Part A:

1)How do you create a React app?
Ans: Create React App is a comfortable environment for learning React, and is the best way to start building a new single-page application in React.

It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You'll need to have Node >= 14.0.0 and npm >= 5.6 on your machine. To create a project, run:

npm create-react-app my-app
cd my-app
npm start

{ since some systems it work's npm in some systems it works npx} .Create React App doesn't handle backend logic or databases; it just creates a frontend build pipeline, so you can use it with any backend you want. Under the hood, it uses Babel and webpack, but you don't need to know anything about them.

When you're ready to deploy to production, running npm run build will create an optimized build of your app in the build folder. You can learn more about Create React App from its README and the User Guide.

---

2)Explain JSX with a code example. Why can't browsers read it?
Ans: React is a JavaScript Library which is used to create web applications. React uses JSX that allows us to write JavaScript objects inside the HTML elements. JSX is neither a string nor HTML. It is a syntax extension of JavaScript. JSX makes code easy and understandable.

```
const num = 20 + 30;
const ele = <h1>{num} is 50</h1>;
```

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.

---

3)Question not there in question bank

---

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

Ans: 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.

Prerequisites: The pre-requisites for this project are:

React Knowledge
Class Components

Creating Application: The below command will help you to start a new react app.

npx create-react-app testapp

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

cd testapp

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

EX:

Child1:-

```
import React, { Component } from 'react';

class Child1 extends Component {
render() {
        return (
        <li>
                This is child1 component.
        </li>
        );
}
}

export default Child1;
```

App.js:-

```
import React, { Component } from 'react';
import Child1 from './components/child1';
class App extends Component {
render() {
        return (
        <div>
                <div>This is a parent component</div>
                <Child1 />
                <Child2 />
        </div>
        );
}
}

export default App;
```

---

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

Ans: 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}

---

6)Write a sample code to update the state of a component in React?
Ans: 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.
1. 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.
Example :

Filename: App.js :-

```
// The App.js file
import React,{Component} from 'react';

class App extends Component {
constructor(){
        super()
        this.state={
        text : 'Welcome to Geeksforgeeks'
        }
}
```

```
goPremium(){
        this.setState({
        text:'Subscription successful'
        })
}
render() {
        return (
                <div>
                        <h1>{this.state.text}</h1>
                        <button onClick={() => this.goPremium()}>
                        Go Premium
                        </button>
                </div>
        );
}
}

export default App;
```

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. The syntax is given below :

```
setState({text:'Updated Content'})
```

Example:

Filename: App.js :-

```
// App.js file
import React, {useState} from "react";

function App(){
        const [state, setState] = useState({
```

```
        text:'Welcome to Geeksforgeeks'
        });
        return (
                <div>
                        <h1>{state.text}</h1>
                        <button onClick={() => setState({
                        text:'Subscription successful'
                        })}>
                        Go Premium
                        </button>
                </div>
        );
};

export default App;
```

---

7)How do you implement React routing.

Ans:- React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

Let us create a simple application to React to understand how the React Router works. The application will contain three components: home component, about a component, and contact component. We will use React Router to navigate between these components.

Installing React Router: React Router can be installed via npm in your React application. Follow the steps given below to install Router in your React application:

Step 1: cd into your project directory i.e geeks.
Step 2: To install the React Router use the following command:

npm install – -save react-router-dom

After installing react-router-dom, add its components to your React application.

Adding React Router Components: The main Components of React Router are:

BrowserRouter: BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.

Routes: It's a new component introduced in the v6 and a upgrade of the component. The main advantages of Routes over Switch are:
Relative s and s
Routes are chosen based on the best match instead of being traversed in order.
Route: Route is the conditionally shown component that renders some UI when its path matches the current URL.
Link: Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.
To add React Router components in your application, open your project directory in the editor you use and go to app.js file. Now, add the below given code in app.js.

```
import {
   BrowserRouter as Router,
   Routes,
   Route,
   Link
} from 'react-router-dom';
```

---

8)Web Application without Redux.
Ans:- There are multiple ways to manage the state without redux in any web app. 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([]);
```

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import StudentList from './StudentList.jsx';
const studentList = [
  {id:1,name:'John Doe',grade:1,school:'React Redux School'},
  {id:2,name:'Jane Doe',grade:2,school:'React Redux School'},
  {id:3,name:'Terry Adams',grade:3,school:'React Redux School'},
  {id:4,name:'Jenny Smith',grade:4,school:'React Redux School'}
];
if (localStorage.getItem("students") === null) {
  localStorage.setItem('students', JSON.stringify(studentList));
}
class App extends Component {
```

```
constructor(props) {
  super(props);
  this.state = {
    studentList: []
  }
  this.editStudentSubmit = this.editStudentSubmit.bind(this);
  this.deleteStudent = this.deleteStudent.bind(this);
  this.addNewStudent = this.addNewStudent.bind(this);
}
componentWillMount() {
  let studentList = JSON.parse(localStorage.getItem("students"));
  this.setState((prevState, props) => ({
    studentList: studentList
  }));
}
addNewStudent() {
  this.setState((prevState, props) => ({
    studentList: [...prevState.studentList, {
      id: Math.max(...prevState.studentList.map(function(o){
        return o.id
      })) + 1,name: '', grade: 1, school: ''
    }]
  }));
}
deleteStudent(id) {
  let r = window.confirm("Do you want to delete this item");
  if (r === true) {
    let filteredStudentList = this.state.studentList.filter(
      x => x.id !== id
    );
     this.setState((prevState, props) => ({
        studentList: filteredStudentList
     }));
     localStorage.setItem(
       'students',
       JSON.stringify(filteredStudentList)
     );
  }
}
editStudentSubmit(id,name, grade, school) {
  let studentListCopy = this.state.studentList.map((student) => {

    if (student.id === id) {
      student.name = name;
```

```
      student.grade = grade;
      student.school = school;
    }
    return student;
  });
  this.setState((prevState, props) => ({
    studentList: studentListCopy
  }));
  localStorage.setItem(
    'students',
    JSON.stringify(studentListCopy)
  );
 }
 render() {
  return (
    <div className="container-fluid">
      <div className="row mt-3"><div className="col-lg-12">
        <div className="card">
          <div className="card-header">
            Student Registry
          </div>
          <div className="card-body">
            <table className="table table-hover">
              <thead
className="thead-dark"><tr><th>Name</th><th>Grade</th><th>School</th><th>Edit/Save</t
h><th>Delete</th></tr></thead>
              <StudentList
                deleteStudent={this.deleteStudent}
                studentList={this.state.studentList}
                editStudentSubmit={this.editStudentSubmit}
              />
            </table>
          <button
            className="btn btn-dark pull-left"
            onClick={this.addNewStudent}>
            Add New
          </button>
        </div>
      </div>
    </div>
   </div>
  </div>
  );
 }
```

```
}
export default App;
```

---

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

Ans: Redux is an open-source JavaScript library for managing application state. Created by Dan Abramov and Andrew Clark, it was released in 2015 and has since found widespread adoption on the client-side for building reactive user interfaces.

Redux is built on three core principles:

Provide a single source of truth for the application state
Ensure application state is read-only and require changes to be made by emitting a descriptive action
Perform state tree transformations based on aforementioned actions by using pure reducer functions
These core principles are realised through the store, reducer, and action concepts.

-

A store maintains the application state. If we want to use data from the application state, we read it through an interface provided to us by the store. If we want to change the application state, we dispatch an action into the store to trigger the change. If we want to get notified when a change happens, we subscribe to the store with a listener function. The store is our single source of truth for the current state of the application itself.

A reducer is a pure function that performs state transformations — it is the building block of a store. Therefore, in order to create a store we must first describe a reducer. Reducer functions take the state tree and transform it in all manner of ways, depending on the action dispatched into the store. Reducers do not need to change the state for all actions, but no matter what action they take (or don't), a reducer will always return a new revision of the application state.

---

10) List some of the cases when you should use Refs.
Ans: There are a few good use cases for refs:

Managing focus, text selection, or media playback.
Triggering imperative animations.
Integrating with third-party DOM libraries.
Avoid using refs for anything that can be done declaratively.

For example, instead of exposing open() and close() methods on a Dialog component, pass an isOpen prop to it.
