React Functional Component - Refs

Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

React Refs

- React works on the concept of breaking the code into smaller components, these small components help us to focus on specific areas.
- In React all the data flow happens through state and props.
- Whenever a state or prop changes the component is re-rendered.
- Sometimes there is a requirement to modify a component that is outside of the workflow, in such cases refs in react come to the rescue.
- State and props are the default way to update a component in React. Whenever a state or prop is changed, the component is re-rendered.
- React team has made refs in react which act as a bridge.

React Refs

- This bridge allows a component to access or modify an element that the ref is attached to.
- Refs provides us with a way to bypass state updates and re-renders.
- This is very useful but we should not use refs as an alternative to state or props.
- When we need to update a component without actually causing a re-render, refs in react come to the rescue.
- They allow us to directly modify the DOM element without using state.
- We know that react uses the state to trigger re-render but refs allow us to do modifications without triggering a re-render.

Creating Refs

- Both Functional and Class based components support refs.

Class Components

- To create a ref in the class-based component we use the React.createRef() method.
- Refs are set to an instance property so that they can be used as a reference throughout the component.
- This all happens when the component is created.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)
    this.firstRef = React.createRef()
  }
  render() {
    return <div ref={this.firstRef} />
  }
}
```

Creating Refs

Functional Components

- In functional components we utilize hooks, so we do not need to use the createRef() method, instead, we use the useRef() hook to create refs in functional components.

```
import React, { useRef } from 'react'
const ActionButton = () => {
  const buttonRef = useRef(null)
  return <button ref={buttonRef}>Hello</button>
}
export default ActionButton
```

Adding a Ref to a DOM Element

- Accessing a DOM node can be done using the useRef hook.
  - ✓ Import the useRef hook
  - ✓ Declare a ref inside the component
  - ✓ Pass the ref to the DOM node as a ref attribute.

```
import {useRef} from 'react'

const myRef = useRef(null)

<div ref={myRef}></div>
```

- The useRef hook returns an object with a single property called current.
- Here, initially, the current property will be set to null.
- When a DOM node is created for this div, a reference to this node will be placed in myRef.current and this reference can be accessed throughout the component lifecycle.

Adding a Ref to a Class Component

- In Class Components refs are created using the React.createRef() method.
- This method is used to create refs and then attach them to the component using the ref attribute.
- In Class components, when the component is created, refs are assigned to an instance property.

```
export class MyComp extends Component {
  constructor(props) {
    super(props)
    this.myRef = React.createRef()
  }
  render() {
    return <div ref={this.myRef} />
  }
}
```

How to Update a Refs Value?

- The ref consists of an object which has a current property.
- The value we assign to a ref is stored inside this current property.
- This property can be accessed using ref.current and we can simply assign the value to this property.

```
//importing hook
import { useRef } from 'react'

export default function Counter() {
  //calling the useRef hook
  let ref = useRef(0)

  function handleClick() {
    ref.current = ref.current + 1
    console.log('Clicked ' + ref.current + ' times')
  }

  return <button onClick={handleClick}>Click me!</button>
}
```

React Refs

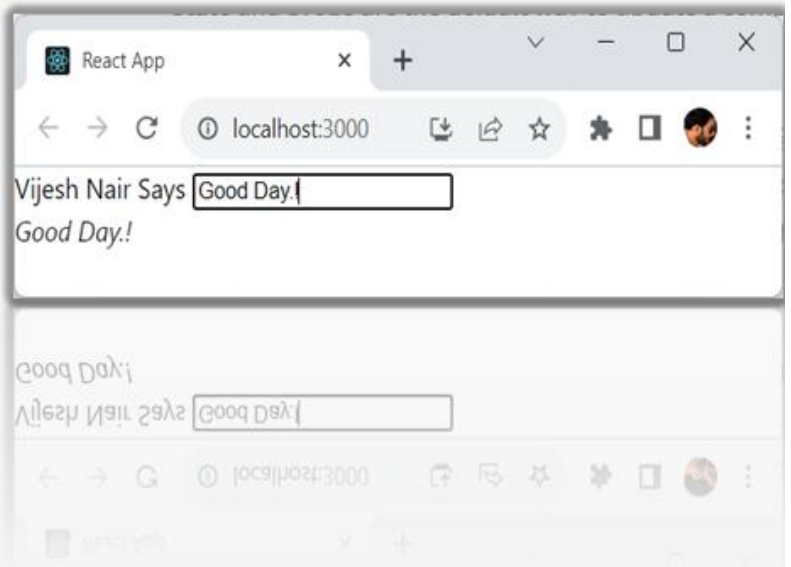
D: > Adv_React > refsexample > src > JS App.js > ...

```
1   import React, { Component } from 'react';
2   import ReactDOM from 'react-dom';
3   import { render } from 'react-dom';
4
5
6   // using refs
7   class App extends React.Component {
8     constructor() {
9         super();
10        this.state = { sayings: "" };
11    }
12    update(e) {
13        this.setState({ sayings: this.refs.anything.value });
14    }
15    render() {
16        return (
17            <div>
18                Vijesh Nair Says <input type="text" ref="anything"
19                    onChange={this.update.bind(this)} />
20                <br />
21                <em>{this.state.sayings}</em>
22            </div>
23        );
24    }
25  }
26  ReactDOM.render(< App />, document.getElementById('root'));
27
28  export default App;
```

```
D:\Adv_React>npx create-react-app refsexample

Creating a new React app in D:\Adv_React\refsexample.
```

React Refs

```
Windows PowerShell

D:\Adv_React>cd refsexample

D:\Adv_React\refsexample>npm start

> refsexample@0.1.0 start
> react-scripts start

(node:12936) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is d
eprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:12936) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is
 deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view refsexample in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.37:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



React - Hooks

React Hooks

- Hooks were first made available in React 16.8 in February 2019.
- React Hooks offer us other means to access features like life Cycle, manage the state of your component, or perform an after-effect when specific changes are made to the state(s) without the need to create classes.
- It allows you to use state and other React features without writing a class.
- Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.
- Hooks are backward-compatible.
- Hooks doesn't violate any existing React concepts. Instead, Hooks provide a direct API to react concepts such as props, state, context, refs and life-cycle

React Hooks

When to use a Hooks?

- If we write a function component, and then to add some state to it, previously we do this by converting it to a class. But, now we can do it by using a Hook inside the existing function component.

Rules of Hooks

- Hooks are not just regular JavaScript functions. They have some special rules that we need to follow to use them correctly and avoid bugs.
- The two main rules of Hooks are:
  - ✓ Only call Hooks at the top level of your component. Don't call them inside loops, conditions, or nested functions.
  - ✓ Only call Hooks from React function-based components or custom Hooks. Don't call them from regular JavaScript functions or class-based components.

Why use Hooks?

1. **Use of 'this' keyword:**

- It is easier to understand the concept of props, state, and uni-directional data flow but using 'this' keyword might lead to struggle while implementing class components.
- One also needs to bind event handlers to the class components.
- React developers team also observed that classes don't concise efficiently which leads to hot reloading being unreliable which can be solved using Hooks.

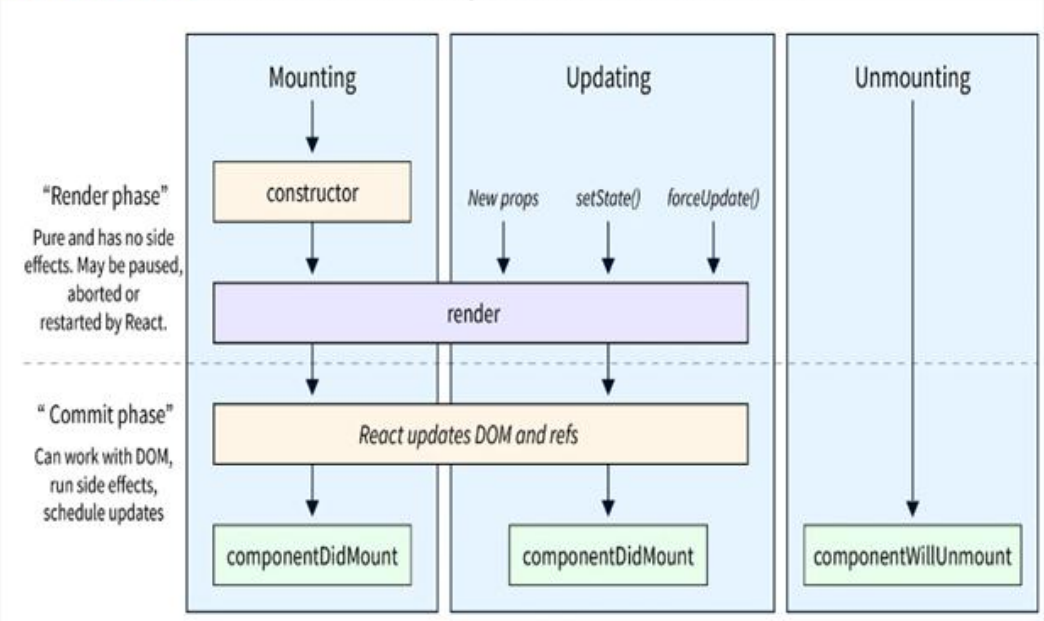
2. **Reusable stateful logics:**

- There is no mechanism to "attach" reusable behavior to a component in React (for example, connecting it to a store).
- Though this problem can be solved by the use of Higher-order components (HOC) and render props patterns it results in making the code base inefficient which becomes hard to follow as one ends up wrapping components in several other components to share the functionality.
- Hooks let us share stateful logic in a much better and cleaner way without changing the component hierarchy.

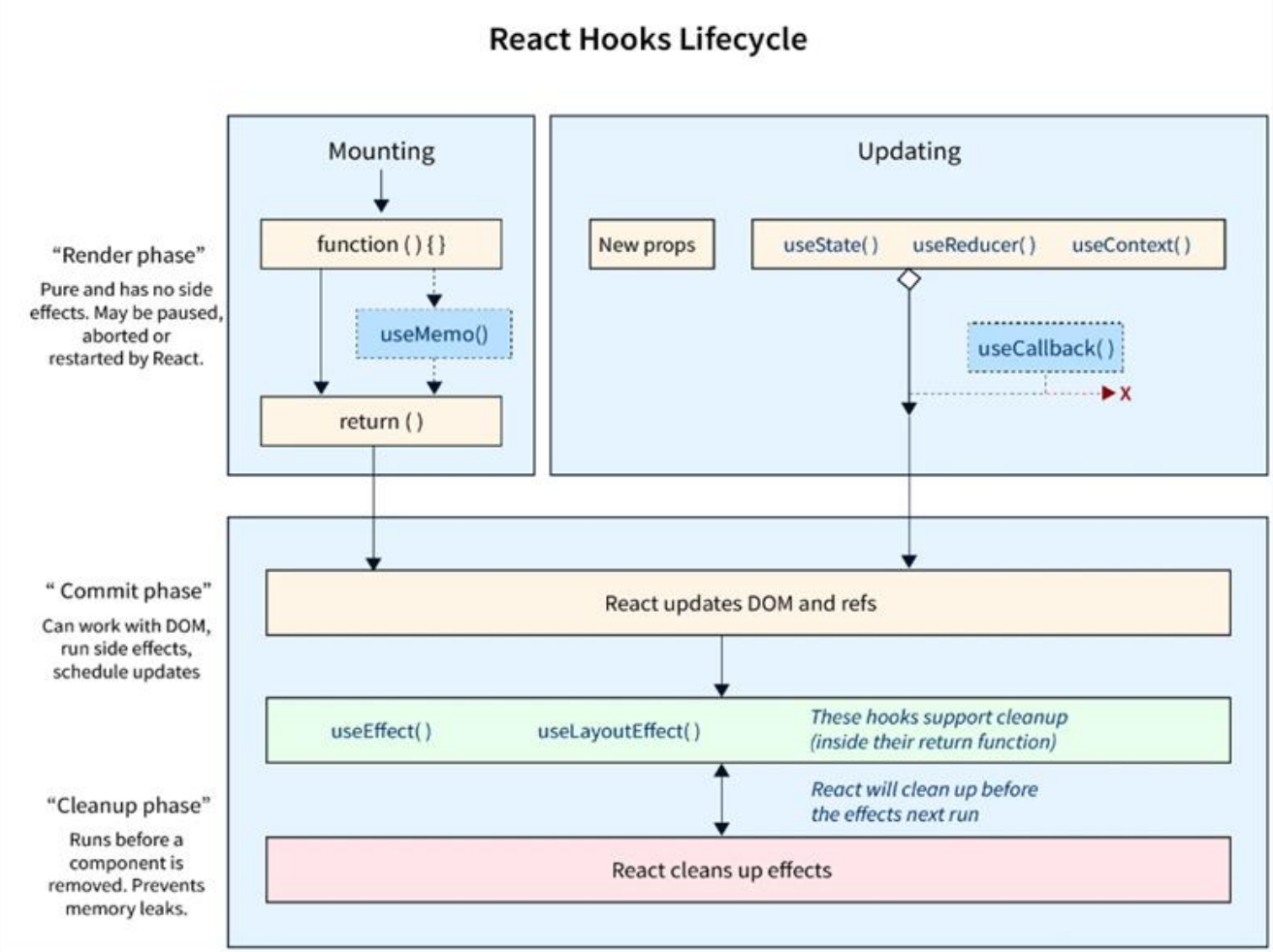
Why use Hooks?

3. Simplifying complex scenarios:

- While creating components for complex scenarios such as data fetching and subscribing to events it is likely that all related code is not organized in one place are scattered among different life cycle methods.
- Hooks solve this problem by rather than forcing a split based on life-cycle method Hooks to let you split one component into smaller functions based on what pieces are related.



Why use Hooks?



Hooks - Example

Re-Write the code in *App.js* inside *src* folder.

What is the useState Hook?

The state of your application is bound to change at some point. This could be the value of a variable, an object, or whatever type of data exists in your component.

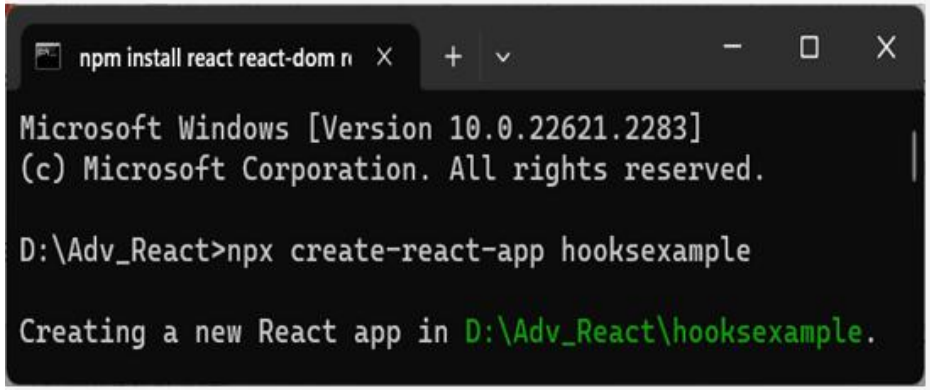
To make it possible to have the changes reflected in the DOM, we have to use a React hook called `useState`. It looks like this:

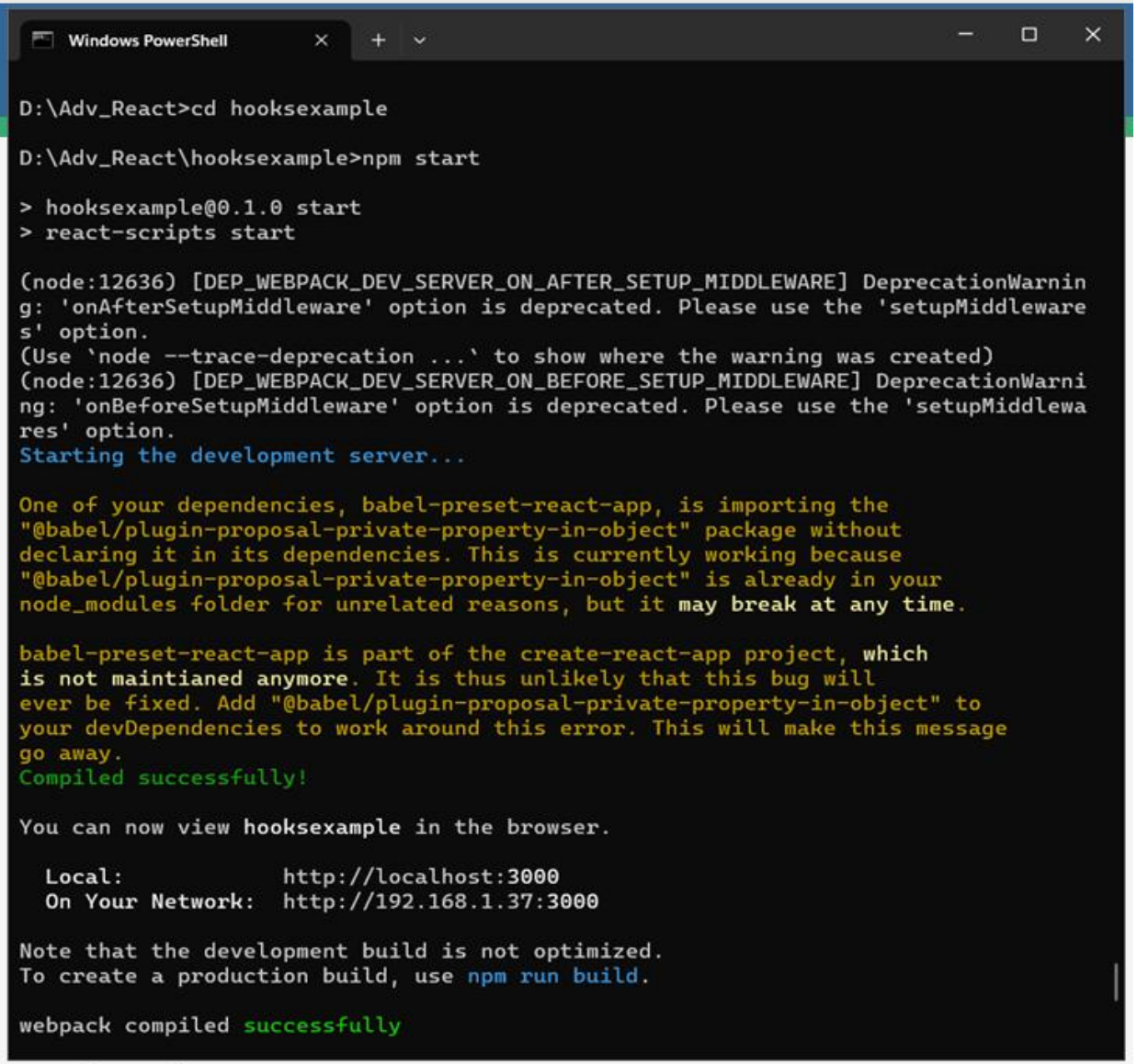
JS App.js ×

D: > Adv_React > hooksexample > src > JS App.js > ...

```
1  import React, { useState } from 'react';
2
3  function CountApp() {
4    // Declare a new state variable, which we'll call "count"
5    const [count, setCount] = useState(0);
6
7    return (
8      <div>
9        <p>You clicked {count} times</p>
10       <button onClick={() => setCount(count + 1)}>
11         Click me
12       </button>
13     </div>
14   );
15 }
16 export default CountApp;
```

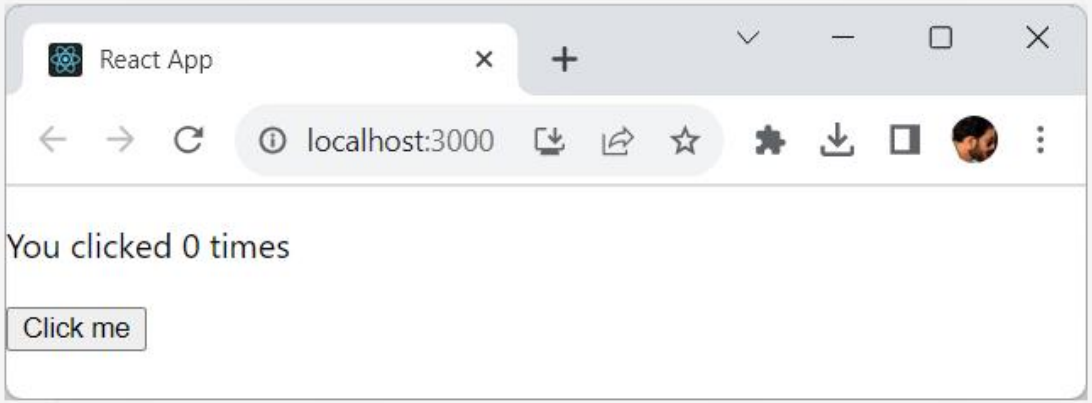
Hooks

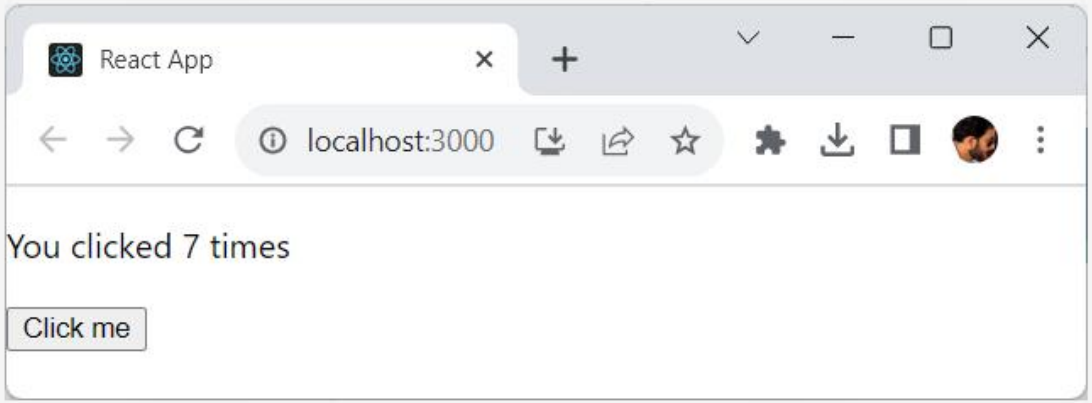




Vijesh Nair

Hooks





React – useEffect Hooks

useEffect

- The useEffect hook is a function in React that allows developers to perform side effects in a functional component.
- This can include things like data fetching, setting up subscriptions, responding to the component's lifecycle events, or updating the DOM in response to changes in state or props.
- The useEffect react hook is called after every render and takes a callback function as an argument, which contains the code for the side effect.
- This allows for a cleaner and more declarative approach to managing side effects in functional components.
- useEffect hook can be used to handle lifecycle events, such as componentDidMount, componentDidUpdate, and componentWillUnmount, in a functional component.
- This hook is called "useEffect" as it is used to perform side effects in a functional component. The term "effect" refers to any changes to the applicable state or behavior that happen as a result of the code in the hook.
- This can include updating the DOM, fetching data, or subscribing to a data source.

How do I Use useEffect in React?

The following is the proper technique to implement the side effect in our User component:

- We import useEffect from “react.”
- In our component, we call it above the returning JSX.
- We supply an array and a function as its two arguments.

Here is the basic syntax:

```
// 1. import useEffect
import { useEffect } from 'react';

function MyComponent() {
  // 2. call it above the returned JSX
  // 3. pass two arguments to it: a function and an array
  useEffect(() => {}, []);

  // return ...
}
```

How do I Use useEffect in React?

Structure of useEffect hook

The useEffect hook accepts two arguments where the second argument is optional

Syntax:

```
useEffect(<FUNCTION>, <DEPENDECY>)
```

We know that the useEffect() is used for causing side effects in functional components and it is also capable of handling componentDidMount(), componentDidUpdate(), and componentWillUnmount() life-cycle methods of class-based components into the functional components.

Ways of controlling side effects in useEffect hook

1. To run useEffect on every render do not pass any dependency

```
useEffect(()->{
    // Example Code
})
```

2. To run useEffect only once on the first render pass any empty array in the dependecy

```
useEffect(()->{
    // Example Code
}, [] )
```

3. To run useEffect on change of a particular value. Pass the state and props in the dependency array

```
useEffect(()->{
    // Example Code
}, [props, state] )
```

useEffect - Example

D: > Adv_React > useeffectexample > src > JS App.js > ...

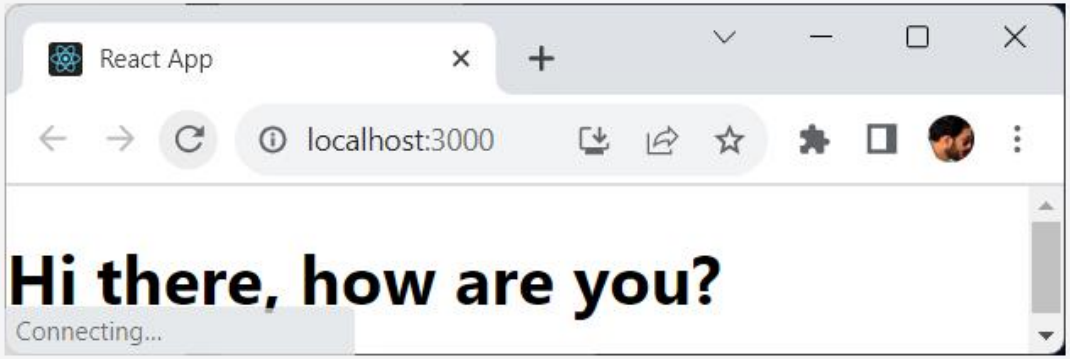
```
1   import React, { useState, useEffect } from 'react';
2
3   const App = () => {
4     const [message, setMessage] = useState('Hi there, how are you?');
5
6     useEffect(() => {
7       console.log('trigger use effect hook');
8
9       setTimeout(() => {
10        setMessage("I'm fine, thanks for asking.");
11      }, 1000)
12    })
13
14    return <h1>{message}</h1>
15  };
16
17  export default App;
```

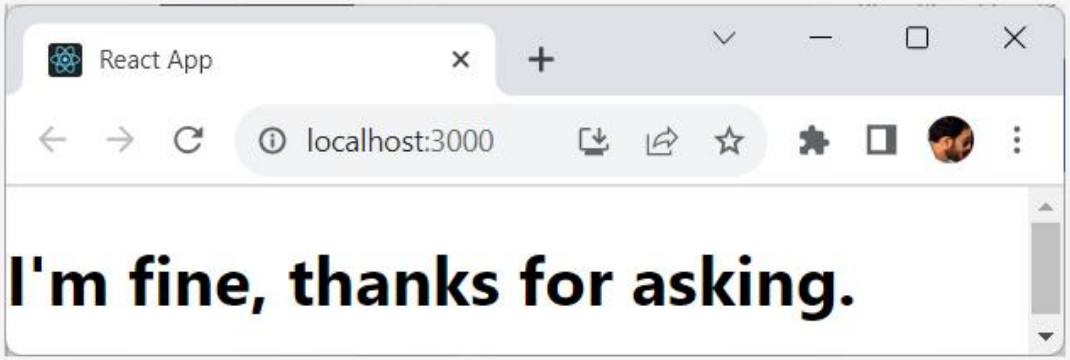
useEffect - Example

```
npm install react react-dom n
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Adv_React>npx create-react-app useeffectexample

Creating a new React app in D:\Adv_React\useeffectexample.
```

```
Windows PowerShell

D:\Adv_React>cd useeffectexample

D:\Adv_React\useeffectexample>npm start

> useeffectexample@0.1.0 start
> react-scripts start

(node:9452) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAf
terSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:9452) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onB
eforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.
Compiled successfully!

You can now view useeffectexample in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.16.18:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

useEffect - Example





React – Flow architecture

React

- React architecture is a collection of components responsible for building a software's User Interface (UI).
- It as an organization of the codebase that helps to build unique project.
- For example, a React architecture will include several UI components such as buttons, forms, API services, Central State Management services, etc.
- Because of this ReactJs architectural freedom, it is the most preferred JavaScript library (framework) for building front-end applications.
- React elements: JavaScript representation of HTML DOM. React provides an API, React.createElement to create React Element.
- JSX: is an XML based, extensible language that supports HTML syntax with slight modifications. JSX can be compiled for React Elements and used to build user interfaces.

React

- Create a new folder (project name) where you want to make your react app using the below command:

```
mkdir newfolder
```

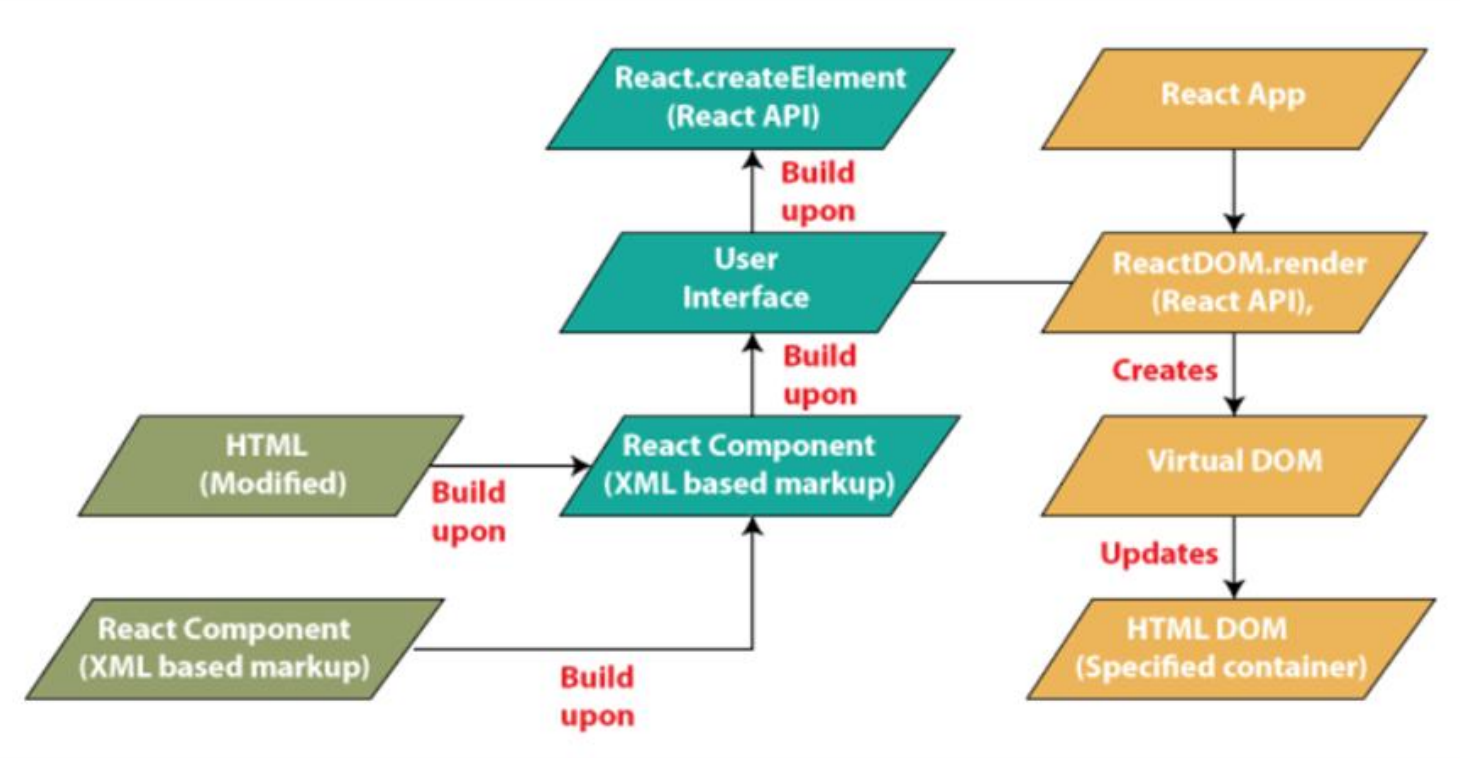
Note: The *newfolder* in the above command is the name of the folder and can be anything

- Move inside the same folder using the below command:

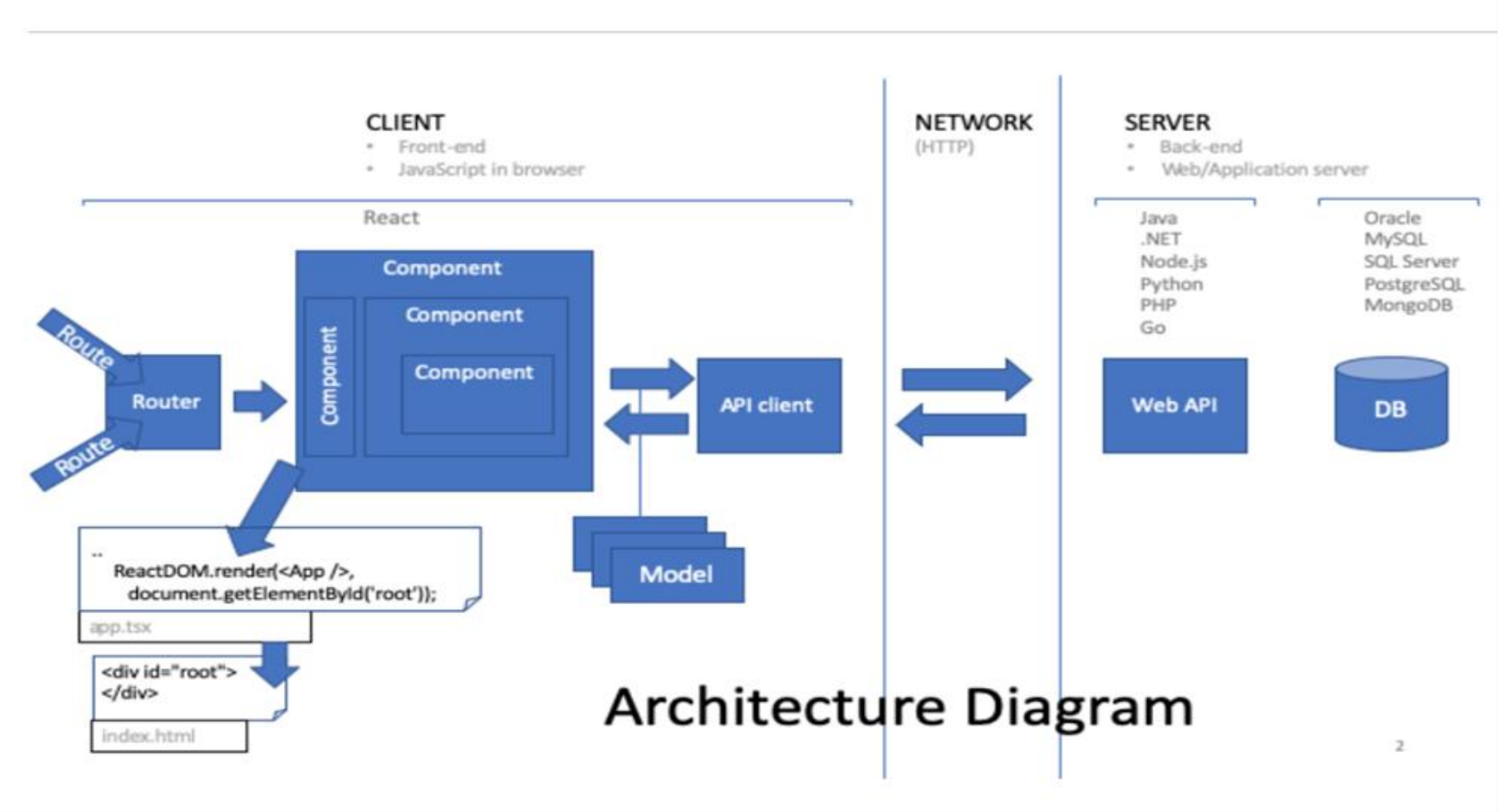
```
cd newfolder (your folder name)
```

- The React app calls the reactdom.render method, bypassing the user interface created using the React component (coded in JSX or React Elementor format) and the container to render the user interface.
- ReactDOM.render processes JSX or React Elements and emits virtual DOM.
- The virtual DOM will be merged and rendered in the container.

React



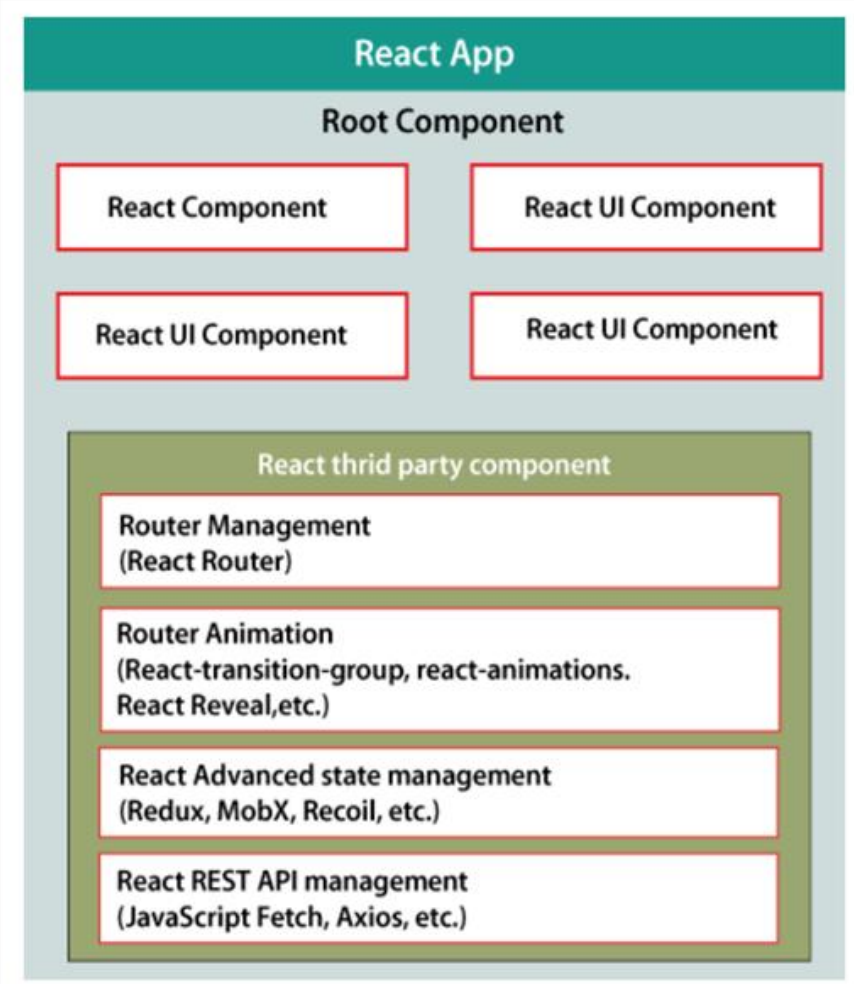
React


CLIENT
Front-end
JavaScript in browser
NETWORK
(HTTP)
SERVER
Back-end
Web/Application server
React
Java
.NET
Node.js
Python
PHP
Go
Oracle
MySQL
SQL Server
PostgreSQL
MongoDB
Component
Component
Component
Component
Route
Route
Router
API client
Web API
DB
Model
ReactDOM.render(<App />,
document.getElementById('root'));
app.tsx

index.html
Architecture Diagram
2

React Application Architecture

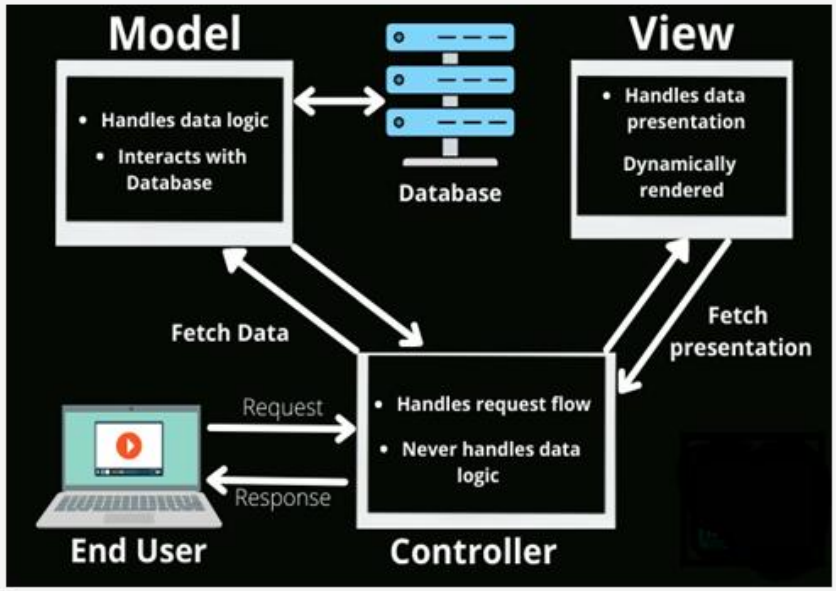
- React app starts with a single root component.
- The core component is created by using one or more components.
- Each component can be nested with any other component at any level.
- Composition is one of the core concepts of the React library. Therefore, each component is built by composing smaller components rather than inheriting one component from another.
- Most of the components are user interface components.
- React apps may include third-party components for a specific purpose, such as routing, animation, state management, etc.



Model-View Controller Framework

What is MVC?

- The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller.



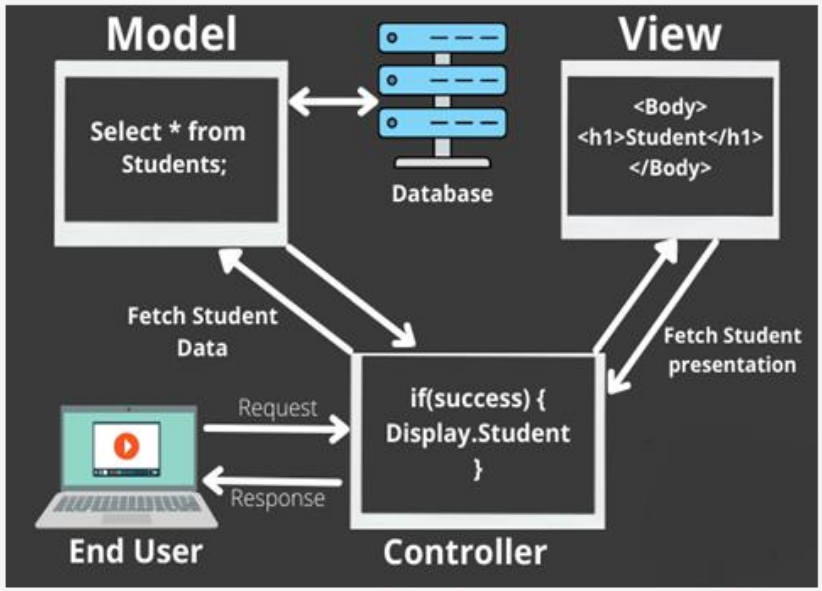
- Each architectural component is built to handle specific development aspects of an application.
- It isolates the business logic and presentation layer from each other.
- It was traditionally used for desktop graphical user interfaces (GUIs).
- Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. It is also used for designing mobile apps.

History of MVC?

- The MVC pattern was first introduced in 1979 by computer scientist Trygve Mikkjel Heyerdahl Reenskaug.
- He wanted to come up with a solution on how to break up a complex user application into smaller manageable components.
- The MVC pattern was first used in the programming language Small Talk.
- One of the original names for the pattern was going to be Model-View-Editor but it was changed to Model-View-Controller.
- Throughout the 1980's and early 90's, the MVC pattern was primarily used in desktop applications. But by the late 1990's, it became pretty popular within web applications.
- In today's web applications, the MVC pattern is a popular design choice for organizing your code.

Working of the MVC Framework

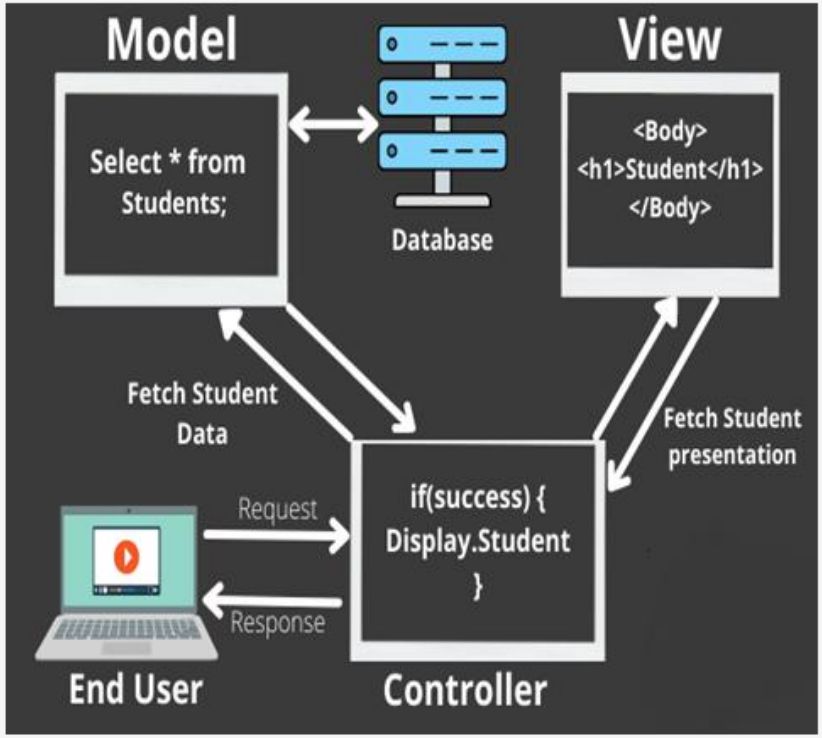
- Let's imagine an end-user sends a request to a server to get a list of students studying in a class.
- The server would then send that request to that particular controller that handles students.
- That controller would then request the model that handles students to return a list of all students studying in a class.



- The model would query the database for the list of all students and then return that list back to the controller.
- If the response back from the model was successful, then the controller would ask the view associated with students to return a presentation of the list of students.
- This view would take the list of students from the controller and render the list into HTML that can be used by the browser.

Working of the MVC Framework

- The controller would then take that presentation and returns it back to the user. Thus ending the request.
- If earlier the model returned an error, the controller would handle that error by asking the view that handles errors to render a presentation for that particular error.
- That error presentation would then be returned to the user instead of the student list presentation.



MVC in React?!

- MVC is an object-oriented programming (OOP) pattern, and React isn't an object-oriented library, is it?
- It's a functional programming (FP) library, JavaScript library from Facebook, that is designed to create interactive UIs.
- React isn't soley OOP or FP; it's a mix of both and that's OK. Not only that, but it fits inside a broader application that can be modeled with either approach.
- *React isn't an MVC framework. React is a library for building composable user interfaces.*
- One key difference between React and MVC is that React is focused specifically on building user interfaces, while MVC is a broader architectural pattern that encompasses the entire application.
- React uses a declarative approach to building user interfaces, which makes it easier to reason about and understand the code, and it also allows for better performance through the use of a virtual DOM (Document Object Model).

MVC in React?!

- Traditionally, in MVC, models and controllers are doing something in the backend.
- React only handles the state management and user interaction in the frontend, which is why in traditional MVC, React is only considered the V.
- Web frameworks that use the MVC pattern include, Ruby on Rails, ASP.NET MVC, Laravel, and Angular.

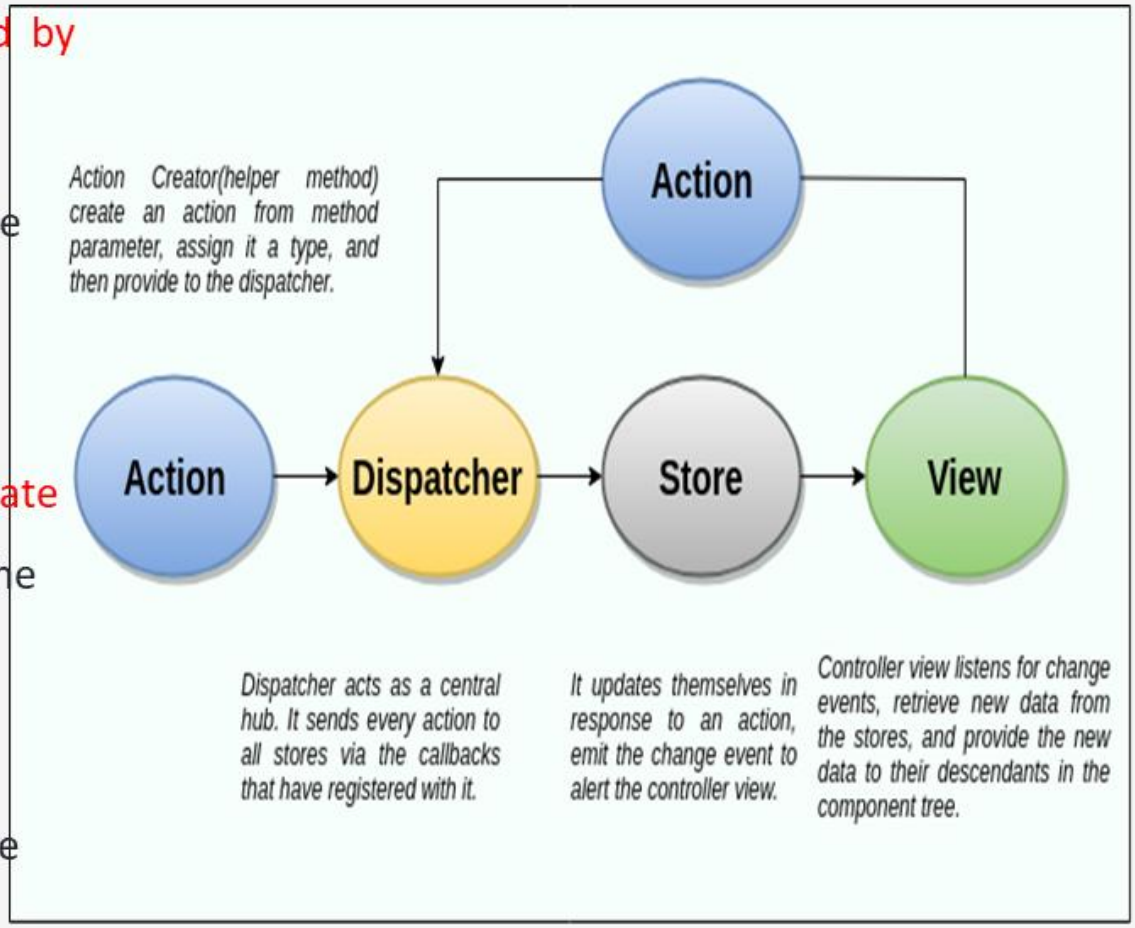
Flux

What is Flux?

- Flux is an application architecture that Facebook uses internally for building the client-side web application with React.
- It is neither a library nor a framework.
- It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model.
- It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.
- It reduces the runtime errors.
- The Facebook team created a data management pattern called Flux after seeing the need for data and state management.
- Flux React is a pattern for managing data flow in React applications.

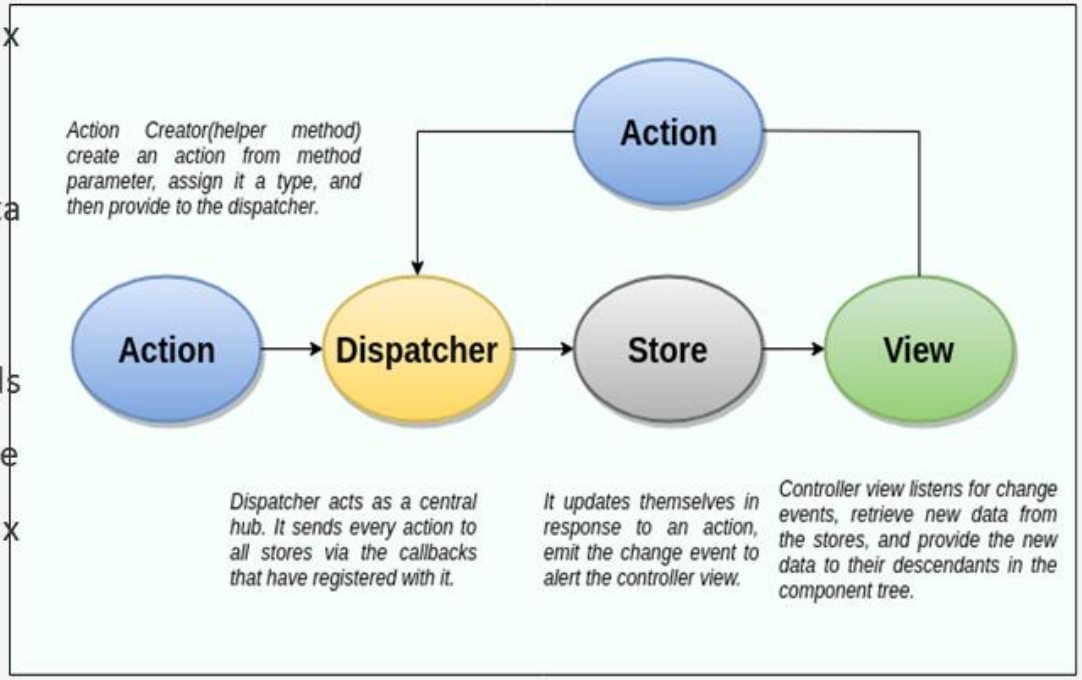
Components of a Flux architecture

- **Actions:** These are the events that are triggered by the user or the system, such as a button click.
- **Dispatcher:** This is the central hub that receives the actions and distributes them to the appropriate stores.
- **Stores:** These are the components that hold the state of the application. They receive the actions from the dispatcher and update the state accordingly.
- **Views:** These are the React components that represent the user interface. They receive the state from the stores and render the appropriate view.



Components of a Flux architecture

- Although, Controllers exists in both MVC & Flux, but Flux controller-views (views) found at the top of the hierarchy.
- It retrieves data from the stores and then passes this data down to their children.
- Additionally, action creators - dispatcher helper methods used to describe all changes that are possible in the application. It can be useful as a fourth part of the Flux update cycle.



- In Flux application, data flows in a single direction(unidirectional). This data flow is central to the flux pattern.
- The dispatcher, stores, and views are independent nodes with inputs and outputs.
- The actions are simple objects that contain new data and type property.

Flux: Dispatcher

- It is a central hub for the React Flux application and manages all data flow of your Flux application.
- It is a registry of callbacks into the stores. It has no real intelligence of its own, and simply acts as a mechanism for distributing the actions to the stores.
- All stores register itself and provide a callback. It is a place which handled all events that modify the store.
- When an action creator provides a new action to the dispatcher, all stores receive that action via the callbacks in the registry.

SN	Methods	Descriptions
1.	register()	It is used to register a store's action handler callback.
2.	unregister()	It is used to unregisters a store's callback.
3.	waitFor()	It is used to wait for the specified callback to run first.
4.	dispatch()	It is used to dispatches an action.
5.	isDispatching()	It is used to checks if the dispatcher is currently dispatching an action.

Flux: Stores – Views - Actions

Stores

- It primarily contains the application state and logic.
- It is similar to the model in a traditional MVC.
- It is used for maintaining a particular state within the application, updates themselves in response to an action, and emit the change event to alert the controller view.

Views

- It is also called as controller-views.
- It is located at the top of the chain to store the logic to generate actions and receive new data from the store.
- It is a React component listen to change events and receives the data from the stores and re-render the application.

Actions

- The dispatcher method allows us to trigger a dispatch to the store and include a payload of data, which we call an action.
- It is an action creator or helper methods that pass the data to the dispatcher.

Bundling the application

What is a bundler in React?

- A bundler in React is a tool that allows developers to package their code into a single file or bundle.
- This bundle can then be used to run the application in the browser.
- Bundlers are used to reduce the size of the code and improve the performance of the application.
- Common bundlers used with React are Webpack and Parcel.

What do bundlers do?

- They allow developers to bundle their code, including libraries and frameworks, into a single file.
- This makes it easier to deploy and run the application, as well as making the code more efficient and maintainable.
- In other words, a bundler is a tool that helps you manage the dependencies and assets of your React application, and package them into a single file (or a few files) that can be served to the browser.

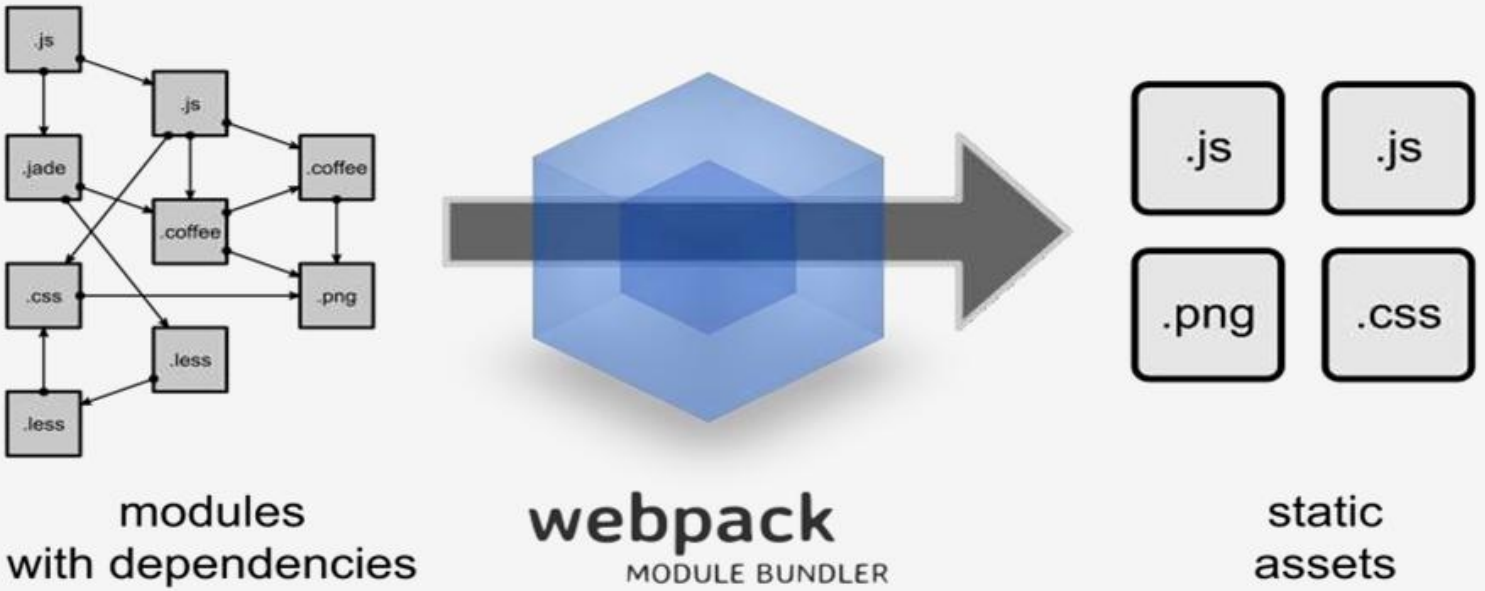
Functions of bundler in React

- **Optimization:** Bundlers allow you to optimize your application by minifying, compressing and tree shaking your code, which can make it faster to load and run.
- **Asset management:** Bundlers can handle different types of assets, such as images, fonts, and styles. It also allow to transpile code like javascript, css.
- **Code splitting:** Bundlers enable you to split your code into multiple chunks that can be loaded on demand, which can make your application more efficient by only loading the code that is necessary for the current user.
- **Modularity:** Bundlers allow you to use a modular approach to your code, by using import/export statements to organize your code in a way that makes it easy to reuse and manage.
- **Cross-browser compatibility:** Some bundlers can provide cross-browser compatibility through a set of loaders, which make sure that the code works well across different browsers.

Web pack

What is Webpack?

- Webpack is a module bundler. It takes modules with dependencies and generates static assets representing those modules.
- You can use webpack for both your client side JS files as well as your server side node JS files.



What does webpack do?

- Manage dependencies
  - require
  - import (e.g. CommonJS, AMD, ES6, etc.)
- Build tasks - convert and preprocess
  - Minify
  - Combine
  - Sass / less conversion
  - Babel transpile
- It combines the build system and module bundling, i.e. instead of building sass and optimizing images in one part of project, and then combining the script files in another, it combines everything in the module itself

Why should I use webpack – The Plus Factor

- **Code Splitting :** Split the dependency tree into chunks and load on demand
  - Maintain the dependency tree
  - Create chunks of the modules and load them from webpack runtime environment
  - Reduce initial loading time
  - Reduce the number of HTTP requests / for HTTP 2.0 use plugins to reduce the chunk size to optimize it for caching
  - Optimize the combined file so as to reduce the load time, while ensuring avoiding of code redundancy

Code Splitting

- It's the idea that you define, in your code, "split points": areas of your code that could be easily split off into a separate file and then **load on-demand**.
- Syntax:

```
// This is a split point
require.ensure([], () => { //use require for AMD
      // All the code in here, and everything that is imported will be in a separate file
      const library = require('some-big-library');
      $('foo').click(() => library.doSomething());
 }, <'name of the chunk'>);
```

Installing Webpack:

```
npm install --save-dev webpack
```

Code Splitting

- A split point creates a chunk for the dependencies.
- Chunk is automatically checked for repeatition of modules and split accordingly.
- A chunk is loaded at runtime as jsonp and is loaded only once.
- This will remove all modules in the vendor chunk from the app chunk. The bundle.js will now contain just your app code, without any of its dependencies. These are in vendor.bundle.js.

```
var webpack = require("webpack"); module.exports = { entry: {
app: "./app.js",
vendor: ["jquery", "underscore", ...], }, output: { filename: "bundle.js" },
plugins: [
new webpack.optimize.CommonsChunkPlugin(/*chunkName=*/"vendor", /* filename=*/ "vendor.bundle.js"
) ] };
```

Features of Webpack

- Entry: We all know webpack makes a dependency graph and the starting of this graph is known as the entry or entry point. From the starting point of the dependency graph, it will follow all the dependencies to know what it has to bundle.
- Output: Output tells webpack where to put the bundles that it had made and what will be its format.
- Loaders: Loaders convert different types of files like images and CSS into a module before adding them to the dependency graph.
- Plugins: Plugins provide functionality. It can provide much functionality like printing something on running the webpack, minifying, optimization of bundles, etc.

Advantages of Webpack

- **It speeds the development journey:**
  - ✓ If we are using webpack then your page does not need to reload fully on having a small change in javascript.
  - ✓ This same benefit can be accessed for CSS if we will use loaders. It also reduced the load time of the website during debugging and because of this, our time can be saved.
- **It removed the problem of overwriting the global variables: E**very file created will become a module. Hence every variable created in this file will be in the local scope. So the problem of overwriting of global variables that we were facing will be solved.

Advantages of Webpack

•**It provides splitting of code:** Since it supports a module system that's why files will be treated as modules.

- ✓ Since the file will be treated as a module that means we can use one file's features into another.
- ✓ So despite having different files, we can access the same benefit. So it actually helps us in splitting our code into different modules.

•**It provides Minification:**

- ✓ Minification means minimizing the code without changing its functionality.
- ✓ It removes all the whitespace, line breaks that are consuming spaces.
- ✓ It also removes unnecessary code and changes the long variable names. So doing this reduces file size and minimizes the code.

•**It supports feature flagging:** Feature flagging is a software engineering approach by which we send code to different environments during feature testing. So webpack not only helps in development, it helps in testing as well.

Thank You!