Step-by-step guide to build ASP.NET Core Web API CRUD operations using **Visual Studio 2022**, **Entity Framework Core**, and **Oracle Database**.

Step 1: Create a New ASP.NET Core Web API Project

1. Open **Visual Studio 2022**
2. Click **Create a new project**
3. Select **ASP.NET Core Web API**
4. Click **Next**
5. Enter project name:

```
ProductsApi
```

6. Choose project location and Keep the default selections
7. Click **Create**

Step 2: Add Required NuGet Packages

Go to **Soluion Explorer** > Right-click **ProjectName** > **Manage NuGet Packages**

Install the following:

- **Microsoft.EntityFrameworkCore (9.0.0)**
- **Microsoft.EntityFrameworkCore.Design (9.0.0)**
- **Microsoft.EntityFrameworkCore.Tools (9.0.0)**
- **Oracle.EntityFrameworkCore (9.23.26000)**

Step 3: Add Database Connection String

Open `appsettings.json` and add:

```
"ConnectionStrings": {
  "DefaultConnection": "User Id=system;Password=admin@123;Data
Source=//localhost:1521/xe;"
}
```

Replace values according to your Oracle setup.

Step 4: Create the Entity Model

Right-click **Models** folder → Add → Class → `Product.cs`

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ProductsApi.Models
{
    public class Product
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(200)]
        public string Name { get; set; } = null!;

        [Column(TypeName = "NUMBER(18,2)")]
        public decimal Price { get; set; }

        [MaxLength(100)]
        public string? Category { get; set; }
    }
}
```

Step 5: Create the DbContext Class

Right-click **Data** folder → Add → Class → `AppDbContext.cs`

```
using Microsoft.EntityFrameworkCore;
using ProductsApi.Models;

namespace ProductsApi.Data
{
    public class AppDbContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options) { }

         protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            foreach (var entity in modelBuilder.Model.GetEntityTypes())
            {
                // TABLE name => UPPERCASE
                entity.SetTableName(entity.GetTableName().ToUpper());
```

```
                // COLUMN names => UPPERCASE
                foreach (var prop in entity.GetProperties())
                {
                    var tableName = entity.GetTableName();
                    var columnName =
prop.GetColumnName(StoreObjectIdentifier.Table(tableName, entity.GetSchema()));

                    prop.SetColumnName(columnName.ToUpper());
                }
            }
        }

    }
}
```

Step 6: Register DbContext in Program.cs

Open `Program.cs` and update it:

```
using Microsoft.EntityFrameworkCore;
using ProductsApi.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services
builder.Services.AddControllers();

// Configure Oracle EF Core
builder.Services.AddDbContext<AppDbContext>(options =>

options.UseOracle(builder.Configuration.GetConnectionString("DefaultConnection")))
;

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

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

Step 7: Create EF Core Migrations (Visual Studio)

Option A: Use Package Manager Console (recommended)

1. View → Other Windows → **Package Manager Console**
2. Run:

```
Add-Migration InitialCreate
```

3. Apply migration:

```
Update-Database
```

EF Core will generate the `Products` table.

Step 8: Create ProductsController (CRUD)

Right-click **Controllers** → Add → Controller

Choose:

- `API Controller - Empty`
- Name : ProductsController.cs

API Controller – `ProductsController.cs`:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ProductsApi.Data;
using ProductsApi.Models;

namespace ProductsApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
    {
        private readonly AppDbContext _context;

        public ProductsController(AppDbContext context)
        {
            _context = context;
        }
```

```
        // GET: api/products
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Product>>> GetAll()
        {
            return await _context.Products.ToListAsync();
        }

        // GET: api/products/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Product>> Get(int id)
        {
            var product = await _context.Products.FindAsync(id);
            if (product == null)
                return NotFound();

            return product;
        }

        // POST: api/products
        [HttpPost]
        public async Task<ActionResult<Product>> Post(Product product)
        {
            _context.Products.Add(product);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
        }

        // PUT: api/products/5
        [HttpPut("{id}")]
        public async Task<IActionResult> Put(int id, Product product)
        {
            if (id != product.Id)
                return BadRequest();

            _context.Entry(product).State = EntityState.Modified;

               await _context.SaveChangesAsync();
               return Ok(new { message = "Resource updated in server" });
        }

        // DELETE: api/products/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(int id)
        {
            var product = await _context.Products.FindAsync(id);
            if (product == null)
                return NotFound();

            _context.Products.Remove(product);
            await _context.SaveChangesAsync();

             return Ok(new { message = "Resource deleted from server" });
```

```
        }
    }
}
```

Step 9: Run and Test the API

1. Press **F5** or click **Run**
2. Browser opens
3. Navigate to **/swagger**
4. Use Swagger UI to test operations:
   GET POST PUT DELETE
5. POST request (from Swagger)**

```
{
  "name": "Laptop",
  "price": 750.00,
  "category": "Electronics"
}
```

Step 10: Verify in Oracle Database

Execute:

```
SELECT * FROM Products;
```

Rows should appear after testing Swagger POST.
