✨ Member-only story

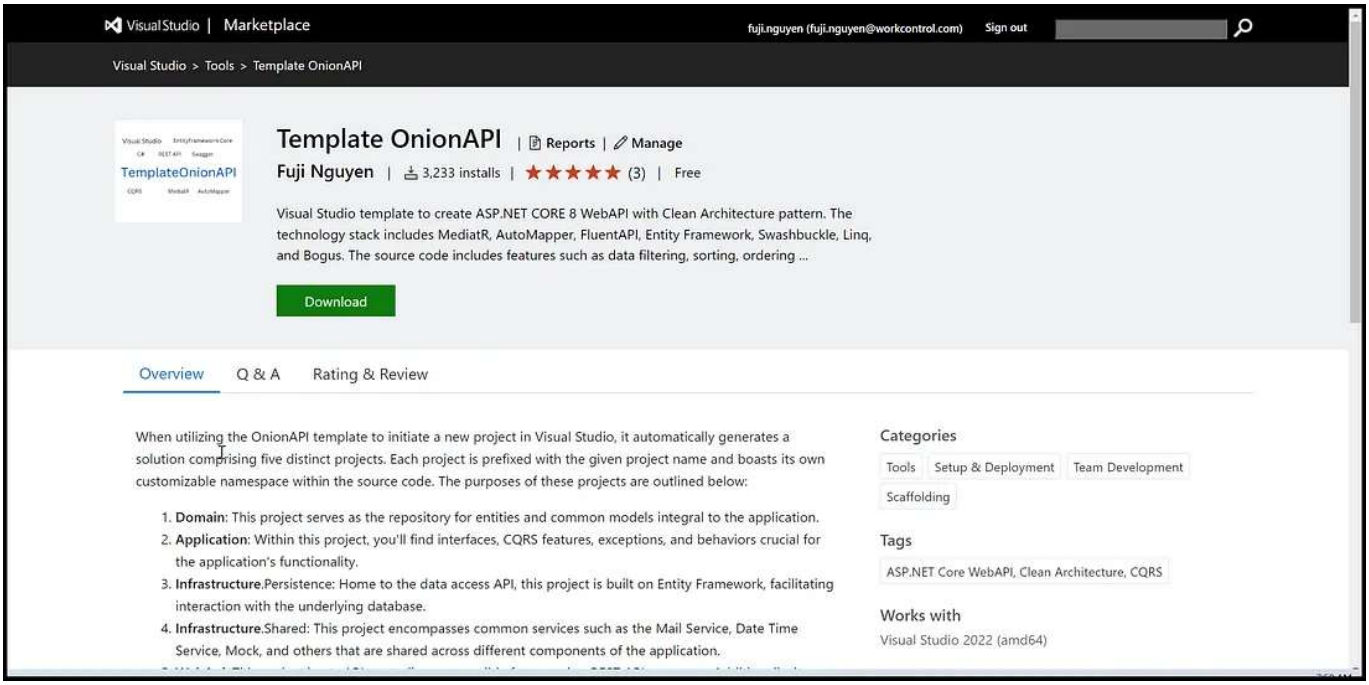
Get started with the Visual Studio template OnionAPI to scaffold ASP.NET WebAPI project for .NET 8

Fuji Nguyen · Follow

Published in Scrum and Coke · 10 min read · Jun 2, 2023

> *If you're in search of a recipe for constructing sophisticated REST APIs, your answer lies in the Template OnionAPI. It comes equipped with a variety of convenient features and adheres to best practices, enabling you to swiftly launch your project. In fact, using the OnionAPI template can trim down your coding time by a minimum of 40 hours compared to initiating a REST API project from the ground up. That's a substantial amount of time that you can engage in non-tech related activities such as reading books, playing musical instruments, or*

> *pursuing other hobbies helps in relaxation and mental rejuvenation.*
>
> **— Fuji Nguyen, Author of OnionAPI Template**



The Template OnionAPI, available on the Visual Studio Marketplace, has been downloaded 3,233 times as of the latest data.

Introduction

NET version 8 was officially released on November 14, 2023. NET 8 delivers **thousands of performance, stability, and security improvements**, as well as platform and tooling enhancements that help increase developer productivity and speed of innovation.

Developers can use the new release of Visual Studio template OnionAPI v8 to scaffold NET 8 REST API projects based on the Clean Architecture pattern as described in **Figure 1**.

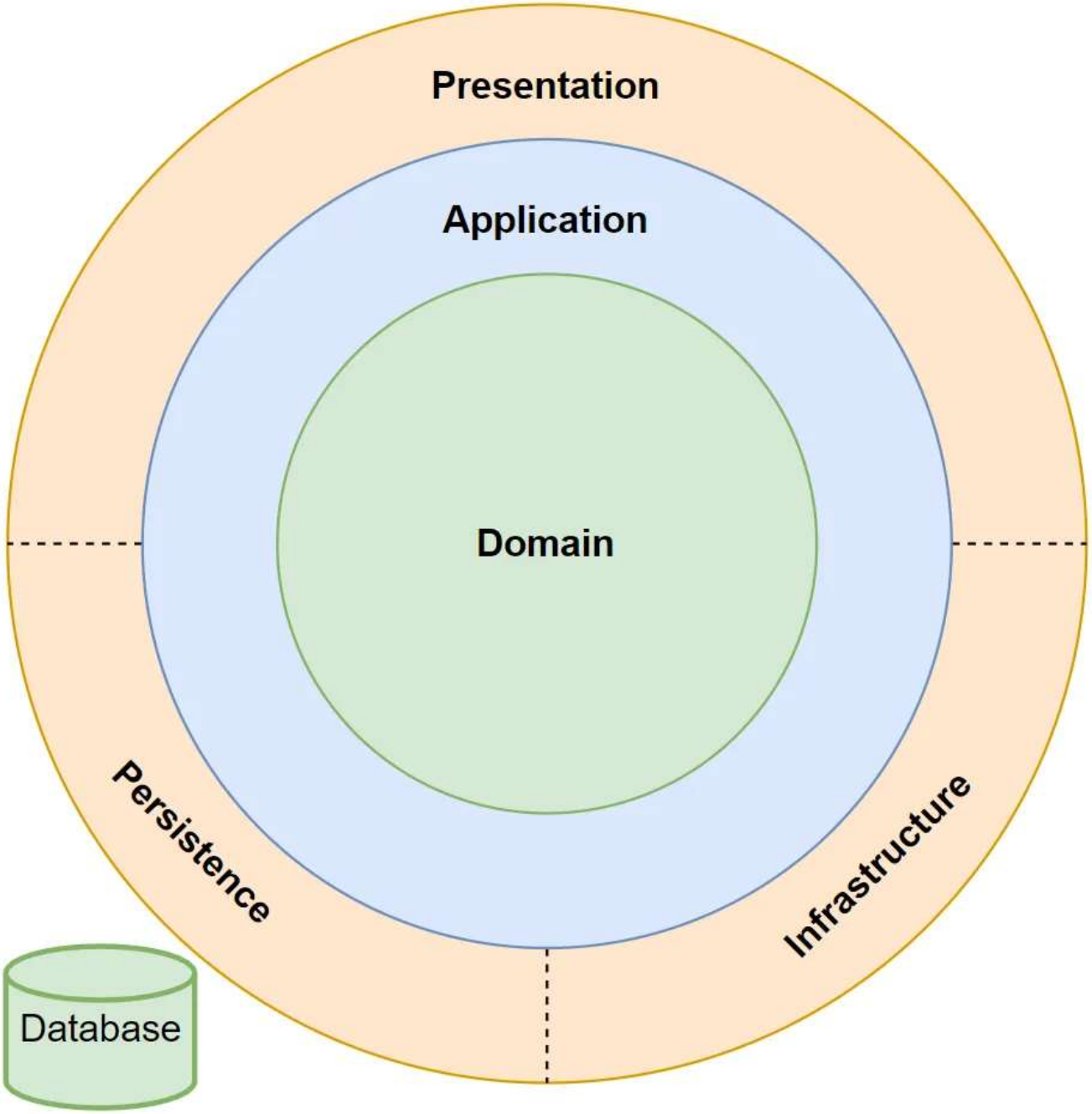


Figure 1- Clean Architecture Onion View

When you use the OnionAPI template to create a new project in Visual Studio, it will generate a solution with five projects inside. Each project is prefixed with the project name and has its own customizable namespace in the source code. Here's what each project is for:

1. **Domain:** This project contains entities and common models.

2. **Application:** This project includes interfaces, CQRS features, exceptions, and behaviors.
3. **Infrastructure.Persistence:** This project is home to the data access API, which is based on Entity Framework.
4. **Infrastructure.Shared:** This project includes common services like the Mail Service, Date Time Service, Mock, and more.
5. **WebApi:** This project is where you'll find API controllers to expose REST API resources and endpoints that are accessible via Swagger.

Content

The tutorial contains 4 parts:

Part 1: Download and install the template
In this part, you'll learn how to download and install the OnionAPI template for Visual Studio.

Part 2: Scaffold a new Web API project
In this part, you'll use the OnionAPI template to create a new ASP.NET CORE Web API project.

Part 3: Get familiar with the generated source code
In this part, you'll take a closer look at the source code that was generated by the template.

Part 4: Test run the Web API project
In this final part, you'll test out the Web API project to make sure everything is working correctly.

Prerequisites

1. Visual Studio version 2022 v17.8.1 or higher (.NET 8 support)
2. Familiar with CAT architecture pattern for modern app SPA/Mobile

Part 1: Download and install the template

To download and install a Visual Studio template from the Visual Studio Marketplace, follow these steps:

1. Open Visual Studio 2022 and go to the “Extensions” menu.
2. Select “Extensions > Manage Extensions” from the menu.
3. In the “Manage Extensions” window, select the “Online” tab on the left side.
4. Use the search bar to find the template you want to download. Search for “*template onionapi*”. See **Figure 2** for visual aids.
5. When you find the template, click the “Download” button.
6. Visual Studio will download and install the template. You may need to restart Visual Studio for the template to be available.

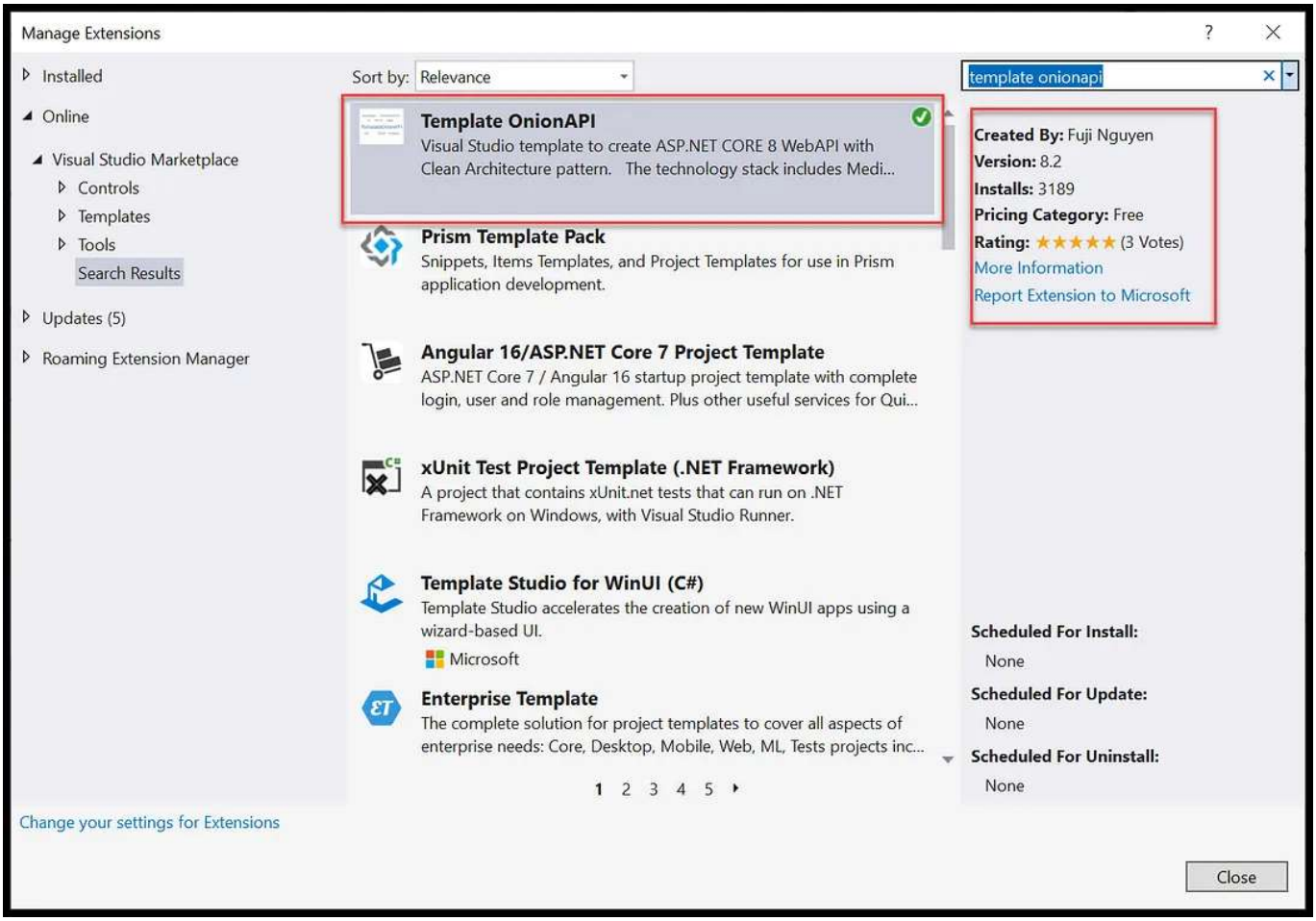


Figure 2 — Search for the Visual Studio template OnionAPI and select Download

Part 2: Scaffold a new Web API project

To use the Visual Studio template to scaffold a new Web API project, follow these steps:

1. Open Visual Studio and select *Create a new project* from the start page or the File menu.
2. In the *Create a new project* dialog, search and select the *OnionAPI Template* template and click **Next**. See **Figure 3** for visual aids
3. In the *Configure your new project* dialog, enter a name for your project (ex. *MyNetcore8WebAPI*) and select a location to save it. Then, click **Create**.

4. The template for Onion API will create a new ASP.NET Core Web API project that follows the Clean Architecture.

Figure 3 — Search and select the template OnionAPI

Part 3: Get familiar with the generated source code

Figure 4 contains a screenshot of the source code scaffolded by the OnionAPI template for the sample project *MyNetcore8WebAPI*. The screenshot depicts the organization of 5 projects, grouped into 3 folders *Core*, *Infrastructure*, and *Presentation*, which are corresponding to the layers described in the Clean Architecture pattern.

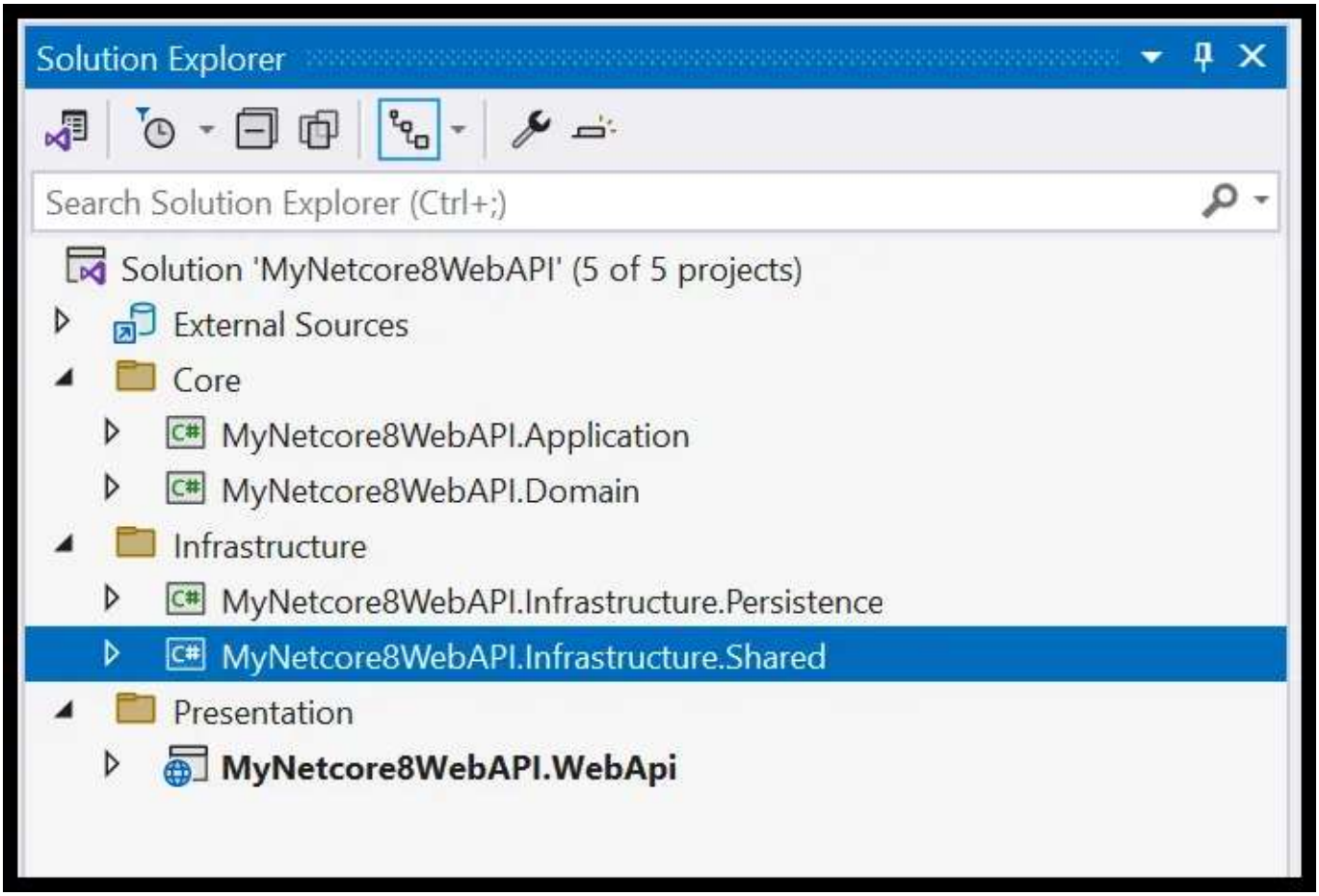


Figure 4 — Scaffolded source code organization within the Visual Studio solution

It can be a bit overwhelming at first, with all those separate projects and layers of dependencies. Don't worry. Below is the quick Google map navigation (lol) to help you find your way.

1. Are you wondering where all the API controllers are located? They're nestled snugly in the *Controllers* folder inside the *WebAPI* project. Need a visual aid? Just check out **Figure 5** for some guidance. But don't think the controllers are doing all the work on their own — oh no! We've got a clever little CQRS pattern in place, where the controllers simply acknowledge the requests and then hand them off to the Handlers for some serious processing.

2. Wondering where the Handlers are hiding? Look no further than the *Features* folder inside the *Application* project. Consult **Figure 6** for visual

guidance. So what exactly do the Handlers do, you might ask? They handle the business logic of an application, and if necessary, call on the trusty DB repository to retrieve or save some data. And if you're worried about any rogue data sneaking through, fear not — the *Fluent Validation* is wired in and is responsible for keeping everything validated.

3. Ready to hunt down the DB repositories? Look no further than the *Repositories* folder inside the *Persistence* project. Check out **Figure 7** for some visual guidance. But what exactly is a repository, you might ask? It's a design pattern that provides a consistent way to work with data sources like databases, web services, and APIs. In other words, it helps to keep the data access logic separate from the business logic of an application.

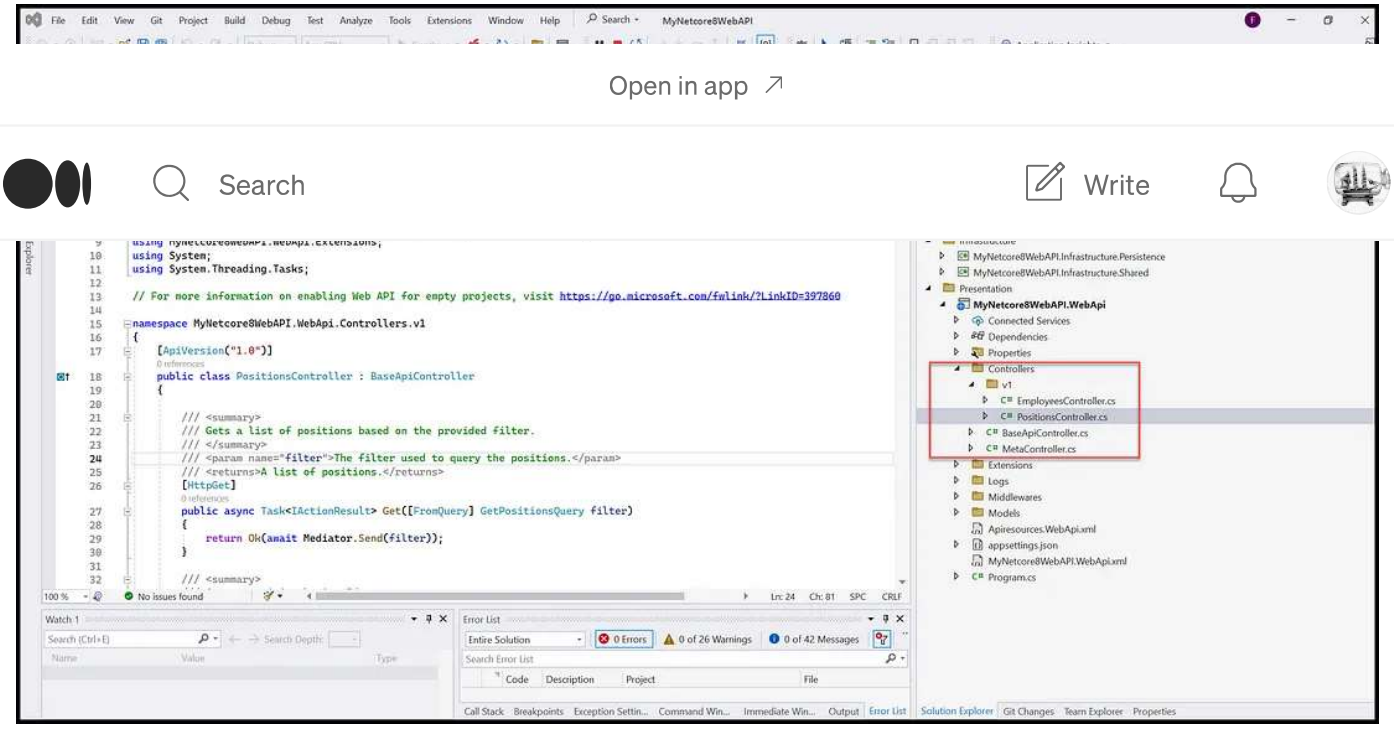


Figure 5 — Location of Controllers in the WebAPI project

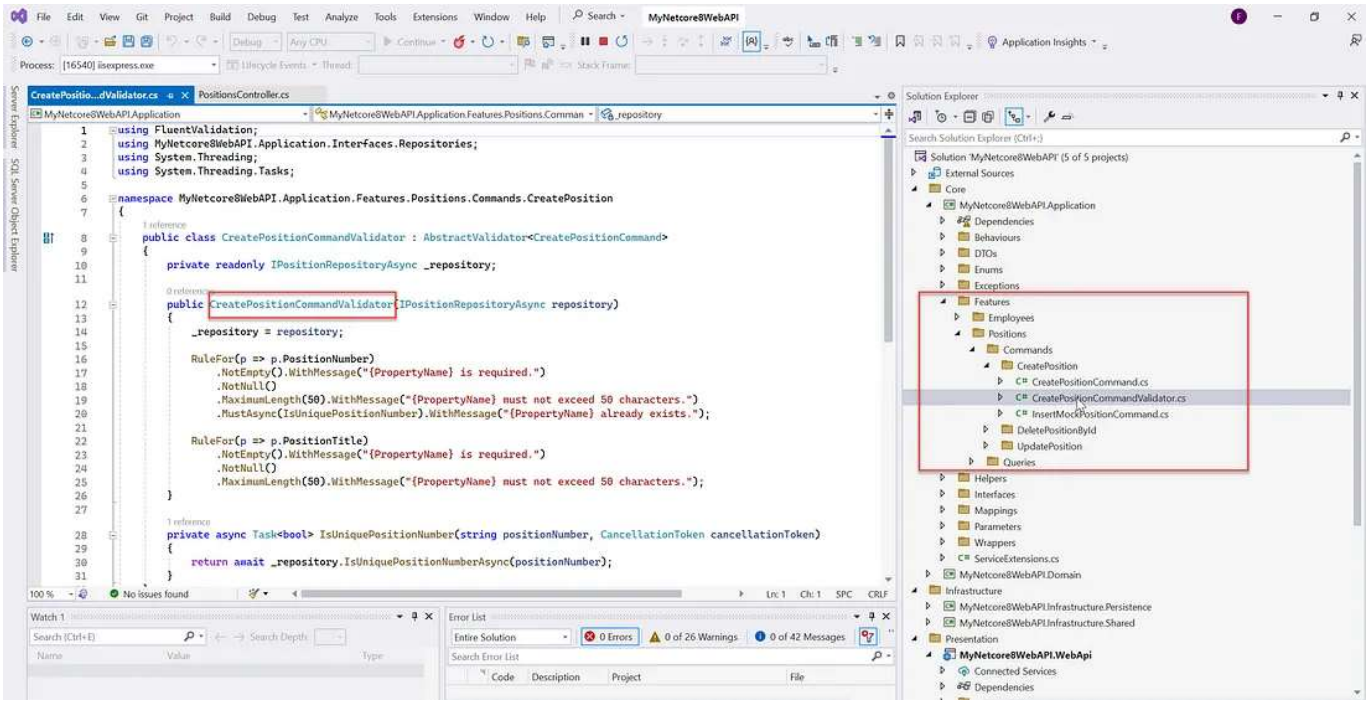


Figure 6 — Location of the Handlers in the Application project

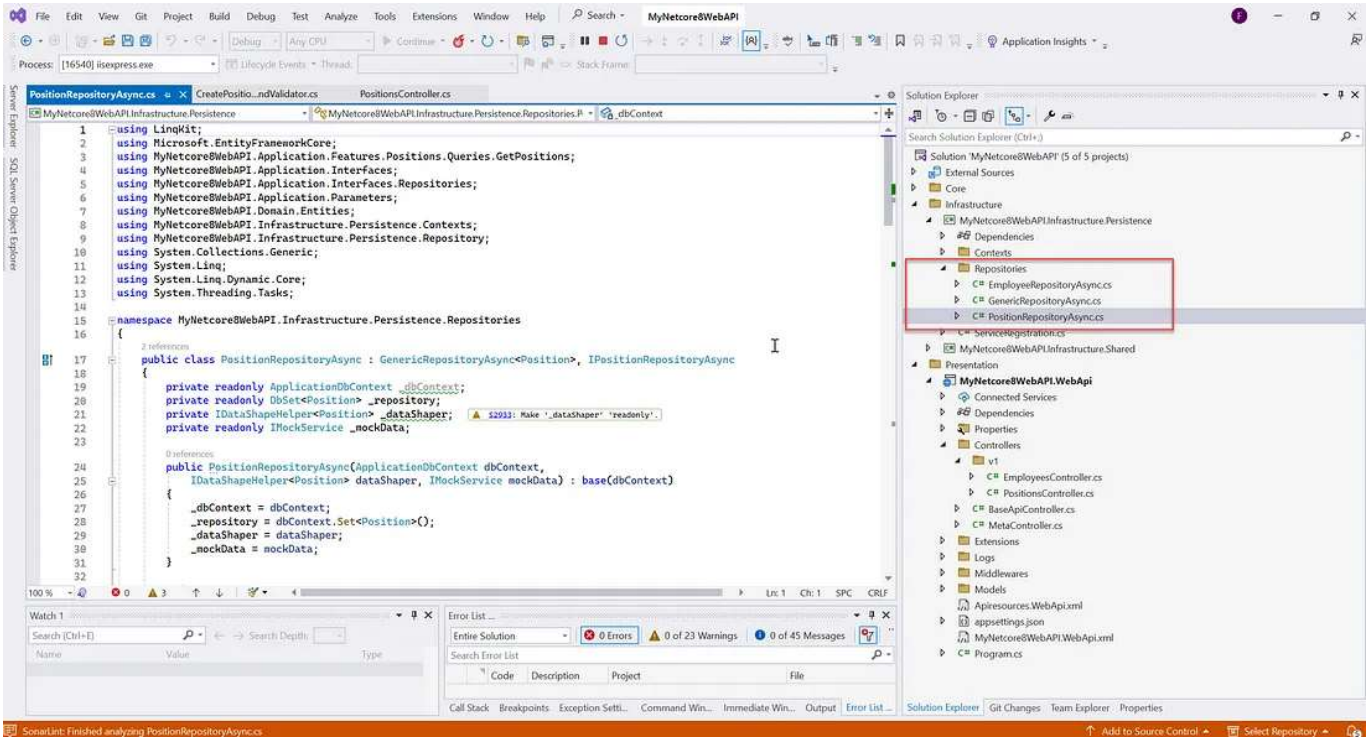


Figure 7 — Location of Repositories in the Persistence project

Part 4: Test Run the Web API project

The OnionAPI template creates two REST resources for your convenience. First up, we've got the Employees resource, which is a mock API where the data is generated dynamically by the *Bogus* library at runtime. Want to learn more about *Bogus*? Just check out the link in the References section later on.

But that's not all — the OnionAPI template also creates a Positions resource, which gets its data from an actual database table. And just to make things extra convenient, that table is automatically seeded with 1,000 rows of fake data the first time the project runs.

Ready to try out the REST API in Swagger? Here's how to get started:

1. Click on the Employees > GET option to expand the screen.
2. Click on the Try it out button.
3. Enter any desired parameters. For example, search for the word *Chief* in the *employeeTitle* field.
4. Specify the order by column name. For example, enter *employeeTitle* in the OrderBy field.
5. Click on the Execute button.
6. Review the JSON output and marvel at the power of the REST API.

And if you need a little extra guidance along the way, just consult **Figure 8** for visual aid.

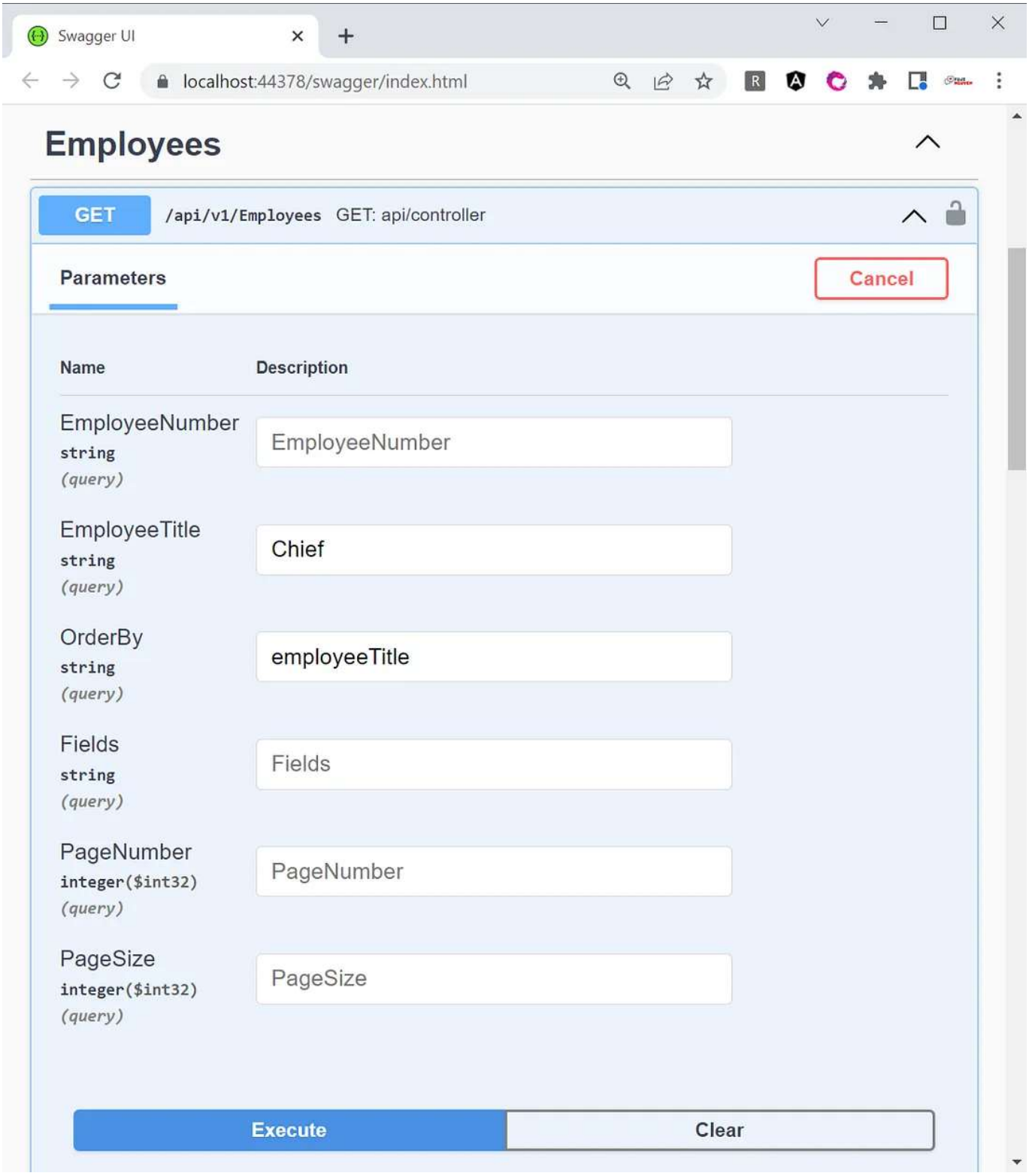


Figure 8 — Example of search criteria

Need a little help understanding the JSON output? Let's take a closer look at **Figure 9**, which is an example screenshot. As you can see, we're using server-side paging for performance reasons. And if you look at the response body,

you'll notice that it includes a "wrapper" *with some friendly meta data that's easy for the frontend app to consum*e. Here's what all those fields mean:

- *pageNumber*: This is the pagination index, such as 1, 2, 3, etc. The default is page 1.
- *pageSize*: This is the number of records per page. The default is 10.
- *recordsFiltered*: This is the number of records that match the search criteria. In this example, the search criteria is "Chief".
- *recordsTotal*: This is the total number of employees in the database.
- *succeeded*: This is a boolean value (true/false) indicating the status of the webapi response.
- *message*: This is a friendly error message, if any.
- *errors*: This is a system error, if any.
- *data*: This is the response data, in JSON format.

So there you have it, a breakdown of all the key fields in the JSON output.

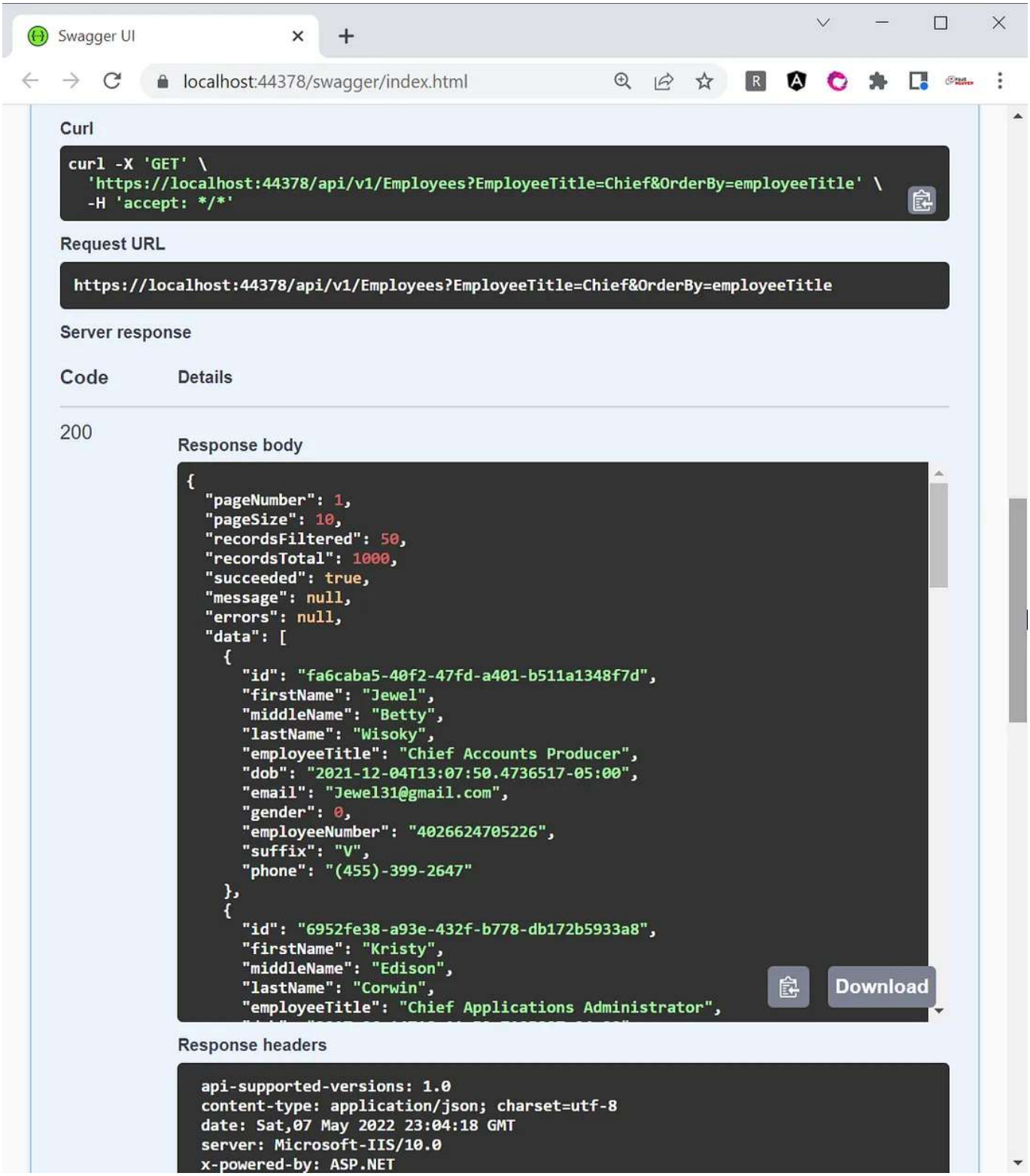


Figure 9 — Example of JSON output including the wrapper showing the pageNumber, pageSize, etc.

As you can see in **Figure 9**, the data response consists of eleven columns. But what if you want to *shape* the response with fewer columns to minimize the amount of data sent over the network? No problem! Just input the column names you want in the Fields area, as shown in **Figure 10**. For example, if the

frontend app only needs the firstName, lastName, and employeeTitle columns, you can specify those and the JSON response will contain only those three columns in the "Data" section, as shown in **Figure 11**.

So there you have it, a simple way to shape the JSON response to fit your apps.

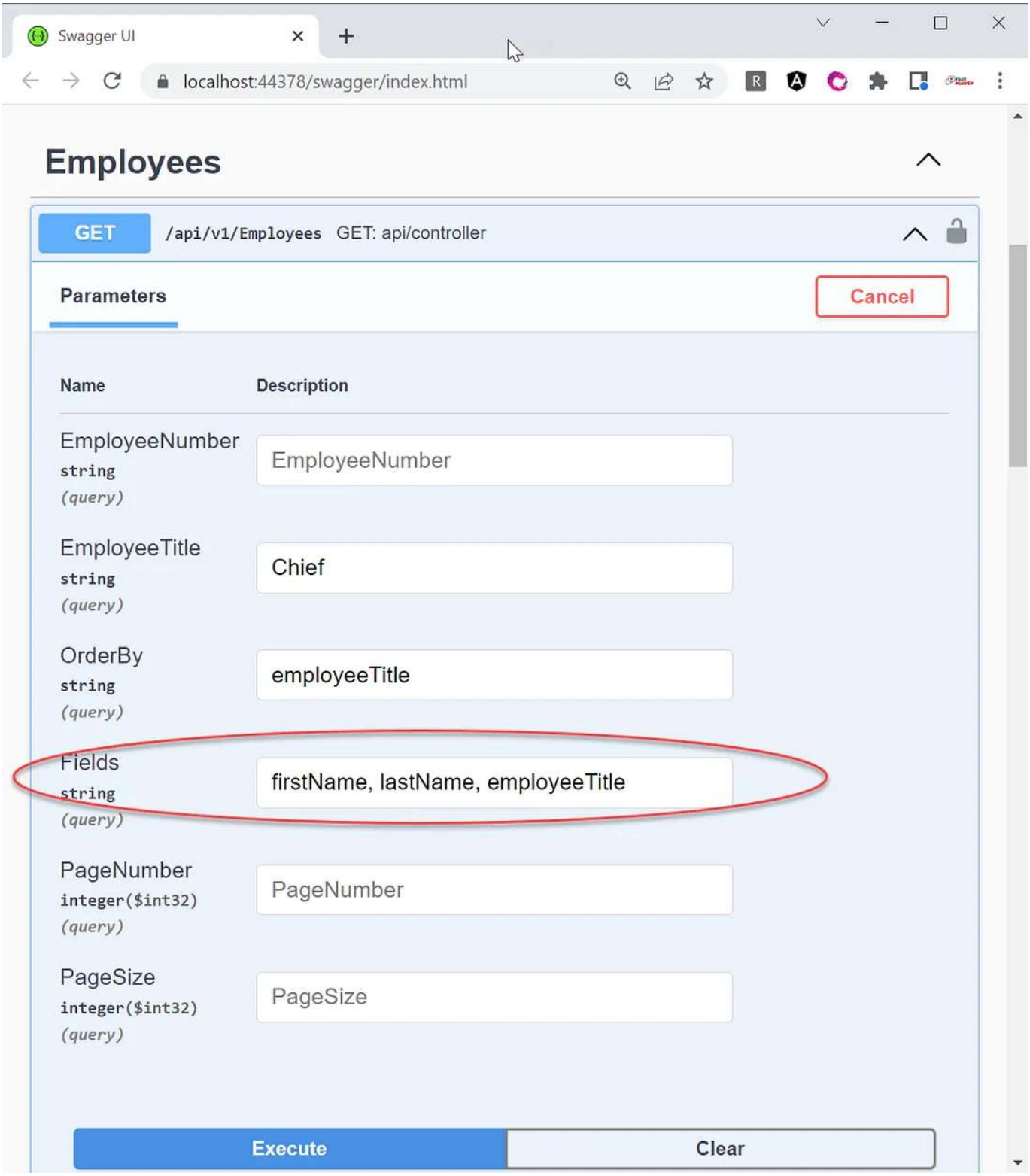


Figure 10 — Specify the columns to return to in the Fields parameter

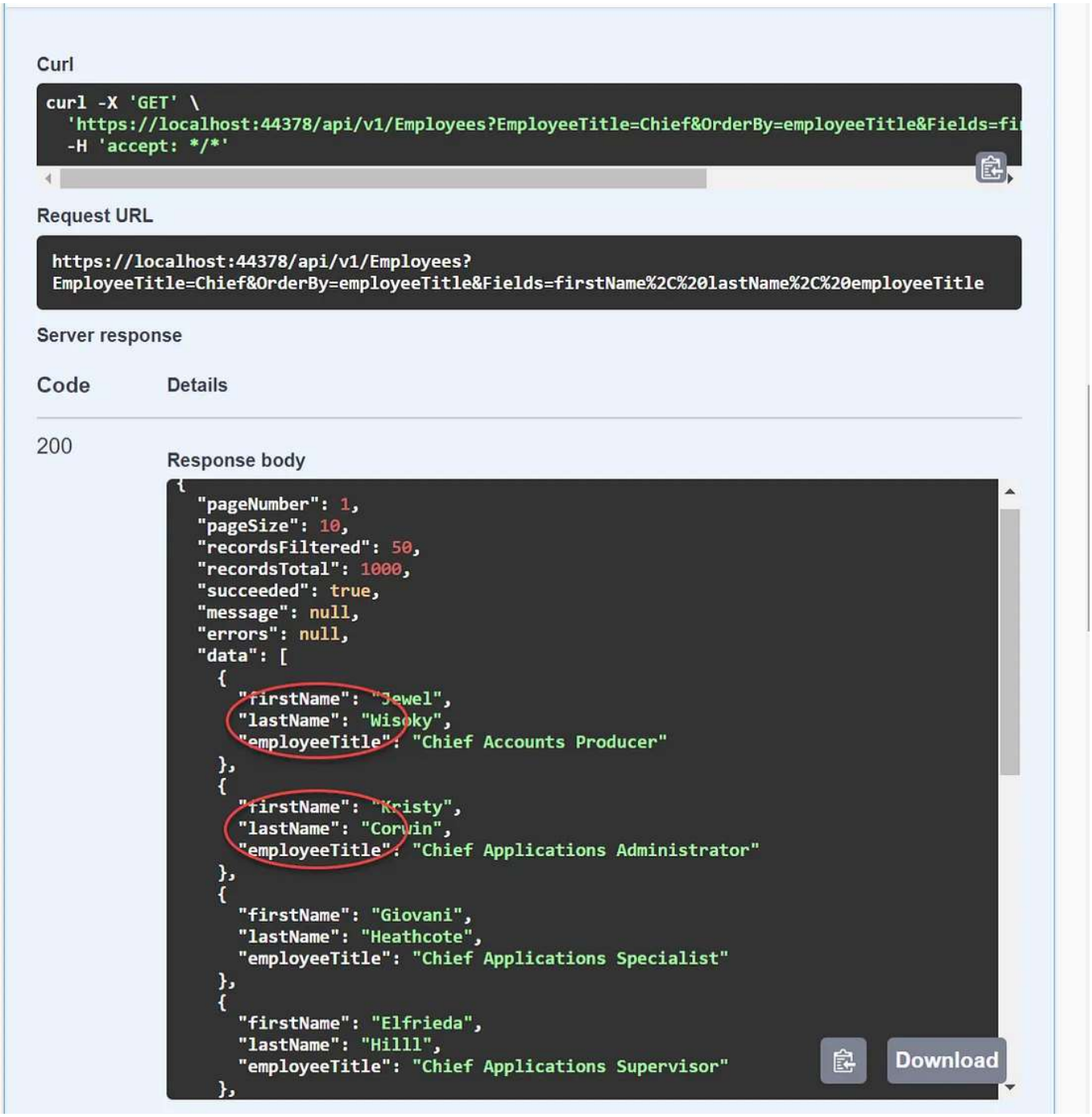


Figure 11 — Result of JSON output with fewer columns when specifying the column names in the Fields parameter

The scaffolded source code is a real treasure trove of useful features, and that includes data modification example code for the Positions resource. Just take a look at **Figure 12** and you'll see all sorts of handy methods like POST, GET, PUT, and DELETE to support CRUD (create, read, update, and delete) operations.

So there you have it, everything you need to get started with data modification using the Positions resource as a point of reference.

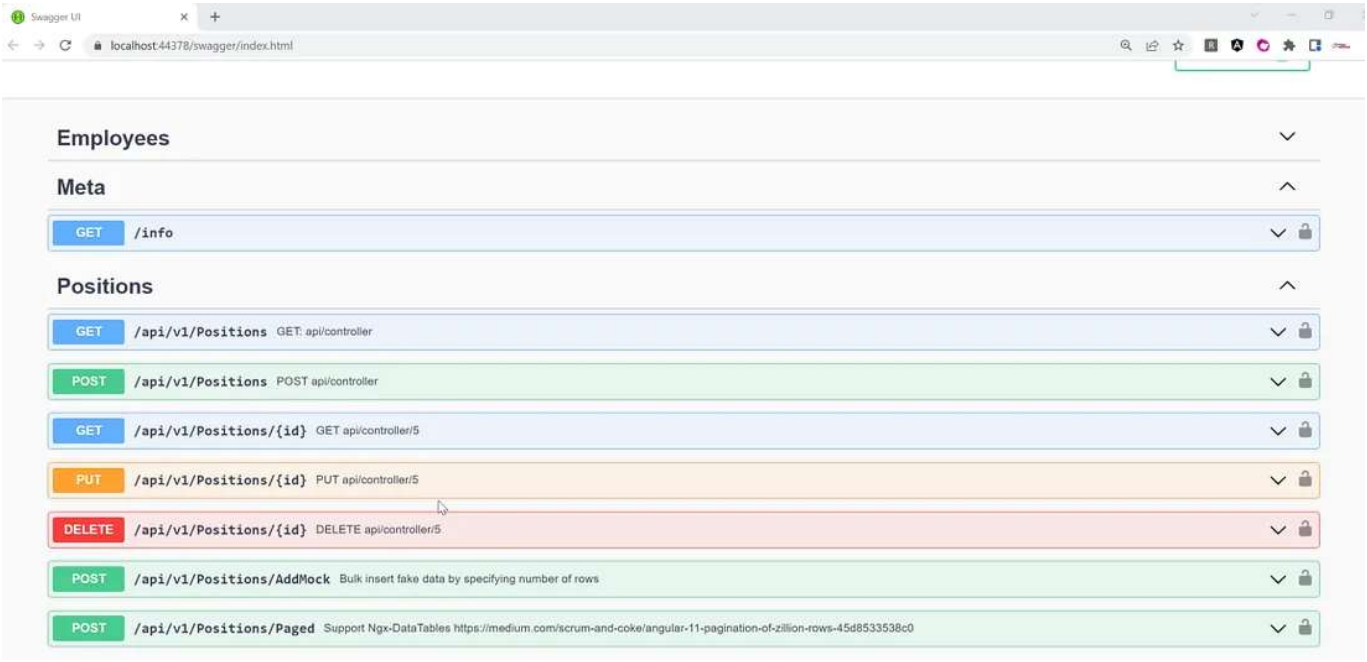


Figure 12 — The REST methods GET, POST, PUT and DELETE to support CRUD

Serilog is integrated into the template to help us keep track of all those pesky WebAPI requests and error events. And the best part? You can see all those structured log events in both the output console and text files. Just consult **Figure 13** for a little visual guidance on where to find them. So go ahead, take a peek at those logs and see what secrets they hold. And remember, with Serilog on our side, we'll always know exactly what's going on in our codebase.

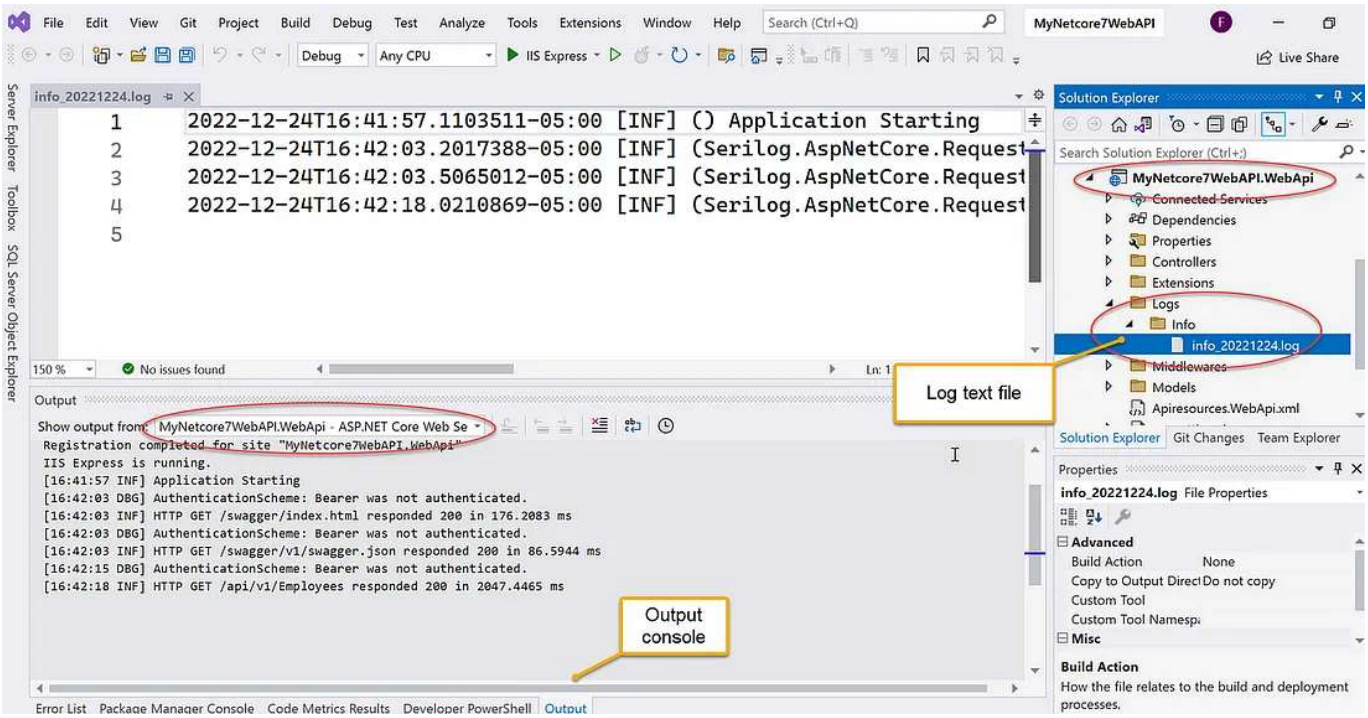


Figure 13 — Serilog console and text file

OnionAPI Template in Action

I created a tutorial series focused on creating dynamic and user-friendly web applications using Single Page Application (SPA) architecture and modern tools such as Angular 15, Bootstrap 5, and .NET REST API. The WebAPI layer in this tutorial series was generated with the powerful OnionAPI template.

Recommended Contents

Below are blogs related to the Template OnionAPI, where you will discover practical examples, key concepts, and essential features within the scaffolded code:

1. Building .NET 8 Web API for a Warehouse Application Using Visual Studio Template OnionAPI: This blog provides a detailed walkthrough on creating a .NET 8 Web API specifically for warehouse applications, utilizing the Visual Studio Template OnionAPI. It covers the setup, configuration, and customization aspects of the template. Data seeding is implemented with the Bogus library.
2. Building a Generic Repository Interface in .NET Using Generics: A Practical Guide: In this guide, you'll learn how to efficiently construct a generic repository interface using .NET's generics feature. It's a practical resource for understanding and implementing reusable and scalable data access layers in .NET applications.
3. Optimizing Bulk Inserts in EF Core: A Comprehensive Guide: This comprehensive guide delves into optimizing bulk insert operations in Entity Framework Core. It's an essential read for developers looking to enhance performance and efficiency in data handling and manipulation. Mock data is implemented with the Bogus library.

Summary

If you're involved in managing development, you know how challenging it can be to keep everything running smoothly. Governance can be a real pain, and it can be hard to find tools that help improve processes and add value. That's where the OnionAPI template comes in. It's designed to increase consistency and decrease delivery time, making your life just a little bit easier.

But wait, there's more! The OnionAPI also provides boilerplate code for all sorts of advanced REST API features, like data filtering, sorting, shaping,

paging, and CRUD. And the best part is, these features are written as base classes, so you can easily inherit and extend them to meet the specific needs of your project. So don't struggle with messy code and complex processes any longer — give the OnionAPI template a try and see how it can transform your development experience.

> Thanks for reading! Hope you found it useful. Want more? Hit the "Follow" button below my profile. With your support, I'll keep creating awesome content for you. Have a great day ahead!
> — Fuji Nguyen

Csharp Rest Api Visual Studio Template

Written by Fuji Nguyen

Follow

802 Followers · Editor for Scrum and Coke

DX Advocate, OpenSource Contributor, Pickleball Player - Improves software dev experience, contributes to opensource projects, and plays pickleball for leisure.

More from Fuji Nguyen and Scrum and Coke

Fuji Nguyen in Knowledge Pills

How to call REST API in Angular?

To call a REST API in Angular, you can use the HttpClient service, which is part of the...

2 min read · Dec 16, 2022

11

Fuji Nguyen in Scrum and Coke

Crystal Report to PDF with Angular and WebAPI

Crystal Report in modern Angular and REST API

8 min read · Oct 16, 2021

35 4

Fuji Nguyen in Scrum and Coke

Building .NET 8 Web API for a Warehouse Application Using...

Imagine you've been tasked with developing a Web API to support an order fulfillment...

14 min read · Dec 12, 2023

Fuji Nguyen in Knowledge Pills

What is startup.cs file in ASP.NET Core?

In an ASP.NET Core application, the Startup class is responsible for configuring the...

2 min read · Dec 20, 2022

2 9

See all from Fuji Nguyen See all from Scrum and Coke

Recommended from Medium

Oleksandr Redka in C# Programming

EntityFramework-friendly computed properties

Putting together all of the nicest things from the latest dotnet

11 min read · Dec 29, 2023

10

Levent Ozturk

NET Aspire

I try to explain “What is NET Aspire”!

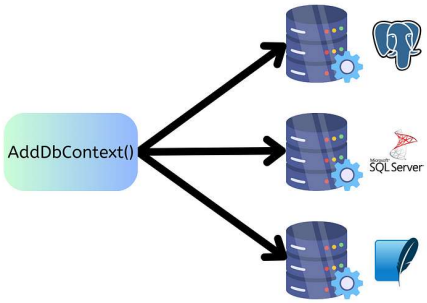
5 min read · Dec 26, 2023

85

Lists

Productivity

234 stories · 263 saves



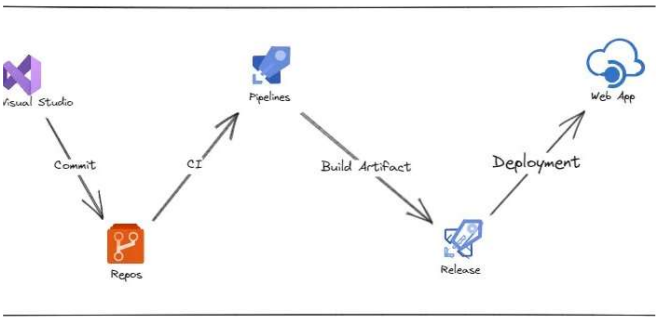
Aniekan Umanah

Multiple DB Providers

One Context Multiple Databases (ASPNET CORE)

5 min read · Sep 25, 2023

5



Muhammad Rafay

Azure DevOps CI/CD for ASP .NET Core on Azure App Service—Part...

In my previous post, we successfully deployed a basic “Hello World” application and a Basic...

7 min read · Sep 16, 2023

7

Thomas Pentenrieder in medialesson

if-elseif-else in Azure Bicep

Azure Bicep supports conditional deployments making it easy to create...

1 min read · Jul 31, 2023

Mahesh Patidar in Globant

SpecFlow- a Free and Open Source BDD Framework for .NET

—

7 min read · Nov 28, 2023


See more recommendations