Here’s a **complete guide and reference implementation** for your **ASP.NET Core Web API project** — *Online Movie Store with Serilog Logging*.  
It includes all required files, controller logic, model definition, and logging configuration.

**Project Overview**

You’ll create a simple ASP.NET Core Web API called **dotnetapp** that:

* Manages movie data using MovieController.
* Uses **Serilog** for logging user actions and errors.
* Interacts with a SQL Server database using **Entity Framework Core**.

**Step 1: Create the Project**

dotnet new webapi -n dotnetapp

cd dotnetapp

**Step 2: Install Required Packages**

Install Entity Framework Core and Serilog packages:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Serilog.AspNetCore

dotnet add package Serilog.Sinks.File

**Step 3: Create the Movie Model**

Create file: Models/Movie.cs

namespace dotnetapp.Models

{

public class Movie

{

public int Id { get; set; } // Auto-incremented

public string Title { get; set; }

public string Director { get; set; }

public decimal Price { get; set; }

}

}

**Step 4: Create the Database Context**

Create file: Data/MovieDbContext.cs

using Microsoft.EntityFrameworkCore;

using dotnetapp.Models;

namespace dotnetapp.Data

{

public class MovieDbContext : DbContext

{

public MovieDbContext(DbContextOptions<MovieDbContext> options)

: base(options) { }

public DbSet<Movie> Movies { get; set; }

}

}

**Step 5: Configure Database Connection (appsettings.json)**

Update your connection string:

{

"ConnectionStrings": {

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

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning"

}

},

"AllowedHosts": "\*"

}

**Step 6: Configure Serilog (Program.cs)**

Replace the entire Program.cs with:

using dotnetapp.Data;

using Microsoft.EntityFrameworkCore;

using Serilog;

var builder = WebApplication.CreateBuilder(args);

// Configure Serilog for file logging

Log.Logger = new LoggerConfiguration()

.WriteTo.File("logs/movieapp\_log.txt", rollingInterval: RollingInterval.Day)

.CreateLogger();

builder.Host.UseSerilog();

// Add services

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

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

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

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

**Step 7: Create the MovieController**

Create file: Controllers/MovieController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using dotnetapp.Data;

using dotnetapp.Models;

using Serilog;

namespace dotnetapp.Controllers

{

[ApiController]

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

public class MovieController : ControllerBase

{

private readonly MovieDbContext \_context;

public MovieController(MovieDbContext context)

{

\_context = context;

}

// GET: api/Movie

[HttpGet]

public async Task<IActionResult> Index()

{

try

{

var movies = await \_context.Movies.ToListAsync();

Log.Information("Fetched all movies successfully. Count: {Count}", movies.Count);

return Ok(movies);

}

catch (Exception ex)

{

Log.Error(ex, "Error fetching movie list.");

return StatusCode(500, "Internal server error");

}

}

// GET: api/Movie/search?query=Inception

[HttpGet("search")]

public async Task<IActionResult> Search(string query)

{

try

{

var result = await \_context.Movies

.Where(m => m.Title.Contains(query))

.ToListAsync();

if (!result.Any())

{

Log.Warning("No movies found for search query: {Query}", query);

return NotFound("No movies found");

}

Log.Information("Search successful for query: {Query}", query);

return Ok(result);

}

catch (Exception ex)

{

Log.Error(ex, "Error during search for query: {Query}", query);

return StatusCode(500, "Search failed");

}

}

// GET: api/Movie/details/1

[HttpGet("details/{id}")]

public async Task<IActionResult> Details(int id)

{

try

{

var movie = await \_context.Movies.FindAsync(id);

if (movie == null)

{

Log.Warning("Movie with ID {Id} not found", id);

return NotFound("Movie not found");

}

Log.Information("Details retrieved for movie ID: {Id}", id);

return Ok(movie);

}

catch (Exception ex)

{

Log.Error(ex, "Error fetching details for movie ID: {Id}", id);

return StatusCode(500, "Error retrieving details");

}

}

// POST: api/Movie/purchase/1

[HttpPost("purchase/{id}")]

public async Task<IActionResult> Purchase(int id)

{

try

{

var movie = await \_context.Movies.FindAsync(id);

if (movie == null)

{

Log.Warning("Purchase failed - movie ID {Id} not found", id);

return NotFound("Movie not found");

}

Log.Information("Movie purchased successfully - ID: {Id}, Title: {Title}", movie.Id, movie.Title);

return Ok($"Movie '{movie.Title}' purchased successfully!");

}

catch (Exception ex)

{

Log.Error(ex, "Error processing purchase for movie ID: {Id}", id);

return StatusCode(500, "Error processing purchase");

}

}

}

}

**Step 8: Set Port to 8080**

Edit Properties/launchSettings.json:

"applicationUrl": "http://0.0.0.0:8080"

**Step 9: Run EF Core Migrations**

dotnet new tool-manifest

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

dotnet dotnet-ef migrations add "InitialSetup"

dotnet dotnet-ef database update

**Step 10: Run the Application**

dotnet build

dotnet run

Then visit:

* http://localhost:8080/swagger (for API testing)
* Check logs in: dotnetapp/logs/movieapp\_log.txt

**Step 11: Verify Logging**

Examples of log entries (logs/movieapp\_log.txt):

2025-10-19 19:55:23 [INF] Fetched all movies successfully. Count: 3

2025-10-19 19:56:02 [WRN] No movies found for search query: Titanic

2025-10-19 19:57:10 [INF] Movie purchased successfully - ID: 2, Title: Inception