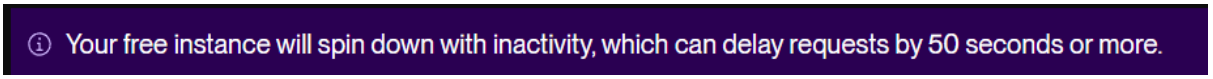
Documentation

(FastAPI + Pydantic + MongoDB)

🚨**Important Note:**🚨

We are using Render's free version, hence there may be a delay for requests once the server is inactive.



1. Introduction

This backend application is built using **FastAPI**, **Pydantic**, and **MongoDB**.
The main purpose of the backend is to **receive requests from the client**, **process the data**, **store or retrieve data from the database**, and **send proper responses back to the client**.

The application performs **CRUD operations**, which include:

- Create
- Read
- Update
- Delete

2. How Client Requests Reach the Backend

Request Flow

The client sends a request using an **API endpoint**
Example:
POST /students

1. Client requests
2. The request reaches the **FastAPI server**..
3. FastAPI checks:
    - The **URL (endpoint)**
    - The **HTTP method** (GET, POST, PUT, DELETE)
4. FastAPI matches the request with the correct **route function**.

Example

If the client sends:

GET /students

FastAPI understands:

> "The client wants to read all student data."

3. How the Backend Interacts with the Database

Database Used: MongoDB

- MongoDB stores data in **documents**
- Data is stored in **collections**
- Each student is stored as one document

Backend–Database Connection

1. When the backend server starts, it connects to **MongoDB** using a database connection file.
2. This connection stays active while the server is running.
3. Whenever data is needed, the backend uses this connection to:
    - Insert data
    - Read data
    - Update data
    - Delete data

Role of Pydantic

Before data is sent to MongoDB:

- **Pydantic validates the data**
- It checks:
    - Data type (number, string)
    - Required fields
- If data is invalid, it is rejected immediately

This ensures that **only correct data enters the database**.

4. How CRUD Operations Are Executed

CRUD operations are the core functionality of this backend.

4.1 Create Operation (CREATE)

Purpose:

To add new student data to the database.

Working Principle:

1. Client sends a **POST request** with student details.
2. FastAPI receives the request.
3. Pydantic checks if the data is valid.
4. If valid, backend sends the data to MongoDB.
5. MongoDB stores the data as a new document.

Result:

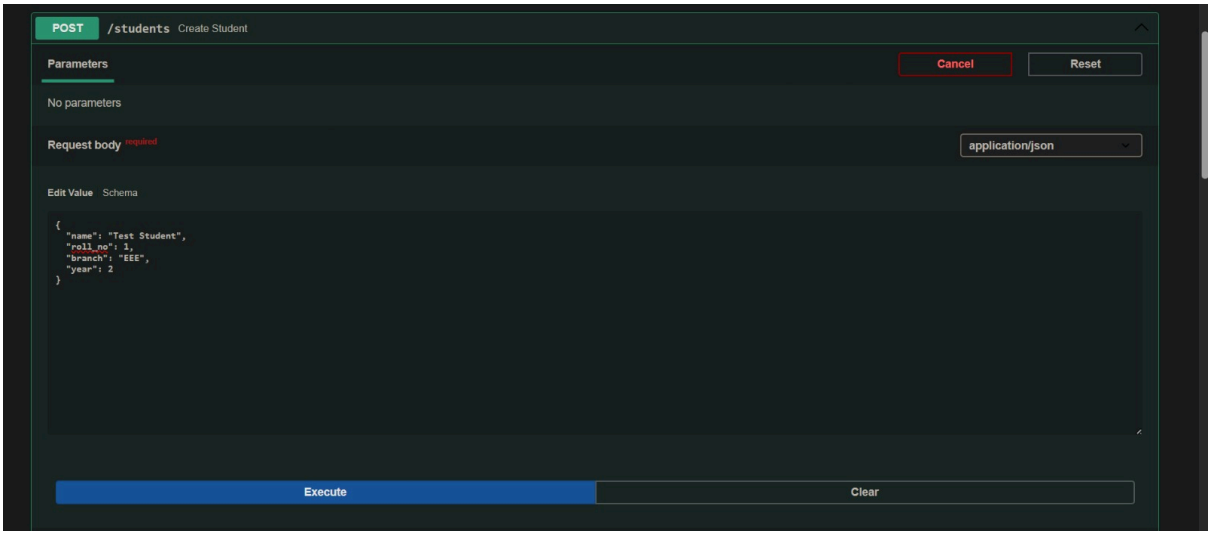
A new student record is created successfully.

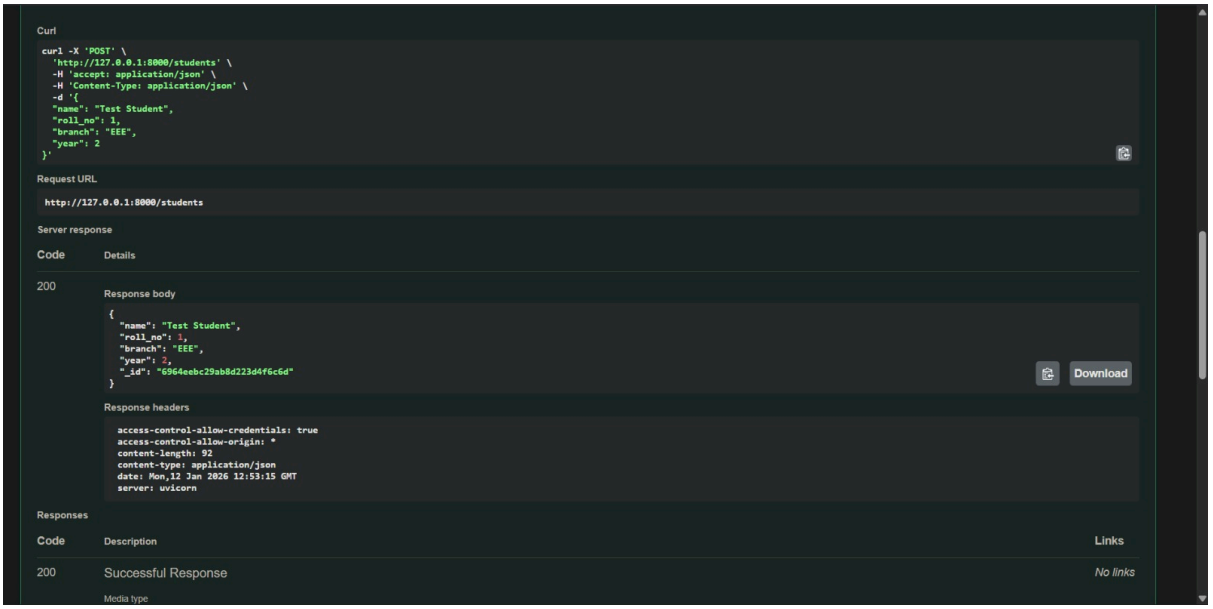
Testing:

As shown, for testing we made use of:

FastAPI Swagger UI (auto-generated API documentation)

MongoDB Atlas (database verification)





4.2 Read Operation (READ)

Purpose:
To fetch student data from the database.

There are two types:

- Read all students
- Read one student by ID

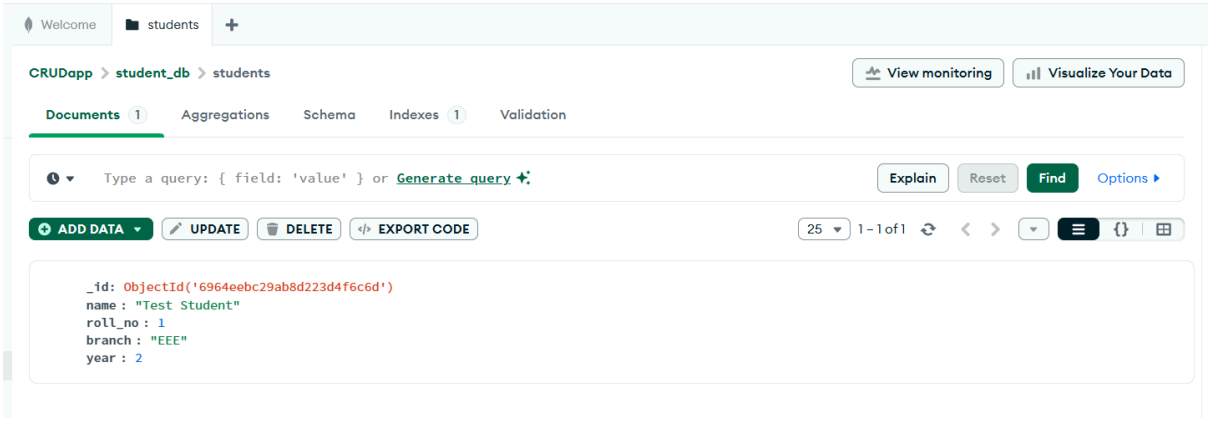
Working Principle:

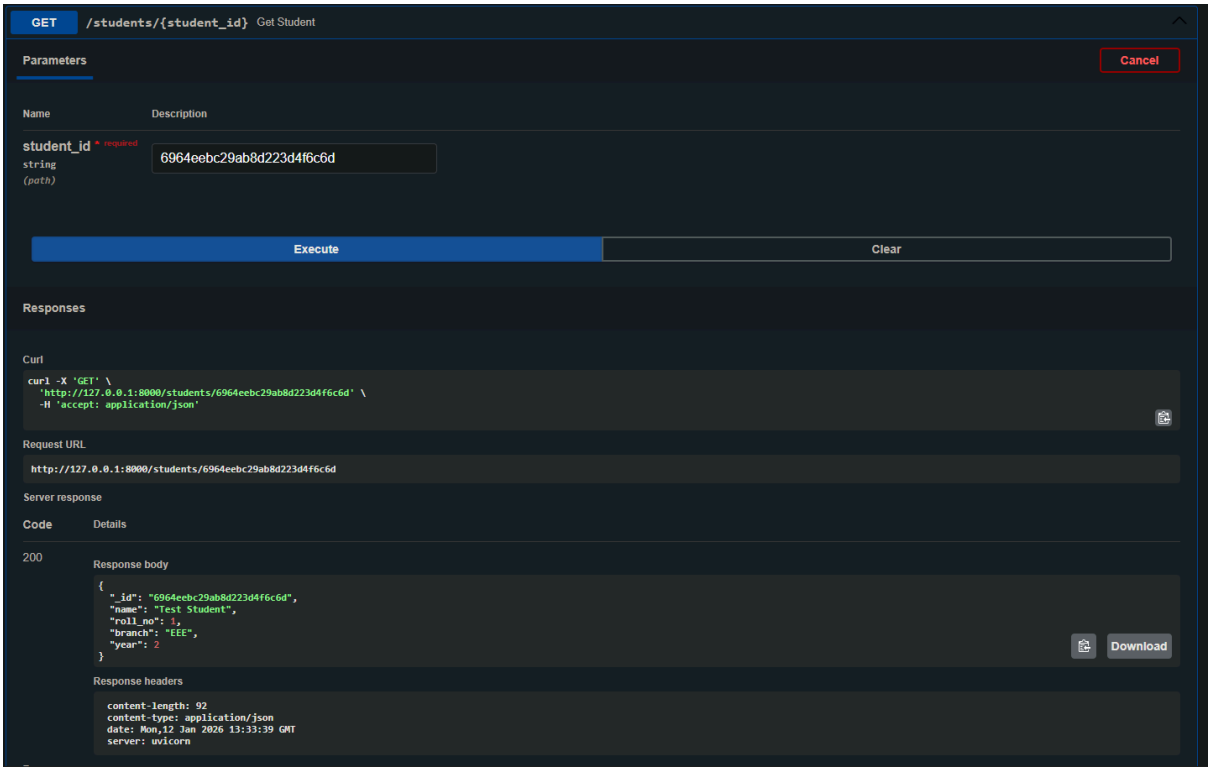
1. Client sends a **GET request**.
2. FastAPI receives the request.
3. Backend queries MongoDB.
4. MongoDB returns the requested data.
5. Backend sends the data back to the client.

Result:
Student data is displayed to the client.

Testing:

Below is the MongoDB Atlas entry for a test student, we can read details of a student using the id: ‘6964eebc29ab8d223d4f6c6d’





4.3 Update Operation (UPDATE)

Purpose:
To modify existing student data.

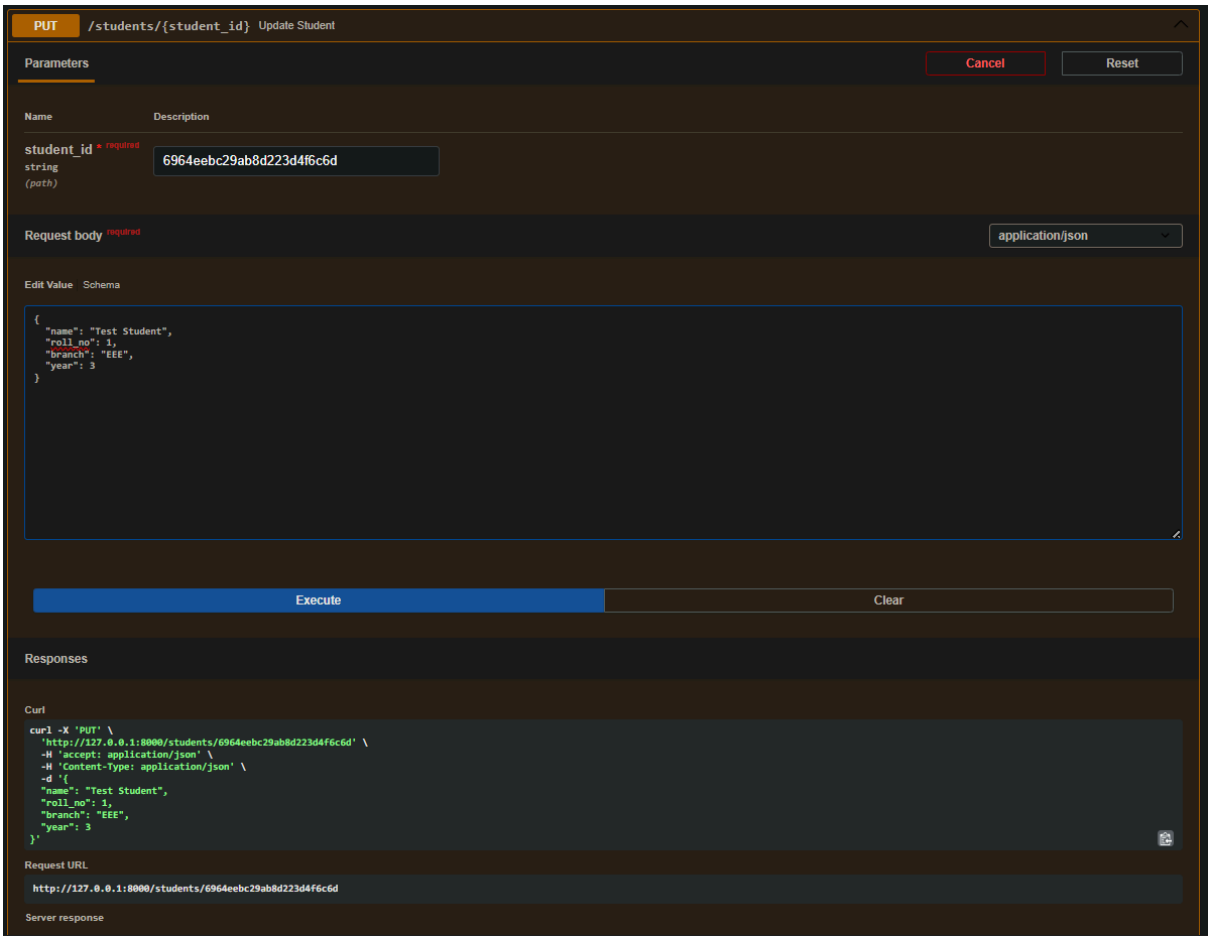
Working Principle:

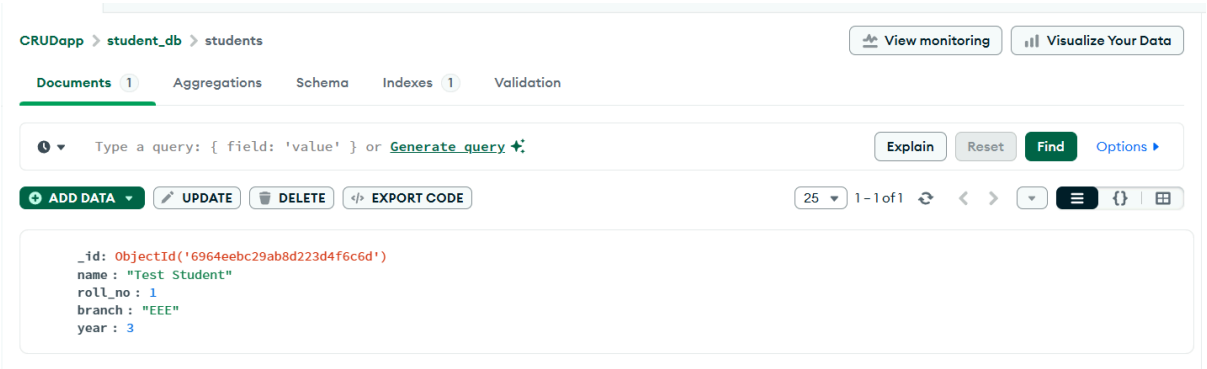
1. Client sends a **PUT request** with updated data.
2. Backend checks if the student ID exists.
3. Pydantic validates the new data.
4. Backend updates the data in MongoDB.
5. MongoDB confirms the update.

Result:
Student details are updated successfully.

Testing:

Altering of Test students year.





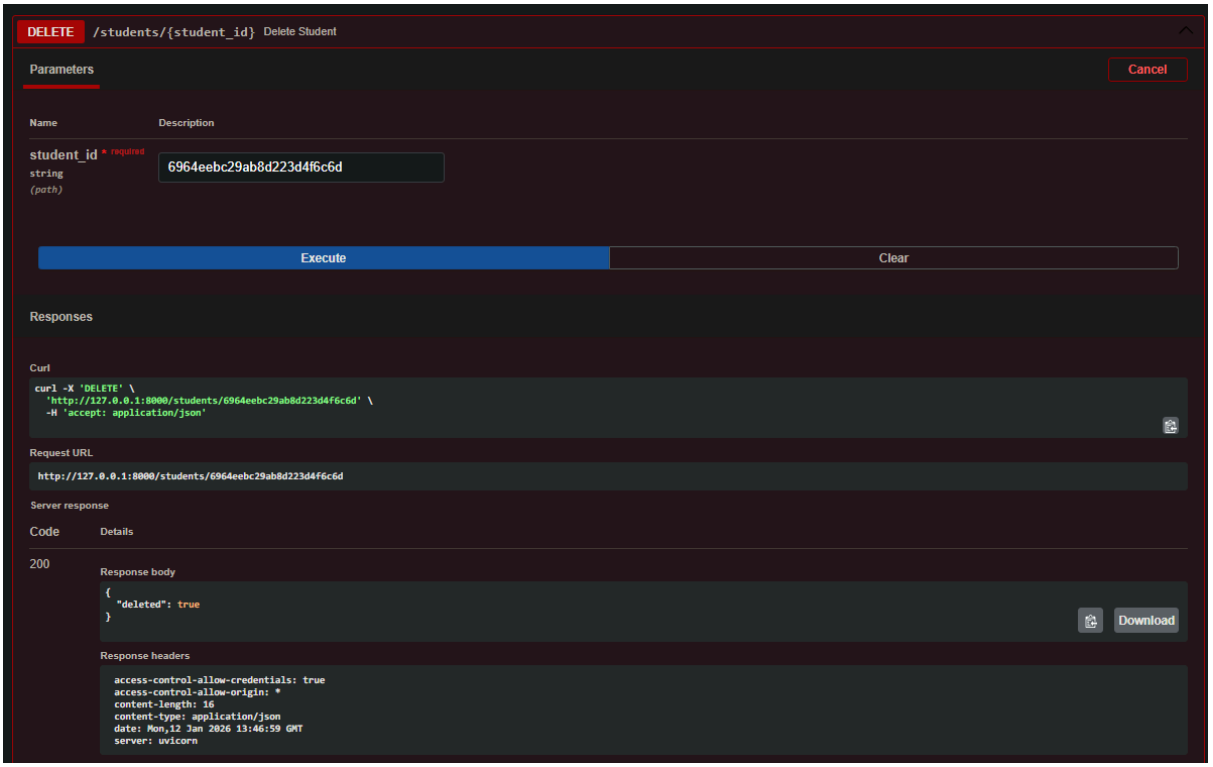
4.4 Delete Operation (DELETE)

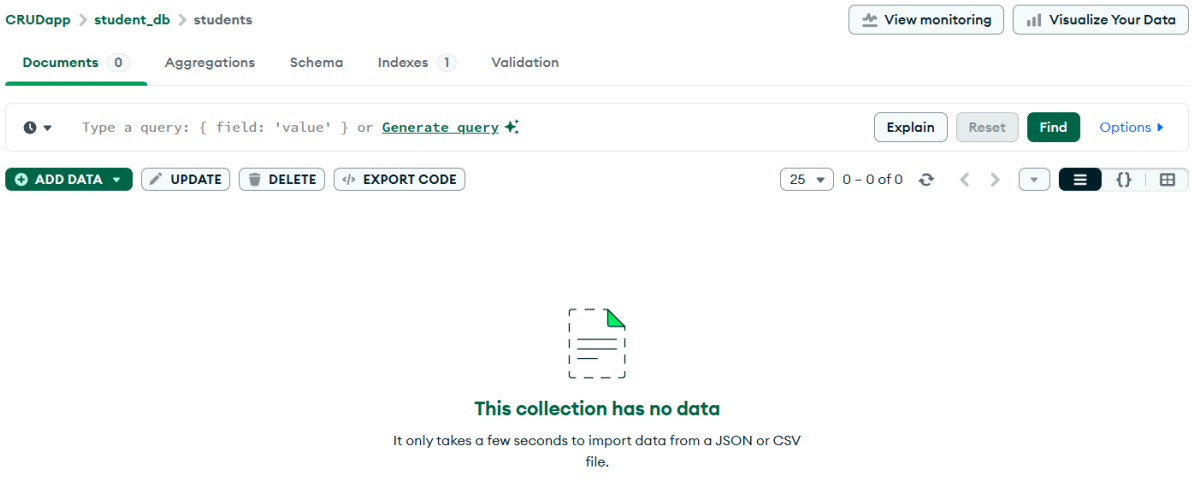
Purpose:
To remove a student record from the database.

Working Principle:

1. Client sends a **DELETE request** with a student ID.
2. Backend checks if the record exists.
3. Backend sends delete command to MongoDB.
4. MongoDB removes the document.

Result:
The student record is deleted permanently.





5. How Success and Error Responses Are Returned

Success Responses

When an operation is completed successfully:

- Backend sends a **success message**
- Data may also be returned (for read operations)

Example Success Response:

```
{
  "message": "Student added successfully"
}
```

Error Responses

Errors can occur due to:

- Missing fields
- Wrong data type
- Invalid ID
- Student not found
- Server issues

Error Handling Process:

1. FastAPI detects the error.
2. Backend sends an error message with status code.
3. Client receives clear information about the error.

Example Error Response:

```
{
  "error": "Student not found"
}
```

6. Database Schema

Database name: student_db
Collection name: students
Documentation structure:

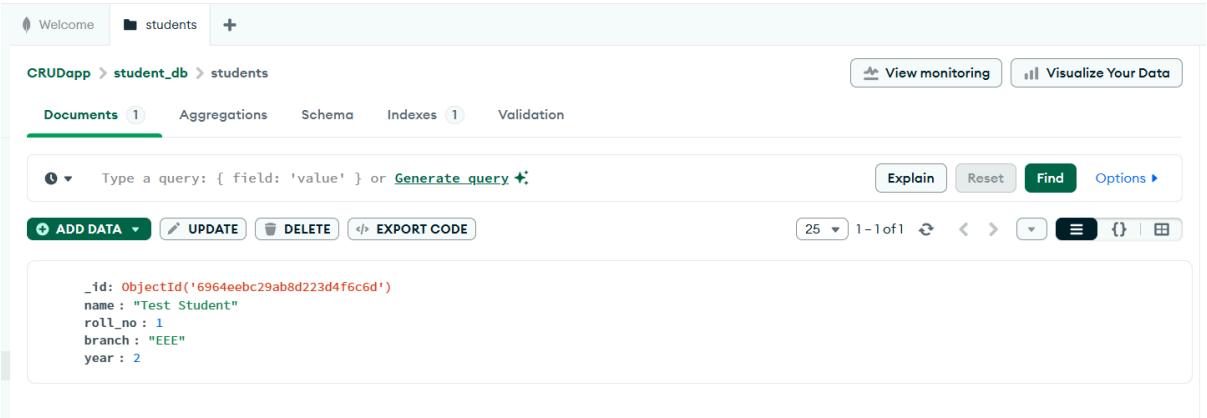
```
{
 "_id": "ObjectId",
  "name": "String",
  "roll_no": "Number",
  "branch": "String",
  "year": "Number"
```

}

Field descriptions:

Field name	Datatype	Description
id	ObjectId	Auto-generated unique identifier by MongoDB
name	String	Name of student
roll_no	Integer	Roll no. of student
branch	String	B.Tech Branch of student
year	Integer	Year of studying

Example:



7. API Documentation

https://crudapp-ez05.onrender.com/docs

Overall Working Flow (Summary)

1. Client sends a request
2. FastAPI receives and routes the request
3. Pydantic validates the data
4. Backend interacts with MongoDB
5. CRUD operation is performed
6. Success or error response is returned

Conclusion

The backend application works by efficiently handling client requests, validating data, performing CRUD operations using MongoDB, and returning meaningful responses. FastAPI ensures fast request handling, Pydantic ensures data correctness, and MongoDB provides flexible data storage.

This working principle makes the application **reliable**, **secure**, and **easy to maintain**.