What is User Authentication & Authorization?

Authentication

Verifying a user's or an entity's identity is the process called **Authentication**. It entails validating the user's credentials, such as a username and password, to ensure that the user is who they claim to be.

Authorization

The process of authorising or refusing access to particular resources or functions within an application is known as **Authorization**. Once a user has been verified as authentic, the program checks their level of authorization to decide which areas of the application they can access

How to Set Up the Project Environment

Create a folder

The folder you just created will contain two sub folders called the `client` and `server`. Run the commands below in your terminal to create the sub folders:

```
mkdir client
```

This will create the `client` sub folder.

```
mkdir server
```

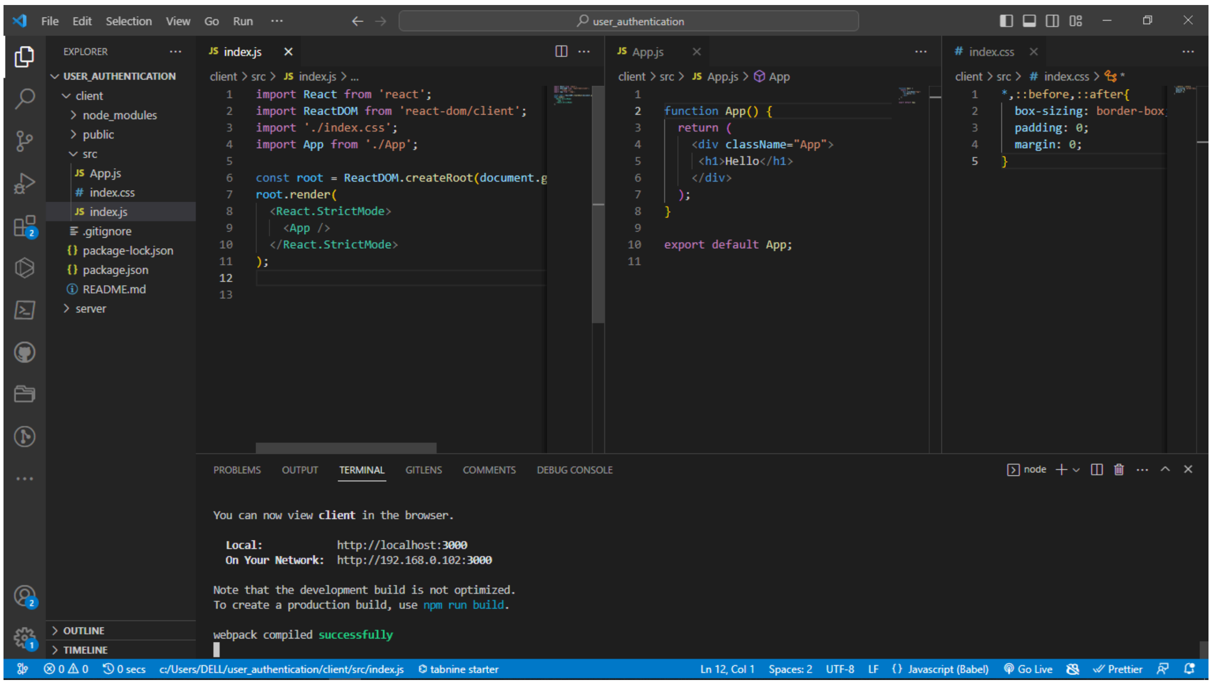
How to Create a New React Application

Open your terminal and run the below command to create a new React application.

But first, you will need to go into the `client` folder using `cd client`, then run the following command:

```
npx create-react-app
```

Before we move to the server directory, you will need to remove some boilerplate in your React application. Your `client` should look like the image below;



File Edit Selection View Go Run
user_authentication
EXPLORER
USER_AUTHENTICATION
client
node_modules
public
src
App.js
index.css
index.js
.gitignore
package-lock.json
package.json
README.md
server
OUTLINE
TIMELINE
PROBLEMS OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE
node
c:/Users/DELL/user_authentication/client/src/index.js
tabnine starter
Ln 12, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) Go Live Prettier

client > src > index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.g
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

```

client > src > App.js > App

```

function App() {
  return (
    <div className="App">
     <h1>Hello</h1>
    </div>
  );
}

export default App;

```

client > src > index.css

```
*,::before,::after{
  box-sizing: border-box
  padding: 0;
  margin: 0;
}
```

```
You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.102:3000

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

webpack compiled successfully
```

- ## **Node.js and Express.js Installation and Configuration**

run `mkdir server` in your terminal to get into the `server` sub folder. After getting into the `server` sub folder, run the following command to initialize the backend application:

```
npm init --yes
```

The `npm init --yes` command in Node.js creates a new `package.json` file for a project with default settings, without asking the user any questions.

The `--yes` or `-y` flag tells npm to use default values for all prompts that would normally appear during the initialization process.

The server folder should now contain a `package.json` file just like so:

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

To install Express.js and other dependencies, run the following commands in your terminal:

```
npm install express cors bcrypt cookie-parser nodemon jsonwebtoken
mongoose dotenv
```

The above commands install the following dependencies:

- `Express.js`, which is our Node.js web application framework.
- `bcrypt`, which helps us hash the user's password.
- `cookie-parser` is the the cookie-parser middleware that handles cookie-based sessions. It extracts information from cookies that may be required for authentication or other purposes.
- `nodemon` is a tool used to automatically restart a Node.js application whenever changes are made to the code.
- `CORS` is a middleware used to enable Cross-Origin Resource Sharing (CORS) for an Express.js application.
- `jsonwebtoken` helps us create and verify JSON Web Tokens.
- `dotenv` allows you to store configuration data in a `.env` file, which is typically not committed to version control, to separate sensitive information from your codebase. This file contains key-value pairs that represent the environment variables.

Create a new file called `index.js` in the root directory of your `server` sub folder of your application. The `index.js` file will contain our Node.js server.

In the `index.js` file of your `server`, add the following code:

```
const express = require("express");

const app = express();
const PORT = 4000;

app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});
```

Before you start the server, update your `package.json` file in the server by adding the code below:

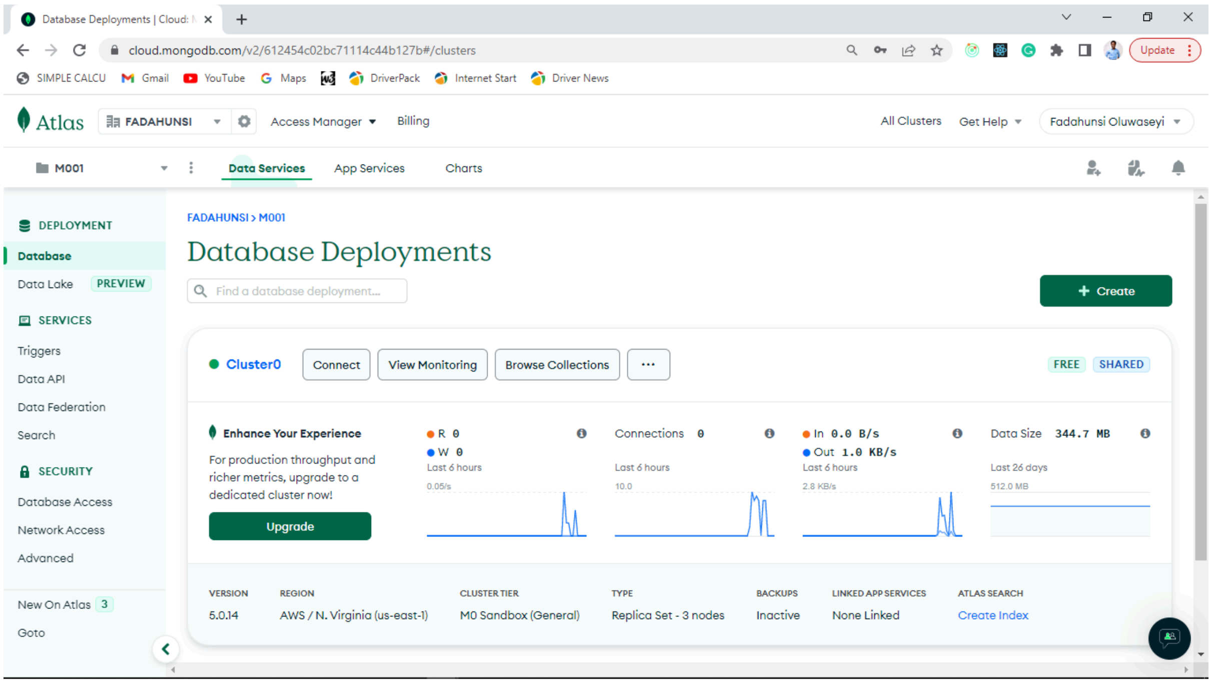
```
 "scripts":{
   start: "nodemon index.js",
   test: 'echo "Error: no test specified" && exit 1',
};
```

This will make sure your application restarts on any update. Now, you can start your `server` by running `npm start` in your terminal.

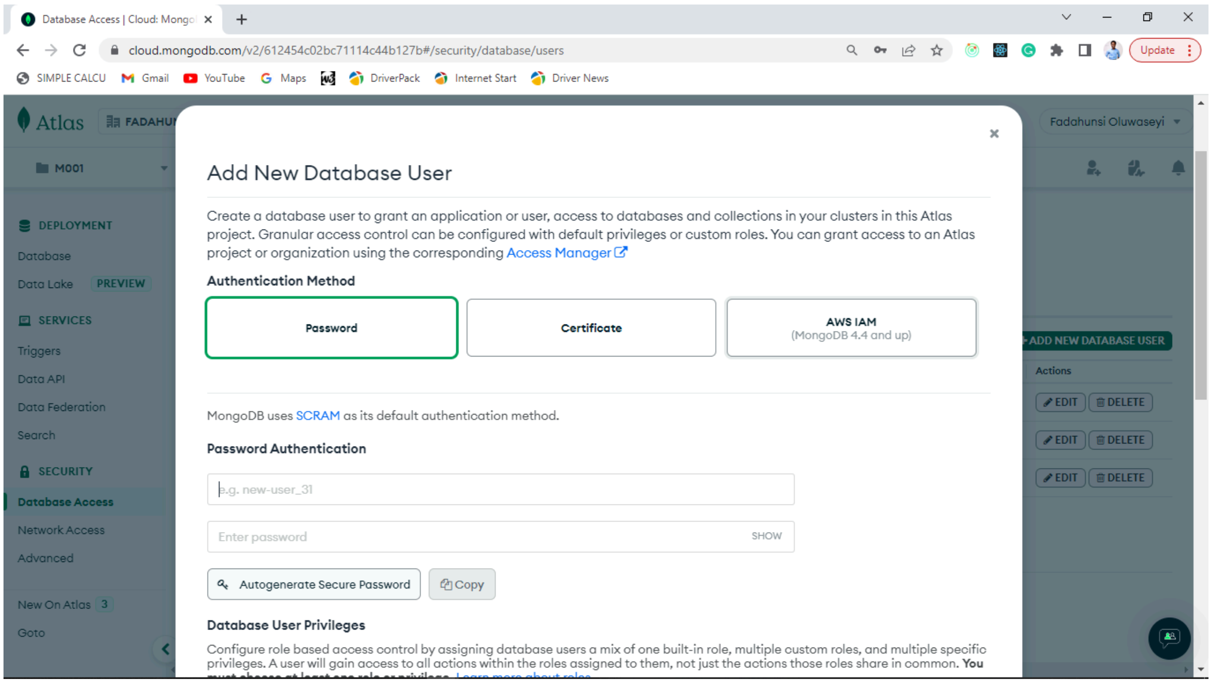
● **Set Up MongoDB**

To link your database to your backend, follow the procedures below.

STEP 1: Go into your MongoDB cloud clusters, which should look like the image below:



STEP 2: Click on the Database Access, which is on the left of the sidebar. Click on `ADD NEW DATABASE USER` which will pop up a modal, like the image below:



Create a database user to grant an application or user, access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding Access Manager

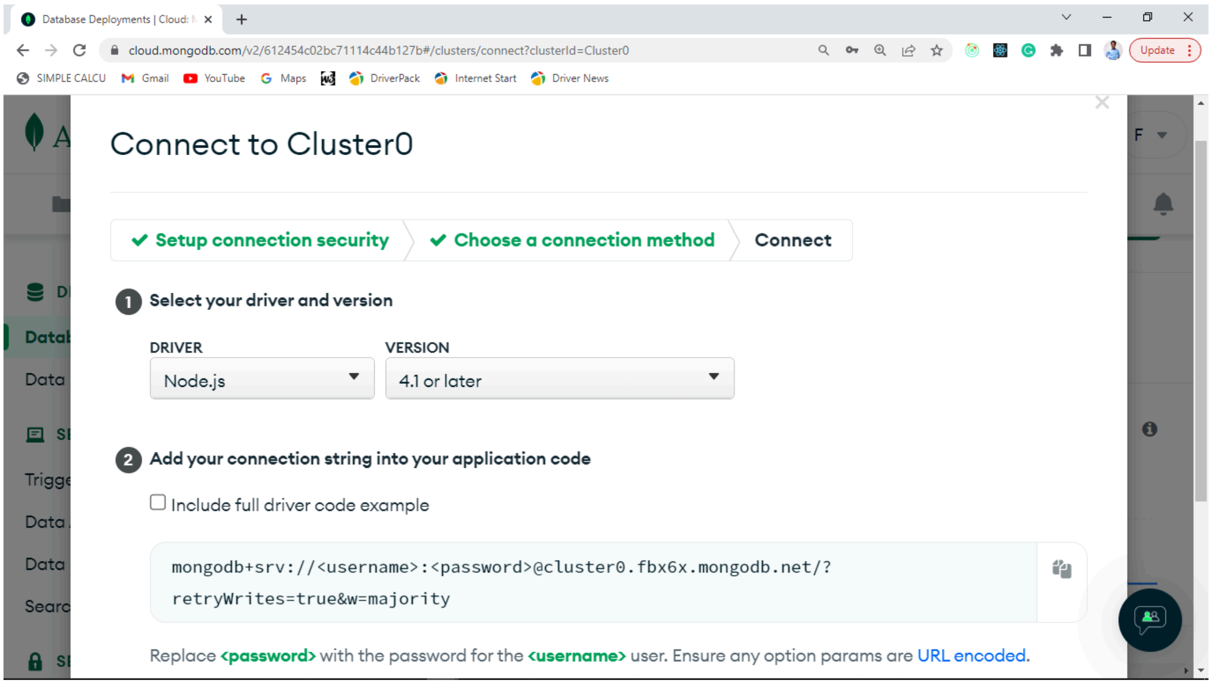
MongoDB uses SCRAM as its default authentication method.

Configure role based access control by assigning database users a mix of one built-in role, multiple custom roles, and multiple specific privileges. A user will gain access to all actions within the roles assigned to them, not just the actions those roles share in common. You

STEP 3: Fill out the `Password Authentication` with your desired username and password for the database of this particular project.

STEP 4: Before saving this, click the `Built-in Role` dropdown, and select `Read and write to any database`. Now, go ahead to click `Add user`.

STEP 5: Click on `Database`, and on the left side of the sidebar, click the `connect` button, which is beside `View Monitoring`. A modal popup will be displayed, then click `connect your application` and copy the code snippet you find there.


Connect to Cluster0

✔ Setup connection security ✔ Choose a connection method Connect

1 Select your driver and version

DRIVER: Node.js
VERSION: 4.1 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://<username>:<password>@cluster0.fbx6x.mongodb.net/?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **<username>** user. Ensure any option params are URL encoded.

You will replace `<username>` and `<password>` with the username and password you created in `STEP 3` in your `index.js` file in the server folder.

Before going into your `index.js` file, you will create a `.env` file in your `server` directory, which will contain your `MONGODB_URL`, `PORT`, `database_name`, and `database_password` like the code below:

```
MONGO_URL =

"mongodb+srv://database_name:database_password@cluster0.fbx6x.mongodb.net/?retryWrites=true
&w=majority";
PORT = 4000;
```

Once you're done with this, go into your `index.js` in your `server` directory, and update it with the code below:

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const app = express();
require("dotenv").config();
const { MONGO_URL, PORT } = process.env;

mongoose
  .connect(MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("MongoDB is  connected successfully"))
  .catch((err) => console.error(err));

app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});

app.use(
  cors({
    origin: ["http://localhost:4000"],
    methods: ["GET", "POST", "PUT", "DELETE"],
    credentials: true,
  })
);

app.use(express.json());
```

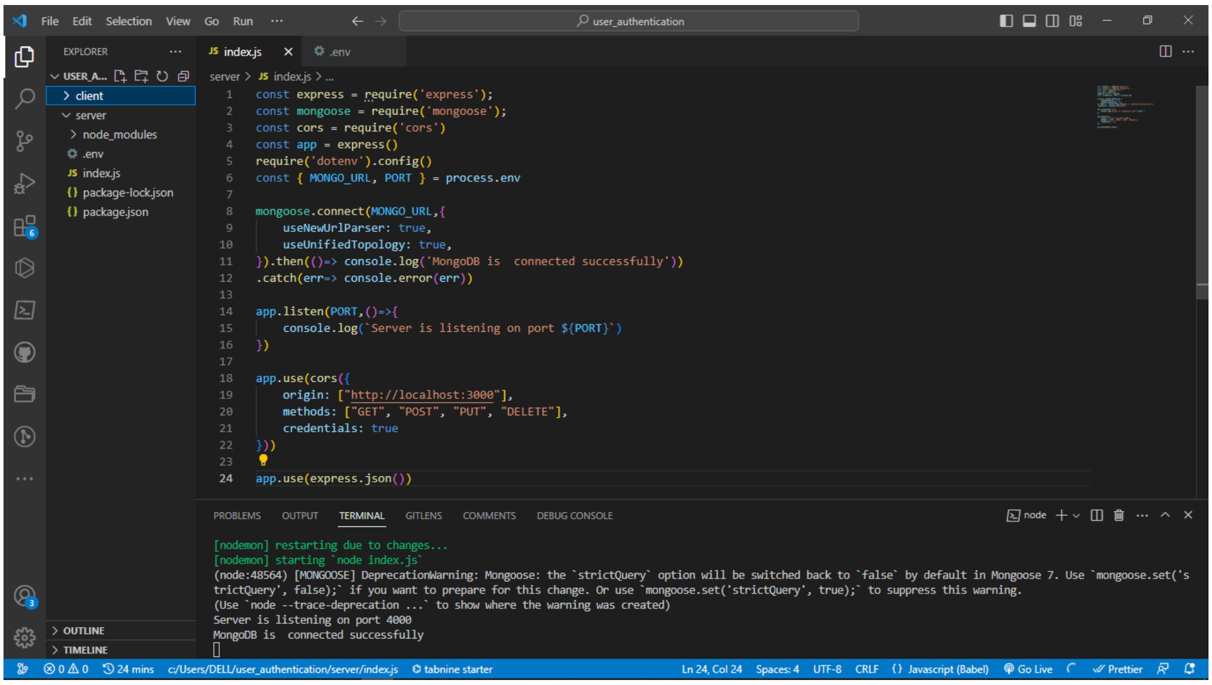
In the code above, we are configuring our application to be able to have access to the `.env` file. You can get the information in your `.env` file by doing `process.env`.

So you're destructing the values from the `.env` file by doing `process.env` so you don't repeat yourself (DRY) which is a good engineering practice.

- CORS (Cross origin resource sharing): You can allow requests from other domains to access the resources on your server by using the `cors()` express middleware function. The CORS headers that your server should include in the response can be specified using the function's optional configuration object parameter, which is taken as a parameter by the function which is the `origin`, `methods` and `credentials`.

- express.json(): The `express.json()` will add a `body` property to the `request` or `req` object. This includes the request body's parsed JSON data. `req.body` in your route handler function will allow you to access this data.
- useNewUrlParser: This property specifies that Mongoose should use the new URL parser to parse MongoDB connection strings. This is set to true by default.
- useUnifiedTopology: This property specifies that Mongoose should use the new Server Discovery and Monitoring engine. This is set to false by default.

After following the steps above, you will restart your application by doing `npm start` in your `server` directory. Your terminal should look like the image below;



```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors')
const app = express()
require('dotenv').config()
const { MONGO_URL, PORT } = process.env

mongoose.connect(MONGO_URL,{
    useNewUrlParser: true,
    useUnifiedTopology: true,
}).then(()=> console.log('MongoDB is  connected successfully'))
.catch(err=> console.error(err))

app.listen(PORT,()=>{
    console.log(`Server is listening on port ${PORT}`)
})

app.use(cors({
    origin: ["http://localhost:3000"],
    methods: ["GET", "POST", "PUT", "DELETE"],
    credentials: true
}))

app.use(express.json())
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
(node:48564) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set('s
trictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is listening on port 4000
MongoDB is  connected successfully
```

• <u>How to Handle the LOGIN Route</u>

Open the `AuthController.js` file in the `Controllers` directory, and update it with the code below:

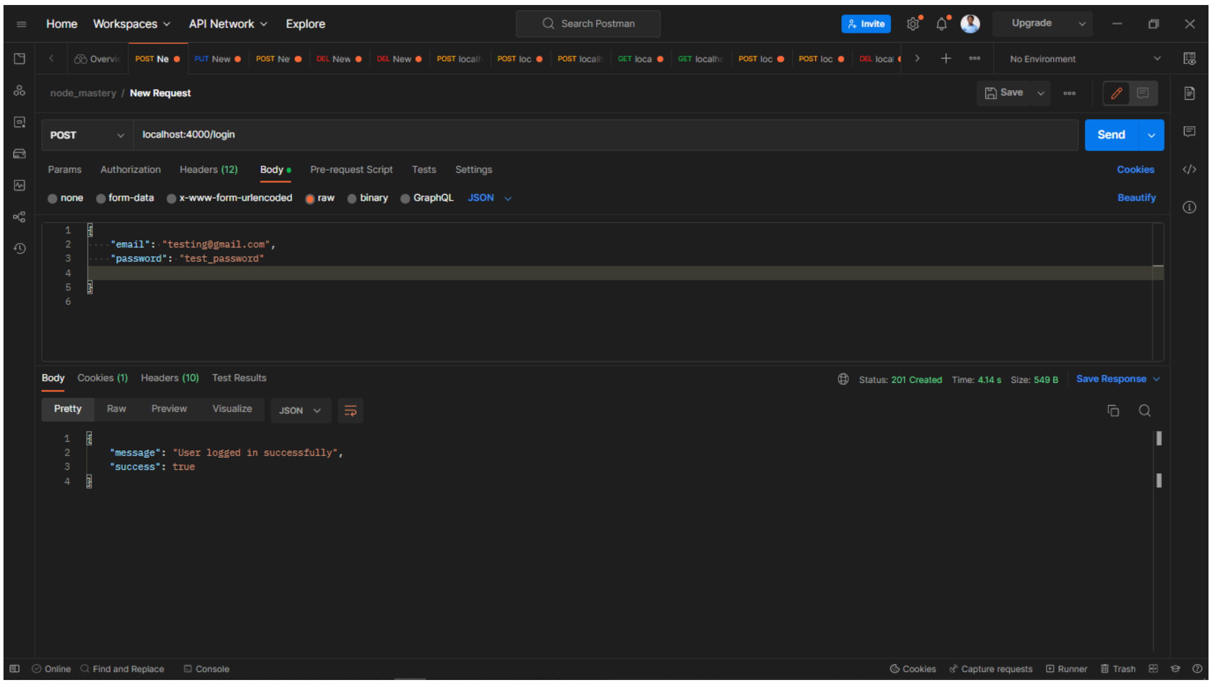
```
module.exports.Login = async (req, res, next) => {
  try {
    const { email, password } = req.body;
    if(!email || !password ){
        return res.json({message:'All fields are required'})
      }
      const user = await User.findOne({ email });
      if(!user){
        return res.json({message:'Incorrect password or email' })
      }
      const auth = await bcrypt.compare(password,user.password)
      if (!auth) {
        return res.json({message:'Incorrect password or email' })
      }
       const token = createSecretToken(user._id);
       res.cookie("token", token, {
         withCredentials: true,
         httpOnly: false,
       });
       res.status(201).json({ message: "User logged in successfully", success: true });
       next()
    } catch (error) {
      console.error(error);
    }
}
```

You are determining in the code above whether the `email` and `password` match any previously stored `user` in the database.

Then add the following code to the file `AuthRoute.js` in the `Routes` directory:

```
const { Signup, Login } = require('../Controllers/AuthController')
const router = require('express').Router()

router.post('/signup', Signup)
router.post('/login', Login)

module.exports = router
```

Now, let's go ahead to test the application:



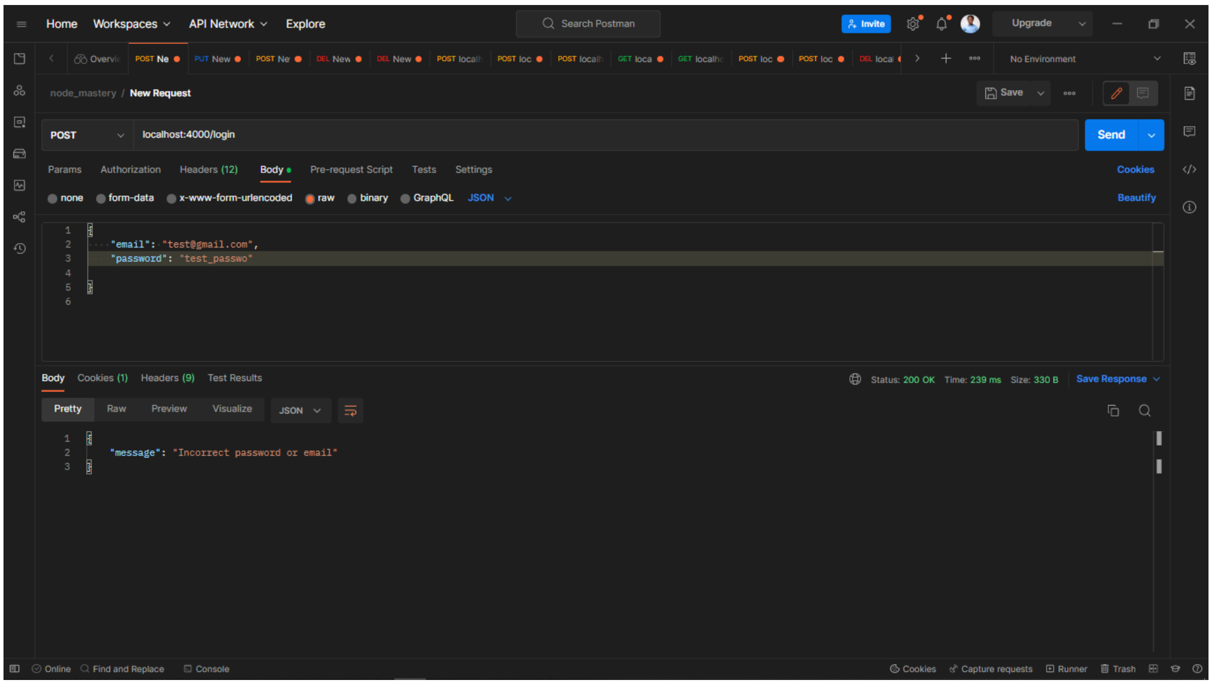
POST localhost:4000/login

```
{
    "email": "testing@gmail.com",
    "password": "test_password"

}
```

Status: 201 Created Time: 4.14 s Size: 549 B

```
{
    "message": "User logged in successfully",
    "success": true
}
```

If you try to use an unregistered `email` or `password`, you'll get the message below:



POST localhost:4000/login

```
{
    "email": "test@gmail.com",
    "password": "test_passwo"

}
```

Status: 200 OK Time: 239 ms Size: 330 B

```
{
    "message": "Incorrect password or email"
}
```

• How to Handle the HOME Route

Now, you will create a `AuthMiddleware.js` file, in the `Middlewares` directory, and paste in the code below:

```
const User = require("../Models/UserModel");
require("dotenv").config();
```

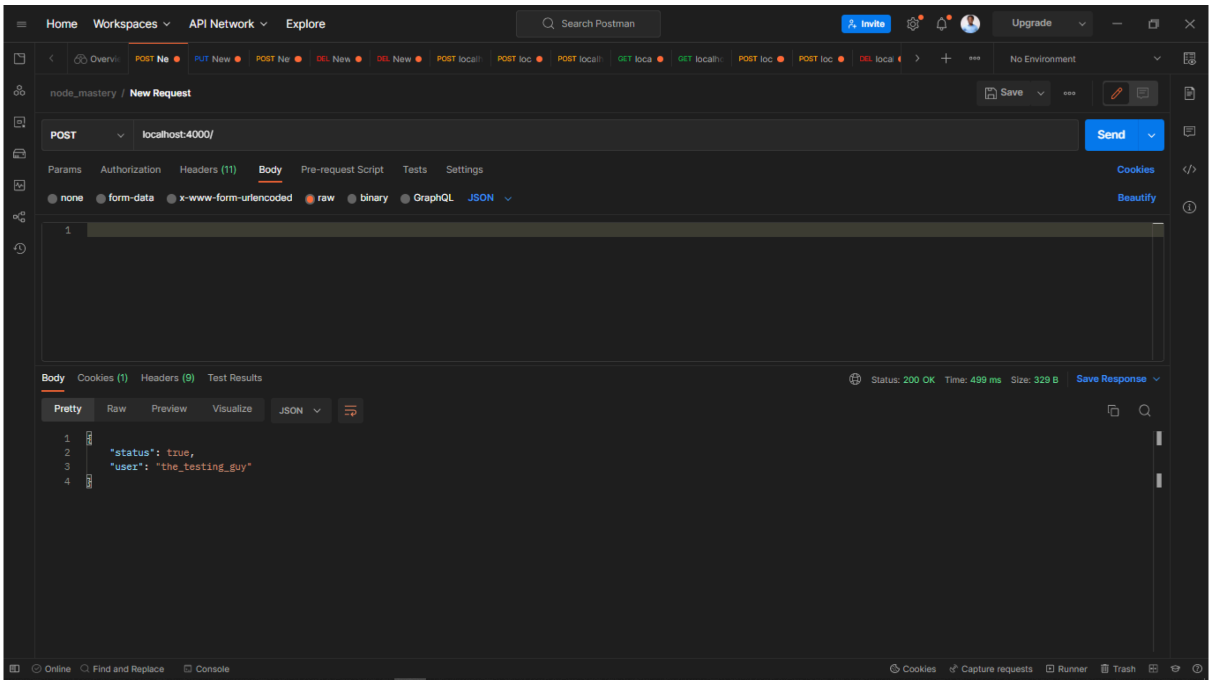
```
const jwt = require("jsonwebtoken");

module.exports.userVerification = (req, res) => {
  const token = req.cookies.token
  if (!token) {
    return res.json({ status: false })
  }
  jwt.verify(token, process.env.TOKEN_KEY, async (err, data) => {
    if (err) {
     return res.json({ status: false })
    } else {
      const user = await User.findById(data.id)
      if (user) return res.json({ status: true, user: user.username })
      else return res.json({ status: false })
    }
  })
}
```

The code above checks if the user has access to the route by checking if the `token`s match.

Next, update the `AuthRoute.js` file in the `Routes` directory with the code below:

```
router.post('/',userVerification)
```

Now, you can go ahead to test your route. It should look like the image below:



node_mastery / New Request

POST localhost:4000/

Body | Cookies (1) | Headers (9) — Status: 200 OK Time: 499 ms Size: 329 B

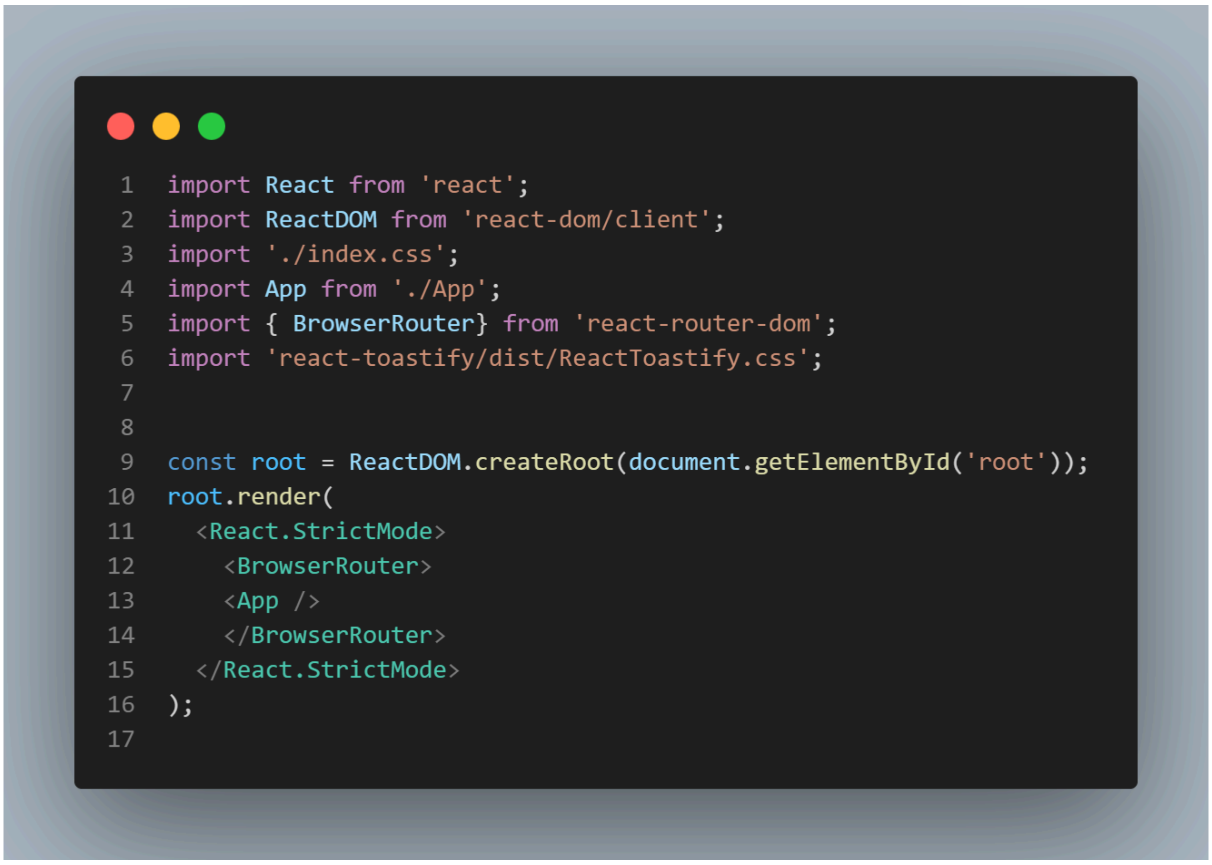
```
{
    "status": true,
    "user": "the_testing_guy"
}
```

● <u>**How to Implement the Frontend**</u>

To get started, go into the `client` directory and install the following in your terminal:

```
npm install react-cookie react-router-dom react-toastify axios
```

Now, update the `index.js` file in the `client` directory with the code snippet below:



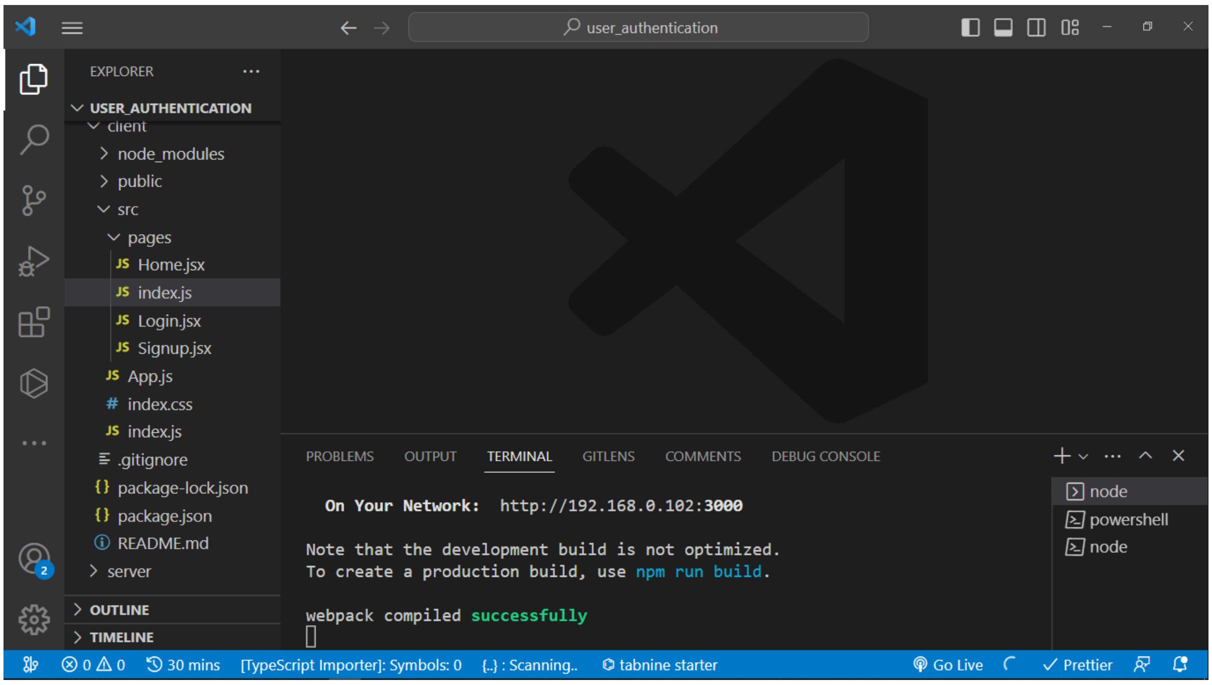
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter} from 'react-router-dom';
import 'react-toastify/dist/ReactToastify.css';


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <App />
    </BrowserRouter>
  </React.StrictMode>
);

```

In the code above, wrapping your `App` component with `BrowserRouter` is necessary to enable client-side routing and take advantage of its benefits in your application.

NB: Remove the `React.StrictMode` later when you are testing the application and your data is being fetched twice.

Also, import `react-toastify` so it can be available in your application.

Now, go ahead to create the `pages` directory in your `client` directory, which will contain the `Home.jsx` file, `Login.jsx` file, `Signup.jsx` and `index.js` to export the components. Your folder should look like the image below:



```
  On Your Network:  http://192.168.0.102:3000

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

webpack compiled successfully
```

Now, fill the `Login.jsx`, `Signup.jsx`, and `Home.jsx`, respectively, with the code below. These snippets below, are functional components which will be modified later in this guide.

NB: This can be automatically generated by typing the shortcut `rafce` + `enter` in the file you want to add the snippet in your visual studio code editor. Make sure this extension is installed in your visual studio code for this to work.

`Login.jsx`:

```
import React from "react";

const Login = () => {
  return <h1>Login Page</h1>;
};

export default Login
```

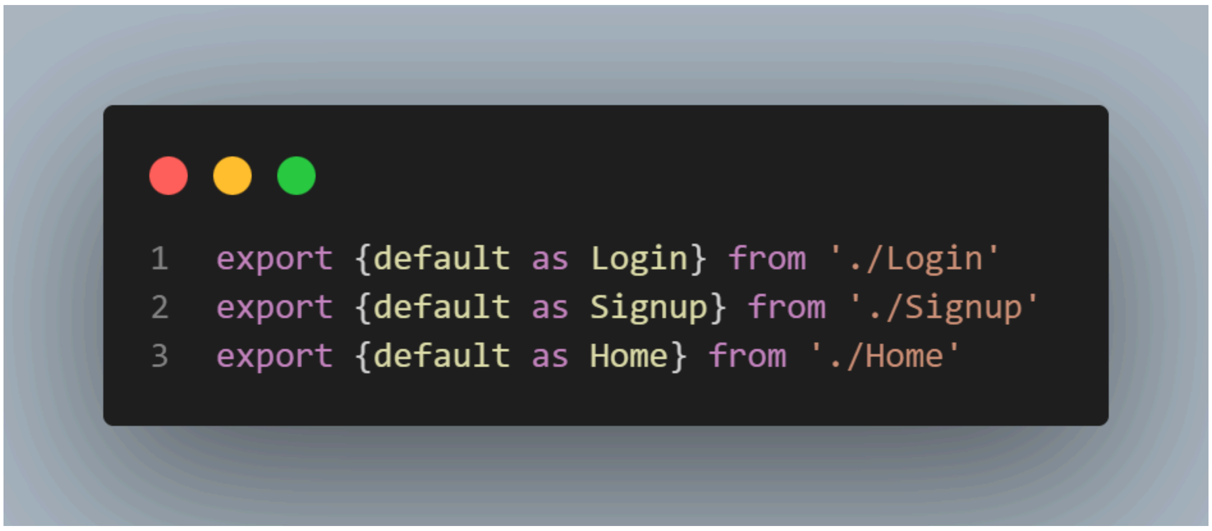
`Signup.jsx`:

```
import React from "react";

const Signup = () => {
  return <h1>Signup Page</h1>;
};

export default Signup
```

`Home.jsx`:

```
import React from "react";

const Home = () => {
  return <h1>Home PAGE</h1>;
};

export default Home
```

After that's done, you will go into the `index.js` file in the `pages` directory to export the newly created components. It should look like the image below:



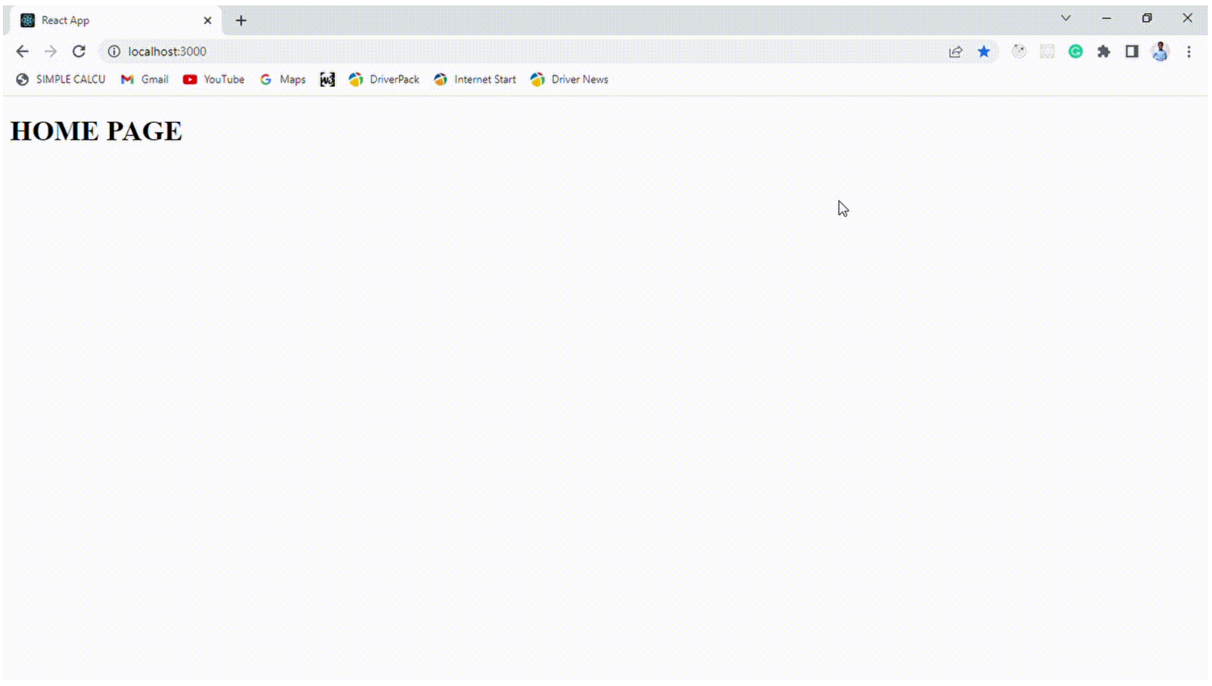
```
export {default as Login} from './Login'
export {default as Signup} from './Signup'
export {default as Home} from './Home'
```

The method shown above makes importing components easier by requiring only one import line.

Now, update the `App.js` file in the `src` directory with the code below.

```
import { Route, Routes } from "react-router-dom";
import { Login, Signup } from "./pages";
import Home from "./pages/Home";

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
      </Routes>
    </div>
  );
}

export default App;
```

The routes will be made available in your application using the above code. The example below will help to clarify:



HOME PAGE

● <u>How to Handle the Signup Logic</u>

In the `Signup.jsx` file in the `pages` directory, paste the following code snippet:

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import axios from "axios";
import { ToastContainer, toast } from "react-toastify";

const Signup = () => {
  const navigate = useNavigate();
  const [inputValue, setInputValue] = useState({
    email: "",
    password: "",
    username: "",
  });
  const { email, password, username } = inputValue;
  const handleOnChange = (e) => {
    const { name, value } = e.target;
    setInputValue({
      ...inputValue,
      [name]: value,
    });
  };

  const handleError = (err) =>
    toast.error(err, {
      position: "bottom-left",
    });
  const handleSuccess = (msg) =>
    toast.success(msg, {
      position: "bottom-right",
    });

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post(
        "http://localhost:4000/signup",
        {
          ...inputValue,
        },
        { withCredentials: true }
      );
      const { success, message } = data;
      if (success) {
        handleSuccess(message);
        setTimeout(() => {
          navigate("/");
        }, 1000);
      } else {
        handleError(message);
      }
    } catch (error) {
      console.log(error);
    }
    setInputValue({
      ...inputValue,
      email: "",
      password: "",
      username: "",
    });
  };

  return (
    <div className="form_container">
      <h2>Signup Account</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="email">Email</label>
          <input
            type="email"
            name="email"
            value={email}
            placeholder="Enter your email"
            onChange={handleOnChange}
          />
        </div>
        <div>
          <label htmlFor="email">Username</label>
          <input
            type="text"
            name="username"
            value={username}
            placeholder="Enter your username"
            onChange={handleOnChange}
          />
        </div>
        <div>
          <label htmlFor="password">Password</label>
          <input
            type="password"
            name="password"
            value={password}
            placeholder="Enter your password"
            onChange={handleOnChange}
          />
        </div>
        <button type="submit">Submit</button>
        <span>
        Already have an account? <Link to={"/login"}>Login</Link>
      </span>
    </form>
    <ToastContainer />
  </div>
);
};

export default Signup;
```

● How to Handle the Login Logic

Add the following code snippet to the `Login.jsx` file in the `pages` directory:

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import axios from "axios";
import { ToastContainer, toast } from "react-toastify";

const Login = () => {
  const navigate = useNavigate();
  const [inputValue, setInputValue] = useState({
    email: "",
    password: "",
  });
  const { email, password } = inputValue;
  const handleOnChange = (e) => {
    const { name, value } = e.target;
    setInputValue({
      ...inputValue,
      [name]: value,
    });
  };

  const handleError = (err) =>
    toast.error(err, {
      position: "bottom-left",
    });
  const handleSuccess = (msg) =>
    toast.success(msg, {
      position: "bottom-left",
    });

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post(
        "http://localhost:4000/login",
        {
          ...inputValue,
        },
        { withCredentials: true }
      );
      console.log(data);
      const { success, message } = data;
      if (success) {
        handleSuccess(message);
        setTimeout(() => {
          navigate("/");
        }, 1000);
      } else {
        handleError(message);
      }
    } catch (error) {
      console.log(error);
    }
    setInputValue({
      ...inputValue,
      email: "",
      password: "",
    });
  };

  return (
    <div className="form_container">
      <h2>Login Account</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="email">Email</label>
          <input
            type="email"
            name="email"
            value={email}
            placeholder="Enter your email"
            onChange={handleOnChange}
          />
        </div>
        <div>
          <label htmlFor="password">Password</label>
          <input
            type="password"
            name="password"
```

```
              value={password}
              placeholder="Enter your password"
              onChange={handleOnChange}
            />
          </div>
          <button type="submit">Submit</button>
          <span>
            Already have an account? <Link to={"/signup"}>Signup</Link>
          </span>
        </form>
        <ToastContainer />
      </div>
  );
};

export default Login;
```

• How to Handle the Home Page Logic

Copy and paste the following code snippet into the `Home.jsx` file located in the `pages` directory:

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { useCookies } from "react-cookie";
import axios from "axios";
import { ToastContainer, toast } from "react-toastify";

const Home = () => {
  const navigate = useNavigate();
  const [cookies, removeCookie] = useCookies([]);
  const [username, setUsername] = useState("");
  useEffect(() => {
    const verifyCookie = async () => {
      if (!cookies.token) {
        navigate("/login");
      }
      const { data } = await axios.post(
        "http://localhost:4000",
        {},
        { withCredentials: true }
      );
      const { status, user } = data;
```

```
        setUsername(user);
        return status
          ? toast(`Hello ${user}`, {
              position: "top-right",
            })
          : (removeCookie("token"), navigate("/login"));
      };
      verifyCookie();
    }, [cookies, navigate, removeCookie]);
    const Logout = () => {
      removeCookie("token");
      navigate("/signup");
    };
    return (
      <>
        <div className="home_page">
          <h4>
            {" "}
            Welcome <span>{username}</span>
          </h4>
          <button onClick={Logout}>LOGOUT</button>
        </div>
        <ToastContainer />
      </>
    );
  };

  export default Home;
```

Ensure that the styles below are copied into your `index.css` file:

```
*,
::before,
::after {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}

label {
  font-size: 1.2rem;
  color: #656262;
}

html,
body {
```

```css
  height: 100%;
  width: 100%;
}

body {
  display: flex;
  justify-content: center;
  align-items: center;
  background: linear-gradient(
    90deg,
    rgba(2, 0, 36, 1) 0%,
    rgba(143, 187, 204, 1) 35%,
    rgba(0, 212, 255, 1) 100%
  );
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}

.form_container {
  background-color: #fff;
  padding: 2rem 3rem;
  border-radius: 0.5rem;
  width: 100%;
  max-width: 400px;
  box-shadow: 8px 8px 24px 0px rgba(66, 68, 90, 1);
}

.form_container > h2 {
  margin-block: 1rem;
  padding-block: 0.6rem;
  color: rgba(0, 212, 255, 1);
}

.form_container > form {
  display: flex;
  flex-direction: column;
  gap: 1.4rem;
}

.form_container div {
  display: flex;
  flex-direction: column;
  gap: 0.3rem;
}

.form_container input {
  border: none;
  padding: 0.5rem;
  border-bottom: 1px solid gray;
  font-size: 1.1rem;
  outline: none;
```

```css
}

.form_container input::placeholder {
  font-size: 0.9rem;
  font-style: italic;
}

.form_container button {
  background-color: rgba(0, 212, 255, 1);
  color: #fff;
  border: none;
  padding: 0.6rem;
  font-size: 1rem;
  cursor: pointer;
  border-radius: 0.3rem;
}

span a {
  text-decoration: none;
  color: rgba(0, 212, 255, 1);
}

.home_page {
  height: 100vh;
  width: 100vw;
  background: #000;
  color: white;
  display: flex;
  justify-content: center;
  align-items: center;
  text-transform: uppercase;
  font-size: 3rem;
  flex-direction: column;
  gap: 1rem;
}

.home_page span {
  color: rgba(0, 212, 255, 1);
}

.home_page button {
  background-color: rgb(27, 73, 83);
  color: #fff;
  cursor: pointer;
  padding: 1rem 3rem;
  font-size: 2rem;
  border-radius: 2rem;
  transition: ease-in 0.3s;
  border: none;
}
```

```css
.home_page button:hover {
  background-color: rgba(0, 212, 255, 1);
}


@media only screen and (max-width: 1200px){
  .home_page{
    font-size: 1.5rem;
  }
  .home_page button {
    padding: 0.6rem 1rem;
    font-size: 1.5rem;
  }

}
```