

Full -stack Strategies and Development



Job Listings Backend API Documentation (CW1)

Submitted By

Ram Naresh Yadav

Student ID: 10290454

Course Instructor:

SIHAN HAROON

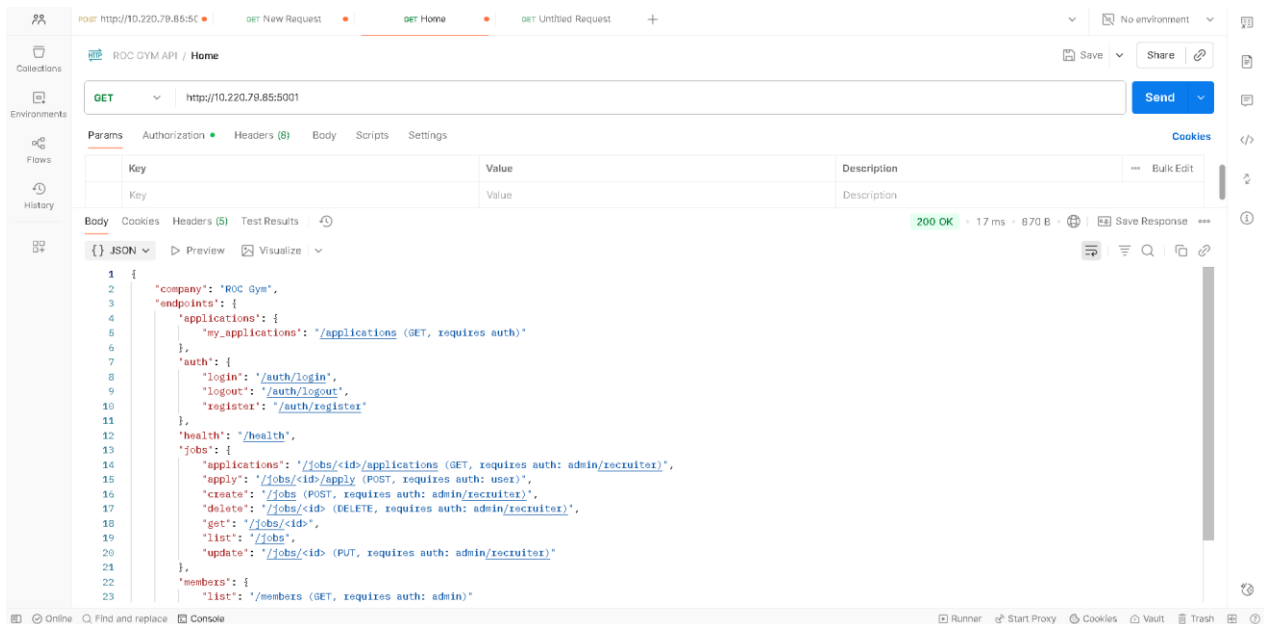
Project Overview

The job listings and employee management system at ROC Gym are the main focus of this project. It includes comprehensive details about job openings, personnel information, and activities associated with the job at ROC Gym. The dataset is perfect for CRUD (Create, Read, Update, Delete) operations because it is made to manage a well-defined and structured data model.

A MongoDB database will house the data in JSON format, allowing for scalable future growth, flexible document querying, and effective data storage.

Test Home Endpoint

Public Endpoints (Authentication) Endpoint



- **Endpoint:** GET ROC GYM Job listing
- **Description:** Serves as a built-in API documentation endpoint.
- **Response:** 200 ok
- **Query Parameters:**
 - `/jobs/<id>/applications (GET)`: Retrieve applications for a specific job.
 - `/jobs/<id>/apply (POST)`: Apply for a specific job.

- /jobs (POST): Create a new job listing.
- /jobs/<id> (DELETE): Delete a job listing by its ID.
- /jobs/<id> (GET): Retrieve a job listing by its ID.
- /jobs (GET): List all job listings.
- /jobs/<id> (PUT): Update a job listing by its ID.
- /auth/login (POST): Authenticate a user and retrieve a JWT token.
- /auth/register (POST): Register a new user (user/recruiter/admin).

2. Test: Health Check

The screenshot shows a REST client interface with the following details:

- Request:** GET http://10.220.79.85:5001/health
- Response:** 200 OK, 284 ms, 250 B
- Body (JSON):**

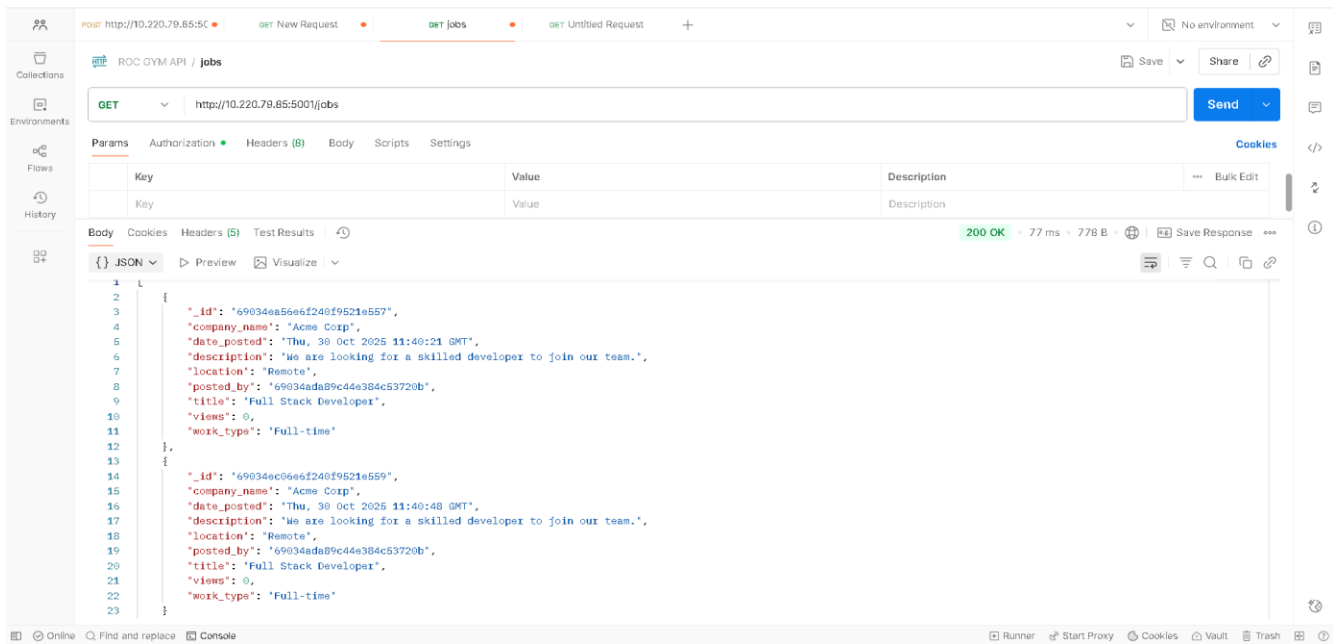
```
{
  "database": "connected",
  "status": "healthy",
  "timestamp": "2025-11-06T10:43:14.675166"
}
```

Endpoint: Get/health

Description: Quick status check for server health and database connectivity.

Authentication: Not required

3. Test: Get Jobs (Empty List):



Endpoint: Get/jobs

Description: Retrieve applications for a specific job.

Response:

Success (200): Retrieve a single job listing by ID.

Error (401): invalid or missing token

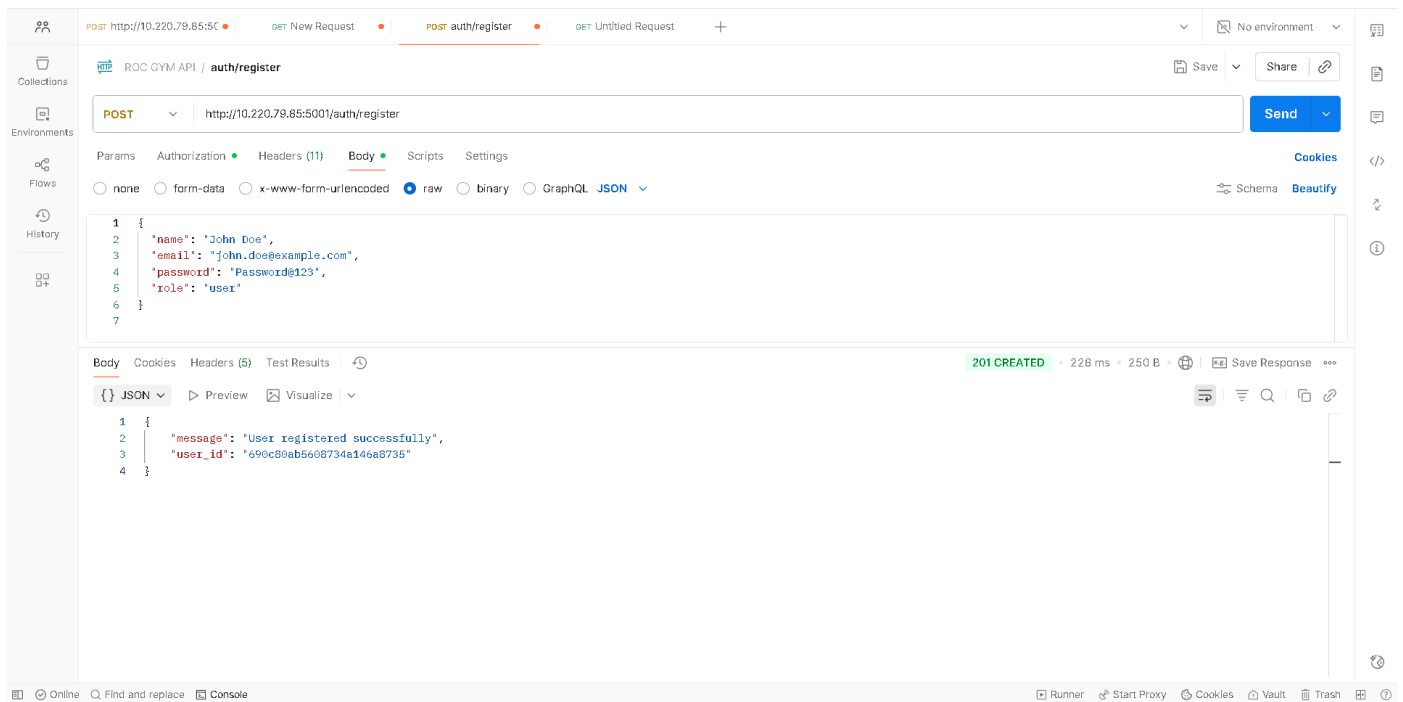
Delete User Account

Endpoint: DELETE/auth/delete

Description: Delete the currently authenticated user's account.

1. Test_register_user:

AUTHENTICATION ENDPOINT



Endpoint: POST/auth/register

Description: Registers a new user in the system

Request Body:

- Username (string, required): User's unique username.
- Password (string, required): User's password
- Role (string optional): user, recruiter and admin.

2. test_register_duplicate_email:

The screenshot shows a REST client interface with a sidebar on the left containing icons for Collections, Environments, Flows, and History. The main area displays a POST request to `http://10.220.79.85:5001/auth/register`. The request body is a JSON object:

```
1 {
2   "name": "John Doe",
3   "email": "john.doe@example.com",
4   "password": "Password@123",
5   "role": "user"
6 }
```

The response is a **409 CONFLICT** with a status bar indicating 49 ms and 261 B. The response body is shown in JSON format:

```
1 {
2   "debug": {
3     "email": "john.doe@example.com"
4   },
5   "error": "User with this email already exists."
6 }
```

The bottom status bar includes links for Online, Find and replace, Console, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

Endpoint: Post/auth/login

Description: register with duplicate email

Response:

- Error (409) conflict

3. test_register_invalid_role:

The screenshot shows a Postman interface for a POST request to `http://10.220.79.85:5001/auth/register`. The request body is a JSON object:

```
1 {
2   "name": "John Doe",
3   "email": "john.doe@example.com",
4   "password": "Password@123",
5   "role": "manager"
6 }
7
```

The response status is **400 BAD REQUEST** (7 ms, 209 B). The response body is a JSON object:

```
1 {
2   "error": "Invalid role specified"
3 }
```

Checks:

- Endpoint: Post/Auth/register
- Error (400): invalid role specified
- Rejects roles outside “user”, “recruiter”, “admin”

The screenshot shows a Postman interface for a POST request to `http://10.220.79.85:5001/auth/login`. The request body is a JSON object:

```
1 {
2   "name": "John Doe",
3   "email": "john.doe@example.com",
4   "password": "Password@123",
5   "role": "manager"
6 }
7
```

The response status is **200 OK** (224 ms, 352 B). The response body is a JSON object:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9keiIjOiJkaWZlbnkiwYzgwYWI1NjA4NzMyTE0NlE4NmZlIiwicm9sZSI6ImVzZXIiLCJ1eHAiOiJlMzNjI0MzEyMzR9.E-KsEz0W7qmHJfXPejciYKQo53t8NcQgZRAe317UY"
3 }
```

Test login success:

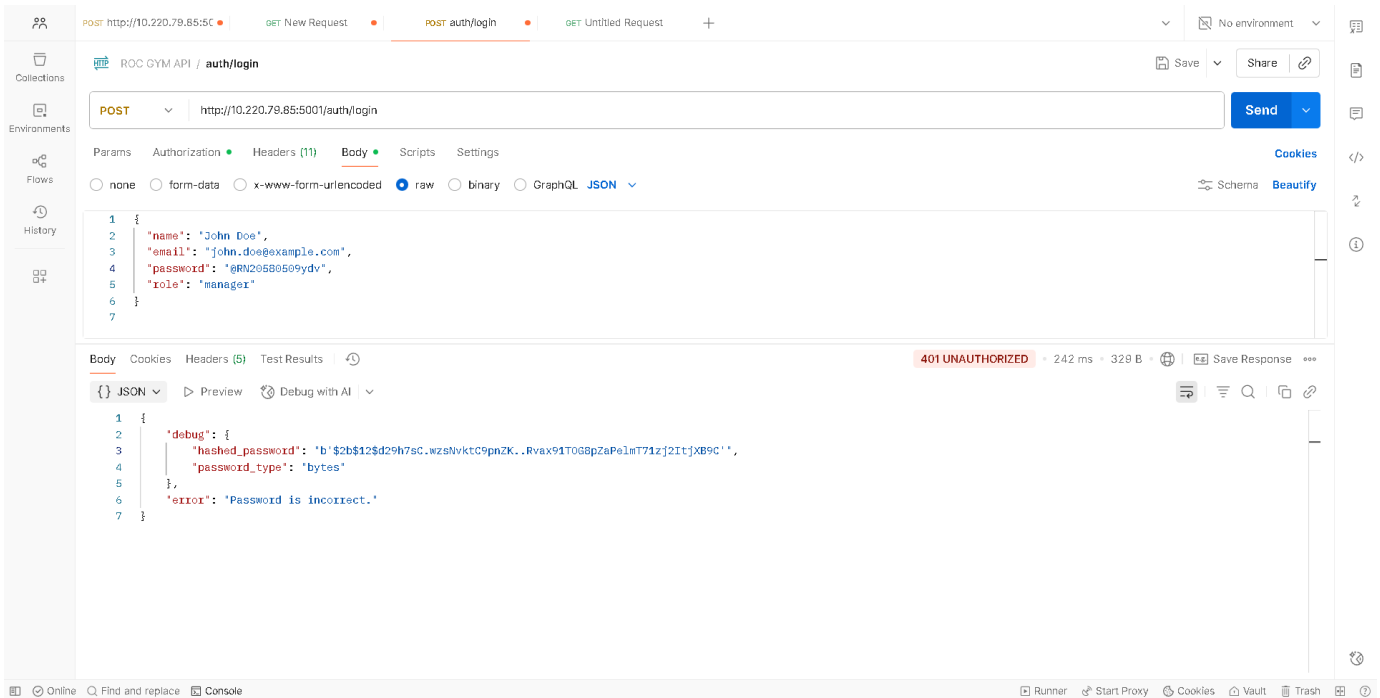
Checks:

- Endpoint: Post/auth/login
- Token is returned
- **Authentication:** Basic authentication(username and password)

Response:

- **Success (200):** Returns JWT token
- **Error (401):** invalid credentials
- **Error (404) :** user not found.

4. test_login_wrong_password:



Checks:

- Endpoint: Post/Auth/login
- Description: password incorrect
- Status = 401 unauthorized
- Proper error message

5. test_login_nonexistent_user

The screenshot shows a REST client interface with a sidebar on the left containing 'Collections', 'Environments', 'Flows', 'History', and a 'New' button. The main area displays a POST request to 'http://10.220.79.85:5001/auth/login' with a JSON body:

```
{
  "name": "John Doe",
  "email": "rohitmahaseeth@gmail.com",
  "password": "eRN20580509ydv",
  "role": "manager"
}
```

. The 'Body' tab is selected, showing the response:

```
{
  "debug": {
    "email": "rohitmahaseeth@gmail.com"
  },
  "error": "User with that email does not exist."
}
```

. The status bar indicates a '401 UNAUTHORIZED' response with a 36 ms duration and 268 B size. The bottom status bar includes 'Online', 'Find and replace', 'Console', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and a help icon.

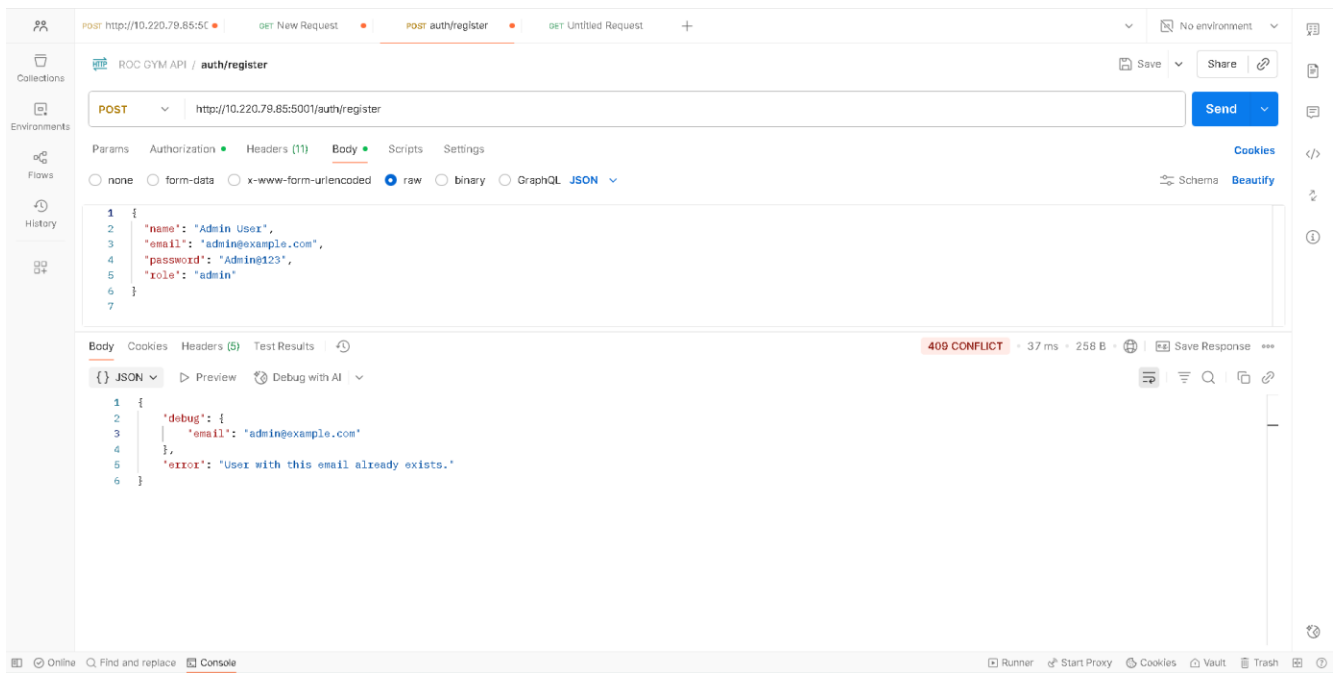
Checks:

- Endpoint: Post/Auth/Login
- **Status** = 401 unauthorized
- Error message matches.

JOB MANAGEMENT Test

A. Register Admin/Recruiter/User (3 requests)

➤ Admin Register

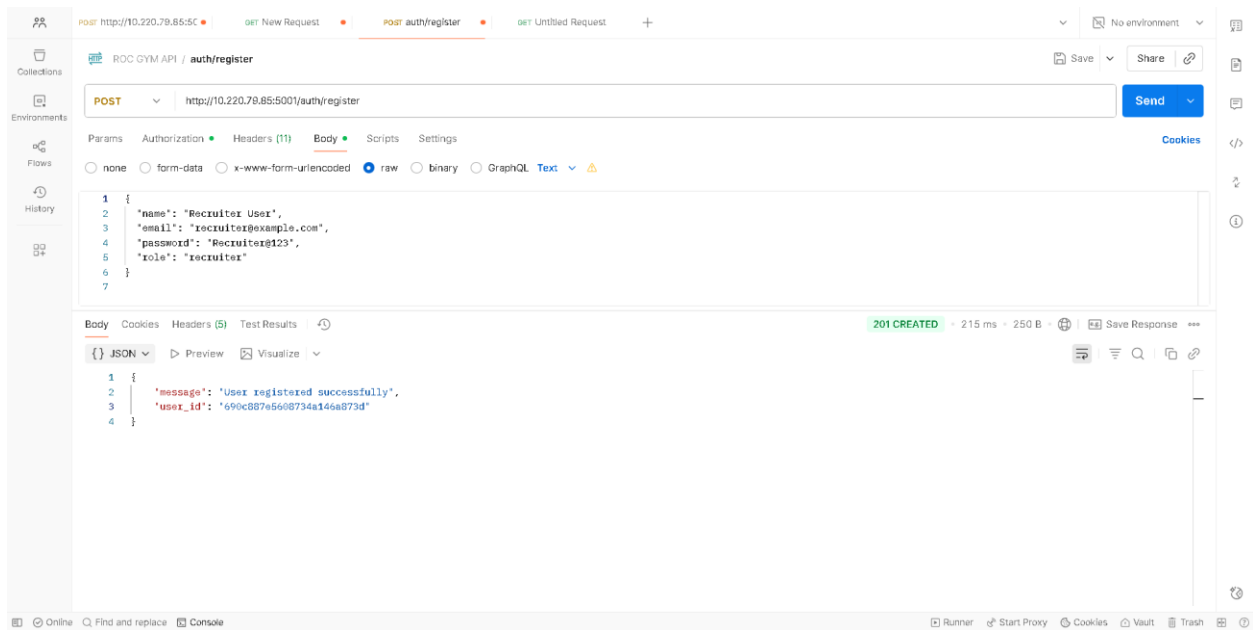


Endpoint: Post/Auth/login

Description: it stores different token for user, recruiter and admin

Error (409): conflict

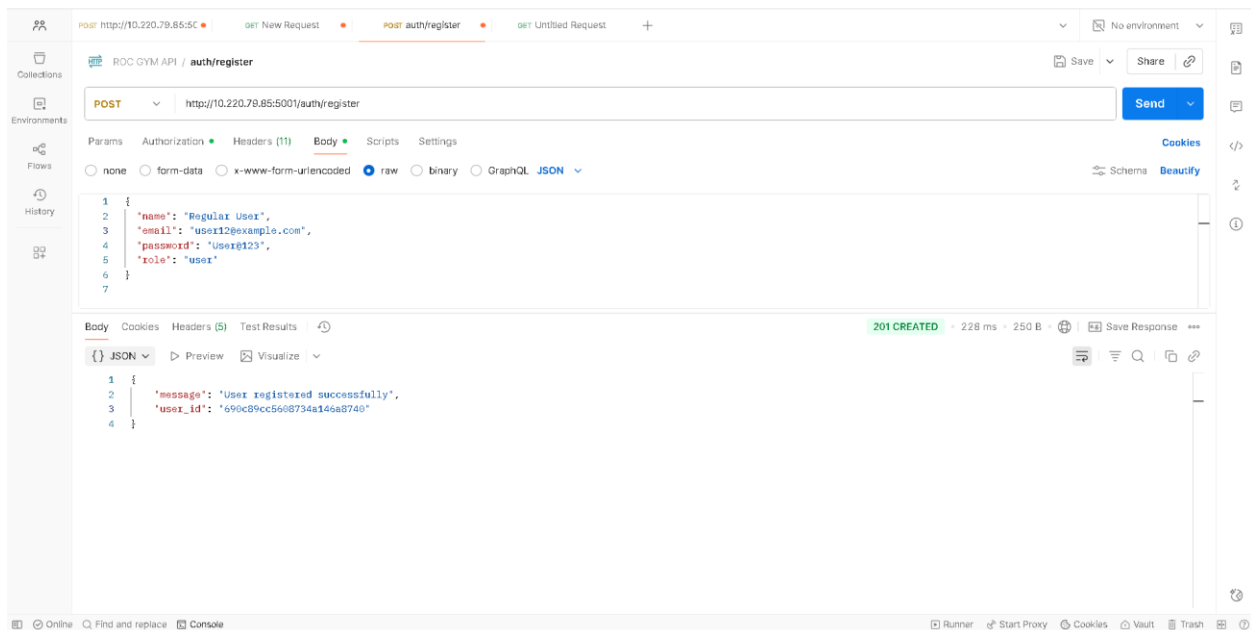
For recruiter



Endpoint: Post/Auth/Login

Description:

FOR register USER



B. Login & store tokens (3 requests)

1). Login & store tokens (admin)

The screenshot displays a REST client interface with the following details:

- Request:**
 - Method: **POST**
 - URL: **http://10.220.79.85:5001/auth/login**
 - Body (JSON):

```
1 {
2   "name": "Admin User",
3   "email": "admin@example.com",
4   "password": "password123",
5   "role": "admin"
6 }
7
```
- Response:**
 - Status: **200 OK**
 - Body (JSON):

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9kaWkiOiJkbG9kaWVlIiwiaWF0IjoxNzYyMDEzOTQyLCJpYXN0bnVzc2UiOiJkbG9kaWVlIn0.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9kaWkiOiJkbG9kaWVlIiwiaWF0IjoxNzYyMDEzOTQyLCJpYXN0bnVzc2UiOiJkbG9kaWVlIn0"
3 }
```

The interface also shows a sidebar with 'Collections', 'Environments', 'Flows', and 'History'. The bottom status bar includes 'Online', 'Find and replace', 'Console', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and a help icon.

2. Login & store tokens (user)

The screenshot displays a REST client interface with the following details:

- Request:**
 - Method: **POST**
 - URL: **http://10.220.79.85:5001/auth/login**
 - Body (raw):

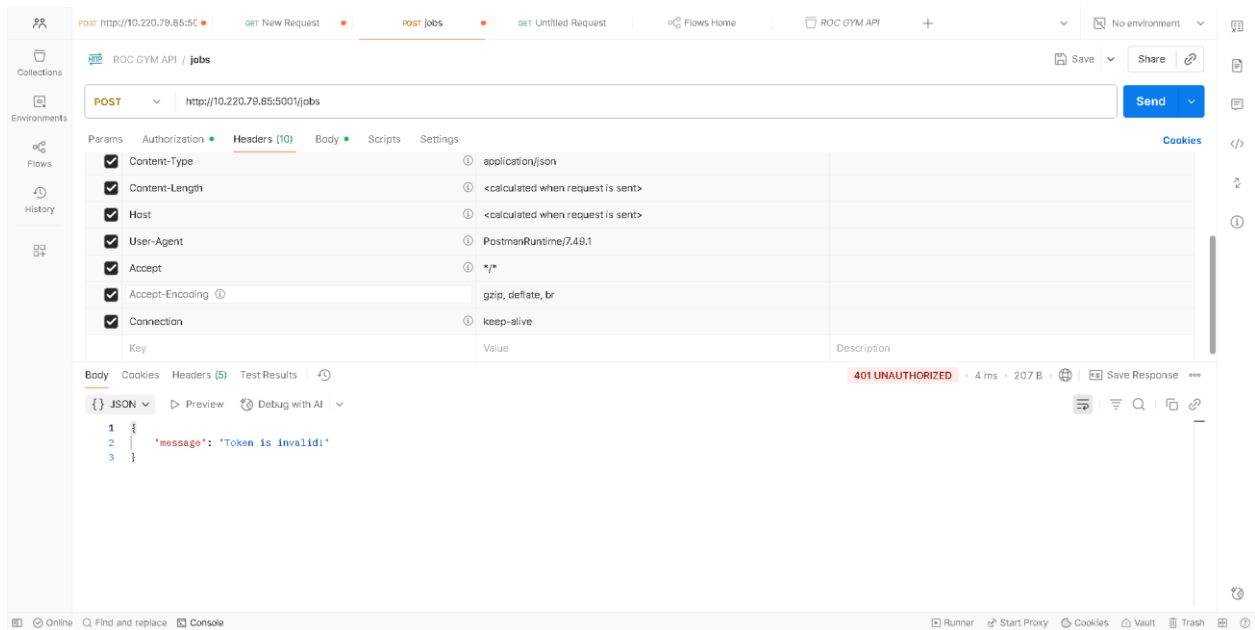
```
1 {
2   "name": "Regular User",
3   "email": "user@example.com",
4   "password": "password123",
5   "role": "user"
6 }
7
```
- Response:**
 - Status: **200 OK**
 - Body (JSON):

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX21kIjoiaWJkbWZQZDc4NmRhODg1YThlbnZg1Yjg1Iiwicm9sZSI6InVzZXIiLCJleHAiOjE3NjI0MzQxOTN9.kuvQZlJCIBI8AXvveY4vqjMETfsVnVXiJvDL9TN2GB"
3 }
```

The interface includes a sidebar with navigation options (Collections, Environments, Flows, History) and a bottom status bar with various tool icons.

3. Login & store tokens (recruiter)

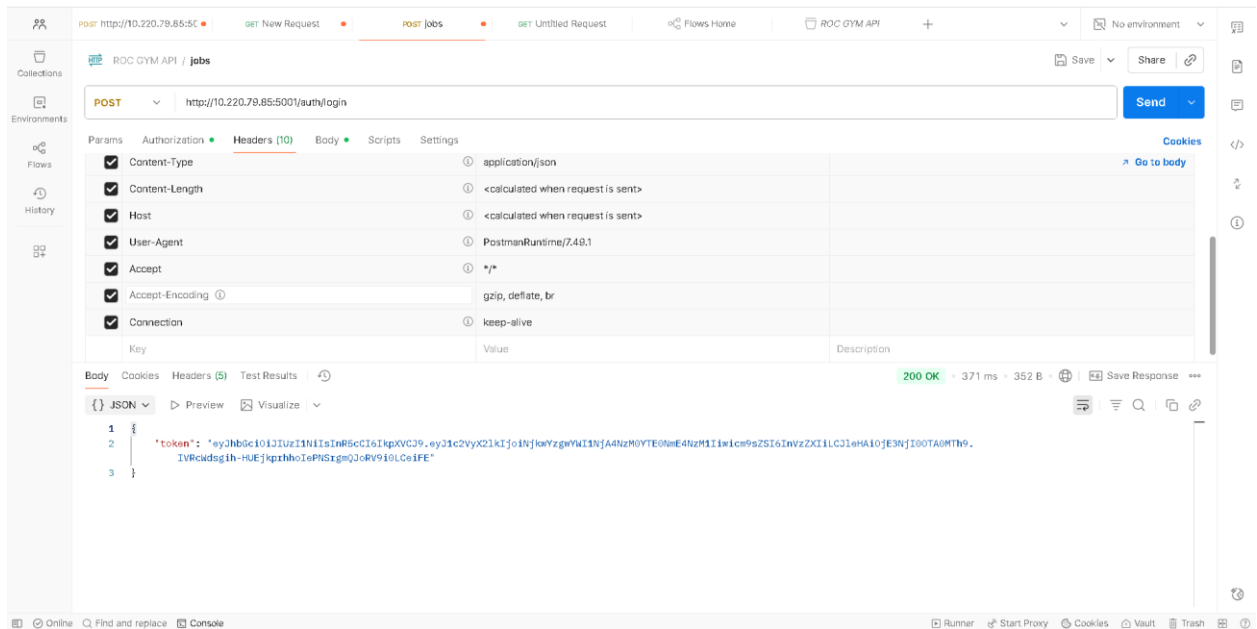
1) Test Protected Route Without Token



Expected Result:

- **Status: 401 Unauthorized**
- **Response message: something like "Missing or invalid token".**

2. Test Protected Route With Token



Expected Result:

- **Status:** 200 OK
- **Response:** List of jobs (or data relevant to the route)
- **Purpose:** Obtain and authentication token

Dataset (ROC Gym)

1. Dataset Overview

The job listings and employee management system at ROC Gym are the main focus of this project. It includes comprehensive details about job openings, personnel information, and activities associated with the job at ROC Gym. The dataset is perfect for CRUD (Create, Read, Update, Delete) operations because it is made to manage a well-defined and structured data model.

A MongoDB database will house the data in JSON format, allowing for scalable future growth, flexible document querying, and effective data storage.

Key Attributes:

- **Company_name:** ROC Gym
- **title:** Job title (Fitness Trainer, Front Desk Executive, Nutrition Specialist)
- **description:** Job responsibilities, qualifications, and experience required.
- **location:** Gym branch location (e.g., Downtown, Central, Westside).
- **Work_type:** Nature of employment (Full-time, Part-time, Contract).
- **Salary_range:** Expected monthly or annual pay range.
- **Requirements:** Skills or certifications required for the position.
- **Views:** Number of times the job listing has been viewed.

This dataset also integrates **trainer and member-related details** to enhance the ROC Gym's recruitment and management process.

3. Application Domain

Recruitment and Management of Fitness Centers is the application domain. By giving ROC Gym a single backend platform to handle employee hiring and member engagement, the system helps the gym achieve its operational objectives.

Key Features:

- Create and manage job listings for ROC Gym staff positions.
- Allow trainers and staff to apply directly for available positions.
- Enable job seekers to search for roles based on location, job type, or specialization.
- Record and track job listing view count to monitor popularity and visibility.
- Maintain up-to-date data for administrators and recruiters.

Target Users:

- **Administrators:** Oversee all employment postings, personnel files, and access restrictions.
- **Recruiters:** Advertise job vacancies and manage applications for particular gym locations.
- **Candidates/Trainers:** Look for and apply for positions that fit their areas of expertise.
- **Guest Users:** Look through available positions and basic ROC Gym information without logging in.

4. Implementation Plan

The backend will be developed using **MongoDB** as the database. RESTful APIs will be implemented to handle CRUD operations efficiently.

Endpoint	HTTP Method	Description
/jobs	POST	Create a new job listing
/jobs	GET	Retrieve all job listings
/jobs/<id>	GET	Retrieve a single job listing by ID
/jobs/<id>	PUT	Update an existing job listing
/jobs/<id>	DELETE	Delete a job listing
/jobs/<id>/apply	POST	Submit a job application

Endpoint	HTTP Method	Description
/members	GET	Retrieve gym member details (for admin view)

5. User Roles and Permissions

Role	Permissions
Admin	Full access: can create, update, and delete any job listing or member record.
Recruiter	Can post and manage job listings related to their branch only.
Trainer/Employee (Authenticated User)	Can view available jobs, update profiles, and apply for internal positions.
Guest User (Unauthenticated)	Can view job listings and general gym information but cannot apply or modify data

Endpoint	Method	Description
/jobs	POST	Create a new job listing
/jobs	GET	Get all job listings
/jobs/:id	GET	Get a specific job listing by ID
/jobs/:id	PUT	Update an existing job listing
/jobs/:id	DELETE	Delete a job listing
/jobs/:id/apply	POST	Submit a job application
/members	GET	Retrieve member details (Admin only)

7. Conclusion

The creation of a thorough backend system for ROC Gym's personnel and employment administration is presented in this project. The system guarantees effective data handling, scalability, and real-time updates by leveraging a structured dataset within MongoDB and a RESTful API design. While the addition of job postings and gym-specific features aligns the application with ROC Gym's operational needs in recruitment and management, role-based access improves security.

url: mongodb://localhost:27017/