

COMP3123 – Assignment 1

Student Name: Rishamnoor Kaur

Student ID: 101508552

Course: COMP 3123 – Full Stack Development

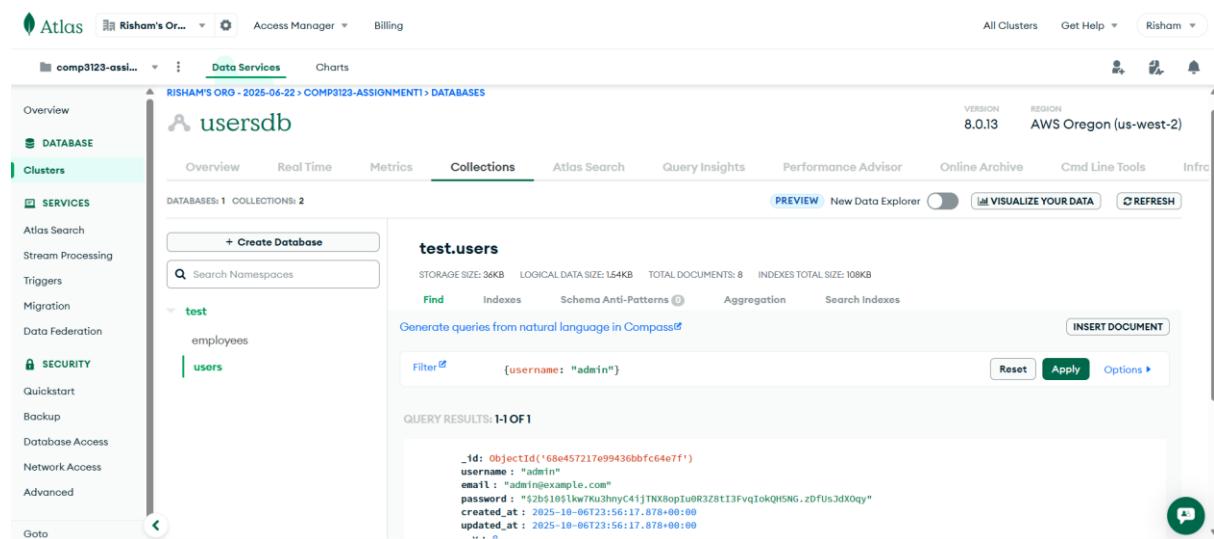
Instructor: Pritesh Patel

Sample User for Testing

```
{  
  "username": "admin",  
  "email": "admin@example.com",  
  "password": "admin123"  
}
```

MongoDB Console Screenshots

User Database



The screenshot shows the MongoDB Atlas interface for the 'usersdb' database. The left sidebar includes sections for Clusters, Services (Atlas Search, Stream Processing, Triggers, Migration, Data Federation), and Security (Quickstart, Backup, Database Access, Network Access, Advanced). The main area displays the 'test' database with two collections: 'employees' and 'users'. The 'users' collection is selected, showing its storage details (STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 154KB, TOTAL DOCUMENTS: 8, INDEXES TOTAL SIZE: 108KB) and various management tabs like Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A search bar at the top right allows for natural language queries. The bottom section shows the query results for a filter applied to the 'username: "admin"' field, displaying one document.

_id	username	email	password	created_at	updated_at
68e457217e99436bbfc64e7f	admin	admin@example.com	\$2b\$10\$1kw7kU3hnyC4ijTNX8opIuR3Z8tI3FvqIoQH5NG.zDFUs3dXoqy	2025-10-06T23:56:17.878+00:00	2025-10-06T23:56:17.878+00:00

Employee Database

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with various options like Overview, Clusters, Services, Security, and Migration. The main area shows a database named 'usersdb' with a collection named 'test.employees'. The collection details indicate it has 1 document and 240B logical size. A query builder is present, and the results show one document:

```
_id: ObjectId('58dc985b201422f989abee98')
first_name: "John"
last_name: "Doe"
email: "john.doe@example.com"
position: "Software Engineer"
salary: 10000
```

Testing Signup and Login

1 Signup Endpoint

✓ Test Case 1 — Successful Signup

The screenshot shows a Postman request to the '/register' endpoint. The request method is POST, and the URL is `http://localhost:5000/user/signup`. The body is set to raw JSON with the following payload:

```
{
  "username": "pritesh",
  "email": "pritesh@example.com",
  "password": "admin123"
}
```

The response status is 201 Created, and the response body is:

```
{
  "message": "User registered successfully",
  "user_id": "68e45b007e99436bbfc64e81"
}
```

✗ Test Case 2 — Missing Fields

The screenshot shows a POST request to `http://localhost:5000/user/signup`. The request body contains:

```
1 {
2   "username": "romeo",
3   "email": "romeo@example.com"
4 }
```

The response status is **400 Bad Request**, indicating a validation error. The JSON response body is:

```
1 {
2   "errors": [
3     {
4       "type": "field",
5       "msg": "Password is required",
6       "path": "password",
7       "location": "body"
8     },
9     {
10       "type": "field",
11       "msg": "Password must be at least 6 characters long",
12       "path": "password",
13       "location": "body"
14     }
15   ]
16 }
```

✗ Test Case 3 — Duplicate Email or Username

The screenshot shows a POST request to `http://localhost:5000/user/signup`. The request body contains:

```
1 {
2   "username": "pritesh",
3   "email": "pritesh@example.com",
4   "password": "admin123"
5 }
```

The response status is **400 Bad Request**, indicating a validation error. The JSON response body is:

```
1 {
2   "error": "Username or email already exists"
3 }
```

✗ Test Case 4 — Invalid Email Format

HTTP COMP3123-Assignment1 / register

POST http://localhost:5000/user/signup

Params Authorization Headers (9) **Body** Scripts Settings

None form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {
2 "username": "juliet",
3 "email": "notanemail",
4 "password": "risham123"
5 }

Save Share **Send**

Cookies Schema Beautify

Body Cookies Headers (8) Test Results

400 Bad Request 6 ms 390 B Save Response

{ } JSON Preview Debug with AI

1 {
2 "errors": [
3 {
4 "type": "field",
5 "value": "notanemail",
6 "msg": "Invalid email address",
7 "path": "email",
8 "location": "body"
9 }
10]

✗ Test Case 5 — Weak Password Length

HTTP COMP3123-Assignment1 / register

POST http://localhost:5000/user/signup

Params Authorization Headers (9) **Body** Scripts Settings

None form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {
2 "username": "juliet",
3 "email": "juliet@example.com",
4 "password": "pas"
5 }

Save Share **Send**

Cookies Schema Beautify

Body Cookies Headers (8) Test Results

400 Bad Request 6 ms 408 B Save Response

{ } JSON Preview Debug with AI

1 {
2 "errors": [
3 {
4 "type": "field",
5 "value": "pas",
6 "msg": "Password must be at least 6 characters long",
7 "path": "password",
8 "location": "body"
9 }
10]

2 Login Endpoint

✓ Test Case 1 — Login with Username

The screenshot shows a POST request to `http://localhost:5000/user/login`. The request body is JSON with fields `usernameOrEmail` and `password`. The response is a 200 OK status with a token and message.

```
POST http://localhost:5000/user/login
```

```
Body (raw) JSON
```

```
{ "usernameOrEmail": "sample00", "password": "password" }
```

```
200 OK
```

```
{"message": "Login successful", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ZTQ1YzM1N2U5OTQzNmJizmM2NGU4NSIsInVzZXJuYW1lIjoic2FtcGxlMDA1LCJpYXQiOjE3NTk3OTYzM0csImV4cCI6MTc1OTc5OTkwN30.MbxxySwEF4pUhJAwWhm_m33GI0wDVgnoep1AJ9jePaU4"}
```

✓ Test Case 2 — Login with Email

The screenshot shows a POST request to `http://localhost:5000/user/login`. The request body is JSON with fields `usernameOrEmail` and `password`. The response is a 200 OK status with a token and message.

```
POST http://localhost:5000/user/login
```

```
Body (raw) JSON
```

```
{ "usernameOrEmail": "sample00@example.com", "password": "password" }
```

```
200 OK
```

```
{"message": "Login successful", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ZTQ1YzM1N2U5OTQzNmJizmM2NGU4NSIsInVzZXJuYW1lIjoic2FtcGxlMDA1LCJpYXQiOjE3NTk3OTYzM0csImV4cCI6MTc1OTc5OTkwN30.rEN6UaPPw179b-vvVxCBmjXRHfNyHzbwRs7AnPpM8"}
```

✗ Test Case 3 — User Not Found

HTTP COMP3123-Assignment1 / Login

POST http://localhost:5000/user/login

Params Authorization Headers (9) Body Scripts Settings Cookies Schema Beautify

Body Cookies Headers (8) Test Results

404 Not Found 76 ms 300 B Save Response

```
1 {  
2   "usernameOrEmail": "notexists@example.com",  
3   "password": "password"  
4 }
```

{ } JSON Preview Debug with AI

```
1 {  
2   "error": "User not found"  
3 }
```

✗ Test Case 4 — Invalid Password

HTTP COMP3123-Assignment1 / Login

POST http://localhost:5000/user/login

Params Authorization Headers (9) Body Scripts Settings Cookies Schema Beautify

Body Cookies Headers (8) Test Results

200 OK 131 ms 295 B Save Response

```
1 {  
2   "usernameOrEmail": "sample00@example.com",  
3   "password": "wrongPassword"  
4 }
```

{ } JSON Preview Visualize

```
1 {  
2   "error": "Invalid password"  
3 }
```

✗ Test Case 5 — Missing Fields

HTTP COMP3123-Assignment1 / Login

POST http://localhost:5000/user/login

Params Authorization Headers (9) Body Scripts Settings Cookies Schema Beautify

Body Cookies Headers (8) Test Results

400 Bad Request 7 ms 370 B Save Response

```
1 {  
2   "usernameOrEmail": "sample00@example.com"  
3 }
```

{ } JSON Preview Debug with AI

```
1 {  
2   "errors": [  
3     {  
4       "type": "field",  
5       "msg": "Password is required",  
6       "path": "password",  
7       "location": "body"  
8     }  
9   ]  
10 }
```

Testing Employee Routes

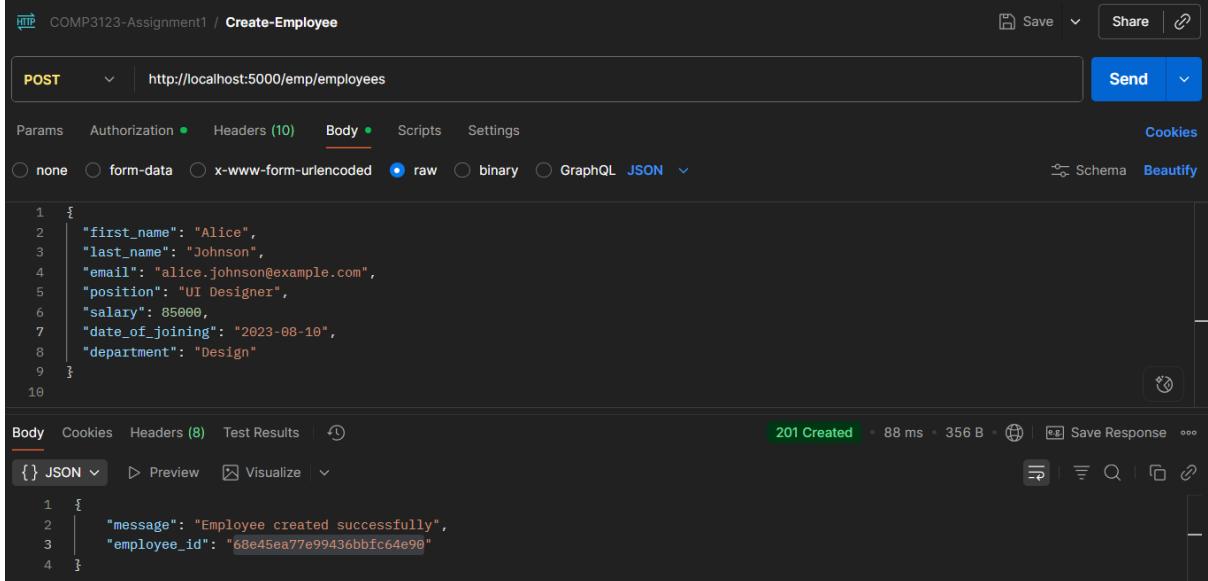
NOTE

All endpoints require authentication — so first **log in** using the /api/v1/user/login route and copy the **JWT token** from the response.

Then, add this header in every request:

Authorization: Bearer <your_token_here>

1 Create Employee



The screenshot shows the Postman application interface. At the top, it says "HTTP COMP3123-Assignment1 / Create-Employee". Below that, a "POST" method is selected with the URL "http://localhost:5000/emp/employees". The "Body" tab is active, showing raw JSON input:

```
1 {
2     "first_name": "Alice",
3     "last_name": "Johnson",
4     "email": "alice.johnson@example.com",
5     "position": "UI Designer",
6     "salary": 85000,
7     "date_of_joining": "2023-08-10",
8     "department": "Design"
9 }
```

Below the body, the response is shown under the "Test Results" tab. It indicates a "201 Created" status with a response time of 88 ms and a response size of 356 B. The response JSON is:

```
1 {
2     "message": "Employee created successfully",
3     "employee_id": "68e45ea77e99436bbfc64e90"
4 }
```

Validation Error Example

HTTP COMP3123-Assignment1 / Create-Employee

POST http://localhost:5000/emp/employees

Params Authorization Headers (10) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "first_name": "",
3   "email": "invalidemail",
4   "salary": "text"
5 }

```

Body Cookies Headers (8) Test Results Debug with AI

400 Bad Request 7 ms 1010 B Save Response

```

6   "msg": "First name is required",
7   "path": "first_name",
8   "location": "body"
9 },
10 {
11   "type": "field",
12   "msg": "Last name is required",
13   "path": "last_name",
14   "location": "body"
15 },
16 {
17   "type": "field",
18   "value": "invalidemail",
19   "msg": "Invalid email",
20   "path": "email",
21   "location": "body"
22 },
23 {
24   "type": "field",
25   "msg": "Position is required",
26   "path": "position",

```

✖ Invalid JWT Token Example

HTTP COMP3123-Assignment1 / Create-Employee

POST http://localhost:5000/emp/employees

Params Authorization Headers (10) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "first_name": "Alex",
3   "last_name": "Johnson",
4   "email": "alex.johnson@example.com",
5   "position": "UI Designer",
6   "salary": 85220,
7   "date_of_joining": "2023-08-10",
8   "department": "Design"

```

Body Cookies Headers (8) Test Results Debug with AI

401 Unauthorized 7 ms 302 B Save Response

```

1 {
2   "error": "Invalid token"
3 }

```

2 Get All Employees

HTTP COMP3123-Assignment1 / Get-Employees

GET http://localhost:5000/emp/employees

Params Authorization Headers (8) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

200 OK · 81 ms · 889 B · Save Response

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```

1 [
2   {
3     "_id": "68dc985b201422f989abee98",
4     "first_name": "John",
5     "last_name": "Doe",
6     "email": "john.doe@example.com",
7     "position": "Software Engineer",
8     "salary": 10000,
9     "date_of_joining": "2025-09-30T00:00:00.000Z",
10    "department": "Engineering",
11    "created_at": "2025-10-01T02:56:27.069Z",
12    "updated_at": "2025-10-06T23:36:13.153Z",
13    "__v": 0
14  },
15  {
16    "_id": "68e45ea77e99436bbfc64e90",
17    "first_name": "Alice",
18    "last_name": "Johnson",
19    "email": "alice.johnson@example.com",
20    "position": "UI Designer",
21    "salary": 85000,
22    "date_of_joining": "2023-08-10T00:00:00.000Z",
23    "department": "Design"
  }
]

```

✗ Error Case: Unauthorized (Missing JWT Token)

HTTP COMP3123-Assignment1 / Get-Employees

GET http://localhost:5000/emp/employees

Params Authorization Headers (8) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

401 Unauthorized · 5 ms · 302 B · Save Response

Body Cookies Headers (8) Test Results

{ } JSON Preview Debug with AI

```

1 {
2   "error": "Invalid token"
3 }

```

3 Get Employee by ID

GET → /employees/:id

HTTP Overview POST Login GET Get-Employees GET GetEmployeeById PUT Update-Employee DEL New Request COMP3123-Assignment1 No environment

HTTP COMP3123-Assignment1 / GetEmployeeById

GET http://localhost:5000/emp/employees/68e45ea77e99436bbfc64e90

Params Authorization Headers (8) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

200 OK · 81 ms · 577 B · Save Response

Body Cookies Headers (8) Test Results

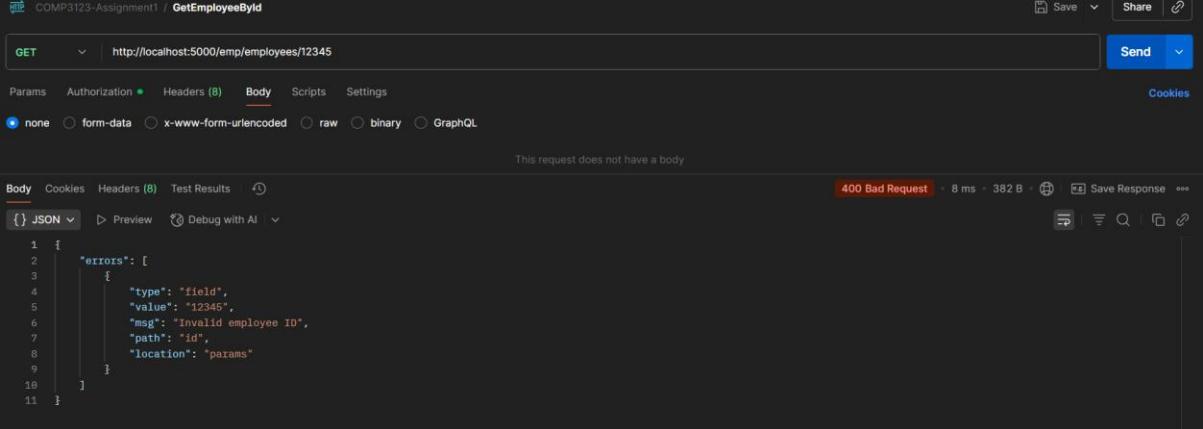
{ } JSON Preview Visualize

```

1 {
2   "_id": "68e45ea77e99436bbfc64e90",
3   "first_name": "Alice",
4   "last_name": "Johnson",
5   "email": "alice.johnson@example.com",
6   "position": "UI Designer",
7   "salary": 85000,
8   "date_of_joining": "2023-08-10T00:00:00.000Z",
9   "department": "Design",
10  "created_at": "2025-10-07T00:28:23.771Z",
11  "updated_at": "2025-10-07T00:28:23.771Z",
12  "__v": 0
13 }

```

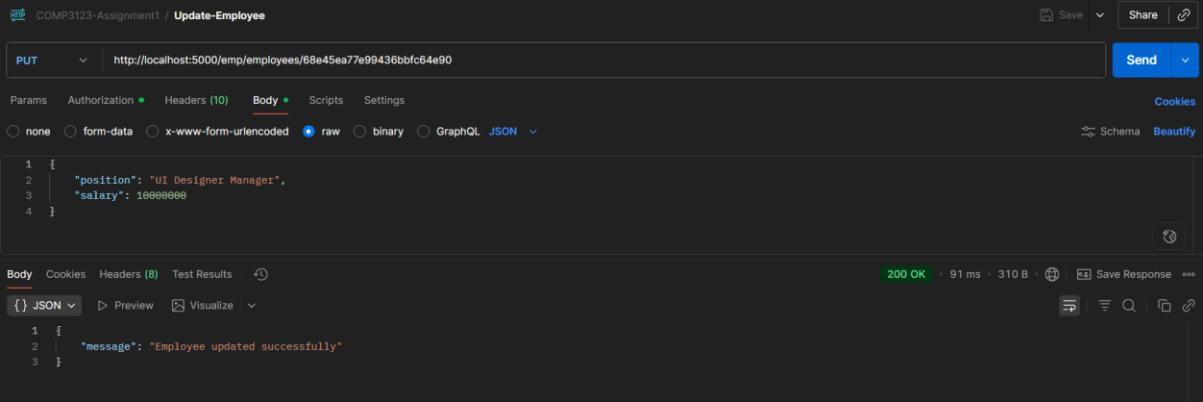
Error Case: Invalid ID



The screenshot shows a Postman request for `http://localhost:5000/emp/employees/12345`. The response is a **400 Bad Request** with the following JSON body:

```
1 {  
2   "errors": [  
3     {  
4       "type": "field",  
5       "value": "12345",  
6       "msg": "Invalid employee ID",  
7       "path": "id",  
8       "location": "params"  
9     }  
10   ]  
11 }
```

Update Employee



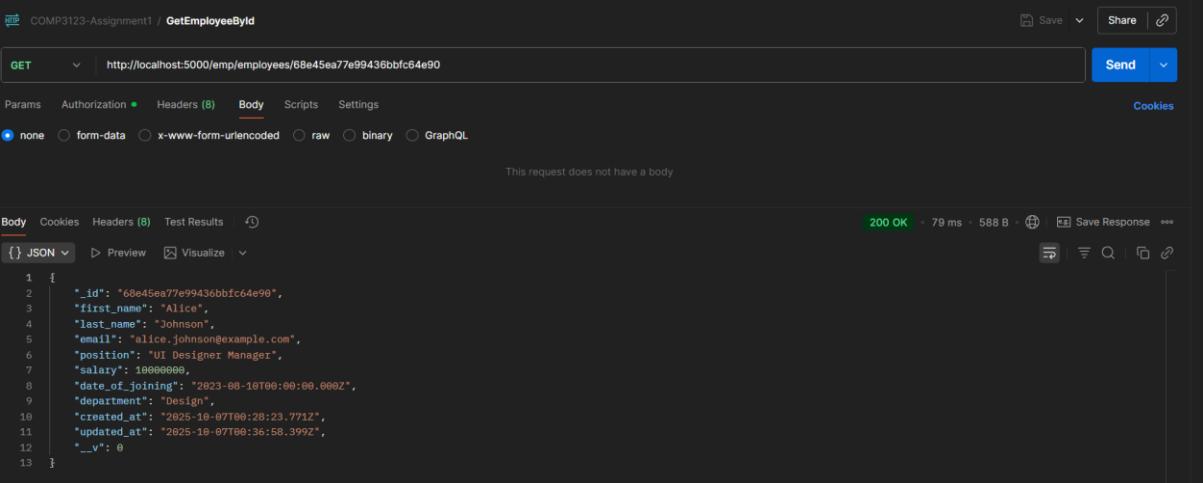
The screenshot shows a Postman request for `http://localhost:5000/emp/employees/68e45ea77e99436bbfc64e90` using the `PUT` method. The response is a **200 OK** with the following JSON body:

```
1 {  
2   "position": "UI Designer Manager",  
3   "salary": 10000000  
4 }
```

Body Cookies Headers (8) Test Results

```
1 {  
2   "message": "Employee updated successfully"  
3 }
```

As shown in Get Employee by ID:



The screenshot shows a Postman request for `http://localhost:5000/emp/employees/68e45ea77e99436bbfc64e90` using the `GET` method. The response is a **200 OK** with the following JSON body:

```
1 {  
2   "_id": "68e45ea77e99436bbfc64e90",  
3   "first_name": "Alice",  
4   "last_name": "Johnson",  
5   "email": "alice.johnson@example.com",  
6   "position": "UI Designer Manager",  
7   "salary": 10000000,  
8   "date_of_joining": "2023-08-10T00:00:00Z",  
9   "department": "Design",  
10  "created_at": "2025-10-07T00:28:23.771Z",  
11  "updated_at": "2025-10-07T00:36:58.399Z",  
12  "__v": 0  
13 }
```

Delete Employee

DELETE → /employees/:id

The screenshot shows a Postman request for a DELETE operation on the URL `http://localhost:5000/emp/employees/68dc985b201422f989abee98`. The request method is set to `DELETE`. The response status is `204 No Content`, with a duration of `94 ms` and a size of `208 B`. The body of the response is empty, containing only the number `1`.

✗ Error Case: Invalid ID

The screenshot shows a Postman request for a DELETE operation on the URL `http://localhost:5000/emp/employees/12345`. The response status is `400 Bad Request`, with a duration of `8 ms` and a size of `382 B`. The response body is a JSON object with an `errors` field:

```
1 {  
2   "errors": [  
3     {  
4       "type": "field",  
5       "value": "12345",  
6       "msg": "Invalid employee ID",  
7       "path": "id",  
8       "location": "params"  
9     }  
10   ]  
11 }
```

✗ Error Case: Unauthorized

The screenshot shows a Postman request for a DELETE operation on the URL `http://localhost:5000/emp/employees/12345`. The response status is `401 Unauthorized`, with a duration of `6 ms` and a size of `320 B`. The response body is a JSON object with an `error` field:

```
1 {  
2   "error": "Invalid JWT token, Unauthorized"  
3 }
```