Namaste React Assignment7:

1. What are the various ways to add images into your App? Explain with code examples?

Ans:

* Importing the images directly: Import the image directly into your React component and use it as a variable.

Example:

import myImage from ‘./images/image1.jpg’

const MyComponent = () =>

{

return(

<div>

<img src={ myImage } alt=”Image”/>

</div>

)

}

* Place images in the public folder and reference them using `process.env.PUBLIC\_URL`

Public folder inside the project main directory, inside the public folder having Images folder and inside the images folder having Images.

<img src={process.env.PUBLIC\_URL + '/images/image.jpg'} alt="My Image" />

But this works only in create-react-app

* Use **require** to dynamically load images.
* const {Food\_Logo} = require('../utils/Constants/')
* <img src={Food\_Logo} alt="Food logo" />
* Use CSS to set background images.

.image-container {

    background-image: url('./public/Images/MyImage.png');

    background-size: cover;

    width: 100px;

    height: 100px;

  }

 <div className="image-container">

        </div>

1. What will happen if you do the console.log(useState())?

Ans:

When you use **console.log(useState())**, it logs an array with two elements:

1. The initial state value.
2. The updater function.

This is the result of the initial invocation of the **useState** hook. Even if you later update the state using the setter, calling **console.log(useState())** won't reflect the updated state value; it will still show the initial state value.

If you want to log the current state value after it has been updated, you should directly log the state variable:

1. How will useEffect() hook behave when we don’t add a dependency array?

Ans: If we don’t add a dependency array, useEffect() hook callback function will be called every time when the component render or re-render.

Example:

useEffect(()=>{

console.log(“UseEffect called”);

})

1. What is SPA?

Ans: SPA stands for Single Page Application.

Single page application is a application where the content will be replaced or updated inside that single page only when the user interacts with the application, no other pages will be loaded, in React we use the Index.html and everything we see inside the React application is the content of Index.html only, based on the user interaction the Index.html page content will be updated or replaced.

1. What is the difference between the client-side routing and server-side routing?

Ans:

In client-side routing, each navigation event does not necessarily trigger a request to the server. Instead, the page updates dynamically, loading only the necessary parts without a complete page reload.

In server-side routing, each navigation event triggers a request to the server. The server processes the request, generates the corresponding HTML content for the entire page, and sends it back to the client. The entire page is then reloaded, resulting in a full-page transition. This approach ensures that each new route or page is fetched from the server, providing fresh content with each navigation.

* **Client-side routing** is commonly associated with SPAs. SPAs leverage client-side routing to dynamically update and change content within the same page without requiring a full page reload. This approach is well-suited for providing a smooth and seamless user experience in applications where interactions occur on a single page.
* **Server-side routing** is not as central to the SPA architecture. In traditional server-side routing, each navigation event typically involves a request to the server for a new page, leading to a full page reload. While it's possible to use server-side rendering (SSR) in conjunction with SPAs for initial page load or SEO purposes, the primary nature of SPAs involves client-side rendering and routing.
* So, in summary, client-side routing is a common approach in SPAs, while server-side routing is not the primary mechanism but may be used in hybrid scenarios or for specific use cases.

Formik library in React:

Without formik:

If we want to store form data inside our local state variables we used to do like this:

const login = () =>

{

const [firstName, setFirstName] = useState(“”);

return(

<form>

<input type=”text” name=”firstName” id=”firstName” placeholder=”Enter your first name”

onChange={(e)=> setFirstName(e.target.value)} />

</form>

)

}

Similarly we also have to create the state variables for lastName, password and add onChange for all the things, this approach is difficult to follow sometimes, so here we can use the formik library.

const[firstName, setFirstName] = useState(“”);

const[lastName, setLastName] = useState(“”);

const[email, setEmail] = useState(“”);

<input type=”text” name=”firstName” id=”firstName” placeholder=”Enter your first name”

value={firstName} onChange={(e)=> setFirstName(e.target.value)} />

With formik:

Npm install formik

const formik = useFormik({

initialValues: {

firstName: “”,

lastName: “”,

email: “”

}

})

<input type=”text” name=”firstName” id=”firstName” placeholder=”Enter your first name”

value={formik.values.firstName} onChange={formik.handleChange} />

<input type=”text” name=”lastName” id=”lastName” placeholder=”Enter your last name”

value={formik.values.lastName} onChange={formik.handleChange} />

<input type=”text” name=”email” id=”email” placeholder=”Enter your email”

value={formik.values.email} onChange={formik.handleChange} />

So like this using formik we have binded the data in two way and managed form state, now we will see how to use this data when we want to submit .

const formik = useFormik({

initialValues: {

firstName: “”,

lastName: “”,

email: “”

},

onSubmit: (values) => {

console.log(values)

}

})

<form onSubmit={formik.handleSubmit}>

import { useFormik } from "formik";

const Login = () =>{

    const formik= useFormik(

        {

            initialValues:{

                firstName: "",

                password: ""

            },

            onSubmit: (values)=>{

                console.log(values)

            }

        }

    )

    return(

        <div>

        <h1>Please Login to your account</h1>

        <form onSubmit={formik.handleSubmit}>

            {/\* <h1>Login page</h1> \*/}

            <div className="input-container">

                <input type="text" name="firstName" id="firstName" placeholder="firstName"

                value={formik.values.firstName} onChange={formik.handleChange}/>

            </div>

            <div className="input-container">

                <input type="password" name="password" id="passsword" placeholder="password"

                value={formik.values.password} onChange={formik.handleChange}/>

            </div>

            <div className="input-container">

                <button type="submit">Login</button>

            </div>

        </form>

        </div>

    )

}

export default Login;

Validations using yup library: formik works well with the yup library for form validations.

Npm install yup

import \* as Yup from “yup”;

inside the useFormik({}) add the validationSchema.

const formik = useFormik(

{

initialValues: {

firstName: “”,

lastName: “”,

email: “”,

},

validationSchema: Yup.object({

firstName: Yup.string().max(15, “maximum length is 15 characters”).min(2, “minimum length is 2 characters”).required(“It is a required field”)

}

)

})

Now below the input type of firstName we can write this:

{formik.errors.firstName? <p>{formik.erros.firstName}</p>: null}

For email validation we can simply do like this:

email: Yup.string().email(“Invalid email address”).required(“This is a required field”)

When we add the validations as required for all the fields like firstName and lastName etc, initially when I go to the firstName to enter it will be empty so it will give “This is a required field” error, but it is also giving “This is required field” error for lastName and email etc even without we touching those fields, so to fix this issue we can use the onBlur and formik.touched.

<div className="input-container">

                <input type="text" name="firstName" id="firstName" placeholder="firstName"

                value={formik.values.firstName} onChange={formik.handleChange}

onBlur={formik.handleBlur}/>

            </div>

formik.touched returns true only when we touched the fields.

By modifying this code:

{formik.errors.firstName? <p>{formik.erros.firstName}</p>: null}

To this will fix our issue:

{formik.touched.firstName && formik.errors.firstName? <p>{formik.errors.firstName}</p>: null}

import { useFormik } from "formik";

import \* as Yup from "yup";

const Login = () =>{

    const formik= useFormik(

        {

            initialValues:{

                firstName: "",

                password: ""

            },

            validationSchema: Yup.object({

                firstName: Yup.string().

                max(15, "maximum length is 15 characters for firstName").

                min(2, "minimum length is 2 characters for FirstName").

                required("This is a required filed"),

                password: Yup.string().required("This is a required field")

            }),

            onSubmit: (values)=>{

                console.log(values)

            }

        }

    )

    return(

        <div>

        <h1>Please Login to your account</h1>

        <form onSubmit={formik.handleSubmit}>

            {/\* <h1>Login page</h1> \*/}

            <div className="input-container">

                <input type="text" name="firstName" id="firstName" placeholder="firstName"

                value={formik.values.firstName} onChange={formik.handleChange}

                onBlur={formik.handleBlur}/>

                {formik.touched.firstName && formik.errors.firstName? <p>{formik.errors.firstName}</p>: null}

            </div>

            <div className="input-container">

                <input type="password" name="password" id="passsword" placeholder="password"

                value={formik.values.password} onChange={formik.handleChange}

                onBlur={formik.handleBlur}/>

                {formik.touched.password && formik.errors.password? <p>{formik.errors.password}</p>: null}

            </div>

            <div className="input-container">

                <button type="submit">Login</button>

            </div>

        </form>

        </div>

    )

}

export default Login;