





**UNIVERSITI TEKNIKAL MALAYSIA MELAKA**  
**FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER**

**BERR 2243**  
**DATABASE & CLOUD SYSTEM**

**SEM 2 2024/2025**

**EXERCISE : WEEK 6**

No	Name	Matrix No	Photo
1	NURUL ANIS HAFIFZA BINTI AMRAN	B122410321	
2	NUR AIN HIDAYAH BINTI ABDUL RAHIM	B122410323	

## Submission Requirements

**GitHub Repository:** Code for JWT authentication, password hashing, and RBAC middleware.

githublink :

- <https://github.com/b122410323/berr2243-25> (b122410323)
- <https://github.com/anishafifza/berr2243-25> (b122410321)

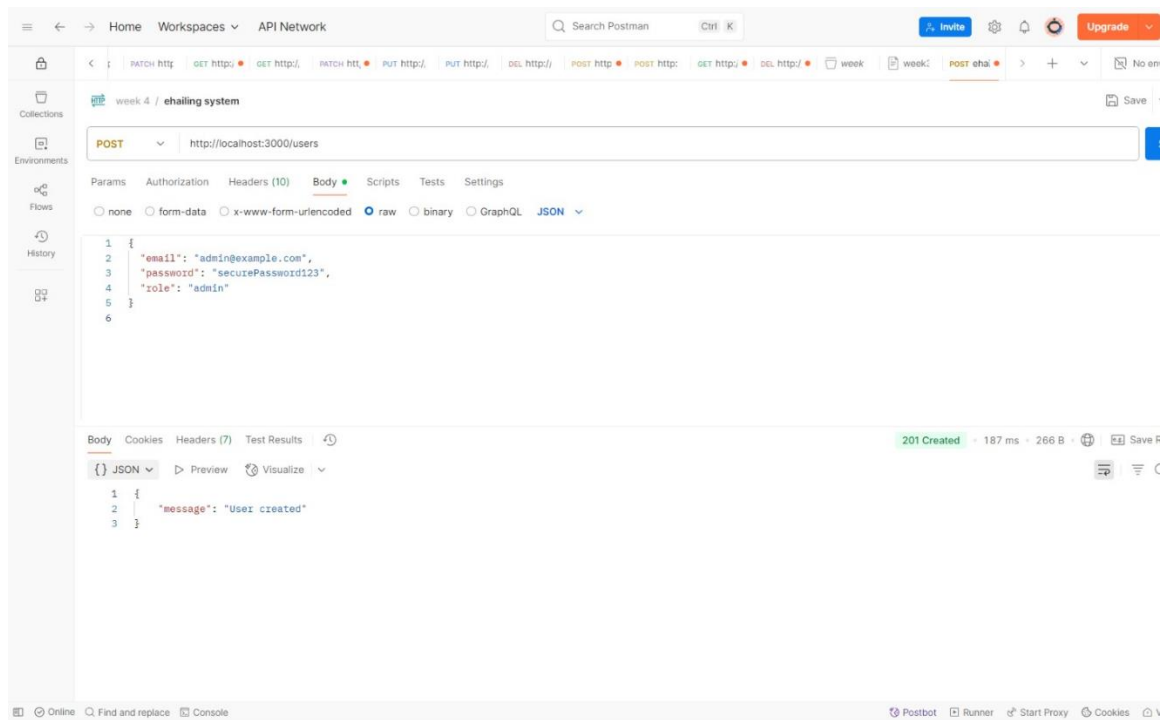
```
Code Blame 142 lines (125 loc) · 4.73 KB Code 55% faster with GitHub Copilot

1  require('dotenv').config();
2  const express = require('express');
3  const { MongoClient, ObjectId } = require('mongodb');
4  const bcrypt = require('bcrypt');
5  const jwt = require('jsonwebtoken');
6  const cors = require('cors');
7
8  const app = express();
9  const port = process.env.PORT || 3000;
10 const saltRounds = 10;
11
12 app.use(express.json());
13 app.use(cors());
14
15 let db;
16
17 // MongoDB Connection
18 async function connectToMongoDB() {
19   const client = new MongoClient(process.env.MONGODB_URI);
20   try {
21     await client.connect();
22     db = client.db();
23     console.log("Connected to MongoDB!");
24
25     // Insert default admin if not exists
26     const existingAdmin = await db.collection('users').findOne({ role: "admin" });
27     if (!existingAdmin) {
28       const hashedPassword = await bcrypt.hash("admin123", saltRounds);
29       await db.collection('users').insertOne({
30         name: "Admin",
31         email: "admin@example.com",
32         password: hashedPassword,
33         role: "admin",
34         status: "active"
35       });
36     }
37   } catch (error) {
38     console.error("Error connecting to MongoDB:", error);
39   }
40 }
```

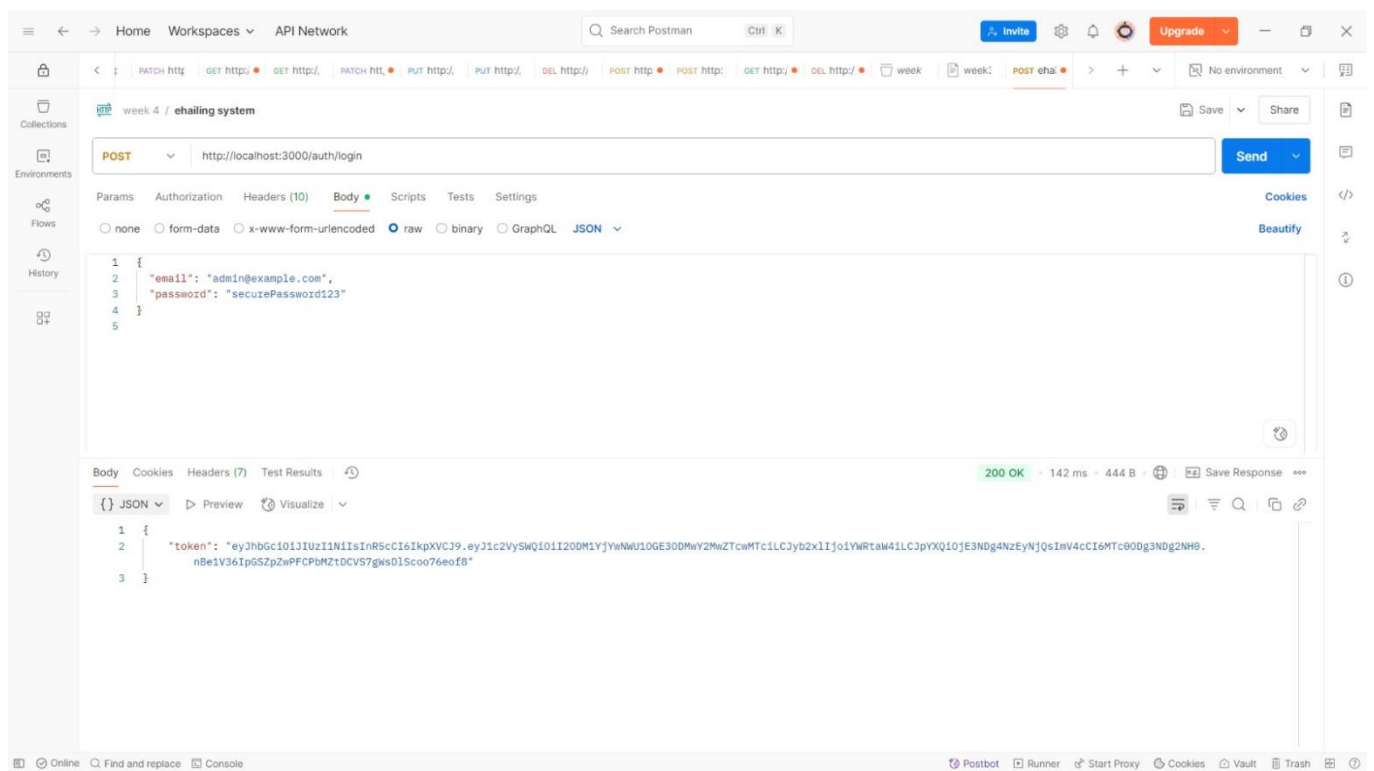
## 2. Postman Collection:

o Exported collection with:

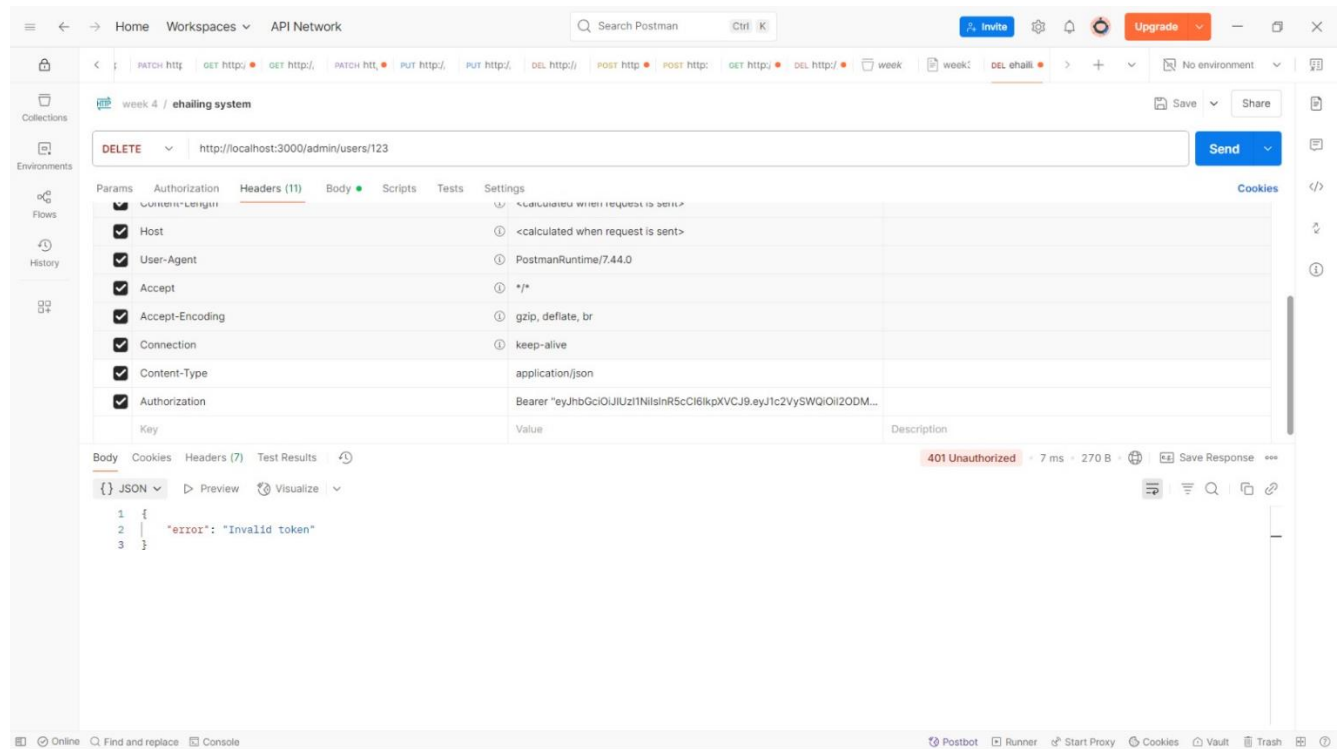
§ Registration request



## § Login request (save token as a variable)



## § Admin endpoint request (using token)



### 3. Lab Report:

- o Answers to questions.
- o Screenshots of Postman tests (successful and failed auth).

#### 1. Token Usage:

- What happens if you omit the Authorization header when accessing /admin/users/{id}?
  - The server will reject the request, usually returning a **401 Unauthorized** status code because the endpoint requires a valid JWT token for authentication. Without the Authorization header, the server doesn't know who you are.
- What error occurs if you use an expired token?
  - get a **401 Unauthorized** error with a message like "Token expired" or "JWT expired". The server checks the token expiry time (exp claim), and if expired, it rejects the request.
- Paste the token generated to <https://jwt.io>, and discuss the content it decodes the token into three parts:
  1. **Header:**  
Usually contains type (typ: "JWT") and signing algorithm (alg: e.g., "HS256").

## 2. Payload:

Contains claims such as:

- sub (subject, e.g., user ID)
- iat (issued at timestamp)
- exp (expiry timestamp)
- role (user role like admin, customer, driver)
- Any other custom claims your app adds.

## 3. Signature:

Used to verify the token hasn't been tampered with, created using the header, payload, and a secret key.

## 2. Role Restrictions:

- If a customer-role user tries to access `/admin/users/{id}`, what status code is returned?
  - Typically, **403 Forbidden** is returned. The user is authenticated but does not have permission (role) to access the admin endpoint
- How would you modify the middleware to allow both admin and driver roles to access an endpoint?
  - instead of checking for just admin, check if the user role is either admin **OR** driver.

## 3. Security:

- Why is the JWT token sent in the Authorization header instead of the request body?
  - **Authorization header** is standardized for sending credentials like tokens.
  - Headers are included in every request, regardless of method (GET, POST, etc.), while the body only exists in methods like POST or PUT.
  - Sending the token in headers keeps it consistent and easier for servers and proxies to handle authentication.
- **How does password hashing protect user data in a breach?**
  - Password hashing transforms passwords into a fixed-length string that can't be reversed (one-way).
  - Even if attackers steal the hashed passwords, they can't get the original passwords easily.
  - Strong hashing algorithms (e.g., bcrypt, Argon2) add salt and multiple rounds, making brute-force attacks very difficult.

## 4. Postman Testing:

- What is the purpose of the **Bearer** keyword in the **Authorization** header?
- It tells the server that the client is sending a Bearer Token, a type of token used in OAuth 2.0
- Format: **Authorization: Bearer <token>**
- Helps the server parse and validate the token properly

- How would you test a scenario where a user enters an incorrect password?

- **Use a Postman:**

**Method: POST**

**Endpoint: /auth/login**

**Body:**

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

- Expected response:

```
{  
  "error": "Invalid credentials"  
}
```

With status **401 Unauthorized**