

Full -stack Stratgies and Development



Job Listings Backend API Documentation (CW1)

Submitted By

Ram Naresh Yadav

Student ID: 10290454

Course Instructor:

SIHAN HAROON

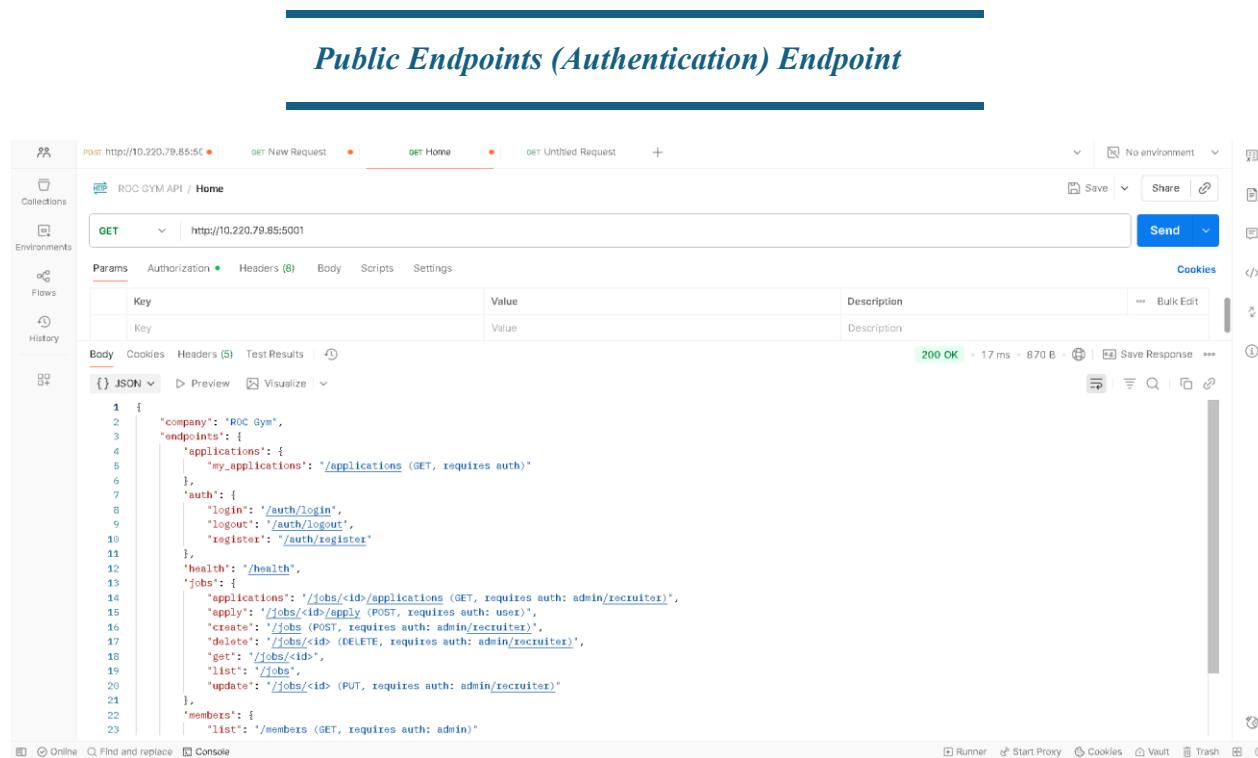
Project Overview

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



The screenshot shows the Postman application interface. A GET request is being tested against the URL `http://10.220.79.85:5001`. The response is successful (200 OK) with a response time of 1.7 ms and a body size of 870 B. The response body is a JSON object:

```
1 {  
2   "company": "ROC Gym",  
3   "endpoints": {  
4     "applications": {  
5       "my_applications": "/applications (GET, requires auth)"  
6     },  
7     "auth": {  
8       "login": "/auth/login",  
9       "logout": "/auth/logout",  
10      "register": "/auth/register"  
11    },  
12    "health": "/health",  
13    "jobs": {  
14      "applications": "/jobs/<id>/applications (GET, requires auth: admin/recruiter)",  
15      "apply": "/jobs/<id>/apply (POST, requires auth: user)",  
16      "create": "/jobs (POST, requires auth: admin/recruiter)",  
17      "delete": "/jobs/<id> (DELETE, requires auth: admin/recruiter)",  
18      "get": "/jobs/<id>",  
19      "list": "/jobs",  
20      "update": "/jobs/<id> (PUT, requires auth: admin/recruiter)"  
21    },  
22    "members": {  
23      "list": "/members (GET, requires auth: admin)"  
24    }  
25  }  
26}  
27
```

- **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 the Postman application interface. On the left, there's a sidebar with tabs for Collections, Environments, Flows, and History. The main area displays a collection named "ROC GYM API / health". A single request is listed: a GET request to "http://10.220.79.85:5001/health". The "Params" tab is selected, showing a key-value pair "Key": "Value". Below the request, the "Test Results" tab is active, displaying a JSON response:

```

1  {
2   "database": "connected",
3   "status": "healthy",
4   "timestamp": "2025-11-06T10:43:14.675166"
5 }

```

The status bar at the bottom indicates "200 OK" with a response time of 284 ms and a size of 250 B. Other buttons visible include "Save Response", "Runner", "Start Proxy", "Cookies", "Vault", "Trash", and a help icon.

Endpoint: Get/health

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

Authentication: Not required

3. Test: Get Jobs (Empty List):

The screenshot shows the Postman application interface. At the top, there are tabs for 'Post http://10.220.79.85:50' (highlighted), 'Get New Request', 'GET jobs' (selected), and 'GET Untitled Request'. On the right, there are buttons for 'Save', 'Share', and 'Send'. Below the tabs, the URL 'http://10.220.79.85:5001/jobs' is entered. The 'Params' tab is selected, showing a single parameter 'Key' with value 'Value'. To the right, there are tabs for 'Cookies', 'Headers (5)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is selected, showing a JSON response with two job objects. Each job object has fields like '_id', 'company_name', 'date_posted', 'description', 'location', 'posted_by', 'title', 'views', and 'work_type'. The response status is '200 OK' with a duration of '77 ms' and a size of '778 B'. At the bottom, there are buttons for 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and a refresh icon.

```
1 L
2 {
3     "_id": "69034ea56eef240f9521e557",
4     "company_name": "Acme Corp",
5     "date_posted": "Thu, 30 Oct 2025 11:40:21 GMT",
6     "description": "We are looking for a skilled developer to join our team.",
7     "location": "Remote",
8     "posted_by": "69034ada89c44e384c53720b",
9     "title": "Full Stack Developer",
10    "views": 0,
11    "work_type": "Full-time"
12 },
13 {
14     "_id": "69034ec06e6f240f9521e559",
15     "company_name": "Acme Corp",
16     "date_posted": "Thu, 30 Oct 2025 11:40:40 GMT",
17     "description": "We are looking for a skilled developer to join our team.",
18     "location": "Remote",
19     "posted_by": "69034ada89c44e384c53720b",
20     "title": "Full Stack Developer",
21     "views": 0,
22     "work_type": "Full-time"
23 }
```

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, Flows, and History. The main area has tabs for POST, GET, PUT, and DELETE, with POST selected. The URL is set to `http://10.220.79.85:5001/auth/register`. The request body is defined as follows:

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

The response status is `201 CREATED`, with a response time of 228 ms and a size of 250 B. The response body is shown as JSON:

```
1 {
2   "message": "User registered successfully",
3   "user_id": "690c80ab5e08734a146a8735"
4 }
```

At the bottom, there are buttons for Runner, Start Proxy, Cookies, Vault, Trash, and Help.

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 the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, Flows, and History. The main area has tabs for POST http://10.220.79.85:5001/auth/register, GET New Request, POST auth/register (which is selected), and GET Untitled Request. The right side has buttons for Save, Share, and Send. Below the tabs, there are sections for Params, Authorization, Headers (11), Body (selected), Scripts, and Settings. Under Body, there are options for none, form-data, x-www-form-urlencoded, raw (selected), binary, and GraphQL, with JSON chosen. The raw tab contains the following JSON payload:

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

Below the body, there are tabs for Body, Cookies, Headers (5), and Test Results. The Test Results tab shows a 409 CONFLICT status with a response time of 49 ms and a response size of 261 B. The response body is:

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

At the bottom, there are buttons for Runner, Start Proxy, Cookies, Vault, Trash, and Help.

Endpoint: Post/auth/login

Description: register with duplicate email

Response:

- Error (409) conflict

3. test_register_invalid_role:

The screenshot shows the Postman interface with a collection named "ROG GYM API". A POST request is being made to `http://10.220.79.85:5001/auth/register`. The request body contains the following JSON:

```
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**, with a duration of 7 ms and a size of 209 B. The response body is:

```
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 the Postman interface with a collection named "ROG GYM API". A POST request is being made to `http://10.220.79.85:5001/auth/login`. The request body contains the following JSON:

```
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**, with a duration of 224 ms and a size of 352 B. The response body is a large, encoded string:

```
E-KsEZow7qWJfXPejc1bYKQo53tBNCQg0ZRAe317Uy
```

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:

The screenshot shows the Postman interface with the following details:

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

```
1 {
2   "name": "John Doe",
3   "email": "john.doe@example.com",
4   "password": "@RM20580509ydv",
5   "role": "manager"
6 }
```

- Response Status:** 401 UNAUTHORIZED
- Response Body (JSON):**

```
1 {
2   "debug": {
3     "hashed_password": "b'$2b$12$d29h7sc.wzsNvktC9pnZK.Rvax91T0G8pZaPemT71zj2ItjXB9c'",
4     "password_type": "bytes"
5   },
6   "error": "Password is incorrect."
7 }
```

Checks:

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

5. test_login_nonexistent_user

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', 'Flows', and 'History'. The main area has tabs for 'POST http://10.220.79.85:501 / auth/login' (selected), 'GET New Request', 'POST auth/login', and 'GET Untitled Request'. Below the tabs, there's a 'Send' button. The request details show a 'POST' method to 'http://10.220.79.85:501/auth/login'. The 'Body' tab is selected, showing raw JSON data:

```
1 {
2   "name": "John Doe",
3   "email": "rohitmahaseth@gmail.com",
4   "password": "GRN20580509ydv",
5   "role": "manager"
6 }
```

The response details show a status of '401 UNAUTHORIZED' with a response time of '36 ms' and a size of '268 B'. The response body is:

```
{} JSON ▾ Preview ⚡ Debug with AI ▾
1 {
2   "debug": {
3     "email": "rohitmahaseth@gmail.com"
4   },
5   "error": "User with that email does not exist."
6 }
```

At the bottom, there are buttons for '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

The screenshot shows the Postman interface with the following details:

- Request URL:** POST http://10.220.79.85:5001/auth/register
- Body Content (JSON):**

```
1 {
2   "name": "Admin User",
3   "email": "admin@example.com",
4   "password": "Admin@123",
5   "role": "admin"
6 }
7
```
- Status:** 409 CONFLICT
- Response Body (JSON):**

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

Endpoint: Post/Auth/login

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

Error (409): conflict

For recruiter

The screenshot shows the Postman interface with a successful API call. The URL is `http://10.220.79.85:5001/auth/register`. The response status is `201 CREATED` with a response time of 215 ms and a size of 250 B. The response body is:

```
{ "message": "User registered successfully", "user_id": "690c887e5600734a146a873d" }
```

Endpoint: Post/Auth/Login

Description:

FOR register USER

The screenshot shows the Postman interface with a successful API call. The URL is `http://10.220.79.85:5001/auth/register`. The response status is `201 CREATED` with a response time of 228 ms and a size of 250 B. The response body is:

```
{ "message": "User registered successfully", "user_id": "690c89cc5608734a146a8740" }
```

B. Login & store tokens (3 requests)

1). Login & store tokens (admin)

The screenshot shows the Postman application interface. A POST request is being made to `http://10.220.79.85:5001/auth/login`. The request body is a JSON object:

```
1 {
2   "name": "Admin User",
3   "email": "admin@example.com",
4   "password": "password123",
5   "role": "admin"
6 }
7 
```

The response status is **200 OK**, with a response time of 223 ms and a size of 354 B. The response body is a JSON object containing a token:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9,ey31c2VyX2lkIjoiNjkwODk2YWEzYWI3ODZlYjQzZjhjMmM2Iiwicm9sZSI6ImFkbWluIiwiZXhwIjoxNzYyNDMzMjcxfo,
3   "FUjP4MA0MeYSGd7hfR319F_sR_hQhgDlnre01500PU"
} 
```

2. Login & store tokens (user)

The screenshot shows the Postman interface with the following details:

- Request URL:** http://10.220.79.85:501/auth/login
- Method:** POST
- Body Content:**

```
1 {
2   "name": "Regular User",
3   "email": "user@example.com",
4   "password": "password123",
5   "role": "user"
6 }
```
- Response Status:** 200 OK
- Response Headers:** 396 ms, 352 B
- Response Body:**

```
{}
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiNjkwM2QzZDc4NmRhODg1YThlNzg1Yjg1Iiwicm9sZSI6InVzZXIiLCJleHAiOiJE3NjI0MzQxOTN9.kuvQz1jCJI8IAxv0vY4vQjMFtFsVnXijvBL9TN2GB"
}
```

3. Login & store tokens (recruiter)

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

```

1 {
2   "name": "Recruiter User",
3   "email": "recruiter@example.com",
4   "password": "Recruiter@123",
5   "role": "recruiter"
6 }
7
  
```

The response status is `200 OK` with a `227 ms` duration and `359 B` size. The response body is a JSON token:

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCIkIkpXVCj9.eyJfc2Vx2IkIjoInjkYzg4N2U1NjA4NzMoYTE0NlE4NzNkIiicm9sZSI6InJlY3J1aXRLciIsImV4cCI6MTc2MjQzNDAwMH0.767FdF82G5y7GQtElAJg2QtEZayRwIKonM3ndYL2w"
3 }
  
```

Job Management

2. test_create_job_as_admin:

Postman screenshot showing a successful GET request to `http://10.220.79.85:5001/jobs`. The request includes the following headers:

- Accept: */*
- Accept-Encoding: gzip, deflate, br
- Connection: keep-alive
- Content-Type: application/json
- Authorization: eyJhbGciOiJIUzI1NiIsInR5cCIkIkpXVCj9.eyJfc2Vx2IkIjoInjkYzg4N2U1NjA4NzMoYTE0NlE4NzNkIiicm9sZSI6InJlY3J1aXRLciIsImV4cCI6MTc2MjQzNDAwMH0.767FdF82G5y7GQtElAJg2QtEZayRwIKonM3ndYL2w

The response status is `200 OK` with a `35 ms` duration and `778 B` size. The response body is a JSON array of two job posts:

```

1 [
2   {
3     "_id": "69034e066e6f240f9521e557",
4     "company_name": "Acme Corp",
5     "date_posted": "Thu, 30 Oct 2025 11:40:21 GMT",
6     "description": "We are looking for a skilled developer to join our team.",
7     "location": "Remote",
8     "posted_by": "69034ada09c44e384c53720b",
9     "title": "Full Stack Developer",
10    "views": 0,
11    "work_type": "Full-time"
12  },
13  {
14    "_id": "69034ec066e6f240f9521e559",
15    "company_name": "Acme Corp",
16    "date_posted": "Thu, 30 Oct 2025 11:40:48 GMT",
17    "description": "We are looking for a skilled developer to join our team.",
18    "location": "Remote"
  }
  
```

Features

1) Test Protected Route Without Token

The screenshot shows the Postman interface with a failed API request. The request URL is `http://10.220.79.85:5001/jobs`. The response status is `401 UNAUTHORIZED`, and the response body contains the message `'message': 'Token is invalid!'`.

Expected Result:

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

2. Test Protected Route With Token

The screenshot shows the Postman interface with a successful API request. The request URL is `http://10.220.79.85:5001/auth/login`. The response status is `200 OK`, and the response body contains a JSON object with a `token` key.

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJsb2dpbiIsInN1YnNwYWNpbmc6MTIwMjAxMjEzMSIsImlhdCI6MTYxNjQyOTk4fQ.4V8cldsgih-HUEjkprhhoIePNsrgmQJoRv9i0LceiFE"
3 }

```

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/