REACT

Note that react is a library not a framework

To install react locally, and create an app

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

Getting Started

Navigate to `src/App.js`, and delete the code inside to get the below code format which will be the starting point of the application. Note that the `App.js` will be the base component.

```
function App(){
    return (
        <div className="container">

        </div>
    )
}

export default App
```

Note that a functional component returns one root element, hence rap the other elements in div or <> </>

Make sure to activate React Javascript for the Emmet in vscode

To test if the application is running, we can create the below code where we create variables outside the return and test if they are picked up inside the return expression

```
function App(){
    const name = "Brad"
    const x = false

    return (
        <div className="container">
            <h1>Hello From {name}</h1>
            <h2>Present {x ? 'Yes': 'No'} </h2>
        </div>
    )
}

export default App
```

`{x ? 'Yes': 'No'}` This is a ternary operator executed inside the component.

For the components that are to be created we are to house them inside a `components` folder under the `src` folder.

The first component in this case will be the `Header` component... To create it, inside the components folder, we create a file by the name `Header.js`

with vscode we can install ES7/React extension that provides snippets, for react elements. e,g `rafce` creates a boiler plate of a react component.

Hence the base code of a functional component is as shown below

```
const Header = () => {
  return (
    <div>Header</div>
  )
}

export default Header
```

For the created component we will have to import it to the `App.js` component.

```
import Header from "./components/Header"

function App(){
    const name = "Brad"
    const x = false

    return (
        <div className="container">
            <h1>Hello From {name}</h1>
            <h2>Present {x ? 'Yes': 'No'} </h2>
            <Header/>
        </div>
    )
}

export default App
```

To create a classed based component

```
check code online
```

Props

Used to propagate data from the parent to the child components, in this case we are to transmit data from the `App` to `Header` component. Props are transmitted in key value pairs, the name of the prop and the value of the prop.

A Prop can be thought as a radio channel which is transmitting data from the parent to the child component. The name of the radio channel together with the message to be transmitted are set in the parent component, while the parent child would have listen for the name of the channel so as to receive the message.

In the example below, we are to send the data `Hello` from the App to the Header component, hence we are to transmit the data through a channel called `title` and in the Header component we will listen for `title`

```
import Header from "./components/Header"

function App(){
    return (
        <div className="container">
            <Header title="Hello"/>
        </div>
    )
}

export default App
```

In the above we are sending `Hello` to the *Header* component, to receive the data we are to write the below code inside the *header* component;

```
const Header = (props) => {
  return (
    <div>
        <h1>{props.title}</h1>
    </div>
  )
}

export default Header
```

Note that the function will receive the **props** as a parameter for the function, and retrieve the data we call `props.title`.

*** Default Props

Inside the child elements we can place default props, in that the defaults will be used if none are placed inside the parent, i.e

```
import Header from "./components/Header"

function App(){
    return (
        <div className="container">
            <Header />
        </div>
    )
}

export default App
```

```
const Header = (props) => {
  return (
    <div>
        <h1>{props.title}</h1>
    </div>
  )
}

Header.defaultProps = {
  title: 'Task Tracker'
}

export default Header
```

The above two illustrates the use of default props.

*** Cleaning the props code,

Instead of passing in the props into the function, we can directly pass the name of the props i.e

```
const Header = ({title}) => {
  return (
    <div>
        {/* <h1>{props.title}</h1> */}
        <h1>{title}</h1>
    </div>
  )
}

Header.defaultProps = {
  title: 'Task Tracker'
}

export default Header
```

Proptypes - We can set the default type of the props to be passed, in this case we are to set the title prop to be a string, hence if we feed in a integer the code fails

```
import PropTypes from 'prop-types'

// const Header = (props) => {
const Header = ({title}) => {
  return (
    <div>
        <h1>Task Tracker</h1>
        {/* <h1>{props.title}</h1> */}
        <h1>{title}</h1>
    </div>
  )
}

Header.defaultProps = {
  title: 'Task Tracker'
}
```

```
Header.propTypes = {
  //title: PropTypes.string,
  title: PropTypes.string.isRequired,
}
export default Header
```

Styling

Direct Css in JS

In react the code uses camel case for the css variables

```
<h1 style={{color:'red', backgroundColor:'black'}} >{title}</h1>
```

Writing in the component

```
const Header = ({title}) => {
  return (
    <div>
        <h1 style={headingStyle} >{title}</h1>
    </div>
  )
}

const headingStyle = {
  color:'red',
  backgroundColor:'black'
}

export default Header
```

Other option for CSS is to place all the CSS inside the index.css file.

Button Component

The button is to be a child of the header component

```
const Button = ({color , text}) => {
  return (
    <button className='btn' style={{backgroundColor: color}}>{text}</button>
  )
}

export default Button
```

As for the `Header` component

```
const Header = ({title}) => {
  return (
    <header className='header'>
      <h1>{title}</h1>
      <Button color='green' text="Add"></Button>
    </header>
  )
}
export default Header
```

Events

```
import PropTypes from 'prop-types'
const onClick = () => {
    console.log('Click')
}
const Button = ({color , text}) => {
  return (
    <button onClick={onClick} className='btn' style={{backgroundColor: color}}>
{text}</button>
  )
}

Button.defaultProps = {
    color: 'steelblue',
}

Button.propTypes = {
    text: PropTypes.string,
}
export default Button
```

TAsks

Creating a tasks component, containing an array of variables.

```
const tasks = [
    {
        id: 1,
        text: "Docs Appointment",
        day: "Feb 5th at 2:30",
        reminder: true,
    },

    {
        id: 2,
        text: "Meeting at School",
        day: "Feb 6th at 1:30PM",
        reminder: true,
    },

    {
        id: 3,
        text: "Shopping",
        day: "Feb 5th at 2:30",
        reminder: false,
    }

]

const Tasks = () => {
  return (
    <>
    {tasks.map((task)=>(<h3 key={task.id}>{task.text}</h3>))}
    </>
  )
}

export default Tasks
```

Note the arrow function on the above `{tasks.map((task)=>(<h3 key={task.id}>{task.text}</h3>))}`

In the above example the variables are placed outside the component.

useState

In this case we are to house the list inside the component, hence the introduction of **useState**.

To incorporate **useState** we will first import it into the component. `import { useState } from "react"`

```
import { useState } from "react"

const Tasks = () => {

    const [tasks, setTasks] = useState([
        {
            id: 1,
            text: "Docs Appointment",
            day: "Feb 5th at 2:30",
            reminder: true,
        },

        {
            id: 2,
            text: "Meeting at School",
            day: "Feb 6th at 1:30PM",
            reminder: true,
        },

        {
            id: 3,
            text: "Shopping",
            day: "Feb 5th at 2:30",
            reminder: false,
        }
    ])

  return (
    <>
    {tasks.map((task)=>(<h3 key={task.id}>{task.text}</h3>))}
    </>
  )
}

export default Tasks
```

To Change tasks, we will have to recreate it, with the use of *setTasks*

`setTasks([...tasks, {newObject}])`

Individual Task Component

```
import {FaTimes} from "react-icons/fa"

const Task = ({task}) => {
  return (
    <div className="task">
        <h3>{task.text} <FaTimes style={{color: "red", cursor: 'pointer'}}/>
</h3>
        <p>{task.day}</p>
    </div>
  )
}

export default Task
```

To install react icons as used in the code above run `npm i react-icons`, After installing restart the server.

To import the icons to the component

`import {FaTimes} from "react-icons/fa"`

Delete Task

To delete a task the chain of delete will be `App => Tasks => Task`

In the App Component

```
//Delete Tasks
    const deleteTask = (id) =>{
        console.log("Delete Task", id)

    }

    return (
        <div className="container">
            <Header/>
            <Tasks tasks={tasks} onDelete = {deleteTask}/>
        </div>
    )
```

In the Tasks component

```
const Tasks = ({tasks, onDelete}) => {
  return (
    <>
    {tasks.map((task)=>(
    <Task
        key={task.id}
        task={task}
        onDelete={onDelete}
    />))}
    </>
  )
}
```

In the task Component

```
import {FaTimes} from "react-icons/fa"

const Task = ({task, onDelete}) => {
  return (
    <div className="task">
        <h3>{task.text} <FaTimes style={{color: "red", cursor: 'pointer'}}
onClick={onDelete} /> </h3>
        <p>{task.day}</p>
    </div>
  )
}

export default Task
```

In the above, the event object will be passed onto the function to pass in the id specifically, we do as shown below

```
<h3>{task.text} <FaTimes style={{color: "red", cursor: 'pointer'}} onClick=
{()=>onDelete(task.id)} />
```

To actually delete an item from the display, we use the function

```
//Delete Tasks
    const deleteTask = (id) =>{
        //To delete the clicked item from the display
        setTasks(tasks.filter((task)=>task.id !== id ))//Takes only items with ids not similar to the one passed in
    }
```

Optional Messaging

To display an option message incase of conditions with the use of ternary operator **?**.

```
{tasks.length > 0 ?
            (<Tasks tasks={tasks} onDelete = {deleteTask}/>) :
            ('No Tasks to Show')}
```

Toggle Task

We are to toggle the task upon doble click, also to show the change in task on the display, we are to use temple llitral on the class name

```
className={`task ${task.reminder ? 'reminder': ""}`}
```

The propagation on the toggle

In the App component

```
//Toggle Reminder
    const toggleReminder = (id)=>{
        setTasks(tasks.map((task)=> task.id === id? {...task, reminder: !task.reminder}: task))
    }

    return (
        <div className="container">
            <Header/>
            {tasks.length > 0 ?
            (<Tasks tasks={tasks} onDelete = {deleteTask} onToggle={toggleReminder}/>) :
            ('No Tasks to Show')}
        </div>
    )
```

On the Tasks Component

```
const Tasks = ({tasks, onDelete, onToggle}) => {
  return (
    //setTasks([...tasks, {newObject}])
    <>
    {tasks.map((task)=>(
    <Task
        key={task.id}
        task={task}
        onDelete={onDelete}
        onToggle = {onToggle}
    />))}
    </>
  )
}
```

On the Task Component

```
const Task = ({task, onDelete, onToggle}) => {
  return (
    <div className={`task ${task.reminder ? 'reminder': ""}`} onDoubleClick=
{()=>onToggle(task.id)}>
        {/* <h3>{task.text} <FaTimes style={{color: "red", cursor: 'pointer'}}
onClick={onDelete} /> </h3> */}
        <h3>
            {task.text}
            <FaTimes style={{color: "red", cursor: 'pointer'}} onClick=
{()=>onDelete(task.id)} />
        </h3>
        <p>{task.day}</p>
    </div>
  )
}
```

Form

To add and add task form, we will have to create its component

```
import { useState } from "react"


const AddTask = ({onAdd}) => {
    const [text, setText] = useState('')
    const [day, setDay] = useState('')
    const [reminder, setReminder] = useState(false)

    const onSubmit = (e) => {
        e.preventDefault()
        if(!text){
            alert("Please Add a Task")
        }

        onAdd({text, day, reminder})
        setText('')
        setDay('')
        setReminder(false)
    }

  return (
    <form className="add-form" onSubmit={onSubmit}>
```

```
        <div className="form-control">
            <label >Tasks</label>
            <input type="text" name="" placeholder="Add Task" value={text}
onChange={(e)=> setText(e.target.value)} />
        </div>
        <div className="form-control">
            <label >Day & Time</label>
            <input type="text" name=""  placeholder="Add Date" value={day}
onChange={(e)=> setDay(e.target.value)}/>
        </div>
        <div className="form-control form-control-check">
            <label >Reminder</label>
            <input type="checkbox" checked={reminder} name="" value={reminder}
onChange={(e)=> setReminder(e.currentTarget.checked)} />
        </div>

        <input type="submit" value="Save Task" className="btn btn-block" />
    </form>

  )
}

export default AddTask
```

In the component above, we will require 3 variables, the *text, day & reminder*, The variables are initialized using the **useState**

```
const [text, setText] = useState('')
    const [day, setDay] = useState('')
    const [reminder, setReminder] = useState(false)
```

The values for the variables are set directly from the input field, which listens for the onchange command.

`onChange={(e)=> setText(e.target.value)}` - to set the text variable

`onChange={(e)=> setDay(e.target.value)}` - to set the day

`onChange={(e)=> setReminder(e.currentTarget.checked)}` - Not that the check box is targeted differently.

The submit of the form triggers the onSubmit, which triggers the `onAdd`, function sending the variables to the tasks variable.

```
const onSubmit = (e) => {
        e.preventDefault()
        if(!text){
            alert("Please Add a Task")
        }

        onAdd({text, day, reminder})
        setText('')
        setDay('')
        setReminder(false)
    }
```

To add the new created task to the overall tasks, in the App component.

```
const addTask = (task) => {
        var id = tasks.slice(-1)[0].id //Creating an id for the new task
        const newTask = {id, ...task} // Combing the id to the new task
        setTasks([...tasks, newTask]) //Adding the newTask to the rest of the
tasks

    }
```

Not that in the above, we are using the spread operator, `...` in the first instance it is used to add the **task** to the **id**, and in the second instance it is used to add the **newTask**, to the **tasks** array of objects, not that in this case they are rapped in **[]**

Toggling form

In this part we will try to toggle the task form. The toggle is to be impleted with the use of the **Add** button.

To start with we are to toggle the form on true or false.

`{showAddTask && <AddTask onAdd={addTask}></AddTask>}` - In this code **&&**, is used a short hand ternary operator in the case where we only have one condition.

In the above code, the variable `showAddTask` is use to house the conditions.

To change the `showAddTask`, we will listen for a click event from `App => Header => Button`

`<Header onAdd={()=>{setShowAddTask(!showAddTask)}}/>` - On the App component.

`Button color='green' text="Add" onAdd ={onAdd} ></Button>` - On the header component

`<button onClick={onAdd} className='btn' style={{backgroundColor: color}}>{text}</button>` - on the button component.

To change the text and color of the button

We are to pass the value of the `showAddTask` from the App=>Header

`<Header onAdd={()=>{setShowAddTask(!showAddTask)}} showAddTask ={showAddTask}/>` - in the App Component

`<Button color={showAddTask ? "red" : "green"} text={showAddTask ? "Close" : "Add"} onAdd ={onAdd} ></Button>` - In the header component

To Deploy, Build Static files

on the console `npm run build`

This creates a build folder for the files to be upload.

To check if the build files are working properly

First install the npm serve globally, by

`npm i -g serve`.

And to open the production files on a server

`serve -s build -p 8000`

Json Server

Creates a mock api server

To install it locally `npm i json-server`.

To be able to properly run it...in the `package.json` file, under the `scripts` add `"server": "json-server --watch db.json --port 5000"` - this runs the sever an creates a `db.json` file which acts as the database.

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "server": "json-server --watch db.json --port 5000"
  },
```

To run the server

`npm run server`,

And also `npm start`

Once everything is set, we are to use the **hook** `{useEffect}`, to import the data from the server.

To fetch data from the backend and set the `Tasks` variable

```
useEffect(()=>{
        const fetchTasks = async () => {
            const res = await fetch('http://localhost:5000/tasks')
            const data = await res.json()
            //return data
            setTasks(data)
        }

        fetchTasks()
    }, [])
```

Note that in the above we can, redisign the code such the fetchTasks is outside the `useEffects`

```
useEffect(()=>{
    const getTasks = async () => {
        const tasks = await fetchTasks()
        setTasks(tasks)
    }
    getTasks()
}, [])

const fetchTasks = async () => {
    const res = await fetch('http://localhost:5000/tasks')
    const data = await res.json()
        return data
}
```

Deleting, Adding, Get, Updating from server

TO delete

```
//Delete Tasks
    const deleteTask = async (id) =>{
        //To delete the task from the server- this function will first be
converted to an async function
        await fetch(`http://localhost:5000/tasks/${id}`,{
            method: 'DELETE'
        })

        //To delete the clicked item from the display
        setTasks(tasks.filter((task)=>task.id !== id ))//Takes only items with
ids not similar to the one passed in
    }
```

To add to the server

```
//Add Tasks
    const addTask =  async (task) => {
        //To add to the back end
        const res = await fetch('http://localhost:5000/tasks',{
            method: "POST",
            headers: {
                'Content-type': 'application/json'
            },
            body: JSON.stringify(task)
        })

        // To add it to the tasks
        const data = await res.json()
        setTasks([...tasks, data])
    }
```

To get a task item

```
const fetchTask = async (id) => {
        const res = await fetch(`http://localhost:5000/tasks/${id}`)
        const data = await res.json()
        return data
    }
```

To update a task

```
//Toggle Reminder
    const toggleReminder = async (id)=>{
        const taskToToggle = await fetchTask(id)
        const updatedTask = {...taskToToggle, reminder: !taskToToggle.reminder}
        const res = await fetch(`http://localhost:5000/tasks/${id}`, {
            method: 'PUT',
            headers: {
                'Content-type': 'application/json'
            },
            body: JSON.stringify(updatedTask)
        })
        const data = await res.json()
        setTasks(tasks.map((task)=> task.id === id? {...task, reminder:
data.reminder}: task))
    }
```

Routing

To use routing, we use the package

/

In this case we are going to place the links below the page, in a footer component

```
const Footer = () => {
  return (
    <footer>
        <p>Copyright &copy; 2022 </p>
        <a href="/about">About</a>
    </footer>
  )
}

export default Footer
```

As for the second page we will create its component

```
const About = () => {
  return (
    <div>
        <h4>Version 1.0.0</h4>
        <a href="/">Go back</a>
    </div>
  )
}

export default About
```

To use routes on the `App Component`, import `BrowserRouter, Route & Routes`

`import { BrowserRouter as Router, Route, Routes } from "react-router-dom"`.

In this case we are to have two routes `/` and `/about`, hence the return of the App Component can be changed to

```
<Router>
    <div className="container">
        <Header onAdd={()=>{setShowAddTask(!showAddTask)}} showAddTask =
{showAddTask}/>
        <Routes>
            <Route path = '/' excat element =
                {
                    <>
                     {showAddTask && <AddTask onAdd={addTask}></AddTask>}
                     {tasks.length > 0 ?(<Tasks tasks={tasks} onDelete =
{deleteTask}                              onToggle={toggleReminder}/>) : ('No
Tasks to Show')}
                    </>
                 } />
             <Route path='/about' element={<About />} ></Route>
         </Routes>
         <Footer />
     </div>
</Router>
```

Not that in the above, in `Route` is nested inside `Routes`.

To prevent the page from reloading, we can replace the `a` tag with `Link`, but to use link we will first have to import it `

```
<Link to="/about">About</Link>

<Link to="/">Go back</Link>
```

To remove the Button in the About page, We are to use conditions on the button depending on the location of the route, i.e to display it if the route is `/`, and vanish when the route is `/about`

Hence import location `import { useLocation } from 'react-router-dom'`, we are to place the condition in the Header Component

```
import PropTypes from 'prop-types'
import { useLocation } from 'react-router-dom'
import Button from './Button'

const Header = ({title, onAdd, showAddTask}) => {
  const  location = useLocation()
  return (
    <header className='header'>
      <h1>{title}</h1>
      {location.pathname === '/' && <Button color={showAddTask ? "red" :
"green"} text=         {showAddTask ? "Close" : "Add"} onAdd ={onAdd} >
</Button>}
    </header>
  )
}
```

Authentification

Registration

Requirements

```
npm i --save @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons
@fortawesome/react-fontawesome
npm i axios
npm i react-router-dom*** Check on this
```

Regex expressions for the fields validation

```
const USER_REGEX = /^[a-zA-Z][a-zA-Z0-9-_]{3,23}$/;
const PWD_REGEX = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%]).{8,24}$/;
const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

For the registration in this example we will house the contents in a `AuthForm.js`, component.

Imports required are:-

```
import { useEffect, useState, useRef } from "react"
import { json, Link, useLocation } from "react-router-dom"
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCheck, faTimes, faInfoCircle } from "@fortawesome/free-solid-svg-
icons";
import axios from "../../api/axios";
```

For validation of the forms fields, we are to use the above stated regex expressions, to ensure that the required format is inputted.

Inside the component expression, we are to first set some variables

```
//The below variables are used when toggling between the login and registration
fields
const location = useLocation()
const condition = location.pathname === '/login'
const url = condition ? '/login': '/register'
```

As for the form variables, `user, pwd & email`, each variable is tied to other variables `isvalid` and `focus`. The valid variable checks if the inputted variable is of the required format as required in the Regex expressions. As for focus checks if the input field of the variable in question is highlighted. Not that the `focus` state is used with `aria` , for screen naration.

```
const [user, setUser] = useState('')
const [validName, setValidName] = useState(false)
const [userFocus, setUserFocus] = useState(false)

const [email, setEmail] = useState('')
const [validEmail, setValidEmail] = useState(false)
const [emailFocus, setEmailFocus] = useState(false)

const [pwd, setPwd] = useState('')
const [validPwd, setValidPwd] = useState(false)
const [pwdFocus, setPwdFocus] = useState(false)

const [matchPwd, setMatchPwd] = useState('')
const [validMatch, setValidMatch] = useState(false)
const [matchFocus, setMatchFocus] = useState(false)
```

Also we are to initialize an `errMsg` variable to store messages passed to the component, and also `success` to store success stage.

```
const [errMsg, setErrMsg] = useState('')
const [success, setSuccess] = useState(false)
```

Next it comes to validating the format of the inputs, in this case we are to use `useEffects`. Note that in use Effect the function will run automatically.

Check w3.schools for examples on `useEffects`...The are three types of use

In this case we are going to pass a dependency to the `useEffect`, in this case the `useEffect` will run only when it detects a change in the dependency. Note that if the dependency is left blank the `useEffect` will run only during the first render, while if the dependecy list is not list, the `useEffect` will run on every render.

```
//To validate the user name - not that the user state is in the dependacy
    useEffect(() => {
        const result = USER_REGEX.test(user);
        setValidName(result)
    }, [user]);

    //To validate the EMAIL
    useEffect(() => {
        const result = EMAIL_REGEX.test(email);
        setValidEmail(result)
    }, [email])

    //To validate the password
    useEffect(() => {
        const result = PWD_REGEX.test(pwd);
        setValidPwd(result)
        const match = pwd === matchPwd;
        setValidMatch(match);
    }, [pwd, matchPwd])
```

Also for each change in the `user,pwd, matchPwd & email` , it is prudent to set the **error message** variable to blank

```
useEffect(() => {
    setErrMsg('')
}, [user, pwd, matchPwd, email])
```

Having set the variables, next is function to handle the submitted form data. In this case we are to use `fetch` for handling the `http` requests to the server... further research is required for `axios` and `flask` integration.

```
e.preventDefault();
    //Performing simple validation to check If button is enabled with JS hack
    const v1 = USER_REGEX.test(user)
    const v2 = PWD_REGEX.test(pwd)
    if(!v1 || !v2){
        setErrMsg("Invalid Entry")
        return;
    }
    try{
        const res = await fetch('/register', {
            method: "POST",
            headers: {
                'Content-type': 'application/json'
            },
            body: JSON.stringify({user, pwd, email})
        })

    setSuccess(true);

    //Clear input fields
    setEmail('')
    setPwd('')
    setUser('')
    } catch(err){
        //Setting Error messages
        if(!err?.response) {
            setErrMsg('No Server Response')
        } else if(err.response?.status === 409){
            setErrMsg('Username Taken')
        } else {
            setErrMsg('Registration Failed')
        }
    }
}
```

For the return of the we are to place the html logic, not that some of the variables are set with change in the input field, taking the userName input field as an example

```
<input
    type="text"
    className="form-control"
    id="userName"
    ref={userRef} //Further research needed on this
    autoComplete="off"
    onChange={(e)=>setUser(e.target.value)} //In this case we are setting the
value for the user variable
    onFocus = {() => setUserFocus(true)}
    onBlur = {() => setUserFocus(false)}                           required
/>
```

Hence the complete component is as indicated below, not that it is used for both the login and registration

```
import { useEffect, useState, useRef } from "react"
import { json, Link, useLocation } from "react-router-dom"
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCheck, faTimes, faInfoCircle } from "@fortawesome/free-solid-svg-
icons";
import axios from "../../api/axios";


const USER_REGEX = /^[a-zA-Z][a-zA-Z0-9-_]{3,23}$/;
const PWD_REGEX = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%]).{8,24}$/;
const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;



const AuthForm = () => {
    const location = useLocation()
    const condition = location.pathname === '/login'
    const url = condition ? '/login': '/register'


    const userRef = useRef();
    const errRef = useRef();
    const [user, setUser] = useState('')
    const [validName, setValidName] = useState(false) //Tied with whether the
userName validates or not
    const [userFocus, setUserFocus] = useState(false) //Tied with whether we are
focused into the user filed or not

    const [email, setEmail] = useState('')
    const [validEmail, setValidEmail] = useState(false)
    const [emailFocus, setEmailFocus] = useState(false)

    const [pwd, setPwd] = useState('')
    const [validPwd, setValidPwd] = useState(false)
    const [pwdFocus, setPwdFocus] = useState(false)

    const [matchPwd, setMatchPwd] = useState('')
    const [validMatch, setValidMatch] = useState(false)
    const [matchFocus, setMatchFocus] = useState(false)

    const [errMsg, setErrMsg] = useState('')
    const [success, setSuccess] = useState(false)

    useEffect(() => {
        userRef.current.focus(); // Note for this to work userRef must be used
inside the html elements
        //Note that the above code functions like e.target.value = It gets the
current value of the the user input
    }, [])

    //To validate the user name - not that the user state is in the dependacy
    useEffect(() => {
        const result = USER_REGEX.test(user);
        console.log(result);
        console.log(user)
        setValidName(result)
    }, [user]);

    //To validate the EMAIL
    useEffect(() => {
        const result = EMAIL_REGEX.test(email);
        console.log(result);
        console.log(email)
        setValidEmail(result)
    }, [email])

    //To validate the password
    useEffect(() => {
        const result = PWD_REGEX.test(pwd);
        console.log(result);
        console.log(pwd)
        setValidPwd(result)
        const match = pwd === matchPwd;
        setValidMatch(match);
    }, [pwd, matchPwd])


    useEffect(() => {
      setErrMsg('')
    }, [user, pwd, matchPwd, email])
    const handleSubmit = async (e) => {
        e.preventDefault();
        //If button is enabled with JS hack
        const v1 = USER_REGEX.test(user)
        const v2 = PWD_REGEX.test(pwd)
        if(!v1 || !v2){
            setErrMsg("Invalid Entry")
            return;
        }
        try{
            // const res = await axios.post(url,
            //     JSON.stringify({user, pwd, email}),
            //     {
            //         headers: {
            //             'Content-type': 'application/json'
            //           },
            //           withCredentials: true
            //     }
            // );

            const res = await fetch('/register', {
                method: "POST",
                headers: {
                    'Content-type': 'application/json'
                },
                body: JSON.stringify({user, pwd, email})
            })

            setSuccess(true);

            //Clear input fields
            setEmail('')
            setPwd('')
            setUser('')

        } catch(err){
            if(!err?.response) {
                setErrMsg('No Server Response')
            } else if(err.response?.status === 409){
                setErrMsg('Username Taken')
            } else {
                setErrMsg('Registration Failed')
            }
        }

    }

  return (
    <div className="tab-content">
        <p ref={errRef} style={errMsg ? errmsg : offscreen }>{errMsg}</p>
        <div className="tab-pane fade show active" id="pills-login"
role="tabpanel">
            <form onSubmit={handleSubmit}>
                {condition ?
                    (<p className="text-center mb-3">Sign in with:</p>) :
                    (<p className="text-center mb-3">Sign up with:</p>)
                }
                <div className="form-outline mb-4">
                    <label className="form-label" htmlFor="userName">
                        Username:
                        {!condition && (
                        <>
                            <FontAwesomeIcon icon={faCheck}  style={validName ?
valid : hide} />
                            <FontAwesomeIcon icon={faTimes} style={validName ||
!user ? hide :  invalid } />
                        </>)}

                    </label>
                    <input
                        type="text"
                        className="form-control"
                        id="userName"
                        ref={userRef}
                        autoComplete="off"
                        onChange={(e)=>setUser(e.target.value)}
                        onFocus = {() => setUserFocus(true)}
                        onBlur = {() => setUserFocus(false)}




                        required
                    />
                    {!condition && (
                    <p id="uidnote" style={userFocus && user && !validName ?
instructions : offscreen}>
                        <FontAwesomeIcon icon={faInfoCircle} />
                        4 to 24 characters.<br />
                        Must begin with a letter.<br />
                        Letters, numbers, underscores, hyphens allowed.
                    </p>
                    )}


                </div>

                {!condition && (
                    <div className="form-outline mb-4">
                        <label className="form-label" htmlFor="email">
                            Email:
                            <FontAwesomeIcon icon={faCheck}  style={validEmail ?
valid : hide} />
                            <FontAwesomeIcon icon={faTimes} style={validEmail ||
!email ? hide :  invalid } />
                        </label>
                        <input
                            type="email"
                            id="email"
                            className="form-control"
                            name="email"
                            placeholder="Email Address"
                            autoComplete="off"
                            onChange={(e)=>setEmail(e.target.value)}
                            onFocus = {() => setEmailFocus(true)}
                            onBlur = {() => setEmailFocus(false)}
                            required
                        />
                        <p id="uidnote" style={userFocus && user && !validName ?
instructions : offscreen}>
                            <FontAwesomeIcon icon={faInfoCircle} />
                            Must be of valid email format.
                        </p>
                    </div>
                )}

                <div className="form-outline mb-4">
                    <label className="form-label" htmlFor="password">
                        Password:
                        {!condition && (
                        <>
                            <FontAwesomeIcon icon={faCheck}  style={validPwd ?
valid : hide} />
                            <FontAwesomeIcon icon={faTimes} style={validPwd ||
!pwd ? hide :  invalid } />
                        </>)}

                    </label>
                    <input
                        type="password"
                        id="password"
                        className="form-control"
                        name="password"
                        onChange={(e)=>setPwd(e.target.value)}
                        onFocus = {() => setPwdFocus(true)}
                        onBlur = {() => setPwdFocus(false)}
                        required
                    />
                    {!condition && (
                        <p id="pwdnote" style={pwdFocus && !validPwd ?
instructions : offscreen}>
                            <FontAwesomeIcon icon={faInfoCircle} />
                            8 to 24 characters.<br />
                            Must include uppercase and lowercase letters, a
number and a special character.<br />
                            Allowed special characters: <span>! @ # $ % </span>
                        </p>
                    )}

                </div>

                {!condition && (
                    <div className="form-outline mb-4">
                        <label className="form-label"
htmlFor="registerRepeatPassword">
                            Confirm password:
                            <FontAwesomeIcon icon={faCheck}  style={validMatch
&& matchPwd ? valid : hide} />
                            <FontAwesomeIcon icon={faTimes} style={validMatch ||
!matchPwd ? hide :  invalid } />
                        </label>
                        <input
```

```
                                type="password"
                                id="confirmPassword"
                                className="form-control"
                                name="confirmation"
                                placeholder="Confirm Password"
                                onChange={(e)=>setMatchPwd(e.target.value)}
                                onFocus = {() => setMatchFocus(true)}
                                onBlur = {() => setMatchFocus(false)}
                                required
                            />
                            <p id="pwdnote" style={matchFocus && !validMatch ?
instructions : offscreen}>
                                <FontAwesomeIcon icon={faInfoCircle} />
                                Must match the first password input field
                            </p>
                        </div>
                    )}

                    {/* Submit button  */}
                    <input
                        type="submit"
                        className="btn btn-primary btn-block mb-4"
                        style={{width: "100%"}}
                        value={`Sign ${condition? "in" : "up"}`}
                        disabled={!validName || !condition && !validEmail ||
!validPwd || !condition && !validMatch ? true : false}
                    />
                    {/* Register buttons */}
                    {condition && (
                        <div className="text-center">
                            <p>Not a member? <Link to="/register" className="reg-
link">Register</Link></p>
                        </div>
                    )}

                </form>
            </div>
        </div>
  )
}

//Styling
const valid = {
    color: "limegreen",
    marginLeft: "0.25rem",
}

const instructions = {
    fontSize: '0.75rem',
    borderRadius: "0.5rem",
    background: "#000",
    color: "#fff",
    padding: "0.25rem",
    position: "relative",
    bottom: "-10px",
}

const offscreen =  {
    position: 'absolute',
    left: "-9999px",
}

const hide = {
    display: "none",
}

const invalid = {
    color: "red",
    marginLeft: "0.25rem",
}

const errmsg = {
    backgroundColor: "lightpink",
    color: "firebrick",
    fontWeight: "bold",
    padding: "0.5rem",
    marginBottom: "0.5rem",
}

const line = {
    display: "inline-block"
}

export default AuthForm
```

Login

For the jsx and variables it is set-up as the above, but in this case only the `user` and `pwd` variables are required.

In this section we are incorporate `useContext` as `AuthProvider`, which is to store the `user` and `pwd` globally. In this case the `user` and `pwd` will be passed from the `AuthForm` to the global scope.

To set up `context`, we will create a context folder and under it create `AuthProvider.js`

```
import { createContext, useState } from "react";

//Init createContext
const AuthContext = createContext({})

export const AuthProvider = ({children}) => {
    const [auth, setAuth] = useState({})
    return (
        //Rapping up the children(tree with the creatContext instance, in this
case                         AuthContext)
        //Also passing the values to the context i.e in the value part
        <AuthContext.Provider value={{auth, setAuth}}>
            {children}
        </AuthContext.Provider>
    )
}
```

```
export default AuthContext; // Contains the variables to be used in the global
scope
```

The above created component is then imported to the index.js app

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { AuthProvider } from './context/AuthProvider';


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

  <React.StrictMode>
    <AuthProvider>
        <App />
    </AuthProvider>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

To use the `Auth Context`, we will have to import `useContext`, and also the `AuthContext` inside the component we want it, hence in this case we are to import it into the `AuthForm` component.

```
import { useEffect, useState, useRef, useContext } from "react"
import AuthContext from './context/AuthProvider' //It contains the global data
```

Note that `AuthContext`, carries the variables to be used in the global state. To retrive the data into our component

```
const {setAuth} = useContext(AuthContext)
```

To set the value for `setAuth` once the form is submiites successfully, inside the `handleSubmit` function

```
setAuth({user, pwd, email})
```

With this we can use the above data on any component of the app.

Protected Routes

In this part, we are to restrict some routes.

In this case we will create a component that will contain logic that will check for authentification, in our case we will create a **RequireAuth** component.

```
import { useContext } from "react";
import { Navigate, Outlet, useLocation } from "react-router-dom";
import AuthContext from "../../context/AuthProvider";

const RequireAuth = () => {
    //Retrive the auth from the global scope
    const {auth} = useContext(AuthContext)

    const location = useLocation();
    return (
        //First we check if the auth object has user
        //If true, using ternary operator, we will house the components inside
the outlet
        //If false, the user is to be navigated to the login page
        //Note that we have set state so that once the user is logged in is
redirect back.
        auth?.user ? <Outlet/> : <Navigate to="/login" state={{from: location}}
replace/>
    )
}

export default RequireAuth
```

To protect the routes, we are to house the routes, the said routes are to be housed under the **RequireAuth** component. With the use of the **Outlet**, we are able to convert a **Route** to function as **Routes**.

Hence inside the **App.js**

```
<Router>
    <div className="App">
        <Nav></Nav>
        <Routes>
            <Route path="/" exact element= {<Welcome></Welcome>}/>
            <Route path="/login" element = {<Auth/>}/>
            <Route path="/register" element = {<Auth/>}/>
            {/* Protected routes */}
            <Route element={<RequireAuth/>} >
                <Route path="/products" element={<Products/>} />
            </Route>
        </Routes>
    </div>
</Router>
```

Role Based Authorization

We can further protect the routes by assigning specific roles to the the paths, i,e admin or user.

To do this we are to modify the **RequireAuth** component,

```
import { useContext } from "react";
import { Navigate, Outlet, useLocation } from "react-router-dom";
import AuthContext from "../../context/AuthProvider";

const RequireAuth = ({allowedRoles}) => {
    //Retrive the auth from the global scope
    const {auth} = useContext(AuthContext)

    const location = useLocation();
    return (
        //We Check if the Auth object has any roles
        //If any, we loop through the roles and check in the allowedRoles if we
have a match
        //If the user is loged in and not authorized we can redirect the user to
an autorized page
        auth?.roles?.find(role => allowedRoles?.includes(role))
            ? <Outlet/>
            : auth?.user
                ? <Navigate to="/unauthorized" state={{from:location}} replace/>
                :<Navigate to="/login" state={{from: location}} replace/>
    )
}

export default RequireAuth
```

And on the **App.js**,

```
<Router>
    <div className="App">
        <Nav></Nav>
        <Routes>
            <Route path="/" exact element= {<Welcome></Welcome>}/>
            <Route path="/login" element = {<Auth/>}/>
            <Route path="/register" element = {<Auth/>}/>
            <Route path="/unauthorized" element = {<Unauthorized/>}/>
            {/* Protected routes */}
            <Route element={<RequireAuth allowedRoles = {[2001, 1984]}/>} >
                <Route path="/products" element={<Products/>} />
            </Route>
            <Route element={<RequireAuth allowedRoles = {[1984]}/>} >
                <Route path="/jobs" element={<Jobs/>} />
            </Route>
        </Routes>
    </div>
</Router>
```

JWT

Json Web Tokens - Its a refresh token

Refresh token - Grant a user authentication privilege's for a longer period of time

Access token - Shorter period of time