**Client-Side: Send Username and Password**

* The client sends a POST request with the username and password.



**Server-Side: Generate JWT Token**

* The server receives the credentials and validates them against the database.
* If the credentials are valid, the server generates a JWT token.

A computer screen with white text

Description automatically generated

**Server-Side: Decode JWT Token**

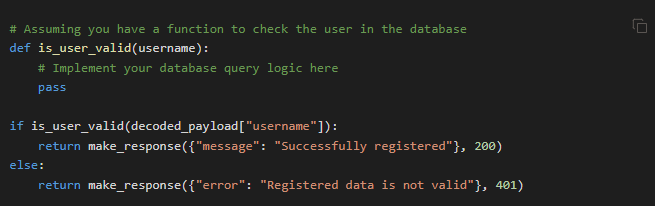
* The client includes the JWT token in the **Authorization** header for subsequent requests.
* The server decodes the JWT token to extract the payload.

A computer screen shot of a program code

Description automatically generated

**Server-Side: Validate User Data**

* The server checks if the user data extracted from the JWT token is present in the database.
* If the user is valid, the server responds with a success message.
* If the user is not valid, the server responds with an error message.



FrontEnd

1.**Create the Login Component**

import React, { useState } from 'react';

import axios from 'axios';

const Login = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const [error, setError] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

try {

const response = await axios.post('http://your-server-url/login', {

username,

password

});

const { token } = response.data;

localStorage.setItem('token', token);

// Redirect to a protected route or show a success message

console.log('Login successful');

} catch (error) {

setError('Invalid username or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleSubmit}>

<div>

<label>Username:</label>

<input

type="text"

value={username}

onChange={(e) => setUsername(e.target.value)}

required

/>

</div>

<div>

<label>Password:</label>

<input

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

required

/>

</div>

{error && <p style={{ color: 'red' }}>{error}</p>}

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

</form>

</div>

);

};

export default Login;

**2.Create a Protected Route Component**

This component will check if the user is authenticated by verifying the JWT token in local storage.

import React, { useEffect, useState } from 'react';

import axios from 'axios';

const ProtectedRoute = () => {

const [data, setData] = useState(null);

const [error, setError] = useState('');

useEffect(() => {

const fetchData = async () => {

const token = localStorage.getItem('token');

if (!token) {

setError('Not authenticated');

return;

}

try {

const response = await axios.get('http://your-server-url/protected-route', {

headers: {

Authorization: `Bearer ${token}`

}

});

setData(response.data);

} catch (error) {

setError('Failed to fetch data');

}

};

fetchData();

}, []);

if (error) {

return <p>{error}</p>;

}

if (!data) {

return <p>Loading...</p>;

}

return (

<div>

<h2>Protected Data</h2>

<pre>{JSON.stringify(data, null, 2)}</pre>

</div>

);

};

export default ProtectedRoute;

**App Component**

Combine the login and protected route components in your main App component.

import React from 'react';

import { BrowserRouter as Router, Route, Switch, Redirect } from 'react-router-dom';

import Login from './Login';

import ProtectedRoute from './ProtectedRoute';

const App = () => {

return (

<Router>

<Switch>

<Route path="/login" component={Login} />

<Route path="/protected" component={ProtectedRoute} />

<Redirect from="/" to="/login" />

</Switch>

</Router>

);

};

export default App;

BackEnd

from flask import Flask, request, jsonify, make\_response

import jwt

import datetime

app = Flask(\_\_name\_\_)

# Mock database

users = {

"user1": "password1",

"user2": "password2"

}

# Secret key for JWT

SECRET\_KEY = "your\_secret\_key"

# Function to validate user credentials

def validate\_user(username, password):

if username in users and users[username] == password:

return True

return False

# Function to check if user is valid

def is\_user\_valid(username):

return username in users

# Login endpoint

@app.route('/login', methods=['POST'])

def login():

userdata = request.get\_json()

username = userdata.get('username')

password = userdata.get('password')

if validate\_user(username, password):

payload = {

"username": username,

"exp": datetime.datetime.utcnow() + datetime.timedelta(hours=1) # Token expiration time

}

jwt\_token = jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

return make\_response(jsonify({"token": jwt\_token}), 200)

else:

return make\_response(jsonify({"error": "Invalid username or password"}), 401)

# Protected route endpoint

@app.route('/protected', methods=['GET'])

def protected():

auth\_header = request.headers.get('Authorization')

if auth\_header:

token = auth\_header.split()[1]

try:

decoded\_payload = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

if is\_user\_valid(decoded\_payload["username"]):

return make\_response(jsonify({"message": "Successfully accessed protected route"}), 200)

else:

return make\_response(jsonify({"error": "Registered data is not valid"}), 401)

except jwt.ExpiredSignatureError:

return make\_response(jsonify({"error": "Token has expired"}), 401)

except jwt.InvalidTokenError:

return make\_response(jsonify({"error": "Invalid token"}), 401)

else:

return make\_response(jsonify({"error": "Authorization header missing"}), 401)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)