**1. Review of Entity Framework Core Integration**

Entity Framework (EF) Core is an Object-Relational Mapper (ORM) that enables .NET developers to work with a database using .NET objects, eliminating the need for most data-access code.

In this project, we have used:

* **Database First Approach** – The database is designed first, and EF Core models are generated from it.
* **DbContext Class** – It manages the database connection and mapping between classes and tables.
* **Stored Procedures with EF Core** – EF Core can execute stored procedures using FromSqlRaw() for read operations or ExecuteSqlRaw() for insert, update, and delete operations.

**2. Project Components**

1. **Database:**
   * StudentCourseDB containing two tables:
     + Students – Stores student details
     + Courses – Stores course information
   * Stored procedures:
     + GetAllStudents
     + sp\_GetStudentById
     + sp\_AddStudent
     + sp\_UpdateStudent
     + sp\_DeleteStudent
2. **Model Classes (in Models folder):**
   * Student.cs
   * Course.cs
   * These define the data structure in C# that maps to database tables.
3. **DbContext Class (ApplicationDbContext.cs):**
   * Contains DbSet<Student> and DbSet<Course>.
   * Manages database connections and configurations.
4. **Controller (StudentsController.cs):**
   * Contains API endpoints to perform CRUD operations using stored procedures and EF Core.

**3. CRUD Operations with EF Core and Stored Procedures**

**A. Retrieve All Students**

* **Purpose:** To fetch all student records along with their course details.
* **EF Core Code:**

var students = \_context.Students.FromSqlRaw("EXEC GetAllStudents").ToList();

* **Key Point:** The stored procedure must return all columns defined in the Student model, including CourseId.

**B. Retrieve Student by ID**

* **Purpose:** To fetch a specific student record based on their ID.
* **EF Core Code:**

var student = \_context.Students

.FromSqlRaw("EXEC sp\_GetStudentById @Id={0}", id)

.AsEnumerable()

.FirstOrDefault();

**Validation:** Return 404 if the student is not found.

**C. Add a New Student**

* **Purpose:** To insert a new student record into the database.
* **EF Core Code:**

\_context.Database.ExecuteSqlRaw(

"EXEC sp\_AddStudent @Name={0}, @Age={1}, @Grade={2}, @CourseId={3}",

student.Name, student.Age, student.Grade, student.CourseId

);

**Validation:** Check if all required fields are provided before executing.

**D. Update Student Details**

* **Purpose:** To update an existing student record.
* **EF Core Code:**

\_context.Database.ExecuteSqlRaw(

"EXEC sp\_UpdateStudent @Id={0}, @Name={1}, @Age={2}, @Grade={3}, @CourseId={4}",

id, student.Name, student.Age, student.Grade, student.CourseId

);

**Validation:** Ensure the student exists before updating.

**E. Delete Student**

* **Purpose:** To delete a student record from the database.
* **EF Core Code:**

\_context.Database.ExecuteSqlRaw("EXEC sp\_DeleteStudent @Id={0}", id);

**Validation:** Confirm that the record exists before deletion.

**4. Testing the API**

**A. Using Swagger**

1. Run the project (dotnet run).
2. Open Swagger at: http://localhost:<port>/swagger.
3. Test endpoints:
   * GET /api/students – Fetch all students.
   * GET /api/students/{id} – Fetch by ID.
   * POST /api/students – Add new student.
   * PUT /api/students/{id} – Update details.
   * DELETE /api/students/{id} – Delete a record.

**B. Using Postman**

1. Create a new collection for StudentApi.
2. Add requests for all endpoints with proper HTTP methods.
3. Use JSON body in POST and PUT requests like:

{

"name": "John Doe",

"age": 23,

"grade": "A",

"courseId": 2

}

**5. Common Errors and Fixes**

|  |  |  |
| --- | --- | --- |
| **Error** | **Cause** | **Fix** |
| CourseId missing in FromSql | Stored procedure not returning all columns | Include CourseId in SELECT statement |
| Invalid column name | Property name mismatch | Ensure property names match table columns |
| 500 Internal Server Error | Missing validation or incorrect SQL syntax | Check stored procedure parameters and data types |
| Cannot connect to DB | Connection string issue | Verify appsettings.json connection string |

**6. Validation and Best Practices**

* Use **Data Annotations** in model classes for validation (e.g., [Required], [StringLength], [Range]).
* Implement **try-catch blocks** in controllers to handle exceptions gracefully.
* Ensure **stored procedures** have proper error handling using TRY-CATCH in SQL.
* Follow **RESTful conventions** for API routes and responses.
* Always **log errors** in case of failure for debugging.

**8. Mini Exercise**

Build and test all API endpoints:

1. Fetch all students.
2. Fetch student by ID.
3. Add a new student.
4. Update an existing student.
5. Delete a student record.

**Snapshots :**

A screenshot of a computer

AI-generated content may be incorrect.

Database and tables created successfully in SSMS

A screenshot of a computer

AI-generated content may be incorrect.

Stored procedures verified and executed in SSMS (sp\_GetStudents)

A screenshot of a computer program

AI-generated content may be incorrect.

Project structure shown in Visual Studio

A screen shot of a computer

AI-generated content may be incorrect.

Connection string configured in appsettings.json

A screenshot of a computer

AI-generated content may be incorrect.

ApplicationDbContext class with DbSets for Student and Course

A screenshot of a computer program

AI-generated content may be incorrect.

StudentsController implemented with CRUD endpoints

A screen shot of a computer

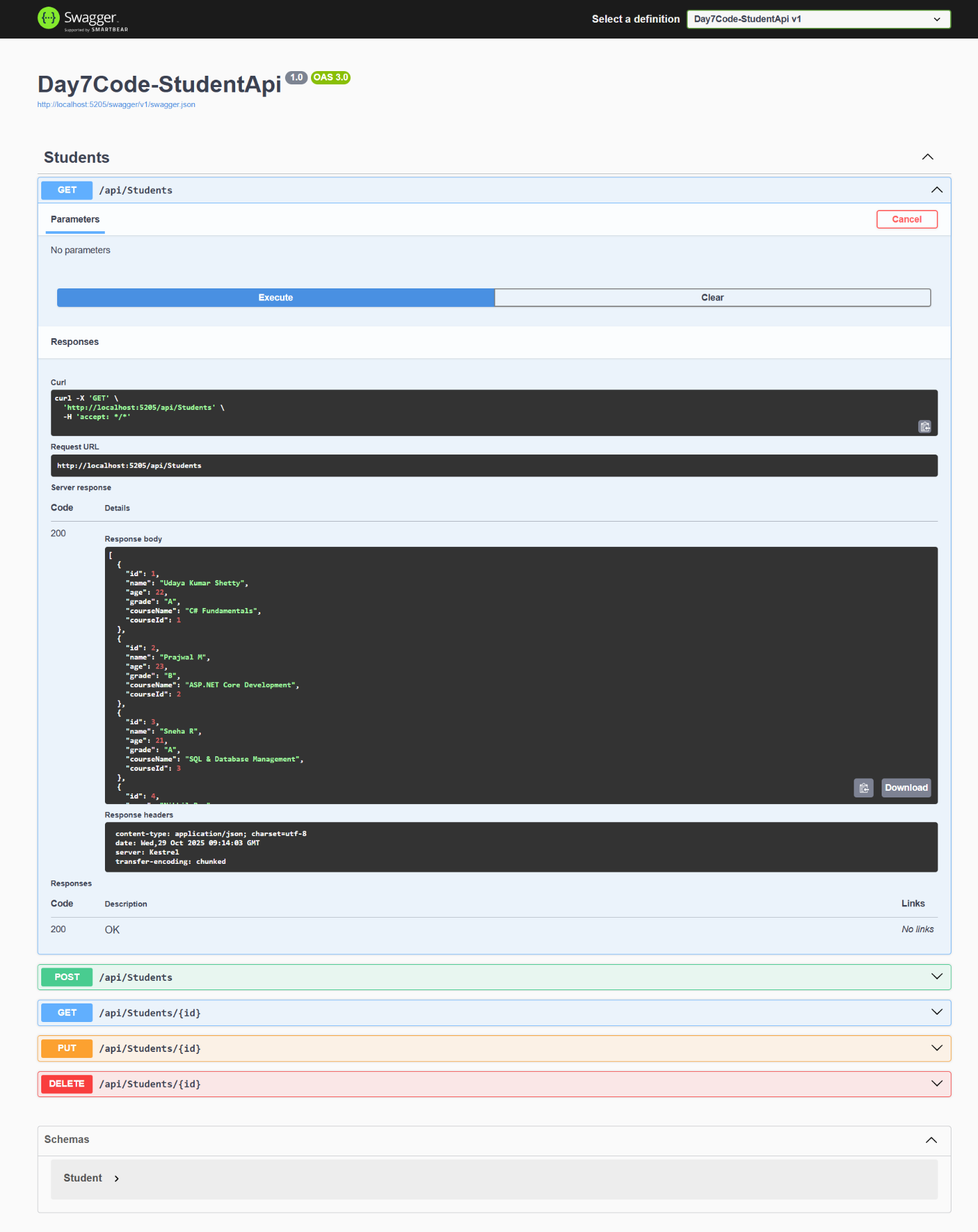
AI-generated content may be incorrect.

dotnet build completed successfully in terminal

A screenshot of a computer

AI-generated content may be incorrect.

Swagger UI loaded showing all API endpoints



GET /api/students returns full list of students

A screenshot of a computer

AI-generated content may be incorrect.

POST /api/students adds new student successfully

A screenshot of a computer

AI-generated content may be incorrect.

GET /api/students/{id} returns specific student

A screenshot of a computer

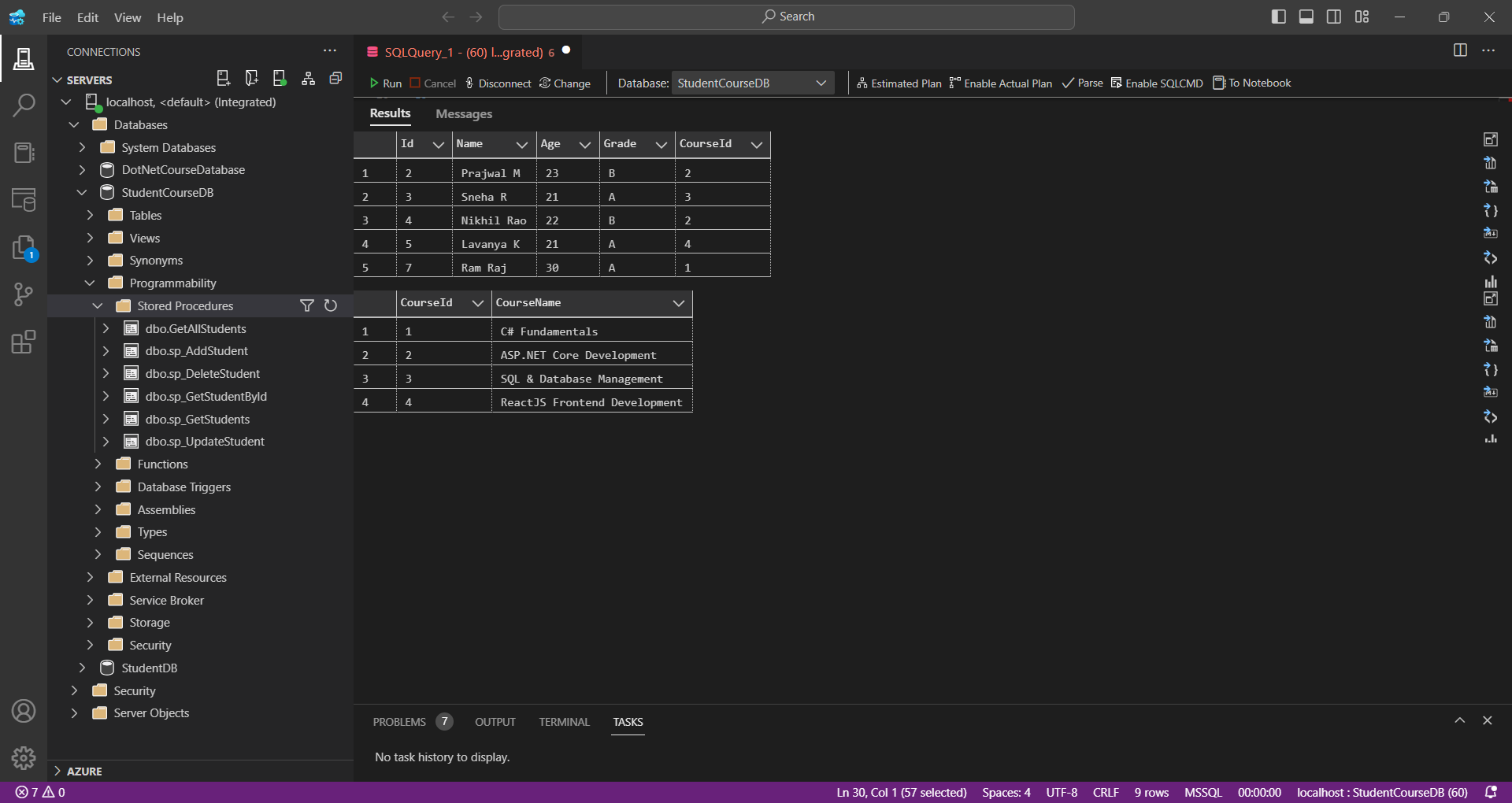
AI-generated content may be incorrect.

PUT /api/students/{id} updates record details

A screenshot of a computer

AI-generated content may be incorrect.

DELETE /api/students/{id} removes record from database



SQL Server view shows updated student and course tables after CRUD operations