Perfect 🔥 — this is a **full end-to-end ASP.NET Core CRUD flow** exactly how it's asked in **interviews, projects, and exams**.

Below are **medium + short + sweet notes** with **clean code snippets** showing:

👉 **Swagger → Controller → Stored Procedure → SQL Server**

🚀 ASP.NET Core CRUD using SQL Server Stored Procedures

(From DB connection to Swagger testing)

🧭 FLOW OVERVIEW (VERY IMPORTANT)

```pgsql
Swagger UI
  ↓
API Controller (ASP.NET Core)
  ↓
SqlConnection + SqlCommand
  ↓
Stored Procedure (SQL Server)
  ↓
Database Table
```

🗄️ STEP 1: SQL Server (Table)

```sql
CREATE TABLE Students
(
    StudentId INT IDENTITY PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100)
);
```

⚙️ STEP 2: SQL Server Stored Procedures (CRUD)

➕ INSERT

```sql
CREATE PROCEDURE InsertStudent
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @Email VARCHAR(100)
AS
BEGIN
    BEGIN TRY
        INSERT INTO Students VALUES (@FirstName, @LastName, @Email)
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE() AS ErrorMessage
    END CATCH
END
```

📖 GET

```sql
CREATE PROCEDURE GetStudents
AS
BEGIN
    SELECT * FROM Students
END
```

✏️ UPDATE

```sql
CREATE PROCEDURE UpdateStudent
    @StudentId INT,
    @Email VARCHAR(100)
AS
BEGIN
    BEGIN TRY
        UPDATE Students
        SET Email = @Email
        WHERE StudentId = @StudentId
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
    END CATCH
END
```

❌ DELETE

```sql
```

```
CREATE PROCEDURE DeleteStudent
    @StudentId INT
AS
BEGIN
    BEGIN TRY
        DELETE FROM Students WHERE StudentId = @StudentId
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
    END CATCH
END
```

🔌 STEP 3: Connection String (`appsettings.json`)

```json
"ConnectionStrings": {
  "DefaultConnection": "Server=.;Database=SchoolDB;Trusted_Connection=True;"
}
```

🧱 STEP 4: DTO Model (Data Transfer Object)

```csharp
public class StudentDto
{
    public int StudentId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
}
```

🎯 STEP 5: ASP.NET Core Controller (CRUD)

```csharp
[ApiController]
[Route("api/[controller]")]
public class StudentController : ControllerBase
{
    private readonly IConfiguration _configuration;

    public StudentController(IConfiguration configuration)
    {
        _configuration = configuration;
    }
```

➕ INSERT (Swagger → Body → Stored Procedure)

```csharp
[HttpPost("insert")]
public IActionResult Insert(StudentDto student)
{
    using SqlConnection con =
        new SqlConnection(_configuration.GetConnectionString("DefaultConnection"));

    SqlCommand cmd = new SqlCommand("InsertStudent", con);
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.AddWithValue("@FirstName", student.FirstName);
    cmd.Parameters.AddWithValue("@LastName", student.LastName);
    cmd.Parameters.AddWithValue("@Email", student.Email);

    con.Open();
    cmd.ExecuteNonQuery();

    return Ok("Student Inserted Successfully");
}
```

📖 GET (Swagger → GET Method)

```csharp
[HttpGet("get")]
public IActionResult Get()
{
    using SqlConnection con =
        new SqlConnection(_configuration.GetConnectionString("DefaultConnection"));

    SqlCommand cmd = new SqlCommand("GetStudents", con);
    cmd.CommandType = CommandType.StoredProcedure;

    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return Ok(dt);
}
```

✏️ UPDATE

```csharp
[HttpPut("update")]
public IActionResult Update(StudentDto student)
{
    using SqlConnection con =
        new SqlConnection(_configuration.GetConnectionString("DefaultConnection"));

    SqlCommand cmd = new SqlCommand("UpdateStudent", con);
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.AddWithValue("@StudentId", student.StudentId);
    cmd.Parameters.AddWithValue("@Email", student.Email);

    con.Open();
    cmd.ExecuteNonQuery();

    return Ok("Student Updated");
}
```

❌ DELETE

```csharp
[HttpDelete("delete/{id}")]
public IActionResult Delete(int id)
{
    using SqlConnection con =
        new SqlConnection(_configuration.GetConnectionString("DefaultConnection"));

    SqlCommand cmd = new SqlCommand("DeleteStudent", con);
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.AddWithValue("@StudentId", id);

    con.Open();
    cmd.ExecuteNonQuery();

    return Ok("Student Deleted");
}
```

🧪 STEP 6: Swagger Testing

POST (Insert)

```json
{
  "firstName": "Guru",
  "lastName": "Raj",
  "email": "guru@gmail.com"
}
```

PUT (Update)

```json
{
  "studentId": 1,
  "email": "newmail@gmail.com"
}
```

DELETE

```swift
/api/student/delete/1
```

🛡️ Where TRY–CATCH Should Be Used?

Layer	TRY–CATCH
Stored Procedure	✅ Mandatory
Controller	Optional
SELECT	Optional

✍️ SHORT NOTES (Exam / Interview)

- Swagger passes JSON → Controller
- Controller calls Stored Procedure

- Stored Procedure executes SQL logic
- INSERT/UPDATE/DELETE must use TRY–CATCH
- DTO is used to receive Swagger input
- SqlCommand + CommandType.StoredProcedure

🍬 SWEET NOTES (One-Liners)

- Swagger → API → Stored Procedure → SQL
- DTO carries request data
- Stored Procedures improve security
- TRY–CATCH prevents SQL failures
- SqlConnection handles DB communication

If you want next, I can give you:

✅ **Exception handling with proper HTTP codes**
✅ **Async/await version**
✅ **Repository pattern**
✅ **Real project folder structure**
✅ **Interview Q&A (🔥 very important)**

Just say the word 😄