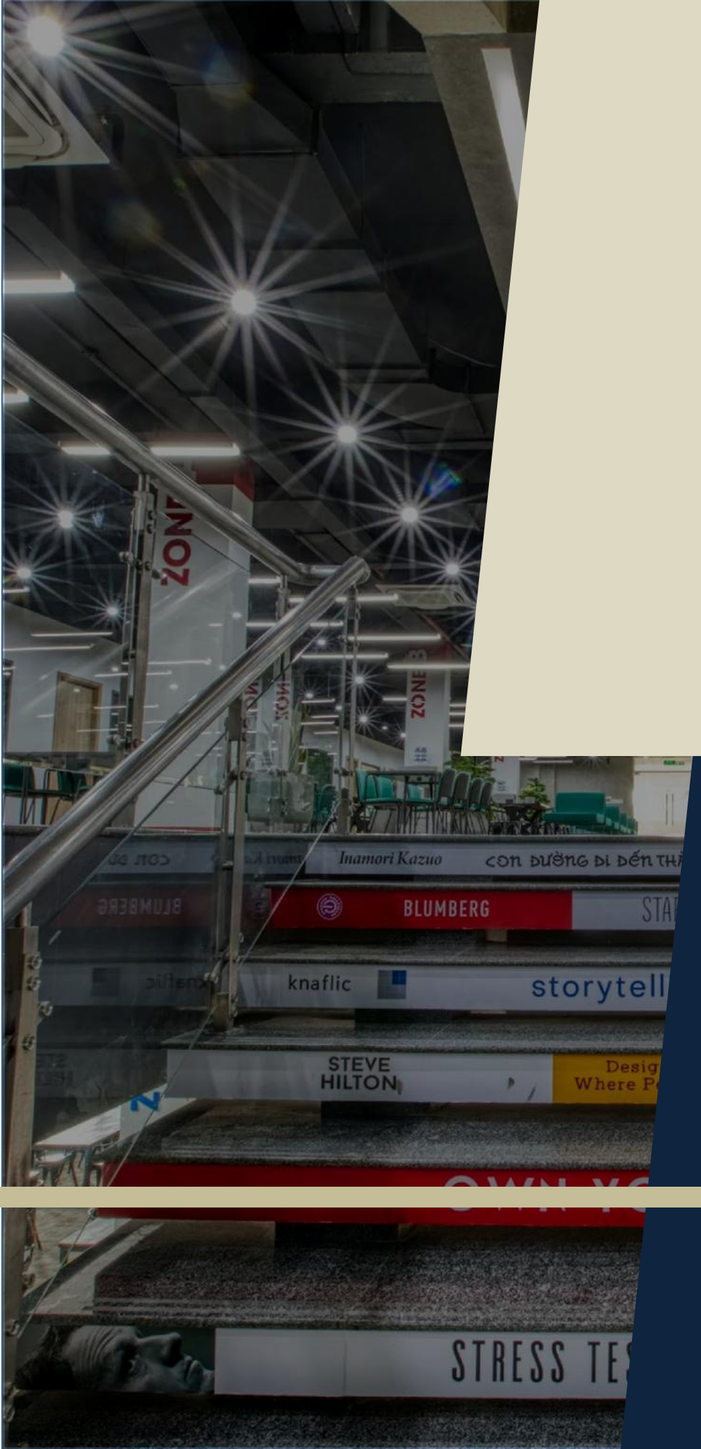


EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Backend Development

Chapter 2: ASP.NET Core

TABLE CONTENT

- Client-server architecture and Backend Frameworks
- Introduction to .Net Framework, .Net, ASP.Net, and ASP.Net Core
- ASP.Net Get Started.

Client-Server Architecture and Backend Frameworks

Client-Server Architecture:

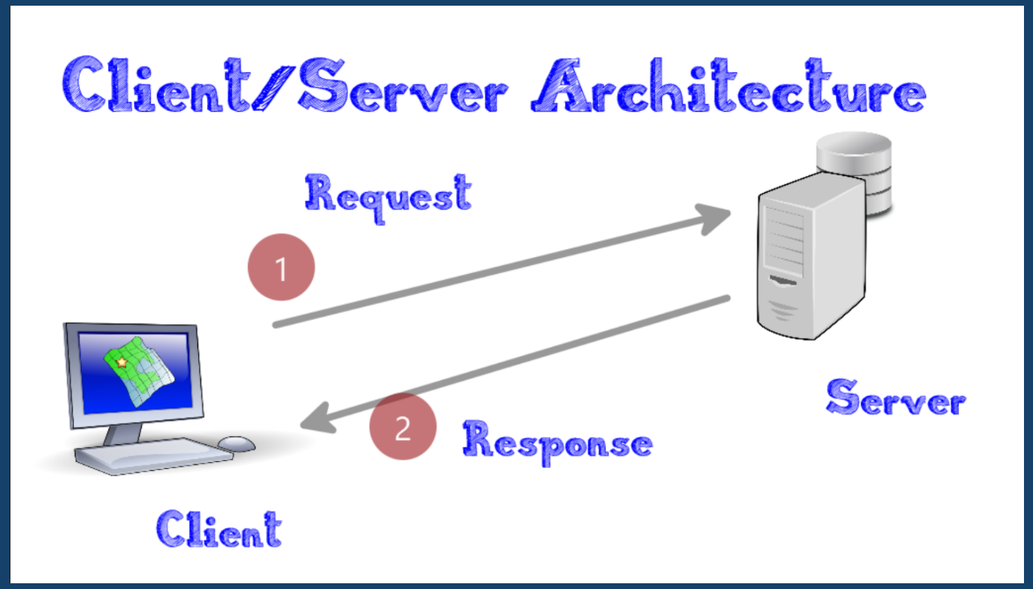
- A model that divides a system into:
  - **Client**: sends requests, displays UI (browser, mobile app, etc.)
  - **Server**: processes requests, accesses data, returns responses
- Communicates via **HTTP/HTTPS**, commonly using **REST API** and **JSON**

Basic Flow:

- Client sends a Request
- Server processes and queries data
- Server sends back a Response

Advantages:

- Separation of frontend/backend
- Easier maintenance and scalability
- Supports multiple platforms (web, mobile...)



Backend Frameworks

What are Backend Frameworks?

- Tools that help build backend/server-side applications
- Provide built-in support for HTTP handling, routing, database access, authentication, and security

Popular Backend Frameworks:

Framework	Language	Highlights
Express.js	JavaScript	Lightweight, fast, widely used
Django	Python	Fast development, secure, full-stack
Spring Boot	Java	Powerful, used in enterprise apps
Laravel	PHP	Clean syntax, strong DB features
ASP.NET Core	C# (.NET)	High performance, multi-platform

.NET Framework

Released: 2002.

Support Platform : Only run in **Windows OS**.

Mục đích: Develop desktop application (Windows Forms, WPF), web applications (ASP.NET), web services (WCF), and develop the application for the Enterprise.

Source Code: Closed source.

.NET Core

Released: 2016.

Support platforms: Multi-platform (Windows, macOS, Linux).

Purpose: Build modern web applications, microservices, cloud applications, and highly scalable services.

Source code: Open source, developed and maintained by a global community.

.NET Framework - .NET Core

	.NET Framework	.NET Core
Operating System	Windows only	Cross-platform: Windows, macOS, Linux
Source code	Close Source	Open-source, community contributions
Performance	Lower than .NET Core	High performance due to optimization and lightweight design
Architecture	Monolithic: Integrates many features into a large application	Modular: Uses NuGet packages, includes only necessary components
Dependency Injection	Supported through external libraries like Unity or Ninject	Built-in, powerful DI system
Middleware Pipeline	Limited customization of the request processing pipeline	Flexible middleware pipeline, easy to customize
Container Support	Little support, not optimized for containerization	Optimized for containers, easy to deploy on Docker and Kubernetes
Configuration	Uses complex Web.config file	Simpler configuration through appsettings.json file and flexible code configuration
Security	Integrated with Windows security features like Active Directory	Provides modern security features like OAuth, JWT, easy to integrate with external security services
Framework Support	Limited, heavily dependent on .NET Framework	Good support for many frameworks and new technologies like Blazor, gRPC
Updates and Support	Slow updates, mainly focused on maintaining legacy applications	Frequent updates, receives new features quickly
Cloud Deployment	Not optimized for cloud	Optimized for cloud deployment

WHAT IS ASP.NET ?

ASP.NET is a robust web application development framework created by Microsoft. It empowers developers to build dynamic web applications, web services, and APIs using programming languages such as C# or VB.NET. ASP.NET supports various development models, enabling the creation of efficient, secure, and maintainable web applications.

ASP.NET

Release: Launched in 2002 as part of the .NET Framework.

Supported platforms: Runs only on the Windows operating system.

Architecture: Built on the .NET Framework with models such as Web Forms, MVC (from ASP.NET MVC 1.0 onwards), and Web API.

ASP.NET Core

Release: Launched in 2016 as part of .NET Core, later becoming part of the .NET platform from .NET 5 onwards.

Supported platforms: Cross-platform (Windows, macOS, Linux).

Architecture: Modular, lightweight, and performance-optimized design, supporting models such as MVC, Razor Pages, Blazor, and Web API.

ASP.NET - ASP.NET Core

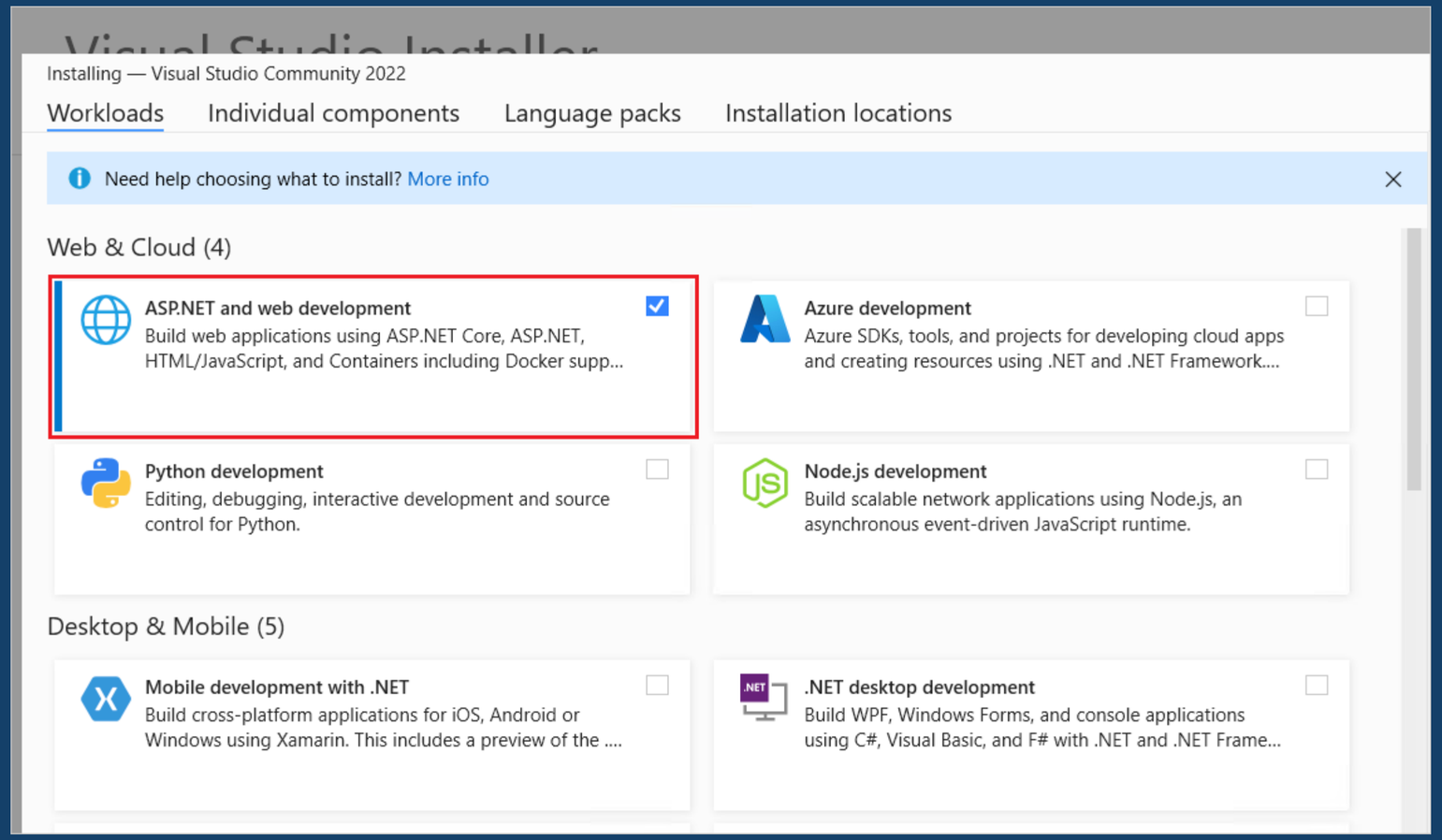
	ASP.NET	ASP.NET Core
Supported Platforms	Windows only	Cross-platform: Windows, macOS, Linux
Source Code	Primarily closed-source	Open-source, community-driven
Performance	Lower performance compared to ASP.NET Core	High performance due to optimization and lightweight design
Architecture	Monolithic: Integrates many features into a large application	Modular: Uses NuGet packages, includes only necessary components
Middleware	Limited customization of the request processing pipeline	Flexible middleware pipeline, easy to customize
Dependency Injection	Supported through external libraries	Built-in, powerful DI system
Container Support	Limited support, not optimized for containerization	Optimized for containers, easy to deploy on Docker and Kubernetes
Configuration and Deployment	Uses complex Web.config file	Simpler configuration through appsettings.json file and flexible code-based configuration
Security	Integrated with Windows security features	Provides modern security features, supports OAuth, JWT, etc.
Framework Support	Limited, heavily dependent on .NET Framework	Supports a wide range of frameworks and emerging technologies like Blazor, gRPC
Updates and Support	Slow updates, primarily focused on maintenance	Frequent updates, receives new features quickly

Get started with ASP.NET Core

App type	Scenario	Tutorial
Web app	New server-side web UI development	Get started with Razor Pages
Web app	Maintaining an MVC app	Get started with MVC
Web app	Client-side web UI development	Get started with Blazor
Web API	RESTful HTTP services	Create a web API†
Remote Procedure Call app	Contract-first services using Protocol Buffers	Get started with a gRPC service
Real-time app	Bidirectional communication between servers and connected clients	Get started with SignalR

ASP.NET Core APIs

Prerequisites: .NET Core 8.0



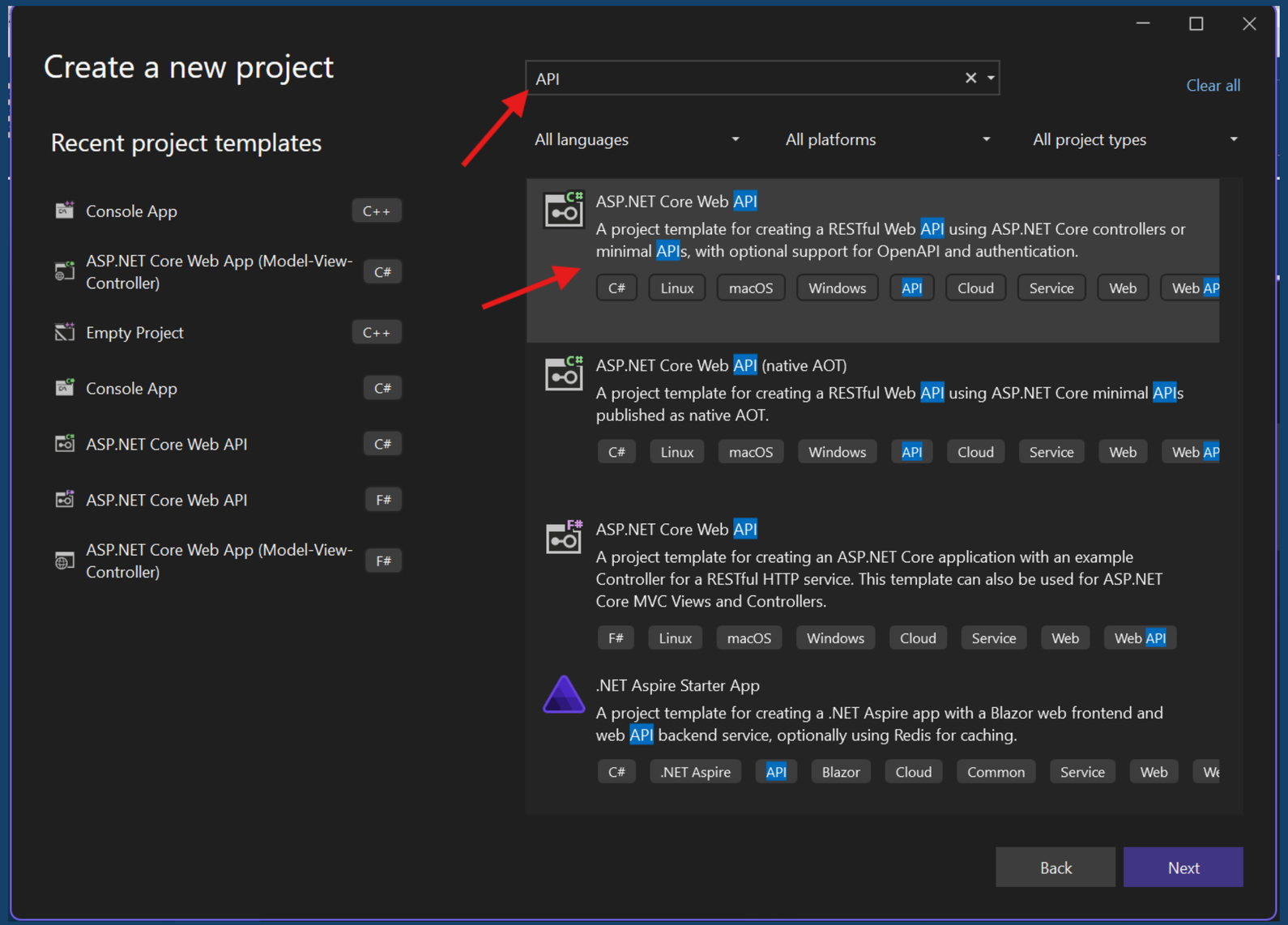
ASP.NET Core APIs

Create a web app:

- Start Visual Studio and select **Create a new project**.
- In the **Create a new project** dialog, select **ASP.NET Core Web API > Next.**
- In the **Configure your new project** dialog:
  - Enter “LibraryManagement” for Project name. It's important to name the project “LibraryManagement”. Capitalization needs to match each namespace when code is copied.
  - The Location for the project can be set to anywhere.
- Select Next.
- In the Additional information dialog:
  - Select .NET 8.0 (Long Term Support).
  - Verify that Do not use top-level statements is unchecked.
- Select Create.

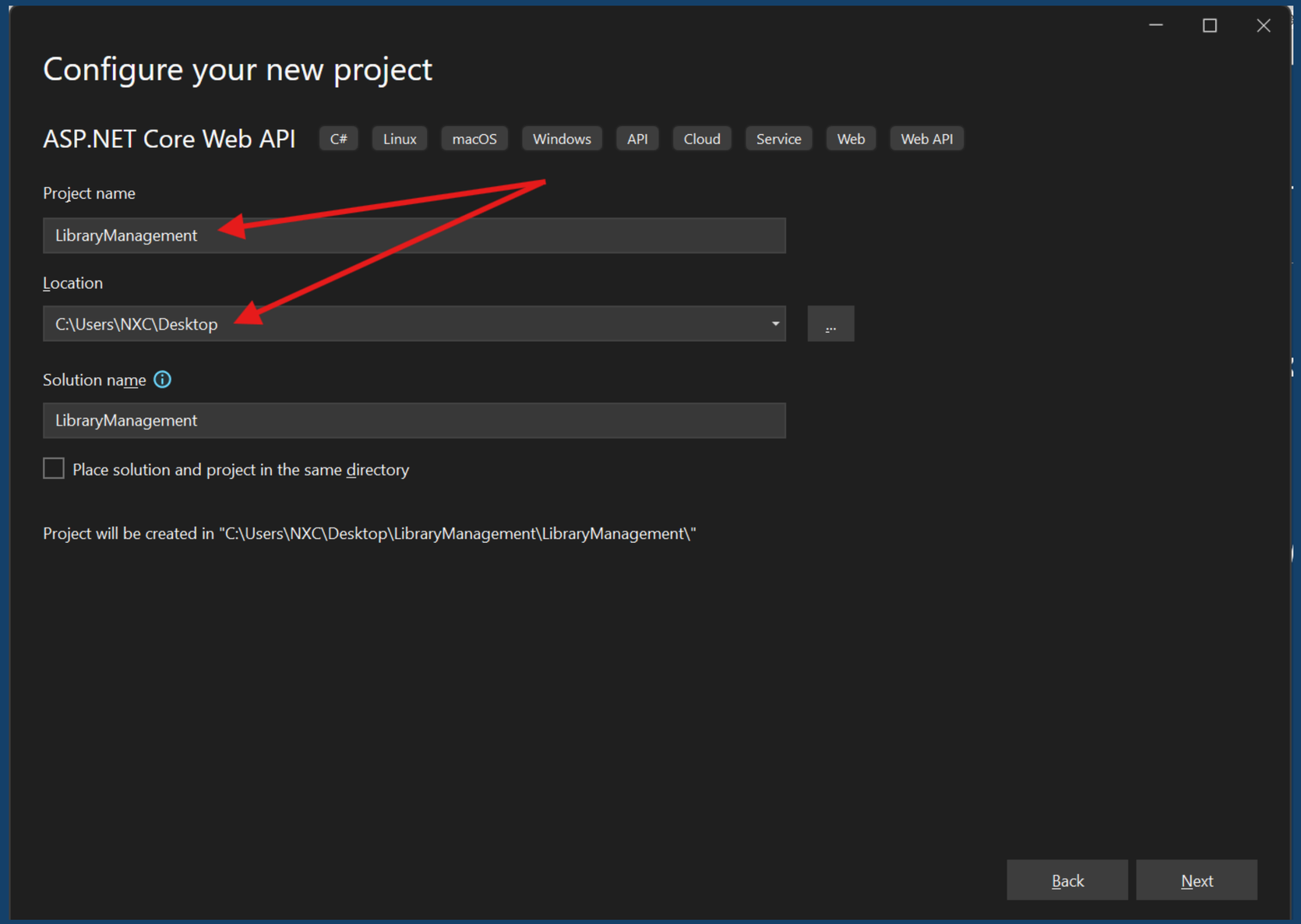
ASP.NET Core API

Create a web app:



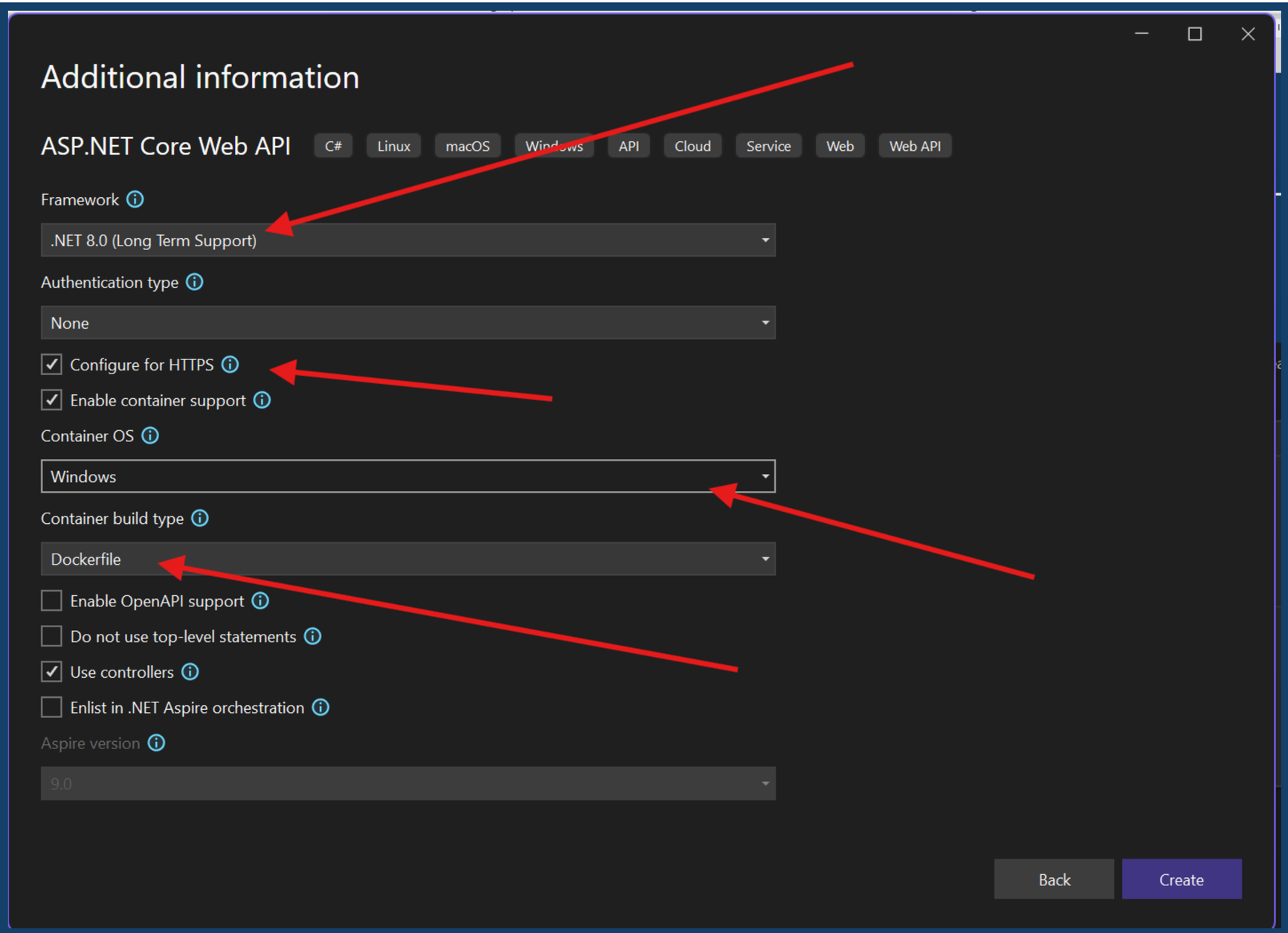
ASP.NET Core API

Create a web app:



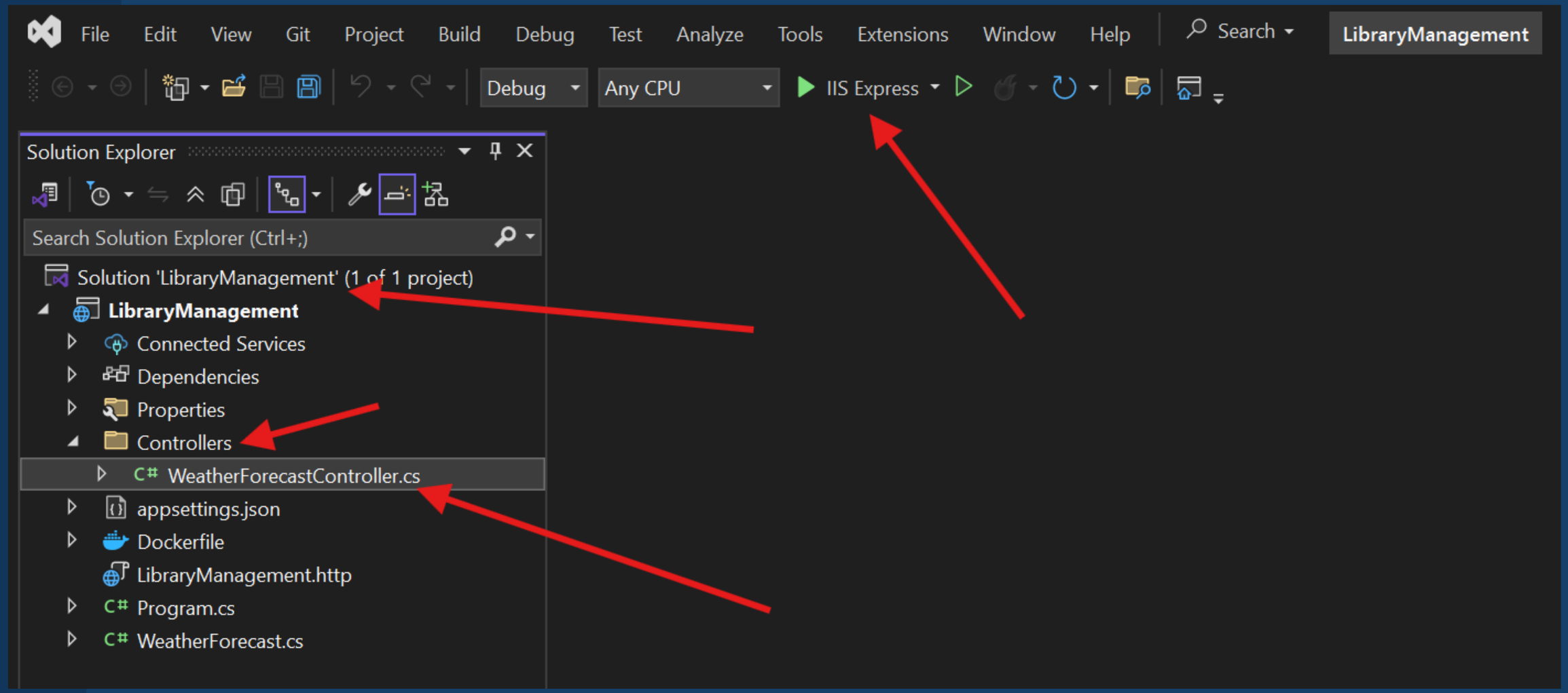
ASP.NET Core API

Create a web app:



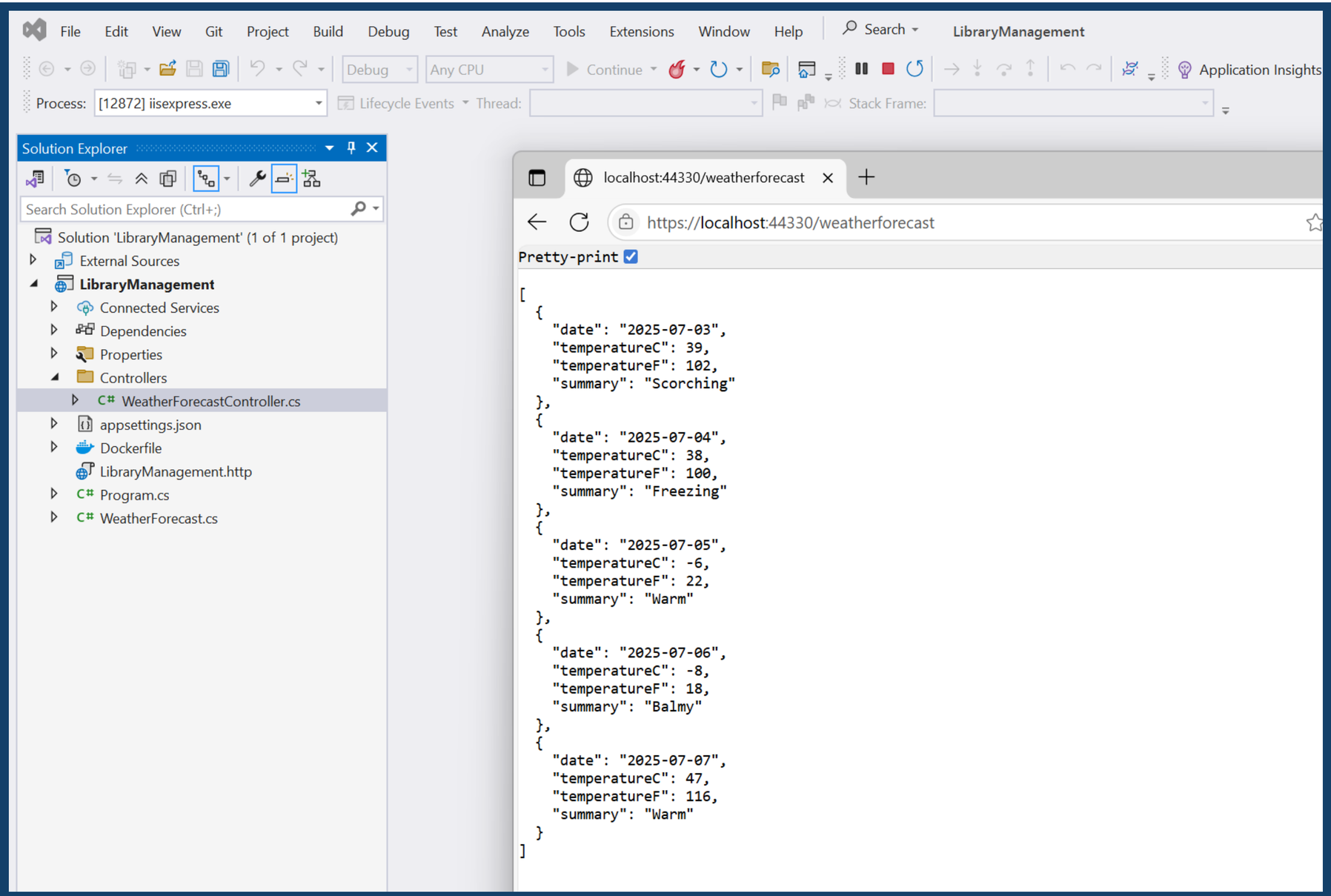
ASP.NET Core API

Run the app



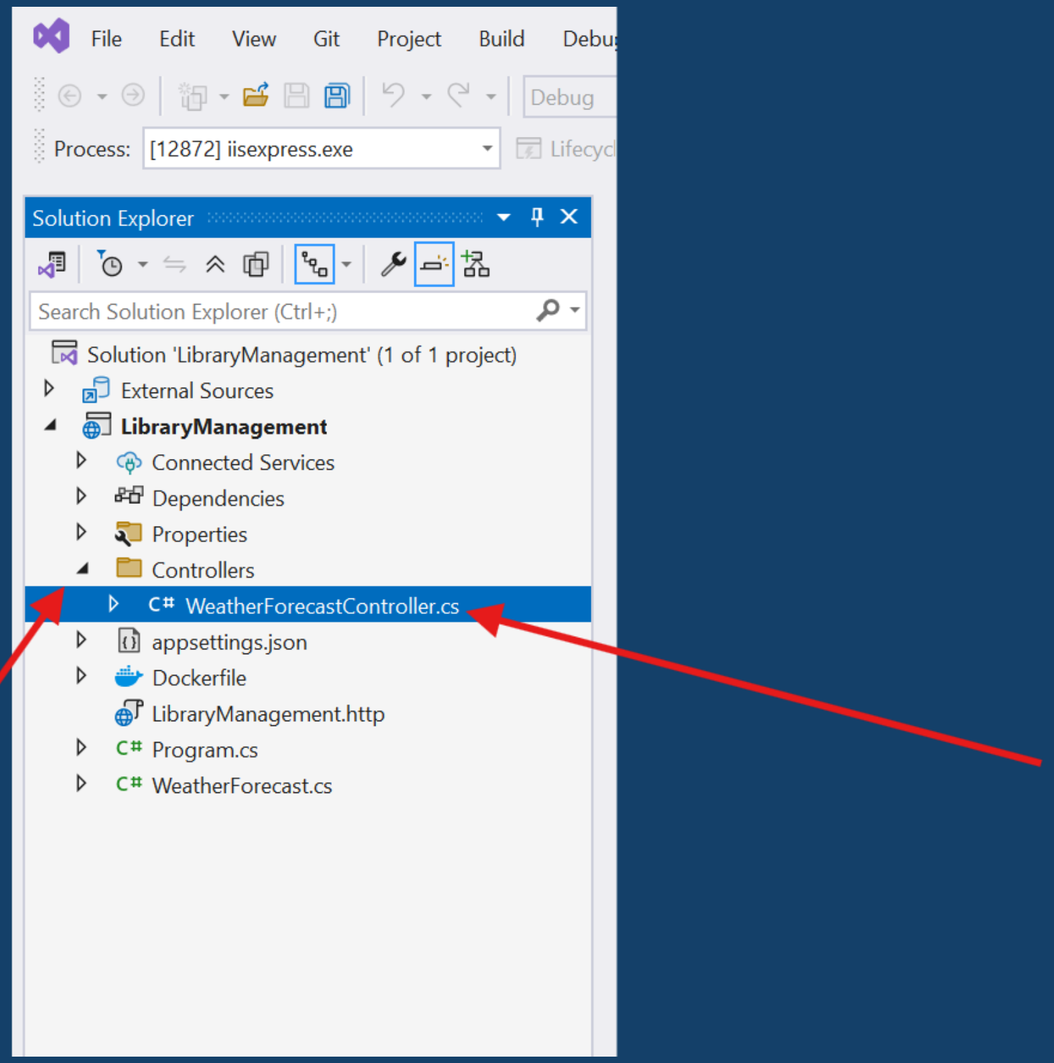
ASP.NET Core API

Run the app



ASP.NET Core API

Folder struct



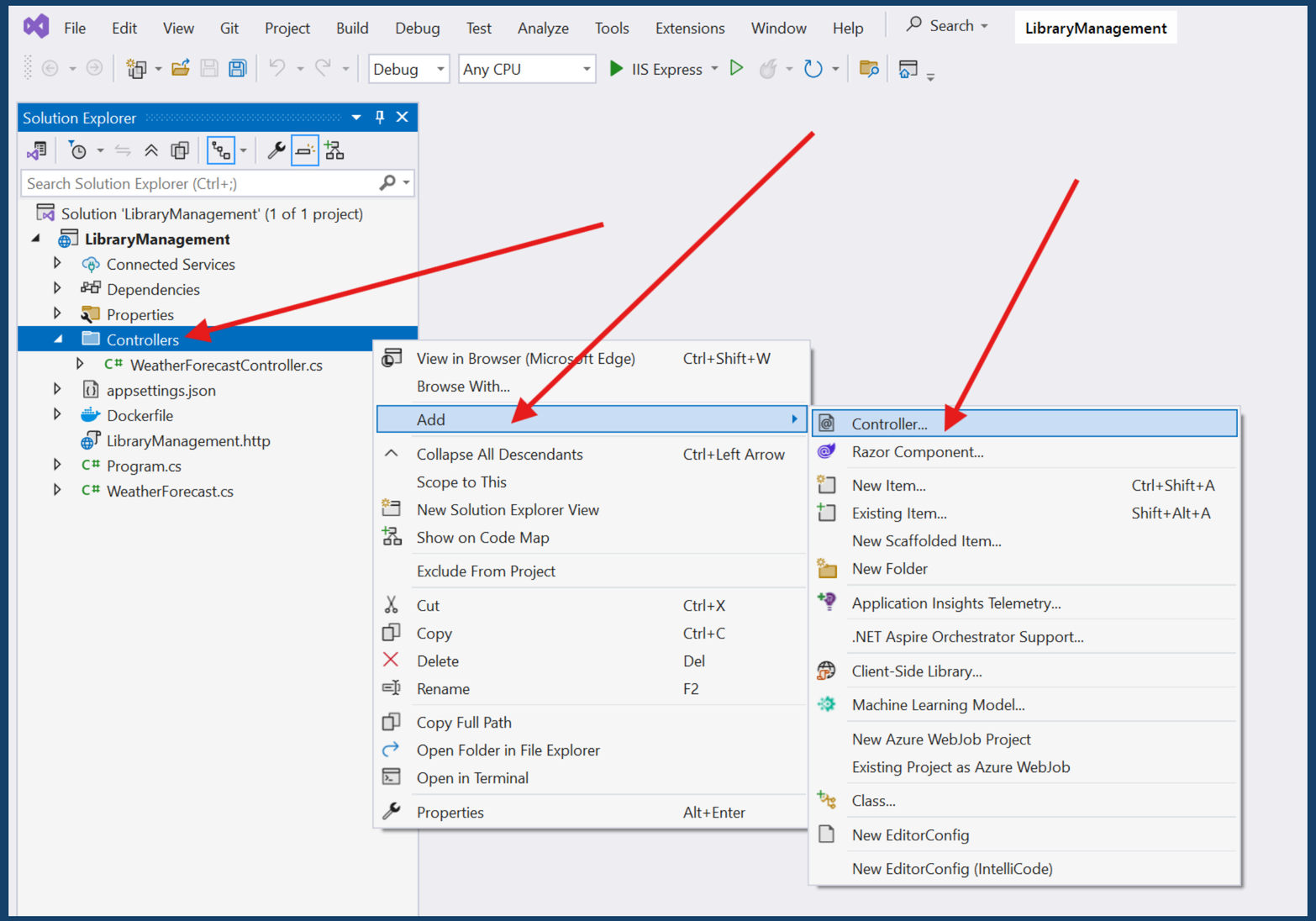
ASP.NET Core API

Controller

- Handles user input and interactions.
- Processes requests, retrieves model data, and selects the appropriate view for response.
- Example: Handles URL requests like => localhost:44330/weatherforecast.

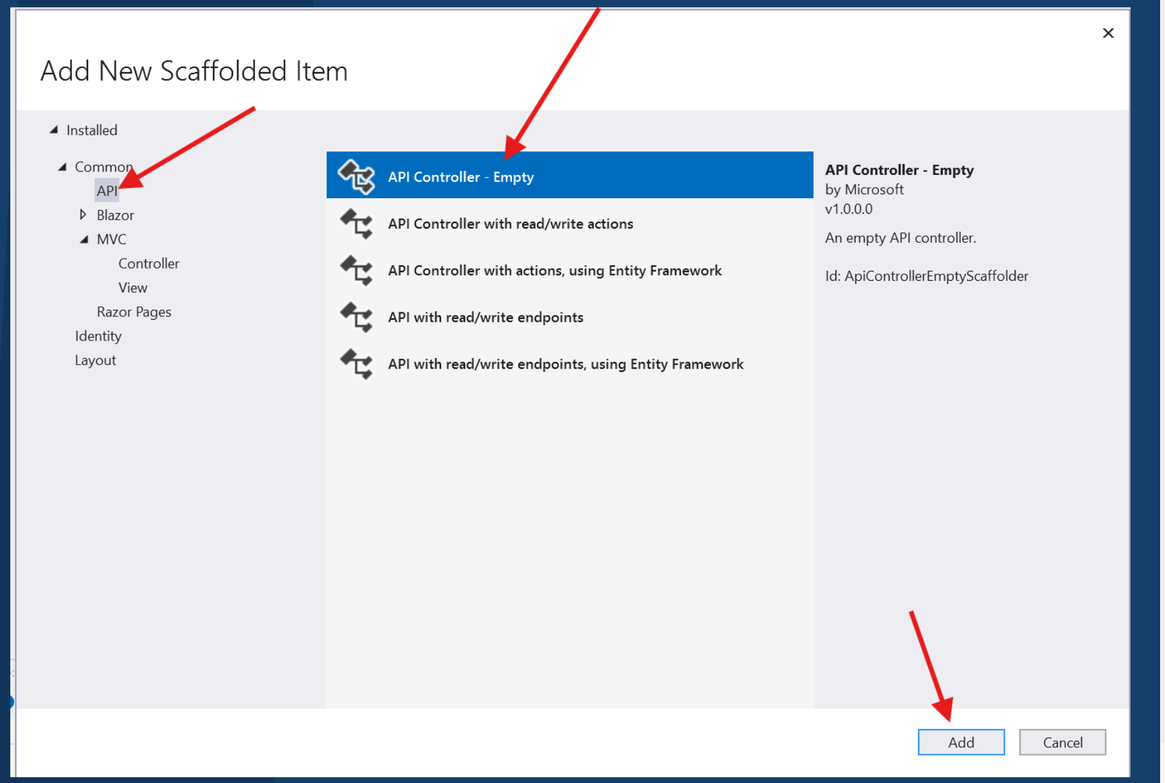
ASP.NET Core API

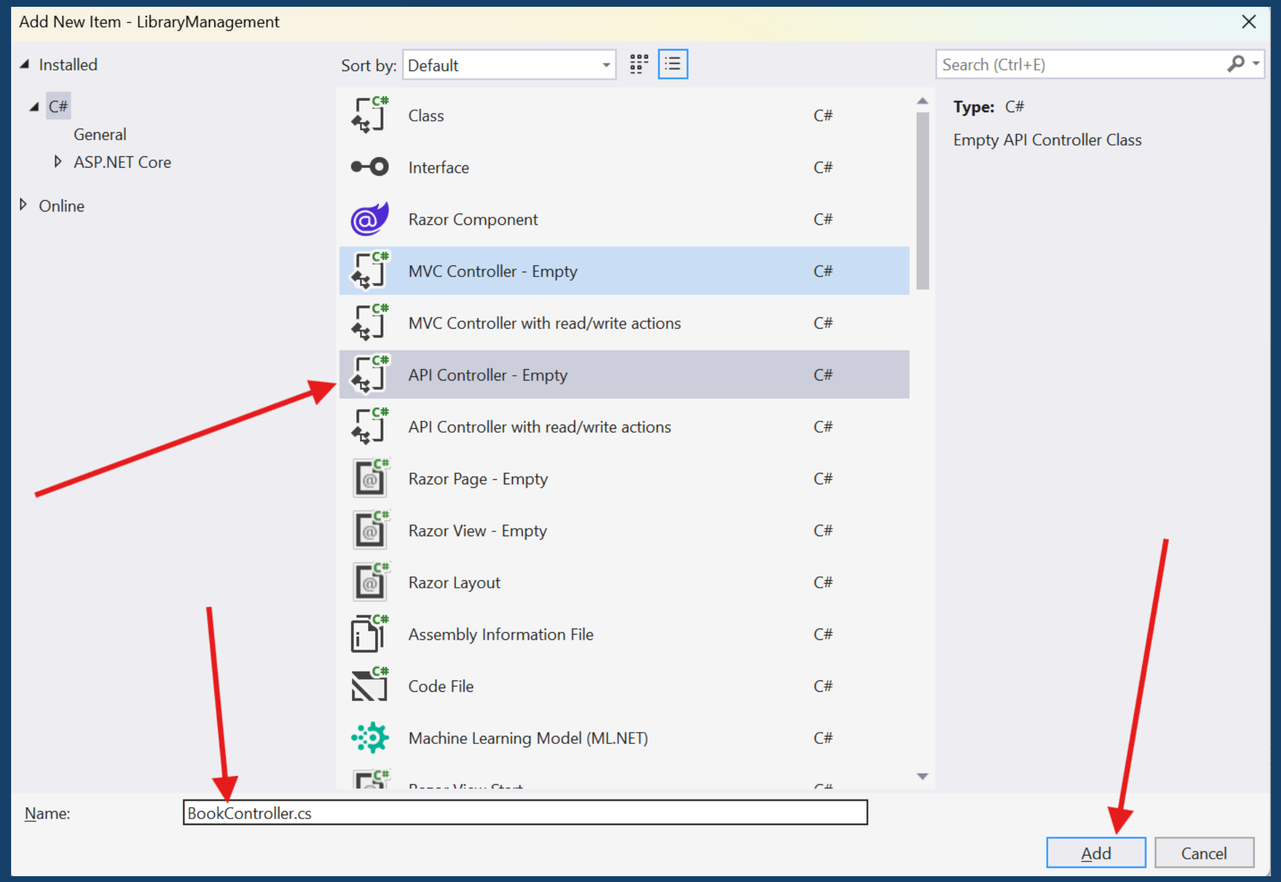
Controller - Add a controller



ASP.NET Core MVC

Controller – Add a controller





ASP.NET Core MVC

Controller – Index method

BookController.cs

LibraryManagement | LibraryManagement.Controllers.BookController | GetBooks()

```
1   using Microsoft.AspNetCore.Http;
2   using Microsoft.AspNetCore.Mvc;
3
4   namespace LibraryManagement.Controllers
5   {
6       [Route("api/[controller]")]
7       [ApiController]
        0 references
8       public class BookController : ControllerBase
9       {
            5 references
10          public class BookDto
11          {
                3 references
12              public string Title { get; set; }
                3 references
13              public string Author { get; set; }
14          }
15
16          [HttpGet]
            0 references
17          public ActionResult<IEnumerable<BookDto>> GetBooks()
18          {
19              var books = new List<BookDto>
20              {
21                  new BookDto { Title = "The Great Gatsby", Author = "F. Scott Fitzgerald" },
22                  new BookDto { Title = "To Kill a Mockingbird", Author = "Harper Lee" },
23                  new BookDto { Title = "1984", Author = "George Orwell" }
24              };
25              return Ok(books);
26          }
27      }
28  }
29
```

localhost:44330/api/Book

https://localhost:44330/api/Book

Pretty-print


```
[
  {
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald"
  },
  {
    "title": "To Kill a Mockingbird",
    "author": "Harper Lee"
  },
  {
    "title": "1984",
    "author": "George Orwell"
  }
]
```

ASP.NET Core API

Controller – HTTP Endpoint

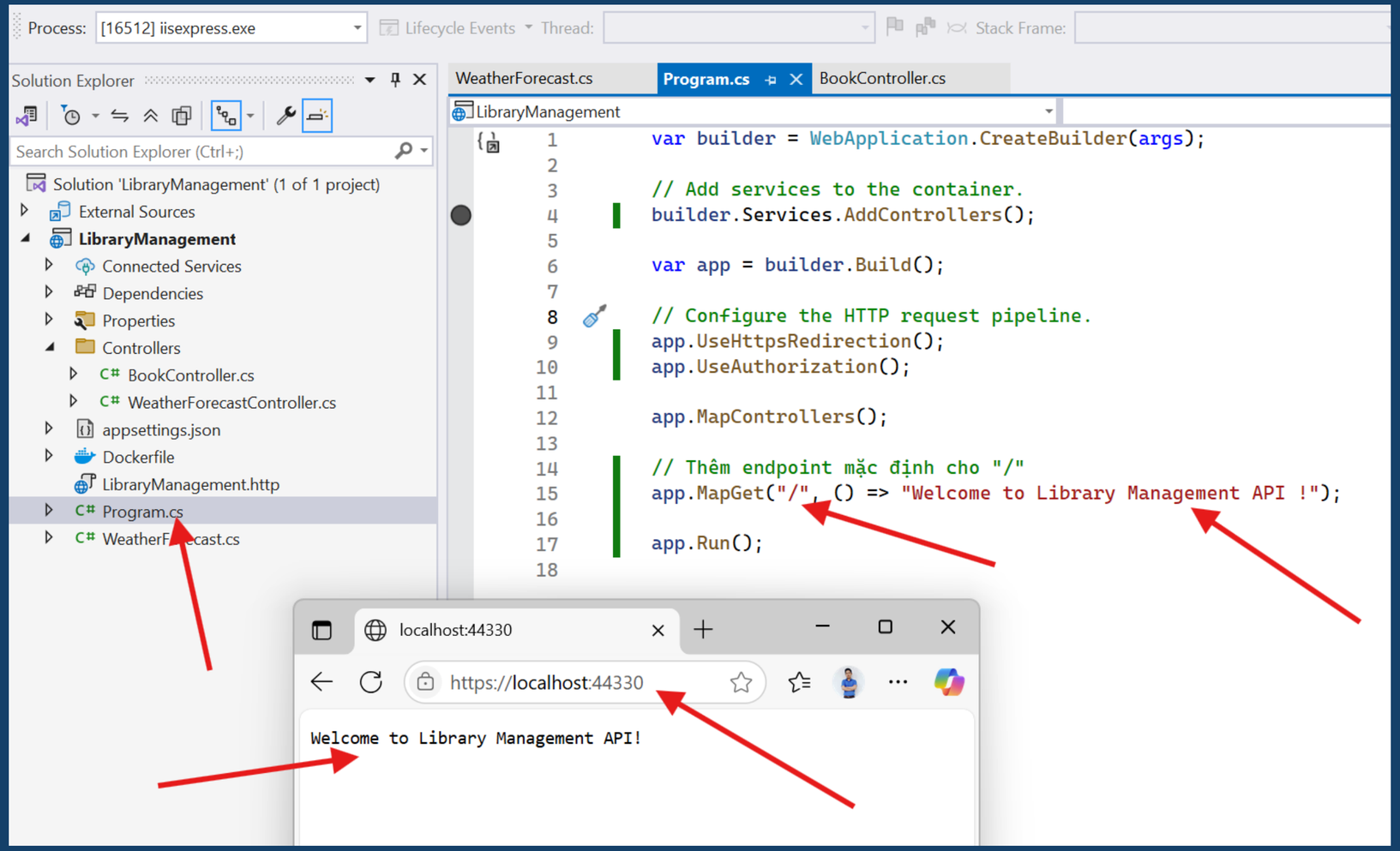
Every public method in a controller is callable as an HTTP endpoint
An HTTP endpoint:
Is a targetable URL in the web application, such as https://domain:port/Book.
Combines:

- The protocol used: HTTPS.
- The network location of the web server, including the TCP port: https://localhost:44330.
- The target URI: Book List.

ASP.NET Core API

Controller – Default Endpoint

Every time a user accesses the website domain, the default path will be set in Program.cs



Model

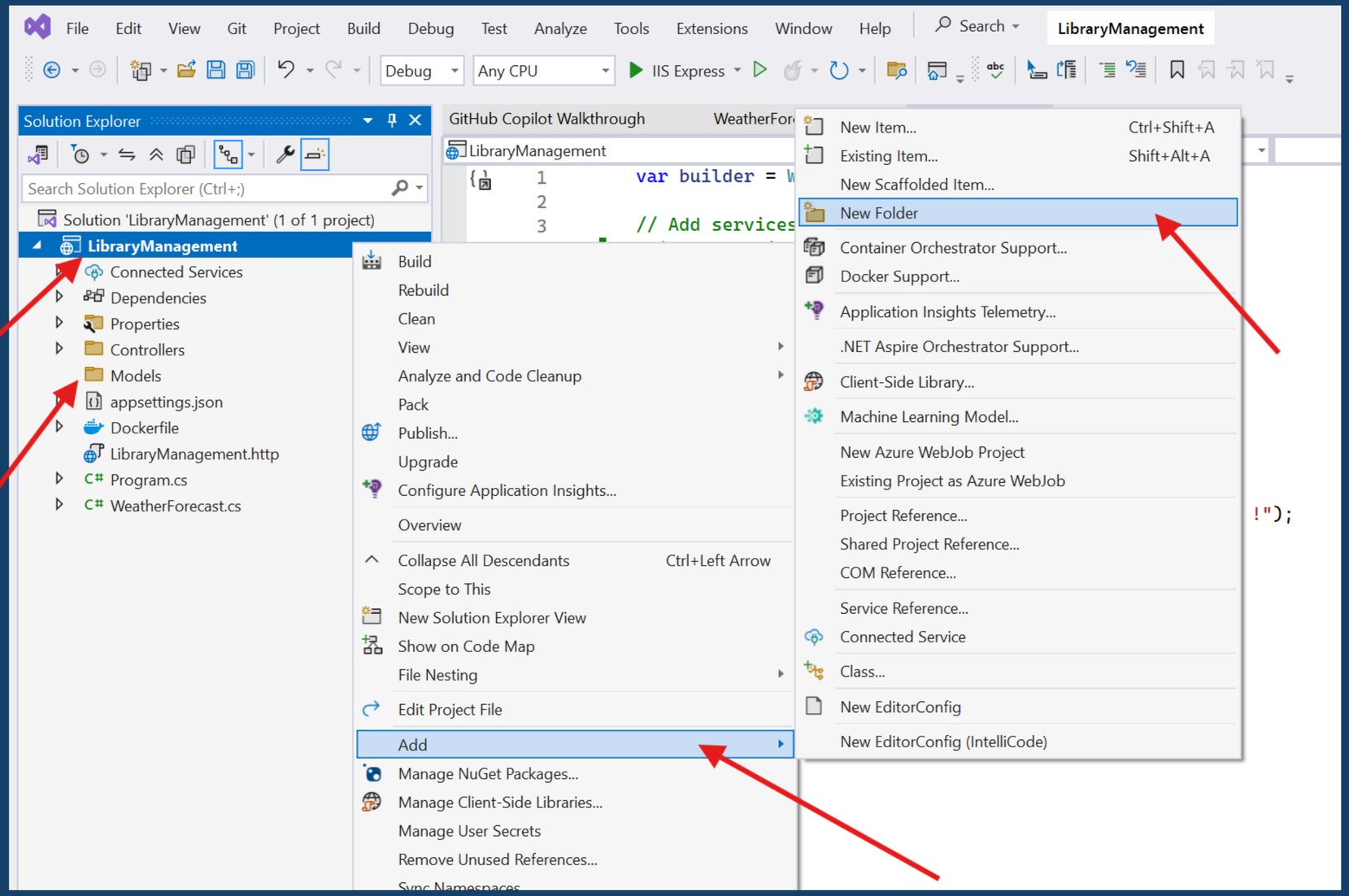
Model

These model classes are used with Entity Framework Core (EF Core) to work with a database. EF Core is an object-relational mapping (ORM) framework that simplifies the data access code that you have to write.

The model classes created are known as POCO classes, from Plain Old CLR Objects. POCO classes don't have any dependency on EF Core. They only define the properties of the data to be stored in the database.

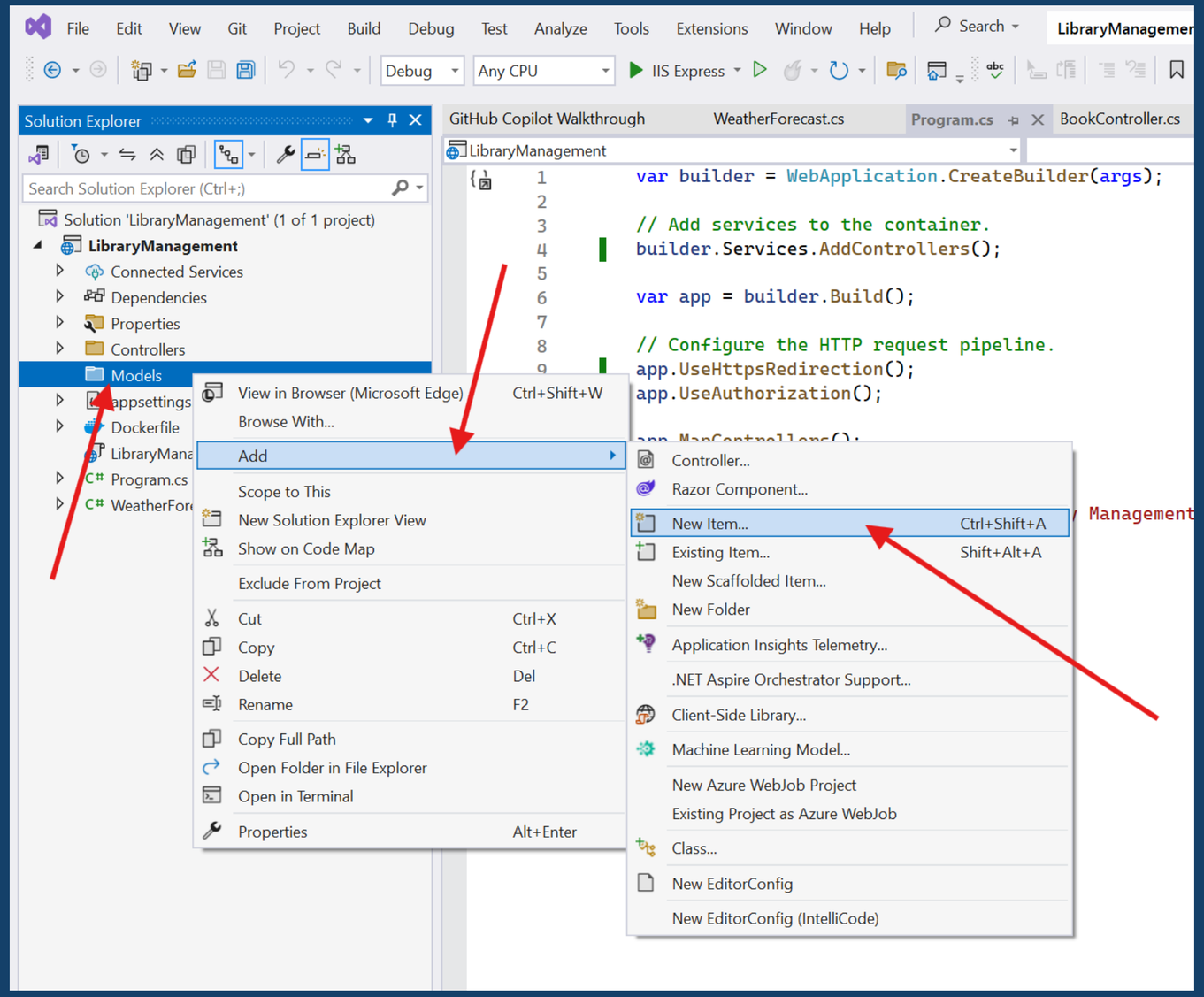
ASP.NET Core API

Model – Add a model to an ASP.NET Core API app



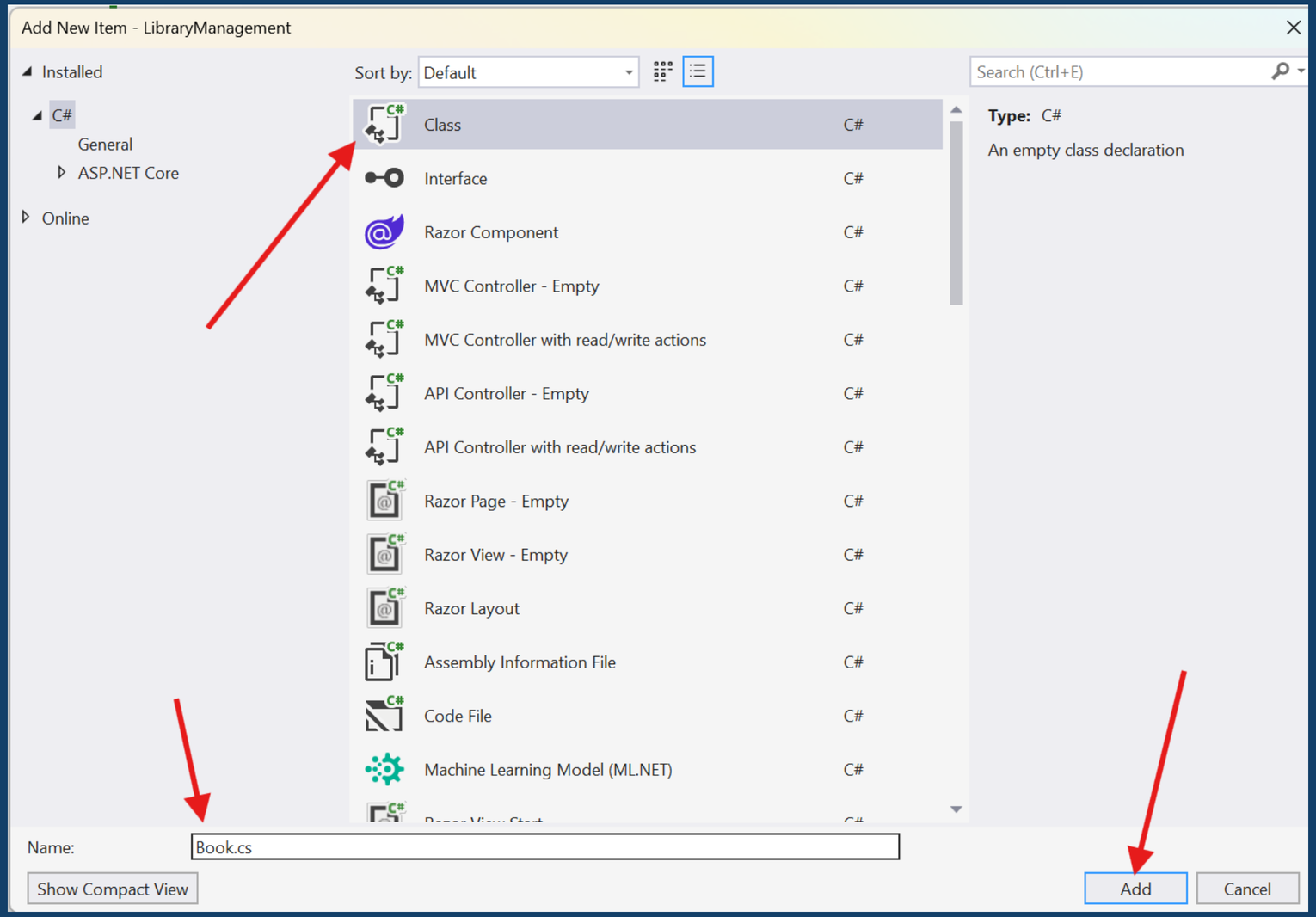
ASP.NET Core API

Model – Add a model to an ASP.NET Core API app



ASP.NET Core API

Model – Add a model to an ASP.NET Core API app



ASP.NET Core API

Model – Add a model to an ASP.NET Core API app

The **Book** class contains an Id field, which is required by the database for the primary key.

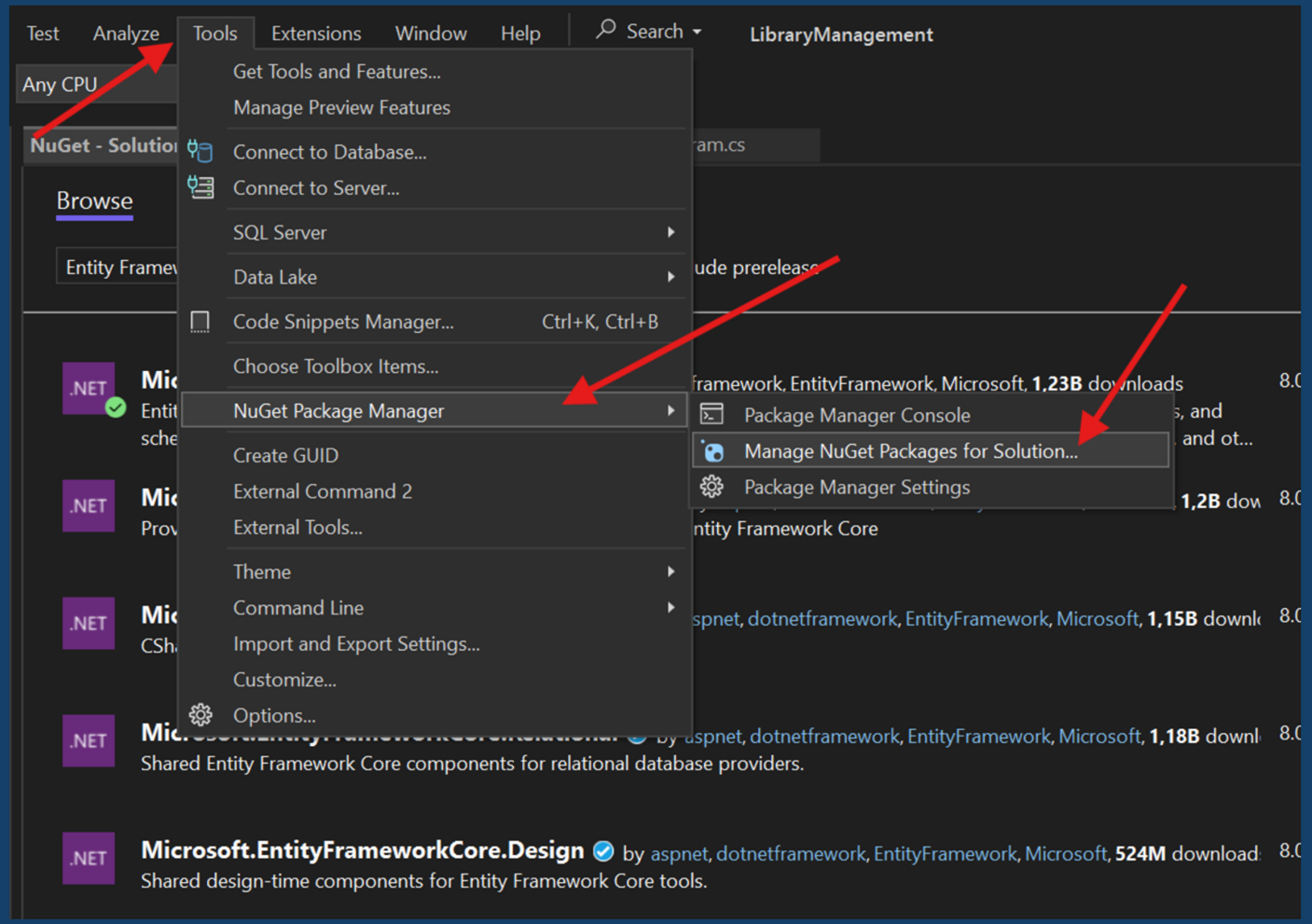
The DataType attribute on ReleaseDate specifies the type of the data (Date). With this attribute:

- The user isn't required to enter time information in the date field.
- Only the date is displayed, not time information.

```
nent                                        LibraryManagement.M
namespace LibraryManagement.Models
{
    7 references
    public class Book
    {
        0 references
        public int Id { get; set; }
        5 references
        public string NameBook { get; set; }
        5 references
        public string Author { get; set; }
    }
}
```

ASP.NET Core API

Model – Entity Framework

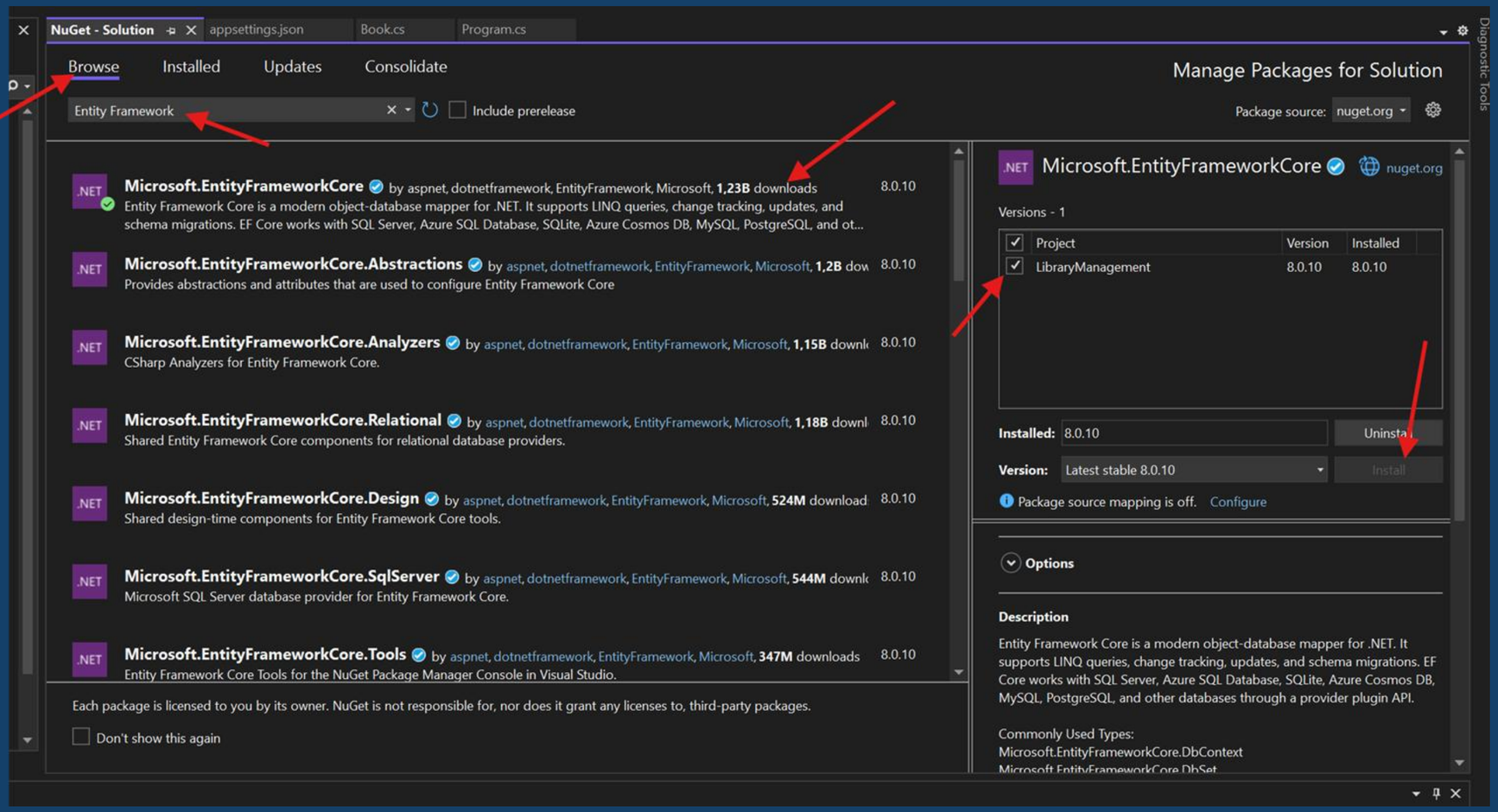


ASP.NET Core MVC

Model – Entity Framework

Microsoft.EntityFrameworkCore.SqlServer

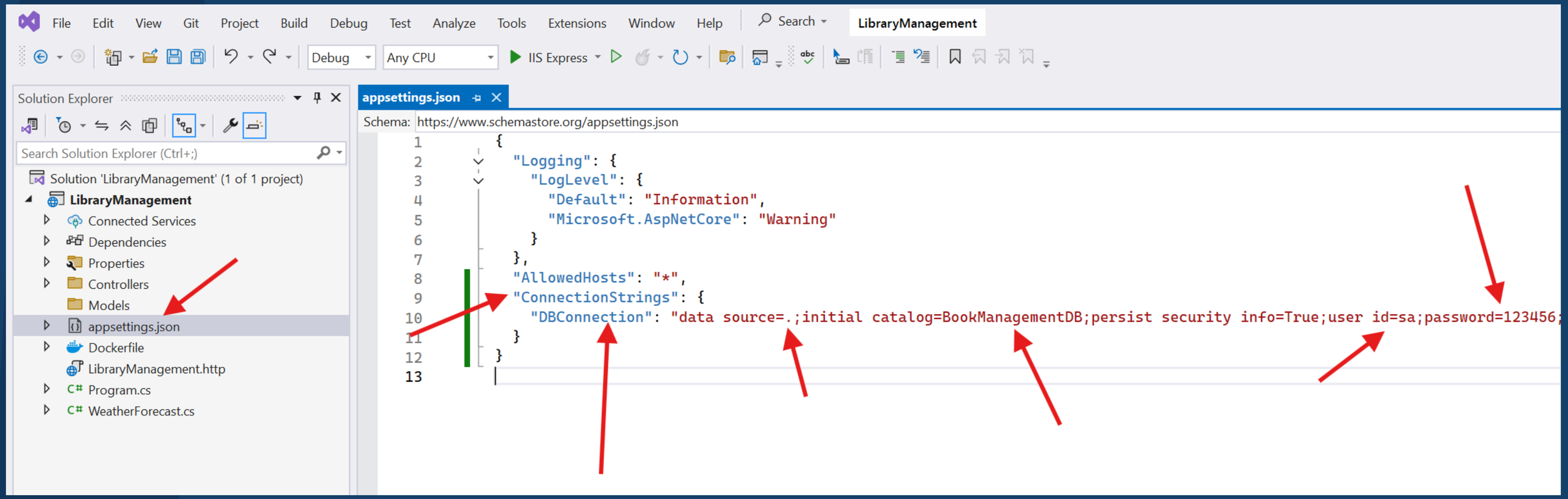
Microsoft.EntityFrameworkCore



ASP.NET Core MVC

Model – appsetting.json

```
"ConnectionStrings": {
    "DBConnection": "data source=.;initial catalog=BookManagementDB;persist security info=True;user
id=sa;password=123456;MultipleActiveResultSets=True;encrypt=false"
  }
```



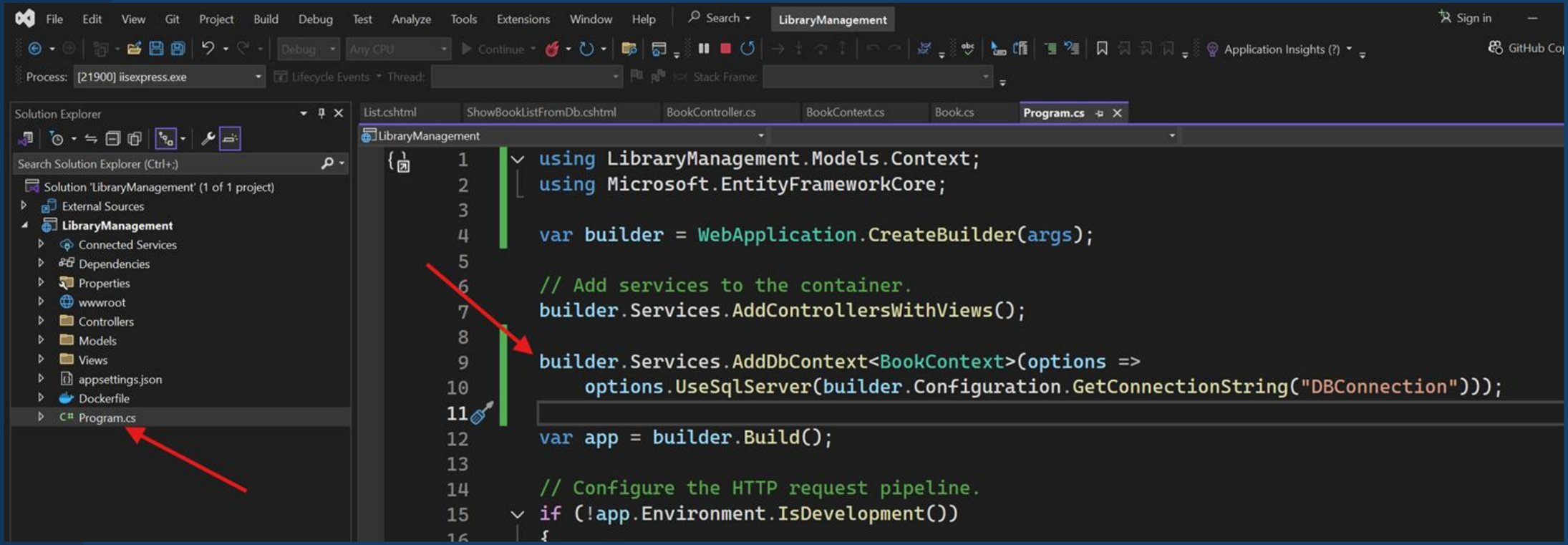
ASP.NET Core API

Model – Dependency injection

- ASP.NET Core is built with dependency injection (DI). Services, such as the database context, are registered with DI in Program.cs. These services are provided to components that require them via constructor parameters.
- In the Controllers/MoviesController.cs file, the constructor uses Dependency Injection to inject the MvcMovieContext database context into the controller. The database context is used in each of the CRUD methods in the controller.
- Scaffolding generated the following highlighted code in Program.cs:

ASP.NET Core API

Model – Dependency injection



ASP.NET Core API

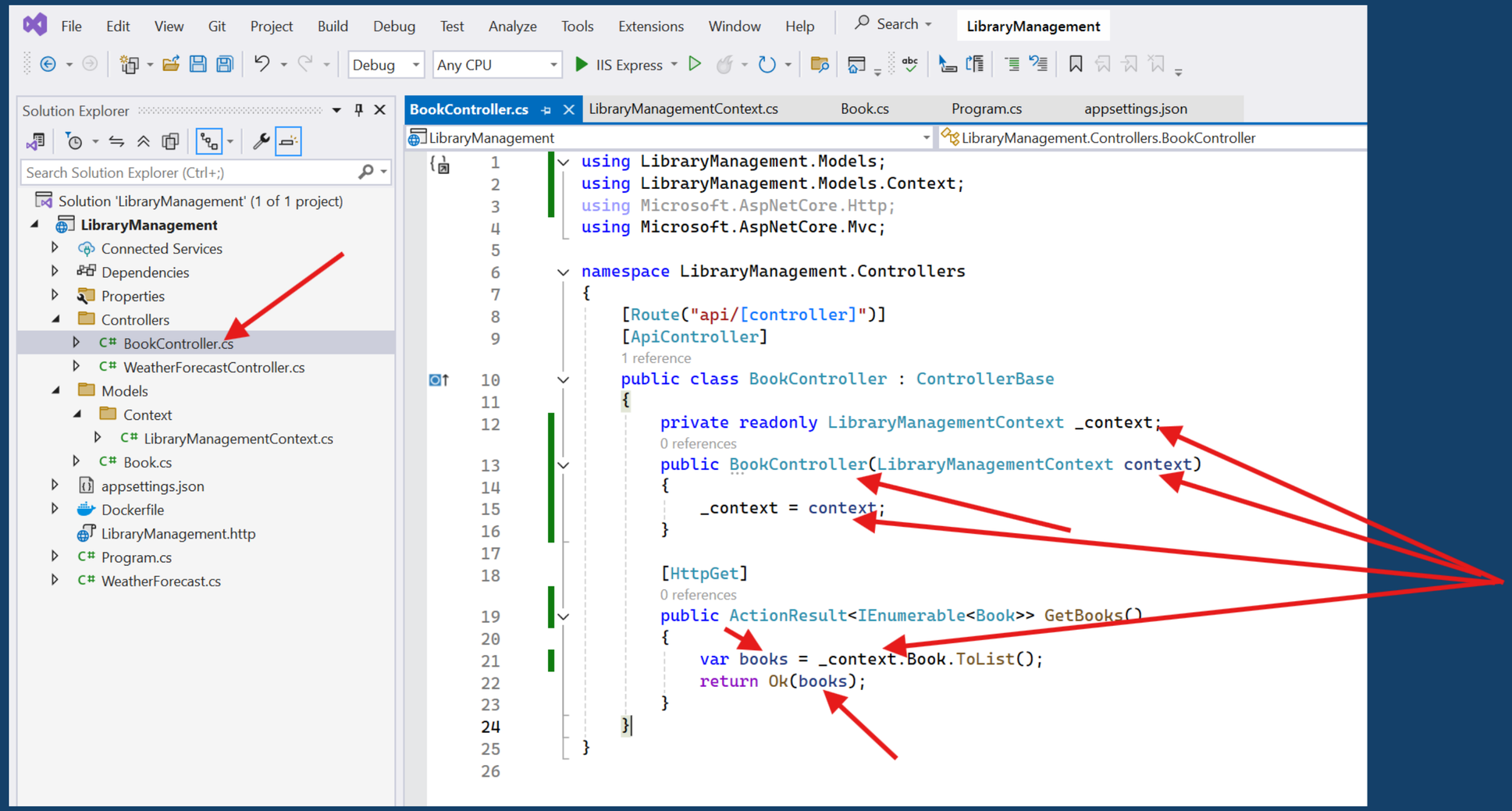
Model – Dbcontext

```
using Microsoft.EntityFrameworkCore;

namespace LibraryManagement.Models.Context
{
    2 references
    public class LibraryManagementContext : DbContext
    {
        0 references
        public LibraryManagementContext(DbContextOptions<LibraryManagementContext> options) : base(options) { }
        0 references
        public DbSet<Book> Book { get; set; }
    }
}
```

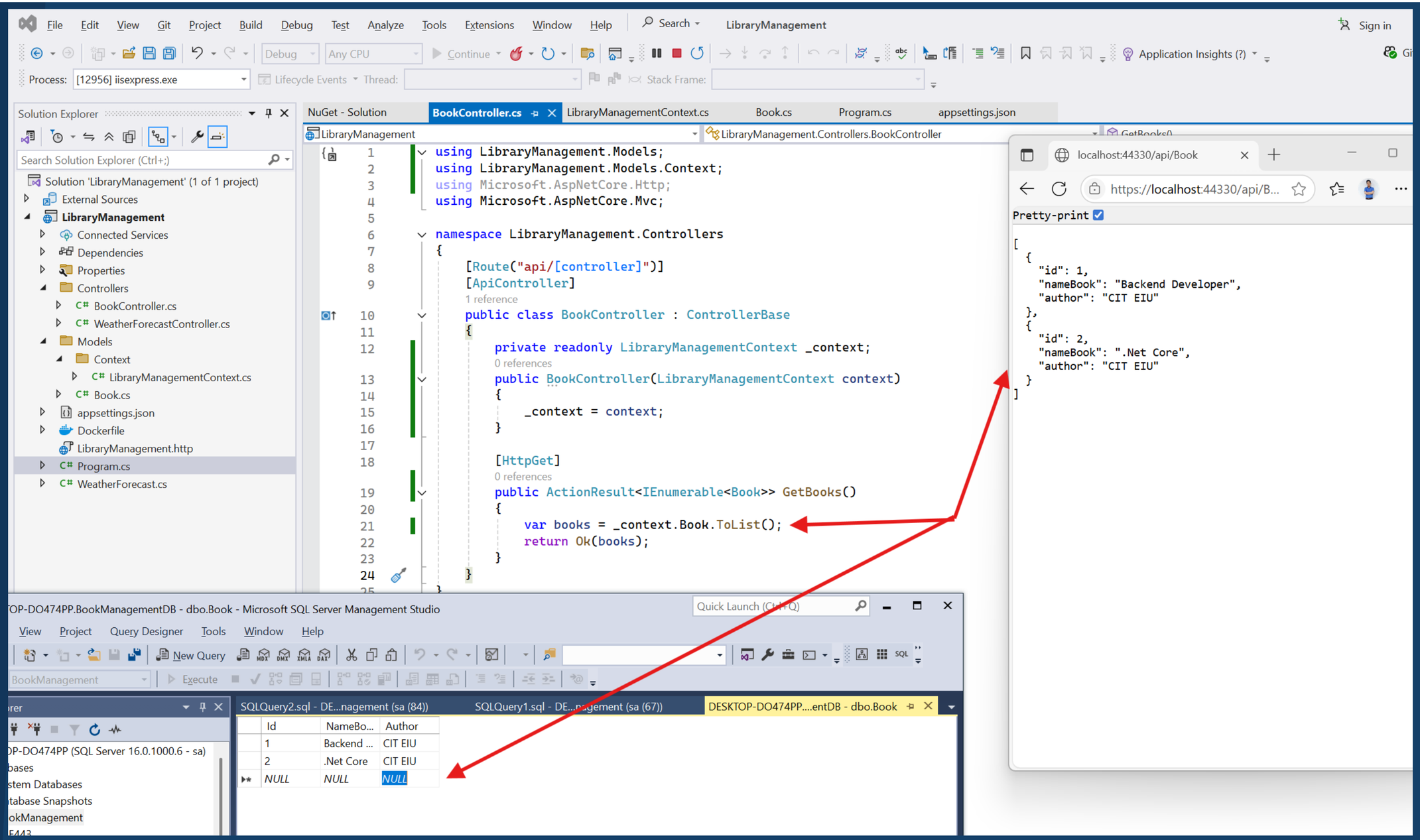
```
namespace LibraryManagement.Models
{
    1 reference
    public class Book
    {
        0 references
        public int Id { get; set; }
        0 references
        public string NameBook { get; set; } = string.Empty;
        0 references
        public string Author { get; set; } = string.Empty;


    }
}
```

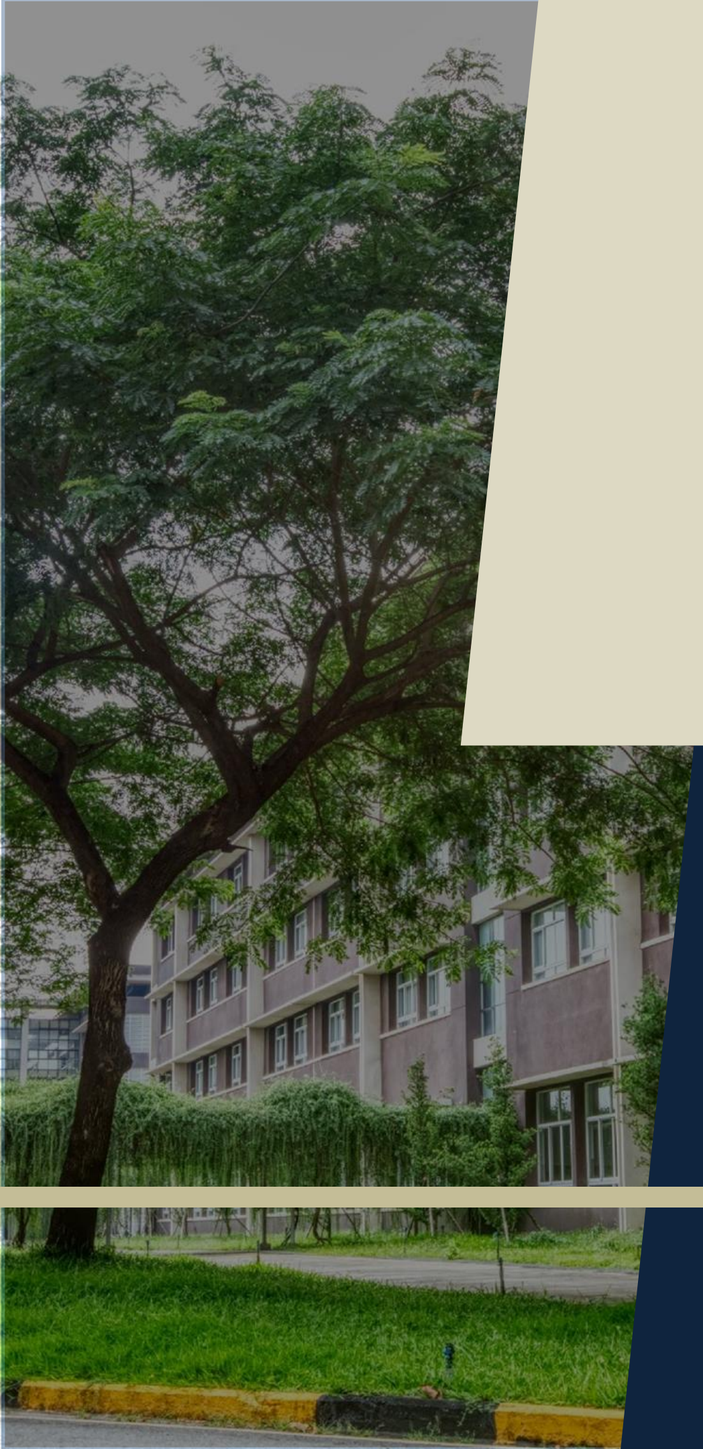
ASP.NET Core API

Model – Dependency injection in the controller



ASP.NET Core API





EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Start your future at EIU

Thank You