

## Day 4 –Validation, DTOs, and Error Handling in ASP.NET Core      Date: 19-10-2025

**Objective:** Learn to validate user input, structure data with DTOs, and handle errors gracefully in ASP.NET Core. Build secure, maintainable APIs with clear feedback and robust logic.

### 1. Objective

Today’s goal is to improve your existing CRUD Web API by:

- Adding **input validation** using Data Annotations.
- Implementing **DTOs** to separate API models from database entities.
- Adding **global exception handling** and proper HTTP responses.

This ensures your backend is **secure, robust, and maintainable**.

### 2. Why Validation, DTOs, and Error Handling Are Important

| Concept                           | Purpose  |
|-----------------------------------|--|
| <b>Validation</b>                 | Ensures that only correct and meaningful data enters the database.               |
| <b>DTO (Data Transfer Object)</b> | Prevents over-posting and hides internal database structure from external users. |
| <b>Error Handling</b>             | Prevents API crashes, provides user-friendly messages, and improves debugging.   |

#### Part 1 – Data Validation

---

### 3. What is Data Validation?

Validation ensures that user input meets specific rules before being stored in the database. It prevents bad data, SQL injection, and runtime errors.

In ASP.NET Core, validation is usually done through **Data Annotations** in your model classes.

### 4. Common Data Annotations

| Attribute                    | Purpose                          | Example                                     |
|------------------------------|----------------------------------|---|
| [Required]                   | Field must not be null or empty. | [Required] public string Name { get; set; } |
| [StringLength(50)]           | Limits the string length.        | [StringLength(50)]                          |
| [Range(1,100)]               | Limits numeric range.            | [Range(1,100)]                              |
| [EmailAddress]               | Validates email format.          | [EmailAddress]                              |
| [RegularExpression()<br>( )] | Validates pattern.               | [RegularExpression(@"^[A-Z]{2}\d{4}\$")]    |

## 5. Example – Applying Validation to Model

```
public class Student
{
    [Key]
    public int Id { get; set; }

    [Required(ErrorMessage = "Name is required")]
    [StringLength(50, ErrorMessage = "Name cannot exceed 50 characters")]
    public string Name { get; set; }

    [Range(1, 100, ErrorMessage = "Age must be between 1 and 100")]
    public int Age { get; set; }

    [StringLength(10)]
    public string? Grade { get; set; }
}
```

If invalid data is sent, ASP.NET automatically returns **400 Bad Request** with a JSON error message.

## 6. Handling Invalid Model Data

You must always check `ModelState.IsValid` in the controller:

```
[HttpPost]
public async Task<IActionResult> AddStudent(Student student)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    _context.Students.Add(student);
    await _context.SaveChangesAsync();
}
```

```
        return CreatedAtAction(nameof(GetStudentById), new { id = student.Id }, student);
    }
}
```

If validation fails, the API automatically returns which fields are invalid.

## Part 2 – Using DTOs (Data Transfer Objects)

---

### 7. What is a DTO?

A **DTO** is a lightweight class used to transfer data between client and server. It hides unnecessary database details and protects sensitive fields.

For example, you might not want to expose fields like database IDs or internal metadata to the frontend.

### 8. Why Use DTOs?

| Reason          | Explanation   |
|-----------------|---|
| Security        | Prevents users from setting restricted fields like IDs.     |
| Clarity         | Ensures only necessary fields are visible in API responses. |
| Maintainability | Model changes won't break client code.                      |

### 9. Example – Creating a Student DTO

**Create DTO Class (Models/DTOs/StudentDto.cs):**

```
namespace StudentApi.Models.DTOs
{
    public class StudentDto
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public string? Grade { get; set; }
    }
}
```

Now, use it in your controller instead of exposing the full Student model.

## 10. Using DTOs in Controllers

[HttpPost]

```
public async Task<IActionResult> AddStudent(StudentDto studentDto)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var student = new Student
    {
        Name = studentDto.Name,
        Age = studentDto.Age,
        Grade = studentDto.Grade
    };

    _context.Students.Add(student);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetStudentById), new { id = student.Id }, student);
}
```

## Part 3 – Error Handling in Web API

---

### 11. Why Error Handling is Required

Without proper error handling:

- The app may crash on unexpected exceptions.
- Users might see unhandled technical messages.
- Debugging production issues becomes difficult.

Good error handling ensures **graceful failure** and **clear feedback**.

## 12. Using Try-Catch in Controllers

Example:

```
[HttpGet("{id}")]
```

```
public async Task<IActionResult> GetStudentById(int id)
{
    try
    {
        var student = await _context.Students.FindAsync(id);
        if (student == null)
            return NotFound("Student not found");
        return Ok(student);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

### Explanation:

- try → executes normal logic.
- catch → catches exceptions (e.g., database errors).
- Returns status code **500 Internal Server Error** with a custom message.

## 13. Using Global Error Handling (Middleware)

Instead of repeating try-catch in every controller, you can create a **middleware** that catches all exceptions globally.

### Middleware Example:

```
public class GlobalExceptionHandlerMiddleware
{
    private readonly RequestDelegate _next;
```

```

public GlobalExceptionHandler(RequestDelegate next)
{
    _next = next;
}

public async Task InvokeAsync(HttpContext context)
{
    try
    {
        await _next(context);
    }
    catch (Exception ex)
    {
        context.Response.StatusCode = 500;
        await context.Response.WriteAsJsonAsync(new
        {
            Message = "An unexpected error occurred.",
            Details = ex.Message
        });
    }
}

```

#### **Register in Program.cs:**

```
app.UseMiddleware<GlobalExceptionHandler>();
```

This handles all unhandled exceptions in one place and keeps controllers clean.

## **14. Standard API Response Structure**

For production APIs, always return consistent responses:

```

{
    "success": true,

```

```
"message": "Student updated successfully",  
"data": {  
  "id": 1,  
  "name": "Udaya Kumar Shetty",  
  "age": 22,  
  "grade": "A"  
}
```

This helps frontend developers parse responses uniformly.

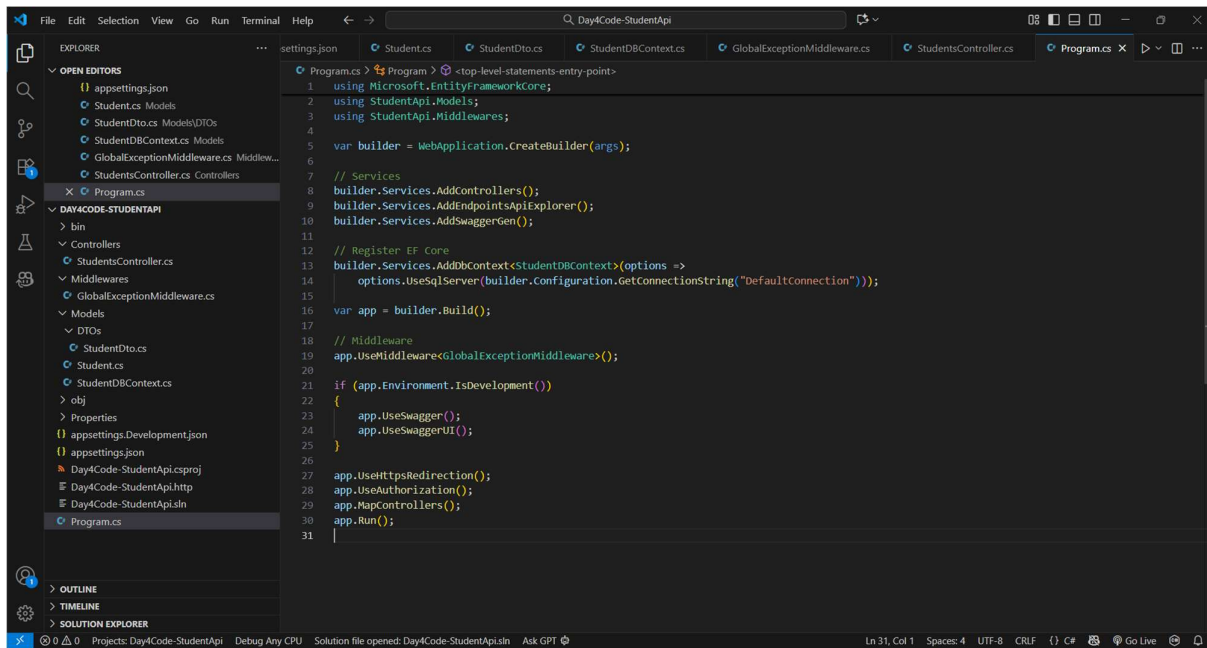
### **Mini Task for Day 4**

#### **Objective:**

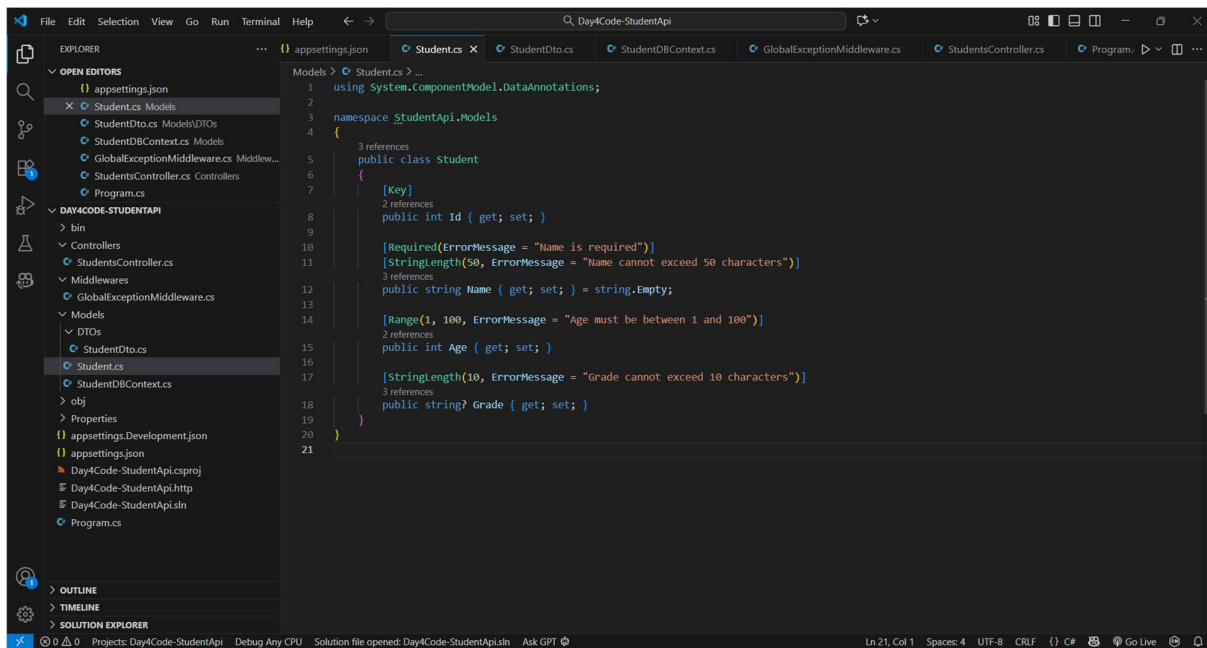
Upgrade your existing StudentApi to include:

1. Validation in the Student model.
2. Use of StudentDto for POST and PUT endpoints.
3. Add try-catch for all controller methods.
4. Optional: Implement global exception middleware.

## Snapshots :

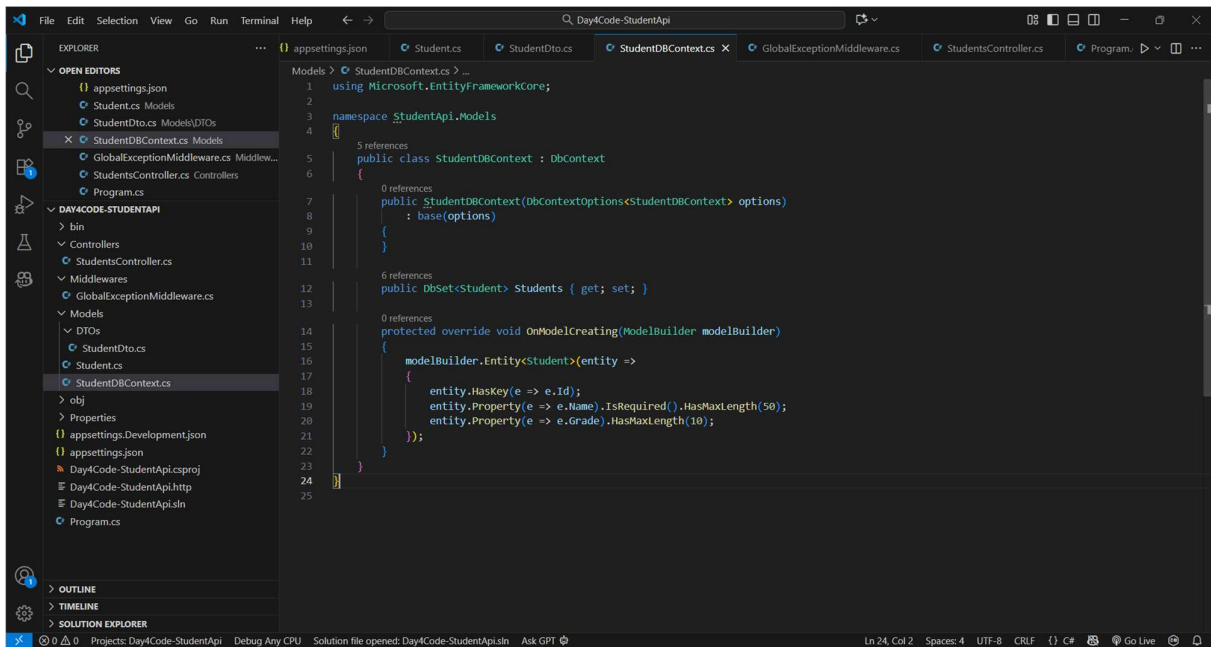


## Code : Program.cs

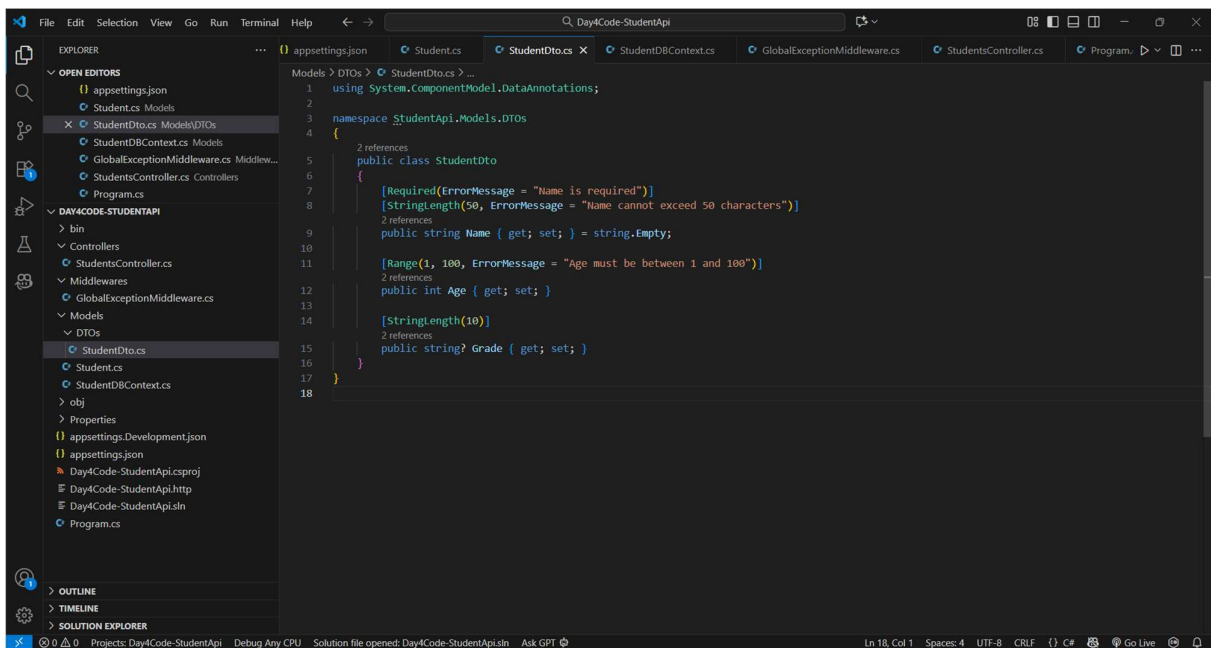


## Code : Student.cs





Code : StudentDbContext.cs



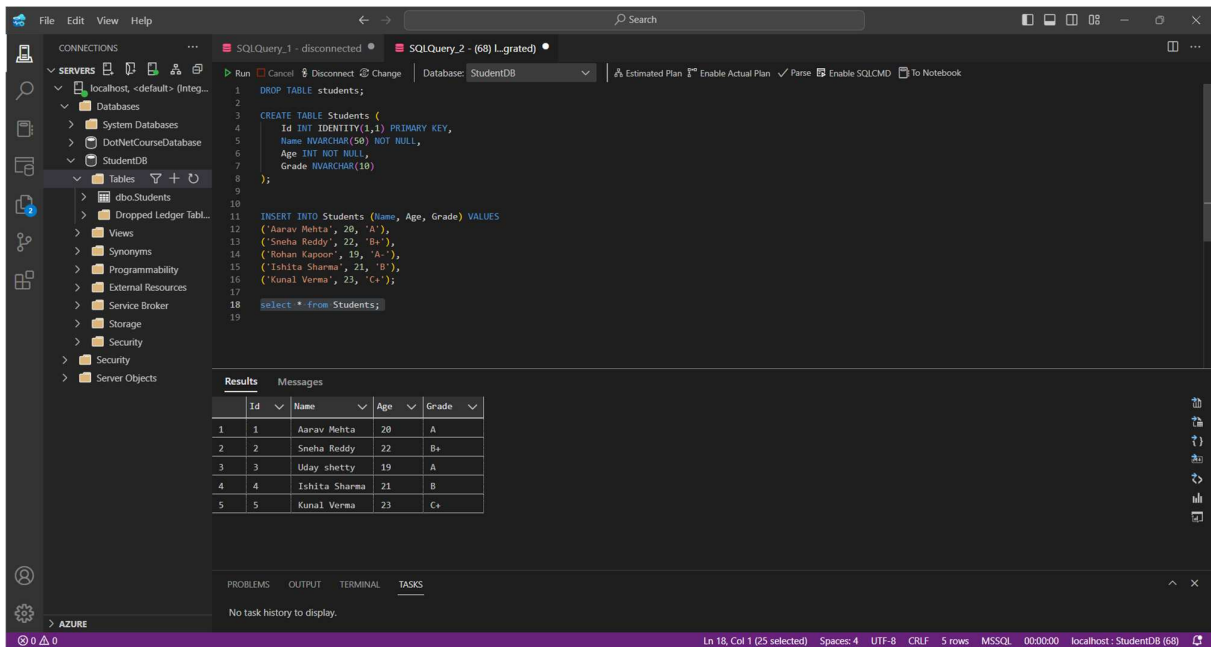
Code : StudentDto.cs

```
1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.EntityFrameworkCore;
3 using StudentApi.Models;
4 using StudentApi.Models.DTOs;
5
6 namespace StudentApi.Controllers
7 {
8     [ApiController]
9     [Route("api/[controller]")]
10    public class StudentsController : ControllerBase
11    {
12        11 references
13        private readonly StudentDbContext _context;
14
15        0 references
16        public StudentsController(StudentDbContext context)
17        {
18            _context = context;
19        }
20
21        // GET: api/students
22        [HttpGet]
23        0 references
24        public async Task<ActionResult> GetAllStudents()
25        {
26            try
27            {
28                var students = await _context.Students.ToListAsync();
29                return Ok(new { Success = true, Data = students });
30            }
31            catch (Exception ex)
32            {
33                return StatusCode(500, new { Success = false, Message = ex.Message });
34            }
35        }
36    }
37}
```

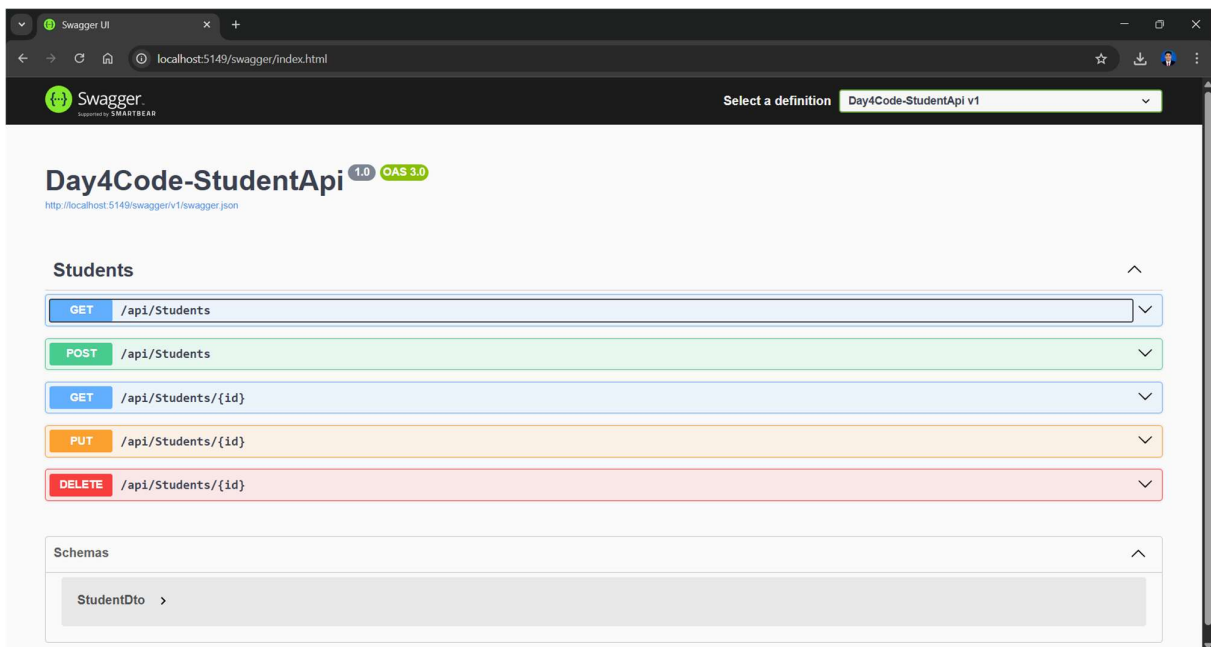
Code : StudentController.cs

```
1 using System.Net;
2 using System.Text.Json;
3
4 namespace StudentApi.Middlewares
5 {
6     2 references
7     public class GlobalExceptionHandlerMiddleware
8     {
9         2 references
10        private readonly RequestDelegate _next;
11
12        0 references
13        public GlobalExceptionHandlerMiddleware(RequestDelegate next)
14        {
15            _next = next;
16        }
17
18        0 references
19        public async Task InvokeAsync(HttpContext context)
20        {
21            try
22            {
23                await _next(context);
24            }
25            catch (Exception ex)
26            {
27                context.Response.ContentType = "application/json";
28                context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
29
30                var errorResponse = new
31                {
32                    Success = false,
33                    Message = "An unexpected error occurred.",
34                    Details = ex.Message
35                };
36
37                var json = JsonSerializer.Serialize(errorResponse);
38                await context.Response.WriteAsync(json);
39            }
40        }
41    }
42}
```


Code : GlobalExceptionHandlerMiddleware.cs



Code : Database Code



Output : All method

 **Swagger**  
OpenAPI 3.0

Select a definition **Day4Code-StudentApi v1**

## Day4Code-StudentApi 1.0 OAS 3.0

<http://localhost:5149/swagger/v1/swagger.json>

### Students

**GET** /api/Students

**Parameters**

No parameters

Execute

Clear

**Responses**

Curl

```
curl -X 'GET' \
  'http://localhost:5149/api/Students' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5149/api/Students
```

Server response

Code

Details

200

Response body

```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "Aarav Mahta",
      "age": 20,
      "grade": "A"
    },
    {
      "id": 2,
      "name": "Sneha Reddy",
      "age": 22,
      "grade": "B+"
    },
    {
      "id": 3,
      "name": "Uday shetty",
      "age": 19,
      "grade": "A"
    },
    {
      "id": 4,
      "name": "Ishita Sharma",
      "age": 21,
      "grade": "B"
    }
  ]
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Tue, 28 Oct 2025 17:32:27 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

| Code | Description | Links    |
|------|-------------|----------|
| 200  | OK          | No links |

**POST** /api/Students

**GET** /api/Students/{id}


**PUT** /api/Students/{id}

**DELETE** /api/Students/{id}

Schemas

StudentDto >

Output : GET (All data from database)


Swagger

Select a definition
Day4Code-StudentApi v1

## Day4Code-StudentApi 1.0 OAS 3.0

http://localhost:5149/swagger/v1/swagger.json

### Students

GET /api/Students

POST /api/Students

Parameters
Cancel
Reset

No parameters

Request body application/json

Edit Value
Schema

```

{
  "name": "Raj",
  "age": 16,
  "grade": "8"
}

```

Execute
Clear

### Responses

Curl

```

curl -X 'POST' \
  'http://localhost:5149/api/Students' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "name": "Raj",
    "age": 16,
    "grade": "8"
  }'

```

Request URL

http://localhost:5149/api/Students

Server response

| Code | Details  |
|------|--|
| 201  | <p>Response body</p> <pre> {   "success": true,   "message": "Student added successfully",   "data": {     "id": 1,     "name": "Raj",     "age": 16,     "grade": "8"   } } </pre> <p>Response headers</p> <pre> content-type: application/json; charset=utf-8 date: Tue, 28 Oct 2025 17:33:12 GMT location: http://localhost:5149/api/Students/8 server: Kestrel transfer-encoding: chunked </pre> |

Responses

| Code | Description | Links    |
|------|-------------|----------|
| 200  | OK          | No links |

GET /api/Students/{id}


PUT /api/Students/{id}

DELETE /api/Students/{id}

### Schemas

StudentDto >

Output : POST Method (Insert data )

 Swagger  
Powered by SMARTBEAR

Select a definition Day4Code-StudentApi v1

## Day4Code-StudentApi 1.0 OAS 3.0

<http://localhost:5149/swagger/v1/swagger.json>

### Students

GET /api/Students

POST /api/Students

GET /api/Students/{id}

Parameters

| Name                     | Description      |
|--------------------------|------------------|
| id <span>required</span> | integer(\$int32) |

1

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5149/api/Students/1' \
-H 'accept: */*'
```

Request URL

```
http://localhost:5149/api/Students/1
```

Server response

| Code | Details   |
|------|---|
| 200  | <div><div>Response body</div><div><pre>{   "success": true,   "data": {     "id": 1,     "name": "Aarav Mehta",     "age": 20,     "grade": "A"   } }</pre></div><div>Download</div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Tue, 28 Oct 2025 17:33:38 GMT server: Kestrel transfer-encoding: chunked</pre></div></div> |

| Code | Description | Links    |
|------|-------------|----------|
| 200  | OK          | No links |

PUT /api/Students/{id}

DELETE /api/Students/{id}

Schemas

StudentDto

Output : GET (Specific data from database)

## Day4Code-StudentApi <sup>1.0</sup> <sup>OAS 3.0</sup>

<http://localhost:5149/swagger/v1/swagger.json>

### Students

- [GET /api/Students](#)
- [POST /api/Students](#)
- [GET /api/Students/{id}](#)
- [PUT /api/Students/{id}](#)

Parameters

Cancel Reset

| Name                   | Description       |
|------------------------|-------------------|
| id <sup>required</sup> | Integer (int32) 1 |

(path)

Request body application/json

Edit Value | Schema

```
{
  "name": "Uday Raj",
  "age": 19,
  "grade": "A"
}
```

Execute Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:5149/api/Students/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "name": "Uday Raj",
    "age": 19,
    "grade": "A"
  }'
```

Request URL

<http://localhost:5149/api/Students/1>

Server response

| Code | Details  |
|------|--|
| 200  | <p>Response body</p> <pre>{   "success": true,   "message": "Student updated successfully",   "data": {     "id": 1,     "name": "Uday Raj",     "age": 19,     "grade": "A"   } }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 28 Oct 2025 17:34:36 GMT server: Kestrel transfer-encoding: chunked</pre> |

Responses


| Code | Description | Links    |
|------|-------------|----------|
| 200  | OK          | No links |

- [DELETE /api/Students/{id}](#)

Schemas

StudentDto >

Output : PUT (Update data )

 **Swagger**  
OpenAPI 3.0

Select a definition Day4Code-StudentApi v1

## Day4Code-StudentApi 1.0 OAS 3.0

<http://localhost:5149/swagger/v1/swagger.json>

### Students

GET

/api/Students

POST

/api/Students

GET

/api/Students/{id}

PUT

/api/Students/{id}

DELETE

/api/Students/{id}

Parameters

Cancel

| Name                              | Description |
|-----------------------------------|-------------|
| <b>id</b> <small>required</small> |             |
| integer(int32)                    | 4           |
| (path)                            |             |

Execute

Clear

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:5149/api/Students/4' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5149/api/Students/4
```

Server response

| Code | Details  |
|------|--|
| 200  | <div><div>Response body</div><pre>{   "success": true,   "message": "Student deleted successfully" }</pre><div><div>Download</div></div><div>Response headers</div><pre>content-type: application/json; charset=utf-8 date: Tue, 28 Oct 2025 17:34:58 GMT server: Kestrel transfer-encoding: chunked</pre></div> |

Responses

| Code | Description | Links    |
|------|-------------|----------|
| 200  | OK          | No links |

Schemas

StudentDto >

Output : DELETE (Delete data )