# Mini Project Report - 016

Master of Computer Application – General
Semester – III

**Sub: Web Technologies**

**Topic: Securing an API Endpoint with JWT**
By
**Name:** SANDRA B
**Reg no.:** 24110222500001

**Faculty Name:** VEERA RAGHAV K

**Faculty Signature:** ________________

**Department of Computer Application**
**Alliance University**
**Chandapura - Anekal Main Road, Anekal**
**Bengaluru - 562 106**

**August 2025**

# Securing an API Endpoint with JWT

## 1. Introduction

In the modern world of software development, security has become one of the most critical aspects of building web applications and APIs. With the rapid growth of digital services, online transactions, and data-driven systems, safeguarding information and ensuring that only authorized users can access sensitive resources is a top priority. Traditional methods of authentication, such as session-based logins or API keys, often fall short in terms of scalability, flexibility, and security. To overcome these limitations, developers widely adopt JSON Web Tokens (JWT), a compact and self-contained mechanism for securely transmitting information between parties as a JSON object.

JWT has become an industry standard for securing API endpoints because of its lightweight structure and ability to handle stateless authentication. Unlike traditional session management, JWT allows authentication to be performed without the need for server-side session storage. This makes it highly suitable for distributed systems, cloud-based services, and microservice architectures. Once a user logs in and receives a signed token, that token can be used to access protected routes or perform certain operations until it expires. This makes JWT-based authentication both efficient and scalable, ensuring that APIs remain secure and only accessible to legitimate users.

The project described in this report demonstrates how to secure an API endpoint using JWT in a Node.js environment with Express.js. The implementation includes creating a simple task management API where users can log in, generate a token, and then use that token to perform CRUD (Create, Read, Delete) operations on tasks. The login route is left open for users to obtain a JWT by providing a username, but all task-related endpoints are protected and require authentication. This ensures that unauthorized users or requests without valid tokens are blocked from accessing the system.

Testing and validation were carried out using Postman, a powerful API testing tool, alongside the browser for open routes. Through this setup, the importance of API authentication becomes clear: while general instructions or welcome messages can be publicly accessible, any sensitive or private data manipulation must be safeguarded with strong authentication mechanisms. By integrating JWT, this project highlights a practical and secure approach to endpoint protection, showcasing how even small applications can adopt industry-standard security practices.

---

## 2. Objective

The goal of this project is to secure an API endpoint using **JSON Web Tokens (JWT)**. JWT ensures that only authenticated users with a valid token can access certain API routes.

---

## 3. Tools & Technologies

**Node.js** (Runtime Environment)

**Express.js** (Framework for API)

**jsonwebtoken** (Library to generate and verify JWT)

**Postman** (API testing tool)

**VS Code** (Code editor)

---

## 4. Code Explanation

### Step 1: Install Dependencies

Inside the project folder, initialize Node.js and install required packages:

```
npm init -y
npm install express jsonwebtoken
```

### Step 2: Server Code (server.js)

The following code was written in VS Code:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();
const PORT = 3000;

const JWT_SECRET = 'mysecretkey';  // Secret key for signing JWT

app.use(express.json());

// In-memory task list
let tasks = [];
let nextId = 1;

// Middleware to verify JWT
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) {
        return res.status(401).json({ error: 'Access denied. Token missing!' });
    }
    jwt.verify(token, JWT_SECRET, (err, user) => {
        if (err) {
            return res.status(403).json({ error: 'Invalid or expired token!' });
        }
        req.user = user;
        next();
    });
}

// Root route
app.get('/', (req, res) => {
    res.send('Welcome! Use /login to get a token and then /tasks.');
});

// Login Route (Generate JWT)
app.post('/login', (req, res) => {
    const { username } = req.body;
    if (!username) {
        return res.status(400).json({ error: 'Username required' });
    }
    const user = { name: username };
    const token = jwt.sign(user, JWT_SECRET, { expiresIn: '1h' });
    res.json({ token });
});

// Protected Routes
```

```
app.post('/tasks', authenticateToken, (req, res) => {
  const { title } = req.body;
  if (!title) {
    return res.status(400).json({ error: 'Task title is required' });
  }
  const newTask = { id: nextId++, title };
  tasks.push(newTask);
  res.status(201).json(newTask);
});

app.get('/tasks', authenticateToken, (req, res) => {
  res.json(tasks);
});

app.delete('/tasks/:id', authenticateToken, (req, res) => {
  const taskId = parseInt(req.params.id);
  const index = tasks.findIndex(task => task.id === taskId);
  if (index === -1) {
    return res.status(404).json({ error: 'Task not found' });
  }
  const deletedTask = tasks.splice(index, 1)[0];
  res.json(deletedTask);
});

// Start server
app.listen(PORT, () => {
  console.log(`Task API running at http://localhost:${PORT}`);
});
```

---

## 5. Learning Outcomes

By working on this project, the following learning outcomes were achieved:

1. **Understanding API Security**
    - Gained knowledge of why APIs need protection and how unauthorized access can lead to security vulnerabilities.

2. **Practical Implementation of JWT**
    - Learned how to generate, sign, and verify JSON Web Tokens for authentication.
    - Understood the concept of token expiration and the importance of secret keys.

3. **Middleware in Express.js**
    - Applied middleware functions to intercept and validate requests before they reach protected routes.
    - Improved understanding of request–response cycles in Node.js applications.

4. **RESTful API Development**
    - Designed and tested API routes (/login, /tasks) with CRUD functionality.
    - Understood the difference between public (open) and private (protected) endpoints.

5. **Testing with Postman**
    - Gained hands-on experience in testing APIs with tokens.
    - Learned how to set headers (Authorization: Bearer <token>) for protected routes.

6. **Error Handling & Response Management**
    - Implemented meaningful error messages for missing tokens, expired tokens, and invalid requests.

7. **Scalability Concepts**
    - Understood how JWT enables stateless authentication, making APIs scalable for distributed applications.

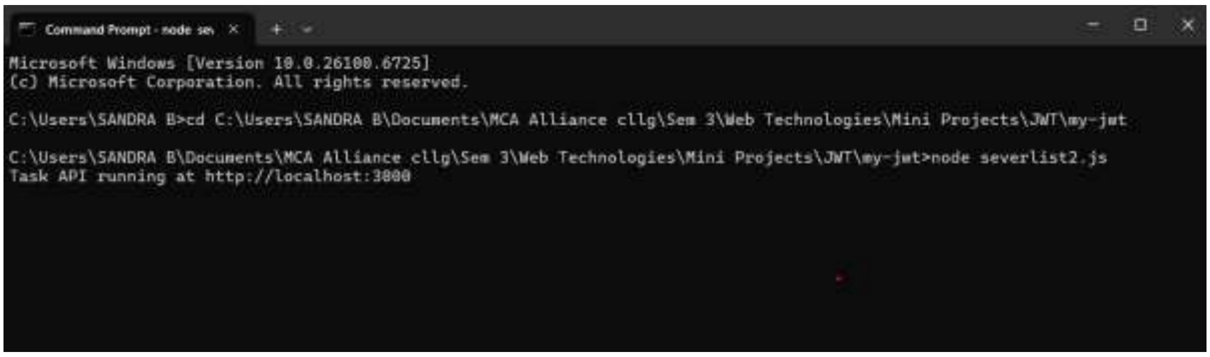
## 6. Output

### A. Run the Server

node server.js

**Output in terminal:**

Task API running at http://localhost:3000



```
Microsoft Windows [Version 10.0.26100.6725]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SANDRA B>cd C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\JWT\my-jwt

C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\JWT\my-jwt>node severlist2.js
Task API running at http://localhost:3000
```

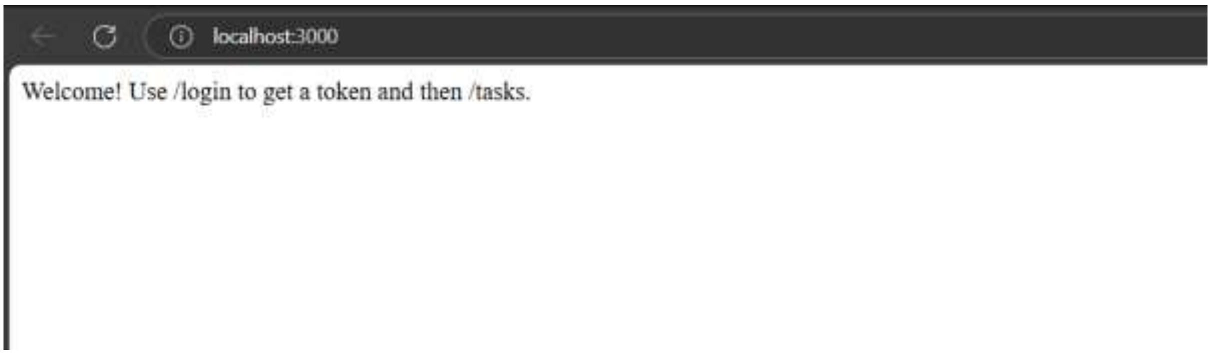
---

### B. Open Browser (Unprotected Route)

**Open:**
http://localhost:3000/

Output:

Welcome! Use /login to get a token and then /tasks.



Welcome! Use /login to get a token and then /tasks.

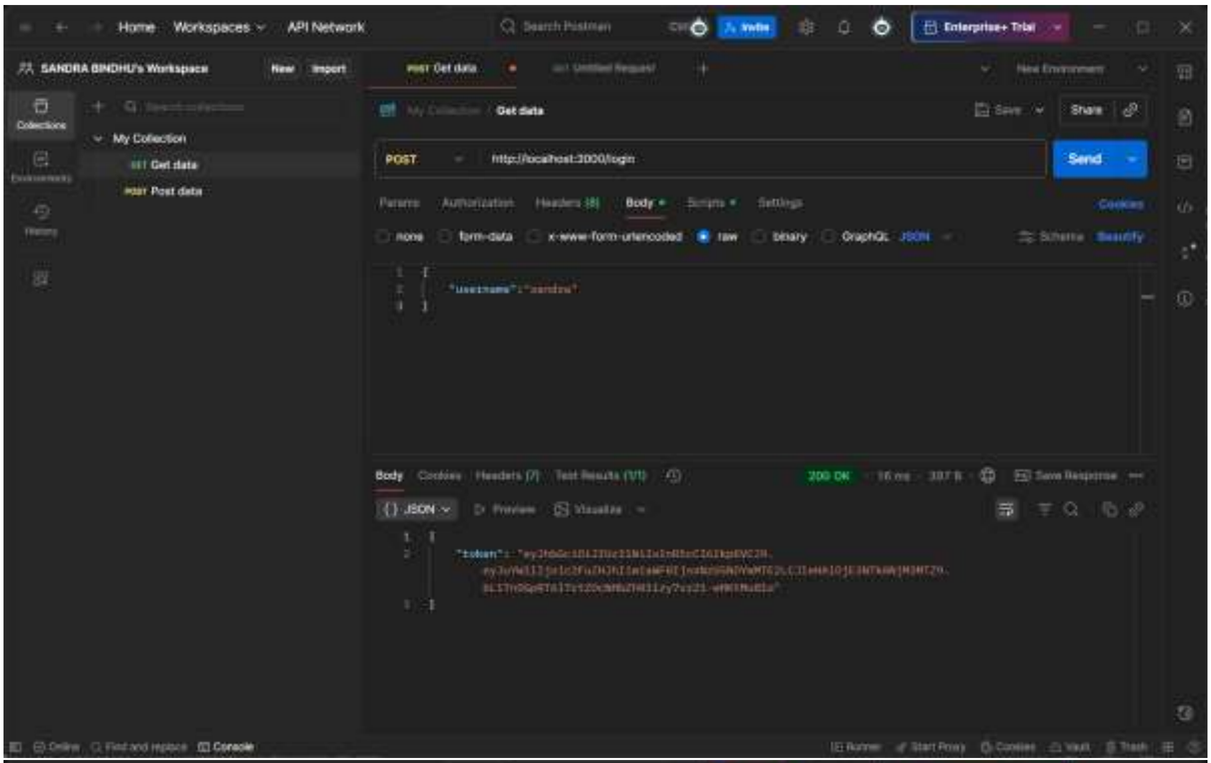
---

### C. Test in Postman

1. Get Token

- Method: POST
- URL: http://localhost:3000/login
- Body (raw JSON):

```
{
  "username": "Raghav"
}
Response:
{
  "token": "eyJhbGciOiJIUzI1NiIsInR..."
}
```



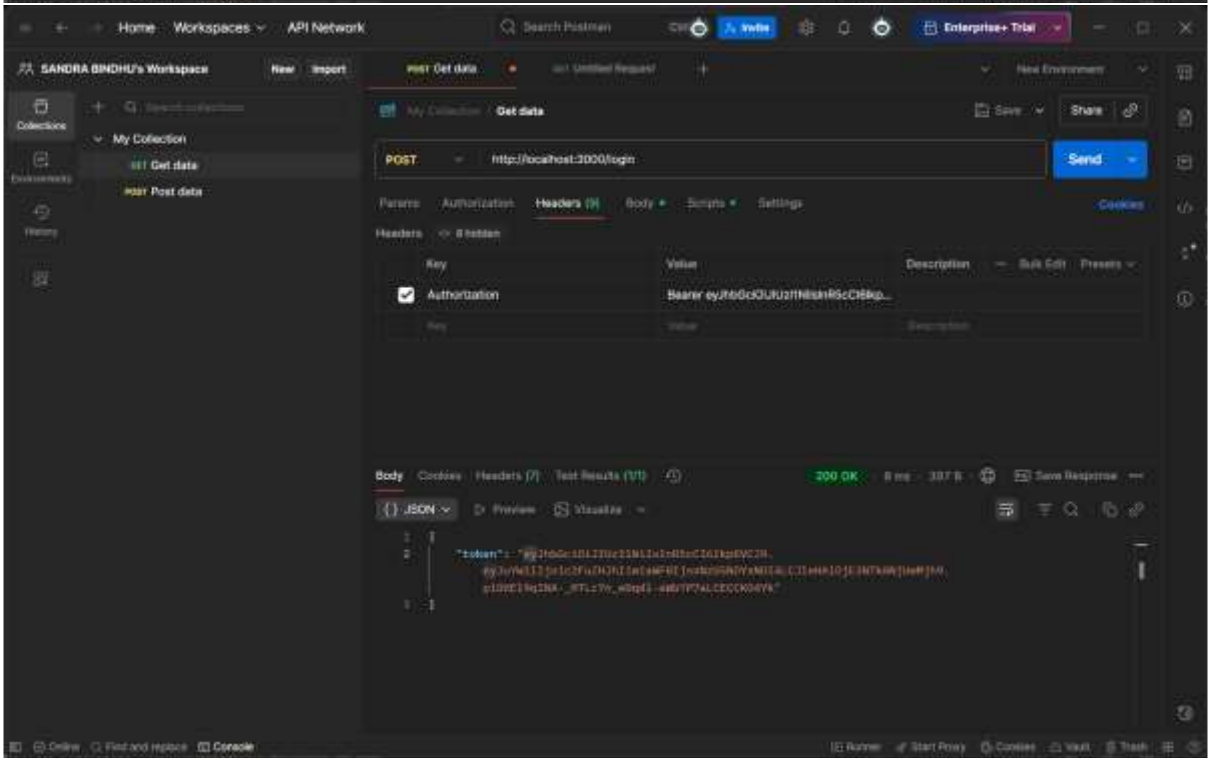
POST http://localhost:3000/login

Body: raw JSON

```
{
  "username":"sandra"
}
```

Response: 200 OK

```
{
  "token": "[illegible]"
}
```



POST http://localhost:3000/login

Headers

| Key | Value | Description |
|---|---|---|
| ✓ Authorization | Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp... | |

Response: 200 OK

```
{
  "token": "[illegible]"
}
```

**7. Access Protected Routes**

**A. Add Task**

- Method: POST
- URL: http://localhost:3000/tasks
- Header:

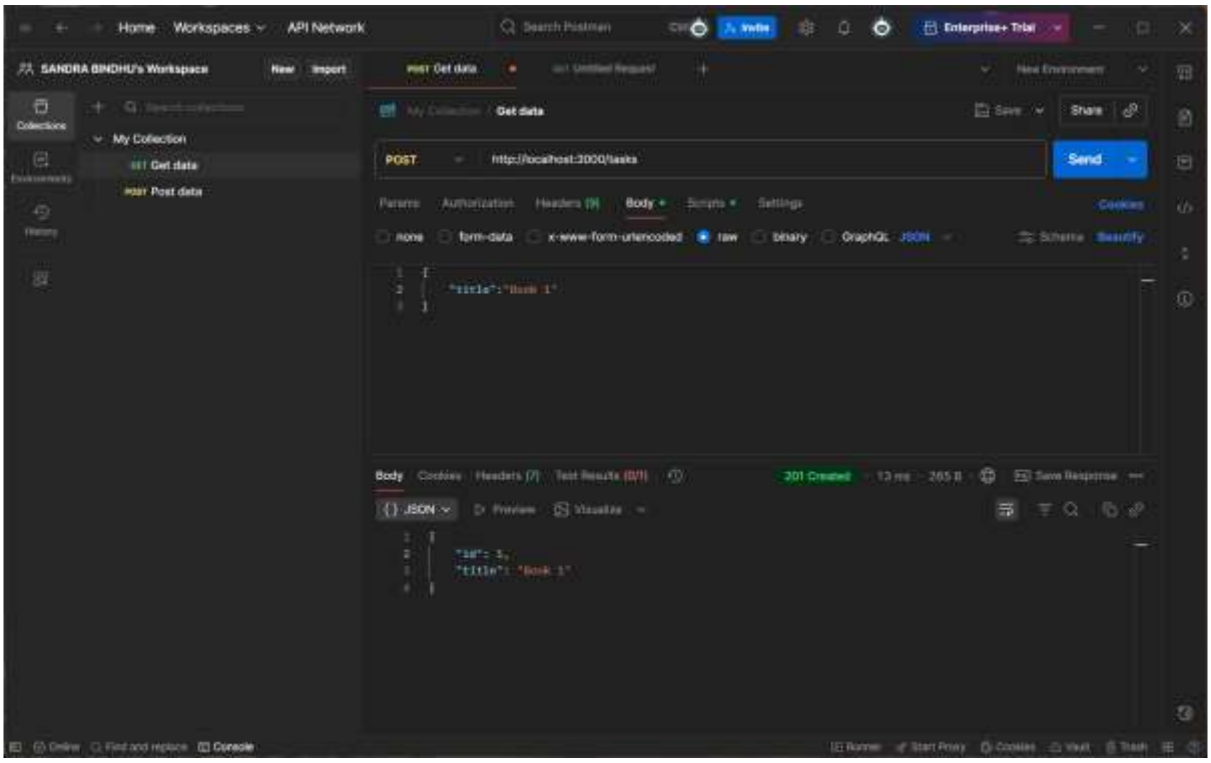
Authorization: Bearer <your_token_here>

- Body (raw JSON):

```
{
  "title": "Learn JWT"
}
```

Response:

```
{
  "id": 1,
  "title": "Learn JWT"
}
```



POST http://localhost:3000/tasks

Body: raw JSON

```
{
  "title":"Book 1"
}
```

201 Created · 13 ms · 265 B

```
{
    "id": 1,
    "title": "Book 1"
}
```

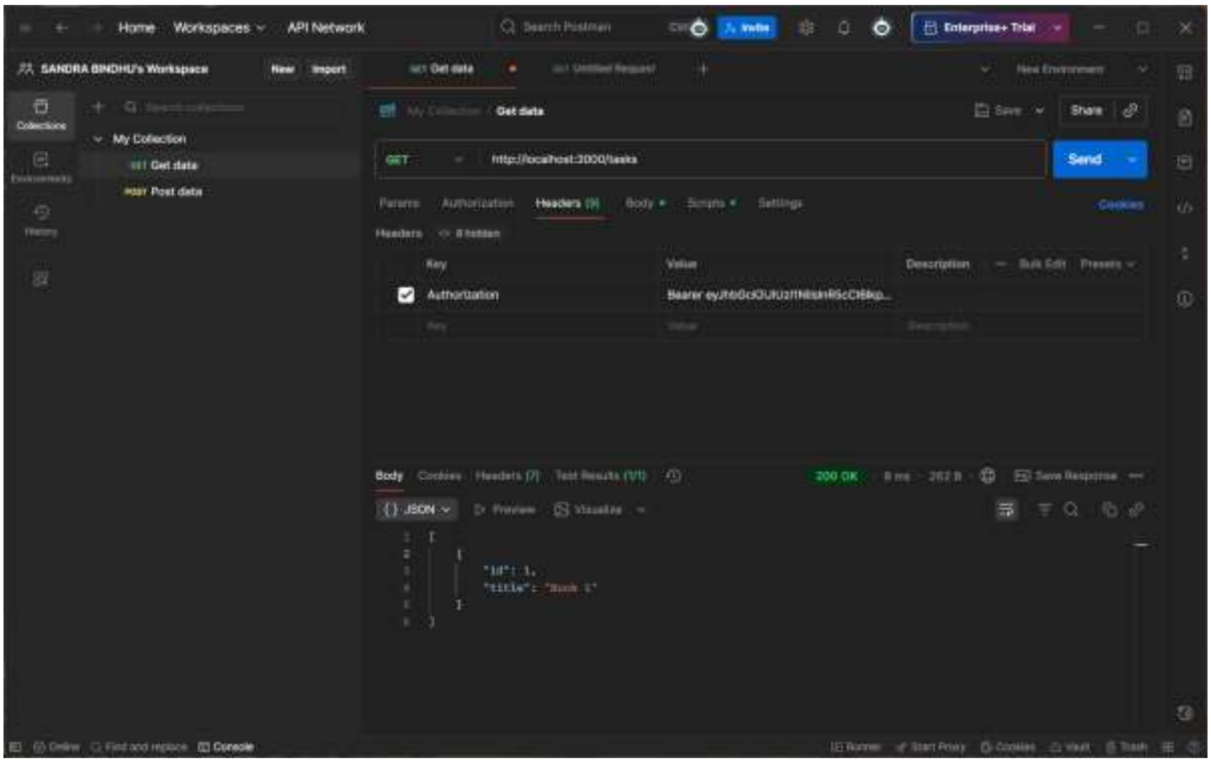
### B. Get Tasks

- Method: GET
- URL: http://localhost:3000/tasks
- Header:

Authorization: Bearer <your_token_here>

Response:

```
[
  { "id": 1, "title": "Learn JWT" }
]
```



GET http://localhost:3000/tasks

| Key | Value | Description |
|---|---|---|
| Authorization | Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp... | |

200 OK · 8 ms · 262 B

```
[
    {
        "id": 1,
        "title": "Book 1"
    }
]
```

---

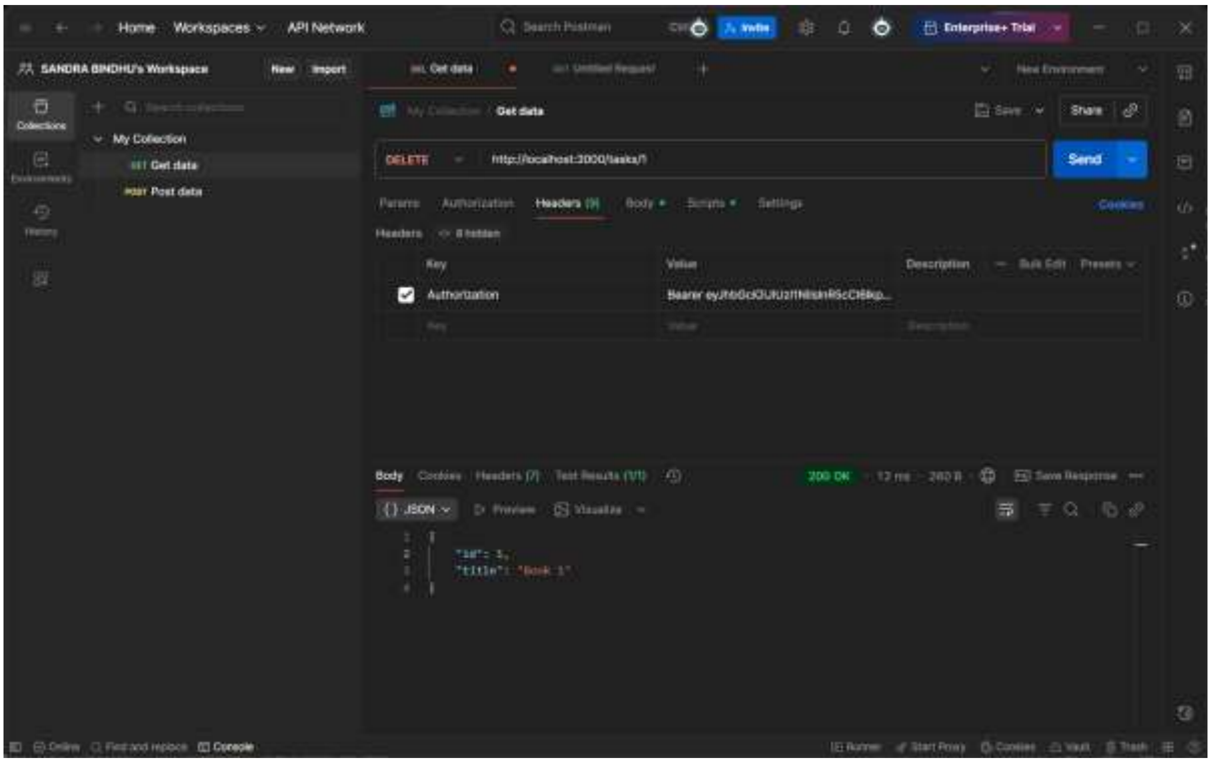
### C. Delete Task

- Method: DELETE
- URL: http://localhost:3000/tasks/1
- Header:

Authorization: Bearer <your_token_here>

Response:

```
{ "id": 1, "title": "Learn JWT" }
```



```
DELETE  http://localhost:3000/tasks/1

Headers (9)
Key            Value
Authorization  Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...

200 OK  12 ms  260 B

{
    "id": 1,
    "title": "Book 1"
}
```

---

## 8. Conclusion

- The API was successfully secured using JWT.
- Open route (/) works in browser without authentication.
- Protected routes (/tasks) require a valid JWT Token and can be tested in Postman.
- This setup demonstrates API authentication best practices.

---