Software Requirements Specification (SRS)

Project Title: Mutual Fund Account Management System

Technology Stack: Spring Boot 3, Java 17, Spring Security, JPA, MySQL, JWT, Swagger, Postman, Lombok, Maven

1. Project Overview

The Mutual Fund Account Management System (MFAMS) is a secure, web-based platform that allows customers to manage mutual fund investments online. The system facilitates seamless user registration, login via JWT-based authentication, mutual fund browsing, transactions like buy/sell, and portfolio tracking. Admin users can manage the mutual fund catalog by adding or updating funds. Customers can invest in available funds and view detailed investment history and current portfolio performance. The system is built using Spring Boot (Java) on the backend, with MySQL as the database. JWT is used for stateless, secure authentication, and Swagger UI enables interactive API exploration. Postman is used for thorough backend testing during development.

Core user roles in this system include:

Admin – manages mutual funds.

User – registers, invests in funds, and monitors transactions and portfolio.

2. Functional Requirements

1. Users should be able to register with their name, email, and password.
2. Registered users can log in and receive a JWT token.
3. Users can view a list of mutual funds.
4. Authenticated users can buy and sell mutual funds.
5. Users can view their complete transaction history.
6. Users can view a portfolio summary showing invested amount, current NAV, units, and profit/loss.
7. Admin users can add new mutual funds to the catalog.
8. JWT authentication should protect all endpoints except register/login.
9. NAV values for funds should update at regular intervals using scheduled tasks.

3. Non-Functional Requirements

1. The system should provide secure access using JWT tokens.
2. It must follow RESTful API standards.
3. The backend must be responsive and handle concurrent requests efficiently.
4. Token-based authentication should remain stateless.
5. The system should be modular, maintainable, and scalable.
6. All APIs should be documented and testable via Swagger UI.
7. Users should receive proper error messages for expired tokens or invalid requests.

4. Entity Relationship (ER) Diagram

This section will include a visual diagram showcasing key entities and relationships:

- User (1) → (N) Transaction
- MutualFund (1) → (N) Transaction

Each transaction contains information like fundId, userId, units, type (BUY/SELL), and amount.

5. API Workflow via Postman

Step 1: Register a New User

POST http://localhost:8080/api/auth/register

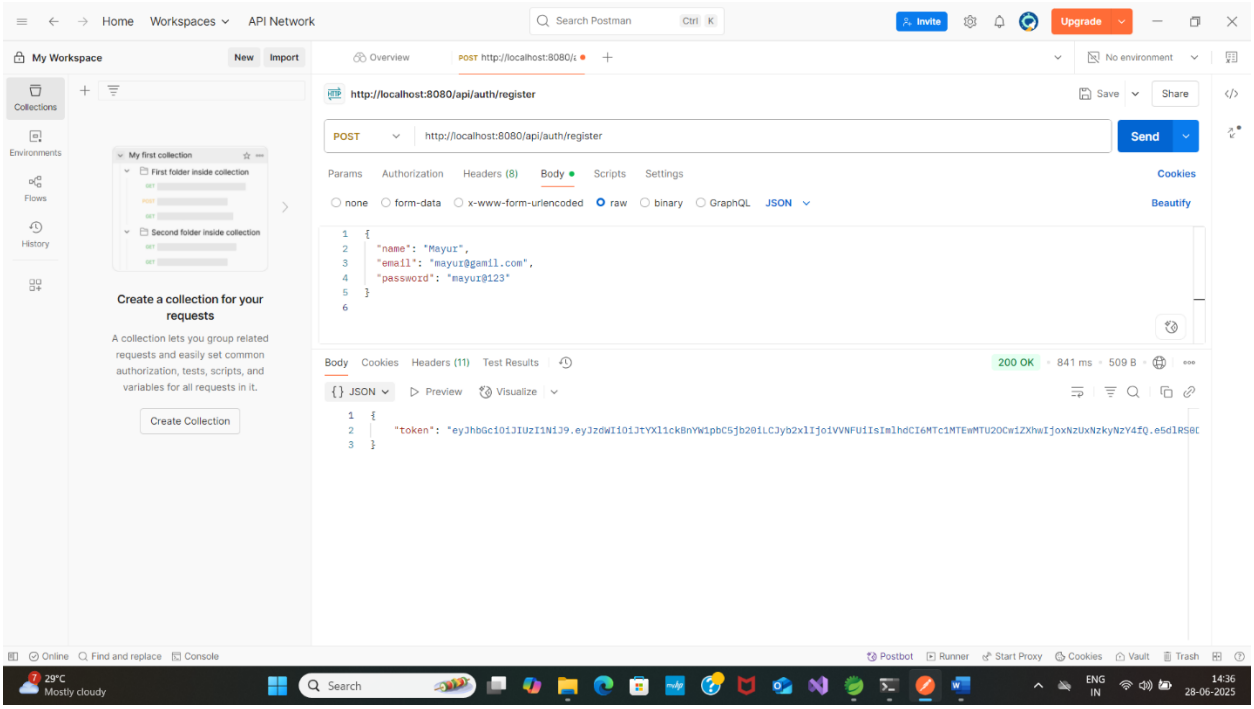
Used to create a new user. No token required.

Request:

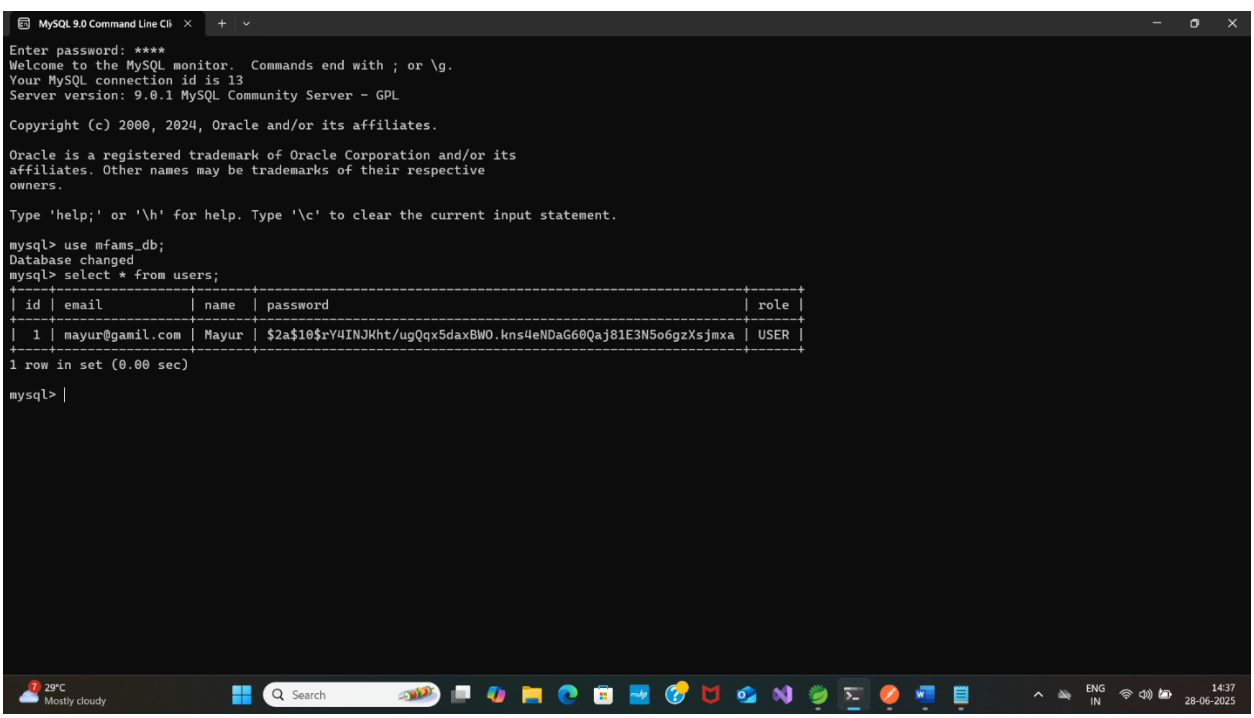
json:

```
{
  "name": "Mayur",
  "email": "mayur@gamil.com",
  "password": "mayur@123"
}
```

Response: Returns a JWT token.



a. Postman request/response



b. SQL users table showing the new user

Step 2: Login as User

POST http://localhost:8080/api/auth/login

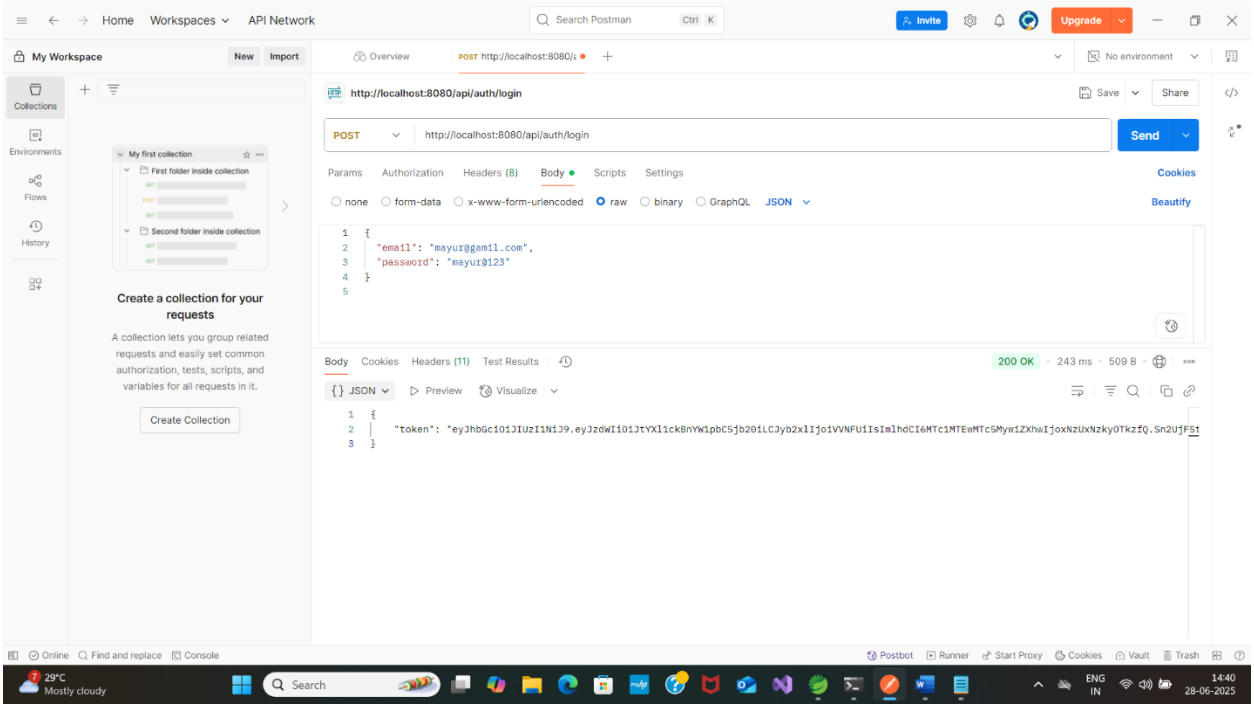
Used to authenticate and receive a new JWT token.

Request:

json:

```
{
  "email": "mayur@gamil.com",
  "password": "mayur@123"
}
```

Response: JWT token string.



c. JWT token string

Step 3: Get Logged-in User Info

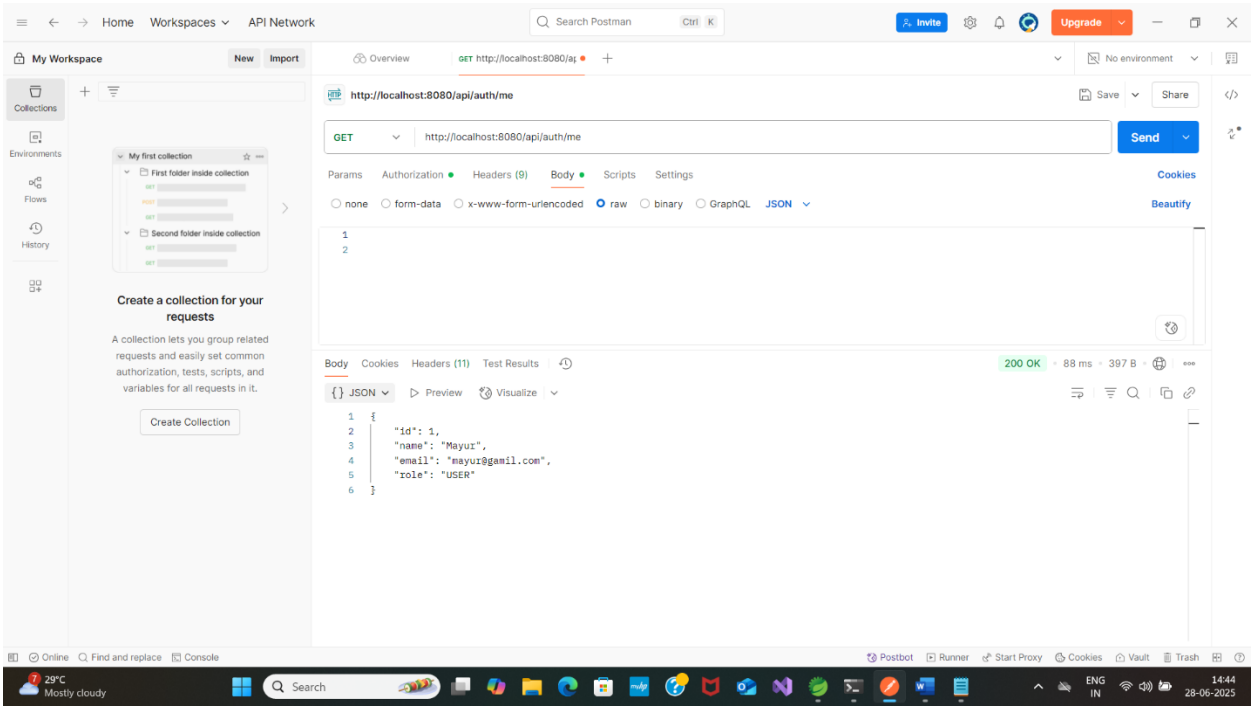
GET http://localhost:8080/api/auth/me

Returns current user details. Requires Authorization header with Bearer token.

Header:

Authorization: Bearer <your_token>

Response:



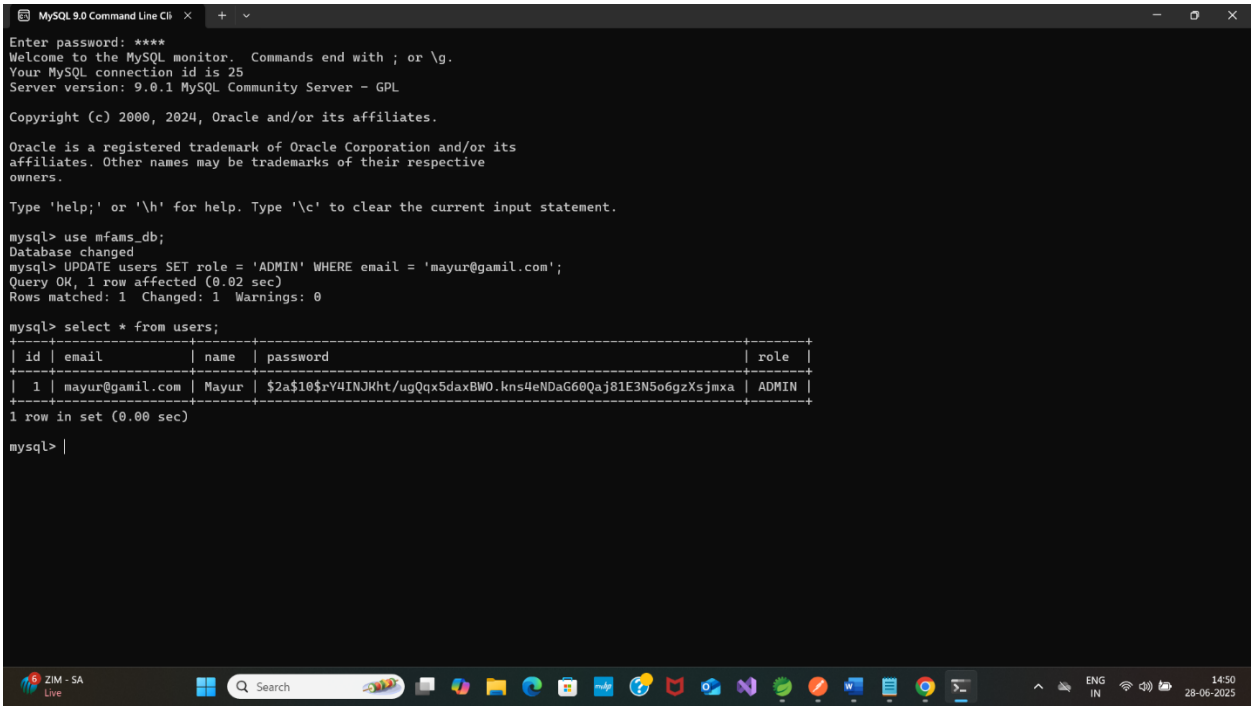
d. json Response showing user info

Step 4: Update the role of the user manually in db.

SQL : UPDATE users SET role = 'ADMIN' WHERE email = 'mayur@example.com';

This updates the role of the user with the given email to 'ADMIN'. After updating, the user will have access to admin-restricted endpoints

Screenshot :



e. Users Table Showing Updated Role

Step 5: Admin Adds Mutual Fund

POST http://localhost:8080/api/funds

Used by Admin to add mutual funds.

Request:

json:

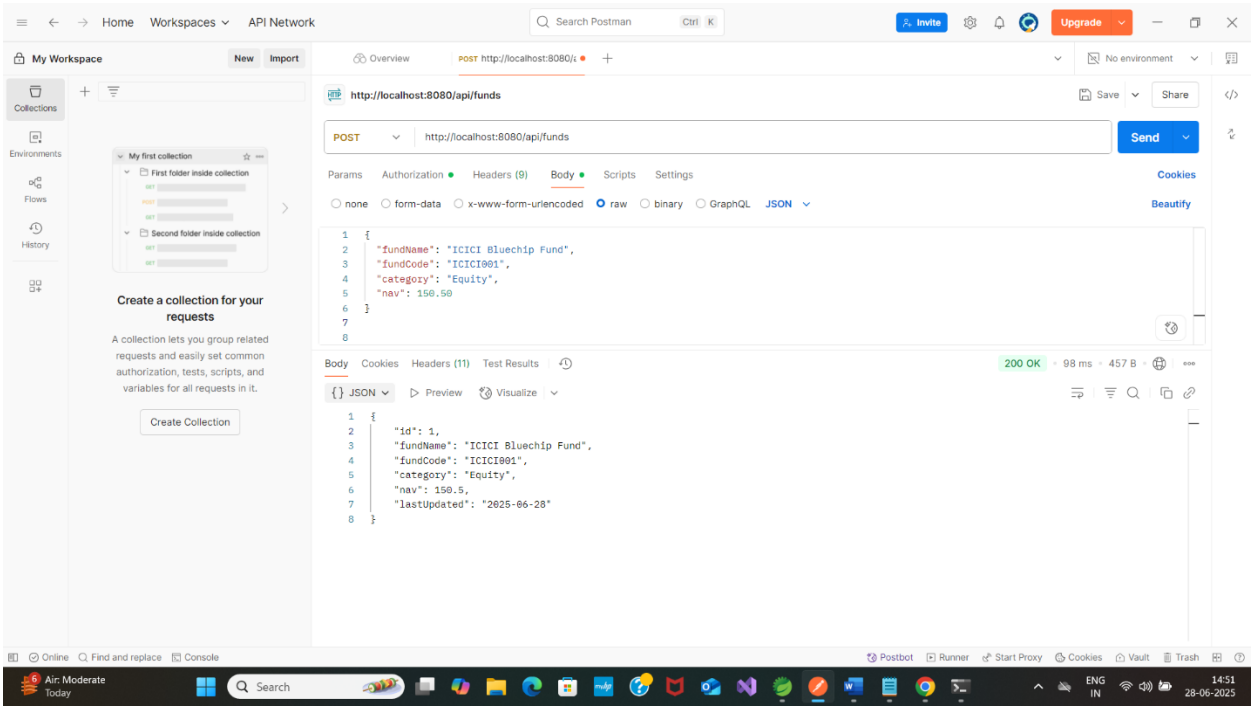
```
{
  "fundName": "ICICI Bluechip Fund",
  "fundCode": "ICICI001",
  "category": "Equity",
  "nav": 150.50
```

}

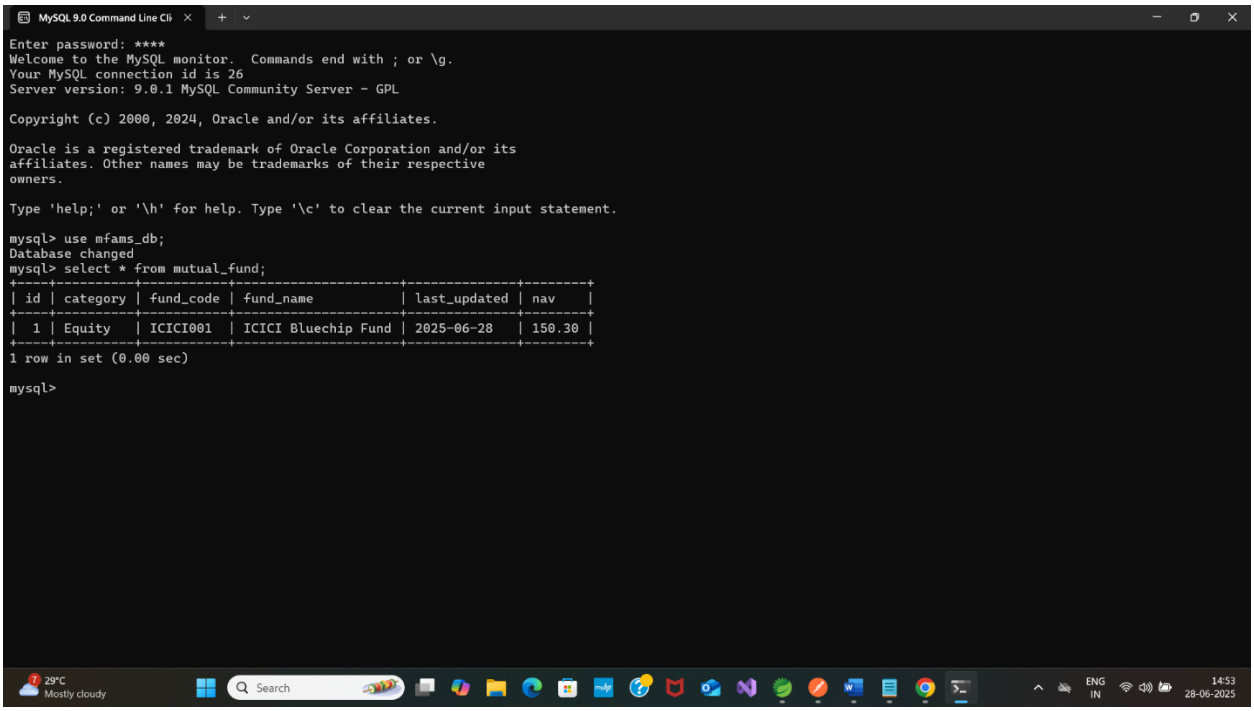
Header:

Authorization: Bearer <admin_token>

Response:



f. Postman Request/Response



g.mutual_fund table

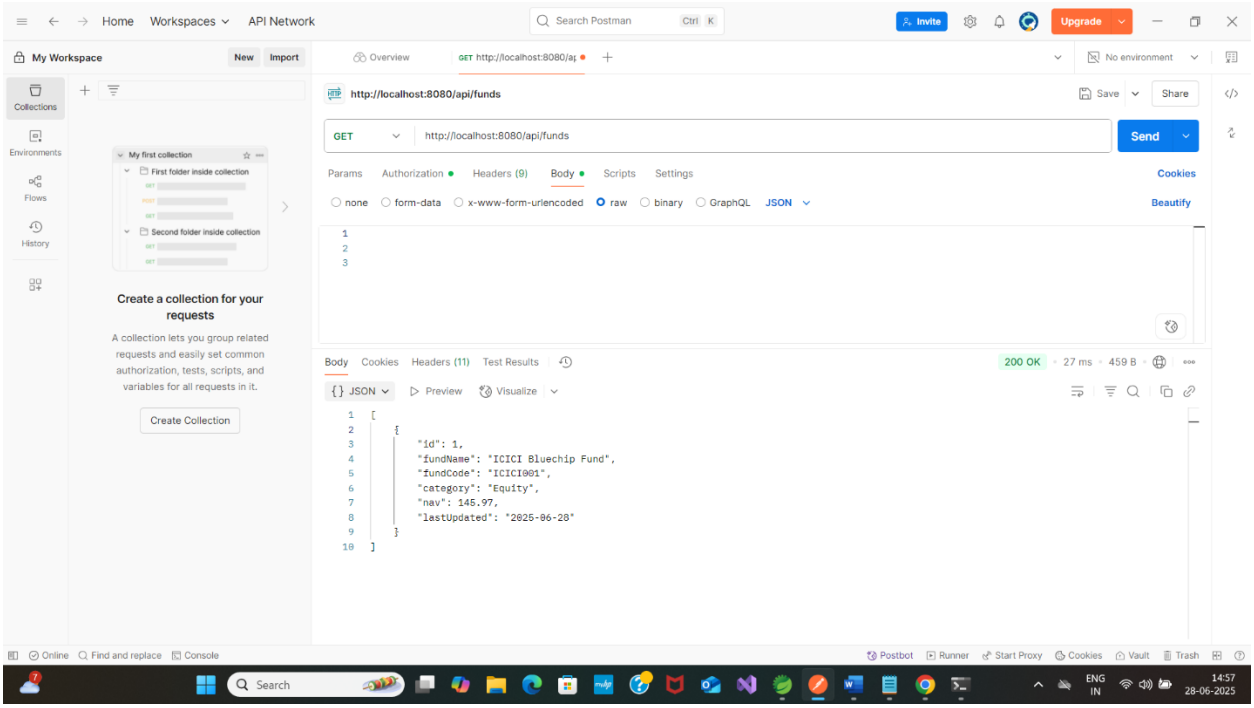
Step 6: Get All Mutual Funds

GET http://localhost:8080/api/funds

Lists all available mutual funds for investment.

Header: Bearer token required.

Response:



h. Showing Fund List

Step 7: Buy Mutual Fund

POST http://localhost:8080/ /api/transactions/buy

Allows a user to buy a specific mutual fund.

Request:

json:

{

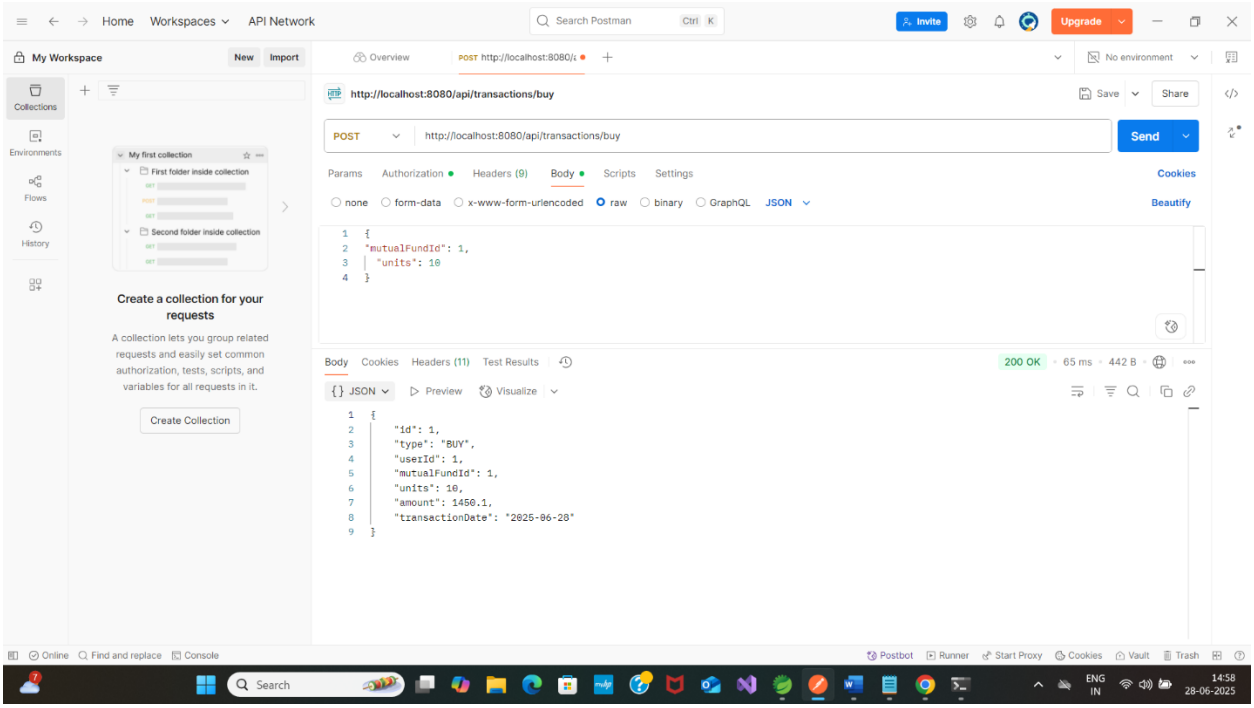
"mutualFundId": 1,

"units": 10

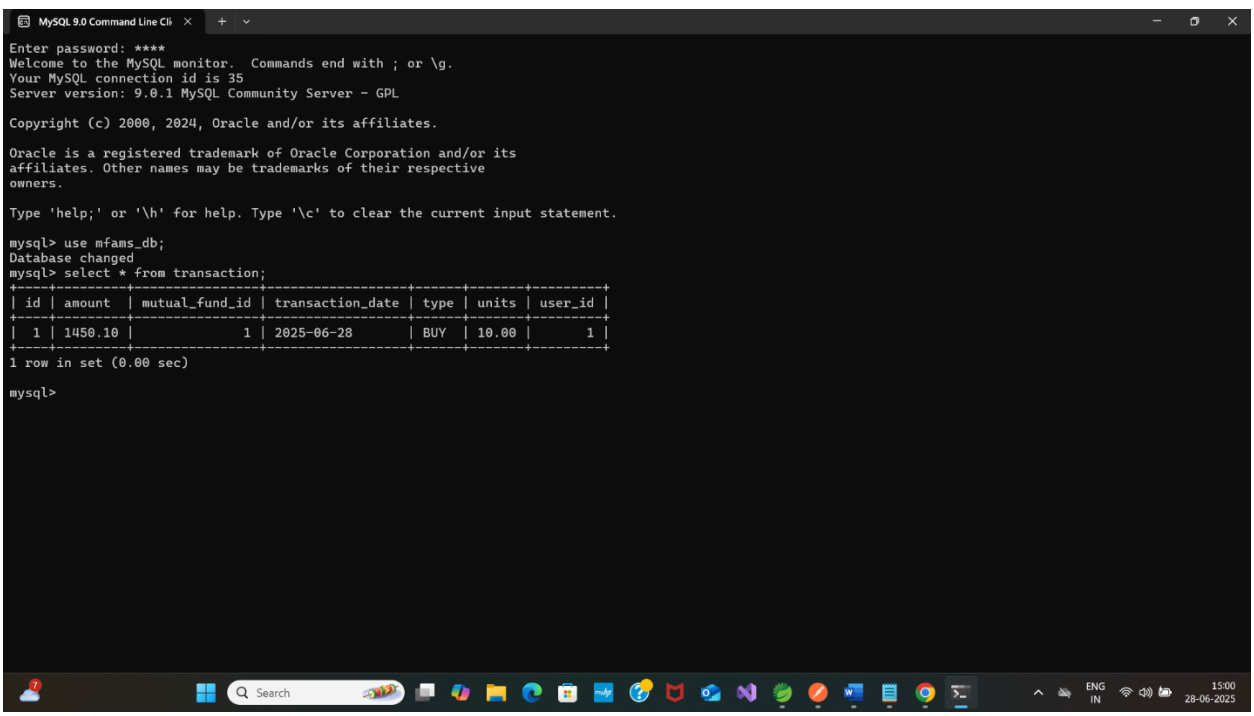
}

Header: Bearer token required.

Response:



i. Buy Response



a. Updated Transaction Table

Step 8: Buy Mutual Fund

POST http://localhost:8080/ /api/transactions /sell

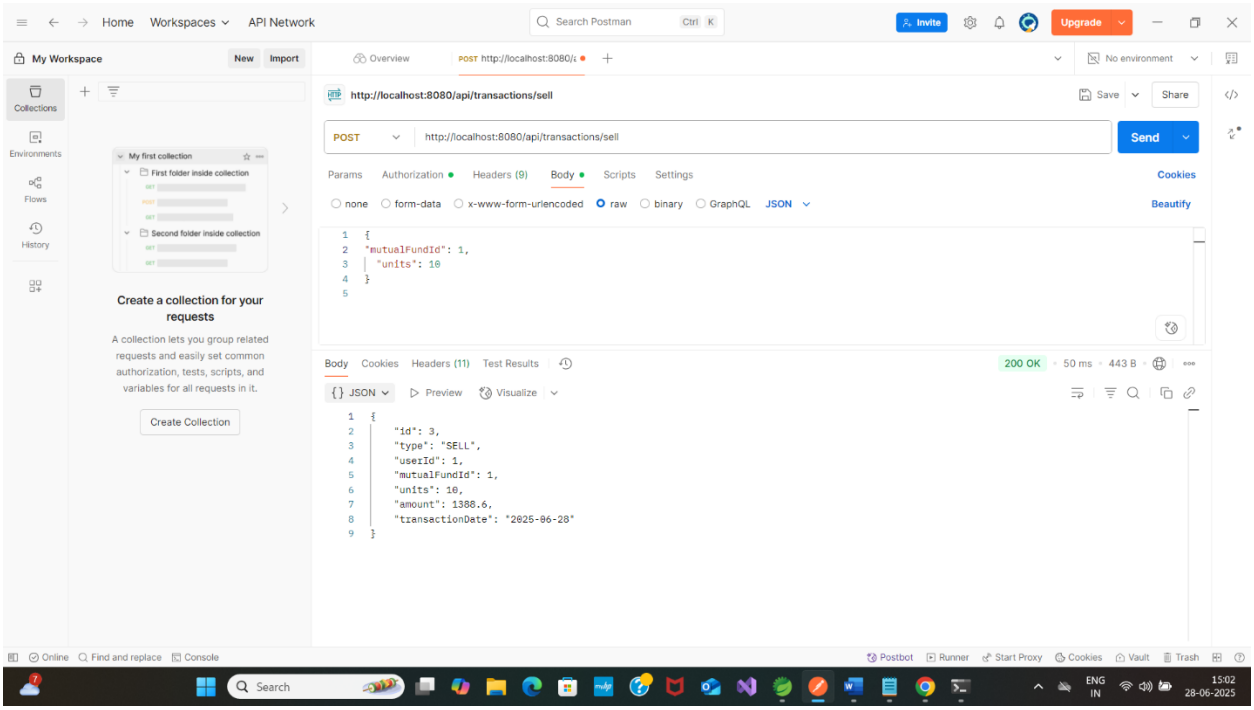
Allows a user to sell previously bought units.

Request:

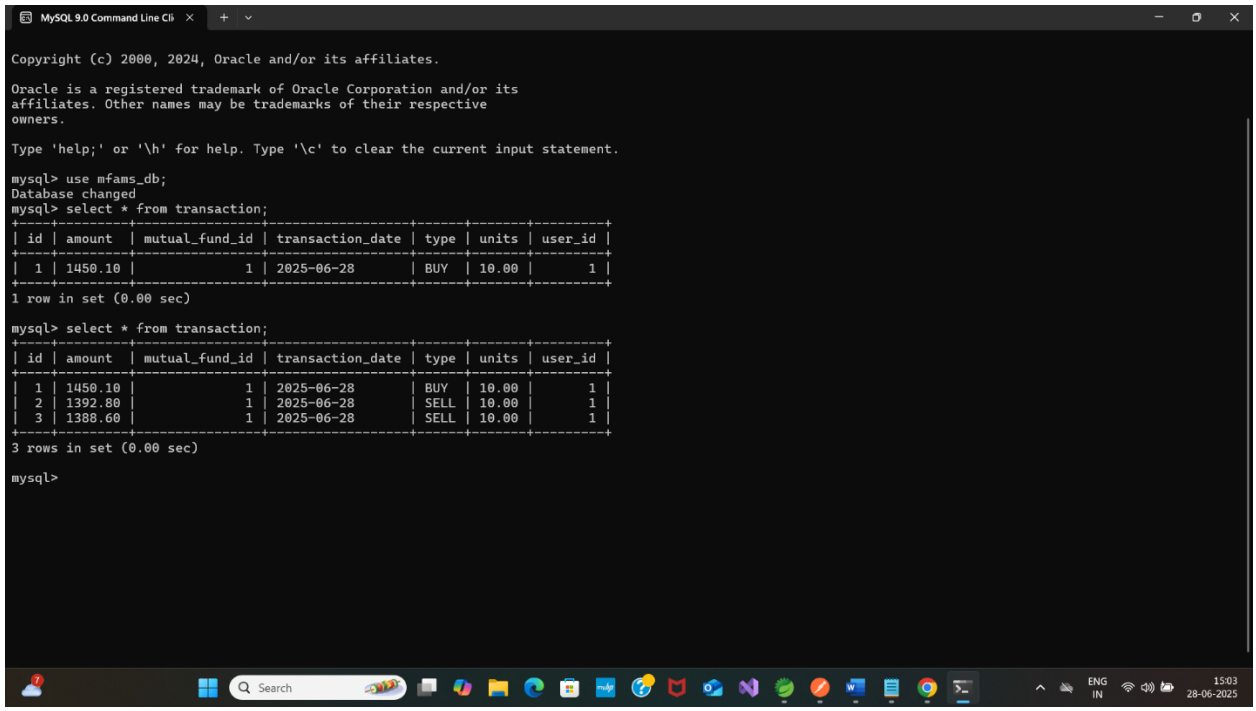
json:

```
{
"mutualFundId": 1,
  "units": 10
}
```

Response:



b. Response After Selling



```
MySQL 9.0 Command Line Cli

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use mfams_db;
Database changed
mysql> select * from transaction;
+----+---------+----------------+------------------+------+-------+---------+
| id | amount  | mutual_fund_id | transaction_date | type | units | user_id |
+----+---------+----------------+------------------+------+-------+---------+
|  1 | 1450.10 |              1 | 2025-06-28       | BUY  | 10.00 |       1 |
+----+---------+----------------+------------------+------+-------+---------+
1 row in set (0.00 sec)

mysql> select * from transaction;
+----+---------+----------------+------------------+------+-------+---------+
| id | amount  | mutual_fund_id | transaction_date | type | units | user_id |
+----+---------+----------------+------------------+------+-------+---------+
|  1 | 1450.10 |              1 | 2025-06-28       | BUY  | 10.00 |       1 |
|  2 | 1392.80 |              1 | 2025-06-28       | SELL | 10.00 |       1 |
|  3 | 1388.60 |              1 | 2025-06-28       | SELL | 10.00 |       1 |
+----+---------+----------------+------------------+------+-------+---------+
3 rows in set (0.00 sec)

mysql>
```

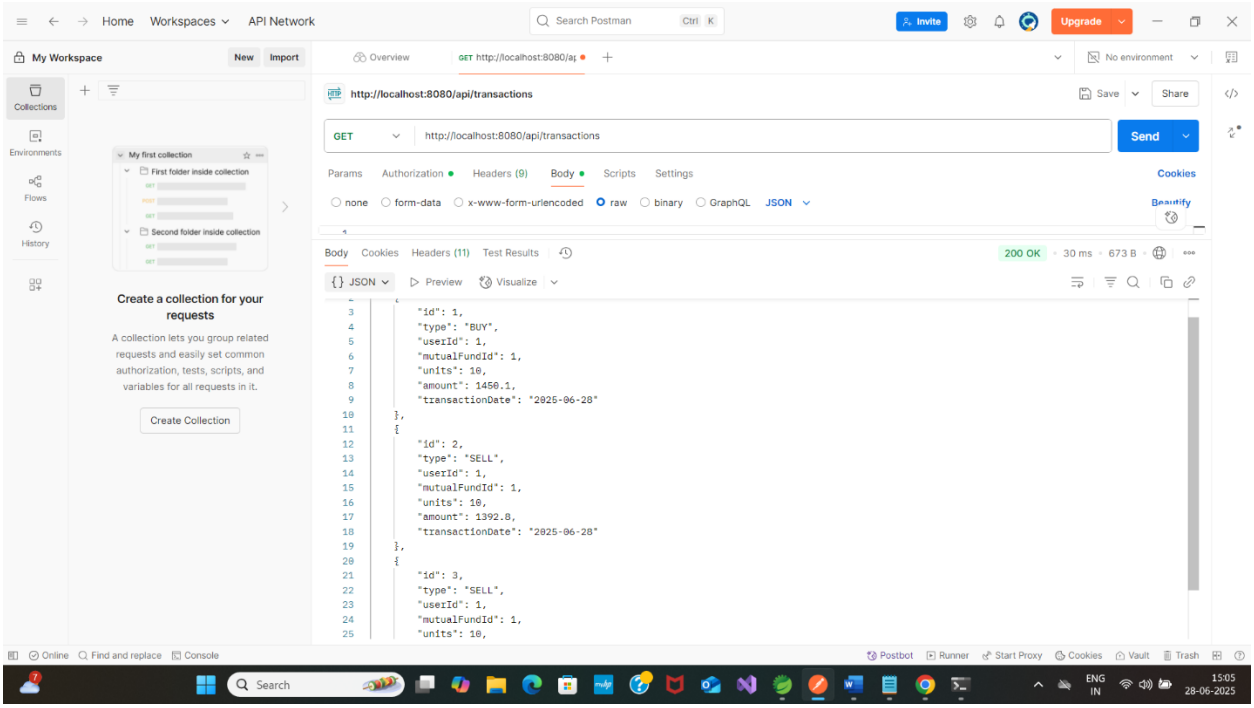
c. Updated Transactions in DB.

Step 9: View All Transactions

GET http://localhost:8080/ /api/transactions

Shows full history of buy/sell.

Response:



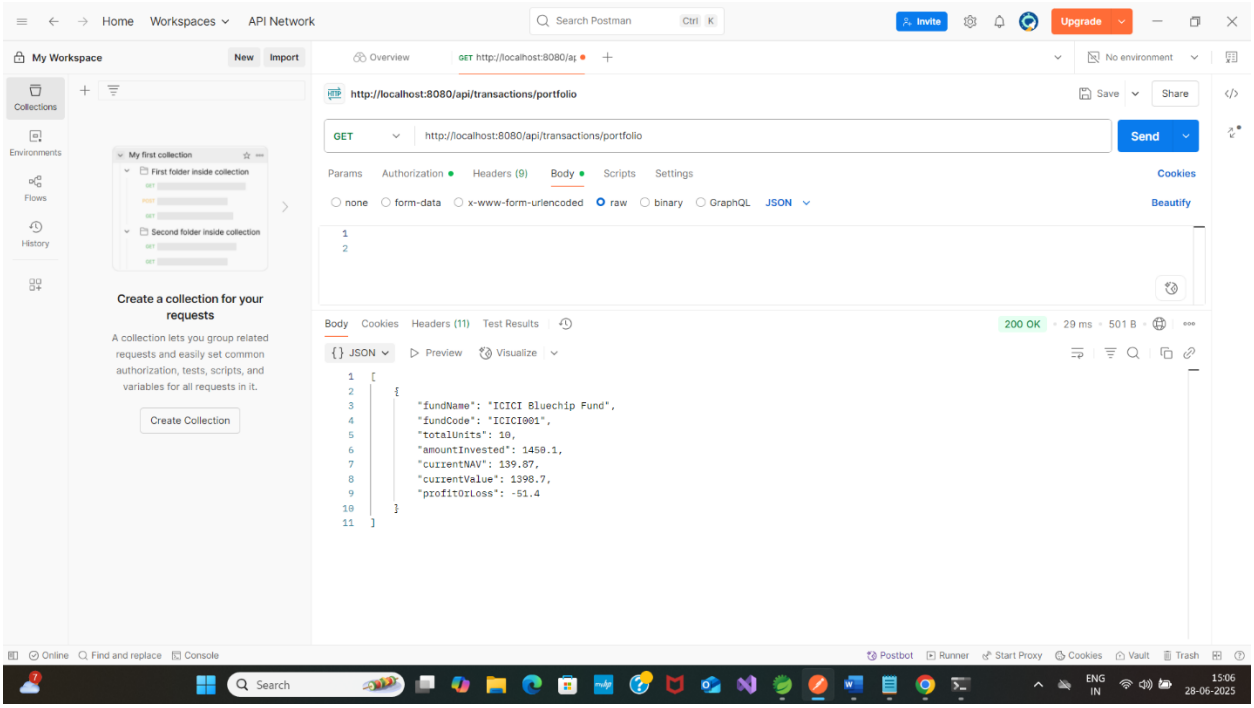
d. Showing Transaction History.

Step 10: View Portfolio Summary

GET http://localhost:8080/ /api/ transactions/ portfolio

Shows total units, NAV, invested amount, and profit/loss for each fund.

Response:



e. Response for Summarized Portfolio.

6. Tools & Technologies Used

1. Backend Framework: Spring Boot 3.0+
2. Authentication: JWT (JSON Web Tokens)
3. Database: MySQL (with JPA & Hibernate)

4. API Documentation: Swagger using Springdoc
5. Testing & Debugging: Postman
6. ORM: Spring Data JPA
7. Utilities: Lombok (for getters/setters), Maven
8. Java Version: Java 17