WAD MODULE-5
PART-A

1.How do you create a React app?

A.

Setting up a React Environment

If you have npx and Node.js installed, you can create a React application by using `create-react-app`.

If you've previously installed `create-react-app` globally, it is recommended that you uninstall the package to ensure npx always uses the latest version of `create-react-app`.

To uninstall, run this command: `npm uninstall -g create-react-app`.

Run this command to create a React application named `my-react-app`:

```
npx create-react-app my-react-app
```

The `create-react-app` will set up everything you need to run a React application.

Run the React Application

Now you are ready to run your first *real* React application!

Run this command to move to the `my-react-app` directory:

```
cd my-react-app
```

Run this command to run the React application `my-react-app`:

```
npm start
```

A new browser window will pop up with your newly created React App! If not, open your browser and type `localhost:3000` in the address bar.

2.Explain JSX with a code example. Why can't browsers read it?

A. JSX is not a valid JavaScript as they are embedded in HTML elements. As JSX is combination of HTML and JavaScript it is not supported by Browsers. So, if any file contains JSX file, Babel transpiler converts the JSX into

JavaScript objects which becomes a valid JavaScript. Thus, browsers understands the code and executes. Browsers can't read JSX because there is no inherent implementation for the browser engines to read and understand them. JSX is not intended to be implemented by the engines or browsers, it is intended to be used by various transpilers to transform these JSX into valid JavaScript code.

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

Example 1

JSX:

```
const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

4.Give a code example to demonstrate embedding two or more components into one.

In the ReactJS Components, it is easy to build a complex UI of any application. We can divide the UI of the application into small components and render every component individually on the web page.

React allows us to render one component inside another component. It means, we can create the parent-child relationship between the 2 or more components.

The below command will help you to

start a new react app.

Next, you have to move to the **testapp** project folder from the terminal.

Create a new **components** folder inside the **src** folder and create two components named child1.jsx and child2.jsx files inside it.

To start the react app, run the below command on your terminal and verify that react app is working fine.

```
npm start
```

5.Give a code example to modularize code in React.

Modularized code is divided into segments or modules, where each file is responsible for a feature or specific functionality. React code can easily be modularized by using the component structure. The approach is to define each component into different files. With each component separated into different files, all we have to do is figure out how to access the code defined in one file within another file. To access one file into another file, React provide the functionality to import and export the files.

Import and Export: It enables us to use code from one file into other locations across our projects, which becomes increasingly important as we build out larger applications.

Export: Exporting a component, or module of code, allows us to call that export in other files, and use the embedded code within other modules.

There are two ways to export code in react:

- **Export Default:** We can use the export default syntax.
- **Named Exports:** We can explicitly name our exports.

Export Default: We can only use export default once per file. The syntax allows us to give any name when we want to import the given module.

Syntax:

```
export default COMPONENT_NAME
```

Named Exports: With named exports, we can export multiple pieces of code from a single file, allowing us to call on them explicitly when we import. And for multiple such exports, we can use a comma to separate two-parameter names within the curly braces.

Syntax:

```
export {CODE1, CODE2}
```

Import: The import keyword enables us to call the modules that we've exported and use them in other files throughout our applications. There are many ways to import the modules in React, and the method that we use depends on how we exported it.

Importing default export: In order to import the default export from a file, we can use only the address and use the keyword import before it, or we can give any name to the import.

Syntax:

```
import ANY_NAME from ADDRESS
```

Importing named exports: Named export code can be imported by giving the name of that module inside curly braces followed by the address of that file containing that module. For multiple modules, we can use a comma to separate two-parameter names within the curly braces.

Syntax:

```
import {Code1, Code2}
```

6.Write a sample code to update the state of a component in React?

The State is an instance of React Component that can be defined as an object of a set of observable properties that control the behavior of the component. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component. The state of a component can be updated during the lifetime.

Generally, there are two types of components in React. The Class Based Components and Functional Components. The method by which we can update the State of a component is different in these two types of

components. We are going to learn them one by one.

Creating react application:

Step 1: Create a React application using the following command:

```
npx create-react-app name_of_the_app
```

Step 2: After creating the react application move to the directory as per your app name using the following command:`cd name_of_the_app`

Update the State of Class-Based Components: Now we are going to learn how to update the state of a class-based component. The steps are discussed below.

- Go inside the App.js file and clear everything.
- At the top of the App.js file import React,{Component} from 'react'.
- Create a Class based component named 'App'. This is the default App component that we have reconstructed.
- Create a state object named text, using this.state syntax. Give it a value.
- Create another method inside the class and update the state of the component using 'this.setState()' method.
- Pass the state object in a JSX element and call the method to update the state on a specific event like button click.

2. Update the State of functional Components: The steps for updating the state of a functional component are given below.

- Clear everything inside the App component of the App.js file.
- At the top of the App.js file import React, {useState} from "react".
- Inside the App component create a state named 'text' using the following syntax. This is the built-in useState method for react functional components. :

```
const [state, setState] = useState({text:'Default value of the text state'});
```

- Pass the 'text' state to the JSX element using '{state.text}' method.
- Update the state on a specific event like button click using the 'setState' method.

7 How do you implement React routing?

Step 1 - Install a React Router

A simple way to install the react-router is to run the following code snippet in the command prompt window.

```
C:\Users\username\Desktop\reactApp>npm install react-router
```

Step 2 - Create Components

In this step, we will create four components. The App component will be used as a tab menu. The other three components (Home), (About) and (Contact) are rendered once the route has changed.

main.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import { Router, Route, Link, browserHistory, IndexRoute } from
'react-router'


class App extends React.Component {

  render() {

    return (

      <div>

        <ul>

        <li>Home</li>
```

```
          <li>About</li>
          <li>Contact</li>
          </ul>
          {this.props.children}
        </div>
    )
  }
}
export default App;

class Home extends React.Component {
  render() {
    return (
      <div>
        <h1>Home...</h1>
      </div>
    )
  }
}
export default Home;

class About extends React.Component {
  render() {
```

```
    return (
      <div>
        <h1>About...</h1>
      </div>
    )
  }
}
export default About;

class Contact extends React.Component {
  render() {
    return (
      <div>
        <h1>Contact...</h1>
      </div>
    )
  }
}
export default Contact;
```

Step 3 - Add a Router

Now, we will add routes to the app. Instead of rendering App element like in the previous example, this time the Router will be rendered. We will also set components for each route.

main.js

```
ReactDOM.render((
  <Router history = {browserHistory}>
    <Route path = "/" component = {App}>
      <IndexRoute component = {Home} />
      <Route path = "home" component = {Home} />
      <Route path = "about" component = {About} />
      <Route path = "contact" component = {Contact} />
    </Route>
  </Router>
), document.getElementById('app'))
```

8.Web Application without Redux

In this article, we will learn about handling state without Redux. There are multiple ways to manage the state without redux in any web app. In this article, we will learn state management by **useState()** hook. It is a react hook that returns two values the state and a function by which we can mutate the state and we can initial state as arguments, in the code below it is an empty array. The component re-renders whenever the state changes. Let's understand this with an example. In this web-app we will maintain the state (which is the tasks array) in the **App** component, pass down the tasks array to the **Task** component to display in a proper format and do the state mutation in the **Tasks** component with the help of function provided by useState() hook.

Syntax:

```
const [state,setState] = useState([]);
```

Creating React Application And Installing Module:

Step 1: Create a React application using the following command.

```
npx create-react-app foldername
```

Step 2: After creating your project folder i.e. foldername, move to it using the following command:

```
cd foldername
```

Step 3: Run the development server by using the following command:

```
npm start
```

9.Web Application with Redux (Using Store and reducers)

Redux is a state managing library used in JavaScript apps. It simply manages the state of your application or in other words, it is used to manage the data of the application. It is used with a library like React.

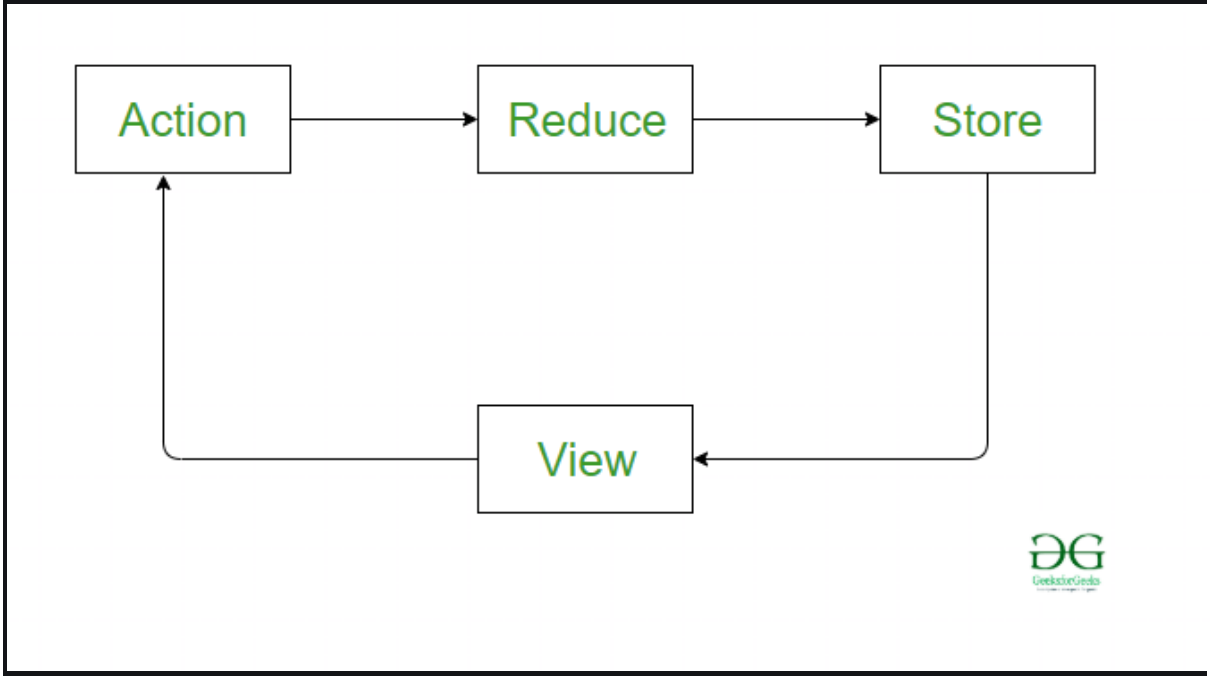
Uses: It makes easier to manage state and data. As the complexity of our application increases. At the start, it is hard to understand but it really helps to build complex applications. In starting, it feels like a lot of work, but it is really helpful.

Before we dive into Redux we should know about some important principles of redux. There are three principles of Redux these are:

- **Single source of truth:** It helps to create universal apps. The state of the application is stored in one object tree called **store**. It means one app, one store.
- **State is read-only (Immutability):** It means that we don't change the state object and its properties directly. Instead of this make a new object, recalculate the new application state and update it with our newly created object. And this is important because all the changes are happening in the same place so everything is needed to be done in strict order and one by one.
- **Changes are made with pure functions (reducers):** Reducers are pure functions that take previous state and action (discuss later) and return the new state.

Building Parts of redux:

1. Action
2. Reducer
3. Store



Redux data flow

Actions: Actions are a plain JavaScript object that contains information. Actions are the only source of information for the store. Actions have a type field that tells what kind of action to perform and all other fields contain information or data. And there is one other term called **Action Creators**, these are the function that creates actions. So actions are the information (Objects) and action creator are functions that return these actions.

Store: The store is the object which holds the state of the application.

Functions associated with Store:

- createStore() – To create a store
- dispatch(action) -To change the state
- getState() – for getting current state of store.

educers: As we already know, actions only tell what to do, but they don't tell how to do, so reducers are the pure functions that take the current state and action and return the new state and tell the store how to do.

10 List some of the cases when you should use Refs.

Following are the cases when refs should be used:

* When you need to manage focus, select text or media playback
* To trigger imperative animations
* Integrate with third-party DOM libraries