

# 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:

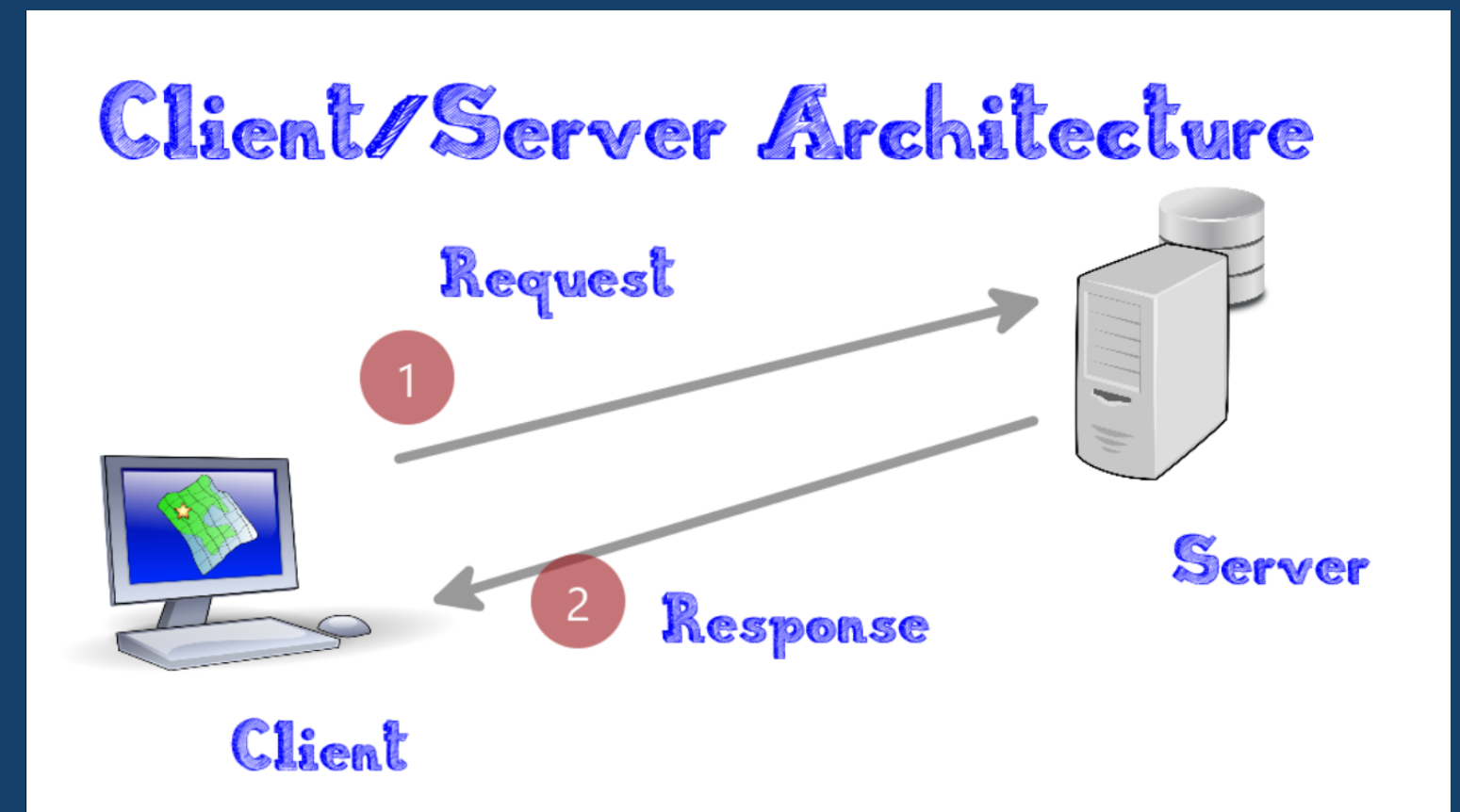
- A model that divides a system into:
  - **Client:** sends requests, displays UI (browser, mobile app, etc.)
  - **Server:** processes requests, accesses data, returns responses
- Communicates via **HTTP/HTTPS**, commonly using **REST API** and **JSON**

## 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...)



## 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

**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.





**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
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

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.



**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.

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

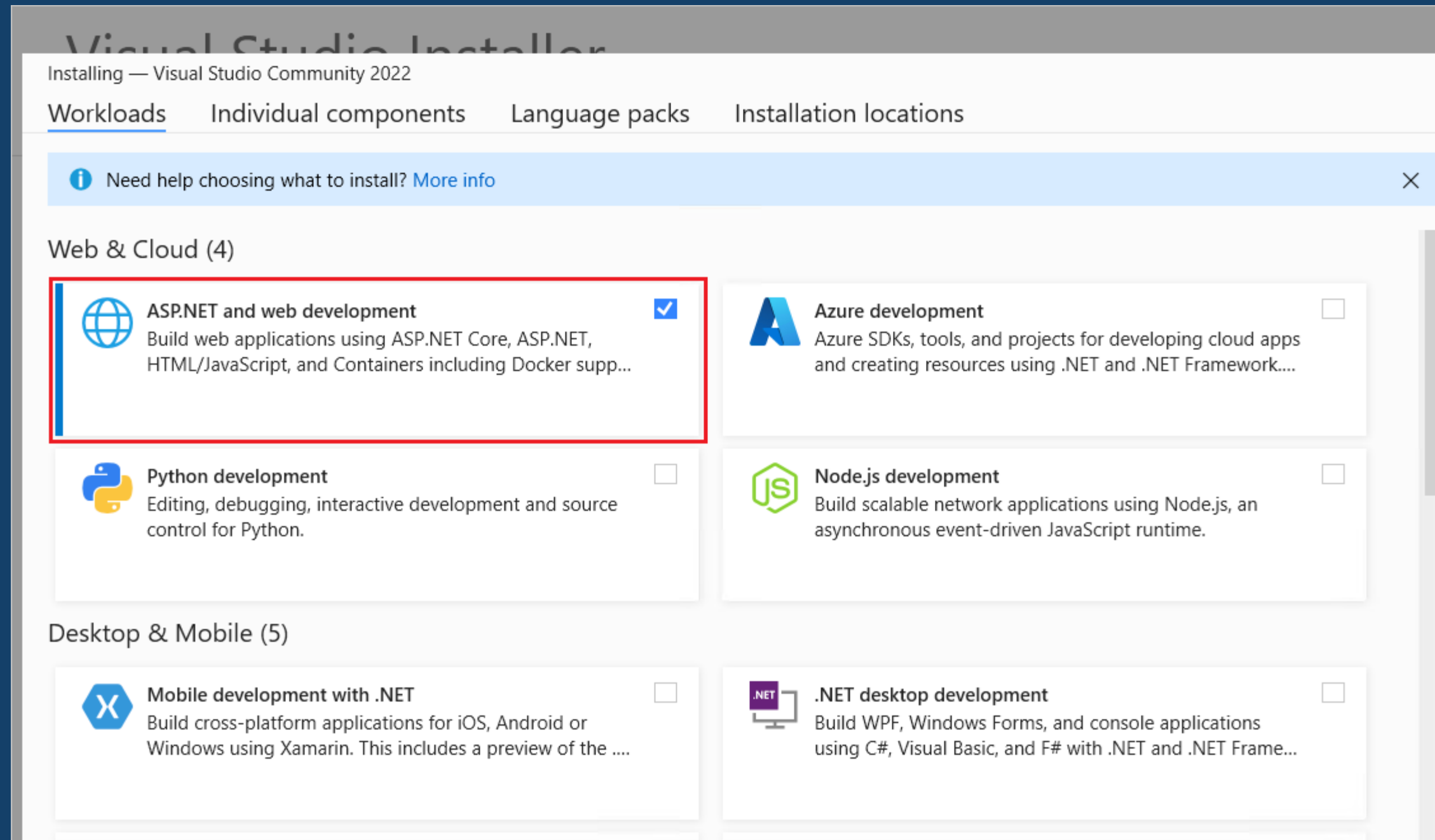
**Supported platforms:** Cross-platform (Windows, macOS, Linux).

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

	ASP.NET	ASP.NET Core
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

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

Prerequisites: .NET Core 8.0

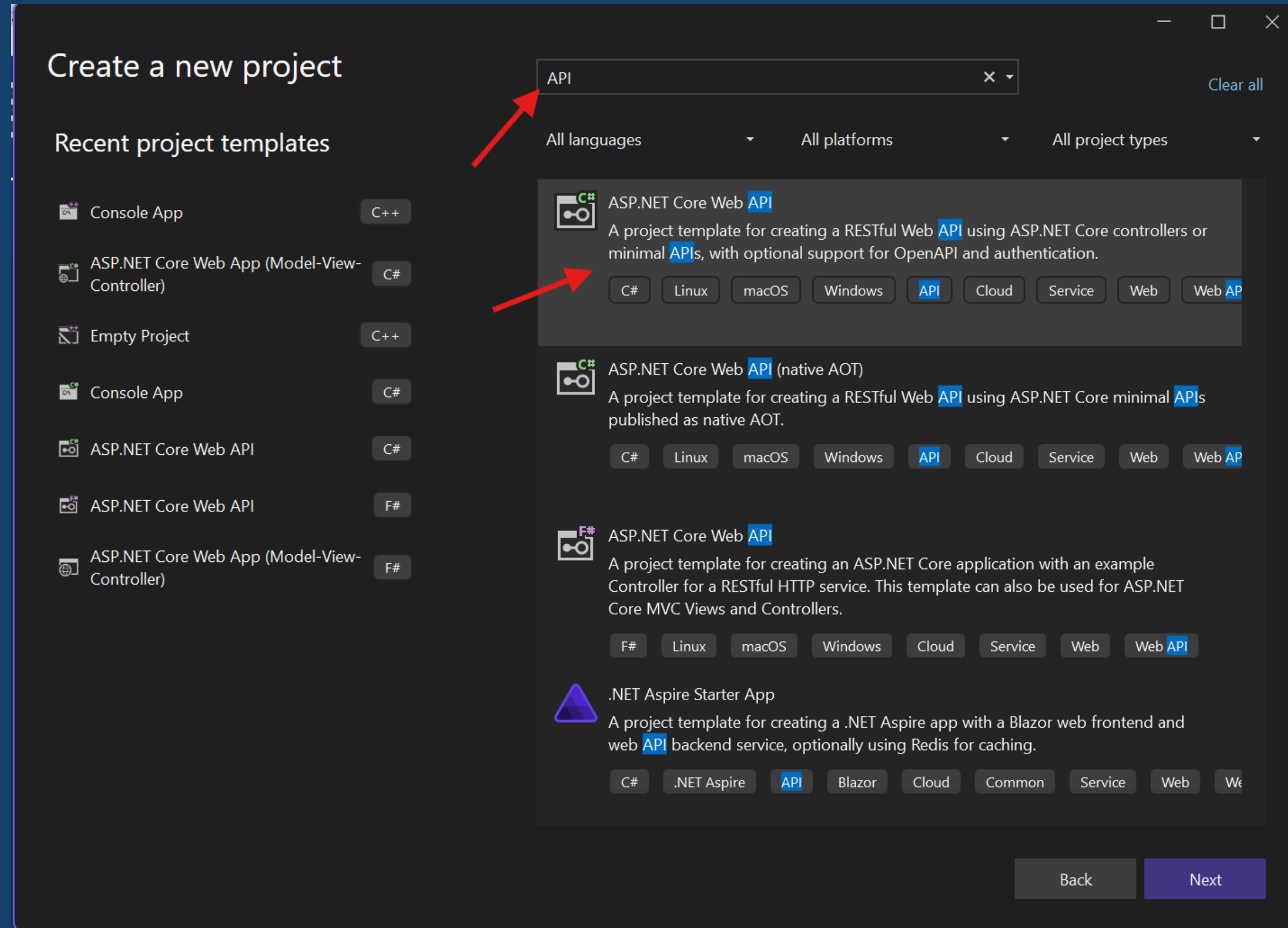


## Create a web app:

- 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.



Create a web app:



Create a web app:

Configure your new project

ASP.NET Core Web API

C#LinuxmacOSWindowsAPICloudServiceWebWeb API

Project name

LibraryManagement

Location

C:\Users\NXC\Desktop

...

Solution name ⓘ

LibraryManagement

☐ Place solution and project in the same directory

Project will be created in "C:\Users\NXC\Desktop\LibraryManagement\LibraryManagement\"

Back

Next

Create a web app:

## Additional information

ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web Web API

Framework ⓘ

.NET 8.0 (Long Term Support)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☒ Enable container support ⓘ

Container OS ⓘ

Windows

Container build type ⓘ

Dockerfile

☐ Enable OpenAPI support ⓘ

☐ Do not use top-level statements ⓘ

☒ Use controllers ⓘ

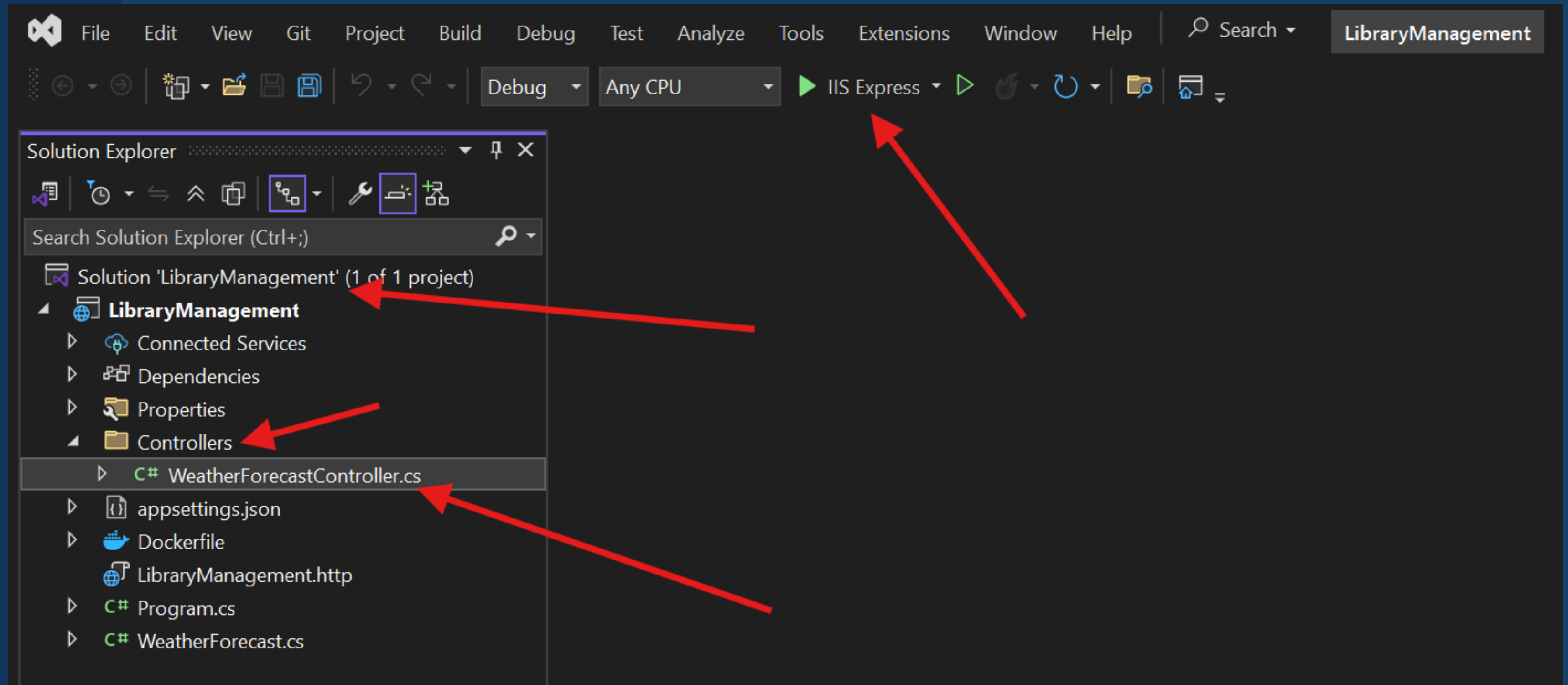
☐ Enlist in .NET Aspire orchestration ⓘ

Aspire version ⓘ

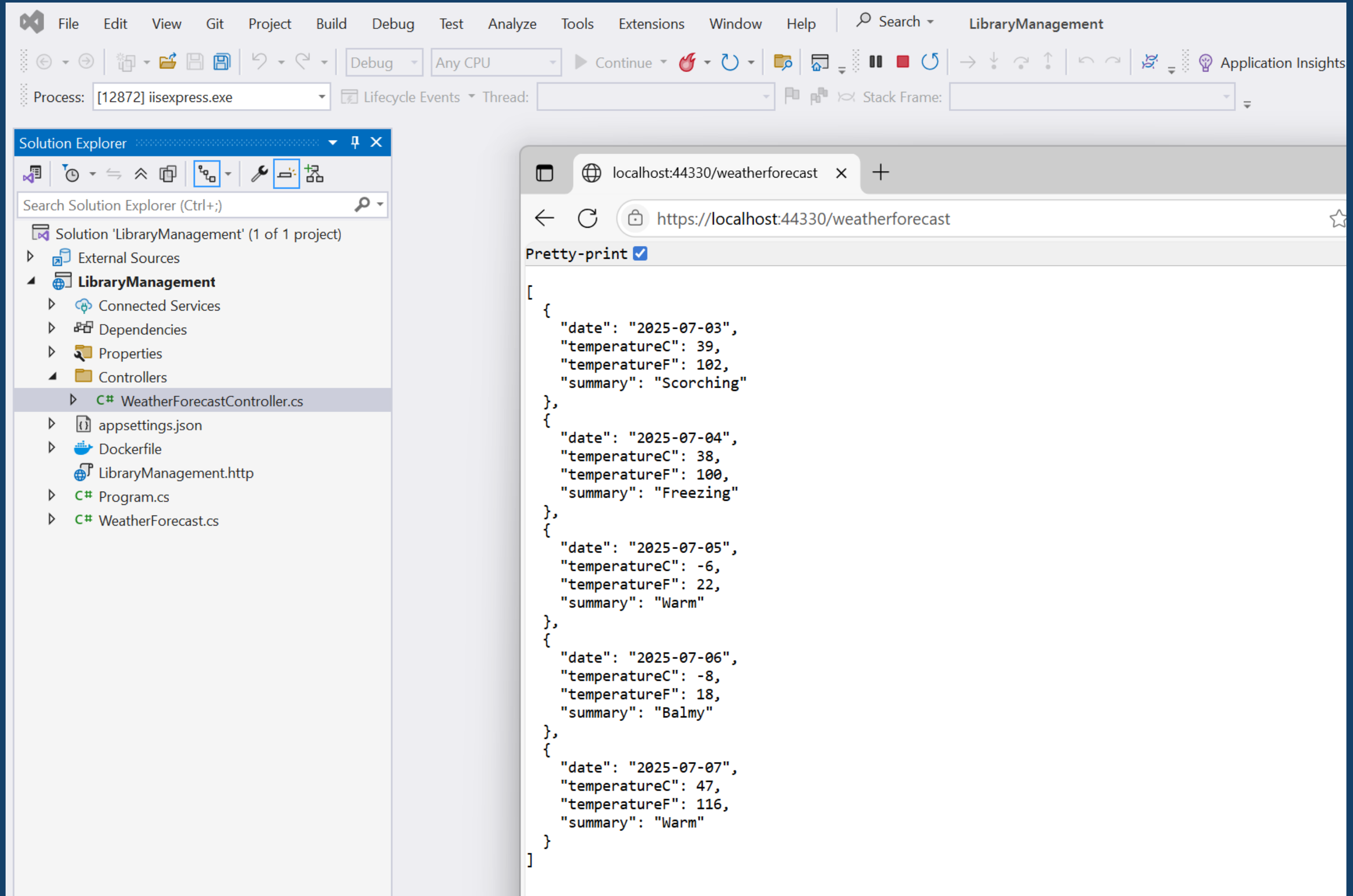
9.0

Back Create

## Run the app



## Run the app



The screenshot displays the Visual Studio IDE with the 'LibraryManagement' project open. The Solution Explorer on the left shows the project structure, including 'WeatherForecastController.cs'. The main editor area shows the code for 'WeatherForecastController.cs'. The web browser on the right shows the API response at 'localhost:44330/weatherforecast', displaying a JSON array of weather forecast data.

**Visual Studio Interface:**

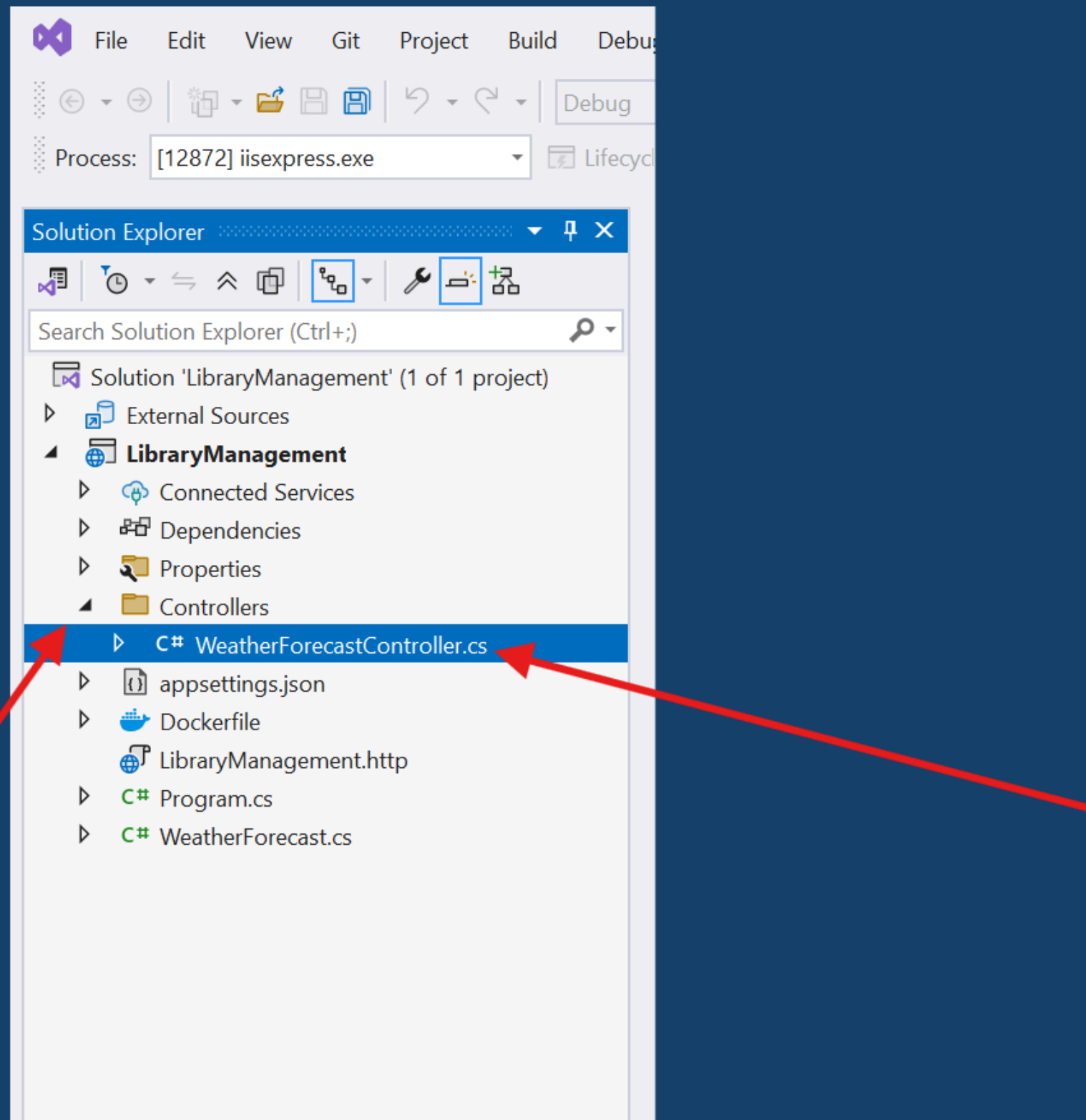
- Menu Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Toolbar:** Includes icons for File Explorer, Solution Explorer, Search, and various development tools.
- Process:** [12872] iisexpress.exe
- Debug Console:** Shows 'Continue' and 'Any CPU'.
- Solution Explorer:**
  - Solution 'LibraryManagement' (1 of 1 project)
  - External Sources
  - LibraryManagement**
    - Connected Services
    - Dependencies
    - Properties
    - Controllers
      - C# WeatherForecastController.cs**
  - appsettings.json
  - Dockerfile
  - LibraryManagement.http
  - C# Program.cs
  - C# WeatherForecast.cs

**Web Browser Interface:**

- Address Bar:** localhost:44330/weatherforecast
- URL:** https://localhost:44330/weatherforecast
- Content:** Pretty-print ☒

```
[
  {
    "date": "2025-07-03",
    "temperatureC": 39,
    "temperatureF": 102,
    "summary": "Scorching"
  },
  {
    "date": "2025-07-04",
    "temperatureC": 38,
    "temperatureF": 100,
    "summary": "Freezing"
  },
  {
    "date": "2025-07-05",
    "temperatureC": -6,
    "temperatureF": 22,
    "summary": "Warm"
  },
  {
    "date": "2025-07-06",
    "temperatureC": -8,
    "temperatureF": 18,
    "summary": "Balmy"
  },
  {
    "date": "2025-07-07",
    "temperatureC": 47,
    "temperatureF": 116,
    "summary": "Warm"
  }
]
```

## Folder struct





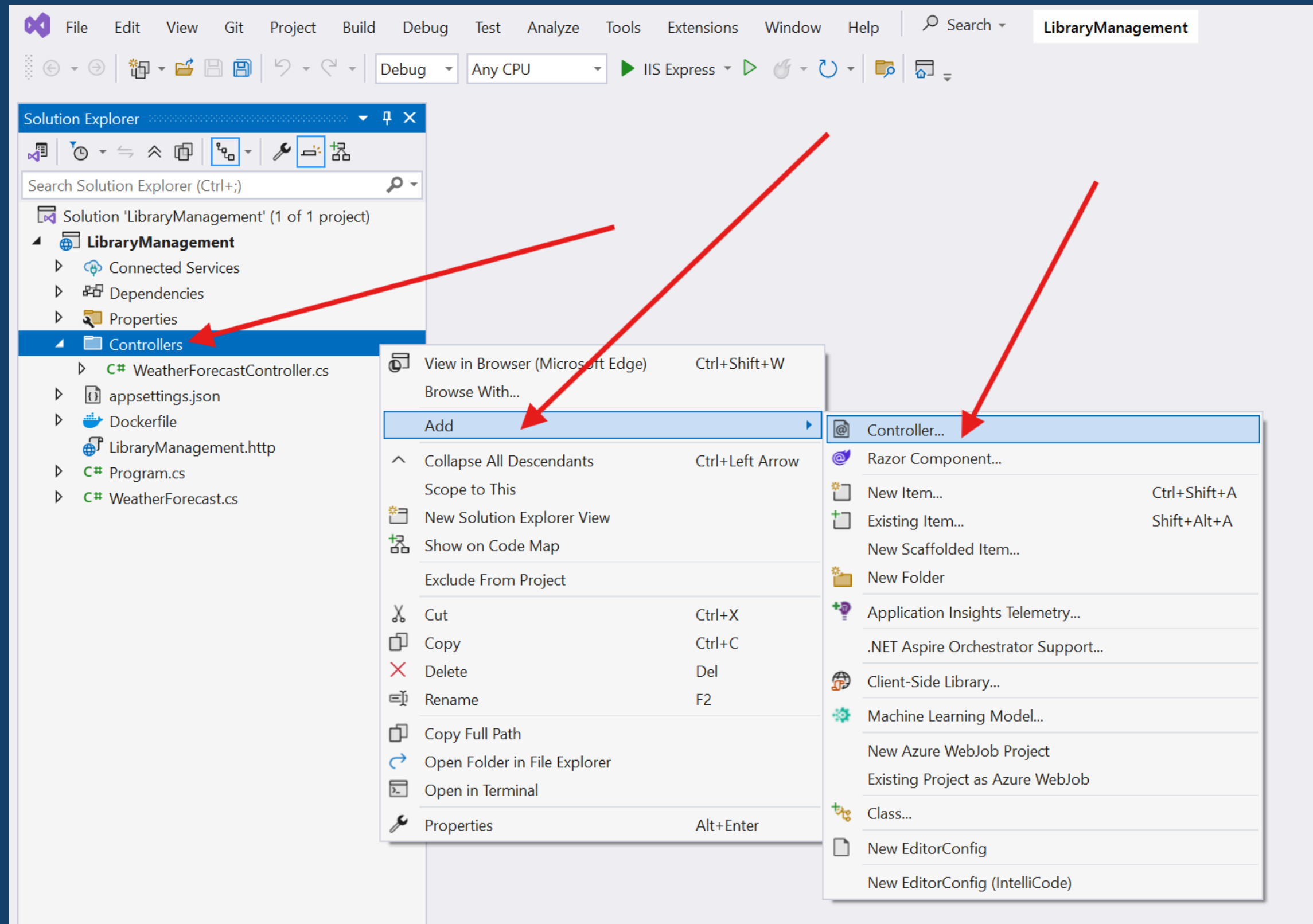
## Controller

- 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.

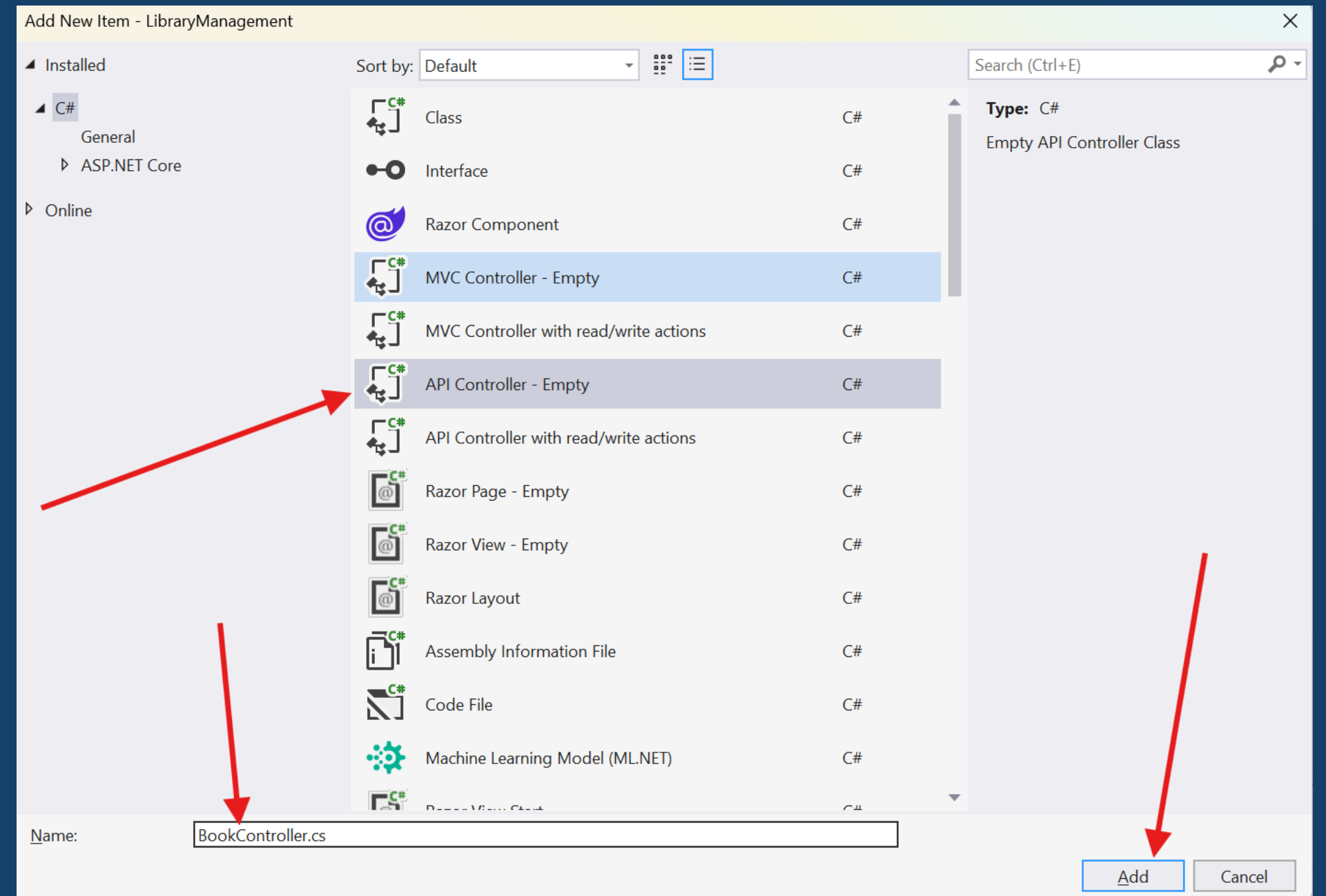
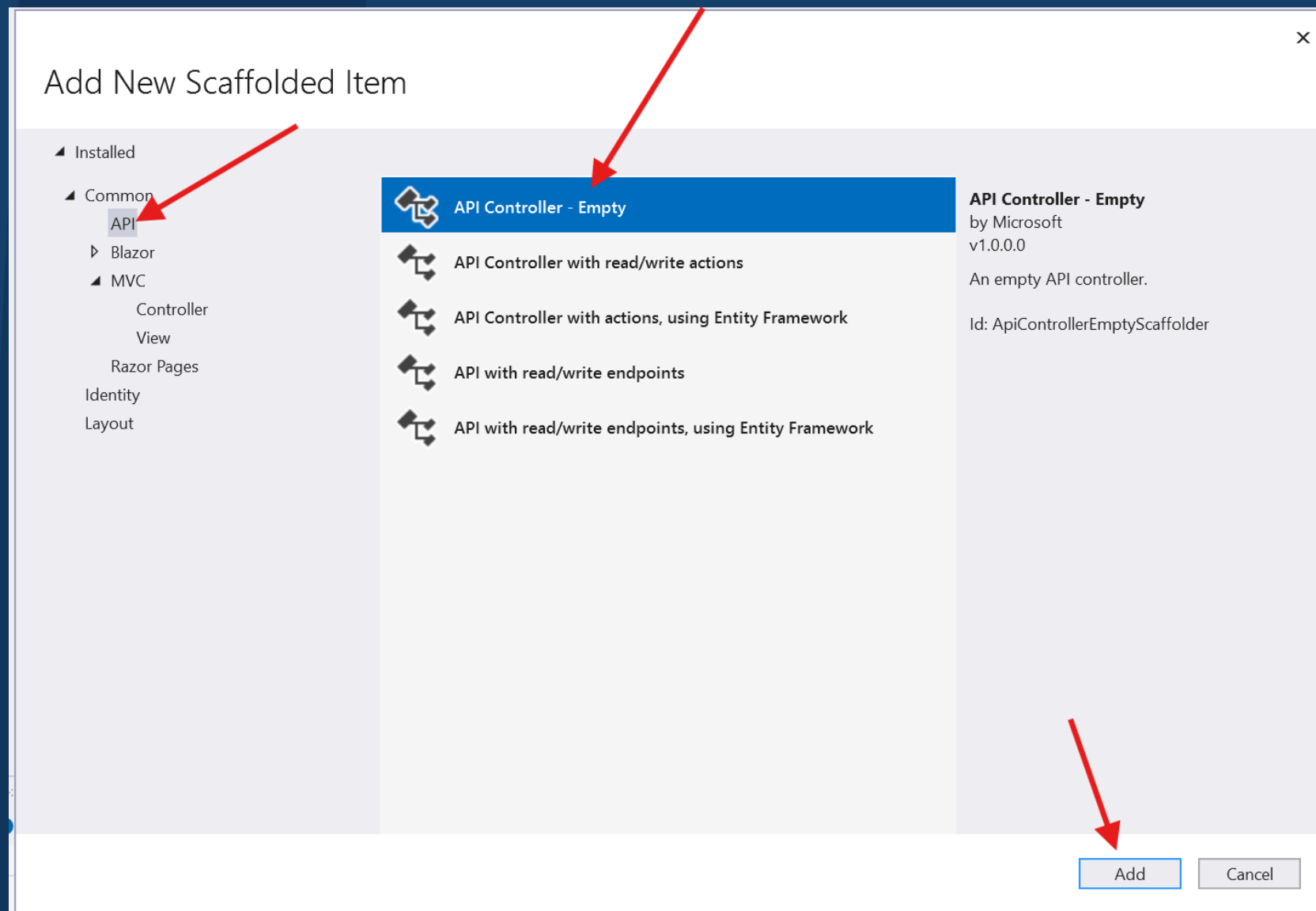
*Asp.net core Web API 8.0*

**Controller**

## Controller - Add a controller



## Controller – Add a controller



## Controller – Index method

The image shows a Visual Studio IDE with the `BookController.cs` file open. The code defines a `BookController` class that inherits from `ControllerBase` and implements the `ApiController` interface. It includes a `BookDto` class with `Title` and `Author` properties. The `GetBooks()` method returns a list of `BookDto` objects representing three books.

```
1 using Microsoft.AspNetCore.Http;
2 using Microsoft.AspNetCore.Mvc;
3
4 namespace LibraryManagement.Controllers
5 {
6     [Route("api/[controller]")]
7     [ApiController]
8     public class BookController : ControllerBase
9     {
10         5 references
11         public class BookDto
12         {
13             3 references
14             public string Title { get; set; }
15             3 references
16             public string Author { get; set; }
17         }
18
19         [HttpGet]
20         0 references
21         public ActionResult<IEnumerable<BookDto>> GetBooks()
22         {
23             var books = new List<BookDto>
24             {
25                 new BookDto { Title = "The Great Gatsby", Author = "F. Scott Fitzgerald" },
26                 new BookDto { Title = "To Kill a Mockingbird", Author = "Harper Lee" },
27                 new BookDto { Title = "1984", Author = "George Orwell" }
28             };
29             return Ok(books);
30         }
31     }
32 }
```

Red arrows indicate the flow of data from the `GetBooks()` method in the controller to the browser window, which displays the JSON response:

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

## Controller – HTTP Endpoint

Every public method in a controller is callable as an HTTP endpoint

An HTTP endpoint:

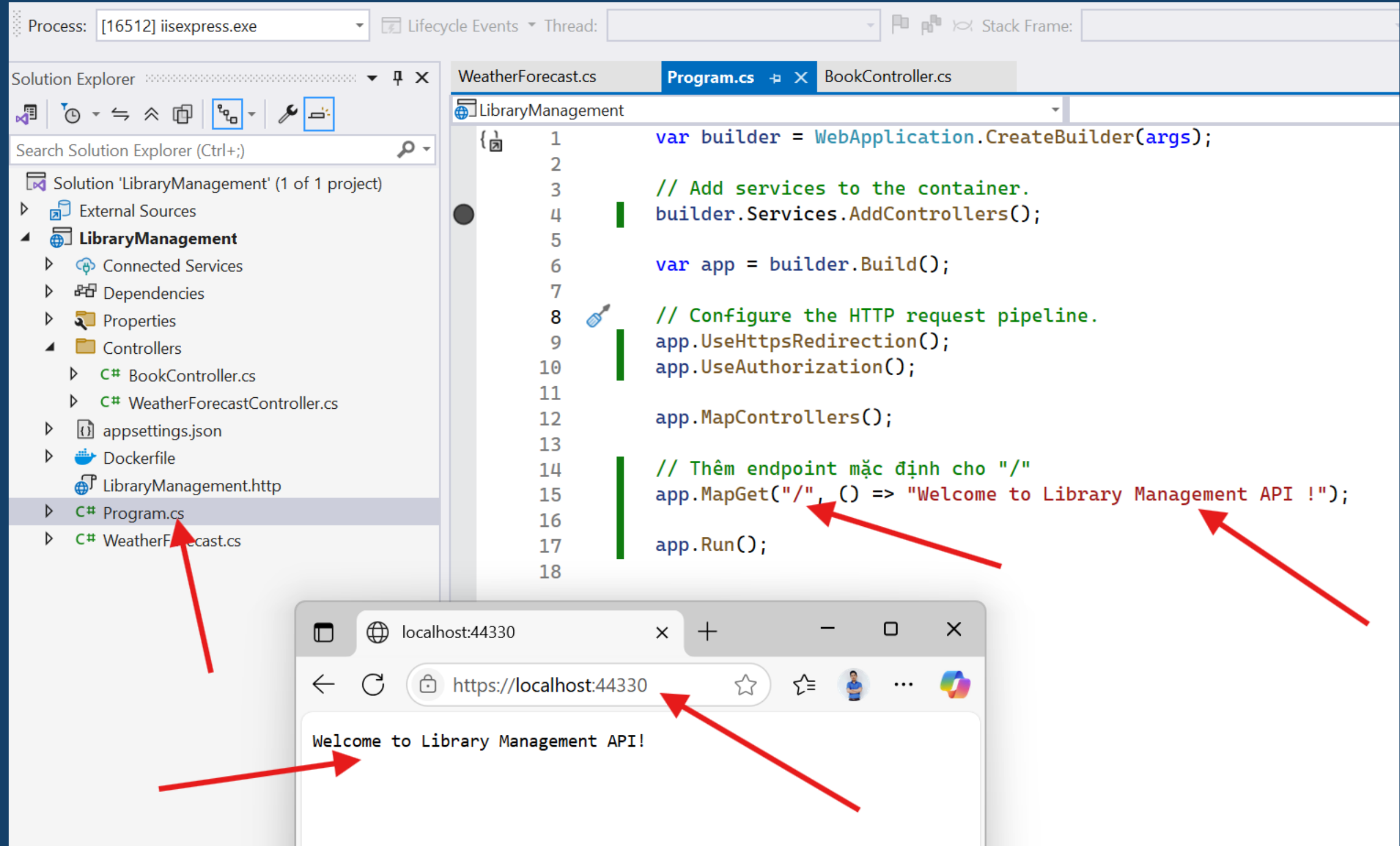
Is a targetable URL in the web application, such as `https://domain:port/Book`.

Combines:

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

## Controller – Default Endpoint

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



The screenshot illustrates the configuration of the default endpoint in an ASP.NET Core API project. The Visual Studio interface shows the Solution Explorer on the left, highlighting the 'LibraryManagement' project and the 'Program.cs' file. The main editor displays the code in 'Program.cs', which configures the application's services and endpoints. A red arrow points from the 'Program.cs' file in the Solution Explorer to the code in the editor. Another red arrow points from the 'app.MapGet("/", () => "Welcome to Library Management API !");' line in the code to a web browser window. The browser window shows the URL 'https://localhost:44330' and the response 'Welcome to Library Management API!', with a red arrow pointing from the browser to the text 'Welcome to Library Management API!'.

```

1  var builder = WebApplication.CreateBuilder(args);
2
3  // Add services to the container.
4  builder.Services.AddControllers();
5
6  var app = builder.Build();
7
8  // Configure the HTTP request pipeline.
9  app.UseHttpsRedirection();
10 app.UseAuthorization();
11
12 app.MapControllers();
13
14 // Thêm endpoint mặc định cho "/"
15 app.MapGet("/", () => "Welcome to Library Management API !");
16
17 app.Run();
18

```



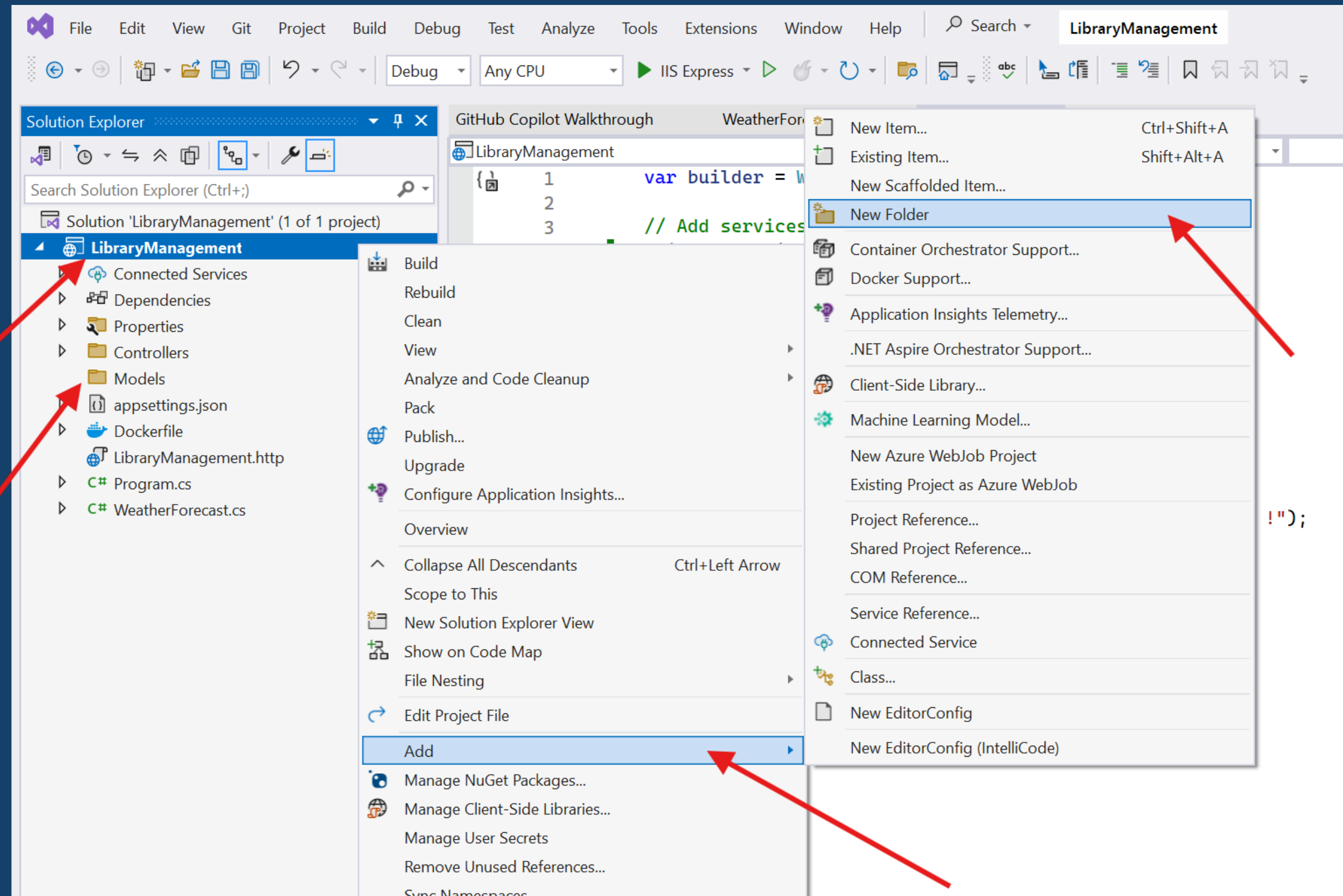
## Model

## Model

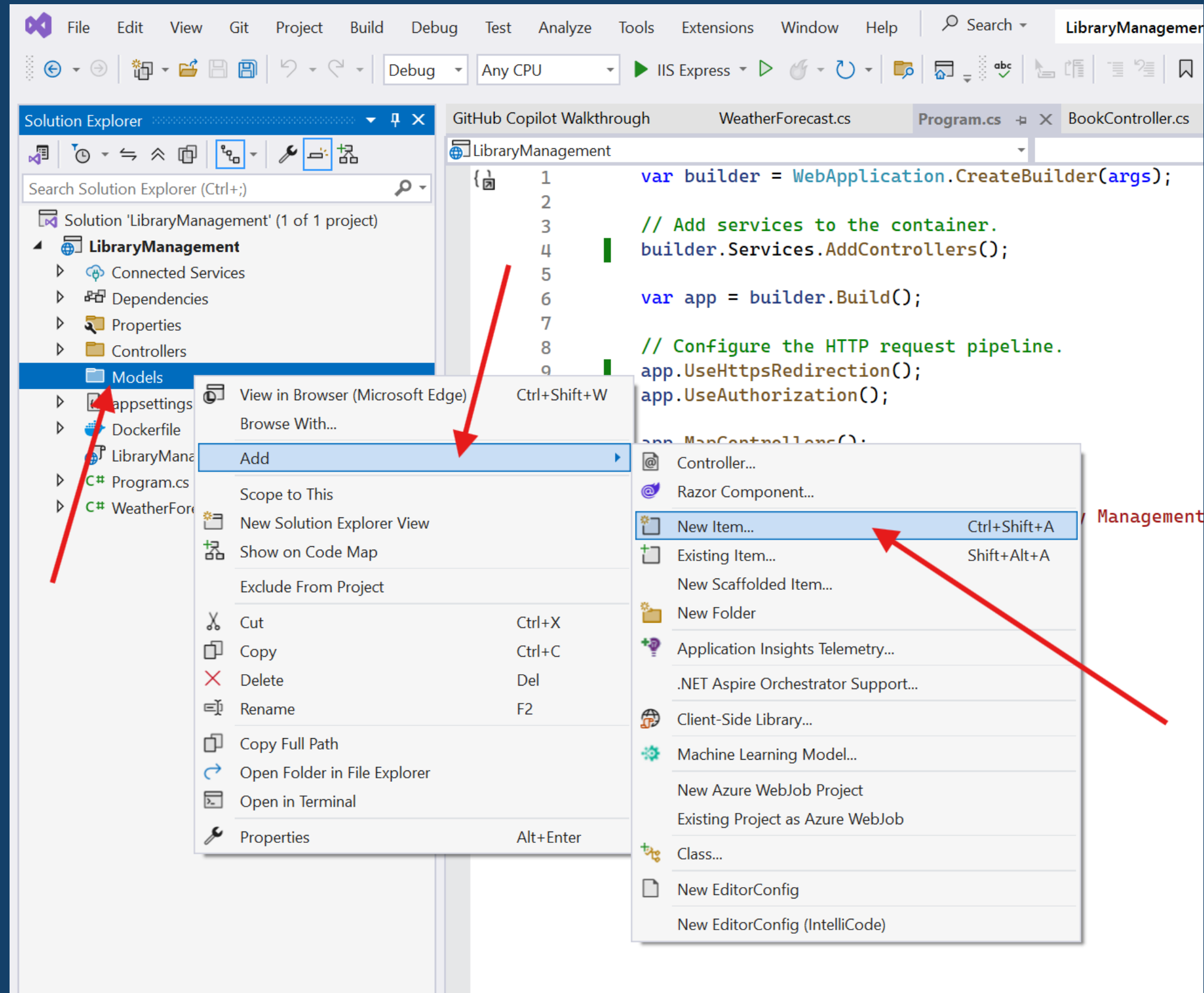
These model classes are used with Entity Framework Core (EF Core) to work with a database. EF Core is an object-relational mapping (ORM) framework that simplifies the data access code that you have to write.

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.

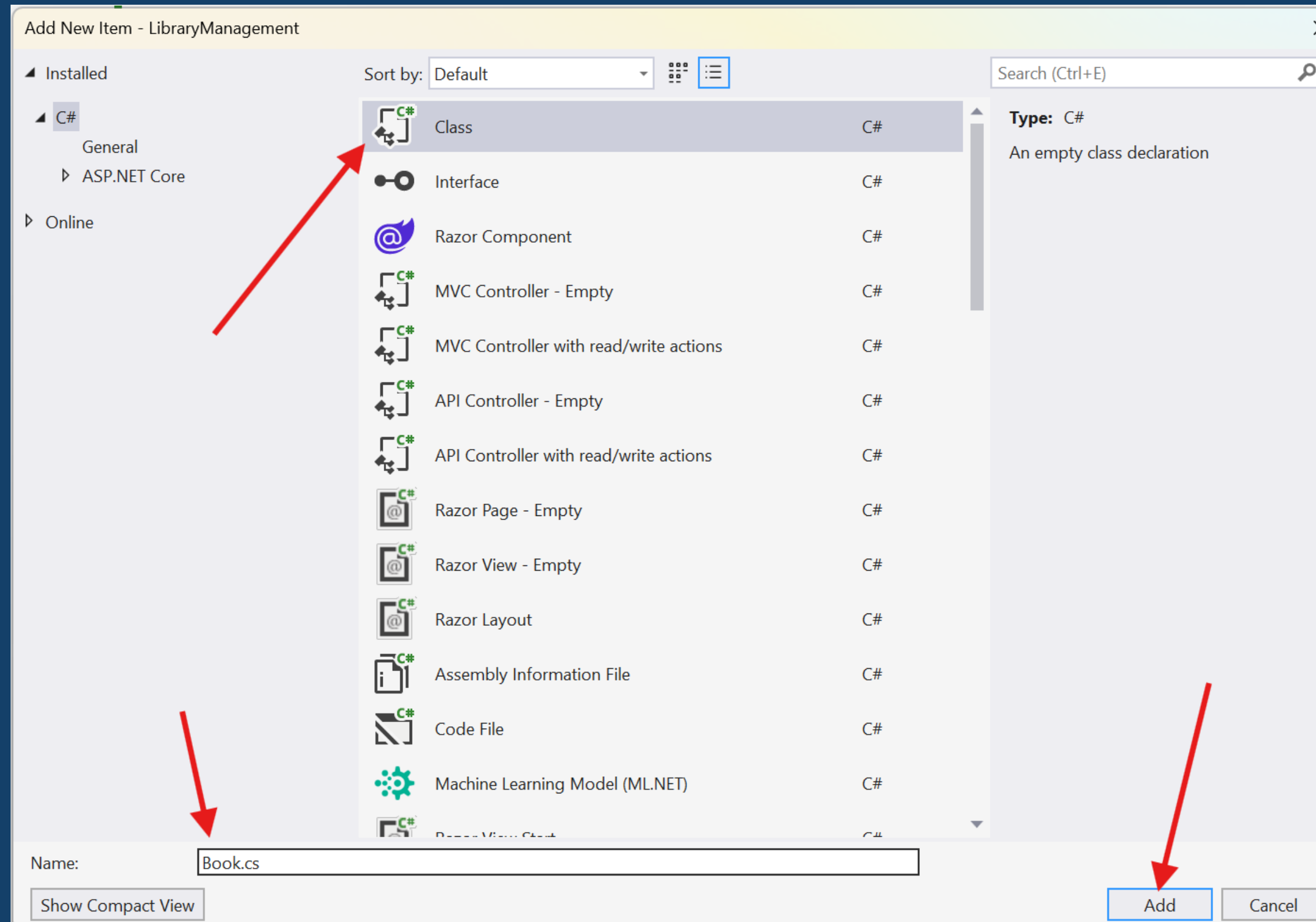
## Model – Add a model to an ASP.NET Core API app



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



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

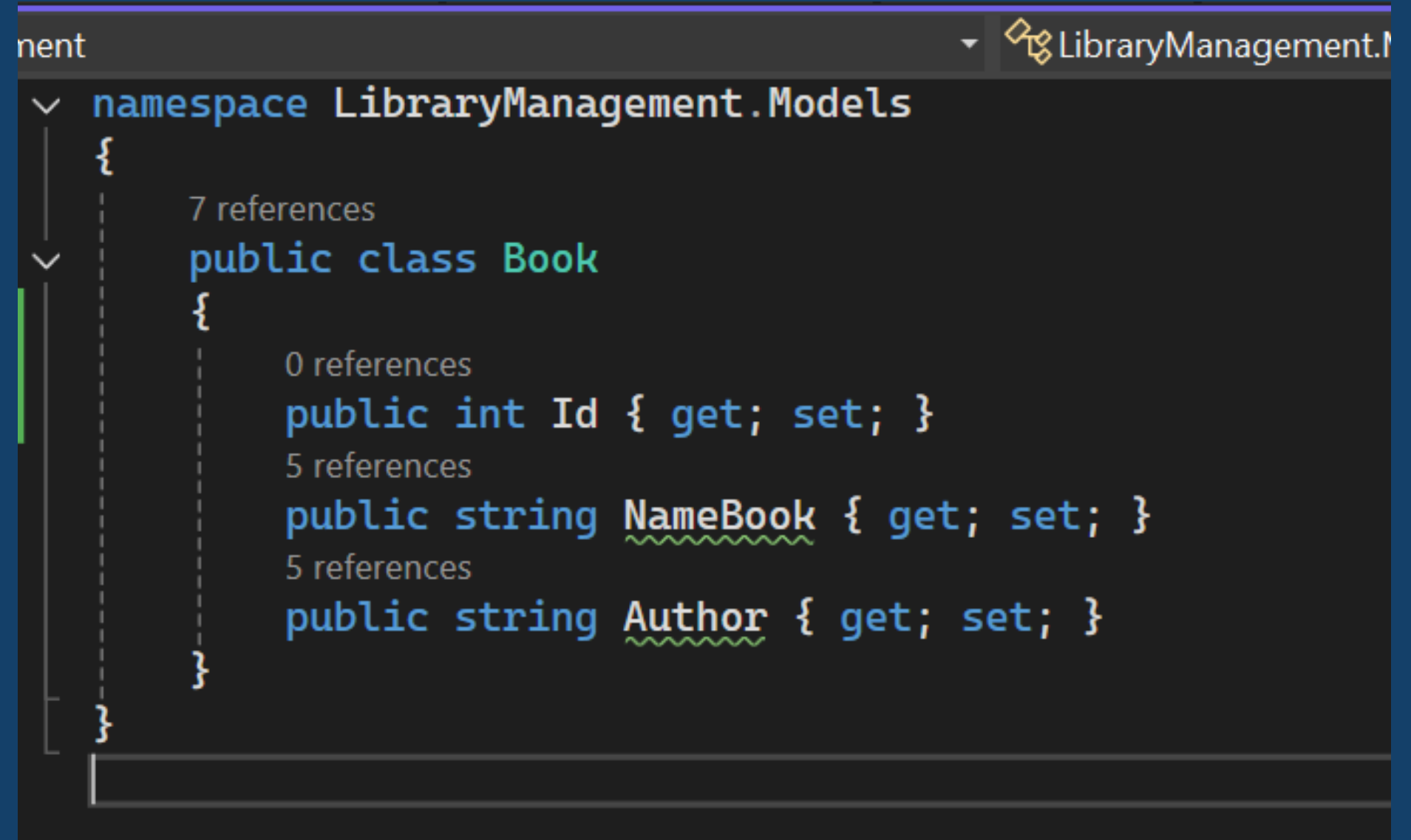


## Model – Add a model to an ASP.NET Core API app

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:

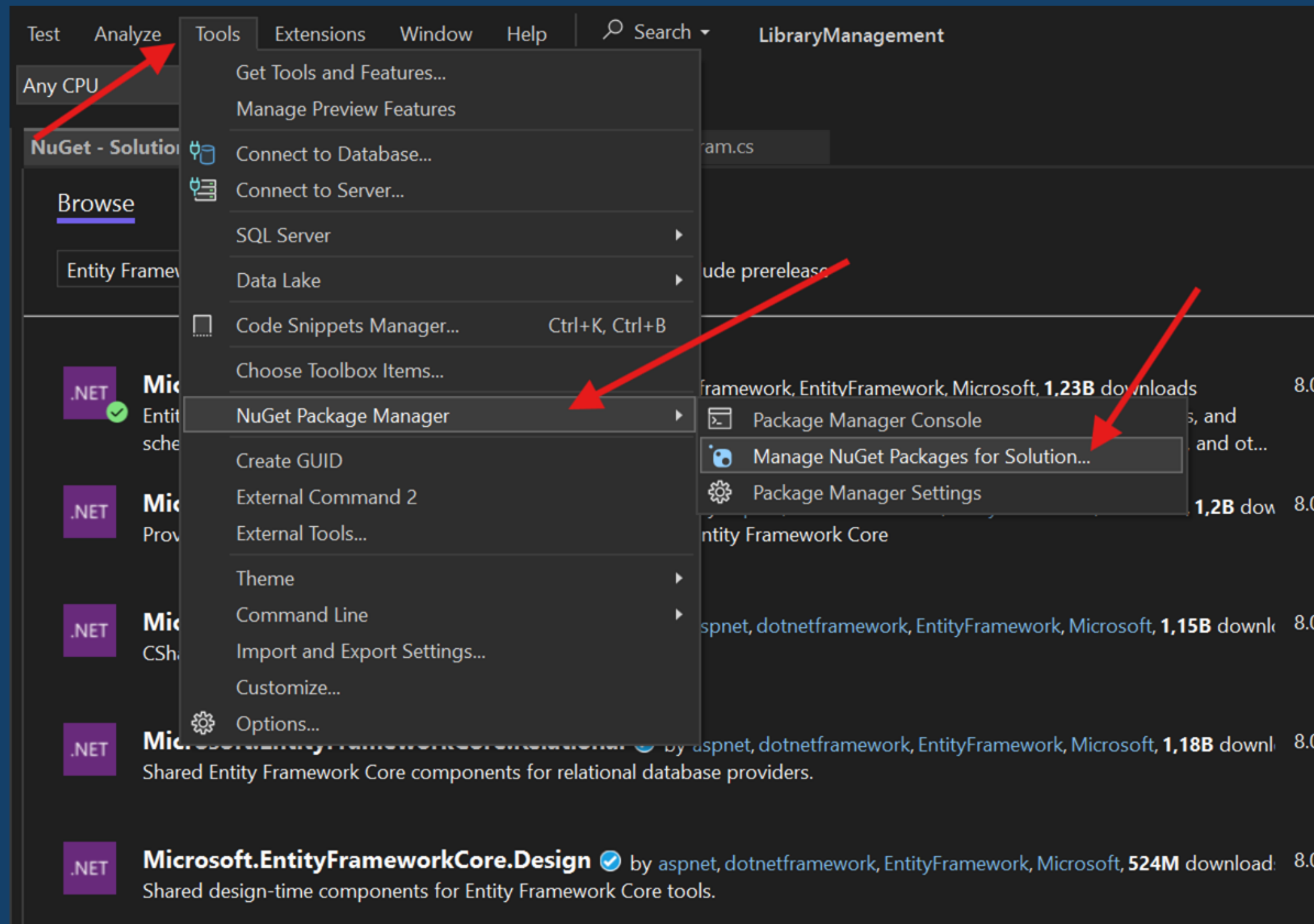
- The user isn't required to enter time information in the date field.
- Only the date is displayed, not time information.



```
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; }
    }
}
```



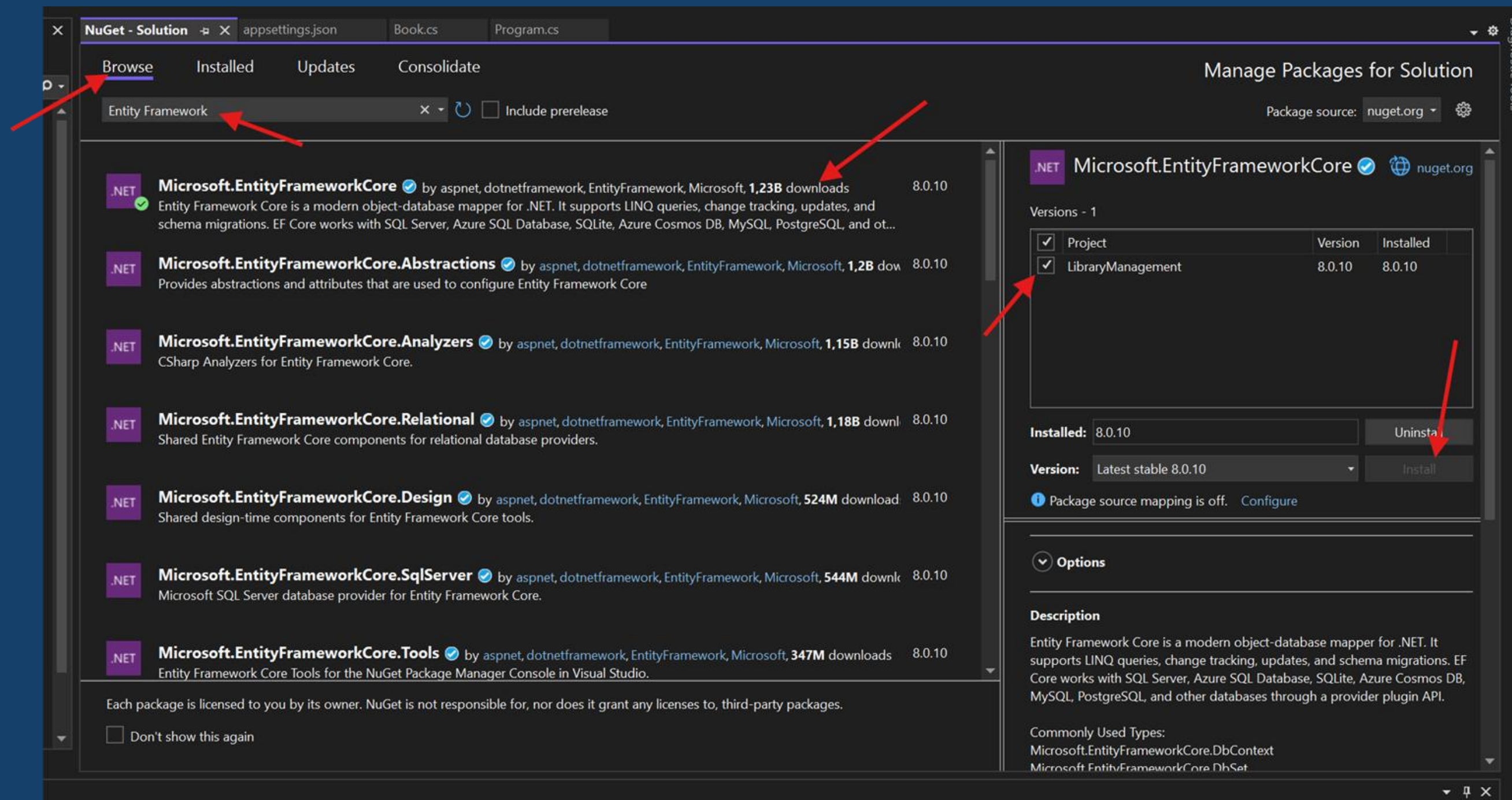
## Model – Entity Framework



## Model – Entity Framework

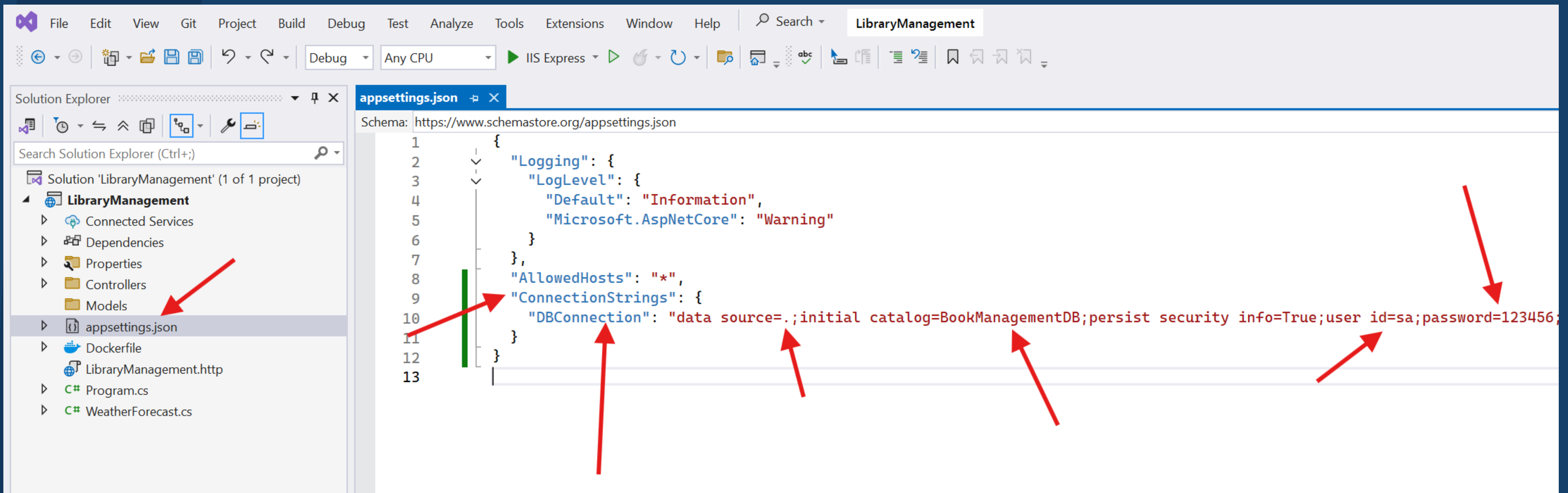
Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore



## Model – appsetting.json

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



The screenshot shows the Visual Studio IDE with the 'LibraryManagement' project open. The Solution Explorer on the left shows the project structure, with 'appsettings.json' selected. The main editor displays the content of 'appsettings.json' with the following schema: <https://www.schemastore.org/appsettings.json>.

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
9   "ConnectionStrings": {
10     "DBConnection": "data source=.;initial catalog=BookManagementDB;persist security info=True;user id=sa;password=123456;MultipleActiveResultSets=True;encrypt=false"
11   }
12 }
13
```

Red arrows highlight the 'appsettings.json' file in the Solution Explorer and the 'DBConnection' string in the code, which matches the example provided in the text above.

## Model – Dependency injection

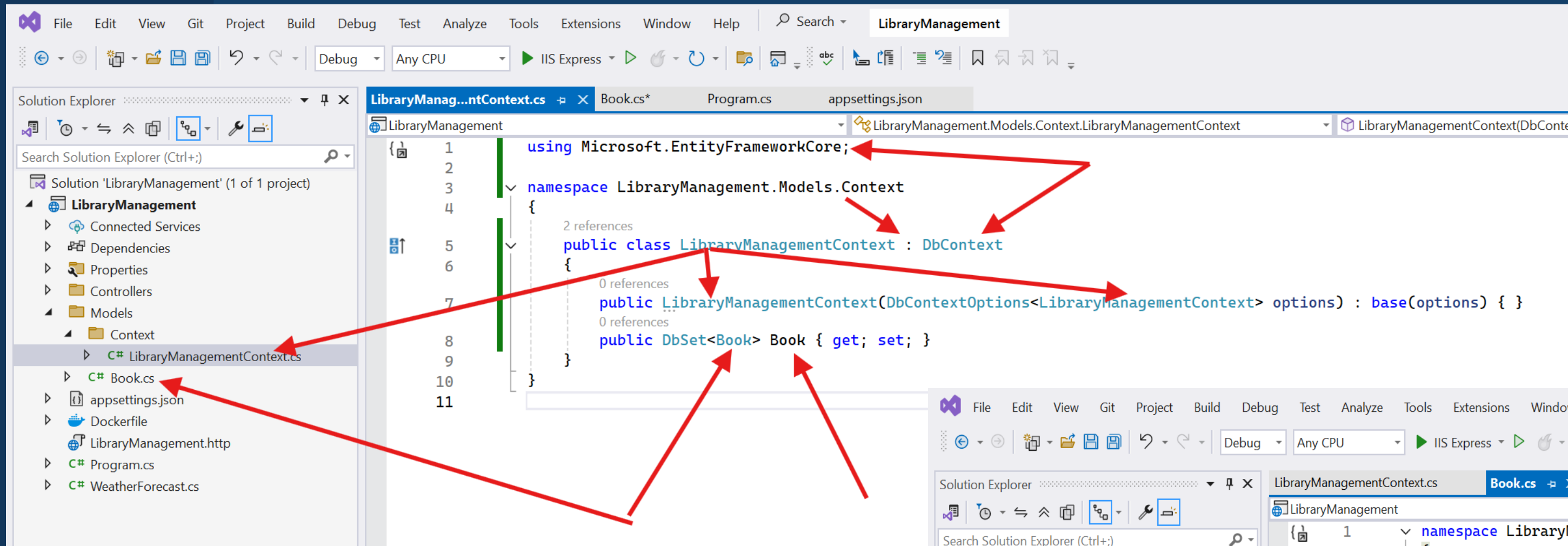
- ASP.NET Core is built with dependency injection (DI). Services, such as the database context, are registered with DI in Program.cs. These services are provided to components that require them via constructor parameters.
- In the Controllers/MoviesController.cs file, the constructor uses Dependency 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:



## Model – Dependency injection

```
1 using LibraryManagement.Models.Context;
2 using Microsoft.EntityFrameworkCore;
3
4 var builder = WebApplication.CreateBuilder(args);
5
6 // Add services to the container.
7 builder.Services.AddControllersWithViews();
8
9 builder.Services.AddDbContext<BookContext>(options =>
10     options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection")));
11
12 var app = builder.Build();
13
14 // Configure the HTTP request pipeline.
15 if (!app.Environment.IsDevelopment())
16 {
```

## Model – Dbcontext



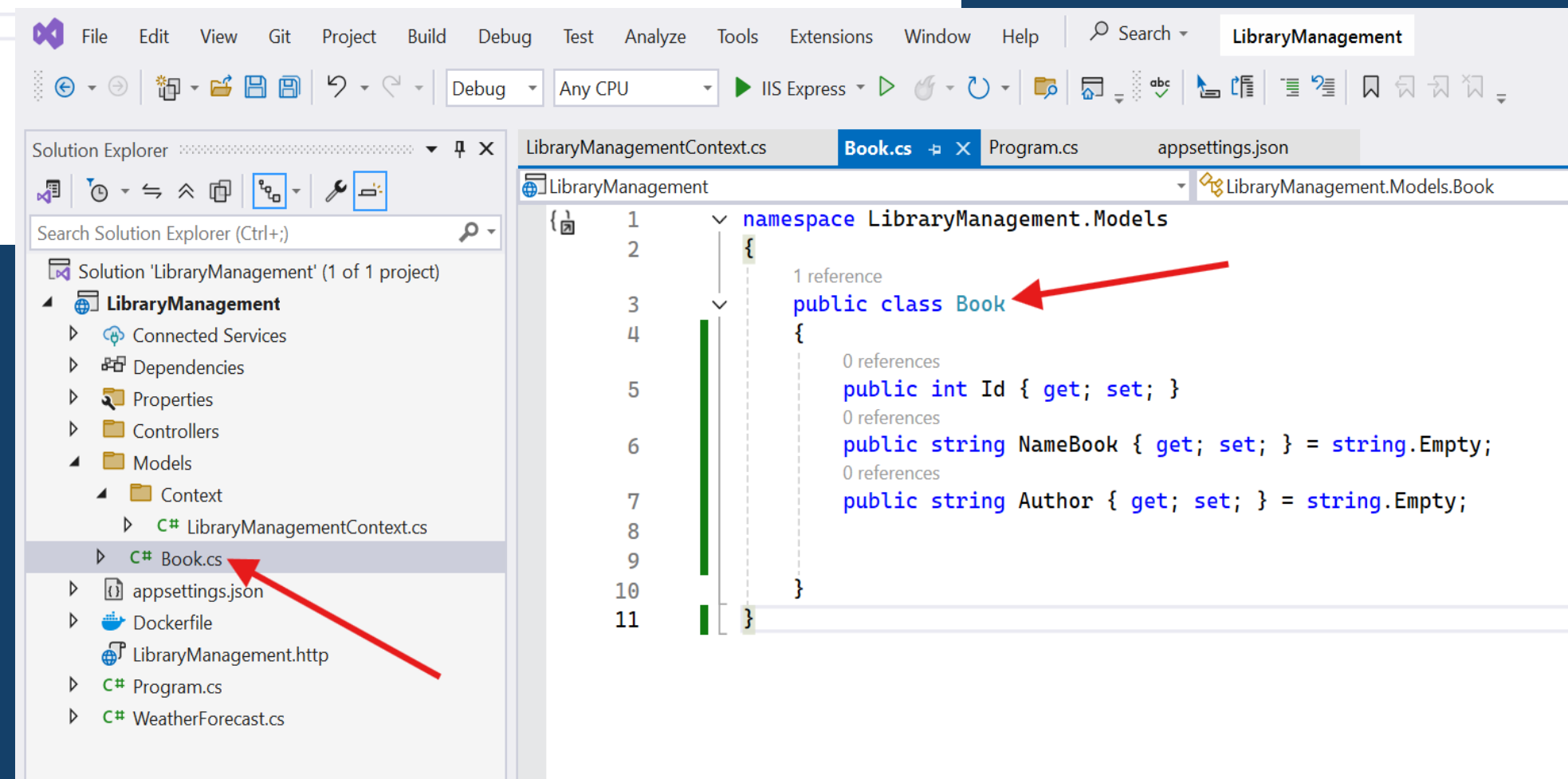
**LibraryManagementContext.cs**

```

1  using Microsoft.EntityFrameworkCore;
2
3  namespace LibraryManagement.Models.Context
4  {
5      2 references
6      public class LibraryManagementContext : DbContext
7      {
8          0 references
9          public LibraryManagementContext(DbContextOptions<LibraryManagementContext> options) : base(options) { }
10         0 references
11         public DbSet<Book> Book { get; set; }
12     }
    
```

**Solution Explorer:**

- LibraryManagement
  - Context
    - C# LibraryManagementContext.cs
  - Models
    - C# Book.cs



**Book.cs**

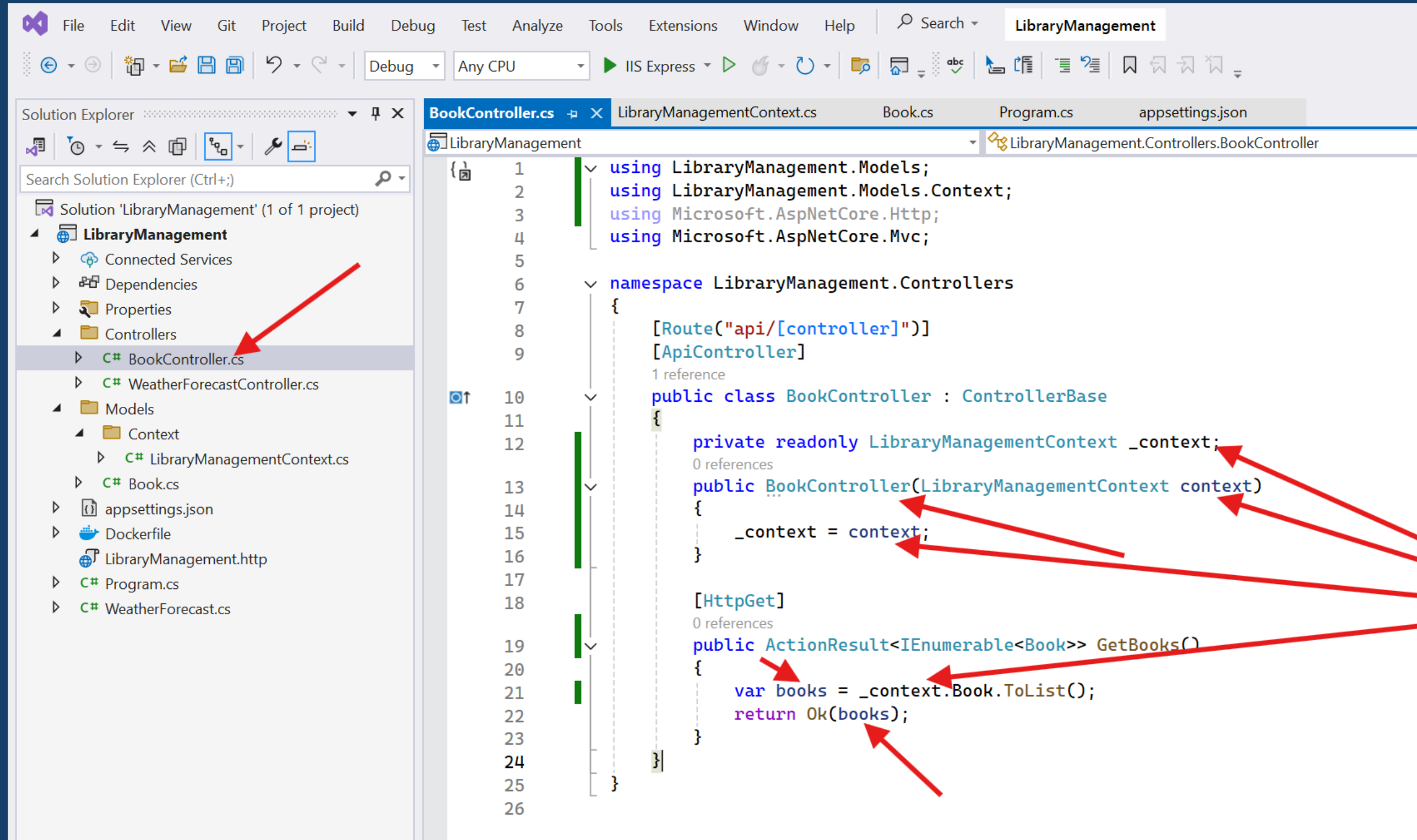
```

1  namespace LibraryManagement.Models
2  {
3      1 reference
4      public class Book
5      {
6          0 references
7          public int Id { get; set; }
8          0 references
9          public string NameBook { get; set; } = string.Empty;
10         0 references
11         public string Author { get; set; } = string.Empty;
12     }
    
```

**Solution Explorer:**

- LibraryManagement
  - Models
    - C# Book.cs

## Model – Dependency injection in the controller



The screenshot shows the Visual Studio IDE with the 'LibraryManagement' project open. The 'Solution Explorer' on the left shows the project structure, with 'BookController.cs' selected under the 'Controllers' folder. The 'Code' window on the right displays the contents of 'BookController.cs'.

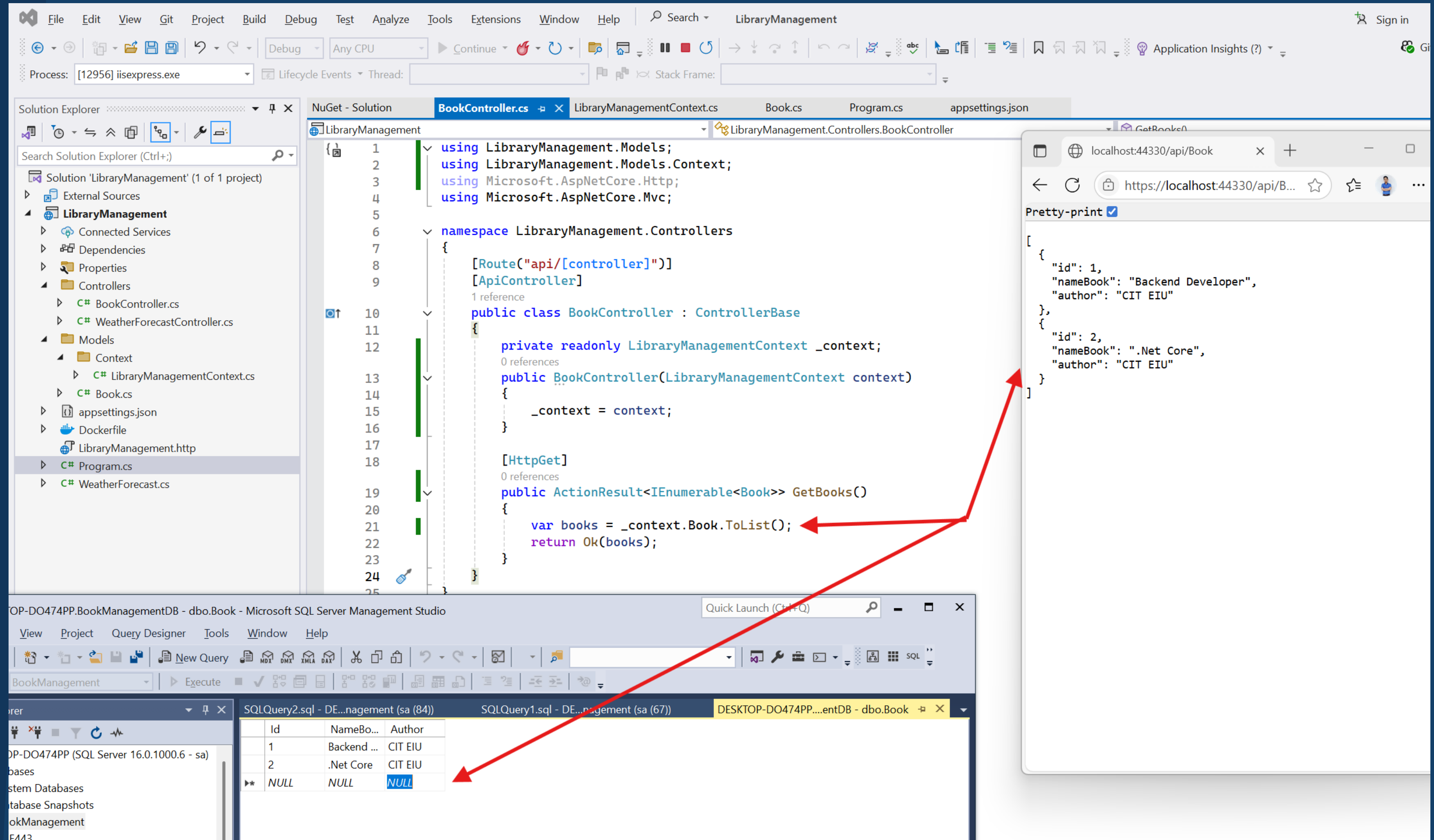
```

1  using LibraryManagement.Models;
2  using LibraryManagement.Models.Context;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.Mvc;
5
6  namespace LibraryManagement.Controllers
7  {
8      [Route("api/[controller]")]
9      [ApiController]
10     1 reference
11     public class BookController : ControllerBase
12     {
13         private readonly LibraryManagementContext _context;
14         0 references
15         public BookController(LibraryManagementContext context)
16         {
17             _context = context;
18         }
19
20         [HttpGet]
21         0 references
22         public ActionResult<IEnumerable<Book>> GetBooks()
23         {
24             var books = _context.Book.ToList();
25             return Ok(books);
26         }
27     }
    
```

Red arrows indicate the flow of dependency injection:

- An arrow points from the 'BookController.cs' file in the 'Solution Explorer' to the file in the 'Code' window.
- An arrow points from the 'context' parameter in the constructor `public BookController(LibraryManagementContext context)` to the `_context` property.
- An arrow points from the `_context` property to its assignment `_context = context;`.
- An arrow points from the `_context` property to its usage in the `GetBooks()` method: `var books = _context.Book.ToList();`.





The image displays the development environment for an ASP.NET Core API, showing the code for the `BookController` and the data it returns.

**Visual Studio - BookController.cs**

```

1  using LibraryManagement.Models;
2  using LibraryManagement.Models.Context;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.Mvc;
5
6  namespace LibraryManagement.Controllers
7  {
8      [Route("api/[controller]")]
9      [ApiController]
10     public class BookController : ControllerBase
11     {
12         private readonly LibraryManagementContext _context;
13
14         public BookController(LibraryManagementContext context)
15         {
16             _context = context;
17         }
18
19         [HttpGet]
20         public IActionResult GetBooks()
21         {
22             var books = _context.Book.ToList();
23             return Ok(books);
24         }
25     }

```

**Browser - localhost:44330/api/Book**

The browser displays the JSON response from the API endpoint:

```

[
  {
    "id": 1,
    "nameBook": "Backend Developer",
    "author": "CIT EIU"
  },
  {
    "id": 2,
    "nameBook": ".Net Core",
    "author": "CIT EIU"
  }
]

```

**SQL Server Management Studio - BookManagementDB - dbo.Book**

The SQL Server Management Studio shows the data in the `Book` table:

Id	NameBo...	Author
1	Backend ...	CIT EIU
2	.Net Core	CIT EIU
NULL	NULL	NULL

Red arrows indicate the flow of data from the database table to the `GetBooks()` method in the `BookController`, which then returns the data as a JSON response to the browser.

*Start your future at EIU*

---

**Thank You**