**Practical - 6**

**CRUD Operation using SQL Server DB and Entity Framework**

**Aim :** Create Web API for employee CRUD Operation using

SQL Server DB and Entity Framework **Code :**

**EmployeesController.cs :**

using EmployeeAPI.Data; using web\_api.Models; using Microsoft.AspNetCore.Mvc; using Microsoft.EntityFrameworkCore;

namespace EmployeeAPI.Controllers

{

[Route("api/[controller]")]

[ApiController] public class EmployeesController : ControllerBase

{ private readonly EmployeeDbContext \_context;

public EmployeesController(EmployeeDbContext context)

{

\_context = context;

}

// GET: api/employees

[HttpGet] public async Task<ActionResult<IEnumerable<Employee>>> GetEmployees() {

return await \_context.Employees.ToListAsync();

}

// GET: api/employees/{id} [HttpGet("{id}")] public async Task<ActionResult<Employee>> GetEmployee(int id)

{ var employee = await \_context.Employees.FindAsync(id); if (employee == null)

{ return NotFound();

} return employee;

}

// POST: api/employees

[HttpPost] public async Task<ActionResult<Employee>> PostEmployee(Employee employee)

{

\_context.Employees.Add(employee); await \_context.SaveChangesAsync(); return CreatedAtAction(nameof(GetEmployee), new { id = employee.Id },

employee);

}

// PUT: api/employees/{id} [HttpPut("{id}")] public async Task<IActionResult> PutEmployee(int id, Employee employee)

{ if (id != employee.Id)

{ return BadRequest();

}

\_context.Entry(employee).State = EntityState.Modified;

try

{ await \_context.SaveChangesAsync();

} catch (DbUpdateConcurrencyException)

{ if (!EmployeeExists(id))

{ return NotFound();

} throw; }

return NoContent();

}

// DELETE: api/employees/{id} [HttpDelete("{id}")] public async Task<IActionResult> DeleteEmployee(int id)

{ var employee = await \_context.Employees.FindAsync(id); if (employee == null)

{ return NotFound();

}

\_context.Employees.Remove(employee); await \_context.SaveChangesAsync(); return NoContent();

}

private bool EmployeeExists(int id) => \_context.Employees.Any(e => e.Id == id);

}

}

**EmployeeDbContext.cs :**

using Microsoft.EntityFrameworkCore; using web\_api.Models;

namespace EmployeeAPI.Data

{ public class EmployeeDbContext : DbContext

{ public EmployeeDbContext(DbContextOptions<EmployeeDbContext> options) : base(options)

{ }

public DbSet<Employee> Employees { get; set; } }

}

**Employee.cs :**

using System.ComponentModel.DataAnnotations; namespace web\_api.Models

{ public class Employee

{ public int Id { get; set; } [Required] [StringLength(50)] public string FirstName { get; set; }

[Required]

[StringLength(50)] public string LastName { get; set; }

[Required] [EmailAddress] public string Email { get; set; }

[Required] [Phone] public string Phone { get; set; }

[Required] [StringLength(50)] public string Position { get; set; }

}

} **Program.cs :**

using EmployeeAPI.Data; using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container. builder.Services.AddDbContext<EmployeeDbContext>(options

=> options.UseSqlServer(builder.Configuration.GetConnectionString("EmployeeDbConnect ion")));

builder.Services.AddControllers(); builder.Services.AddEndpointsApiExplorer(); builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline. if (app.Environment.IsDevelopment())

{ app.UseSwagger(); app.UseSwaggerUI();

}

app.UseAuthorization(); app.MapControllers(); app.Run();

**Appsettings.json :**

{

"ConnectionStrings": {

"EmployeeDbConnection": "Server=ARPIT\_DEVGANIYA;Database=EmployeeDB;Trusted\_Connection=True;Enc rypt=False;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

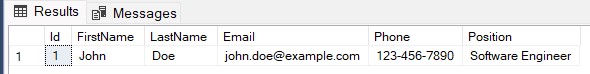
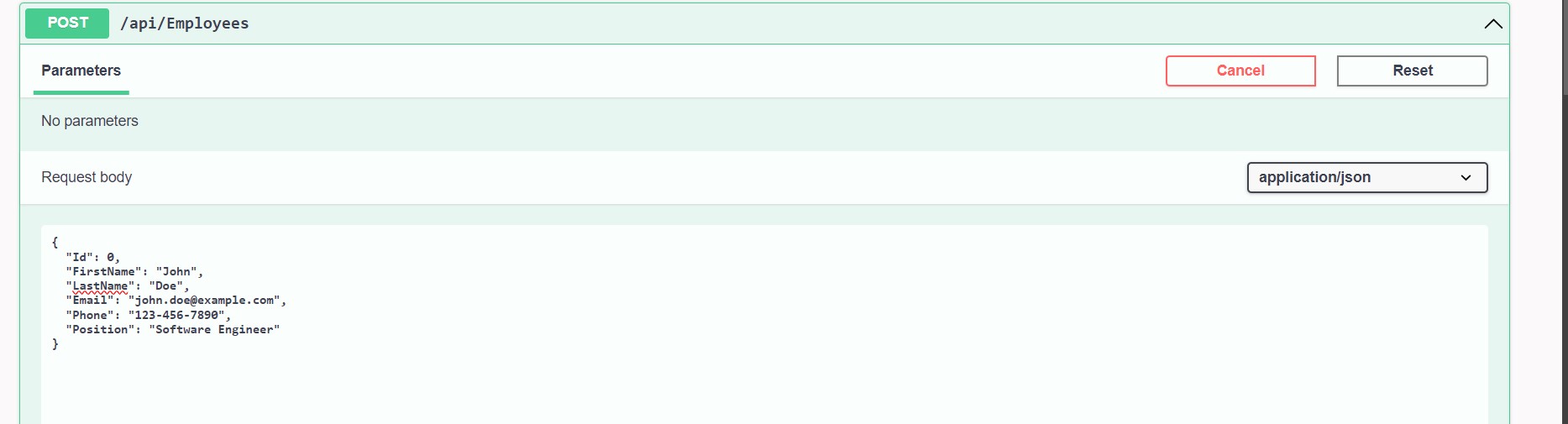
},

"AllowedHosts": "\*"

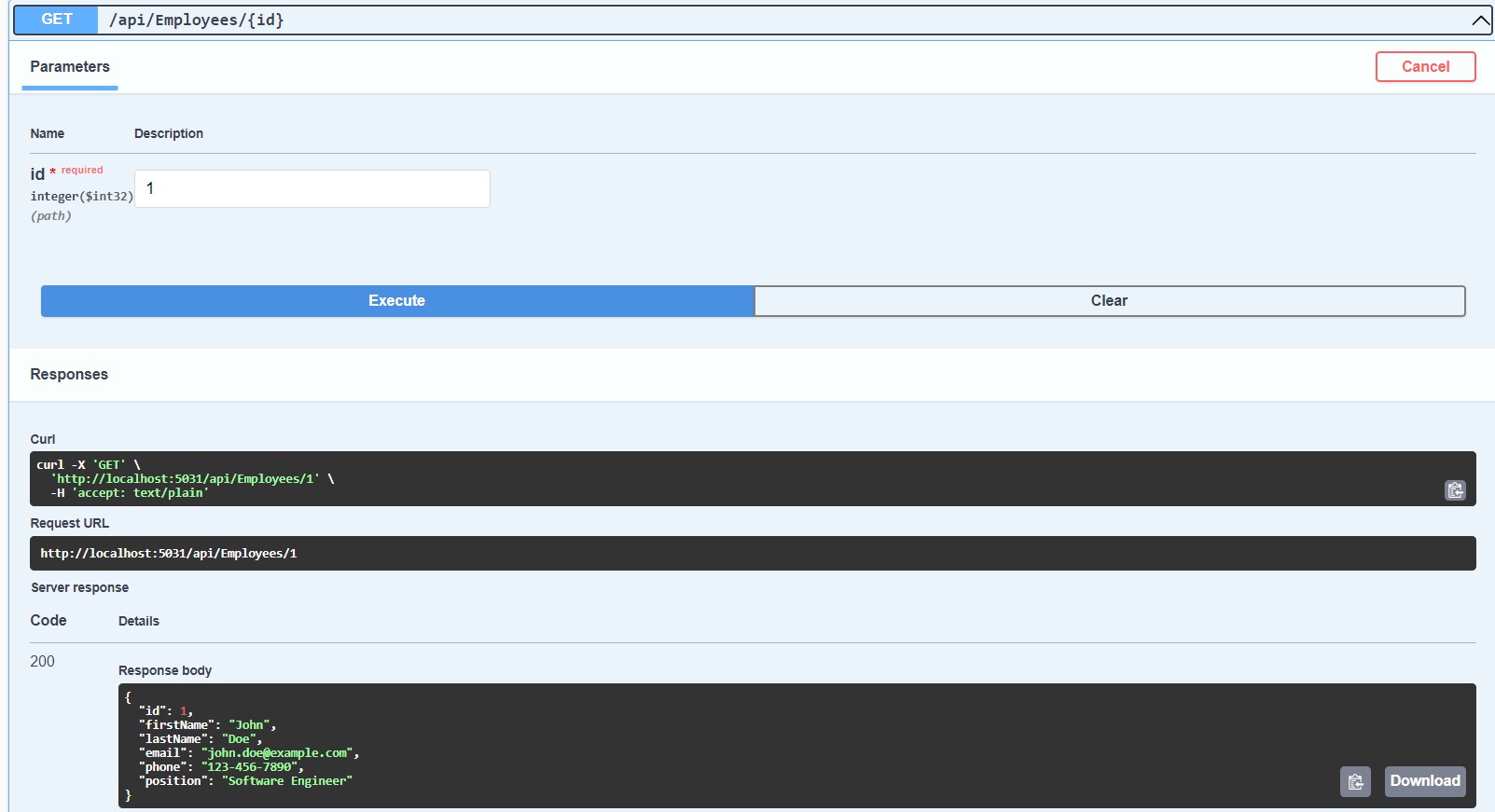
}

**Output :**

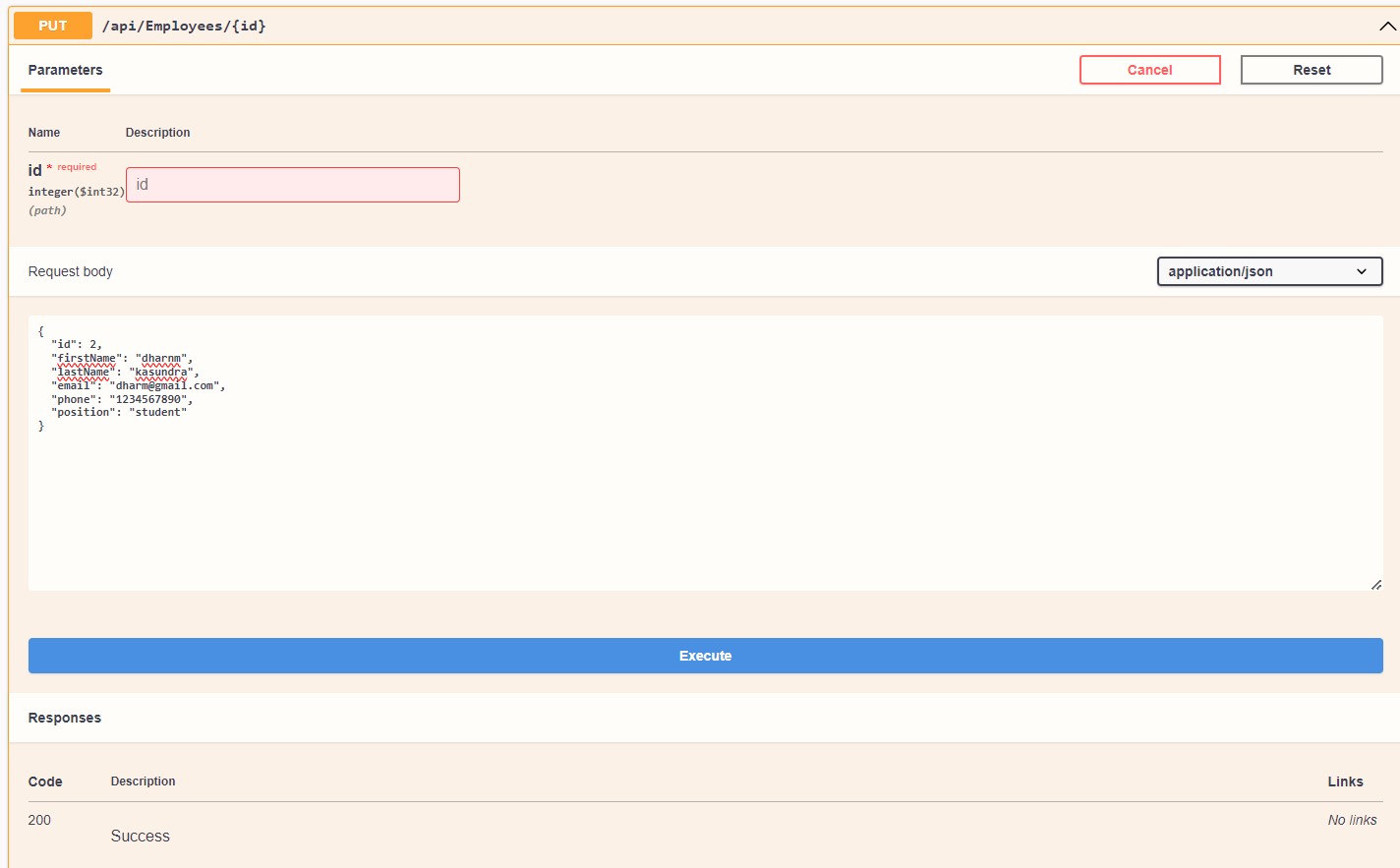
**Post :**



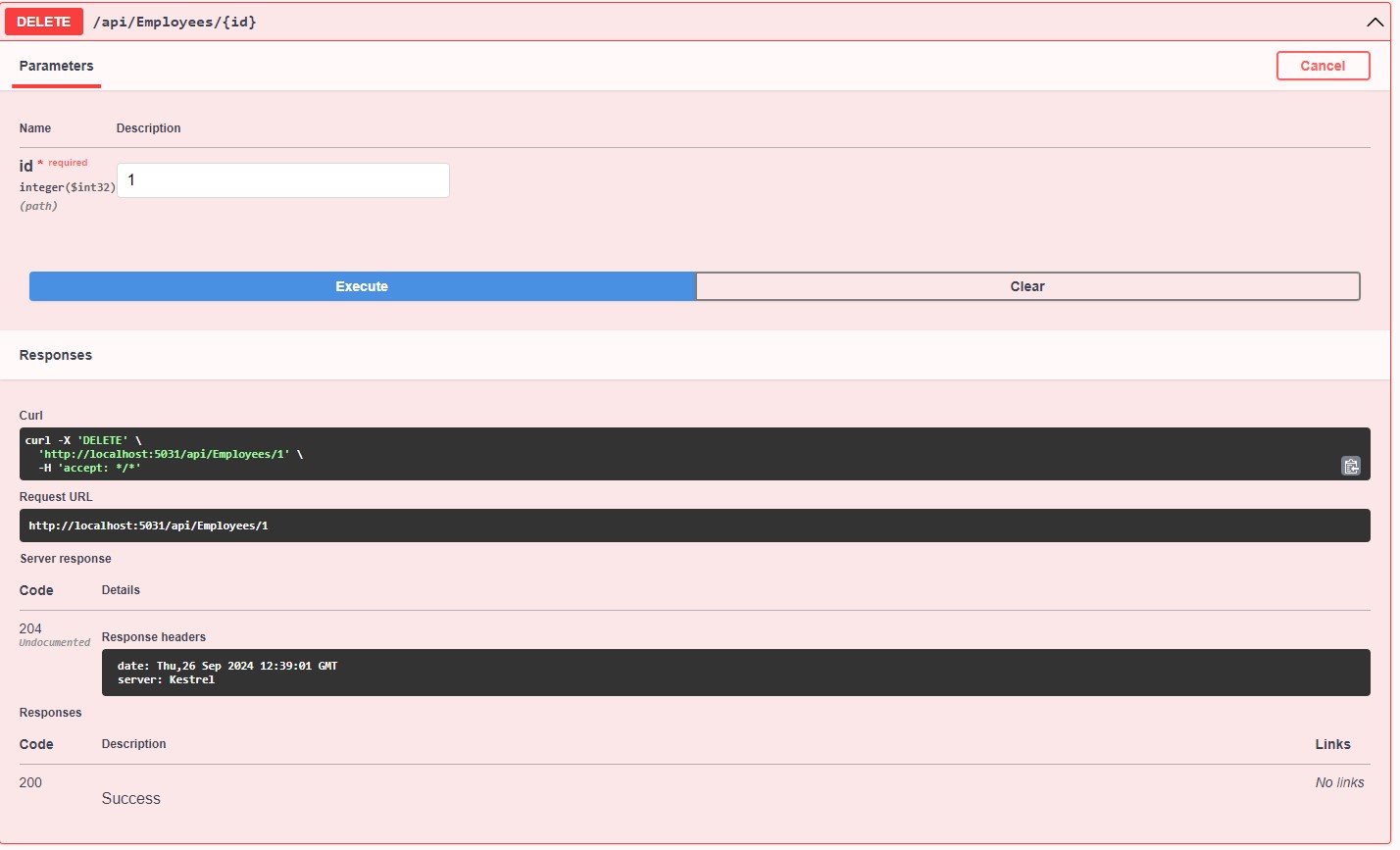
**Get :**



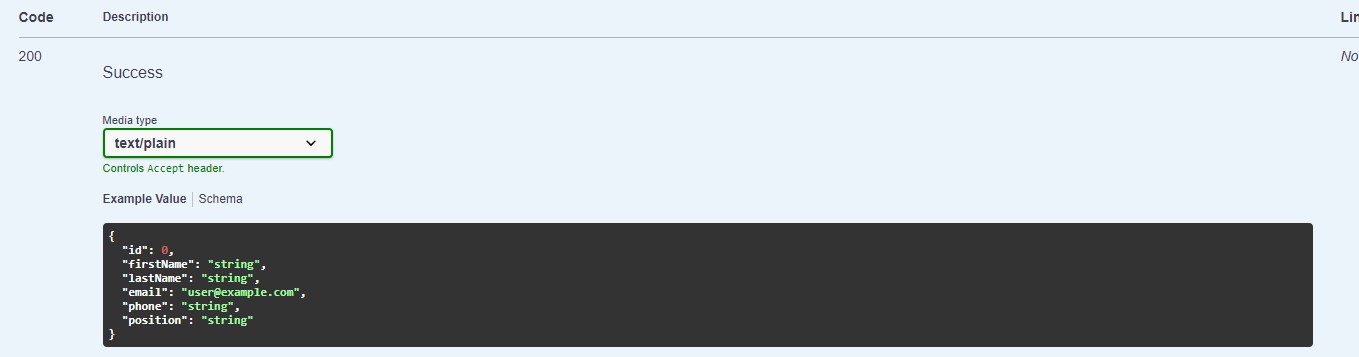
**Put :**

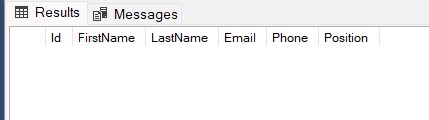


**Delete :**



**Get after delete :**





**Summary:**

**Backend API Development**: Create a .NET 8 Web API to handle CRUD (Create, Read, Update, Delete) operations for a ToDo List. Ensure to test all endpoints for functionality in a database.

**Frontend Design:** Build a responsive user interface using HTML, CSS, and

JavaScript to interact with the backend API, allowing users to manage ToDo tasks.