First: Unit test cases for major functionalities

Test ID	Feature	Description	Input	Expected Output
TC01	Login	Valid student login	Email + password	200 OK + JWT token + user object
TC02	Register	New user registration	Name, email, password, role	201 Created + user object
TC03	Resume Upload	Upload valid PDF resume as student	JWT + PDF file	200 OK + analysis result
TC04	Resume Upload	Upload resume as mentor (unauthorized)	JWT for mentor + file	403 Forbidden
TC05	Get All Users	Admin fetching user list	Admin JWT	200 OK + user list
TC06	Delete User	Admin deletes user by ID	Admin JWT + User ID	200 OK + confirmation message
TC07	Unauthorized Route	Accessing protected route without JWT	None	401 Unauthorized
TC08	Invalid Login	Wrong password	Correct email + wrong password	400 Bad Request
TC09	Duplicate Register	Registering with existing email	Email already in DB	400 Bad Request
TC10	Resume Upload	Uploading DOCX resume	JWT + DOCX file	200 OK + analysis result

---

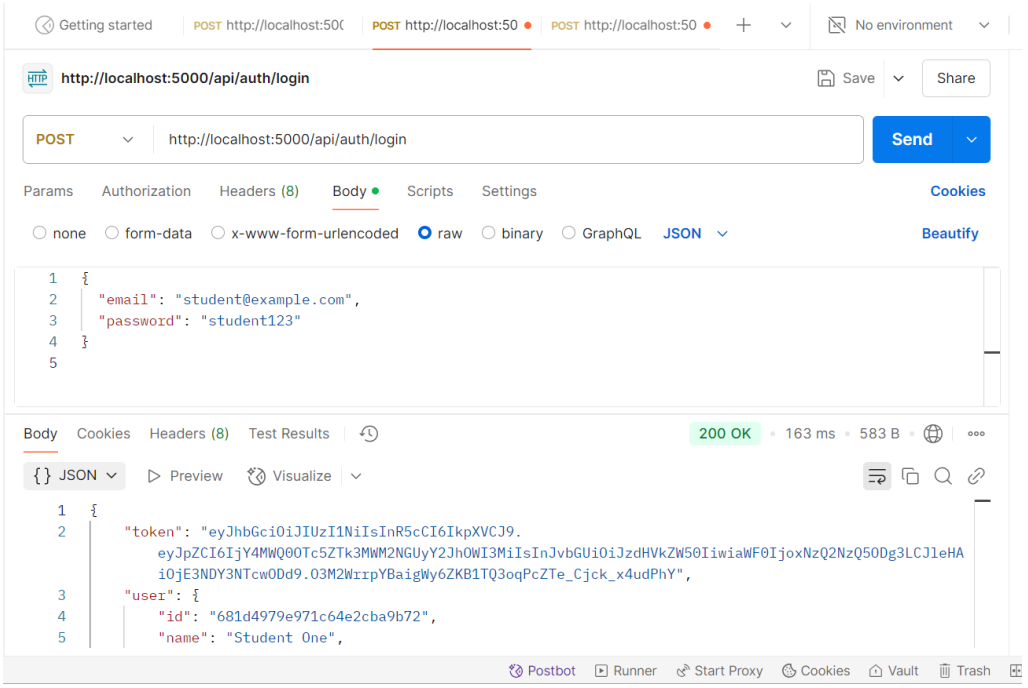
Second: test execution results

Test Case ID: TC01

- **Feature:** Login
- **Input:**

```
{
  "email": "student@example.com",
  "password": "student123"
}
```

- **Expected Output:** 200 OK with JWT token and user object
- **Actual Result:** ✅ Passed — received valid token and user details
- **Screenshot:**

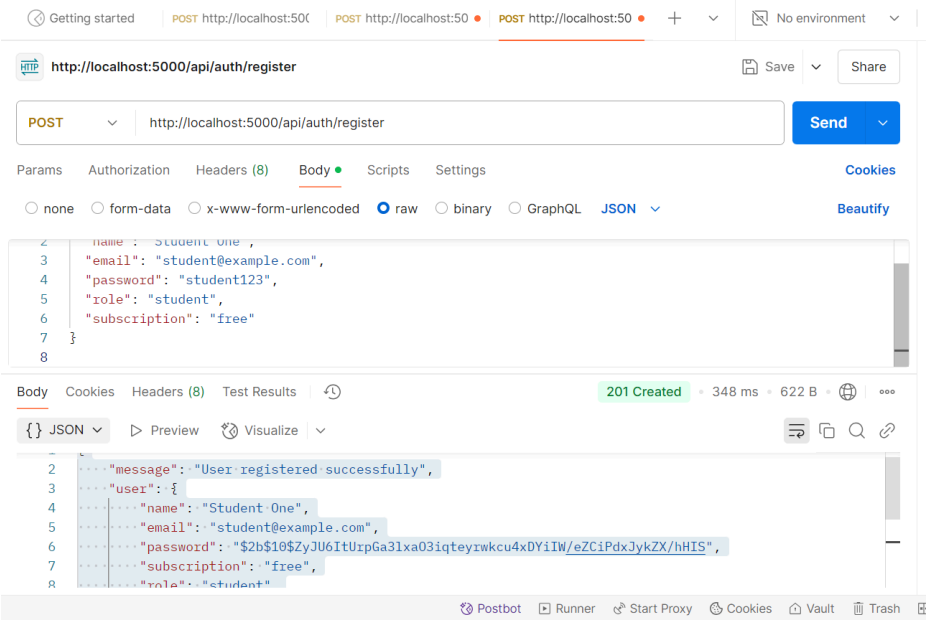


Test Case ID: TC02

- **Feature:** Register
- **Input:**

```
{
  "name": "Student One",
  "email": "student@example.com",
  "password": "student123",
  "role": "student",
  "subscription": "free"
}
```

- **Expected Output:** 201 Created with user object
- **Actual Result:** ✅ Passed — user created successfully
- **Screenshot:**

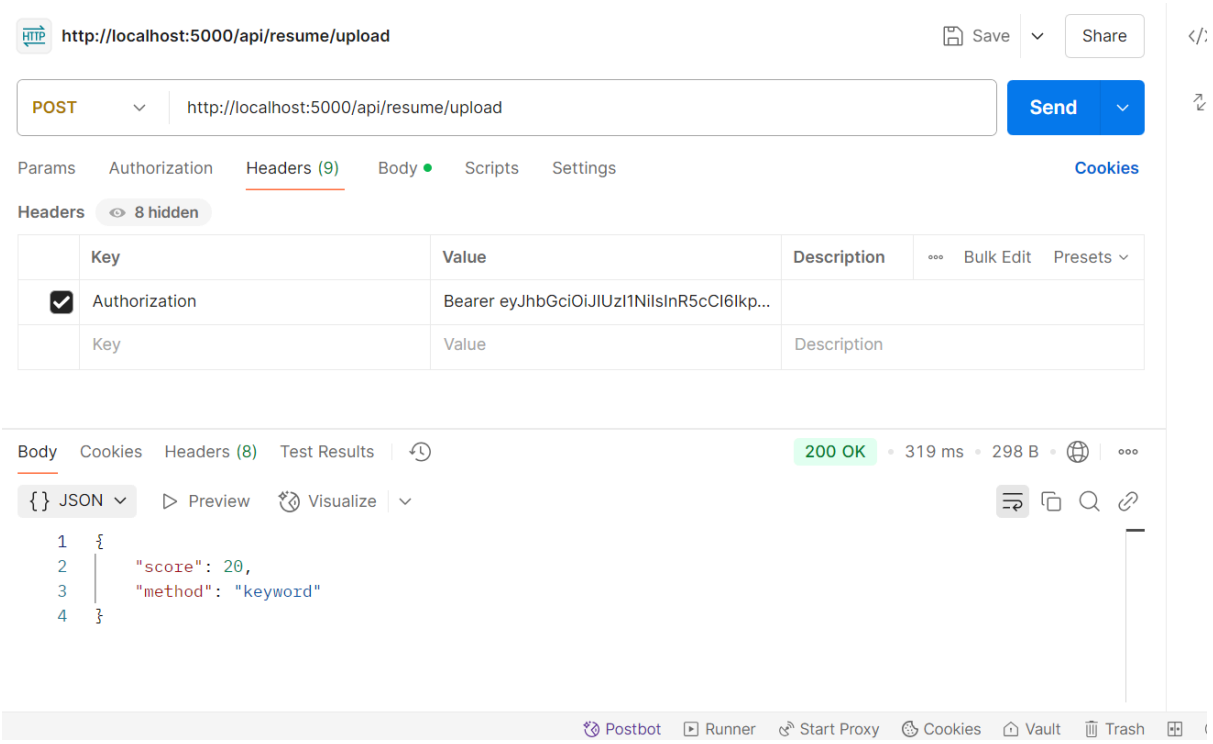


Test Case ID: TC03

- **Feature:** Resume Upload
- **Input:**

  Authorization: Bearer token (student)
  File: PDF or DOCX via `resume` form-data field

- **Expected Output:** 200 OK with resume analysis result
- **Actual Result:** ✅ Passed — analysis result and notification received
- **Screenshot:**

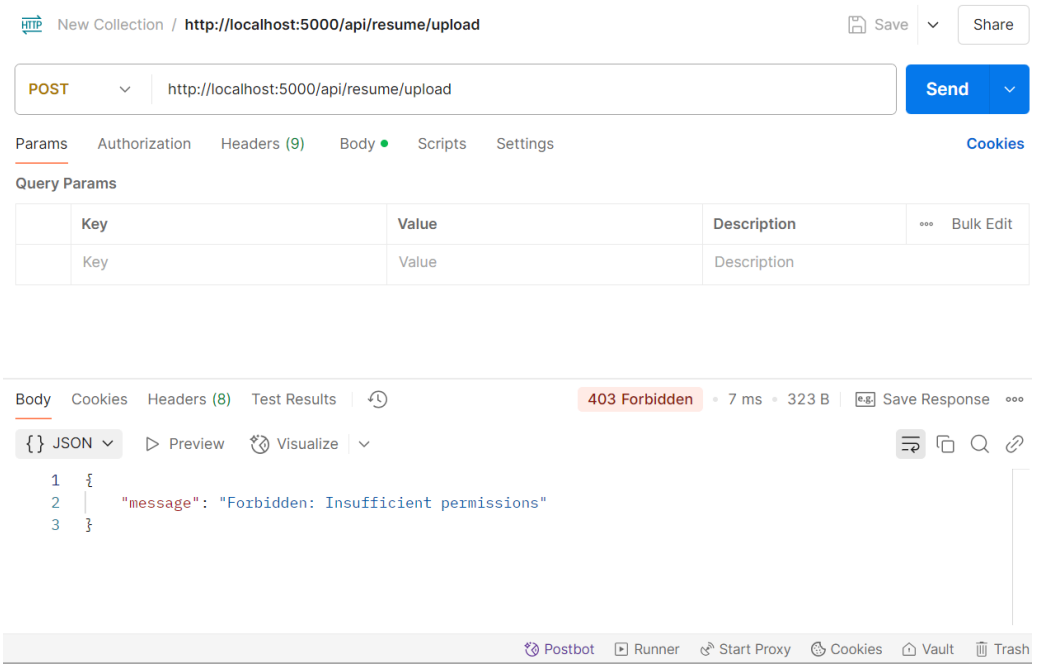


Test Case ID: TC04

- **Feature:** Resume Upload by Invalid Role
- **Input:**

  JWT from mentor account
  Valid resume file

- **Expected Output:** 403 Forbidden
- **Actual Result:** ✅ Passed — access correctly blocked for mentor
- **Screenshot:**



Test Case ID: TC05

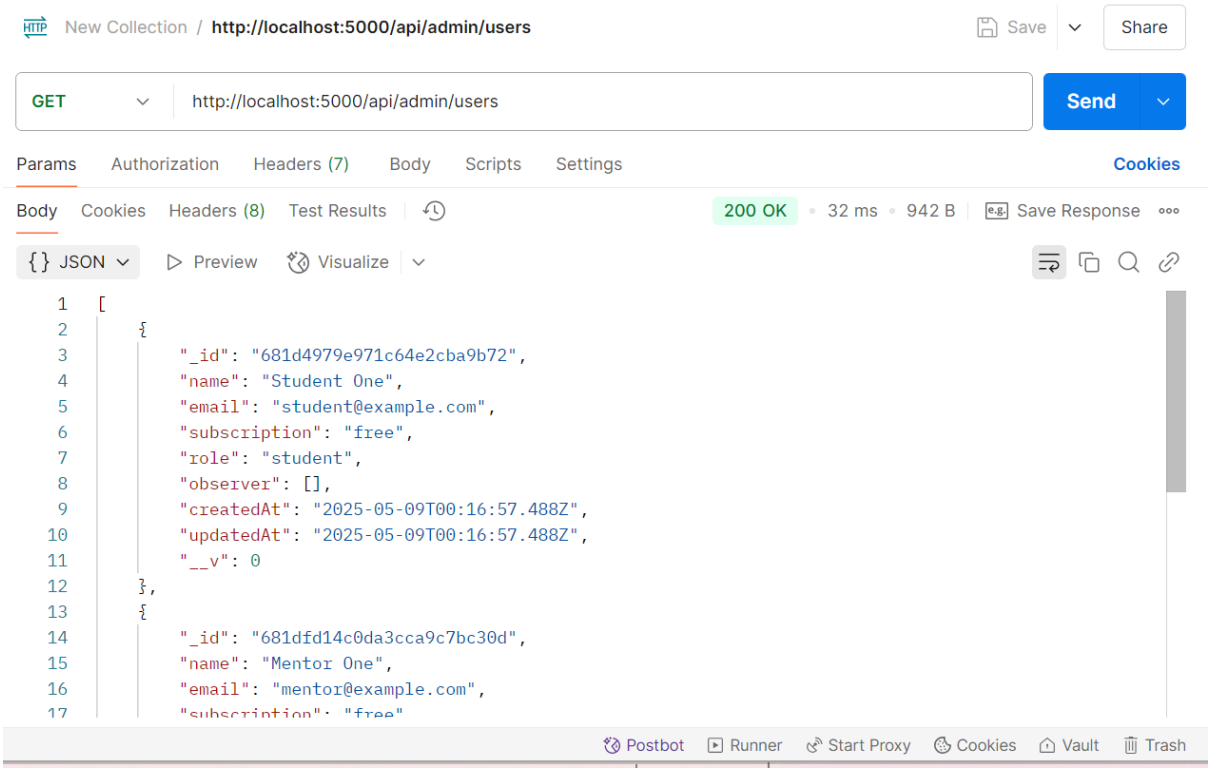
- **Feature:** Admin Fetch All Users
- **Input:**

  ```
  Authorization: Bearer <admin_token>
  ```

- **Expected Output:**

  `200 OK`
  JSON array of all registered users (excluding passwords)

- **Actual Result:** ✅ Passed — admin user successfully retrieved all users
- **Screenshot:**

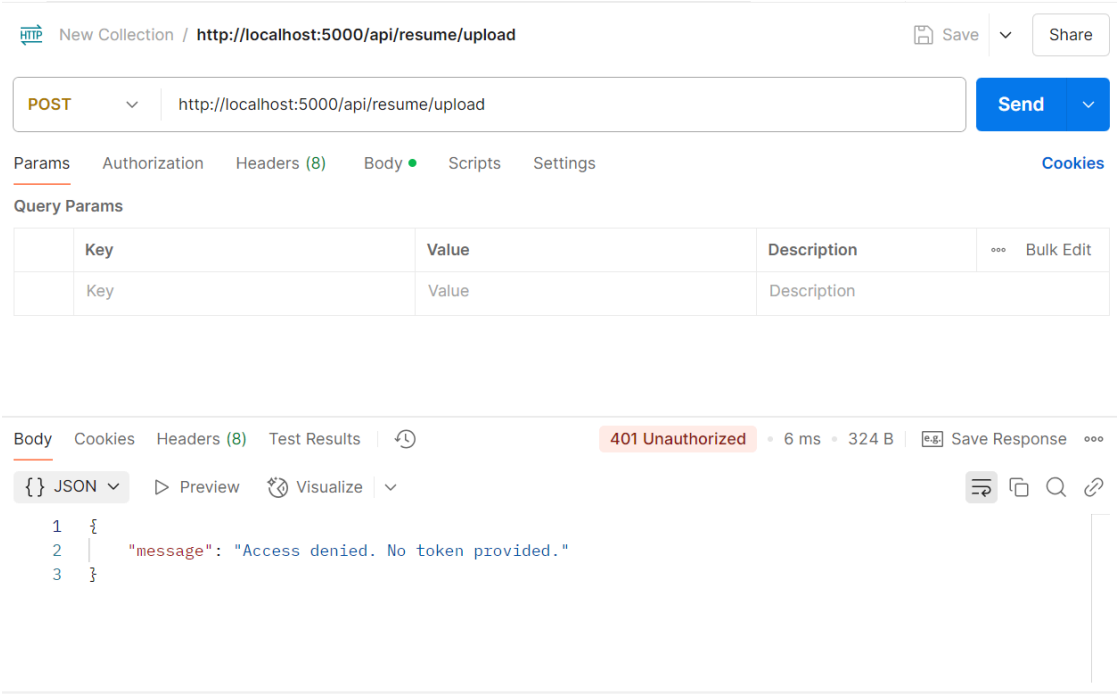


Test Case ID: TC07

- **Feature:** Unauthorized Resume Upload
- **Input:**

  No Authorization header
  Valid file

- **Expected Output:** 401 Unauthorized
- **Actual Result:** ✅ Passed — server blocked access due to missing token
- **Screenshot:**



Third: Code Review Feedback & Improvements

Throughout the development of the SkillSync backend, several improvements were identified and implemented based on ongoing code review and practical testing.

Separation of Concerns

Initially, logic for authentication and resume analysis was partially embedded in route handlers. After reviewing the structure, this was refactored to follow a clear MVC architecture:

- Business logic was moved to dedicated service files (e.g., `AuthService`)

- Controllers were streamlined to handle routing logic only

This improved code readability, maintainability, and testability.

Application of Design Patterns

Multiple design patterns were implemented to promote clean architecture and scalability:

- **Singleton Pattern** was applied in the `AuthService`, ensuring a single instance handles login and registration across the system.
- **Factory Pattern** was used for resume analysis logic, enabling dynamic selection of analyzers based on user subscription (e.g., keyword-based vs. AI-based).
- **Observer Pattern** was implemented in the notification system, allowing flexible support for multiple notification channels and better separation of notification logic.

These patterns improved modularity and made the system easier to extend and manage.

Security & Role Management

Role-based access control was structured using reusable middleware. Each route was protected based on the user's role, ensuring:

- Students could upload resumes
- Mentors could not perform restricted operations
- Admins had access to system-wide management features such as viewing and deleting users

The addition of a dedicated `admin` role ensured separation of concerns between users and platform management.

Testing Feedback

Early testing revealed missing middleware and runtime issues such as missing module imports (e.g., the `path` module in resume analysis). These were identified and fixed during review. Access control was confirmed to work correctly with appropriate 401 (unauthenticated) and 403 (unauthorized) responses.

Areas of Further Improvement

- Automating test cases using `jest` or `mocha` instead of manual Postman-based testing

- Adding input validation using `express-validator` to prevent malformed data
- Expanding admin capabilities to include dashboard statistics or bulk operations
- Enhancing security by reviewing and improving how sensitive data is handled

Summary

The SkillSync backend improved significantly during this iteration. Code was modularized, patterns were implemented effectively, and role-based access was enforced. These improvements fulfilled the Week 7 objectives for testability, maintainability, and scalability.