SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL604 **DATE:**

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

NAME: Anish Sharma

Roll no: I011

EXPERIMENT NO. 09

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

AIM / OBJECTIVE: Enhance APIs with middleware, validation, and security mechanisms such as rate limiting and CORS.

THEORY:

Introduction to API Security & Middleware

APIs (Application Programming Interfaces) play a crucial role in modern web applications, enabling communication between frontend and backend services. Ensuring API security is essential to protect data integrity, prevent unauthorized access, and maintain application performance. **Key Enhancements for Secure APIs**

1. **Middleware in Express.js**: Middleware functions process requests before they reach the route handlers, commonly used for logging, authentication, validation, and security.
2. **Validation**: Ensuring that incoming data follows predefined formats and constraints, reducing vulnerabilities like SQL injection and XSS attacks.
3. **Rate Limiting**: Restricting the number of API requests from a user within a timeframe to prevent abuse.
4. **CORS (Cross-Origin Resource Sharing)**: Controls access to resources from different origins, preventing unauthorized requests.
5. **Authentication & Authorization**: Securing APIs with JWT (JSON Web Tokens) and OAuth to restrict access.

TECHNOLOGIES/PLATFORMS USED:

- **Backend Framework:** Express.js (Node.js)
- **Security Packages:** Helmet, Express-rate-limit, CORS, JSON Web Token (JWT), Joi (for validation)
- **Database:** MongoDB (with Mongoose for schema validation)
- **Version Control:** Git & GitHub

COURSE CODE: DJS22ITL604 **DATE:**

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

STEP-BY-STEP IMPLEMENTATION:

Step 1: Setting Up Middleware in Express.js

```
const express = require('express');
const helmet = require('helmet');
const cors = require('cors');
const rateLimit = require('express-rate-limit');
const app = express();

// Middleware for security headers
app.use(helmet());

// Middleware for CORS
app.use(cors());

// Rate Limiting const limiter = rateLimit({ windowMs: 15 * 60 *
1000, // 15 minutes max: 100, // Limit each IP to 100 requests per
window message: "Too many requests from this IP, please try again
later."
});
app.use(limiter);
// Middleware for JSON parsing
app.use(express.json());
```

Step 2: Data Validation using Joi

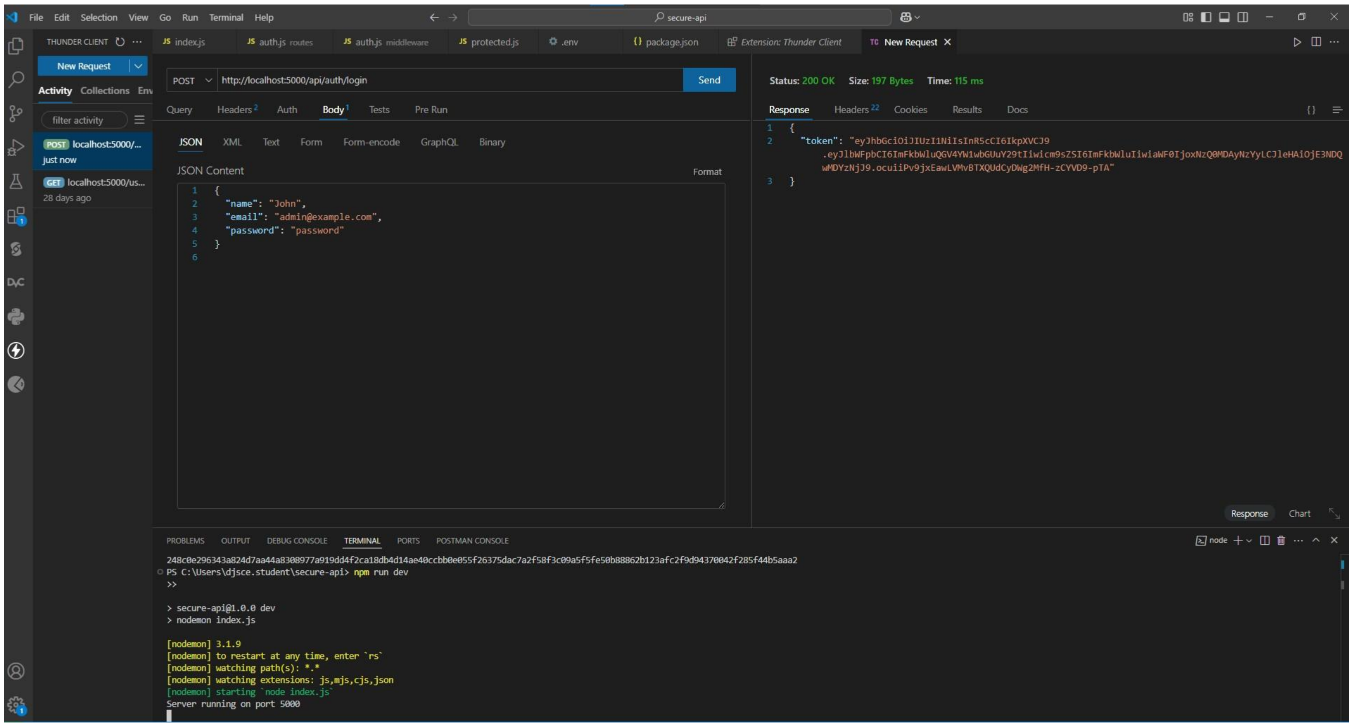
```
const Joi = require('joi');
const userSchema = Joi.object({ name:
  Joi.string().min(3).required(),     email:
  Joi.string().email().required(),
  password: Joi.string().min(6).required()
});


app.post('/register', (req, res) => { const { error }
  = userSchema.validate(req.body);
  if (error) return res.status(400).json({ message: error.details[0].message });
  res.status(201).json({ message: "User registered successfully" });
});
```

Step 3: Implementing JWT Authentication

```
const jwt = require('jsonwebtoken');
const SECRET_KEY = "your_secret_key";


app.post('/login', (req, res) => { const {
  email, password } = req.body;
  if (email !== "test@example.com" || password !== "password") {
     return res.status(401).json({ message: "Invalid credentials" });
  }
  const token = jwt.sign({ email }, SECRET_KEY, { expiresIn: '1h' });
```

```
  res.json({ token });
});


 const authenticateToken = (req, res, next) => {
    const token = req.header('Authorization');
  if (!token) return res.status(401).json({ message: "Access Denied" });


  jwt.verify(token.split(" ")[1], SECRET_KEY, (err, user) => { if (err)
    return res.status(403).json({ message: "Invalid Token" });
    req.user = user; next();
  });
};
```

Step 4: Protecting Routes with JWT Authentication app.get('/dashboard',
authenticateToken, (req, res) => { res.json({ message: "Welcome to the
secure dashboard!", user: req.user });
});



POST http://localhost:5000/api/auth/login

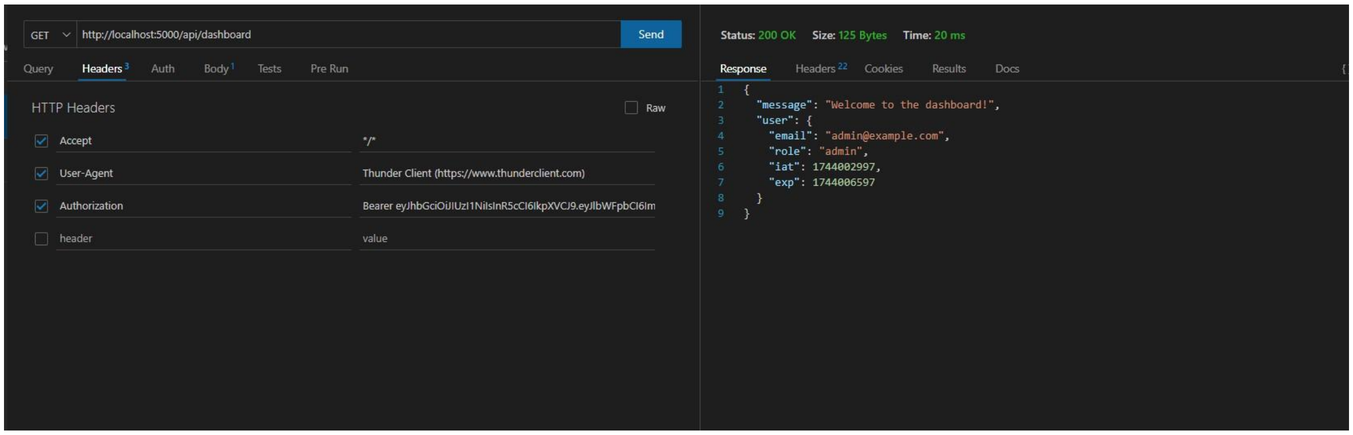
JSON Content

```
{
  "name": "John",
  "email": "admin@example.com",
  "password": "password"
}
```

Status: 200 OK Size: 197 Bytes Time: 115 ms

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
     .eyJlbWFpbCI6ImFkbWluQGV4YW1wbGUuY29tIiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNzQ0MDAyNzYyLCJleHAiOjE3NDQ
     wMDYzNjJ9.ocuiiPv9jxEawLVMvBTXQUdCyDWg2MfH-zCYVD9-pTA"
}
```

```
248c0e296343a824d7aa44a8308977a919dd4f2ca18db4d14ae40ccbb0e055f26375dac7a2f58f3c09a5f5fe50b88862b123afc2f9d94370042f285f44b5aaa2
PS C:\Users\djsce.student\secure-api> npm run dev
>>

> secure-api@1.0.0 dev
> nodemon index.js

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server running on port 5000
```



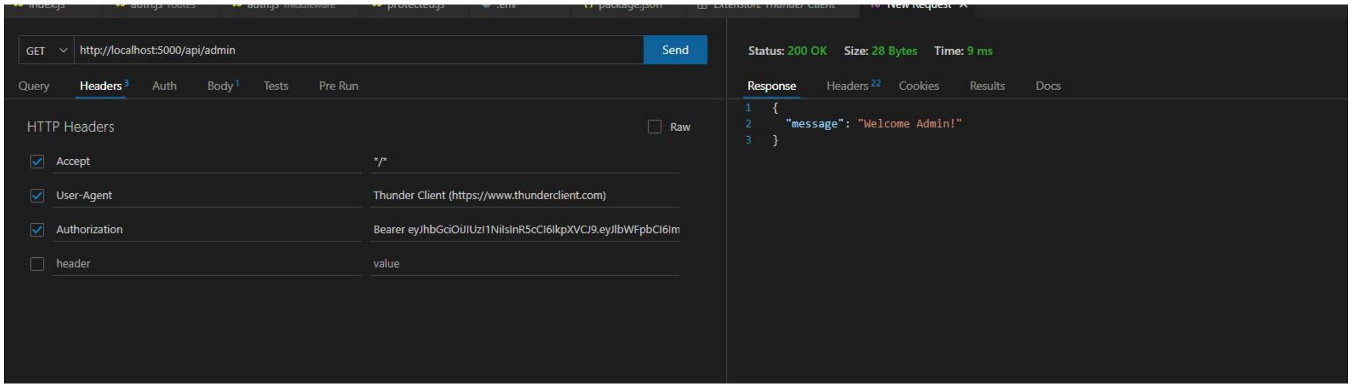
GET http://localhost:5000/api/dashboard

HTTP Headers

✓ Accept	*/*
✓ User-Agent	Thunder Client (https://www.thunderclient.com)
✓ Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im
header	value

Status: 200 OK Size: 125 Bytes Time: 20 ms

```
{
  "message": "Welcome to the dashboard!",
  "user": {
    "email": "admin@example.com",
    "role": "admin",
    "iat": 1744002997,
    "exp": 1744006597
  }
}
```



GET http://localhost:5000/api/admin

HTTP Headers

✓ Accept	*/*
✓ User-Agent	Thunder Client (https://www.thunderclient.com)
✓ Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im
header	value

Status: 200 OK Size: 28 Bytes Time: 9 ms

```
{
  "message": "Welcome Admin!"
}
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL604 **DATE:**

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

BOOKS AND WEB RESOURCES:

1. Express.js Documentation
2. Helmet Security Middleware
3. Express-rate-limit
4. CORS Documentation
5. JSON Web Tokens (JWT)
6. Joi Validation Library

TASK:

1. Implement Role-Based Access Control (RBAC): Restrict access to certain routes based on user roles (admin, user).
2. Protect API from excessive requests and enforce CORS policy.
    a. Implement rate limiting using express-rate-limit.
    b. Restrict API access to only allowed origins using cors.
3. Implement HTTP Security Headers
    a. Prevent security vulnerabilities using HTTP headers.
    b. Install helmet to set security headers:
    c. Apply Content-Security-Policy (CSP), X-XSS-Protection, and X-Frame-Options.

WRITE-UP QUESTIONS:

1. Explain the importance of rate limiting and how it prevents abuse.
2. What is CORS, and how does it impact API accessibility?
3. Why is validation important in API development, and how does Joi h