**Web Engineering CCP Project Report: Backend API for Student Event Management System**

|  |  |
| --- | --- |
|  |  |
| **Student Name:** | Muhammad Saqib Khan (62612) |
| **Course:** | Web Engineering |
| **Program:** | BS (SE) |
| **Project:** | Complex Computing Problem (CCP) |
| **Submission Date:** | June 20, 2025 |

**1. Objective and Scope**

**1.1. Objective**

The primary objective of this project was to design, develop, and implement a complete, robust, and scalable backend solution for a Student Event Management System. The system was built as a RESTful API using modern technologies, including ASP.NET Core 9.0 and Entity Framework Core, to provide a centralized and efficient way to manage events, participants, and feedback. The project emphasizes professional software development practices, including the use of Clean Architecture, version control with Git, and comprehensive API documentation.

**1.2. Scope**

The scope of the project encompasses the full lifecycle of event management from a backend perspective. The implemented features are:

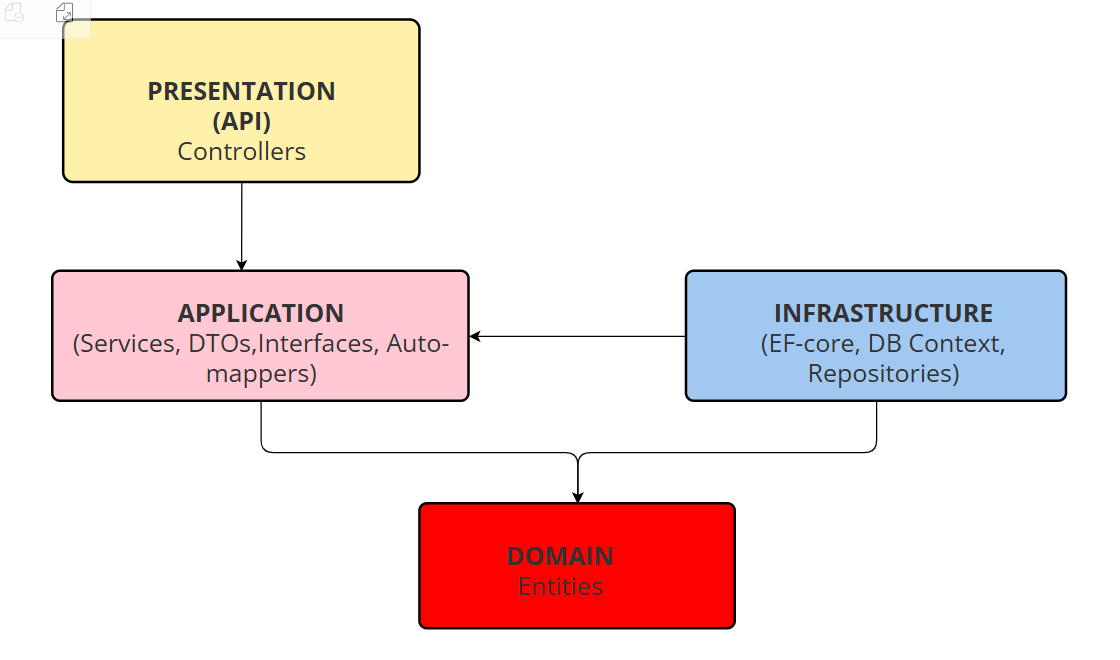
* **Full Event CRUD Functionality:** The API provides endpoints for Creating, Reading, Updating, and Deleting events.
* **Participant Registration:** A dedicated endpoint allows students to register for events. This feature correctly models the many-to-many relationship between students and events.
* **Feedback Submission System:** Students can submit feedback, consisting of a numerical rating and a text comment, for events they attended.
* **Business Logic and Validation:** The system enforces critical business rules, such as:
  + Preventing feedback submission for events that have not yet occurred.
  + Handling cases of duplicate registrations or feedback submissions.
  + Ensuring data integrity through validation and proper error handling.
* **Advanced Querying:** The API supports dynamic searching of events by name or venue and allows sorting of results by date, name, or venue.
* **Layered Architecture:** The solution strictly adheres to Clean Architecture principles, ensuring a clear separation of concerns, high maintainability, and scalability.

**2. System Architecture**

The application is built upon a layered, Clean Architecture pattern. This design ensures that the core business logic (Domain) is independent of external frameworks and technologies, making the system easier to test, maintain, and evolve.

**2.1. Architecture Diagram**

The following diagram illustrates the dependency flow between the layers. The arrows point towards the layer being depended upon, signifying that dependencies flow inwards.

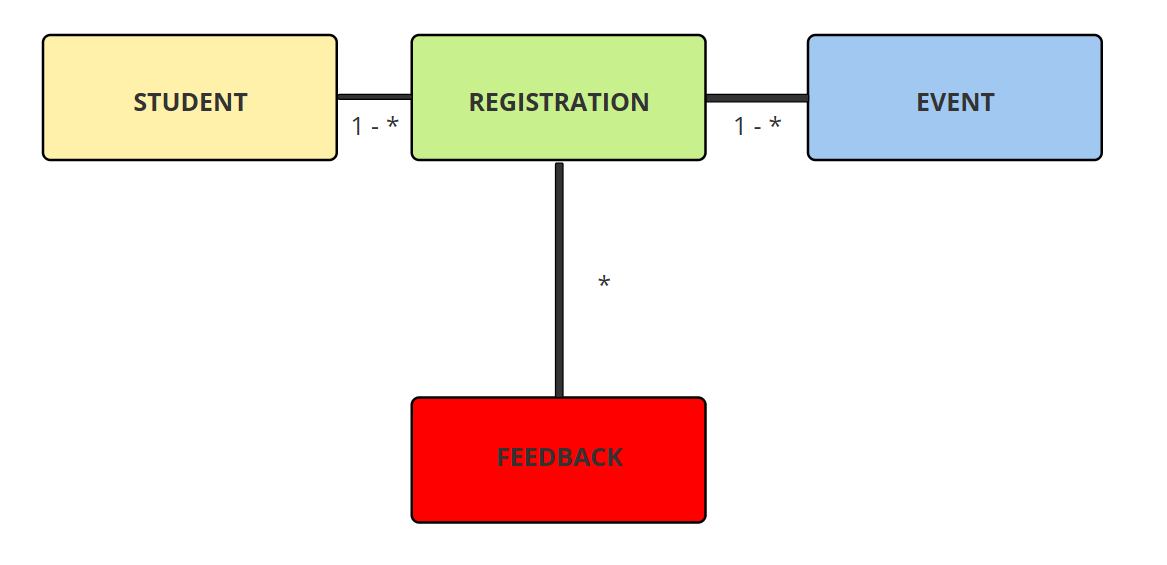


**2.2. Explanation of Layers**

* **Domain Layer (StudentEventManagement.Domain):**  
  This is the innermost layer and the heart of the application. It contains the core business models (entities) that represent the concepts of the system, such as Event, Student, Registration, and Feedback. This layer has zero dependencies on any other layer in the solution, making it completely independent of UI, database, or other external concerns.
* **Application Layer (StudentEventManagement.Application):**  
  This layer orchestrates the use cases of the application. It contains the business logic, service interfaces (IEventService, etc.), and their concrete implementations (EventService). It also defines the Data Transfer Objects (DTOs) used to pass data between the API and the services, as well as AutoMapper profiles for object-to-object mapping. It depends on the Domain layer but is agnostic to the presentation and data persistence technologies.
* **Infrastructure Layer (StudentEventManagement.Infrastructure):**  
  This layer implements the interfaces defined in the Application layer and handles all interactions with external systems. Its primary role in this project is data persistence. It contains the Entity Framework Core ApplicationDbContext, database migrations, and the concrete implementation of data access logic. It depends on the Application and Domain layers.
* **Presentation Layer (StudentEventManagement.API):**  
  This is the outermost layer and serves as the entry point for all external requests. It is an ASP.NET Core Web API project containing the API Controllers. The controllers are lightweight and their sole responsibility is to handle HTTP requests, validate input, call the appropriate service method in the Application layer, and return a formatted HTTP response. This layer depends on the Application and Infrastructure layers for service registration and dependency injection.

**3. Database Design**

The database was designed using a code-first approach with Entity Framework Core, allowing the C# entity classes in the Domain layer to define the database schema.

**3.1. EF Core Model Diagram (Entity Relationships):** The diagram below shows the relationships between the four main tables:

* **Student to Registration (One-to-Many):** One Student can have many Registration records.
* **Event to Registration (One-to-Many):** One Event can have many Registration records.
  + *Together, these two relationships create a logical****Many-to-Many****relationship between Student and Event through the Registration join table.*
* **Event to Feedback (One-to-Many):** One Event can receive many Feedback submissions.

**3.2. Database Schema Details**

* **Events Table:** Stores all information related to an event.
  + **Id** (uniqueidentifier, PK)
  + **Name** (nvarchar(max))
  + **Description** (nvarchar(max))
  + **Venue** (nvarchar(max))
  + **EventDate** (datetime2)
  + **CreatedAt** (datetime2)
* **Students Table:** Stores basic information about a student.
  + **Id** (uniqueidentifier, PK)
  + **Name** (nvarchar(max))
  + **Email** (nvarchar(max))
* **Registrations Table:** A join table linking students to the events they registered for.
  + **Id** (uniqueidentifier, PK)
  + **StudentId** (uniqueidentifier, FK to Students.Id)
  + **EventId** (uniqueidentifier, FK to Events.Id)
  + **RegistrationDate** (datetime2)
  + *Note: A unique index on the composite key (StudentId, EventId) prevents a student from registering for the same event more than once.*
* **Feedbacks Table:** Stores feedback submitted by students for a specific event.
  + **Id** (uniqueidentifier, PK)
  + **EventId** (uniqueidentifier, FK to Events.Id)
  + **StudentId** (uniqueidentifier)
  + **Rating** (int)
  + **Comment** (nvarchar(max))
  + **SubmittedAt** (datetime2)

**4. Validation and Error Handling**

A robust strategy for validation and error handling was implemented to ensure the API is reliable, secure, and easy for developers to use. This was achieved through a multi-layered approach.

**4.1. Input Validation**

Before any request is processed, the system first checks if the data sent by the user is in the correct format. This is done automatically using rules defined on our data models.

* **Required Fields:**The system ensures that essential information, like an event's name or a feedback rating, is not left empty.
* **Length Limits:**Text fields, such as an event description or a feedback comment, are checked to make sure they do not exceed a predefined maximum length.
* **Value Ranges:**Numerical data, like a feedback rating, is validated to ensure it falls within an acceptable range (for example, between 1 and 5).

If any of these rules are broken, the API immediately rejects the request with a 400 Bad Request error, telling the client exactly which field was incorrect.

**4.2. Business Rule Validation**

After the initial checks pass, the system validates the request against important business rules to maintain logical consistency.

* **For Registrations:**
  + The system checks if both the student and the event actually exist before creating a registration.
  + It prevents a student from registering for the same event more than once.
* **For Feedback:**
  + The most important rule is that the system checks the event's date. If the event has not happened yet, feedback is not allowed.
  + The system verifies that the student was actually registered for the event before allowing them to submit feedback.
  + It also prevents a student from submitting feedback for the same event multiple times.

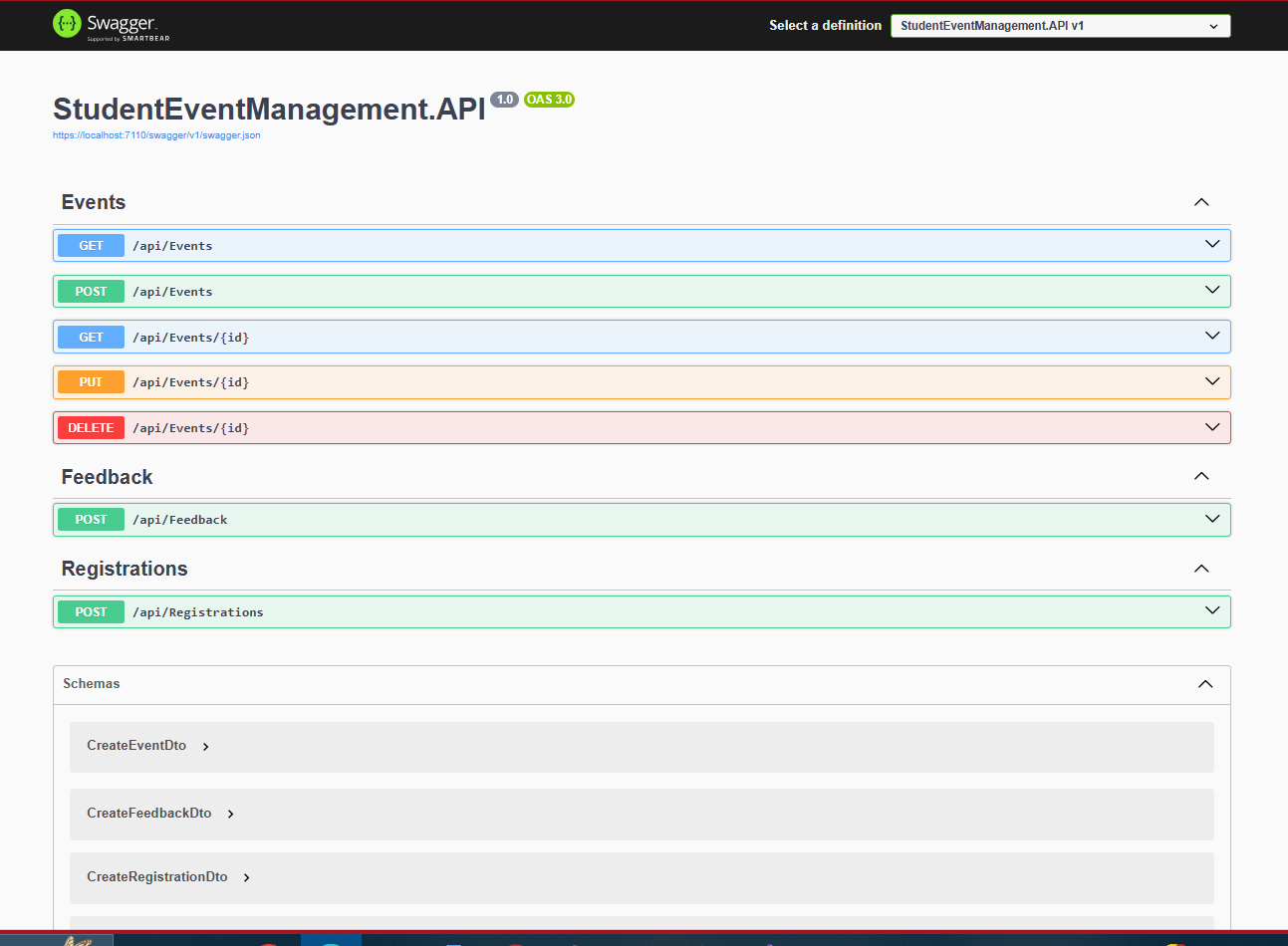
If any of these business rules fail, the API returns a clear error message with an appropriate status code, such as 409 Conflict for duplicates or 400 Bad Request for logical errors.

**4.3. Error Responses**

**The API is designed to return predictable and standard HTTP status codes to communicate the outcome of a request:**

* **404 Not Found**: Returned if a user tries to find, update, or delete something that doesn't exist (e.g., an event with an invalid ID).
* **500 Internal Server Error:**In the case of an unexpected problem on the server, a generic error is returned. This prevents sensitive system details from being exposed.

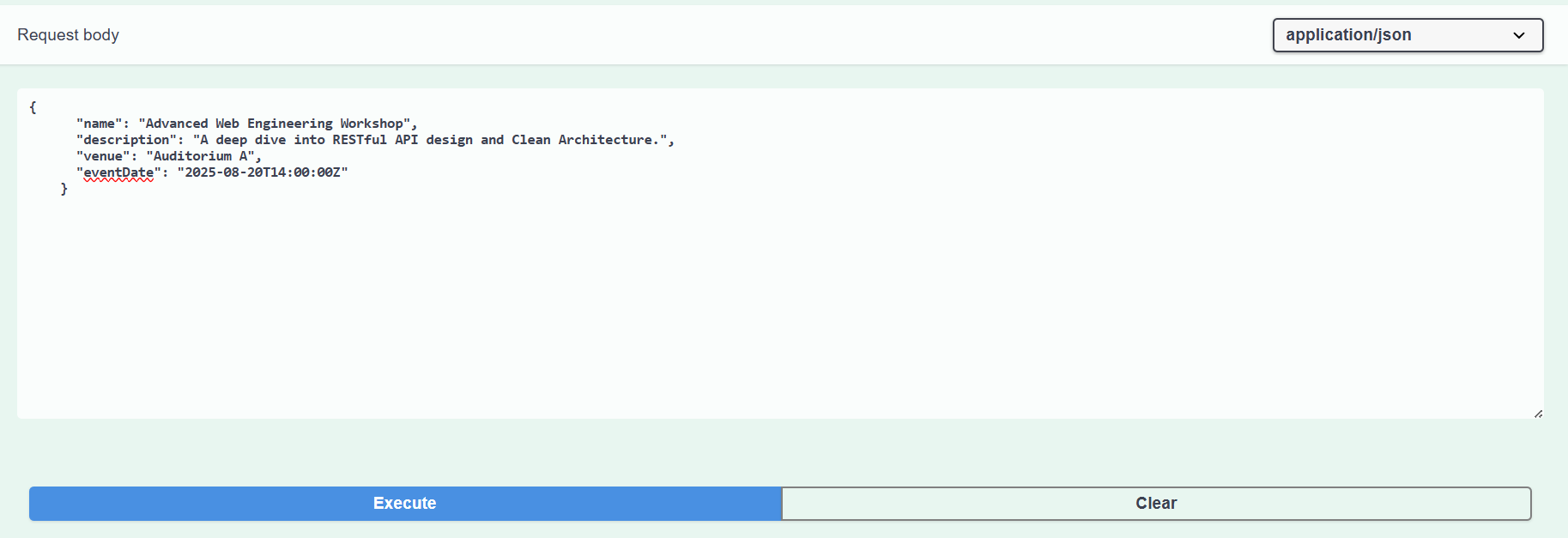
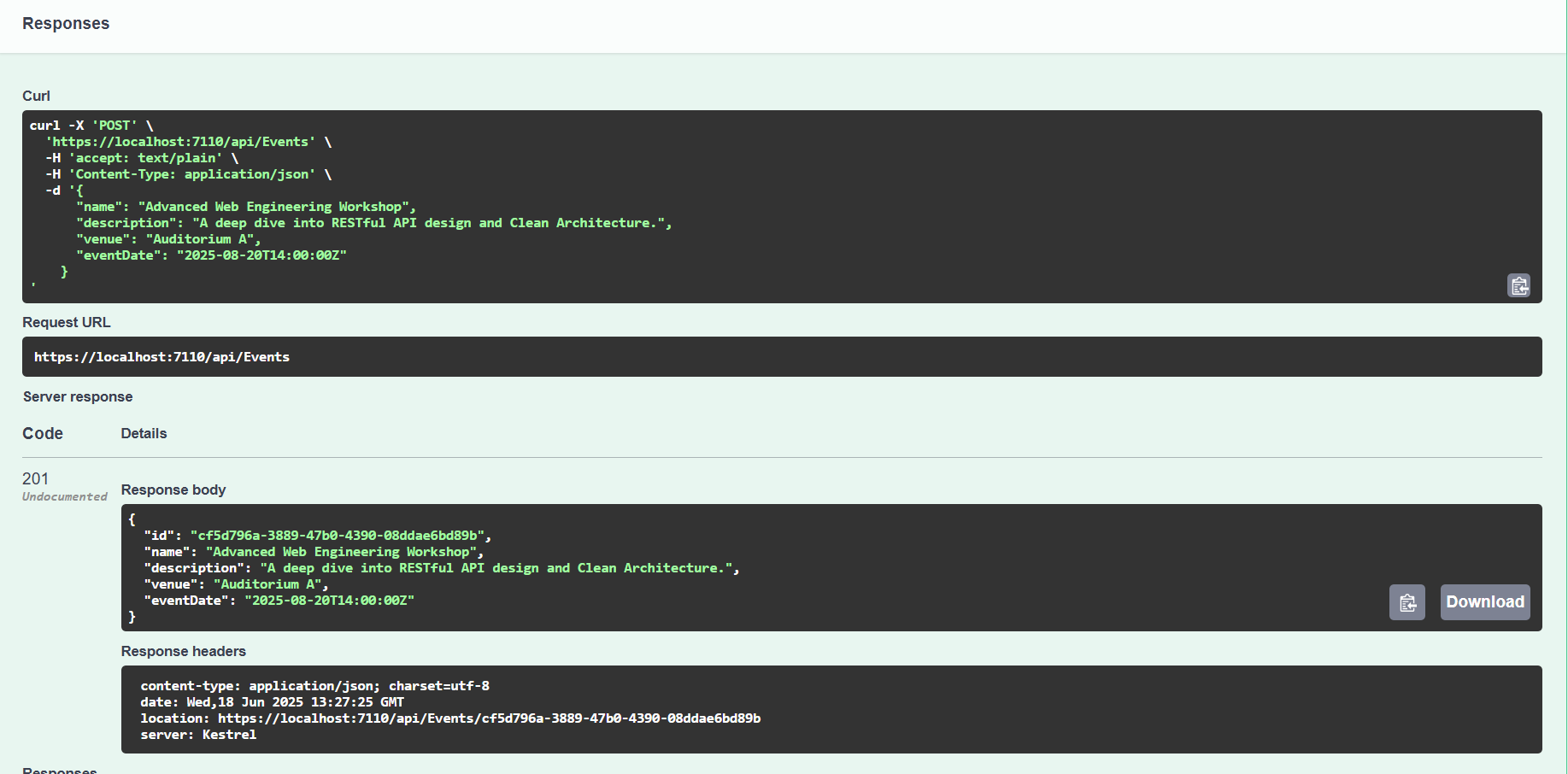
This comprehensive approach ensures that the data in our system remains valid and that users receive clear, helpful feedback when something goes wrong.

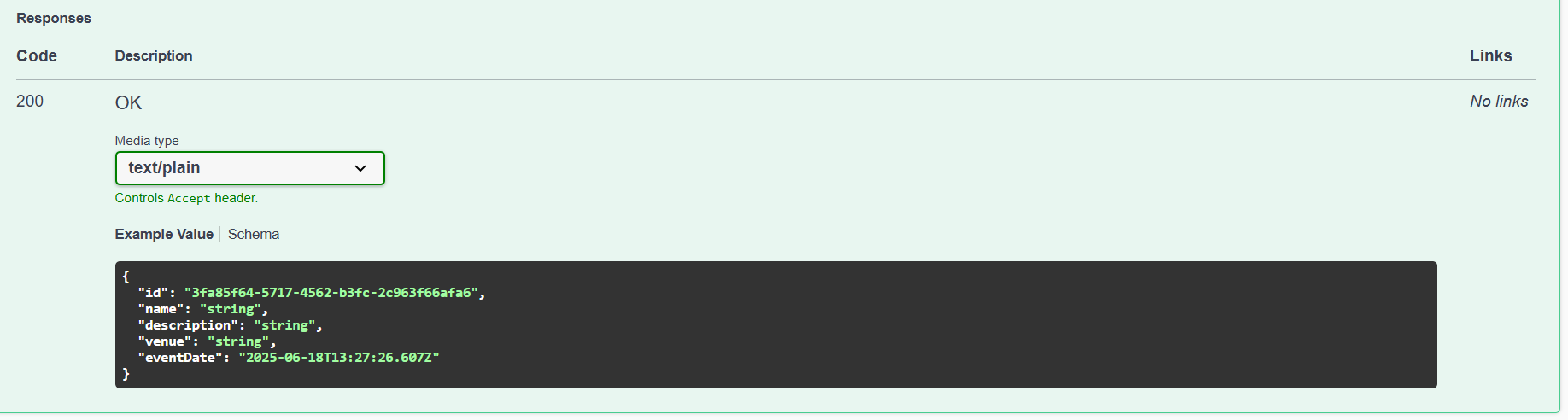
**UI Sample and other screenshots:**

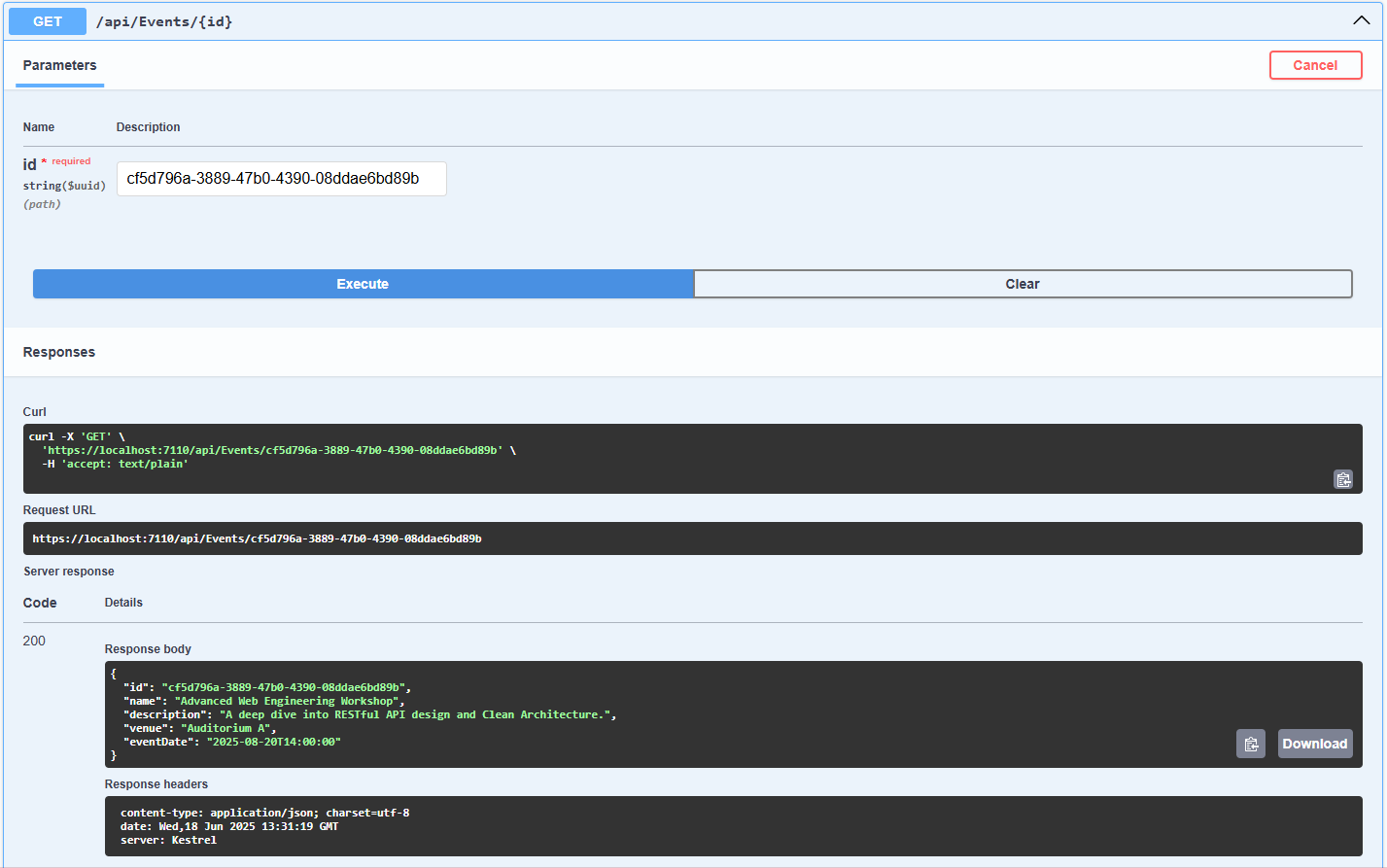
**4. Sample API Calls**

The following are samples of API requests and responses, demonstrating the core functionalities of the system.

**4.1. Creating a New Event**

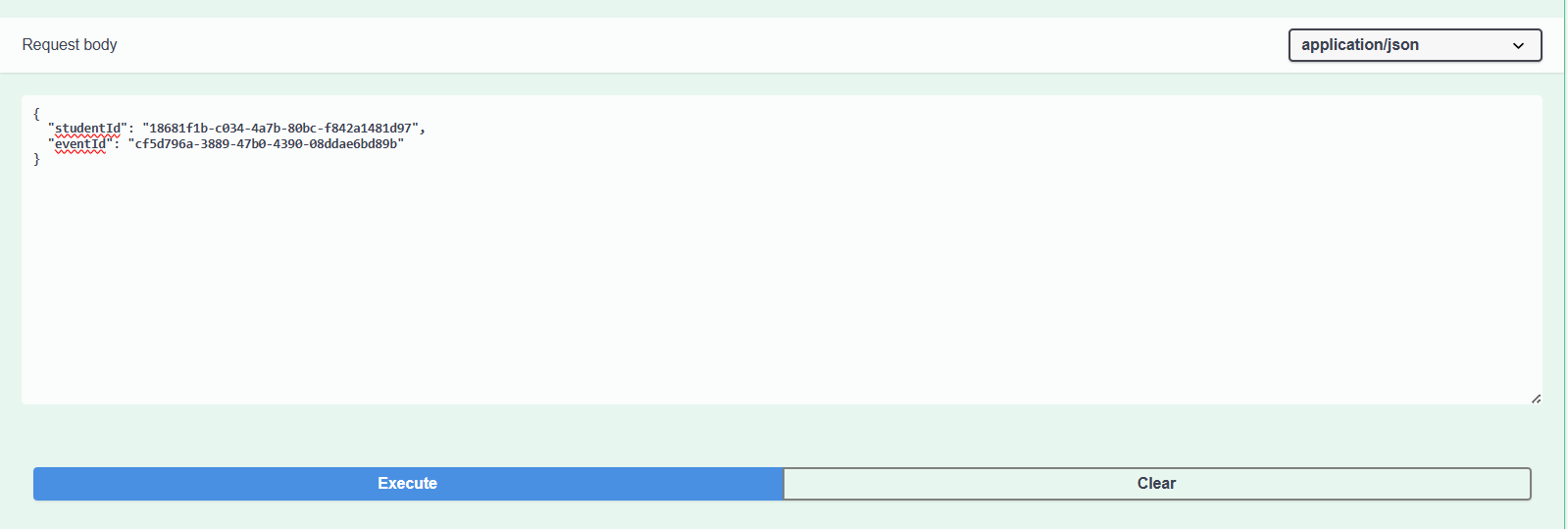
* **Request:** POST /api/events
* **Request Body:** 

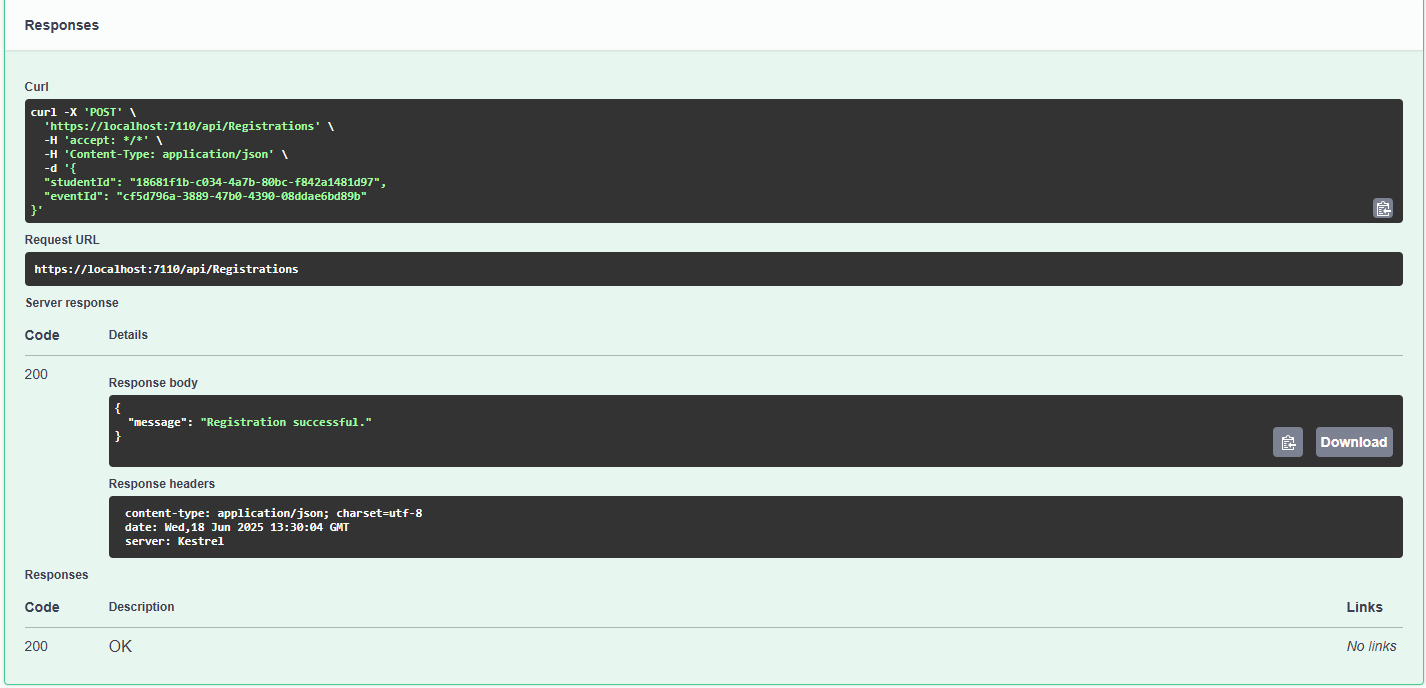


**EVENT NOW APPEARS IN THE GET SECTION by ID:**

**4.2. Registering a Student for an Event**

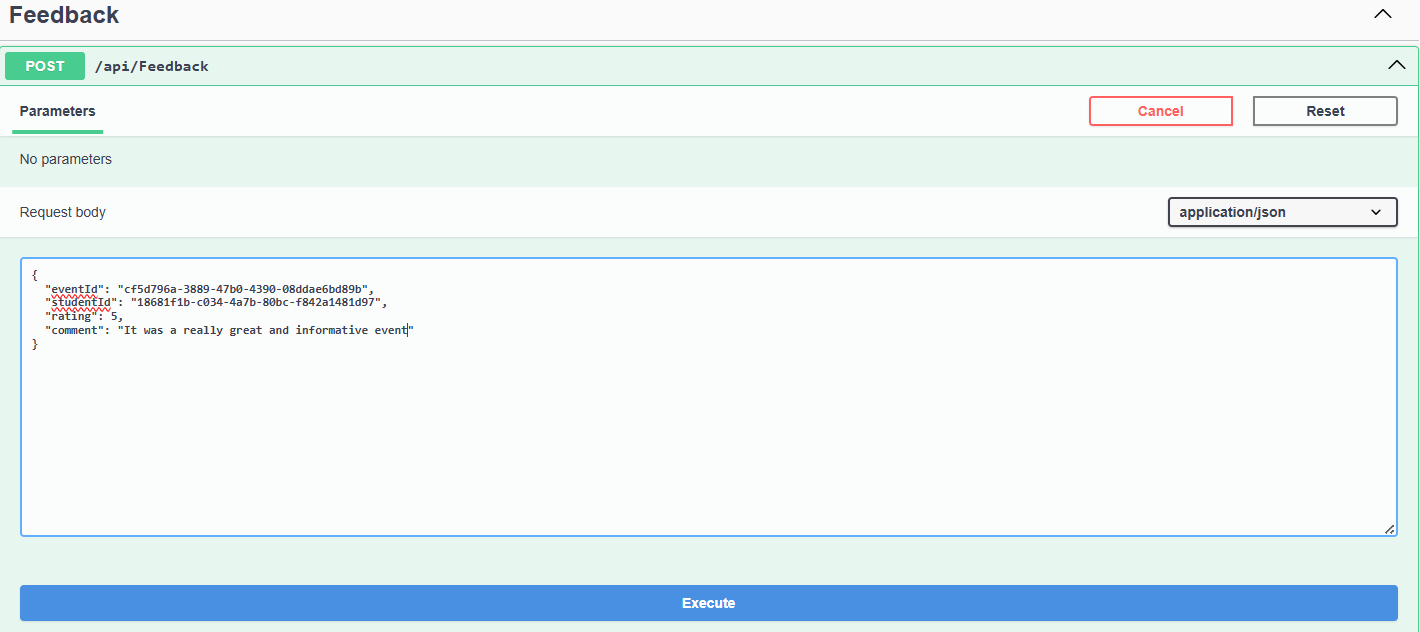
* **Request:** POST /api/registrations
* **Request Body:**

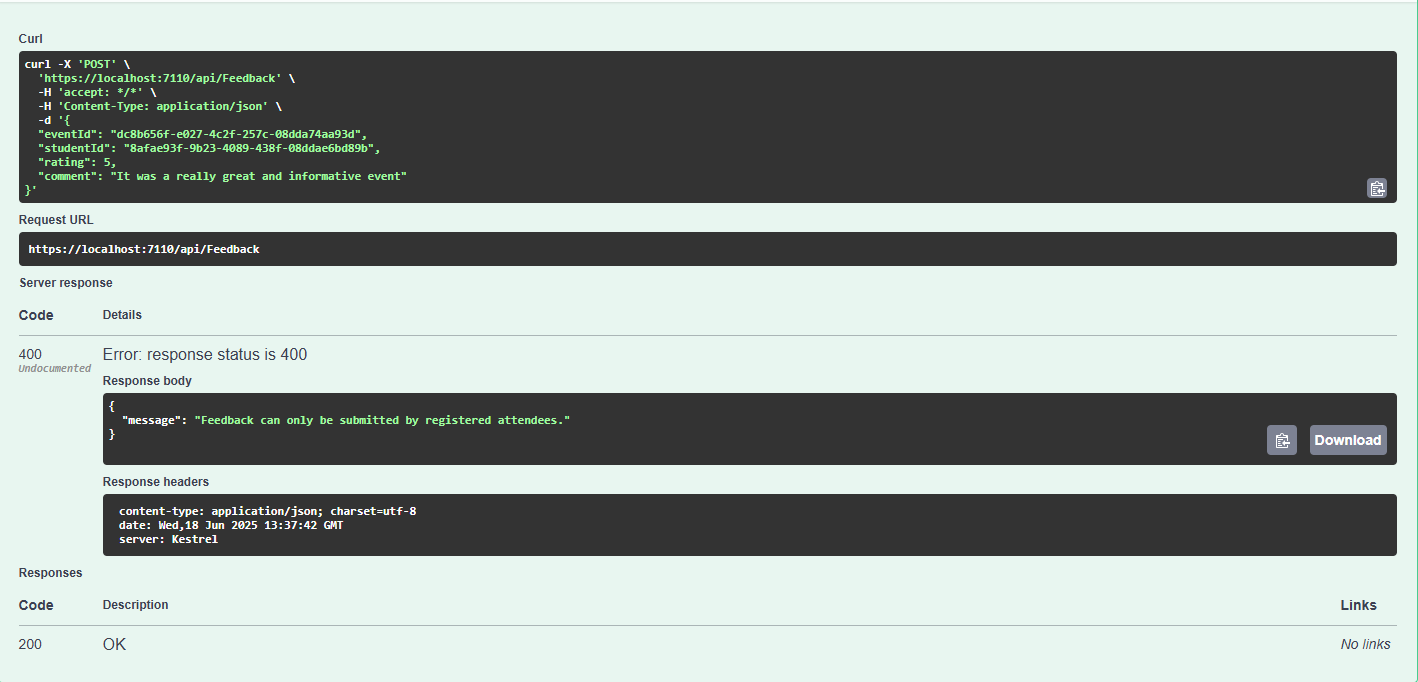


**SUCCESSFUL REGISTRATION:** 

**4.3. Submitting Feedback for a Completed Event**

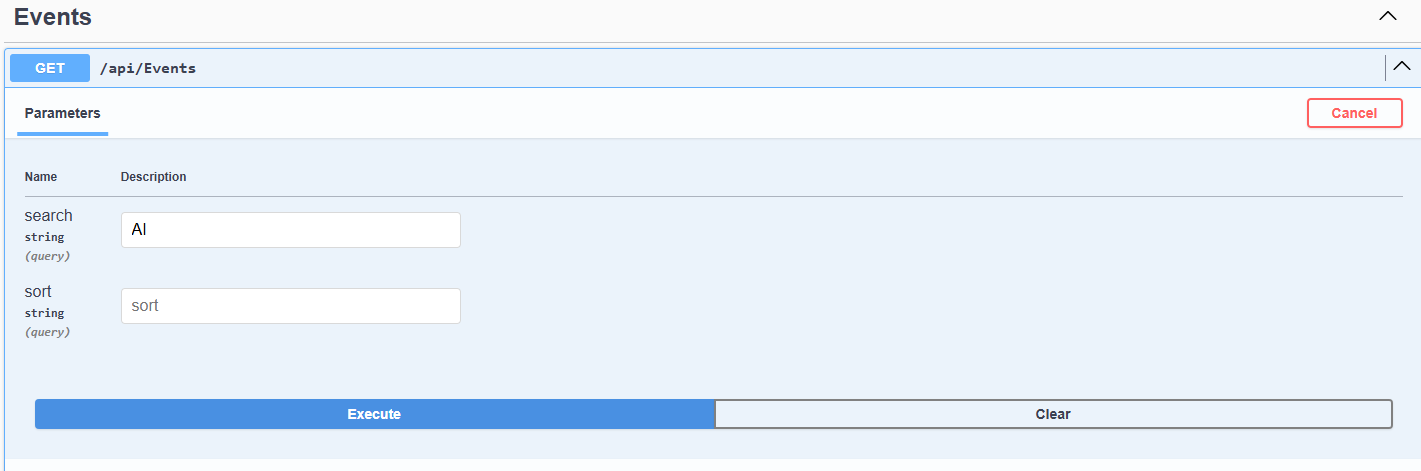
* **Request:** POST /api/feedback
* **Request Body:**

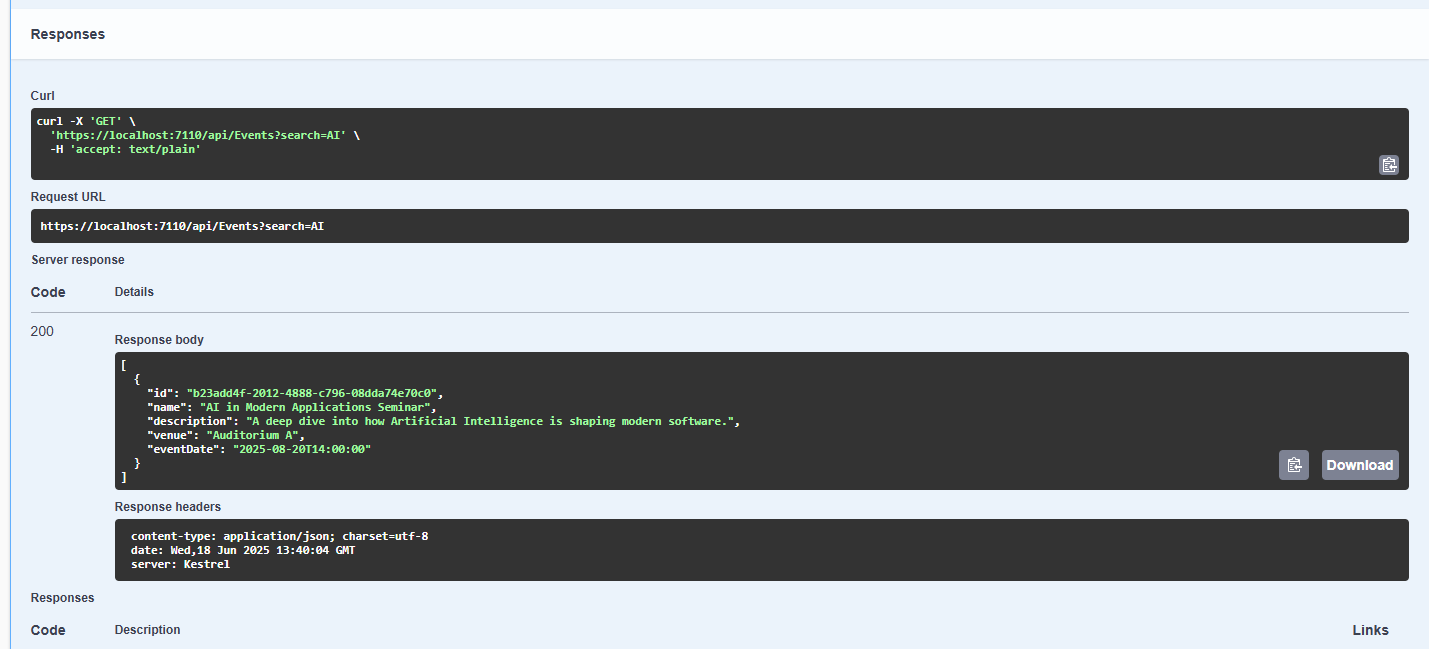
****

* **Successful Response:** Status: 200 OK

**4.4. Searching and Sorting Events**

* **Request:** GET /api/events?search=workshop&sortBy=name
* **Successful Response:** Status: 200 OK
  + The response body would contain a JSON array of all upcoming events containing the word "workshop" in their name or venue, sorted alphabetically by name.

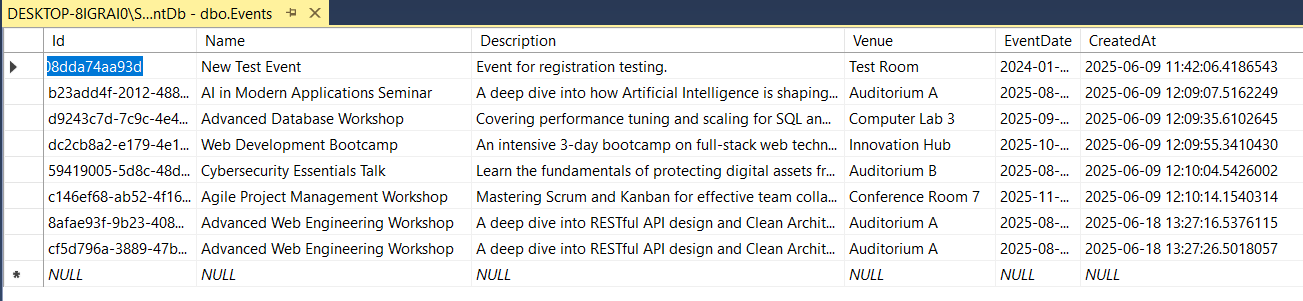


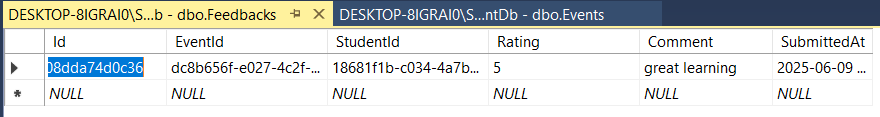
**Successful result of AI events:**

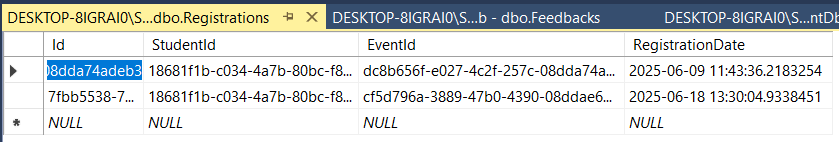
**TOTAL EVENTS:**

****

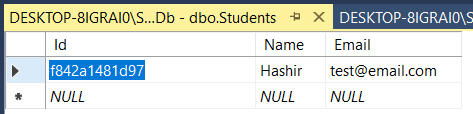
**DATABASE Screenshotes of Tables (SSMS 20):**

**Events Table:** ****

Feedback: 

Registrations: 

Students:



**5. GitHub Repository Link**

The complete source code for this project, including setup instructions in the README.md file, is publicly available at the following GitHub repository:

https://github.com/alihashirrana/StudentEventManagementAPI

**6. Conclusion**

This project successfully achieved all the specified objectives and delivered a fully functional backend API for the Student Event Management System. All required features, from basic CRUD operations to more complex business logic for registrations and feedback, were implemented according to modern software engineering standards. The adoption of Clean Architecture provides a solid foundation for future development, such as adding user authentication (JWT), role-based access control, and real-time notifications. The project demonstrates a comprehensive understanding of ASP.NET Core, Entity Framework Core, RESTful API design, and professional development workflows.