

## REACT.JS - ASSESSMENT(submitted by Sravanthi kommineni)

Email:kommins03@gmail.com

### Part 1 - Basic Knowledge

#### 1.What is React.Js?

**Ans:**React is javascript library which is free and open source for building UI(User Interface) components.It is widely used in web development .React mostly combines basic HTML and Javascript concepts with some beneficial additions.It can be used to develop both web development and mobile apps.There is framework called React native derived from react itself.

#### 2.what are key features of React.Js?

**Ans:**

→**Components** are the building blocks of any react application and a single app contains multiple components and they have their own logic and these components are reused which in turn dramatically reduces the application's development time .

→React Js is known for great performance and reason behind it is **virtual DOM**

→It is user friendly,readable and easy to debug

→It follows **unidirectional data** binding mechanism meaning data flows only in one direction in library ,data is transformed from top to bottom i.e from parent component onto child component

#### 3.what is Jsx?

**Ans:**

JSX stands for "JavaScript eXtension" and it is a syntax extension used in the React JavaScript library to allow developers to write HTML-like code within JavaScript. JSX allows developers to write code that mixes HTML and JavaScript in a familiar and intuitive way, making it easier to build dynamic and interactive user interfaces.

When a JSX code is compiled, it gets transformed into regular JavaScript code that can be understood and executed by the browser. This process is called "transpiling", and it's typically done using a tool like Babel.

In summary, JSX is a powerful and convenient syntax extension that simplifies the creation of dynamic user interfaces in React by allowing developers to write HTML-like code within JavaScript.

#### 4.what is a virtual DOM?

**Ans:**

The Virtual DOM is a concept used in React to optimize the process of updating the user interface. It's essentially a lightweight copy of the actual DOM (Document Object Model), which is a tree-like structure representing the HTML elements in a web page.

When changes are made to the user interface, instead of directly manipulating the actual DOM, React updates the Virtual DOM first. This process is faster and less resource-intensive than updating the actual DOM directly. React then compares the updated Virtual DOM with the previous version to identify the changes that have been made.

Once the changes are identified, React updates the actual DOM with only the necessary changes, minimizing the amount of work that needs to be done and reducing the time it takes to update the user interface. This process is known as "reconciliation".

By using the Virtual DOM, React is able to update the user interface in an efficient and optimized manner, while still providing a familiar and intuitive programming model for developers

## **5.What is the difference between props and state?**

**Ans:**

In React, props and state are two different ways of managing data within a component.

### **Props:**

Props (short for "properties") are used to pass data from a parent component to a child component. They are read-only, meaning that the child component cannot modify the props it receives. Props are passed to a component as attributes, and they are accessed within the component using the `this.props` object.

Here's an example of passing props from a parent component to a child component:

```
// Parent component
function Parent() {
  return <Child name="Alice" age={25} />;
}
```

```
// Child component
function Child(props) {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}
```

In this example, the Parent component passes two props (name and age) to the Child component. The Child component receives these props as the `props` object and displays their values.

### **State:**

State is used to manage data that can change within a component. Unlike props, state is mutable, meaning that a component can modify its own state. State is initialized within the component's constructor and accessed using the `this.state` object.

Here's an example of using state in a component:

```

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>Increment</button>
      </div>
    );
  }
}

```

In this example, the Counter component uses state to keep track of a count value, which is displayed in the component's render method. The component also includes a button that, when clicked, increments the count value by calling the setState method.

In summary, props are used to pass data from a parent component to a child component, while state is used to manage data that can change within a component

## 6.What is the role of Redux in a React.js application?

**Ans:**

Redux is a state management library that is often used in conjunction with React to manage the state of an application. The role of Redux in a React.js application is to provide a predictable and centralized way of managing the application state, making it easier to reason about, test, and debug.

Here are some of the key roles of Redux in a React.js application:

**Centralized state management:** Redux provides a single store where all the application state is stored, making it easier to manage and track changes to the state.

**Predictable state management:** Redux enforces a unidirectional data flow, which makes it easier to understand how state changes are propagated through the application.

**State immutability:** Redux requires that state be immutable, which means that any changes to the state must be made by creating a new copy of the state. This helps prevent bugs and makes it easier to track changes to the state.

**Easy integration with React:** Redux is designed to work well with React, and there are libraries and tools available to make it easy to integrate the two.

Time travel debugging: Redux provides a powerful feature called time travel debugging, which allows you to step through the history of the application state and see how it has changed over time. This can be a powerful tool for debugging complex applications.

Overall, the role of Redux in a React.js application is to provide a centralized and predictable way of managing the application state, making it easier to build complex and scalable applications.

## **7.What is the purpose of React Router?**

### **Ans:**

React Router is a popular library for routing in React applications. The purpose of React Router is to provide a declarative way to handle client-side routing in a single-page application built with React.

Here are some of the main purposes of React Router:

Declarative routing: React Router allows you to declare the routes of your application using a declarative syntax. This makes it easier to understand and manage the routing of your application.

Client-side routing: React Router is designed to work with client-side routing, which means that it handles routing within the browser without requiring a page refresh. This provides a smoother user experience and can improve the performance of your application.

Nested routing: React Router supports nested routing, which allows you to define routes within routes. This makes it easier to organize and manage the routing of complex applications.

Route parameters: React Router allows you to define parameters in your routes, which can be used to pass dynamic data to your components. This can be useful for building dynamic pages and handling user input.

History management: React Router provides a history object that allows you to manage the browser history and navigate programmatically. This can be useful for handling user interactions, such as clicking back or forward buttons in the browser.

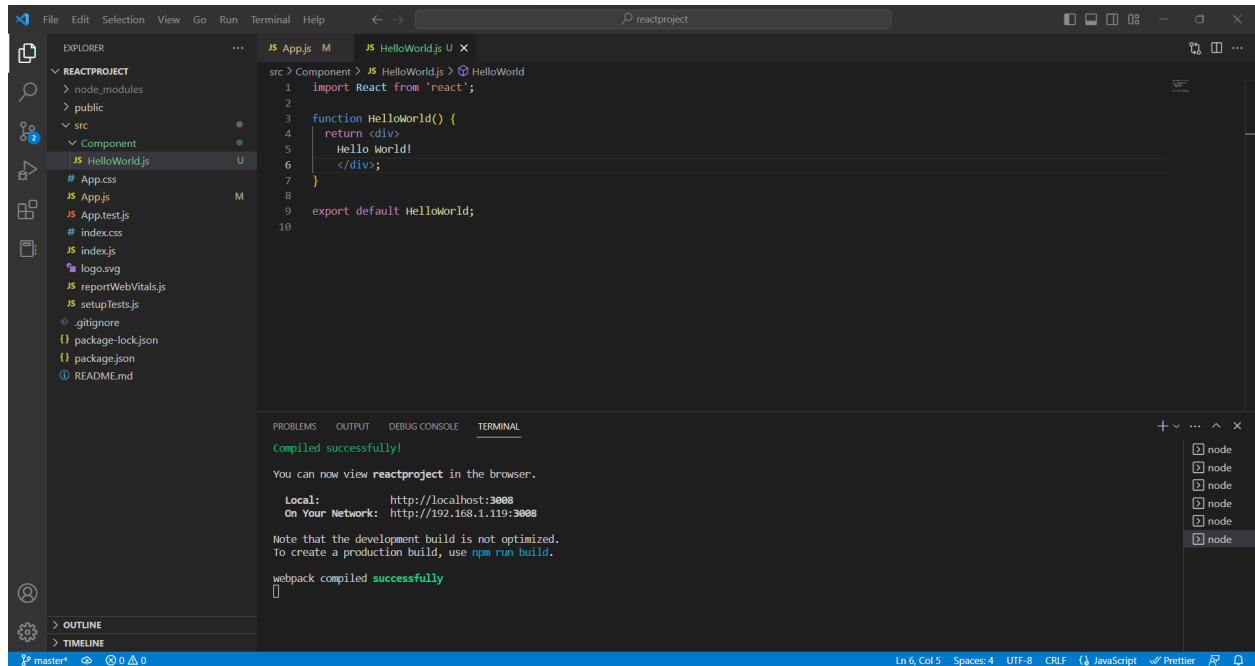
Overall, the purpose of React Router is to provide a declarative and flexible way to handle client-side routing in React applications. It simplifies the process of building complex applications with multiple routes and dynamic content, while providing a smooth and responsive user experience

## Part 2 - Code Implementation:

### 8. Create a simple React component that displays "Hello World!" as text.

Ans:

Here I am Attaching the screenshot my code implementation



The screenshot shows the VS Code editor with a project named 'reactproject'. The Explorer sidebar on the left shows the file structure, including a 'src' directory with a 'Component' folder containing 'HelloWorld.js'. The main editor window displays the code for 'HelloWorld.js':

```
src > Component > JS HelloWorld.js > HelloWorld
1  import React from 'react';
2
3  function HelloWorld() {
4    return <div>
5      Hello World!
6    </div>;
7  }
8
9  export default HelloWorld;
10
```

Below the code editor, the TERMINAL panel shows the output of running the application:

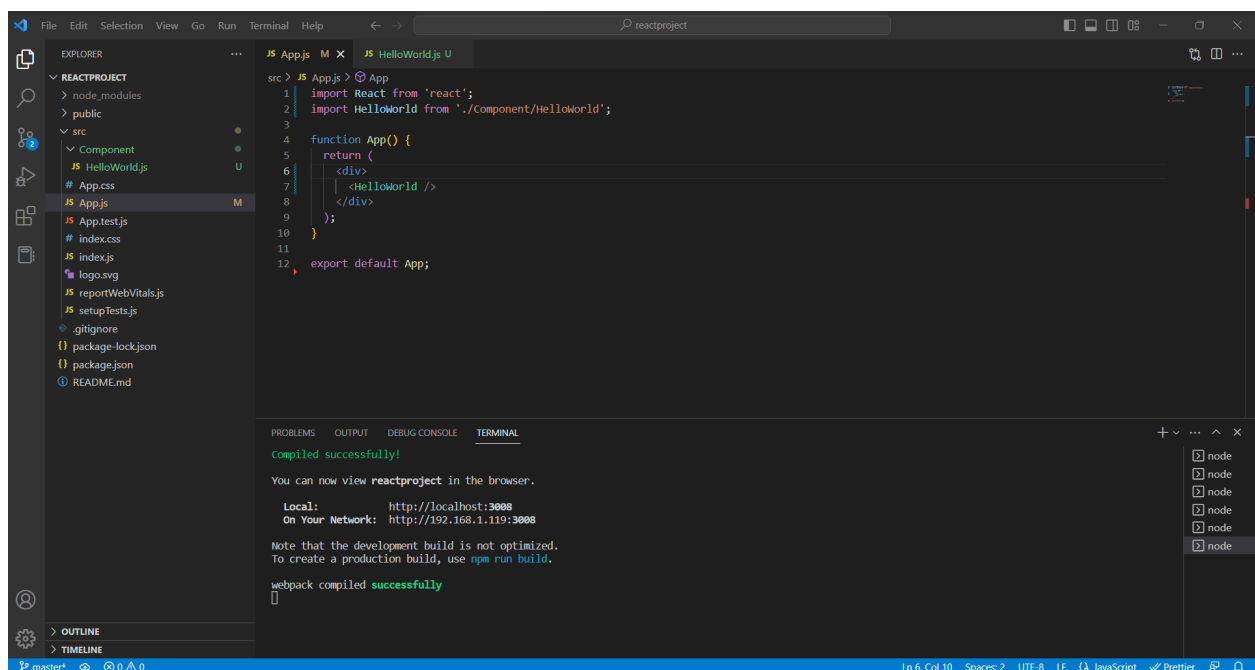
```
Compiled successfully!

You can now view reactproject in the browser.

Local:      http://localhost:3008
On Your Network:  http://192.168.1.119:3008

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

webpack compiled successfully
```



The screenshot shows the VS Code editor with the same project. The Explorer sidebar shows the file structure, including a 'src' directory with a 'Component' folder containing 'HelloWorld.js'. The main editor window displays the code for 'App.js':

```
src > JS App.js > App
1  import React from 'react';
2  import HelloWorld from './component/HelloWorld';
3
4  function App() {
5    return (
6      <div>
7        <HelloWorld />
8      </div>
9    );
10 }
11
12 export default App;
```

Below the code editor, the TERMINAL panel shows the output of running the application:

```
Compiled successfully!

You can now view reactproject in the browser.

Local:      http://localhost:3008
On Your Network:  http://192.168.1.119:3008

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

webpack compiled successfully
```



---

### 9. Create a form component with input fields for name and email.

#### Ans:

Here I have created the component called Form where I have created the input fields for name and email

#### Explanation:

I have defined a functional component called Form that uses the useState hook to manage the state of the name and email input fields. Also given handleSubmit function that will be called when the form is submitted.

The return statement contains the JSX code that defines the input fields and the submit button. Here I have used the htmlFor attribute instead of for to associate the label with the input field in React.

value and onChange attributes to link the input fields to the corresponding state variables. This way, when the user types in the input field, the state variable is updated and the value of the input field is updated automatically.

Finally, I have added an onSubmit event listener to the form element that calls the handleSubmit function when the form is submitted. In the handleSubmit function, can perform any further processing of the form data that is required

#### #code:

```

import React, { useState } from 'react';

function Form() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(`Name: ${name}, Email: ${email}`);
    // Here you can do further processing with the form data.
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="name">Name:</label>
      <input type="text" id="name" name="name" value={name} onChange={(e) =>
setName(e.target.value)} />

      <label htmlFor="email">Email:</label>
      <input type="email" id="email" name="email" value={email} onChange={(e) =>
setEmail(e.target.value)} />

      <input type="submit" value="Submit" />
    </form>
  );
}

export default Form;

```

## 10. Implement a React component that displays a list of items.

**Ans:**

**Explanation:**

Here I have defined a functional component called **ItemList** that accepts a props object as an argument. The props object contains a items property that is an array of items to be displayed in the list.

The return statement contains the JSX code that defines the unordered list (<ul>) and the list items (<li>). Used map method to loop through the items array and create a list item for each item in the array. key attribute to assign a unique key to each list item, which is required by React to optimize rendering performance.

Finally, export the ItemList component so that it can be used in other parts of our application.

**#code**

```
import React from 'react';

function ItemList(props) {
  const { items } = props;

  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}

export default ItemList;
```

To use this component in another component or in the App.js file, we can import the ItemList component and pass an array of items as a prop:

**#Code:**

```
import React from 'react';
import ItemList from './ItemList';

function App() {
  const items = ['Item 1', 'Item 2', 'Item 3'];

  return (
    <div>
      <h1>My Item List</h1>
      <ItemList items={items} />
    </div>
  );
}

export default App;
```

11. Use React Router to create a multi-page application with a navigation menu.

**Ans:**

Firstly, I have installed React Router by running the following command in my terminal:

**npm install react-router-dom**

Next, in my index.js file, I have imported BrowserRouter from react-router-dom and wrapped <App /> component with it:

**#code:**



```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
```

```
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);
```

Then In App.js file, I have created a navigation menu with links to different pages using Link components from react-router-dom.

**#code:**

```
import React from 'react';
import { Route, Link } from 'react-router-dom';
import HomePage from './HomePage';
import AboutPage from './AboutPage';
import ContactPage from './ContactPage';
```

```
function App() {
  return (
    <div>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Route exact path="/" component={HomePage} />
      <Route path="/about" component={AboutPage} />
      <Route path="/contact" component={ContactPage} />
    </div>
  );
}
```

```
    </div>  
  );  
}
```

```
export default App;
```

Here I have three pages: HomePage, AboutPage, and ContactPage. When a user clicks on a link in the navigation menu, they will be directed to the corresponding page. The exact prop on the first Route component ensures that the HomePage component is only rendered when the URL exactly matches "/".

### **Part 3 - Advanced Concepts:**

#### **12.Explain the concept of React hooks.**

##### **Explanation:**

React Hooks is a feature in React that allows developers to use stateful logic and other React features in functional components. Previously, stateful logic was only possible in class components using the "state" keyword.

Hooks are called "hooks" because they allow developers to "hook into" React features from within functional components. Each hook is a JavaScript function that starts with the prefix "use" (e.g., useState, useEffect, useContext).

The useState hook, for example, allows you to add state to a functional component. You can call useState with an initial value and it will return an array with two values: the current state and a function to update that state. You can then use these values in your component just like you would with a class component's state.

The useEffect hook allows you to perform side effects in a functional component. For example, you can use useEffect to fetch data from an API when the component mounts or update the document title based on the component's state.

Hooks are powerful because they allow developers to write more concise and reusable code. They also make it easier to share stateful logic between components without using higher-order components or render props.

In summary, React Hooks are a way to use stateful logic and other React features in functional components, allowing developers to write more concise and reusable code.

#### **13. What are higher-order components in React and how are they useful?**

##### **Explanation:**

Higher-order components (HOCs) are a design pattern in React that allow you to wrap one component with another to share behavior between components. HOCs are functions that take in a component as an argument and return a new component with added functionality.

HOCs are useful for a number of reasons. One reason is that they help with code reuse. Instead of duplicating code across multiple components, you can create a higher-order component that wraps those components and shares the necessary behavior. This can make your code more modular and easier to maintain.

HOCs are also useful for abstracting away complex behavior into a single component. For example, you might create an HOC that handles authentication logic and wraps around components that require authentication. This can simplify the code for those components, since they don't have to worry about the authentication logic themselves.

Another advantage of HOCs is that they can provide additional data or behavior to components. For example, you might create an HOC that provides a component with data from an external API or with certain props that are used throughout your application.

In summary, higher-order components are a design pattern in React that allow you to wrap one component with another to share behavior between components, improve code reuse, abstract away complex behavior, and provide additional data or behavior to components.

#### **14. Explain the difference between server-side rendering and client-side rendering in React**

##### **Explanation:**

Here I will explain with an example Imagine you are a chef preparing a dish for your guests.

In server-side rendering, you prepare the entire dish in your kitchen before serving it to your guests. This means that the server (the kitchen) generates the HTML for the entire page, including all the content and data. When the page is served to the client (the guest), it is already complete and ready to be consumed. This approach is typically faster for the first load, as the server is responsible for generating the entire page before sending it to the client.

In client-side rendering, you present your guests with a set of ingredients and instructions on how to prepare the dish themselves. This means that the server only sends the bare minimum HTML, CSS, and JavaScript required to display the initial view of the page. The client's browser (the guest) then downloads additional JavaScript code and fetches data from the server to dynamically update the page as the user interacts with it. This approach is typically slower for the first load, as the client has to download and process additional code before the page is fully functional.

In the context of React, server-side rendering (SSR) involves using a server to render the initial view of a React application and serve it to the client as static HTML. This improves performance, as it reduces the time it takes for the user to see content on the page. Client-side rendering (CSR), on the other hand, involves loading a minimal HTML file and using JavaScript to render and update the application on the client's browser. This can lead to a better user

experience, as the page is more interactive and responsive, but it can also result in slower initial load times.

## 15. How would you optimize the performance of a React application?

### Explanation:

To optimize the performance of a React application is to implement code splitting. Code splitting involves dividing the application code into smaller chunks that are loaded on-demand instead of loading the entire application code upfront. This reduces the initial load time and improves the application's perceived performance.

To implement code splitting in a React application, you can use a bundler like Webpack or Parcel, which supports code splitting out of the box. Here's how you can implement code splitting using Webpack:

Configure Webpack to use the SplitChunksPlugin to split the application code into multiple chunks.

Use dynamic imports to import modules on-demand in the application code.

Use `React.lazy()` and `Suspense` to lazily load components when they are needed.

Here's an example of how to use `React.lazy()` and `Suspense` to lazily load a component:

### #code:

```
import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}
```

By implementing code splitting, we can significantly improve the performance of a React application, especially for larger applications with many components and dependencies.

## 16. What is the role of context in React?

### Explanation:

Context in React is a mechanism that enables sharing data between components without having to pass the data through props manually at every level of the component tree. It provides a way to pass data down from a parent component to its descendants, even if the descendants are not directly connected.

Context consists of two parts: a Provider component and a Consumer component. The Provider component provides the data that needs to be shared and the Consumer component consumes the data

Example:

```
// Create a context
```

```
const MyContext = React.createContext();
```

```
// Create a provider component
```

```
function MyProvider(props) {
```

```
  const [data, setData] = useState('initial data');
```

```
  return (
```

```
    <MyContext.Provider value={{ data, setData }}>
```

```
      {props.children}
```

```
    </MyContext.Provider>
```

```
  );
```

```
}
```

```
// Create a consumer component
```

```
function MyConsumer() {
```

```
  return (
```

```
    <MyContext.Consumer>
```

```
      ({ data, setData }) => (
```

```
        <div>
```

```
          <p>Data: {data}</p>
```

```
          <button onClick={() => setData('new data')}>Update Data</button>
```

```
        </div>
```

```
      )
```

```
    </MyContext.Consumer>
```

```
  );
```

```
}
```

```
// Use the context in a component tree
```

```
function App() {
```

```
  return (
```

```
    <MyProvider>
```

```
      <div>
```

```
        <h1>My App</h1>
```

```

        <MyConsumer />
    </div>
</MyProvider>
);
}

```

In this example, the MyProvider component provides the data and the MyConsumer component consumes the data. Any descendant component of the MyProvider component can access the data using the MyContext.Consumer component.

Context can be used to pass global data, such as a user's authentication status, or theme information, throughout an application

#### **Part 4 - Real-world Application:**

##### **17. Describe a React.js project you have worked on and your contribution to it.**

**Ans:**

I have worked in Dentridge IT solutions for 2 years . During this period, I have worked for client JVB wellness. JVB wellness. Project is about to build an application which helps people to improve their Health and Quality of life through a targeted approach of personal transformation. JVB offers services like marathon, Triathlon, general health & wellness coaching, data review & analysis and certification programmers. Data reviews and analysis of your workouts, nutrition, sleep, movement and race results. We will get the data from the Health watches like Fitbit, Garmin, Apple, etc. We integrated all those watches on the web and as well in mobile applications. We designed the dashboard where, end-users can view their report summary in leader board and also available for download in the excel format

##### **My Roles and Responsibilities:**

- Developed an appropriate database and collected all required information within the required timeframe and Involved in Requirement Analysis, Client communication, Design, Coding and Deployments.
- Responsible for development of new highly-responsive, web-based user interface
- Construct visualizations that are able to depict vast amounts of data
- Creating RCA (Root Cause Analysis) Documentation.
- Reviewed all technologies and assisted in set up
- Developed a flexible and well-structured front-end architecture, along with the APIs to support it and Documented all database design as per user requirement
- Coordinated and maintained professional relationships with all External clients
- Maintained Knowledge on various technologies and maintained adaptability to changing market

##### **18. How did you ensure the project was maintainable and scalable?**

**Ans:**

I can provide some best practices that can helped me ensure our project maintainability and scalability:

**Modularity:** Breaking down the project into smaller, independent modules that can be easily tested, maintained, and updated separately.

**Code readability:** Writing clean, understandable, and well-organized code that is easy to read, debug, and modify.

**Documentation:** Creating clear and concise documentation that explains the code's purpose, structure, and function to ensure ease of understanding.

**Testability:** Implementing automated testing frameworks and unit testing for all code components to ensure that the project works as intended and can be tested effectively.

**Scalability:** Designing the project with scalability in mind, such as building it on a scalable architecture, and implementing load-balancing, caching, and other techniques that allow it to handle increased traffic and user load.

**Maintainability:** Using version control systems such as Git, maintaining up-to-date dependencies, and adhering to coding standards to ensure the codebase remains maintainable and up-to-date over time.

**Code reviews:** Conducting regular code reviews to identify and correct any issues, ensure code quality, and ensure that code changes do not negatively impact the project's stability or scalability.

**19. What challenges did you face while working on the project, and how did you overcome them?**

**Explanation:**

While working on project,our client asked us to develop a feature where user can download an excel sheet which provides all the Health Tracking Data.so to develop that we used a library which is not effective (open pyxl).So our team faced an lot of issues regarding.I tool an initiative and communicated developer through github and Explained everything and requested them to add a new feature which supports to download the excel

## **BASIC TEST**

### **Building a To-Do List**

Here is the React code first Task:

As this is basic project where there are no complications and creating components is not required so I have done without using components.Also I am attaching screenshot of the website and also Git repo,I have explained every point with the screenshot I have attached

Here is the React code first Task:

```
import { useState } from "react";
import styled from "styled-components";
import "../index.css";
const Container = styled.div`
  display: flex;
  align-items: center;
  flex-direction: column;
`;
const Button = styled.button`
  display: inline-block;
  flex: 1;
  border: none;
  background-color: teal;
  color: white;
  height: 30px;
  width: 50px;
  border-radius: 2px;
  cursor: pointer;
`;
const Text = styled.input`
  border: 2px solid #000;
  width: 200px;
  padding: 5px;
  border-radius: 2px;
  margin: 5px;
`;
const TaskCount = styled.span`
  margin: 10px;
`;
const Tasks = styled.div`
`;
const LIST = styled.li`
  listStyle: "none";
  text-decoration: "line-through";
`;
const App = () => {
  const [input, setInput] = useState("");
  const [completedTaskCount, setCompletedTaskCount] = useState(0);
  const [todoList, setTodoList] = useState([]);
```



```

const handleClick = () => {
  const id = todoList.length + 1;
  setTodoList((prev) => [
    ...prev,
    {
      id: id,
      task: input,
      complete: false,
    }
  ]);
  setInput("");
};

const handleComplete = (id) => {
  let list = todoList.map((task) => {
    let item = {};
    if (task.id === id) {
      if (!task.complete){
        //Here Task is pending, modifying it to complete and increment
the count
        setCompletedTaskCount(completedTaskCount + 1);
      }
      else {
        //Here Task is complete, modifying it back to pending,
decrement Complete count
        setCompletedTaskCount(completedTaskCount - 1);
      }
    }
    item = { ...task, complete: !task.complete };
  } else item = { ...task };
return item;
  });
  setTodoList(list);
};

return (
  <Container>
    <div>
      <h2>Todo List :What is Today's Plan?</h2>
      <Text value={input} onInput={(e) =>setInput(e.target.value)} />
      <Button onClick={() => handleClick()}>Add</Button>
      <Tasks>
        <TaskCount>

```

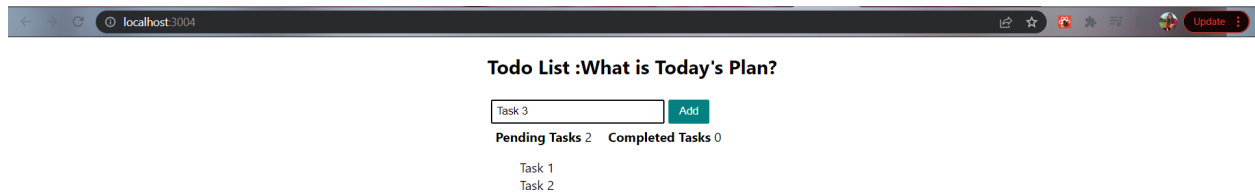
```

        <b>Pending Tasks</b> {todoList.length - completedTaskCount}
      </TaskCount>
      <TaskCount>
        <b>Completed Tasks</b> {completedTaskCount}
      </TaskCount>
    </Tasks>
  </div>
  <ul>
    {todoList.map((todo) => {
      return (
        <LIST
          complete = {todo.complete}
          id={todo.id}
          onClick={() => handleComplete(todo.id)}
          style={{
            listStyle: "none",
            textDecoration: todo.complete && "line-through",
          }}
        >
          {todo.task}
        </LIST>
      );
    })}
  </ul>
</div>
</div>
</Container>
);
};
export default App;

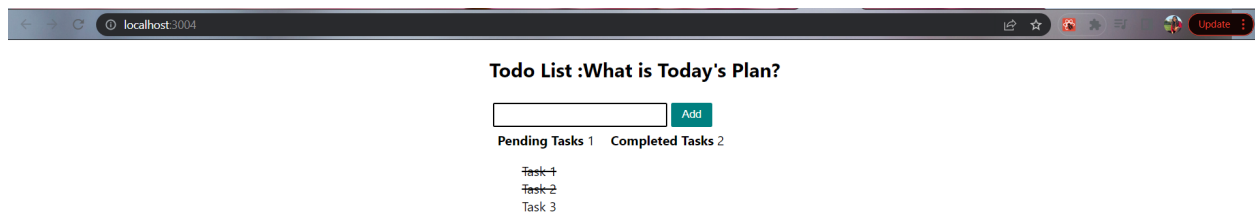
```

**Screenshots:**

→In this project User can able to add new tasks to the list,where once It is added it will display one by one



→User can also mark the completed tasks and have ability to remove and displays number of tasks completed and number of tasks pending :



## MID LEVEL TEST

### Challenge: Create a Submit Information Form

I have defined a validation schema using yup that specifies the required fields and their validation rules.

Used react-hook-form to handle form state and validation. And initialized it with the validation schema

I have defined an onSubmit function that sends the form data to the server using axios. If the submission is successful, It shows a success message using react-toastify and reset the form fields. If the submission fails, It shows an error message using react-toastify.

Rendering the form fields using register from react-hook-form. We also render error messages using the errors object returned by react-hook-form.

Rendeing a submit button and a ToastContainer from react-toastify to show success and error messages.

Finally This will give you a basic submit information form which handles validation and form submission.

```
import { useForm } from 'react-hook-form';
import * as yup from 'yup';
import axios from 'axios';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

const schema = yup.object().shape({
  name: yup.string().required('Name is required'),
  email: yup.string().email('Invalid email').required('Email is required'),
  phone: yup.string().required('Phone number is required'),
});

const SubmitForm = () => {
  const { register, handleSubmit, formState: { errors }, reset } =
    useForm({
      validationSchema: schema,
    });

  const onSubmit = async (data) => {
    try {
      await axios.post('https://your-server-url.com/form', data);
      toast.success('Form submitted successfully');
      reset();
    } catch (error) {
```

```

        toast.error('Form submission failed');
    }
};

return (
    <form onSubmit={handleSubmit(onSubmit)}>
        <div>
            <label htmlFor="name">Name:</label>
            <input {...register('name')} id="name" />
            {errors.name && <span>{errors.name.message}</span>}
        </div>
        <div>
            <label htmlFor="email">Email:</label>
            <input {...register('email')} id="email" />
            {errors.email && <span>{errors.email.message}</span>}
        </div>
        <div>
            <label htmlFor="phone">Phone:</label>
            <input {...register('phone')} id="phone" />
            {errors.phone && <span>{errors.phone.message}</span>}
        </div>
        <button type="submit">Submit</button>
        <ToastContainer />
    </form>
);
};

export default SubmitForm;

```

Screenshot:

localhost:3002

Update

Name:

Email:

Phone:

Form submission failed

## ADVANCED TEST

### Challenge: Dynamic Front Page with TypeScript and React

As the task is to create the dynamic Front-end using react.js and Typescript .I have created a react app with typescript by:

**`npx create-react-app dynamic-frontpage --template typescript`**

So this command will create the react app with type script where the below one displays once it is compiled:

---

Compiled successfully!

You can now view dynamic-frontpage in the browser.

Local: `http://localhost:3003`

On Your Network: `http://192.168.1.119:3003`

Note that the development build is not optimized.

To create a production build, use `npm run build`.

webpack compiled successfully

No issues found.

---

Now I have installed few packages using npm command:

`npm install axios @types/axios`

`npm install react-router-dom @types/react-router-dom`

`npm install styled-components`

Here Axios is a popular HTTP client for fetching data from an API. react-router-dom is used for routing in the React application. styled-components is used for styling.

Now I have created a header component and added search bar in it,Here is the code

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

```
type HeaderProps = {  
  onSearch: (query: string) => void;  
};
```

```
const Header: React.FC<HeaderProps> = ({ onSearch }) => {  
  const [searchQuery, setSearchQuery] = useState("");
```

```
  const handleSearch = (e: React.ChangeEvent<HTMLInputElement>) => {
```

```

    setSearchQuery(e.target.value);
    onSearch(e.target.value);
  };

  return (
    <header>
      <h1>Dynamic Front Page</h1>
      <input type="text" value={searchQuery} onChange={handleSearch} placeholder="Search by name" />
    </header>
  );
};

export default Header;
----->

```

**Now created another component for Item list:**

```

import React from 'react';

type Item = {
  id: number;
  name: string;
  description: string;
  image: string;
};

type ItemListProps = {
  items: Item[];
};

const ItemList: React.FC<ItemListProps> = ({ items }) => {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>
          <h2>{item.name}</h2>
          <p>{item.description}</p>
          <img src={item.image} alt={item.name} />
        </li>
      ))}
    </ul>
  );
};

```



```
export default ItemList;
```

----->

Now to sort the item I have created another component called **SortBy**:

**#code**

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

type SortByProps = {
  options: { label: string; value: string }[];
  onChange: (value: string) => void;
};

const SortBy: React.FC<SortByProps> = ({ options, onChange }) => {
  const [selectedOption, setSelectedOption] = useState(options[0].value);

  const handleChange = (e: React.ChangeEvent<HTMLSelectElement>) => {
    setSelectedOption(e.target.value);
    onChange(e.target.value);
  };

  return (
    <div>
      <label htmlFor="sort-by">Sort by:</label>
      <select id="sort-by" value={selectedOption} onChange={handleChange}>
        {options.map(option => (
          <option key={option.value} value={option.value}>
            {option.label}
          </option>
        ))}
      </select>
    </div>
  );
};

export default SortBy;
```

----->

To add new Item I have created another component called **AddItemForm**

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

```

type AddItemFormProps = {
  onSubmit: (data: { name: string; description: string; image: string }) => void;
};

const AddItemForm: React.FC<AddItemFormProps> = ({ onSubmit }) => {
  const [formData, setFormData] = useState({ name: "", description: "", image: "" });

  const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    onSubmit(formData);
    setFormData({ name: "", description: "", image: "" });
  };

  const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="name">Name:</label>
        <input type="text" id="name" name="name" value={formData.name}
onChange={handleInputChange} />
      </div>
      <div>
        <label htmlFor="description">Description:</label>
        <textarea id="description" name="description" value={formData.description}
onChange={handleInputChange} />
      </div>
      <div>
        <label htmlFor="image">Image URL:</label>
        <input type="text" id="image" name="image" value={formData.image}
onChange={handleInputChange} />
      </div>
      <button type="submit">Add Item</button>
    </form>
  );
};

export default AddItemForm;

```

---

Now adding everything into the App.tsx:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
```

```
import Header from './components/Header';
import ItemList from './components/ItemList';
import SortBy from './components/SortBy';
import AddItemForm from './components/AddItemForm';
```

```
type Item = {
  id: number;
  name: string;
  description: string;
  image: string;
};
```

```
const App: React.FC = () => {
  const [items, setItems] = useState<Item[]>([]);
  const [searchQuery, setSearchQuery] = useState("");
  const [sortOption, setSortOption] = useState('name');
```

```
  useEffect(() => {
    axios.get<Item[]>('/api/items').then(response => setItems(response.data));
  }, []);
```

```
  const handleSearch = (query: string) => {
    setSearchQuery(query);
  };
```

```
  const handleSort = (option: string) => {
    setSortOption(option);
  };
```

```
  const handleAddItem = (data: { name: string; description: string; image: string }) => {
    axios.post('/api/items', data).then(response => {
      setItems([...items, response.data]);
    });
  };
```

```
  const filteredItems = items.filter(item =>
    item.name.toLowerCase().includes(searchQuery.toLowerCase())
  );
```

```
  const sortedItems = filteredItems.sort((a, b) => {
    if (sortOption === 'id') {
```

```

    return a.id - b.id;
  } else if (sortOption === 'name') {
    return a.name.localeCompare(b.name);
  } else {
    // sort by any other attribute
    return 0;
  }
});

return (
  <div>
    <Header onSearch={handleSearch} />
    <SortBy
      options={[
        { label: 'Name', value: 'name' },
        { label: 'ID', value: 'id' },
        { label: 'Other Attribute', value: 'other' },
      ]}
      value={sortOption}
      onChange={handleSort}
    />
    <ItemList items={sortedItems} />
    <AddItemForm onSubmit={handleAddItem} />
  </div>
);
};

export default App;

```