Authrization and athentication app

```
backend/
│
├── controllers/
│   ├── authController.js
│   ├── featureController.js
│
├── middleware/
│   ├── authMiddleware.js
│
├── models/
│   ├── User.js
│
├── routes/
│   ├── authRoutes.js
│   ├── featureRoutes.js
│
├── app.js
├── .env
├── package.json
└── .gitignore
```

```
//authController.js

const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

// Register User
// This is an asynchronous function that handles a user registration
request
exports.register = async (req, res) => {

    // Create a new user object from the request body
    const ruser = {

        // Extract the username, email, password, and role from the
request body
        username : req.body.username,
        email: req.body.email,
        password : req.body.password,
        role : req.body.role
    }

    try {

        // Check if a user with the same username already exists
        const existingUser = await User.findOne({ username:
ruser.username });

        // If a user with the same username exists, return a 400 error
with a message
        if (existingUser) return res.status(400).json({ message: "User
already exists" });

        // Hash the user's password using bcrypt
        const hashedPassword = await bcrypt.hash(ruser.password, 10);

        // Replace the plain text password with the hashed password
        ruser.password = hashedPassword;

        // Create a new User document from the ruser object
        const newUser = new User(ruser);
```

```
        // Save the new user to the database
        const createduser = await newUser.save();

        // Return a 201 success response with a message and the created
user
        return res.status(201).json({ message: "User registered
successfully", createduser });

    } catch (error) {
        // Log the error message to the console
        console.log("register error", error.message);

        // Return a 500 error with a message
        res.status(500).json({ message: "Server error" });
    }
};




// Login User
exports.login = async (req, res) => {
    // Extract the username and password from the request body
    const { username, password } = req.body;

    try {
        // Find a user with the given username
        const user = await User.findOne({ username });

        // If no user is found, return a 400 error with a message
        if (!user) return res.status(400).json({ message: "User not
found" });

        // Compare the provided password with the stored password using
bcrypt
        const isMatch = await bcrypt.compare(password, user.password);

        // If the passwords don't match, return a 400 error with a
message
        if (!isMatch) return res.status(400).json({ message: "Invalid
credentials" });
```

```
        // Generate a JSON Web Token (JWT) with the user's ID and role
        const token = jwt.sign({ id: user._id, role: user.role },
process.env.JWT_SECRET, { expiresIn: '1h' });

        user.password = undefined;

        // Return a 200 success response with a message and the user
        return res.status(200).json({ message: "Login successful",
token, user });
        // Return the token in the response
        // res.json({ token });

    } catch (error) {
        // Log the error message to the console
        console.log("login error", error.message);

        // Return a 500 error with a message
        res.status(500).json({ message: "Server error" });
    }
};
```

//featureController.js

```
// Example Features for Users and Admins

// Feature 1 (User + Admin)
exports.featureOne = (req, res) => {
    res.json({ message: "Feature One (User and Admin)" });
};

// Feature 2 (User + Admin)
exports.featureTwo = (req, res) => {
    res.json({ message: "Feature Two (User and Admin)" });
};

// Feature 3 (Admin Only)
exports.adminFeatureOne = (req, res) => {
    res.json({ message: "Admin Feature One (Admin Only)" });
};

// Feature 4 (Admin Only)
exports.adminFeatureTwo = (req, res) => {
    res.json({ message: "Admin Feature Two (Admin Only)" });
};
```

```
// authMiddleware.js



// Import the jsonwebtoken library, which is used to verify and decode
JWT tokens
const jwt = require('jsonwebtoken');

// Verify JWT Token
// Define a middleware function to verify JWT tokens
exports.authMiddleware = (req, res, next) => {
    // Extract the JWT token from the Authorization header, if it
exists
    const token = req.header('Authorization')?.split(' ')[1];

     // If no token is found, return a 401 error response
    if (!token) return res.status(401).json({ message: "No token,
authorization denied" });

    try {
        // Verify the JWT token using the secret key stored in the
environment variables
        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        // If the token is valid, extract the user data from the
decoded token and attach it to the request object
        req.user = decoded;

        // Call the next middleware function in the chain
        next();

    } catch (error) {
        // If the token is invalid or has expired, return a 400 error
response
        res.status(400).json({ message: "Invalid token" });
    }
};

// Role-based Authorization Middleware
// Define a middleware function to enforce role-based authorization
exports.roleMiddleware = (requiredRole) => {
```

```
    // Return a new middleware function that takes the request,
response, and next function as arguments
    return (req, res, next) => {

        // Check if the user's role matches the required role
        if (req.user.role !== requiredRole) {

            // If the user's role does not match, return a 403 error
response
            return res.status(403).json({ message: "Access denied" });
        }

        // If the user's role matches, call the next middleware
function in the chain
        next();
    };
};
```

// User.js

```
// Import the mongoose module, which is used to interact with the MongoDB database
const mongoose = require('mongoose');

// Define user schema
// Define a new schema for the User model using the mongoose.Schema constructor
const UserSchema = new mongoose.Schema({
    // Define a field for the username, which is a required string that must be unique
    username: { type: String, required: true, unique: true },

    // Define a field for the email, which is a string but not required
    email: { type: String },

    // Define a field for the password, which is a required string
    password: { type: String, required: true },

    // Define a field for the role, which is a string that can only be one of the values in the enum array
    // The default value is 'user' if no value is provided
    role: { type: String, enum: ['user', 'admin'], default: 'user' }
});

// Create a new model called 'User' based on the UserSchema
const User = mongoose.model('User', UserSchema);

// Export the User model to make it available for use in other modules
module.exports = User;
```

//authRoutes.js

```
// Import the express module, which is used to create an Express
application
const express = require('express');

// Import the register and login functions from the authController
module
const { register, login } = require('../controllers/authController');

// Create a new router object using the express.Router() function
const router = express.Router();

// Define a POST route for the '/register' endpoint that calls the
register function
router.post('/register', register);

// Define a POST route for the '/login' endpoint that calls the login
function
router.post('/login', login);

// Export the router object to make it available for use in other
modules
module.exports = router;
```

```
// featureRoutes.js


// Import the express module, which is used to create an Express
application
const express = require('express');

// Import the featureOne, featureTwo, adminFeatureOne, and
adminFeatureTwo functions from the featureController module
const { featureOne, featureTwo, adminFeatureOne, adminFeatureTwo } =
require('../controllers/featureController');

// Import the authMiddleware and roleMiddleware functions from the
authMiddleware module
const { authMiddleware, roleMiddleware } =
require('../middleware/authMiddleware');

// Create a new router object using the express.Router() function
const router = express.Router();



// User and Admin Features
// Define routes for user and admin features that require
authentication
// The authMiddleware function is used to verify the user's
authentication status
router.get('/feature-one', authMiddleware, featureOne);    // Route for
feature one, accessible to all authenticated users
router.get('/feature-two', authMiddleware, featureTwo);    // Route for
feature two, accessible to all authenticated users

// Admin Only Features
// Define routes for admin-only features that require both
authentication and admin role
// The roleMiddleware function is used to verify the user's role
router.get('/admin-feature-one', authMiddleware,
roleMiddleware("admin"), adminFeatureOne);    // Route for admin feature
one, accessible only to admins
router.get('/admin-feature-two', authMiddleware,
roleMiddleware("admin"), adminFeatureTwo);    // Route for admin feature
two, accessible only to admins
```

```
// Export the router object to make it available for use in other
modules
module.exports = router;
```

// app.js

```
// Import the express module, which is used to create an Express
application
const express = require('express');

// Import the mongoose module, which is used to interact with the
MongoDB database
const mongoose = require('mongoose');

// Import the cors module, which is used to enable CORS (Cross-Origin
Resource Sharing) in the application
const cors = require('cors');

// Load environment variables from the .env file using the dotenv
module
require('dotenv').config();

// Import routes for authentication and features
const authRoutes = require('./routes/authRoutes');
const featureRoutes = require('./routes/featureRoutes');

// Create a new Express application
const app = express();

// Middleware
// Enable CORS in the application to allow requests from different
origins
app.use(cors());

// Enable JSON parsing for incoming requests
app.use(express.json());

// Routes
// Mount the authentication routes at the /api/auth endpoint
app.use('/api/auth', authRoutes);
```

```
// Mount the feature routes at the /api/features endpoint
app.use('/api/features', featureRoutes);

// MongoDB Connection
// Connect to the MongoDB database using the MONGO_URI environment
variable
mongoose.connect(process.env.MONGO_URI, {

    // Use the useNewUrlParser option to enable the new URL parser
    useNewUrlParser: true,

    // Use the useUnifiedTopology option to enable the new connection
topology engine
    useUnifiedTopology: true })

    // Log a success message to the console if the connection is
established
    .then(() => console.log('MongoDB connected'))

    // Log an error message to the console if the connection fails
    .catch((err) => console.log(err));

// Set the port number for the application to listen on
const PORT = process.env.PORT || 5000;

// Start the application and listen on the specified port
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

```
//Packege.json

{
  "name": "backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.21.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.7.0",
    "router": "^1.3.8"
  },
  "devDependencies": {
    "nodemon": "^3.1.7"
  },
  "description": ""
}
```

Authrization and athentication app

```
frontend/
├── src/
│   ├── components/
│   │   ├── Login.js
│   │   ├── Register.js
│   │   ├── Navbar.js
│   │   ├── FeatureOne.js
│   │   ├── AdminFeatureOne.js
│   ├── App.js
│   ├── index.js
├── package.json
└── .gitignore
```

//App.js

```
// Import the React library, which is the foundation for building user interfaces
import React from 'react';

// Import the BrowserRouter, Route, and Routes components from the react-router-dom library
// BrowserRouter is used to enable client-side routing, while Route and Routes are used to define routes
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

// Import the Register component, which is used to handle user registration
import Register from './components/Register';

// Import the Login component, which is used to handle user login
import Login from './components/Login';

// Import the FeatureOne component, which is a feature available to all users
import FeatureOne from './components/FeatureOne';

// Import the AdminFeatureOne component, which is a feature available only to administrators
import AdminFeatureOne from './components/AdminFeatureOne';

// Import the Navbar component, which is used to display navigation links
import Navbar from './components/Navbar';

// Import the App.css file, which is used to style the application
import './App.css';

// Define the App function component, which is the top-level component of the application
function App() {
    // Return the JSX that defines the application's structure
    return (
```

```
        // Use the BrowserRouter component to enable client-side
routing
        // Render the Navbar component at the top of the application
        // Use the Routes component to define a collection of routes
        // Define a route for the registration page
        // Define a route for the login page
        // Define a route for the feature-one page
        // Define a route for the admin-feature-one page
        <Router>
            <Navbar />
            <Routes>
                <Route path="/register" element={<Register />} />
                <Route path="/login" element={<Login />} />
                <Route path="/feature-one" element={<FeatureOne />} />
                <Route path="/admin-feature-one"
element={<AdminFeatureOne />} />
            </Routes>
        </Router>
    );
}

// Export the App component as the default export
export default App;
```

//Register.js

```
// Import the React library and its useState and useEffect hooks
import React, { useState, useEffect } from 'react';

// Import the axios library, which is used to make HTTP requests
import axios from 'axios';

// Import the useNavigate hook from the react-router-dom library
import { useNavigate } from 'react-router-dom';

// Import the register.css file, which is used to style the component
import "../CssFiles/register.css";

// Define a functional component called Register
const Register = () => {
    // Initialize state variables for the username, password, email,
role, and message
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [email, setEmail] = useState('');
    const [role, setRole] = useState('user');
    const [message, setMessage] = useState('');

    // Check if the user is already authenticated by checking for a
token in local storage
    const isAuthenticated = localStorage.getItem("token");

    // Get the navigate function from the useNavigate hook
    const navigate = useNavigate()

    // Use the useEffect hook to perform a side effect when the
component mounts
    useEffect(() => {
        // If the user is already authenticated, navigate to the
feature-one page
        if (isAuthenticated) {
            return navigate("/feature-one");
        }
        // If the user is not authenticated, navigate to the register
page
```

```
        else {
            return navigate("/register");
        }
    }, [isAuthenticated, navigate]); // The dependency array includes
isAuthenticated and navigate

    // Define a function to handle the register form submission
    const handleRegister = async () => {
        try {
            // Make a POST request to the register endpoint with the
username, email, password, and role
            const response = await axios.post('/api/auth/register', {
username, email, password, role });

            // Set a success message
            setMessage(response.data.message);

            // Navigate to the login page
            navigate('/login');
        } catch (error) {
            // Set an error message if the registration fails
            setMessage(error.response?.data?.message || "Error
registering");
        }
    };

    // Render the component
    return (
        <div className="register">
            <div className="register-inner">
                <div className="register-header">Register</div>

                {/* Input field for the username */}
                <input type="text" placeholder="Username"
value={username} onChange={(e) => setUsername(e.target.value)} />

                {/* Input field for the email */}
                <input type="email" placeholder="Email" value={email}
onChange={(e) => setEmail(e.target.value)} />

                {/* Input field for the password */}
                <input type="password" placeholder="Password"
value={password} onChange={(e) => setPassword(e.target.value)} />
```

```
                {/* Select field for the role */}
                <select className='select' placeholder="Role"
value={role} onChange={(e) => setRole(e.target.value)}>
                    <option value="user">User</option>
                    <option value="admin">Admin</option>
                </select>

                {/* Register button */}
                <button className="btn"
onClick={handleRegister}>Register</button>
            </div>

            {/* Display the message */}
            {message && <p>{message}</p>}
        </div>
    );
};

// Export the Register component as the default export
export default Register;
```

//Login.js

```
// Import the React library and its useState and useEffect hooks
import React, { useState, useEffect } from 'react';

// Import the axios library, which is used to make HTTP requests
import axios from 'axios';

// Import the useNavigate hook from the react-router-dom library
import { useNavigate } from 'react-router-dom';

// Import the login.css file, which is used to style the component
import "../CssFiles/login.css";

// Define a functional component called Login
const Login = () => {
    // Initialize state variables for the username, password, and
message
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [message, setMessage] = useState('');

    // Get the navigate function from the useNavigate hook
    const navigate = useNavigate();

    // Check if the user is already authenticated by checking for a
token in local storage
    const isAuthenticated = localStorage.getItem("token");

    // Use the useEffect hook to perform a side effect when the
component mounts
    useEffect(() => {
        // If the user is already authenticated, navigate to the
feature-one page
        if (isAuthenticated) {
            return navigate("/feature-one");
        }
        // If the user is not authenticated, navigate to the login page
        else {
            return navigate("/login");
        }
```

```
    }, [isAuthenticated, navigate]); // The dependency array includes
isAuthenticated and navigate

    // Define a function to handle the login form submission
    const handleLogin = async () => {
        try {
            // Make a POST request to the login endpoint with the
username and password
            const response = await
axios.post('http://localhost:5000/api/auth/login', { username, password
});

            // Store the token, role, and username in local storage
            localStorage.setItem('token', response.data.token);
            localStorage.setItem('role', response.data.user.role);
            localStorage.setItem('username',
response.data.user.username);

            // Set a success message
            setMessage("Logged in successfully");

            // Navigate to the feature-one page
            navigate('/feature-one');
        } catch (error) {
            // Set an error message if the login fails
            setMessage(error.response?.data?.message || "Error logging
in");
        }
    };

    // Render the component
    return (
        <div className="login">
            <div className="login-inner">
                <div className="login-header">Login</div>
                <div className="login-form">
                    {/* Input field for the username */}
                    <input type="text" placeholder="Username"
value={username} onChange={(e) => setUsername(e.target.value)} />

                    {/* Input field for the password */}
                    <input type="password" placeholder="Password"
value={password} onChange={(e) => setPassword(e.target.value)} />
```

```
                    {/* Login button */}
                    <button className='btn'
onClick={handleLogin}>Login</button>
                </div>
            </div>
            {/* Display the message */}
            {message && <p>{message}</p>}
        </div>
    );
};

// Export the Login component as the default export
export default Login;
```

//Navbar.js

```
// Import the React library
import React from 'react';

// Import the navbar.css file, which is used to style the component
import '../CssFiles/navbar.css';

// Import the Link and useNavigate components from the react-router-dom
library
import { Link, useNavigate } from "react-router-dom";

// Import the useEffect hook from the React library
import { useEffect } from 'react';

// Define a functional component called Navbar
const Navbar = () => {
    // Get the token, role, and username from local storage
    const token = localStorage.getItem('token');
    const role = localStorage.getItem('role');
    const username = localStorage.getItem('username');

    // Get the navigate function from the useNavigate hook
    const navigate = useNavigate();

    // Use the useEffect hook to perform a side effect when the
component mounts
    useEffect(() => {
        // If there is no token in local storage, navigate to the login
page
        if (!token) {
            navigate('/login');
        }
    }, [token, navigate]); // The dependency array includes token and
navigate

    // Define a function to handle the logout button click
    const handleLogout = () => {
        // Remove the token, role, and username from local storage
        localStorage.removeItem('token');
```

```
        localStorage.removeItem('role');
        localStorage.removeItem('username');

        // Navigate to the login page
        navigate('/login');
    }

    // Render the component
    return (
        <>
            <div className='nav'>
                <div className="logo">
                    {/* Display the username if it exists, otherwise
display "Logo" */}
                    {username ? <h1>{username}</h1> : <h1>Logo</h1>}
                </div>
                <div className="lists">
                    <ul className="list">
                        {/* If the user is an admin, display the admin
feature link and the feature-one link */}
                        {role === "admin" ? <><Link
to="/admin-feature-one"><li>Admin</li></Link> <Link
to="/feature-one"><li>Feature-one</li></Link> </> : null}

                        {/* If the user is a user, display the
feature-one link */}
                        {role === "user" ? <Link
to="/feature-one"><li>Feature-one</li></Link> : null}

                        {/* If the user is logged in, display the
logout link */}
                        {token ? <Link
onClick={handleLogout}><li>Logout</li></Link> :
                        /* If the user is not logged in, display the
register and login links */
                        <><Link
to="/register"><li>Register</li></Link><Link
to={"/login"}><li>Login</li></Link></>}
                    </ul>
                </div>
            </div>
        </>
    );
```

```
};

// Export the Navbar component as the default export
export default Navbar;
```

//FeatureOne.js

```
// Import the React library and its useState and useEffect hooks
import React, { useState, useEffect } from 'react';

// Import the axios library, which is used to make HTTP requests
import axios from 'axios';

// Import the featureone.css file, which is used to style the component
import "../CssFiles/featureone.css";

// Define a functional component called FeatureOne
const FeatureOne = () => {
    // Initialize a state variable called 'message' with an initial
value of an empty string
    const [message, setMessage] = useState('');

    // Use the useEffect hook to perform a side effect when the
component mounts
    useEffect(() => {
        // Get the value of the 'token' key from the localStorage
        const token = localStorage.getItem('token');

        // Make a GET request to the
'http://localhost:5000/api/features/feature-one' endpoint
        axios.get('http://localhost:5000/api/features/feature-one', {
            // Include the Authorization header with the token in the
request
            headers: { Authorization: `Bearer ${token}` }
        }).then((res) => {
            // If the request is successful, update the 'message' state
with the response data
            setMessage(res.data.message);
        }).catch((err) => {
            // If the request fails, update the 'message' state with
the error message
            setMessage(err.response?.data?.message || "Error");
        });
    }, []); // The empty dependency array means the effect will only
run once when the component mounts
```

```
    // Render the component
    return (
        <>
            <div className="feature-one">
                <h2>Feature One</h2>
                <p>This is feature one.</p>
                <p>This Access by both User and Admin</p>
                {message && <p>{message}</p>}
            </div>
        </>
    );
};

// Export the FeatureOne component as the default export
export default FeatureOne;
```

//Adminfeatureone.js

```
// Import the React and React Router DOM libraries
import React, { useState, useEffect } from 'react';

// Import the axios library, which is used to make HTTP requests
import axios from 'axios';

// Import the useNavigate function from the react-router-dom library
import { useNavigate } from 'react-router-dom';

// Import the adminfeature.css file
import '../CssFiles/adminfeature.css';

// Define a functional component called AdminFeatureOne
const AdminFeatureOne = () => {
    // Get the value of the 'role' key from the localStorage
    const t = localStorage.getItem('role');

    // Initialize a state variable called 'message' with an initial
value of an empty string
    const [message, setMessage] = useState('');

    // Get the navigate function from the useNavigate hook
    const navigate = useNavigate();

    // Use the useEffect hook to perform a side effect
    useEffect(() => {
        // Get the value of the 'token' key from the localStorage
        const token = localStorage.getItem('token');

        // Make a GET request to the
'http://localhost:5000/api/features/admin-feature-one' endpoint

axios.get('http://localhost:5000/api/features/admin-feature-one', {
            // Include the Authorization header with the token in the
request
            headers: { Authorization: `Bearer ${token}` }
        }).then((res) => {
            // If the request is successful, update the 'message' state
with the response data
```

```
            setMessage(res.data.message);

            // If the 'role' is not 'admin', navigate to the
'/admin-feature-one' route
            if (t !== 'admin') {
                navigate('/admin-feature-one');
            }
        }).catch((err) => {
            // If the request fails, update the 'message' state with
the error message
            setMessage(err.response?.data?.message || "Access Denied");
        });
    }, [t, navigate]);

    // Render the component
    return (
        <>
            <div className="adminfeature">
                <h2>Admin Feature One</h2>
                {message && <p>{message}</p>}
            </div>
        </>
    );
};

// Export the AdminFeatureOne component as the default export
export default AdminFeatureOne;
```

// packege.json of frontend file

```
{
  "name": "author",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.7.7",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.26.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://localhost:5000"
```

```
}
```