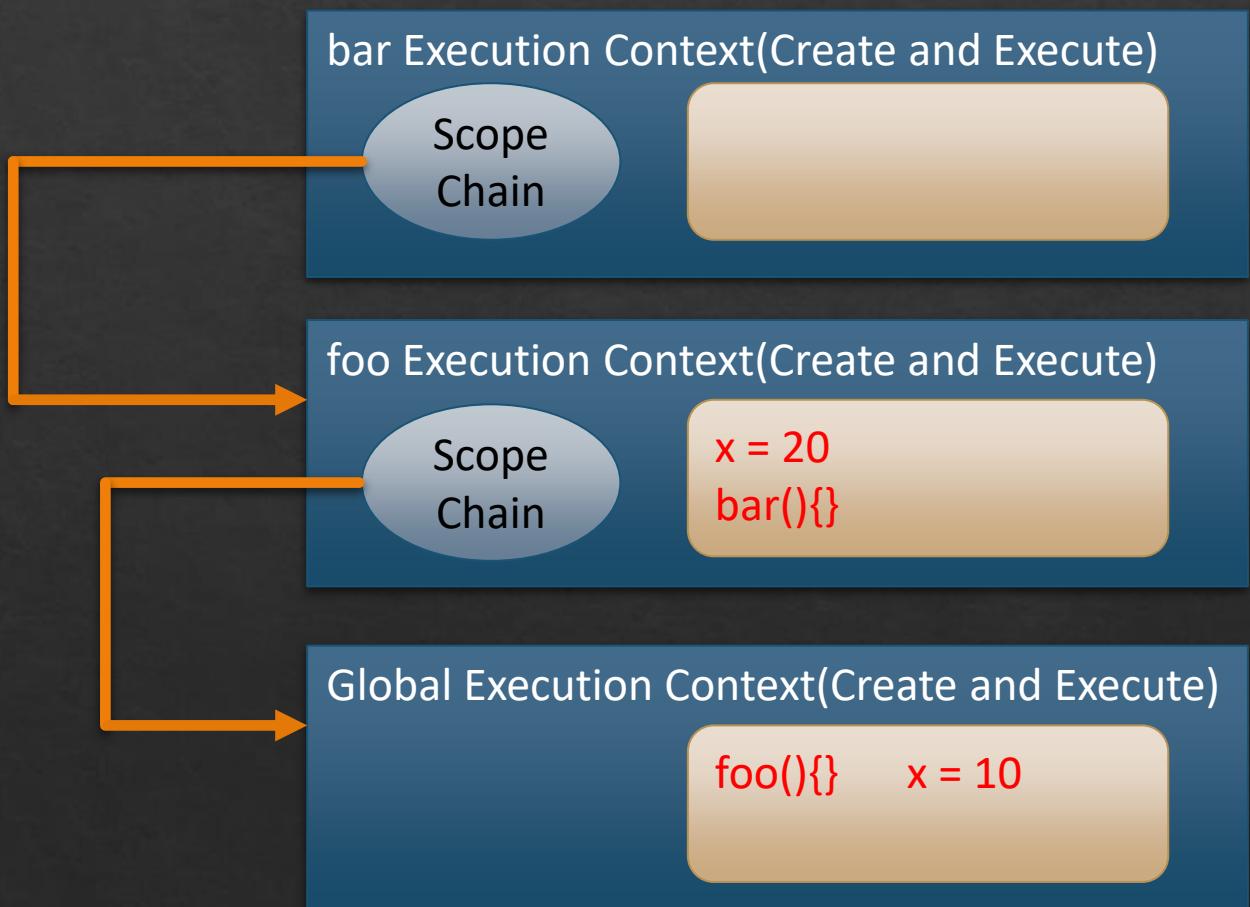
# Execution Context

```
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

Classes

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.

# Object Literal Extensions

Simplified  
Syntax for field  
assignment

Simplified  
Syntax for  
functions

Dynamic field,  
property and  
function names

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  
}
```

one.js

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

two.js

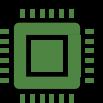
# Node.js



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)

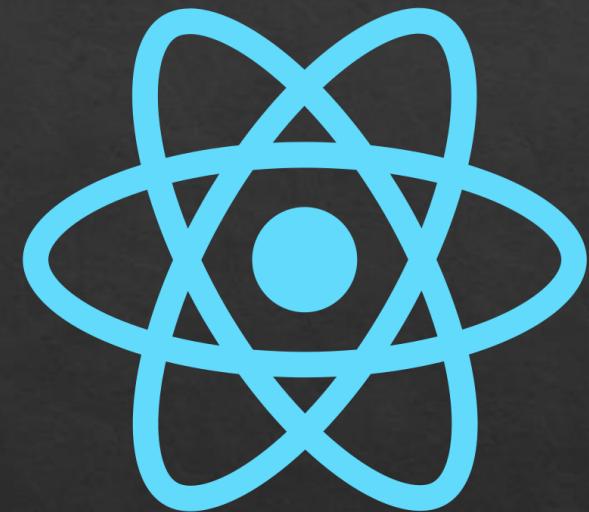
- ❖ NPM is a package manager for the JavaScript programming language.
- ❖ Allows users to consume and distribute JavaScript modules that are available on the registry
- ❖ The default package manager for Node.
- ❖ Comprises of
  - ❖ Command line client, also called npm
  - ❖ An online database called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

# React

- ❖ JavaScript Library: React is a library for building user interfaces.
- ❖ Component-Based: UIs are built using reusable components.
- ❖ Declarative: React allows developers to write declarative code, making UIs predictable and easier to debug.
- ❖ Single-Page Applications (SPAs): Ideal for building interactive SPAs that load content dynamically without full page reloads.



# Core Features

- ❖ **JSX (JavaScript XML):**
  - ❖ Combines JavaScript and HTML-like syntax to make UI structure more readable.
  - ❖ Allows embedding JavaScript logic within HTML-like markup.
- ❖ **Component-Based Architecture:**
  - ❖ UIs are split into reusable components, each managing its own state and logic.
  - ❖ Components can be nested, composed, and reused throughout the app.
- ❖ **Virtual DOM:**
  - React uses a **Virtual DOM** to optimize performance.
  - Changes in the UI are first applied to the Virtual DOM, and only the differences (diffs) are applied to the real DOM.

# Core Features

- ❖ Ecosystem Flexibility:
  - ❖ Can be used with other libraries or frameworks (like Redux for state management or Next.js for server-side rendering).
- ❖ Developer Experience:
  - ❖ Tools like React Developer Tools, JSX, and a clear component structure improve the overall development experience.
- ❖ SEO-Friendly (with SSR):
  - ❖ React can be combined with server-side rendering (SSR) using libraries like Next.js to improve SEO by rendering pages on the server.

# React History

- ❖ 2011: Origins
  - ❖ Developed by Jordan Walke, a software engineer at Facebook.
- ❖ 2013: Open Source
  - ❖ React was first released to the public at JSConf US in May 2013.
- ❖ 2014: React Grows
  - ❖ Instagram adopted React for their web app, marking a major milestone in React's real-world use.
- ❖ 2015: React Native
  - ❖ Launch of React Native, extending React's core concepts to mobile app development for iOS and Android.
- ❖ 2017: React 16 (Fiber Release)
  - ❖ React 16 was released, powered by the Fiber architecture, improving the ability to handle updates efficiently and improving performance.

# 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

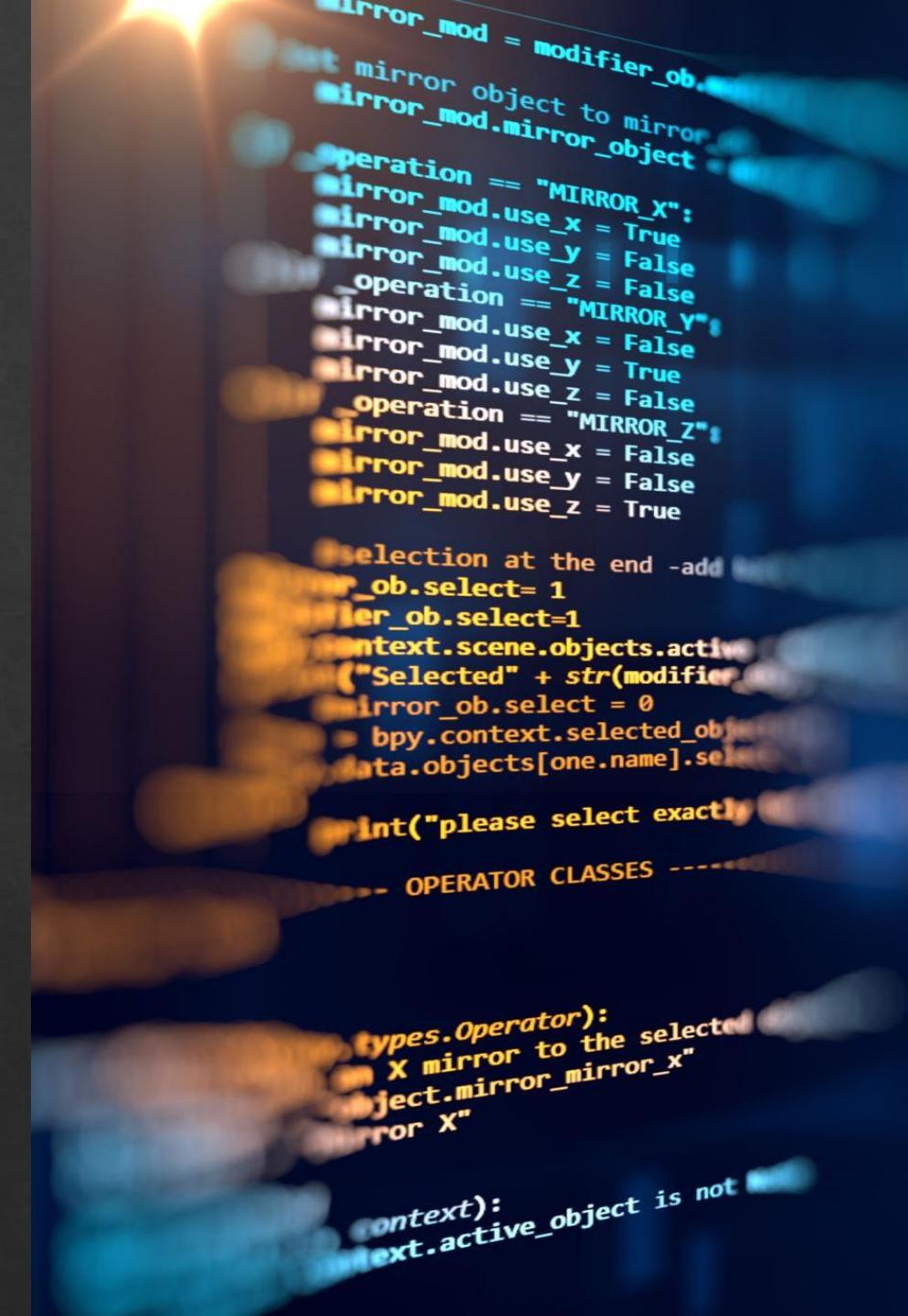
# Creating a React Project

# 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.



# Popular Tool Chains

- ❖ Next.js
  - ❖ A React framework for building production-ready applications with features like server-side rendering (SSR), static site generation (SSG), and API routes.
- ❖ Create React App (CRA)
  - ❖ Officially maintained toolchain for quickly bootstrapping new React applications.
  - ❖ Provides a zero-config setup with support for JSX, ES6+, and development server out of the box.
- ❖ Remix
  - ❖ Popular React-based framework that emphasizes server-side rendering (SSR) and focuses on providing an optimized developer experience for full-stack applications.
- ❖ Vite
  - ❖ A fast build tool and development server, often used for React projects.
- ❖ React Native
  - ❖ Extends React for building mobile applications.

# 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

## 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

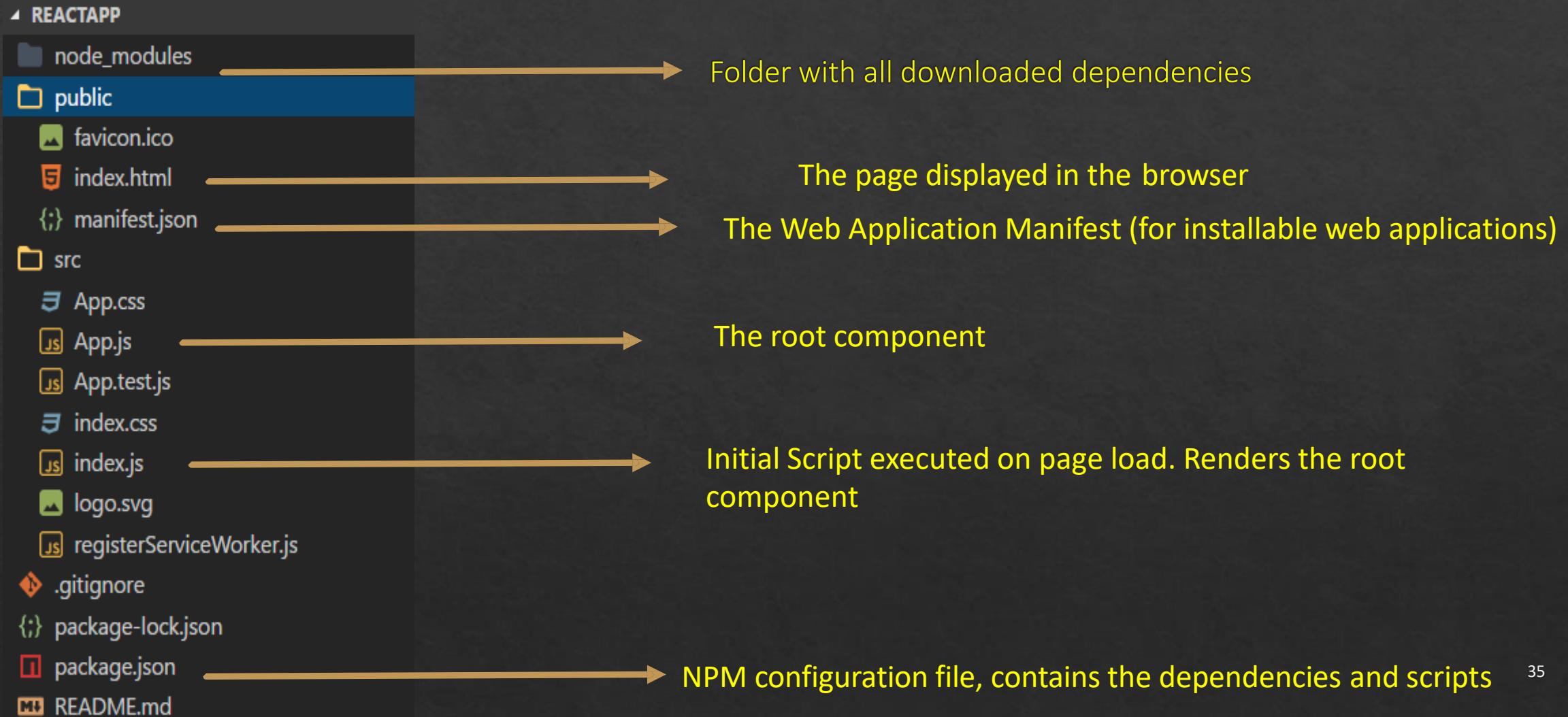
- `npx create-react-app the-react-app --template typescript`

3

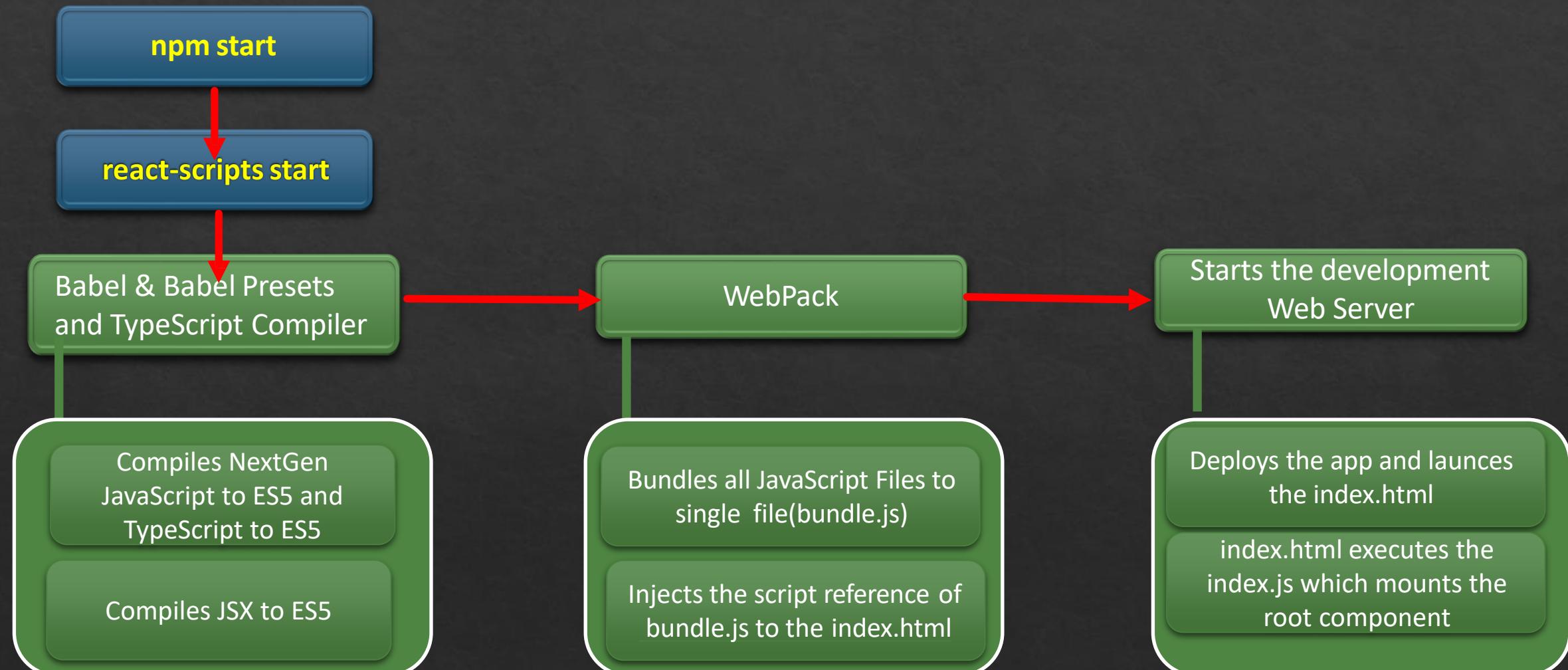
## Start the Application

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

# Project Structure

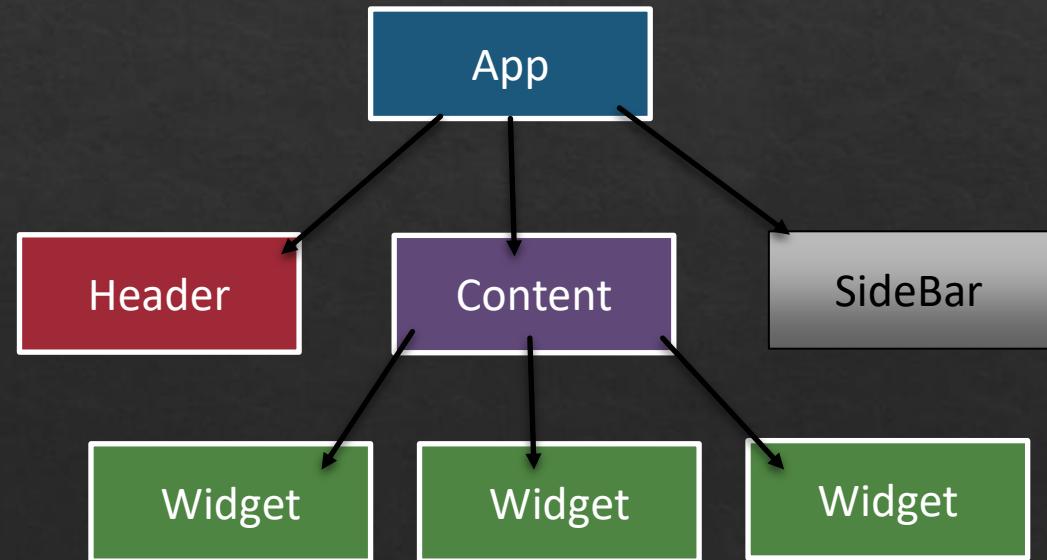
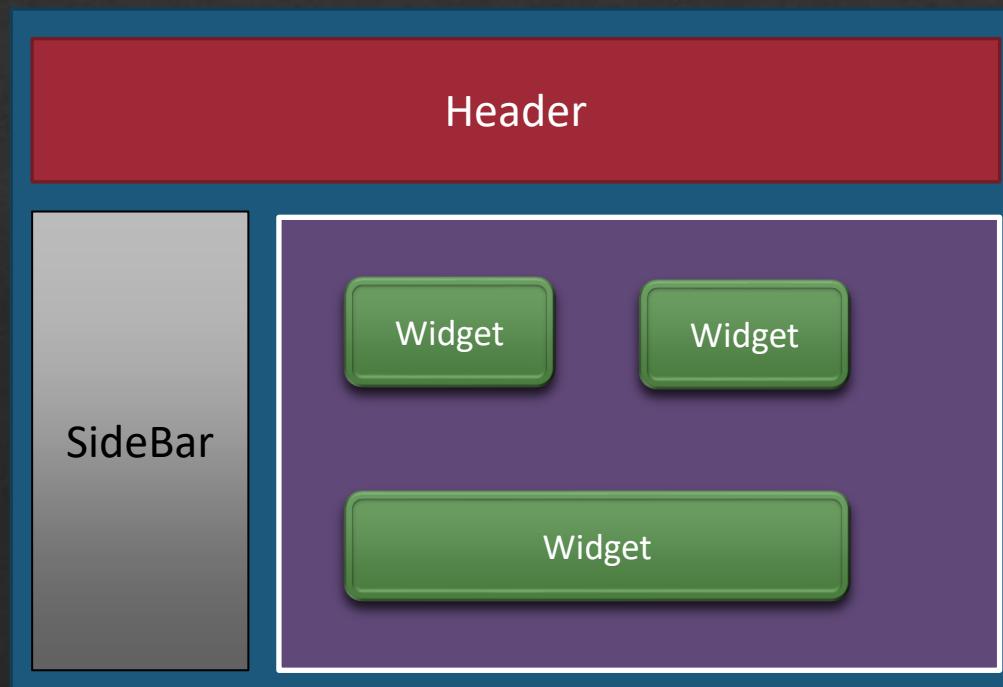


# Project Execution



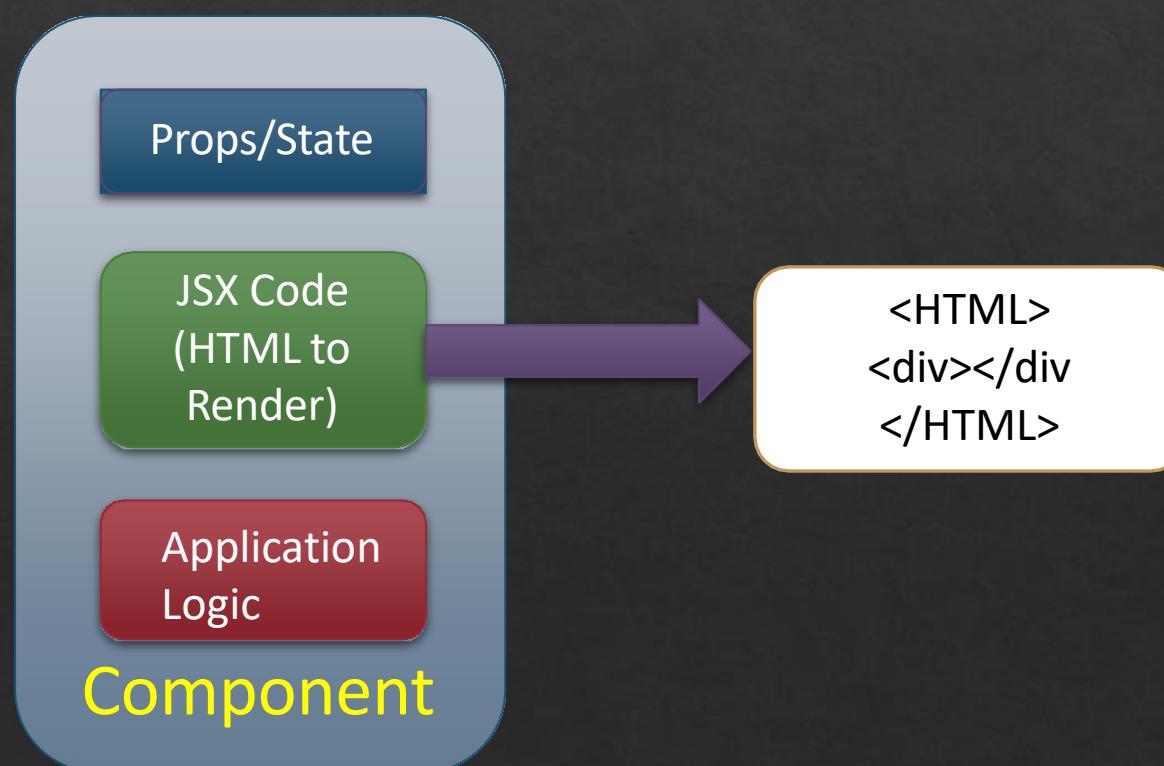
# React Components

- ❖ 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



# 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() }`

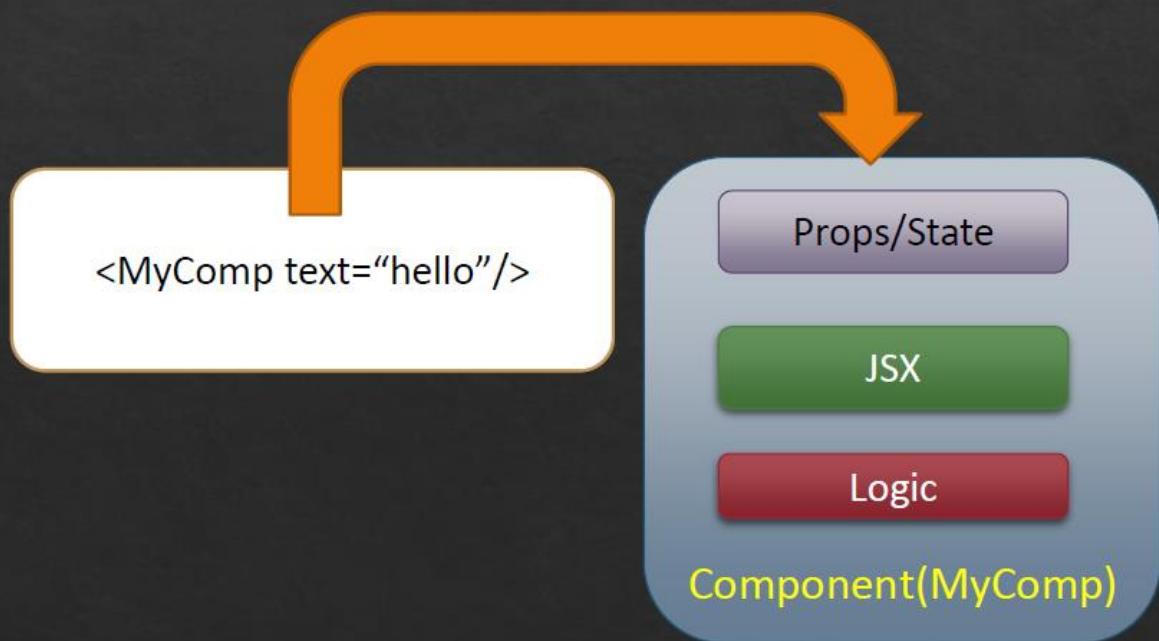
## Components: Properties(props)

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.***



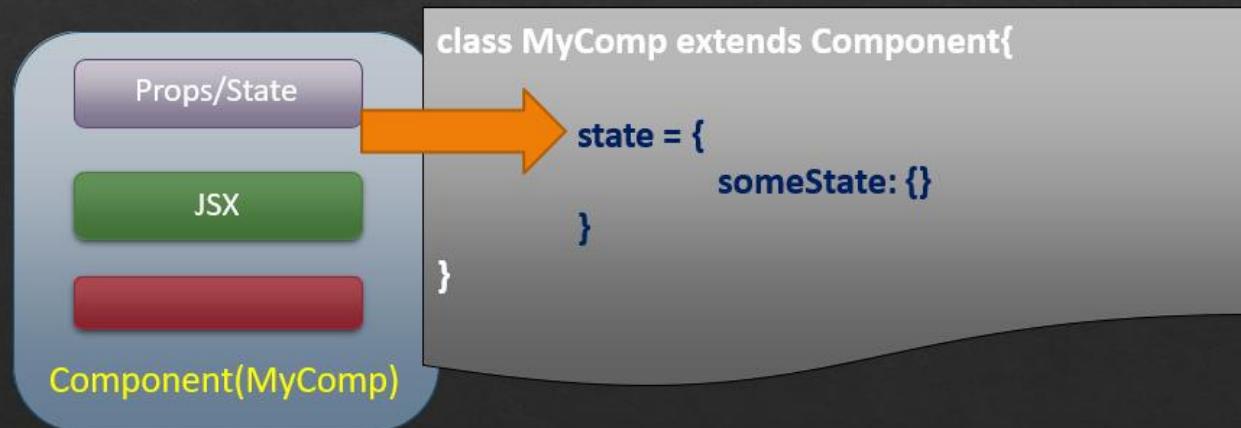
# Component State

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

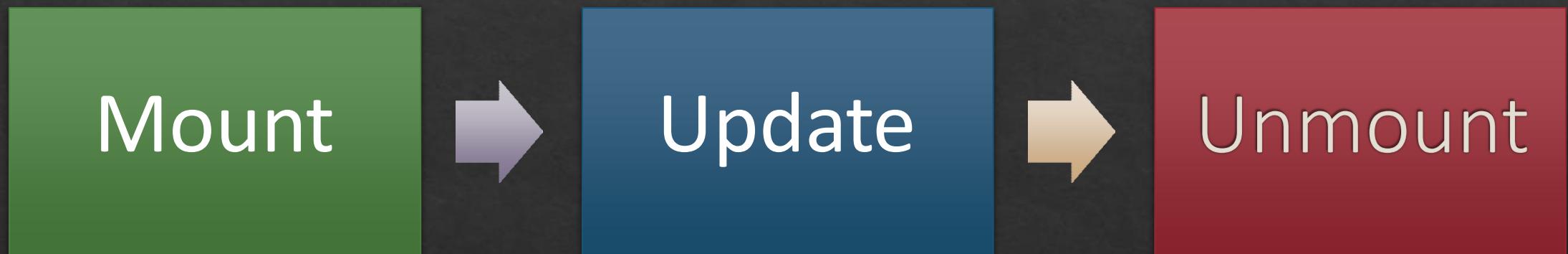
props allow you to pass data from a parent (wrapping) component to a child (embedded) component.

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



# API calls

- ❖ React does not provide any in-built library for API calls to the server.
- ❖ Common Libraries
  - ❖ Axios
  - ❖ Fetch
- ❖ Fetch is part of the W3C specifications and is supported by most modern browsers
- ❖ Axios is an open-source library based on the XMLHttpRequest(XHR) object with a lot of features.

# 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

- ❖ A lightweight, in-memory representation of the real DOM.
- ❖ Allows React to efficiently update the UI.
- ❖ Differs from the real DOM in that it is not rendered directly to the screen.

# Virtual DOM

- ❖ Initial Rendering:
  - ❖ React renders a virtual DOM tree when the app loads.
- ❖ State/Prop Change:
  - ❖ When state or props change, React creates a new virtual DOM tree.
- ❖ Diffing Algorithm:
  - ❖ React compares the new virtual DOM with the previous one (diffing).
- ❖ Reconciliation:
  - ❖ Only the actual changed elements in the real DOM are updated, making it highly efficient.

# 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 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.

# 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

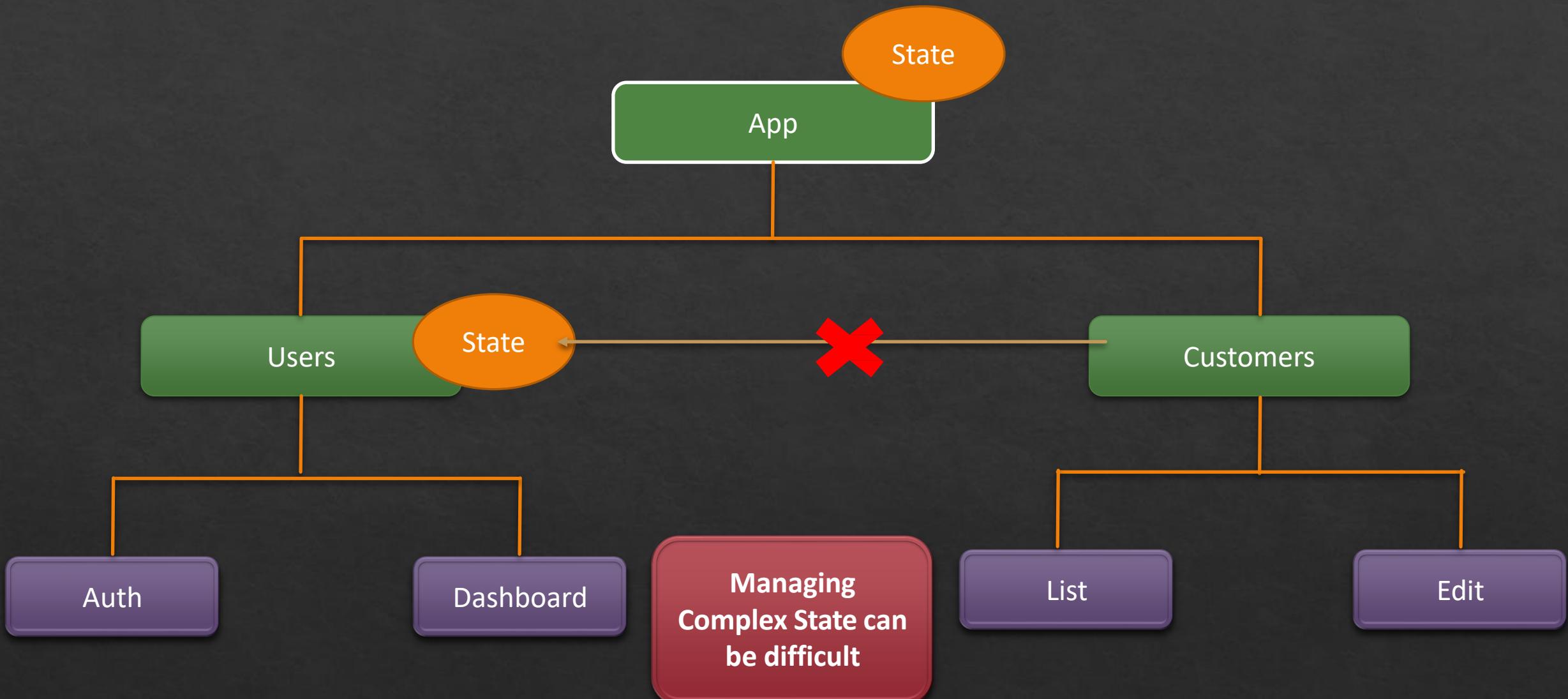
# Redux

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

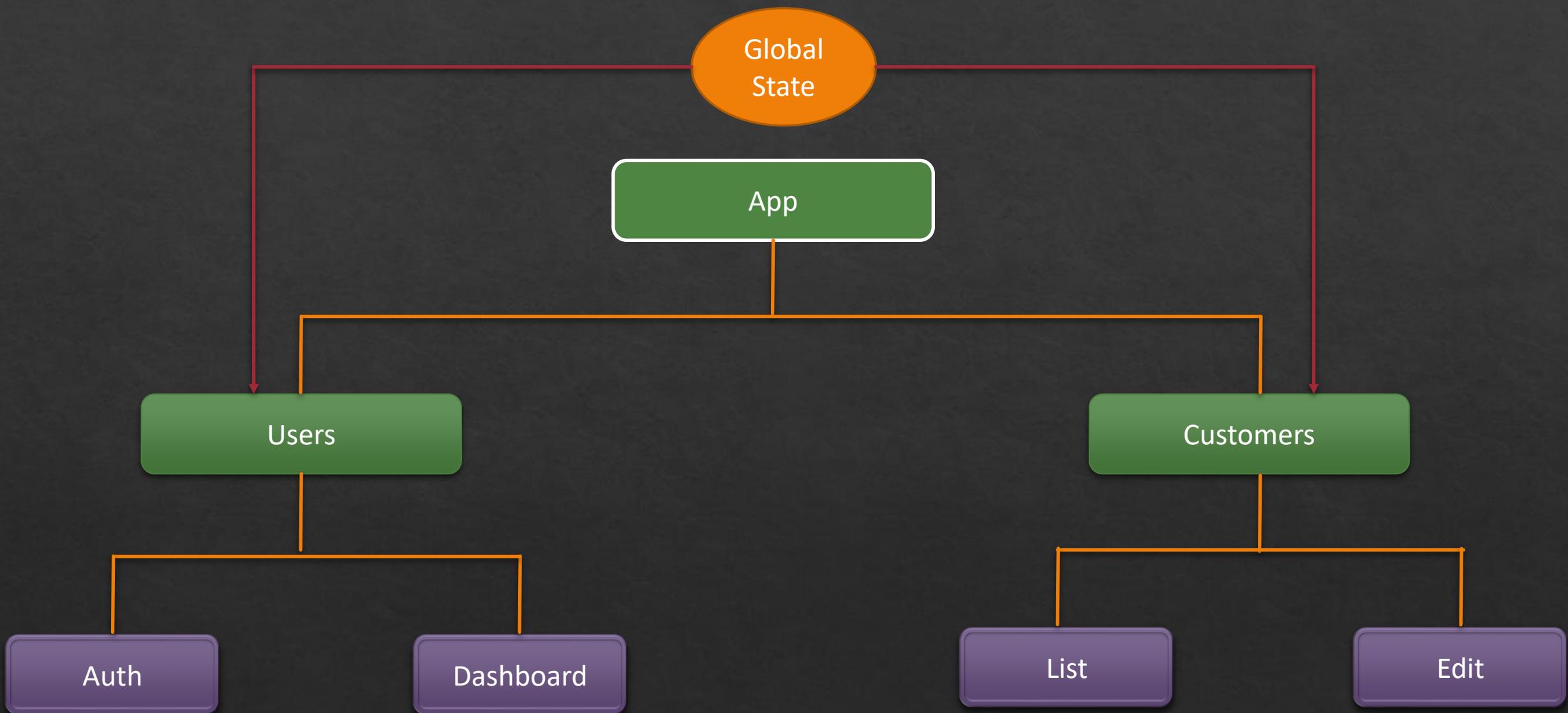
It is primarily used together with React or Angular for building user interfaces.

Redux was built on top of functional programming concepts.

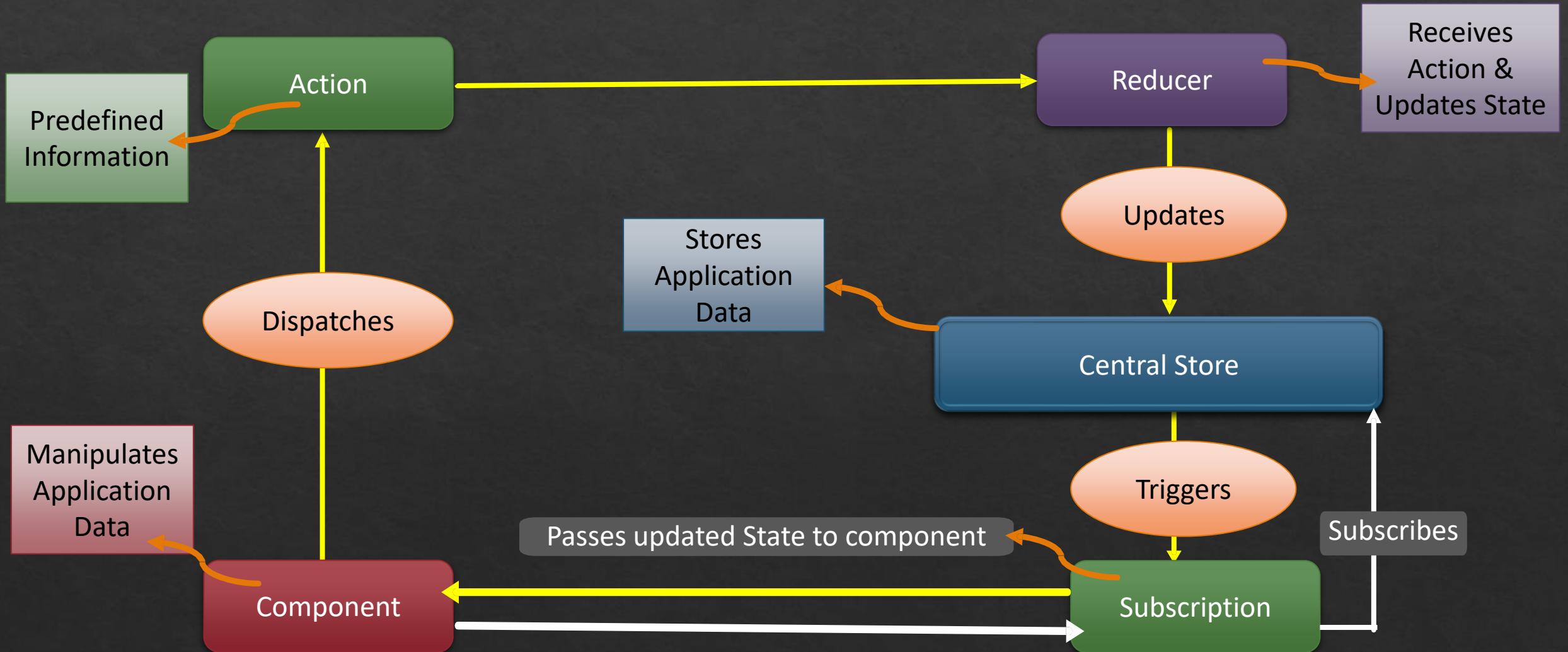
# Why Redux?



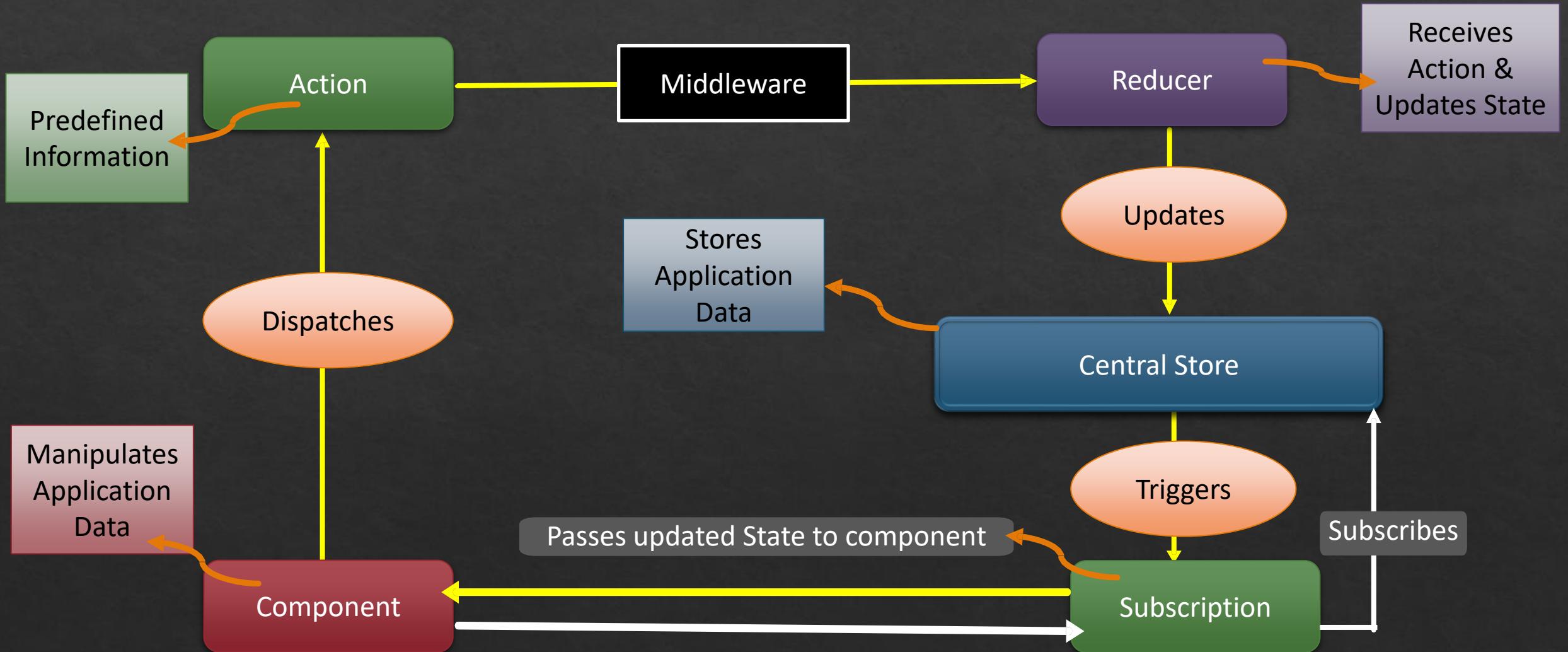
# Why Redux?



# Redux Flow



# Redux Flow



# 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.

# 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.

# 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`

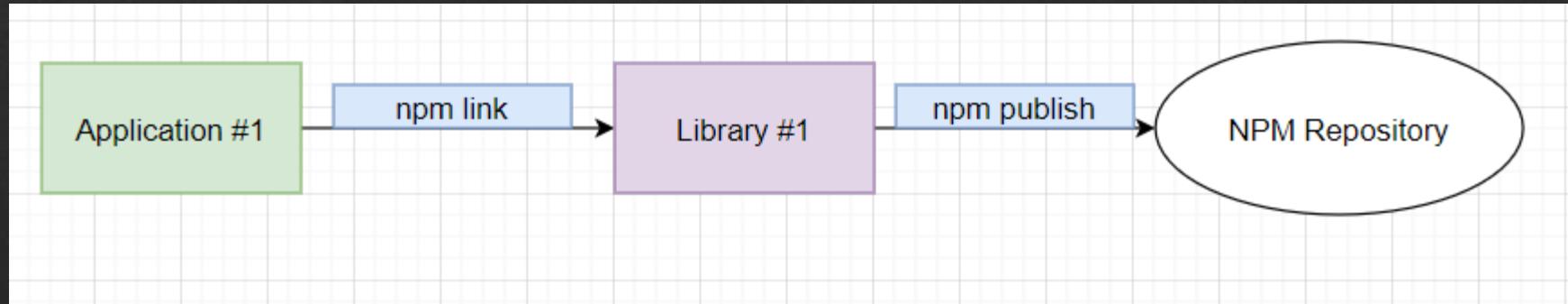
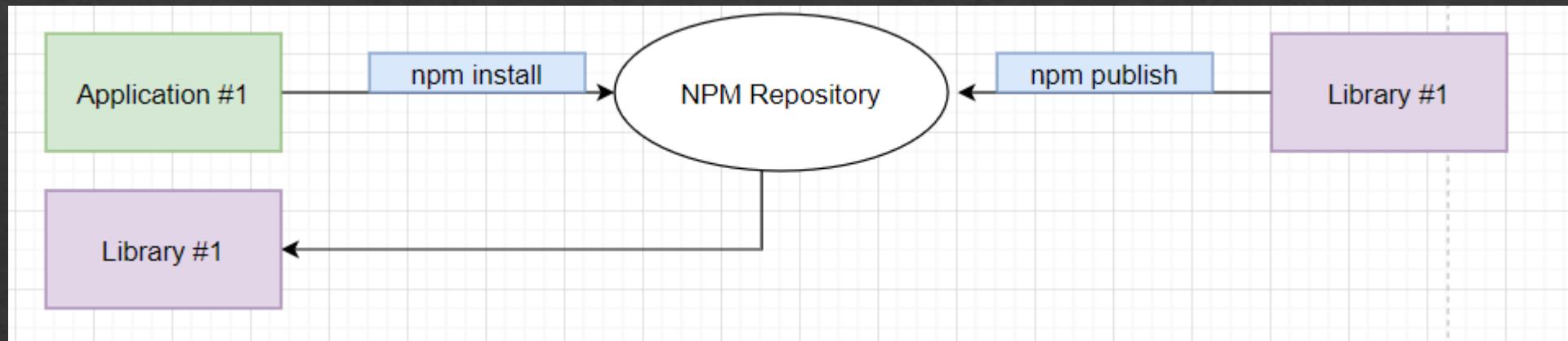
# Error Boundaries

- ❖ 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

# React Library





# Components in React 18

## Client Components:

- Rendered and hydrated in the browser.

## Server Components:

- Rendered on the server with no client-side JavaScript.

# Client Components

- ❖ Rendered on the client (browser) after initial page load.
- ❖ Can use React hooks like useState, useEffect, and event handlers (onClick).
- ❖ Suitable for interactive elements (e.g., buttons, forms).
- ❖ Require JavaScript on the client for interactivity.

# React Server Components

- ❖ Server Components are a new feature introduced in React to improve performance by rendering parts of the UI on the server and sending it as static HTML to the client.
- ❖ No Client-Side JavaScript:
  - ❖ Server Components don't ship any JavaScript to the client. This reduces bundle sizes and improves performance.
- ❖ Seamless Integration:
  - ❖ Server Components work alongside Client Components, providing a seamless developer experience within the same React tree.
- ❖ Improved Performance:
  - ❖ By offloading intensive tasks to the server, Server Components reduce the need for client-side processing, leading to faster load times and better scalability.

# React Server Components

- ❖ Data Fetching:
  - ❖ Server Components are ideal for fetching data from databases or APIs without exposing sensitive logic to the client.
- ❖ Rendering Heavy UIs:
  - ❖ Offload rendering that requires heavy computation or multiple data sources to the server.
- ❖ Automatic Caching:
  - ❖ Server Components enable automatic caching and reduce unnecessary re-renders by separating the UI from client-side interactivity.

# React Server Components: Limitations

- ❖ No Interactivity:
  - ❖ Server Components cannot handle client-side interactions, like event listeners.
  - ❖ Cannot use hooks and other client specific features like Redux, React Context etc.
- ❖ Limited Access to Browser APIs:
  - ❖ Server Components run on the server, so they cannot directly interact with browser-specific APIs (e.g., window, localStorage).

# Client vs. Server Components

- ❖ Rendering Environment
  - ❖ Client Components: Rendered in the browser on the client-side.
  - ❖ Server Components: Rendered on the server and sent as static HTML to the client.
- ❖ Interactivity
  - ❖ Client Components: Can include interactive elements like event listeners (onClick, onChange), hooks (useState, useEffect), and browser-specific APIs (e.g., localStorage, window, document).
  - ❖ Server Components: Cannot include interactivity or use browser-specific APIs. They are designed to handle static data fetching and rendering without any client-side logic.

# Client vs. Server Components

- ❖ Data Fetching
  - ❖ Client Components: Fetch data on the client-side using APIs such as `fetch`, `axios`, or any other client-side fetching method. They can only access client-side environment variables.
  - ❖ Server Components: Fetch data on the server-side before rendering the component. They have access to both client and server environment variables, and can perform secure server-side operations (like database queries, authentication checks, etc.).
- ❖ Rendering Strategy
  - ❖ Client Components: Are hydrated after the initial HTML is sent from the server. This means JavaScript code is sent to the browser and executed to make the component interactive.
  - ❖ Server Components: Render HTML on the server, and the client receives static HTML. No JavaScript is sent to the client for hydration unless a Client Component is nested inside.

# Client vs. Server Components

- ❖ Performance
  - ❖ Client Components: May impact performance since JavaScript needs to be downloaded, parsed, and executed on the client. Hydration adds additional overhead.
  - ❖ Server Components: Better for performance because they only send minimal HTML to the client and don't include client-side JavaScript unless needed. This leads to faster loading times.
- ❖ Bundle Size
  - ❖ Client Components: Increase the bundle size because JavaScript code needs to be sent to the client for execution.
  - ❖ Server Components: Reduce the bundle size since no client-side JavaScript is needed (except for interactive parts), resulting in a smaller footprint and quicker load times.

# Client vs. Server Components

Feature	Client Component	Server Component
Rendered in	Browser (client-side)	Server
Interactivity	Supports interactivity	No interactivity
Data Fetching	Client-side	Server-side
Bundle Size	Increases bundle size	Minimal bundle size
Reactivity	Can use state and effects	No reactivity or state
Hooks	Can use all React hooks	Can only use server hooks ( <code>use</code> )
Access to Browser APIs	Yes	No
Usage	Forms, dynamic UI elements	Static content, SEO pages
Third-party libraries	Can use client-side libraries	Limited to server-side

# Next.js

- ❖ Next.js is a popular React framework
- ❖ Powerful features for building server-side rendered (SSR) and static web applications.
- ❖ Full stack development(JavaScript)

# Next.js Features

Full-Stack  
Framework

File-based Routing

Client-side  
Rendering (CSR)

Server-side  
Rendering (SSR)

Static Site  
Generation (SSG)

Incremental Static  
Regeneration (ISR)

Hybrid Rendering

API Routes

Automatic Code  
Splitting

Built-in CSS and  
Sass Support

Image  
Optimization

Fast Refresh

# Create a Next.js Project

1

## Install NodeJs

- Runtime to run/execute JavaScript on the machine/server
- Use Node 18.8 or above for the latest version(Next 14.x)

2

## Create Next Application

- `npx create-next-app@latest`
- `npx create-next-app@14.2.7`

3

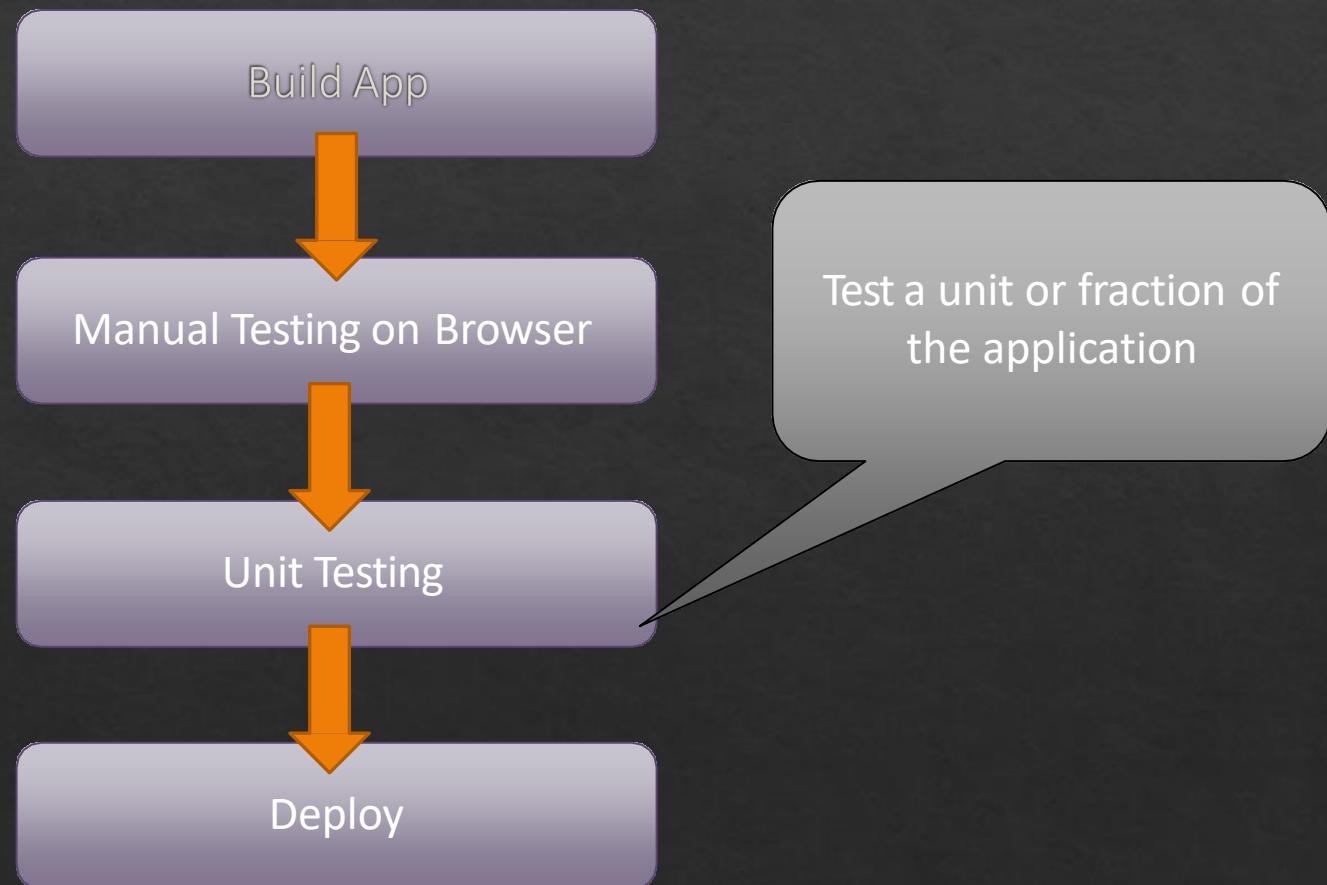
## Start the Application

- **Go to the project directory**
- **Run `npm run dev`**

# Next.js Project options

```
✓ What is your project named? ... the-awesome-app
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use 'src/' directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
? Would you like to customize the default import alias (@/*)? » No / Yes
```

# 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

# 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

# Deployment

Set basePath of Router

- <BrowserRouter basename="/app/">

Set homepage in package.json

- "homepage": "/app"

Build and Optimize npm run build

Server must Always serve index.html

Deploy artifacts to Web Server

# Thank You

ANIL JOSEPH

[anil.jos@gmail.com](mailto:anil.jos@gmail.com)