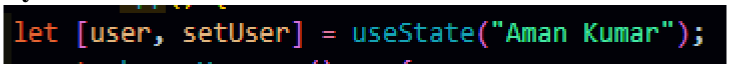
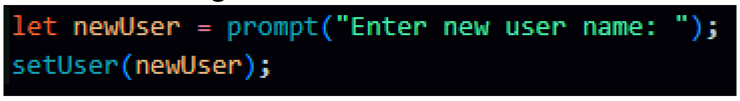
Revising React

1. **Initial Steps:**
    a. Setting up the structure for revising the react.
    b. We will use the vite for making the react folder setup, vite is the bundler for making the folder structure for React.
    c. Added tailwind CSS in the project.
2. **Basic Concepts:**
    a. React + React DOM = Web Applications
    b. React + React Native = Mobile Applications
    c. We need react because when we have to update only single element so using JS our whole tree needs to re-render the application but **by using React** when we click on button to change something it **makes the copy of our real DOM called virtual DOM** and it compare the changes made in the virtual DOM and then it merge the changes into the real DOM.
    d. JSX is the combination for HTML and JS.
    e. Our main working file is **App.jsx** and we export our whole code.
    f. In **Main.jsx** file we just import all codes from the while and export it to the **index.html** file.
    g. We can't directly return the two tags together, to return more than 1 tag we have to use the **empty fragments "<> </>" or use a simple empty div tag <div> </div>** to store all the elements.
    h. To declare any variable, functions; declare them outside the return statement inside the function.
3. **Hooks:** Hooks are special type of functions which provide powerful power to react. Hooks are functional components.
    a. **Use State:** It is a special container used for storing and updating the data into the functional component. **State** means the data which changes over time.
        i. **Syntax:**



```
let [user, setUser] = useState("Aman Kumar");
```

Here, user: It a read only variable, Set user: It is write only variable. Whenever we assign values, we assign to user; in the example you can see that Aman Kumar is assigned to user and set user will be used when we need to change the value of the user.



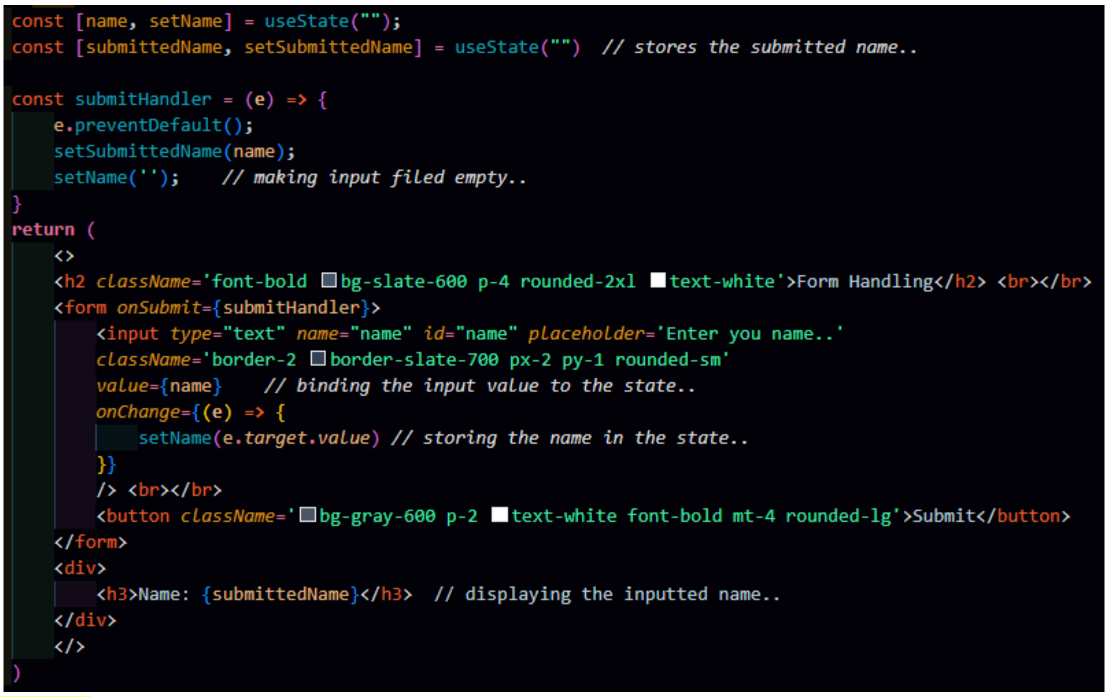
```
let newUser = prompt("Enter new user name: ");
setUser(newUser);
```

Like, we are changing the user's value to new user value using use state.

    b. **Use Effect:** It is used to fetch, load or update the DOM.
    c. **Use Context:** It is used to avoid prop drilling and share data between components.
    d. **Use Memo:** It is to optimize performance by memoizing a calculated value.
4. **From Handling:**
    a. **Preventing the default behavior of the form:** After filling the input filed, when we click on submit button it reloads the page and our text in input field is removed to stop this we will prevent the default behavior of our form.
5. **Two way Binding:**

a. It is used to input the text in the **input field using Hooks NOT directly** putting the values in the field.

b. Example for the two-way binding:



```
const [name, setName] = useState("");
const [submittedName, setSubmittedName] = useState("")  // stores the submitted name..

const submitHandler = (e) => {
    e.preventDefault();
    setSubmittedName(name);
    setName('');    // making input filed empty..
}
return (
    <>
    <h2 className='font-bold bg-slate-600 p-4 rounded-2xl text-white'>Form Handling</h2> <br></br>
    <form onSubmit={submitHandler}>
        <input type="text" name="name" id="name" placeholder='Enter you name..'
        className='border-2 border-slate-700 px-2 py-1 rounded-sm'
        value={name}    // binding the input value to the state..
        onChange={(e) => {
            setName(e.target.value) // storing the name in the state..
        }}
        /> <br></br>
        <button className='bg-gray-600 p-2 text-white font-bold mt-4 rounded-lg'>Submit</button>
    </form>
    <div>
        <h3>Name: {submittedName}</h3>  // displaying the inputted name..
    </div>
    </>
)
```

6. Components:

a. Components allow you to break down a complex UI into smaller, reusable, and independent pieces of code. You can use these components many times.

b. Name of component file **must start from capital letter.**

7. Props/Props Drilling:

a. **Props:** When data passed from parent to a child component.

b. In props we can only send the data from **top to bottom.**

c. **Props Drilling:** When data passed from parent to a child component, even the component who comes between them don't need that data but they have to store and pass that data, so that child component can access it.

   i. App (Parent) → ComponentA → ComponentB → ComponentC (Child)

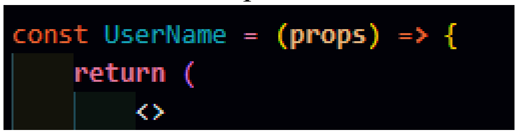
   ii. Child needs the data from Parent but A and B need to store and pass the data.

   iii. **This leads to memory consumption and made our app slow.**

d. **Steps for passing properties (props):**

   i. We will create a card component (for example).

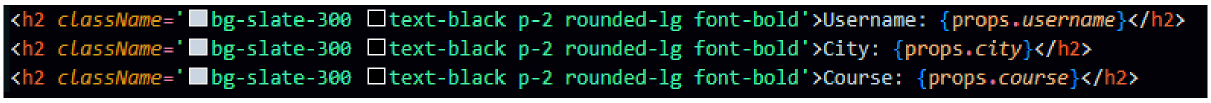
   ii. To pass the data across the components we will pass a parameter in the Card function component.



```
const UserName = (props) => {
    return (
        <>
```

In this case we have passed the **props as parameter.**

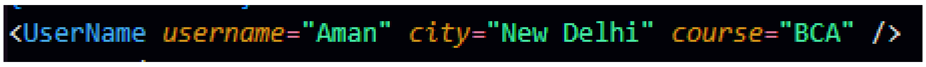
   iii. In the card component wherever we need that data needs to pass, we will use "{}" curly brackets.



```
<h2 className='■bg-slate-300 □text-black p-2 rounded-lg font-bold'>Username: {props.username}</h2>
<h2 className='■bg-slate-300 □text-black p-2 rounded-lg font-bold'>City: {props.city}</h2>
<h2 className='■bg-slate-300 □text-black p-2 rounded-lg font-bold'>Course: {props.course}</h2>
```

We have used the {} brackets wherever we need to pass the data.

iv. When we call the card component in the **App.jsx** file we will pass all the data with the same variable name.



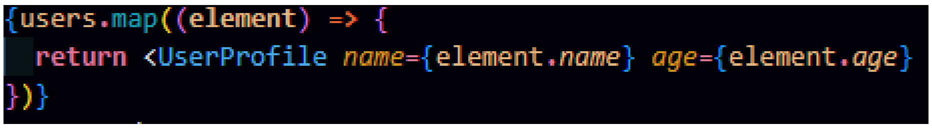
```
<UserName username="Aman" city="New Delhi" course="BCA" />
```

v. Keep in mind that variable passing while calling the components must match with the variable passed in that component.

e. We have created a card with Name, Age, Profession, Profile picture and State.

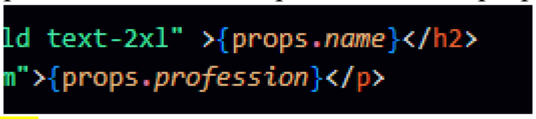
i. We have created a JSON data in **App.jsx** file.

ii. Then, we have used the **map function on JSON** data and in the return statement we have **returned our card with all the properties available in the JSON data**.



```
{users.map((element) => {
  return <UserProfile name={element.name} age={element.age}
})}
```

iii. And in our card component we have passed a parameter and using this parameter we have passed all the properties to their desired place.



```
ld text-2xl" >{props.name}</h2>
m">{props.profession}</p>
```

8. Integrating API:

a. To use API **install Axios** by using **npm install axios**.

b. Created an image component with button to fetch the image from an API- Lorem Picsum.

c. Created 2 states.

i. 1st is for image and a setter of set image.

ii. 2nd is for loading and a setter of set loading.

d. Created a async function to fetch images in which, in try block I have created a variable response and using await with fetch I have passed the API link.

e. To get single image each time, I have used the random and floor methods of Math function.

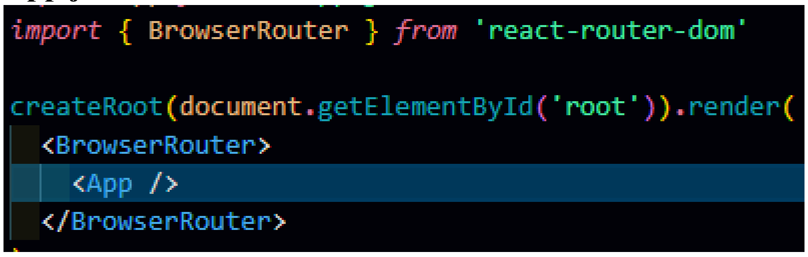
9. Router:

a. Created a pages named folder with files like Home, About and Services, etc.

b. To use React-Router-Dom in React we have to install it by using:

i. Npm install react-router-dom

c. 1st we will import the **browser router** in our **Main.jsx** file and wrap our **App.jsx** file with **Browser Route**.



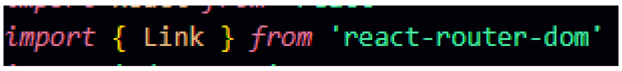
```
import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
```

d. 2nd we will use the **Routes** as a wrapper for **all the route.**

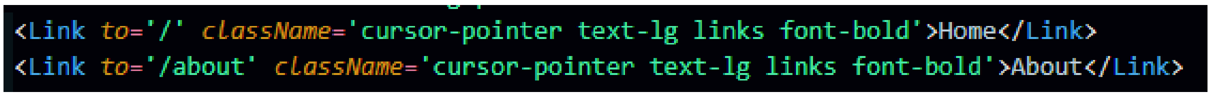
e. 3rd in route we will pass the **path** and then **element**, element is nothing but our **component**.

f. **For default page route, especially for home page, we will pass nothing in the path for home.**

g. To link the pages from the navigation bar, 1st we have to import the Link from the react-router-dom and then **inside the Link tag** we have to use **to='/'** to make the link work.

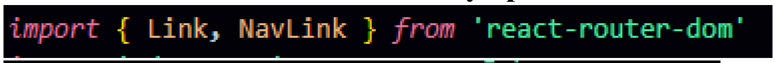


```
import { Link } from 'react-router-dom'
```

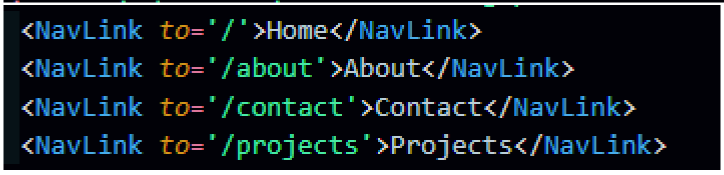


```
<Link to='/' className='cursor-pointer text-lg links font-bold'>Home</Link>
<Link to='/about' className='cursor-pointer text-lg links font-bold'>About</Link>
```

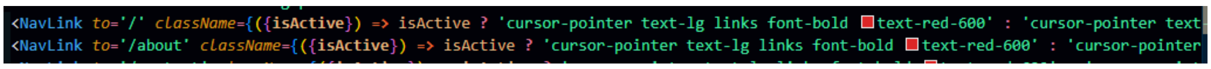
h. To set the active color when you are on some page like if you are on home page then the color of home text in the navigation bar get changed from the original color. To do this we have import **NavLink** from the react-router-dom and inside NavLink we have to use **to='/'** for linking the pages and then **inside the className** we have to use an arrow function in which we will pass the **isActive variable** and we will use the **ternary operator**.



```
import { Link, NavLink } from 'react-router-dom'
```



```
<NavLink to='/'>Home</NavLink>
<NavLink to='/about'>About</NavLink>
<NavLink to='/contact'>Contact</NavLink>
<NavLink to='/projects'>Projects</NavLink>
```



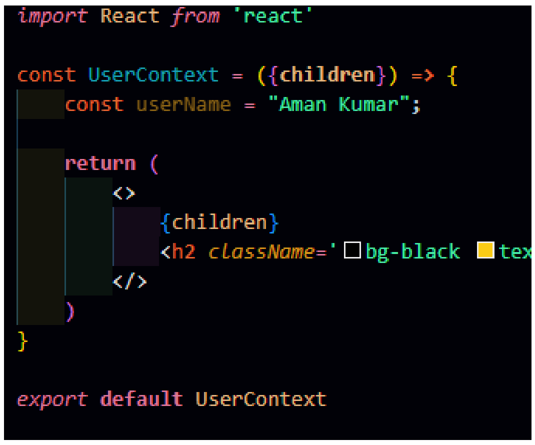
```
<NavLink to='/' className={({isActive}) => isActive ? 'cursor-pointer text-lg links font-bold 🟥text-red-600' : 'cursor-pointer text
<NavLink to='/about' className={({isActive}) => isActive ? 'cursor-pointer text-lg links font-bold 🟥text-red-600' : 'cursor-pointer
```

10. **Context API:** Context API is used to centralise the data and then used by many components. In **props** we have seen that data flow **from top to bottom** only and when. It is also used to **share the data across the components** without passing **props manually** at every level of the component.

a. To use context we will create a context named folder, in which we will create a **user context** file with **jsx**.

b. In **user context jsx file** I have written the code, which I want to be used this as a centralise data for all components.

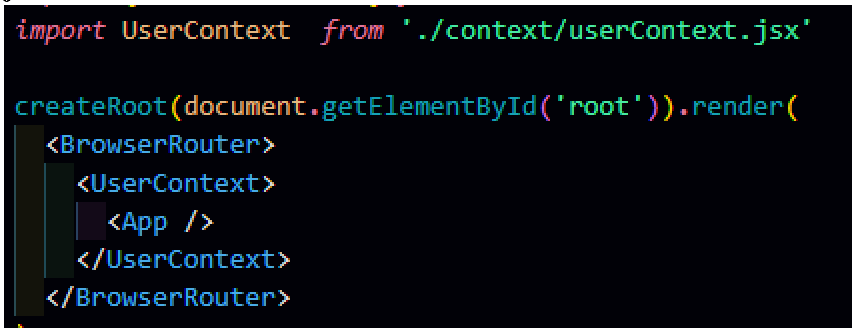
i. It looks like this:



```
import React from 'react'

const UserContext = ({children}) => {
    const userName = "Aman Kumar";

    return (
        <>
            {children}
            <h2 className='⬜bg-black 🟨tex
        </>
    )
}

export default UserContext
```

ii. This **userName** data will be **shared to every component**.

c. To use this data in our **App jsx file** we have **wrap the App component** in **Main jsx file** with user context.



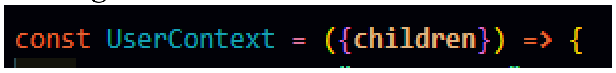
```
import UserContext  from './context/userContext.jsx'

createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <UserContext>
      <App />
    </UserContext>
  </BrowserRouter>
```

d. Now, **App.jsx** become the child of **user context** and whatever things we will do in user context file it will directly reflect in our **App.jsx** output screen.

i. When we will use the **user context** then, whatever things we added in our **App.jsx** will not show because of user context and to fix this all things we want to show on the output screen we have to add in our **user context file** like: Navigation bar, pages, etc.

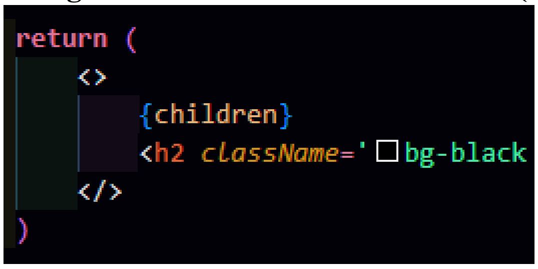
e. The **earlier step is NOT so useful to see our App.jsx** file, to see our App.jsx file without much effort we can **pass the children variable in our user context and can use anywhere we want.**

i. **Passing the variable:**



```
const UserContext = ({children}) => {
```

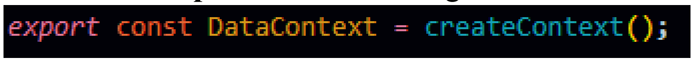
ii. **Using the variable to see the children (App.jsx):**



```
return (
    <>
        {children}
        <h2 className='□bg-black
    </>
)
```

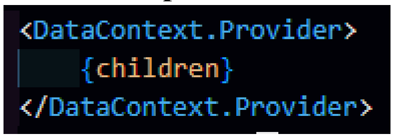
f. To share data across all component we have to:

i. **Create and export a context** using **create context.**



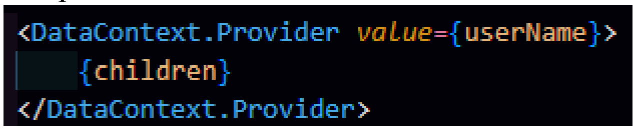
```
export const DataContext = createContext();
```

ii. **We have to provide the context** using **provider.**



```
<DataContext.Provider>
    {children}
</DataContext.Provider>
```

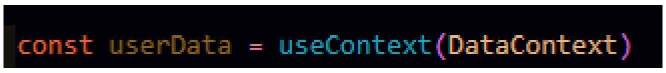
iii. **Then we have to pass the value**, which we want to provide to all the components.



```
<DataContext.Provider value={userName}>
    {children}
</DataContext.Provider>
```

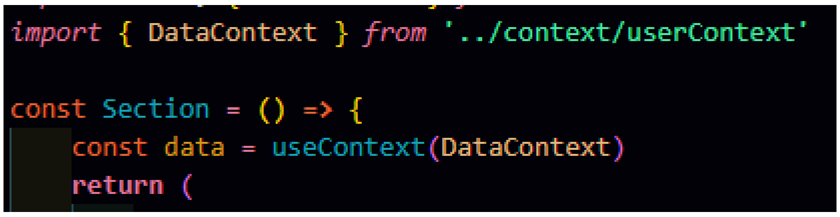
iv. **To use the context wherever we want to use:**

1. We will create a variable in which we will write **use context** and in this use context we will pass our created context.



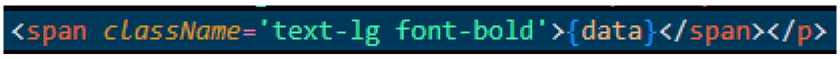
```
const userData = useContext(DataContext)
```

2. To use we will pass our variable in any created **component.**
   - **a.** I want to use this context in my section component, so I will use this context in my section component.



```
import { DataContext } from '../context/userContext'

const Section = () => {
    const data = useContext(DataContext)
    return (
```

   - **b.** To use this data in my section, I will simply pass that variable name anywhere.



```
<span className='text-lg font-bold'>{data}</span></p>
```

Notes completed till

Context API