**COURSE CODE: DJS22ITL604 DATE:27-02-25**

**COURSE NAME: Full Stack Web Development Laboratory CLASS: TYBTech**

**Name: Anish Sharma Div:IT-1-1**

# EXPERIMENT NO. 05

**CO/LO:** CO1-Develop a full stack web application.

**AIM / OBJECTIVE: Authentication and Authorization with JWT: Implement user authentication using JWT tokens for a simple login/signup functionality.**

**THEORY**:

**Introduction to MongoDB**

Authentication and Authorization are key components in web applications to ensure secure user access.

Authentication verifies user identity (e.g., login/signup).

Authorization grants or restricts access based on user roles.

**JWT**

JWT (JSON Web Token) is a secure token format used to transmit data between parties. It is widely used for authentication and authorization in web applications.

A JWT consists of three parts, separated by dots (.):

***Header.Payload.Signature***

**Header:** Contains metadata, including the token type (JWT) and the signing algorithm (e.g., HS256 for HMAC SHA-256).

**Payload:** Contains claims (user information and other data).

**Signature:** Ensures data integrity. It is generated using a secret key.

JWT (JSON Web Token) is a compact, self-contained token used for secure communication between the client and server.

**JWT in Authentication**

1. User logs in with credentials (email & password).
2. Server verifies credentials and generates a JWT.
3. JWT is sent to the client and stored (e.g., in local storage or HTTP-only cookies).
4. Client includes the JWT in each request’s Authorization header (Bearer <token>).
5. Server verifies the JWT, extracts user details, and grants access.

**COURSE CODE: DJS22ITL604 DATE:**

**COURSE NAME: Full Stack Web Development Laboratory CLASS: TYBTech**

**JWT in Authorization**

1. Each request includes the JWT.
2. The server decodes the JWT and checks:
3. Is the token valid? (Signature verification)
4. Has it expired? (Expiration check)
5. Does the user have permission? (Role-based access control)
6. Based on this, access is granted or denied.

Express.js is used for handling server-side logic, while React provides the front-end interface. bcrypt is used for password hashing, and jsonwebtoken for generating JWT tokens.

**PROCEDURE**

**Step 1: Set Up the Project** mkdir jwt-auth-app && cd jwt-auth-app npm init -y 1.2 Install dependencies:

npm install express bcryptjs jsonwebtoken cors dotenv mongoose body-parser 1.3 Install development dependencies:

npm install nodemon --save-dev

**Step 2: Setting Up the Express.js Server** 2.1 Create a file server.js and set up Express: const express = require('express'); const cors = require('cors'); const mongoose = require('mongoose');

const dotenv = require('dotenv');

const authRoutes = require('./routes/authRoutes');

dotenv.config();

|  |  |
| --- | --- |
| **COURSE CODE: DJS22ITL604** | **DATE:** |
| **COURSE NAME: Full Stack Web Development Laboratory** | **CLASS: TYBTech** |

const app = express();

app.use(cors()); app.use(express.json()); app.use('/api/auth', authRoutes);

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true,

}).then(() => console.log('MongoDB connected'))

.catch(err => console.log(err));

app.listen(5000, () => console.log('Server running on port 5000'));

**Step 3: Creating the User Model** Create a models/User.js file: const mongoose = require('mongoose'); const

UserSchema = new mongoose.Schema({

username: { type: String, required: true, unique: true }, email: { type: String, required: true, unique: true }, password: { type: String, required: true }

});

module.exports = mongoose.model('User', UserSchema);

**Step 4: Implementing Authentication Routes** Create routes/authRoutes.js: const express = require('express');

|  |  |
| --- | --- |
| **COURSE CODE: DJS22ITL604** | **DATE:** |
| **COURSE NAME: Full Stack Web Development Laboratory** | **CLASS: TYBTech** |

const bcrypt = require('bcryptjs'); const jwt = require('jsonwebtoken');

const User =

require('../models/User'); const router

= express.Router();

const SECRET\_KEY = process.env.JWT\_SECRET;

// Signup Route router.post('/signup', async (req, res) => { try { const { username, email, password } = req.body; const hashedPassword = await bcrypt.hash(password, 10); const user = new User({ username, email, password: hashedPassword }); await

user.save();

res.status(201).json({ message: 'User registered successfully' });

} catch (error) { res.status(500).json({ error: 'Error registering user' }); }

});

// Login Route router.post('/login', async (req, res) => { try { const { email, password } = req.body; const user = await User.findOne({ email }); if (!user || !(await bcrypt.compare(password, user.password))) { return res.status(400).json({ error: 'Invalid credentials' });

|  |  |
| --- | --- |
| **COURSE CODE: DJS22ITL604** | **DATE:** |
| **COURSE NAME: Full Stack Web Development Laboratory** | **CLASS: TYBTech** |

} const token = jwt.sign({ id: user.\_id }, SECRET\_KEY, { expiresIn: '1h' }); res.json({

token });

} catch (error) { res.status(500).json({ error: 'Error logging in' }); }

});

module.exports = router;

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Step 5: Implementing Authorization Middleware** Create middleware/authMiddleware.js: const jwt = require('jsonwebtoken');

const SECRET\_KEY = process.env.JWT\_SECRET;

module.exports = (req, res, next) => { const token = req.header('Authorization'); if (!token) return res.status(401).json({ error: 'Access Denied' }); try { const verified = jwt.verify(token, SECRET\_KEY); req.user

= verified; next();

} catch (error) { res.status(400).json({ error:

'Invalid Token' }); }

};

|  |  |
| --- | --- |
| **COURSE CODE: DJS22ITL604** | **DATE:** |
| **COURSE NAME: Full Stack Web Development Laboratory** | **CLASS: TYBTech** |

**Step 6: Creating a Protected Route** Update routes/authRoutes.js: const authMiddleware =

require('../middleware/authMiddleware'); router.get('/profile', authMiddleware, (req, res) => { res.json({ message: 'Protected data', user: req.user });

});

**Step 7: Setting Up the React Front-End**

7.1 Initialize React App:

npx create-react-app jwt-auth-client cd jwt-auth-client npm install axios react-router-dom **DEPARTMENT OF INFORMATION TECHNOLOGY**

7.2 Create Login.js Component: import { useState } from 'react'; import axios from 'axios';

const Login = () => { const [email, setEmail] = useState(''); const [password, setPassword] = useState(''); const handleSubmit = async (e)

=> {

e.preventDefault(); const res = await axios.post[('http://localhost:5000/api/auth/login',](http://localhost:5000/api/auth/login%27)  [{](http://localhost:5000/api/auth/login%27) email, password }); localStorage.setItem('token', res.data.token);

}; return

(

**COURSE CODE: DJS22ITL604**  **DATE:**

**COURSE NAME: Full Stack Web Development Laboratory**  **CLASS: TYBTech**

<form onSubmit={handleSubmit}>

<input type='email' placeholder='Email' onChange={(e) => setEmail(e.target.value)} />

<input type='password' placeholder='Password' onChange={(e) => setPassword(e.target.value)} />

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

</form>

);

};

export default Login;

7.3 Implement Protected Route in App.js: import { useEffect, useState } from 'react'; import axios from 'axios';

const Profile = () => { const [data, setData] = useState('');

useEffect(() => {

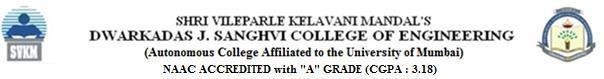
**DEPARTMENT OF INFORMATION TECHNOLOGY**

axios.get[('http://localhost:5000/api/auth/profile', {](http://localhost:5000/api/auth/profile%27) headers: {

Authorization: localStorage.getItem('token') }

}).then(res => setData(res.data.message));

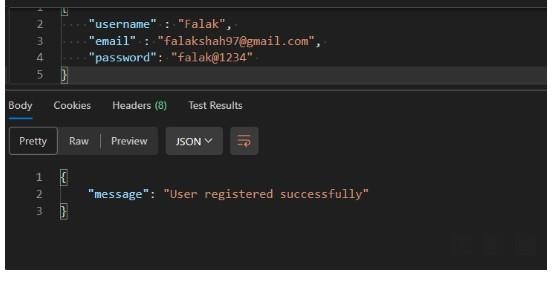
}, []);

return <h1>{data}</h1>;

};

export default Profile; **Observation:**





**BOOKS AND WEB RESOURCES**

1. **"OAuth 2.0 and OpenID Connect: The Definitive Guide"** – by Aaron Parecki
2. **JWT Official Website** [– https://jwt.io/’](https://jwt.io/’)
3. **Auth0 Blog - JWT Authentication Best Practices** [– https://auth0.com/blog](https://auth0.com/blog)

**CONCLUSION**: We implemented user authentication using JWT tokens for a simple login/signup functionality.