Nice — this is a solid microservices + API Gateway assignment. I put together a ready-to-use blueprint you can copy into your two projects (SalesService and ApiGatewayService) so the app runs on ports **8080** and **8081**, supports EF Core migrations, and routes /sales-api/sales through Ocelot to the SalesService.

Below you’ll find:

* file-by-file code for **SalesService** (model, DbContext, controller, Program, launch settings, EF config, Swagger)
* file-by-file code for **ApiGatewayService** (Program, ocelot.json, launch settings)
* commands to create/run/migrate DB and start both services
* notes & troubleshooting tips

I assumed .NET 6 (your dotnet-ef --version 6.0.6 hint). If you use a different SDK target, adjust TargetFramework accordingly.

**1) SalesService (port 8080)**

**Project dependencies**

Add to SalesService.csproj:

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net6.0</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="6.0.16" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="6.0.16">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

</PackageReference>

<PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />

</ItemGroup>

</Project>

(Adjust versions to match your environment if needed.)

**Models/Sale.cs**

namespace SalesService.Models

{

public class Sales

{

public int SalesId { get; set; } // auto-increment identity by EF

public string Name { get; set; } = null!;

public string? Description { get; set; }

public decimal Price { get; set; }

}

}

**Data/SalesDbContext.cs**

using Microsoft.EntityFrameworkCore;

using SalesService.Models;

namespace SalesService.Data

{

public class SalesDbContext : DbContext

{

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

public DbSet<Sales> Sales { get; set; } = null!;

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Sales>(b =>

{

b.HasKey(s => s.SalesId);

b.Property(s => s.SalesId).ValueGeneratedOnAdd();

b.Property(s => s.Name).IsRequired();

b.Property(s => s.Price).HasColumnType("decimal(18,2)");

});

base.OnModelCreating(modelBuilder);

}

}

}

**Controllers/SalesController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SalesService.Data;

using SalesService.Models;

namespace SalesService.Controllers

{

[ApiController]

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

public class SalesController : ControllerBase

{

private readonly SalesDbContext \_context;

public SalesController(SalesDbContext context)

{

\_context = context;

}

// GET: /api/sales

[HttpGet]

public async Task<ActionResult<IEnumerable<Sales>>> GetSales()

{

var salesList = await \_context.Sales.ToListAsync();

return Ok(salesList);

}

// POST: /api/sales

[HttpPost]

public async Task<ActionResult> CreateSales([FromBody] Sales sales)

{

\_context.Sales.Add(sales);

await \_context.SaveChangesAsync();

var response = new

{

message = "Sale created successfully",

data = sales

};

return Created(string.Empty, response);

}

}

}Notes:

* The POST returns a custom JSON object { message, data } and 201 Created (with CreatedAtAction).
* CreatedAtAction uses GetSales path — if you want Get by id, add a GetById(int id) method and change CreatedAtAction accordingly.

**Program.cs (minimal .NET 6 style)**

using Microsoft.EntityFrameworkCore;

using SalesService.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Configure EF Core - use connection string from appsettings

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")

?? "User ID=sa;password=examlyMssql@123;server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False";

builder.Services.AddDbContext<SalesDbContext>(options =>

options.UseSqlServer(connectionString));

var app = builder.Build();

// Configure middleware

if (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseSwagger();

app.UseSwaggerUI(c =>

{

// make swagger root available at /swagger/index.html

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Sales API V1");

});

app.UseRouting();

app.UseAuthorization();

app.MapControllers();

app.Run();

**appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "User ID=sa;password=examlyMssql@123;server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False"

}

}

**Properties/launchSettings.json (ensure port 8080)**

{

"profiles": {

"SalesService": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "swagger",

"applicationUrl": "http://localhost:8080",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

**2) ApiGatewayService (port 8081)**

**Project dependencies**

Add to ApiGatewayService.csproj:

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net6.0</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Ocelot" Version="18.0.0" />

<PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="6.0.0" />

</ItemGroup>

</Project>

(Use compatible Ocelot version for .NET 6; adjust if needed.)

**ocelot.json (place at root of ApiGatewayService project)**

{

"Routes": [

{

"DownstreamPathTemplate": "/api/sales",

"DownstreamScheme": "http",

"DownstreamHostAndPorts": [

{

"Host": "localhost",

"Port": 8080

}

],

"UpstreamPathTemplate": "/sales-api/sales",

"UpstreamHttpMethod": [ "GET", "POST" ]

},

{

"DownstreamPathTemplate": "/api/sales/{everything}",

"DownstreamScheme": "http",

"DownstreamHostAndPorts": [

{

"Host": "localhost",

"Port": 8080

}

],

"UpstreamPathTemplate": "/sales-api/sales/{everything}",

"UpstreamHttpMethod": [ "GET", "POST", "PUT", "DELETE" ]

}

],

"GlobalConfiguration": {

"RequestIdKey": "OcRequestId",

"BaseUrl": "http://localhost:8081"

}

}

Notes:

* The first route maps /sales-api/sales to /api/sales on SalesService.
* The {everything} route helps pass through path segments if you later add /api/sales/{id}.

**Program.cs for ApiGatewayService**

using Ocelot.DependencyInjection;

using Ocelot.Middleware;

var builder = WebApplication.CreateBuilder(args);

// Tell Ocelot where to find its config

builder.Configuration.AddJsonFile("ocelot.json", optional: false, reloadOnChange: true);

builder.Services.AddOcelot(builder.Configuration);

var app = builder.Build();

app.MapGet("/", () => Results.Redirect("/swagger", permanent: false)); // optional

await app.UseOcelot();

app.Run();

**Properties/launchSettings.json (set port 8081)**

{

"profiles": {

"ApiGatewayService": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": false,

"applicationUrl": "http://localhost:8081",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

**3) EF Core / DB Setup & Migrations (SalesService)**

You already noted the dotnet tool steps. Consolidated:

1. In SalesService project root:

dotnet new tool-manifest # (if not already created)

dotnet tool install --local dotnet-ef --version 6.0.6

1. Add migrations & update database:

# Make sure appsettings has correct connection string and SQL Server is running

dotnet dotnet-ef migrations add InitialSetup -p . -s .

dotnet dotnet-ef database update -p . -s .

* -p is project (if you run from project root you can omit).
* -s is startup project; if SalesService is both, it's fine.

1. If using SQLCMD to create DB manually (your notes):

# open new terminal

sqlcmd -U sa

# when prompted enter password: examlyMssql@123

1> create database appdb

2> go

# then exit or use it

Connection string you provided works:

User ID=sa;password=examlyMssql@123; server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False

**4) Running both services (order matters)**

1. Build both:

# from each project

cd dotnetapp/SalesService

dotnet restore

dotnet build

1. Run SalesService (ensure it is on port 8080):

cd dotnetapp/SalesService

dotnet run

# should bind to http://localhost:8080

# open http://localhost:8080/swagger/index.html to verify endpoints

1. Run ApiGatewayService:

cd dotnetapp/ApiGatewayService

dotnet restore

dotnet run

# binds to http://localhost:8081

1. Test via API Gateway:

* GET sales via gateway:
  + GET http://localhost:8081/sales-api/sales
* POST sale via gateway:
  + POST http://localhost:8081/sales-api/sales
  + Body:

{

"name":"Test Sale",

"description":"desc",

"price": 199.99

}

* It should route to SalesService and return 201 Created and the custom JSON.

If your environment uses different hostnames (examly.io preview URLs), Ocelot DownstreamHostAndPorts should refer to the target host reachable from gateway. For local dev, localhost works.

**5) Swagger endpoint note / exam preview link**

You included a preview URL like:  
https://8080-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.premiumproject.examly.io/swagger/index.html

If you want the production/exam runner to view the Swagger page, ensure the SalesService is actually listening on port 8080 (and the online preview host routes that port correctly). The applicationUrl in launchSettings.json needs to match the exam runner's expectation (they often enforce port numbers).

**6) Extra helpful tips & small improvements**

* Add GetById:

[HttpGet("{id:int}")]

public async Task<ActionResult<Sales>> GetById(int id)

{

var item = await \_db.Sales.FindAsync(id);

if (item == null) return NotFound();

return Ok(item);

}

Then change CreatedAtAction(nameof(GetById), new { id = sales.SalesId }, response) for more RESTful location header.

* CORS: If frontend or external caller, enable CORS in SalesService and ApiGatewayService.
* Logging: check app logs to verify Ocelot forwarding and downstream responses.
* Health checks: add a small endpoint /health on SalesService and route it in Ocelot if required.

**7) Troubleshooting common issues**

* **Port in use**: If 8080 / 8081 already used, stop process or change port in launchSettings.json but exam runner expects those exact ports — keep as required.
* **SQL connection failures**: Ensure SQL Server is running and the sa credentials match. If using Docker SQL Server, map ports and set ACCEPT\_EULA and SA password env variables.
* **Ocelot 500 errors**: Check if SalesService is up before calling via gateway. Check gateway logs; increase logging verbosity for Ocelot with Logging configuration in appsettings.json if needed.
* **Swagger behind Ocelot**: Ocelot won't automatically show downstream Swagger; access SalesService Swagger directly at port 8080.

If you want, I can:

* produce a ZIP-style project structure (code files) you can paste,
* add a GetById endpoint and adjust created response to CreatedAtAction(GetById, ...),
* generate Docker Compose to run SalesService + SQLServer + ApiGateway with the correct exposed ports,
* or create a simple Postman collection for testing.

Tell me which of the above you want and I’ll produce the files (or Docker Compose) directly — otherwise you can copy the code above into your two projects and run the steps under **Running both services**.

**You said:**

Test Case Evaluation Type Match Result SalesService\_8080\_PostSales NUnit NA Failure SalesService\_8080\_GetSales NUnit NA Failure ApiGatewayService\_8081\_PostSales NUnit NA Failure ApiGatewayService\_8081\_GetSales NUnit NA Failure Sales\_Properties\_SalesId\_ReturnExpectedDataTypes\_int NUnit NA Success Sales\_Properties\_Price\_ReturnExpectedDataTypes\_decimal NUnit NA Success Sales\_Properties\_Name\_ReturnExpectedDataTypes\_String NUnit NA Success Sales\_Properties\_Description\_ReturnExpectedDataTypes\_String NUnit NA Success