Building a Web API in ASP.NET

Workshop by Johnny, Yassine & Koen

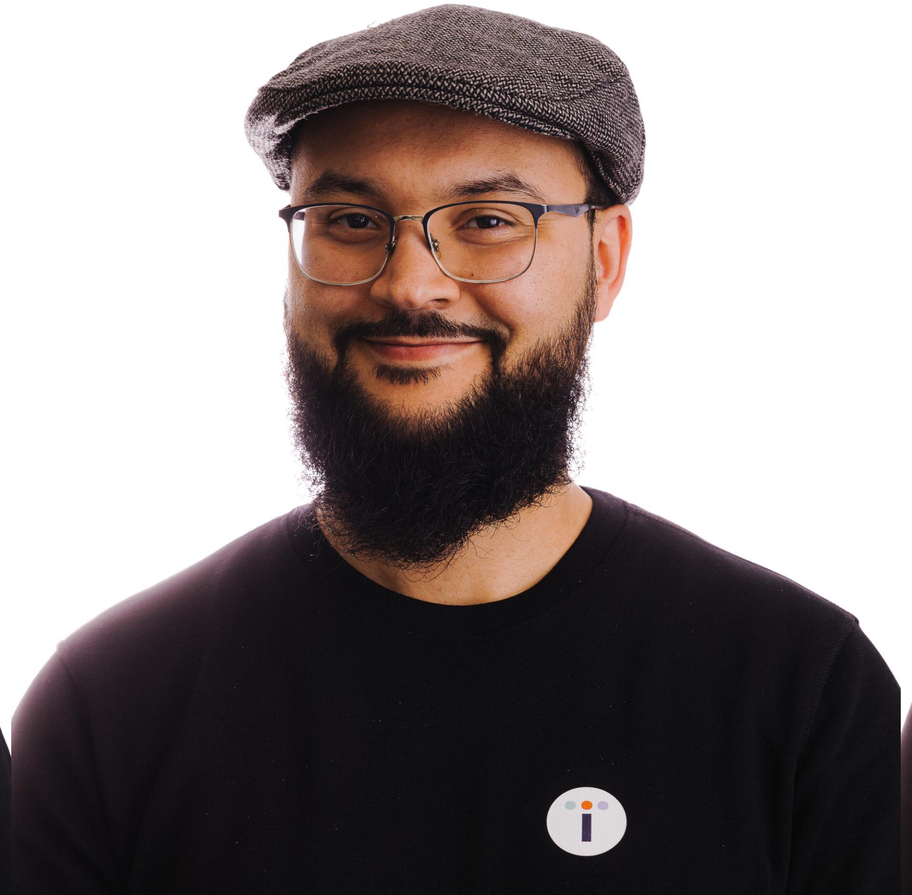
@involved_it

Veldkant 35C, 2550 Kontich

Nice to meet you

Johnny Hooyberghs

Yassine Ikrou



Koen Seeuws

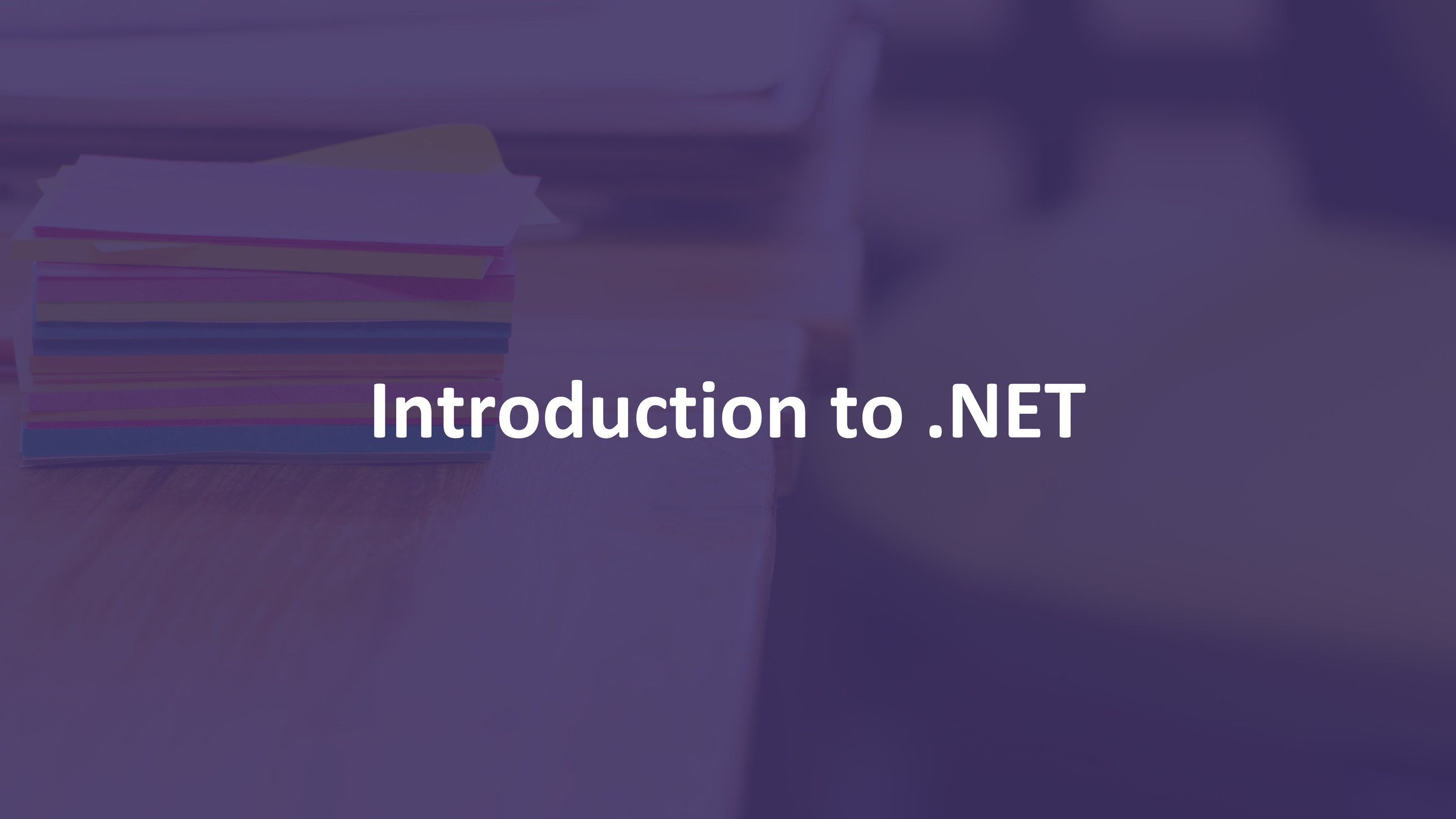
Nice to meet you too

Pieter Vandewaetere

What’s on today

In this session

- Introduction to .NET
- What are we building today?
- What are we building in this session?
- Exploring ASP.NET Web API
- Project setup
- Coding
- Q&A

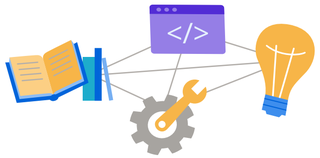


Introduction to .NET

What is .NET?

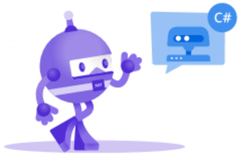
A software development platform

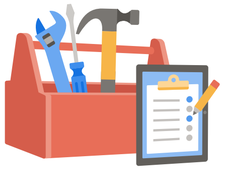
Cross platform and Open Source



Standard set of libraries

Modern object-oriented language





Modern and flexible tooling

What can you build using .NET?

Web

Build web apps and services for Windows, Linux, macOS, and Docker.

Mobile

Use a single codebase to build native mobile apps for iOS, Android, and more.

Desktop

Create native apps for Windows and macOS or build apps that run anywhere with web technologies.

Artificial Intelligence and ML

Build smart apps with C#, OpenAI, and Azure.

Cloud

Consume existing cloud services, or create and deploy your own.

Microservices

Create independently deployable microservices that run on Docker containers.

Game development

Develop 2D and 3D games for the most popular desktops, phones, and consoles.

Internet of Things

Make IoT apps, with native support for the Raspberry Pi and other single-board computers.

And what about its history?

2002

.NET Framework

2004

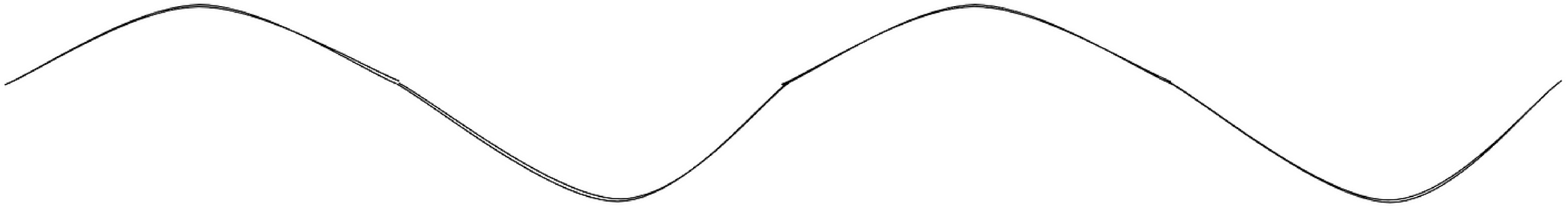
The Mono Project

2009

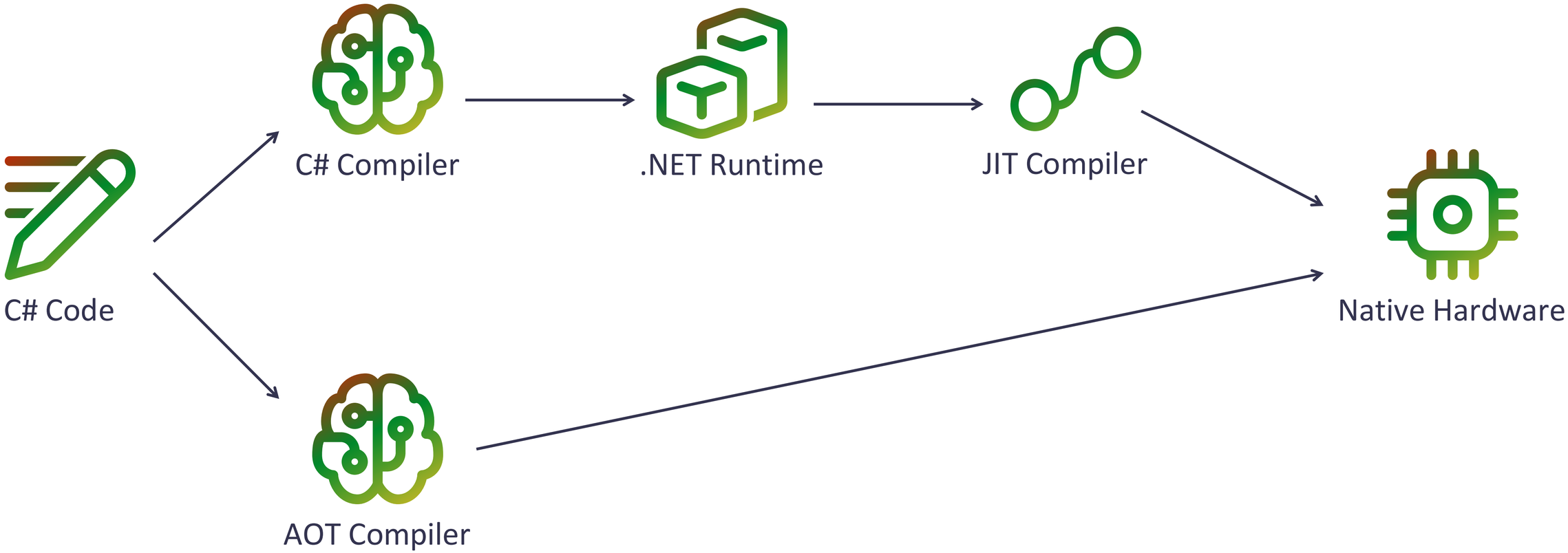
Xamarin

2016

.NET Core

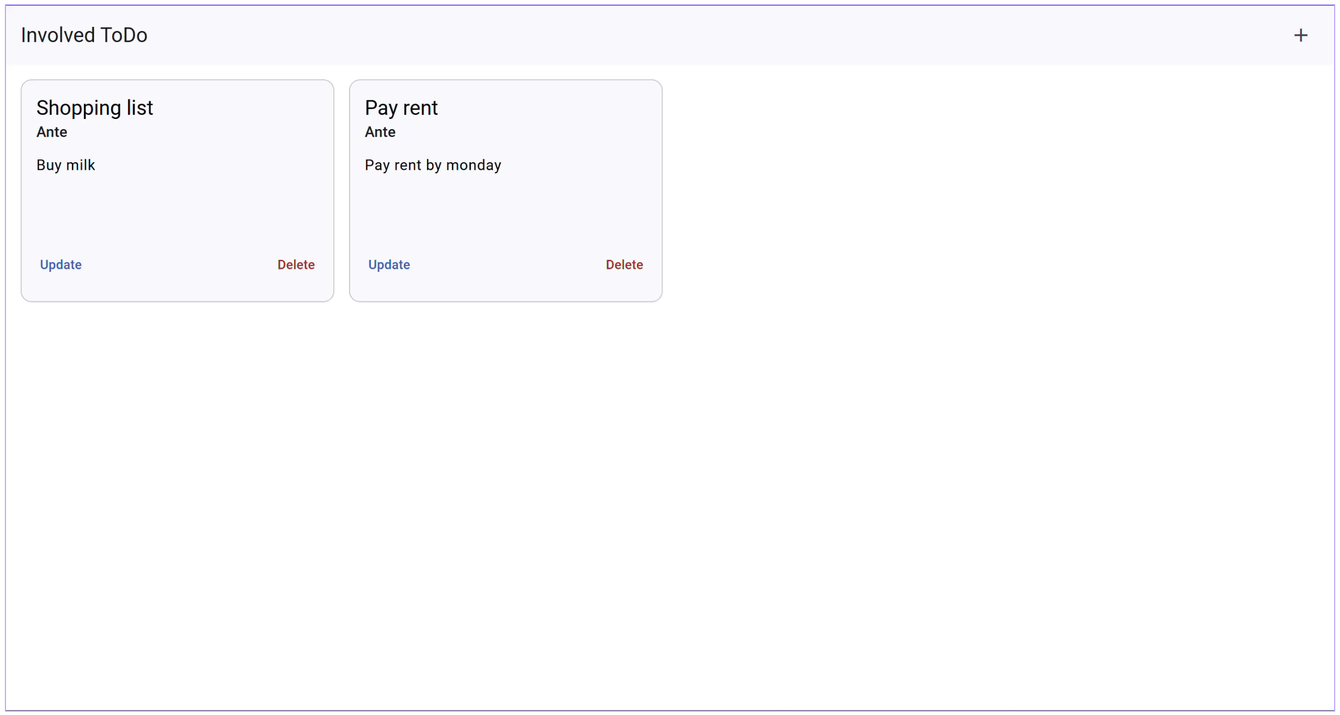


How does .NET run on your system?



What are we building today?

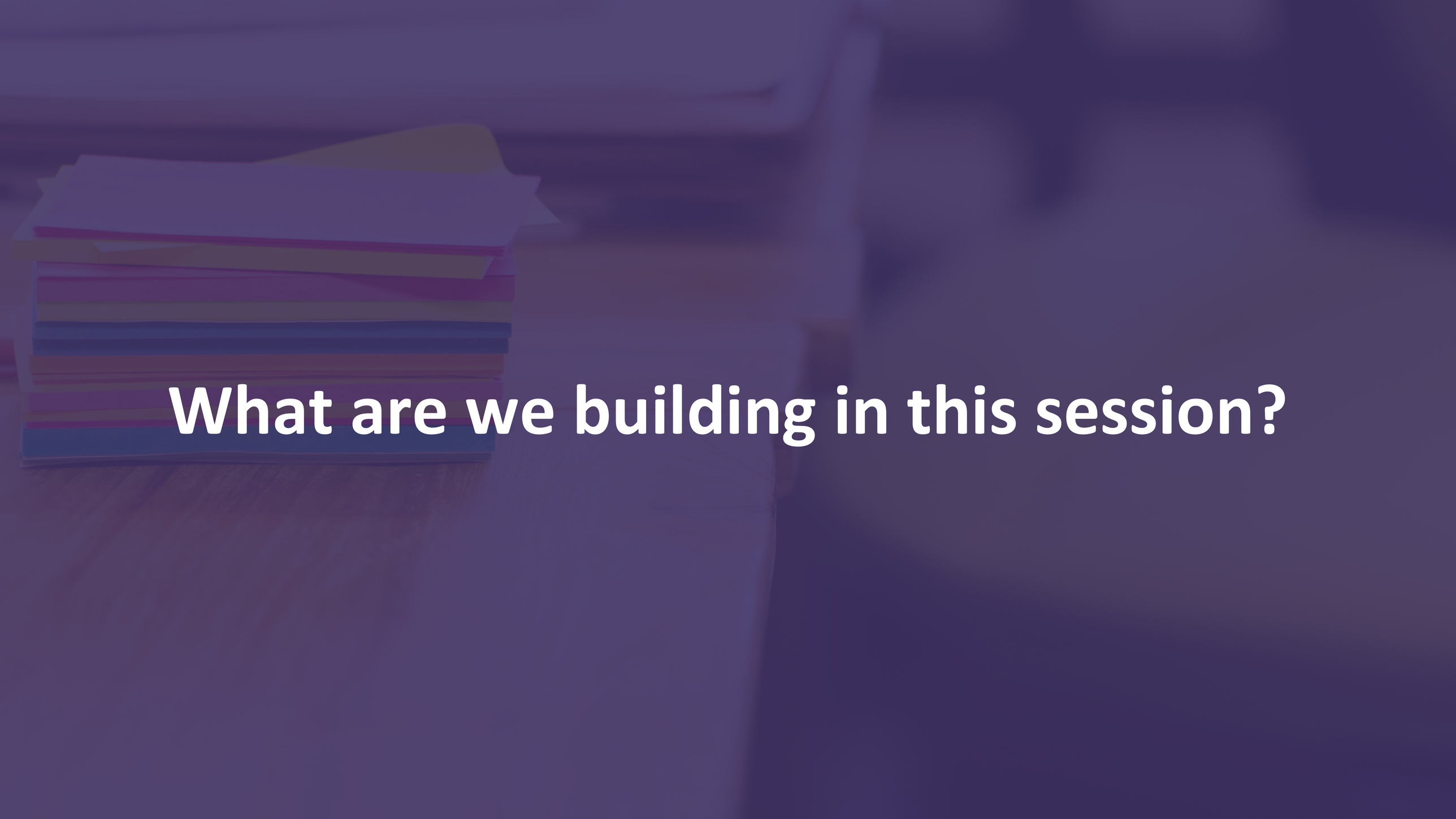
A lightweight TODO app



Functionalities:

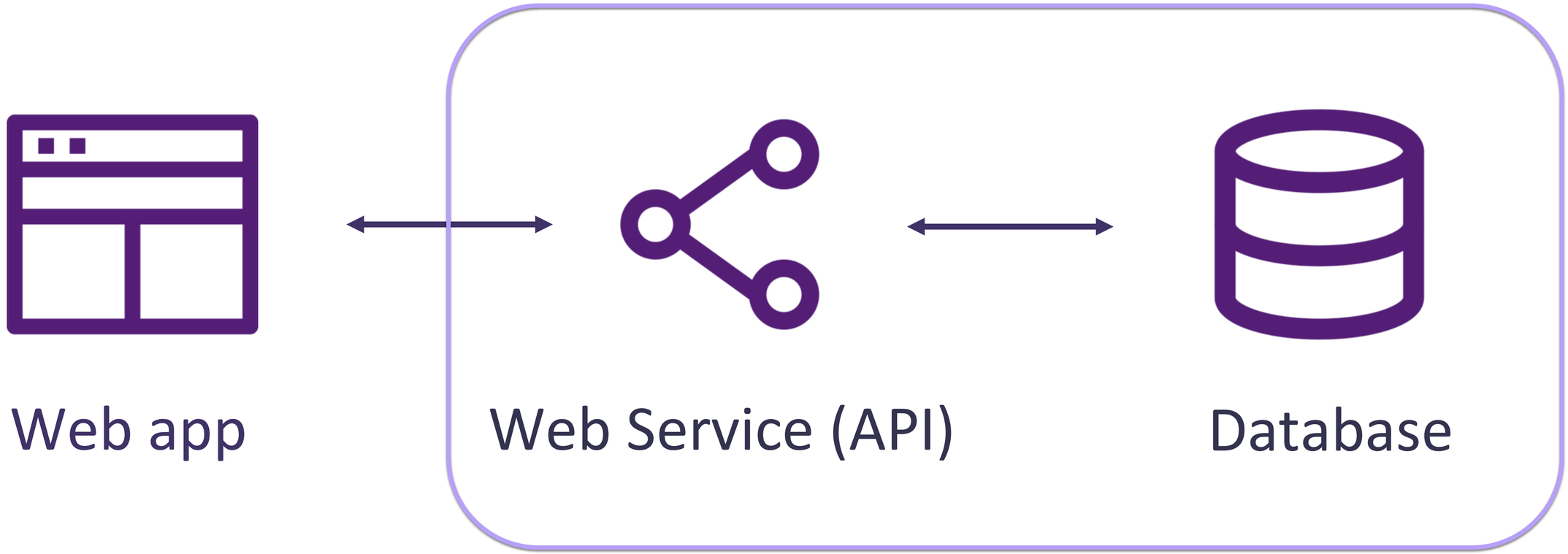
- Add TODO
- Retrieve all TODO's
- Edit TODO
- Delete TODO

Overall Architecture



What are we building in this session?

Just the backend



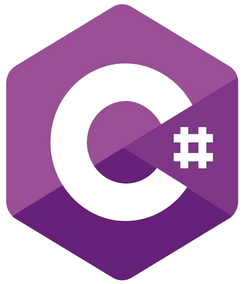
Tooling & technology

IDE

Web Service (API)

Microsoft

ASP.NET



Database

Exploring ASP.NET Web API

What is ASP.NET Web API?

- A framework by Microsoft
- Build HTTP-based services
- Toolkit that allows building applications for communicating over the web

Key concepts

- **API** (Application Programming Interface)
  - Think of it as a messenger
- **HTTP** (Hyper Text Transform Protocol)
  - Language of the web
  - Methods like Get, Post, Put, Delete, …
- **REST** (Representational State Transfer)
  - Industry standard for building APIs

How does it work?

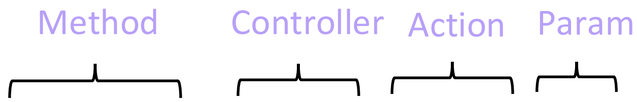
- **Controller and actions**
  - Controller collection of actions
  - Action handles HTTP requests
- **Routes**
  - Returns response with or without data
- **Response**
  - JSON (JavaScript Object Notation)
  - Lightweight and human-readable format
  - HTTP Status code (200, 400, ...)

```
[Route("api/product")]
[ApiController]
public class ProductController
{
        public IActionResult GetAll() ...
}
```

```
api/product/getAll
```

```
[
        { "id": 1, "name": "Laptop", "price": 999 },
        { "id": 2, "name": "Smartphone", "price": 499 }
]
```

Endpoints we'll be building



POST	/Todo/Add
GET	/Todo/GetAll
GET	/Todo/Get/{id}
GET	/Todo/Search
PUT	/Todo/Update
DELETE	/Todo/Delete/{id}

Entity Framework Core

Project setup

Setting up a .NET solution with a .NET Web API project

Creating the solution

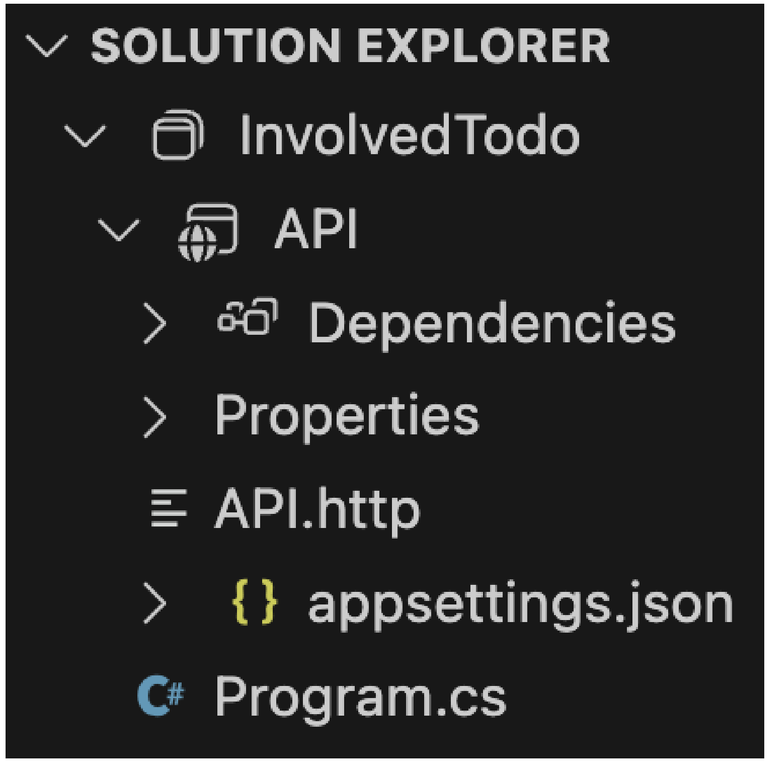
1. Create a new folder in your file system
2. Open the folder in VS Code
3. Open a terminal
4. Execute:

```
> dotnet new sln --name InvolvedTodo
```

5. Right click the new .sln file in VS code and click ‘Open as solution’

Creating the API project

1. Open the solution explorer
2. Right click the solution
3. Click ‘New Project’
4. Choose 'ASP.NET Core Web API'
5. Enter a name for the project, we will use 'API'
6. Leave all the other options as default



Adjust Program.cs

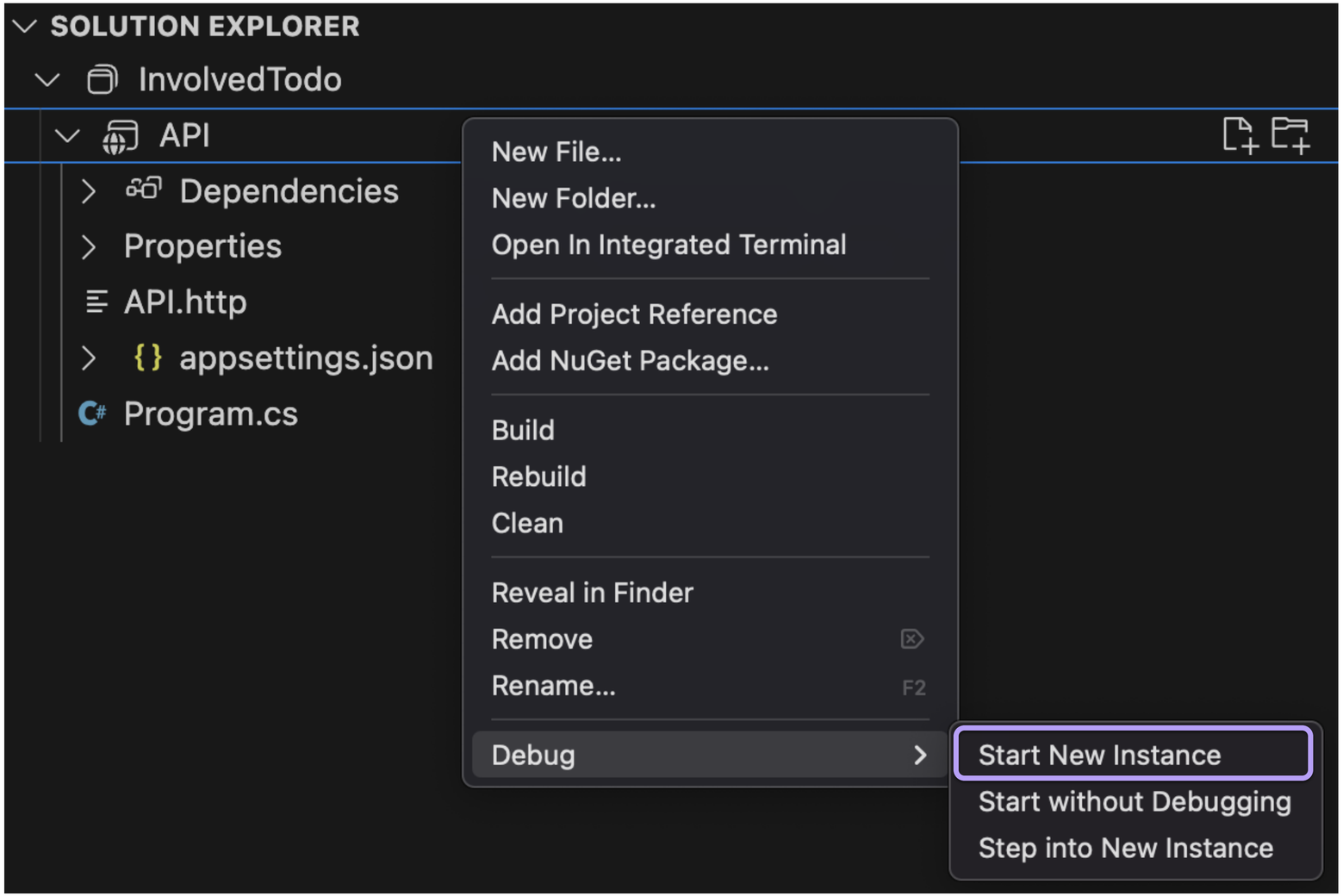
Add controllers to service container

Delete Weather boilerplate code

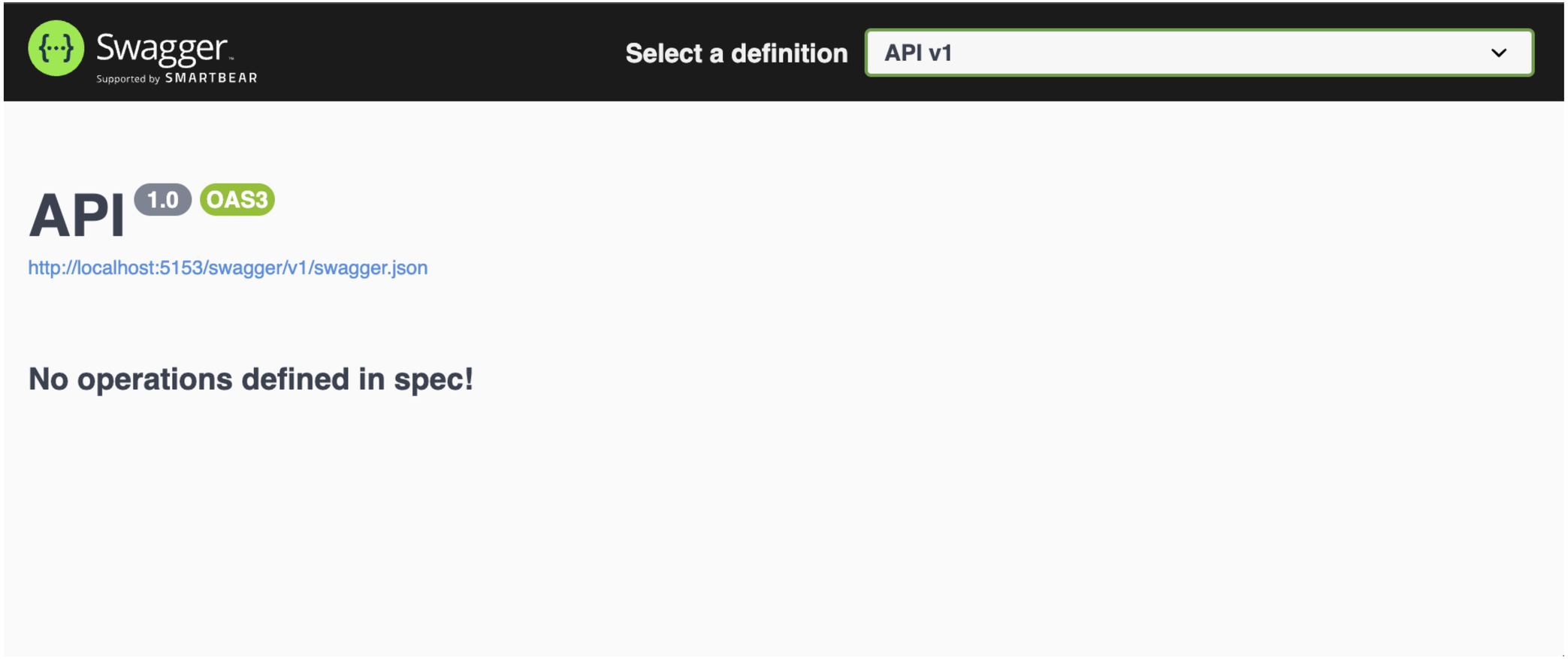
Map controllers

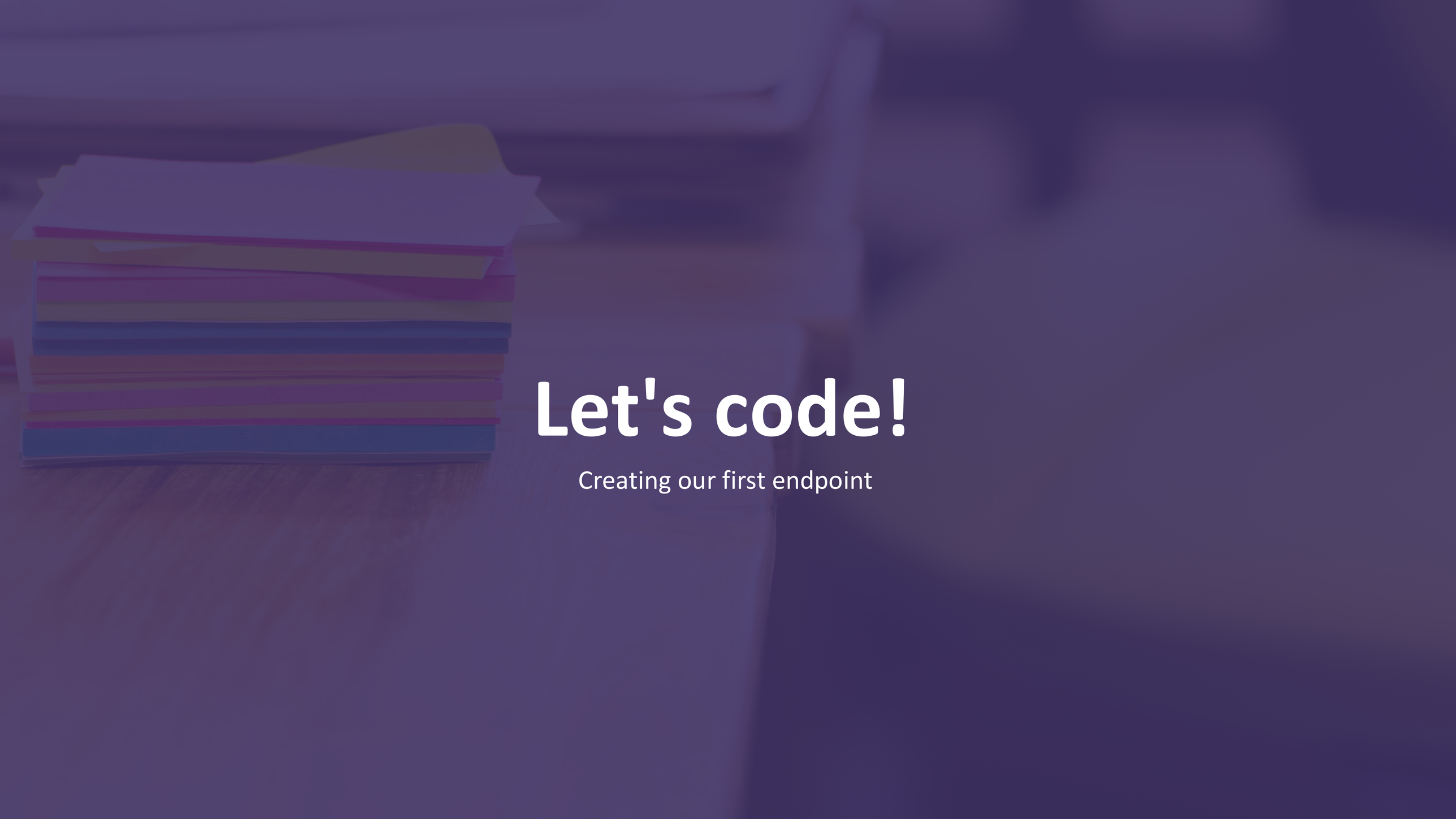
```
1   var builder = WebApplication.CreateBuilder(args);
2
3   builder.Services.AddControllers();
4
5   // Add services to the container.
6   // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
7   builder.Services.AddEndpointsApiExplorer();
8   builder.Services.AddSwaggerGen();
9
10  var app = builder.Build();
11
12  // Configure the HTTP request pipeline.
13  if (app.Environment.IsDevelopment())
14  {
15      app.UseSwagger();
16      app.UseSwaggerUI();
17  }
18
19  app.UseHttpsRedirection();
20
21  app.MapControllers();
22
23  app.Run();
```

Spin up the Web API



Congrats! Your first “empty” .NET Web API





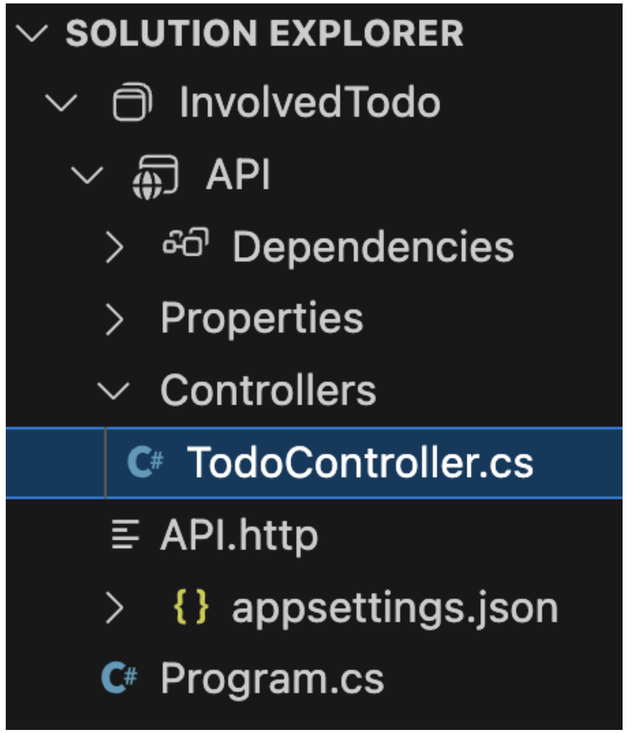
Let's code!

Creating our first endpoint

Creating a Controller

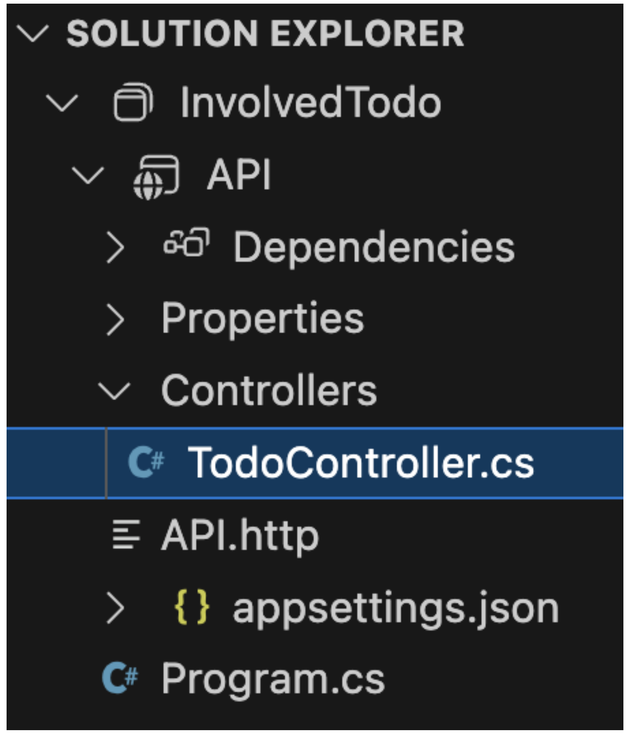
1. Add new folder ‘Controllers’
2. Right click the folder and add new file

   and name it ‘TodoController.cs’

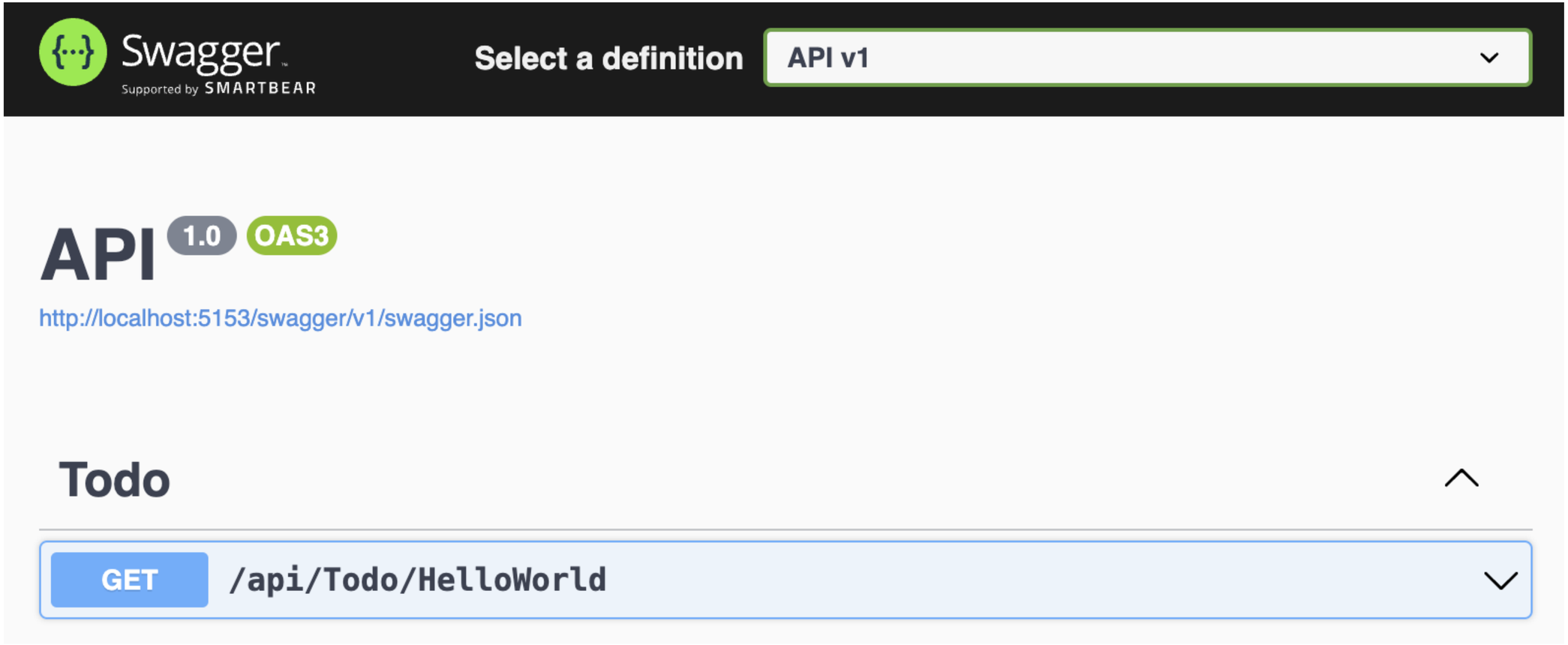


Add action to Controller

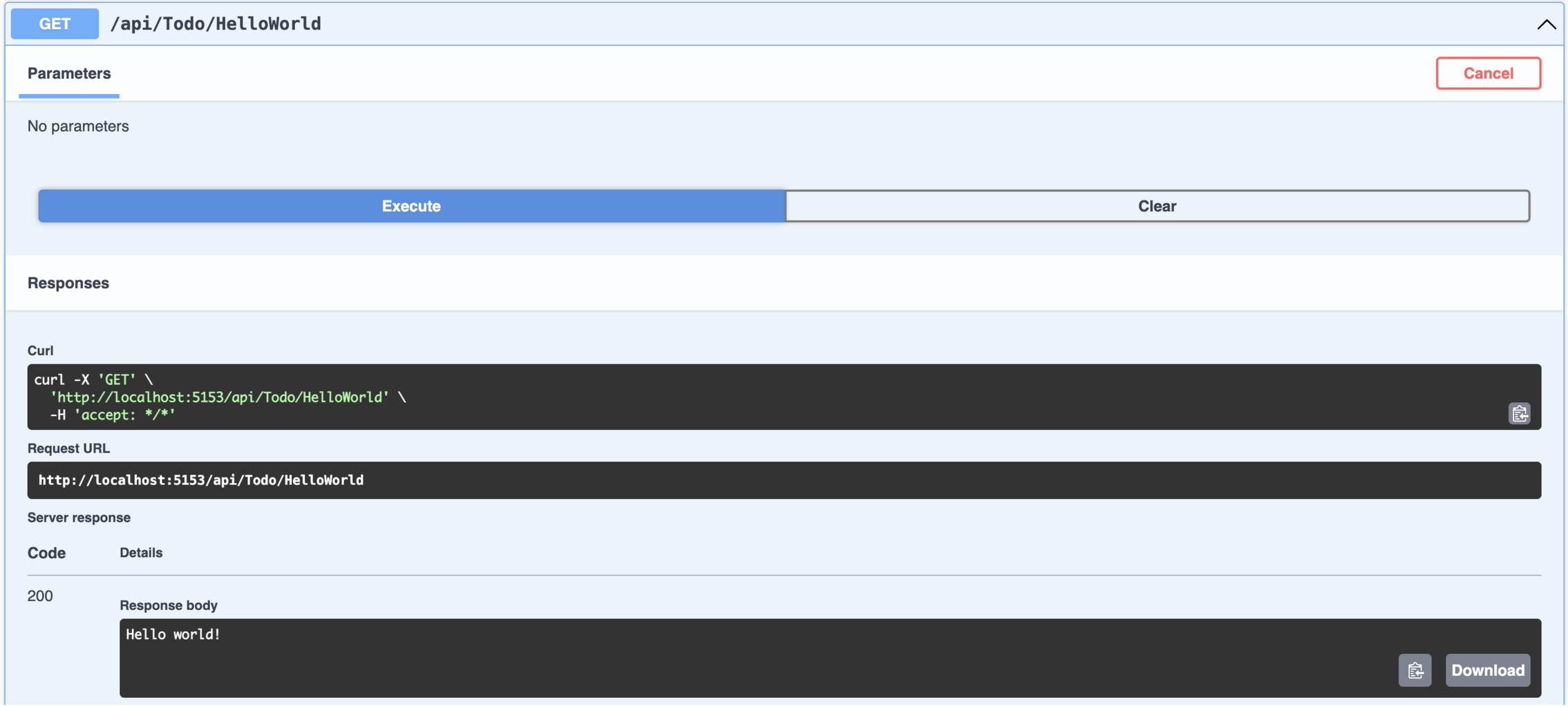
```
1  using Microsoft.AspNetCore.Http;
2  using Microsoft.AspNetCore.Mvc;
3
4  namespace API.Controllers;
5
6  [Route("api/[controller]")]
7  [ApiController]
   0 references
8  public class TodoController : ControllerBase
9  {
10     [HttpGet("[action]")]
       0 references
11     public IActionResult HelloWorld() {
12         return Ok("Hello world!");
13     }
14 }
```



Congrats! Your first Endpoint



Testing our first endpoint



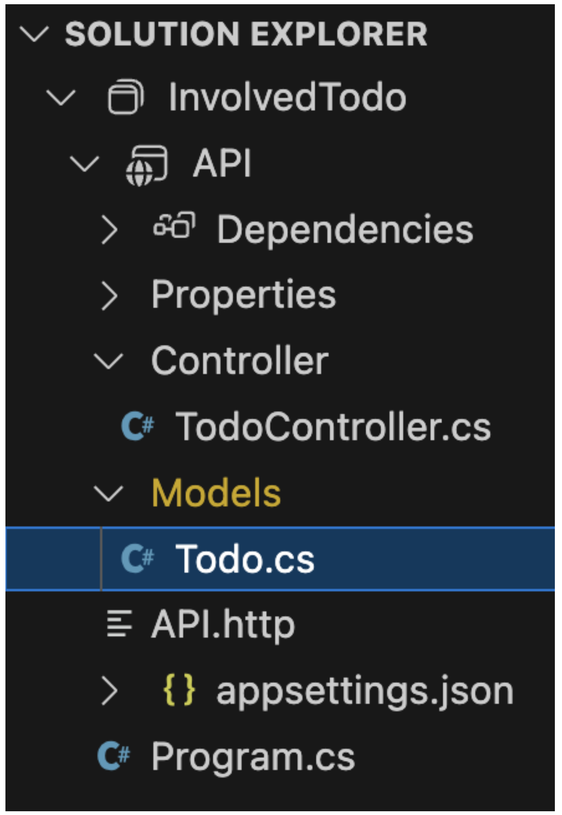
Retrieving data

Returning data from an endpoint in JSON format

Creating a Model

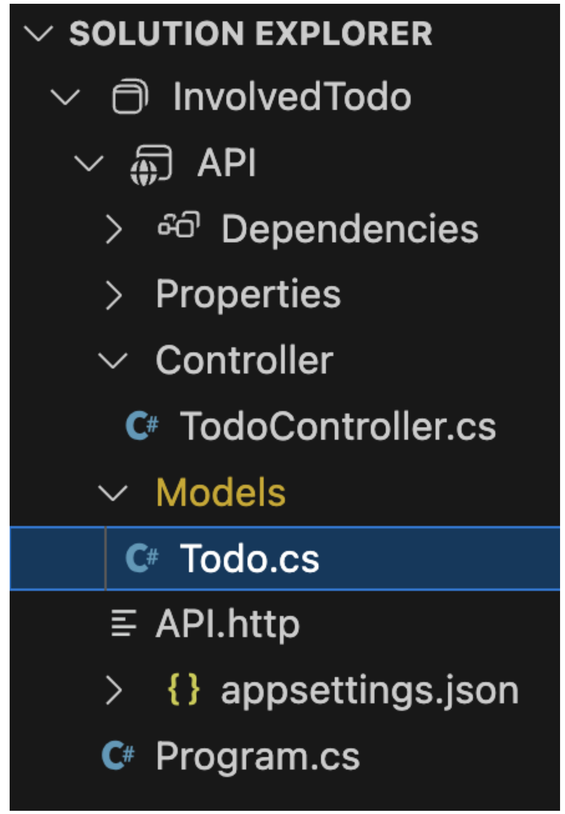
1. Add new folder ‘Models’
2. Right click the folder and add new file

   and name it ‘Todo.cs’



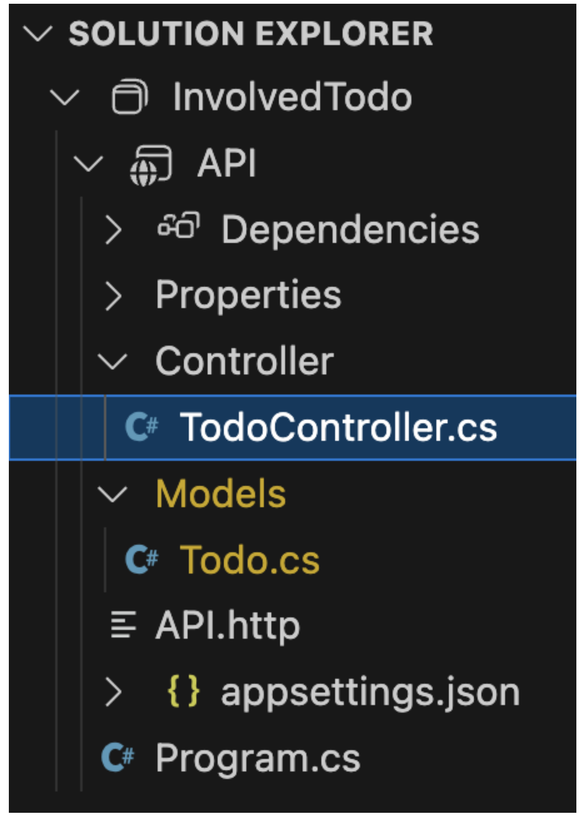
Defining a Model

```
1   namespace API.Models;
2
    0 references
3   public class Todo
4   {
        0 references
5       public int Id { get; set; }
6
        0 references
7       public string Title { get; set; }
8
        0 references
9       public string Assignee { get; set; }
10
        0 references
11      public string? Description { get; set; }
12  }
```

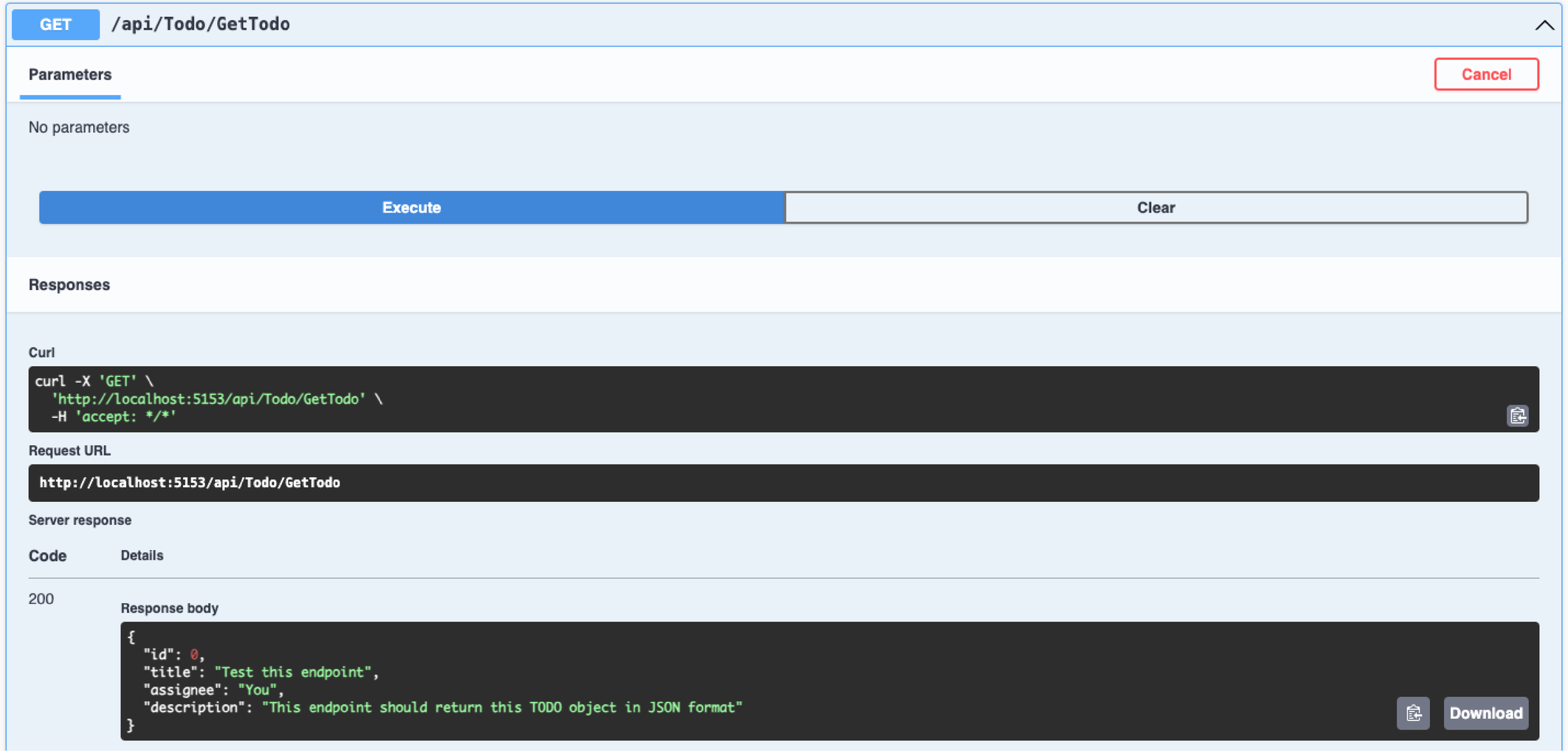


Returning data

```
using API.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace API.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    0 references
    public class TodoController : ControllerBase
    {
        [HttpGet("[action]")]
        0 references
        public IActionResult GetTodo(){
            var todo = new Todo() {
                Title = "Test this endpoint",
                Assignee = "You",
                Description = "This endpoint should return this TODO object in JSON format"
            };

            return Ok(todo);
        }
    }
}
```



Testing the data endpoint



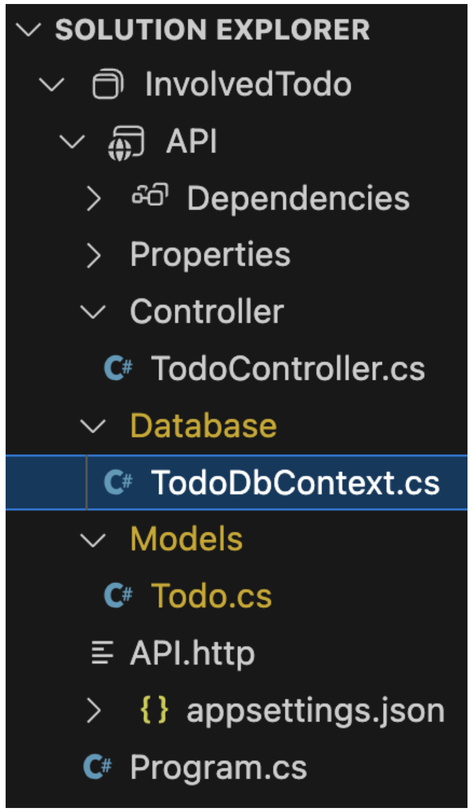
Using a Database

Adding NuGet package(s)

1. Right click the 'API' project
2. Click 'Add NuGet Package'
3. Search for 'Microsoft.EntityFrameworkCore.Sqlite'
4. Select and install version 8.x.x

Defining a DbContext

```
1   using API.Models;
2   using Microsoft.EntityFrameworkCore;
3
4   namespace API.Database;
5
        1 reference
6   public class TodoDbContext(DbContextOptions<TodoDbContext> options) : DbContext(options)
7   {
            0 references
8       public DbSet<Todo> Todos { get; set; }
9   }
```



Adjust Program.cs

Add Sqlite DbContext
To service container

```
using API.Database;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSqlite<TodoDbContext>("Data Source=TodoDatabase.db");

builder.Services.AddControllers();

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();
```

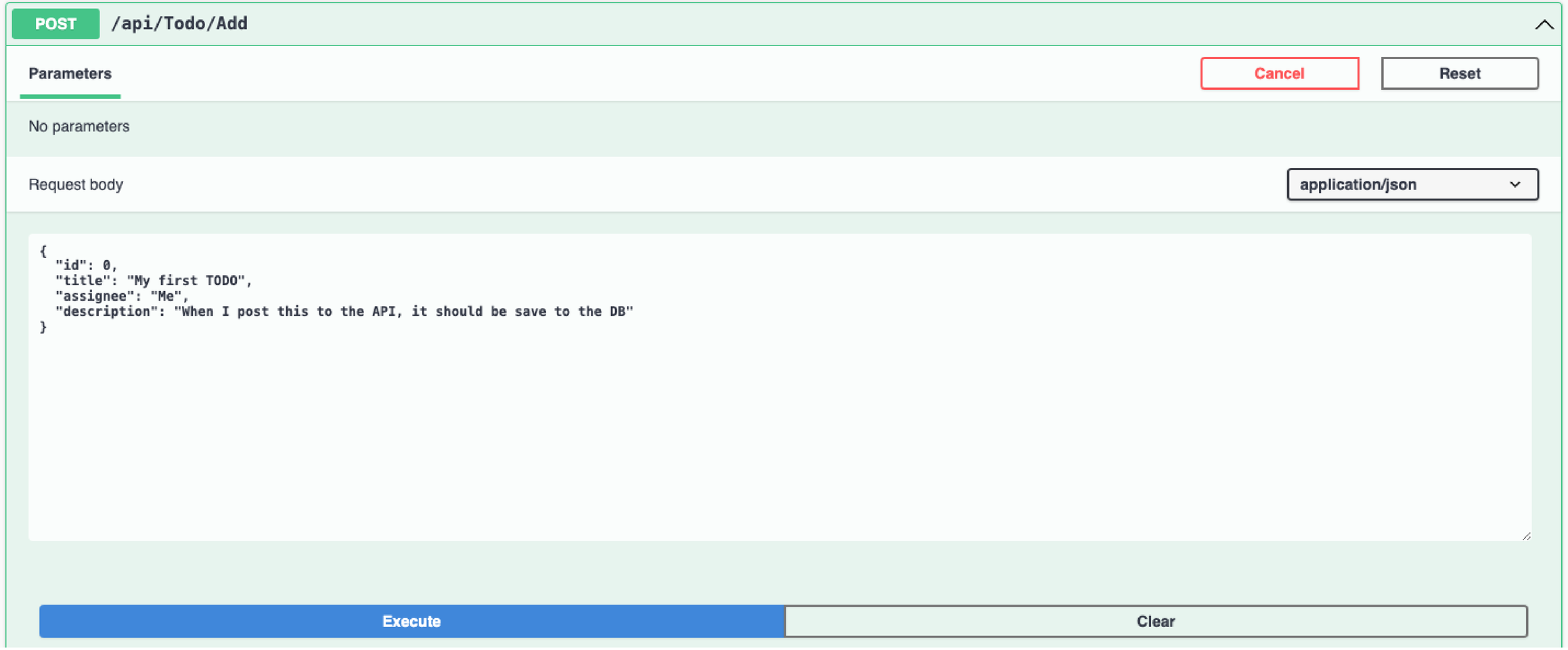
Adding our first TODO

Inject DbContext into Controller

Add todo to DB and Save changes

```
1   using API.Database;
2   using API.Models;
3   using Microsoft.AspNetCore.Http;
4   using Microsoft.AspNetCore.Mvc;
5
6   namespace API.Controller
7   {
8       [Route("api/[controller]")]
9       [ApiController]
            1 reference
10      public class TodoController : ControllerBase
11      {
                4 references
12          private readonly TodoDbContext _todoDbContext;
13
                0 references
14          public TodoController(TodoDbContext todoDbContext) {
15              _todoDbContext = todoDbContext;
16              _todoDbContext.Database.EnsureCreated();
17          }
18
19          [HttpPost("[action]")]
                0 references
20          public IActionResult Add([FromBody] Todo todo)
21          {
22              _todoDbContext.Todos.Add(todo);
23              _todoDbContext.SaveChanges();
24              return Ok(todo);
25          }
26      }
27  }
```

Testing adding a new TODO



Testing adding a new TODO

Server response

Code	Details
200	Response body

```
{
  "id": 1,
  "title": "My first TODO",
  "assignee": "Me",
  "description": "When I post this to the API, it should be save to the DB"
}
```

Download

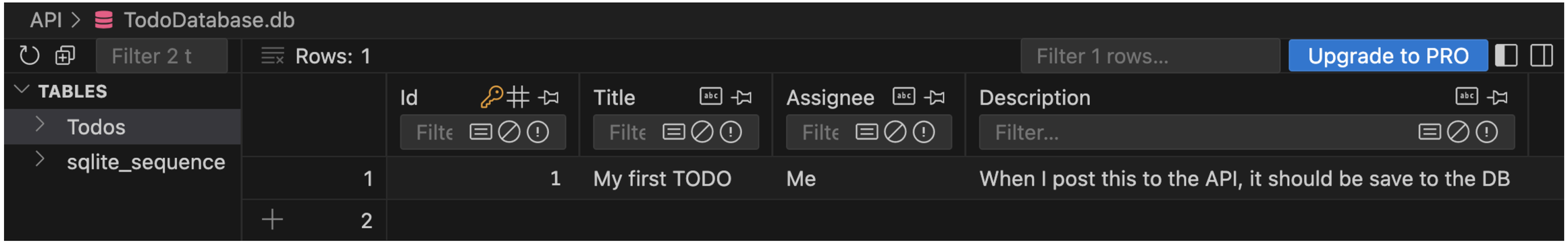
Response headers

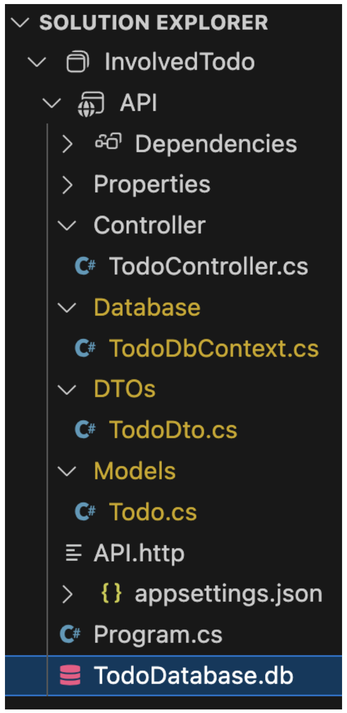
```
content-type: application/json; charset=utf-8
date: Tue,26 Nov 2024 23:25:27 GMT
server: Kestrel
transfer-encoding: chunked
```

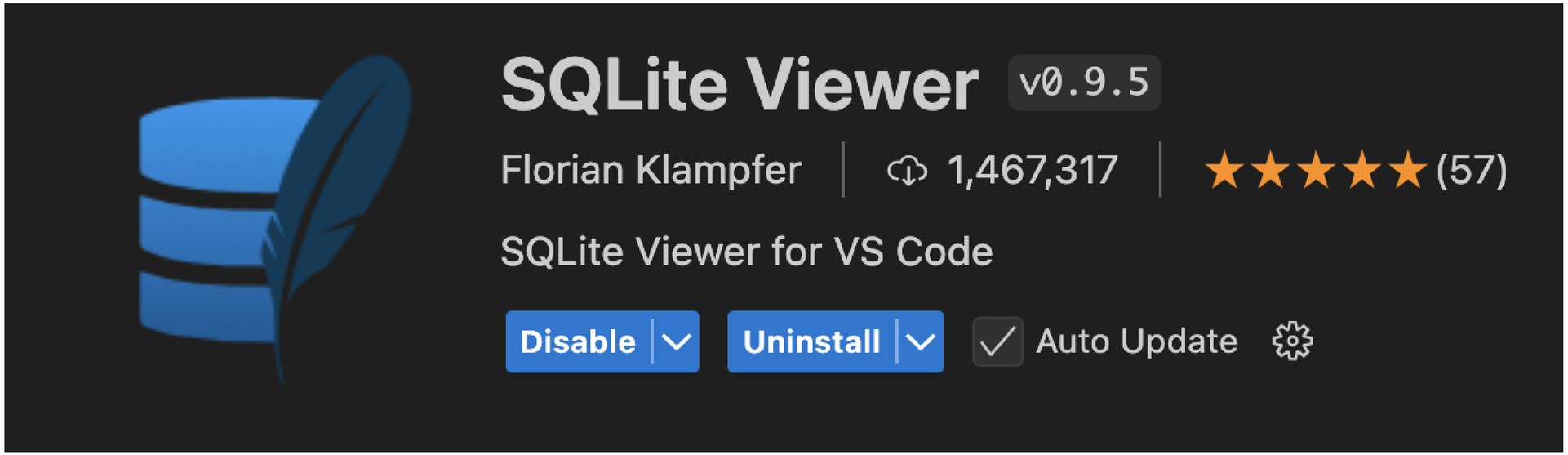
Responses

Code	Description	Links
200	Success	*No links*

Check the Database on new entree







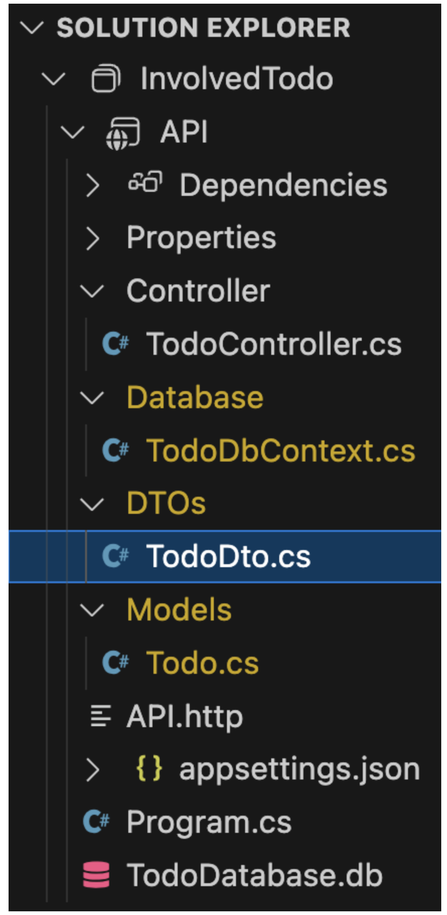
Using DTOs

What is a DTO?

- Short for Data Transfer Object
- Usually consists of fields or properties to hold data
- Acts as a container to carry data between layers (e.g., between the controller and service in a Web API)
- Separation of Concerns, you don't want to expose your DB models to the outside world

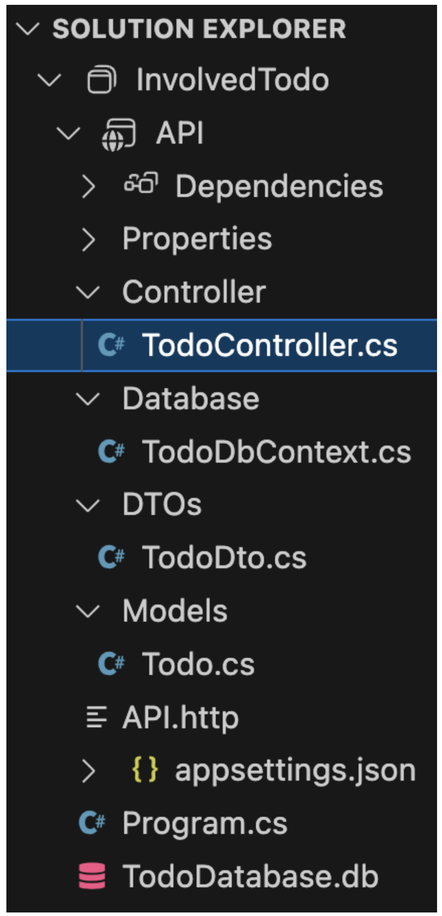
Defining a DTO

```
1   namespace API.DTOs;
2
    0 references
3   public class TodoDto
4   {
        0 references
5       public int Id { get; set; }
6
        0 references
7       public string Title { get; set; }
8
        0 references
9       public string Assignee { get; set; }
10
        0 references
11      public string? Description { get; set; }
12  }
```



Using the DTO

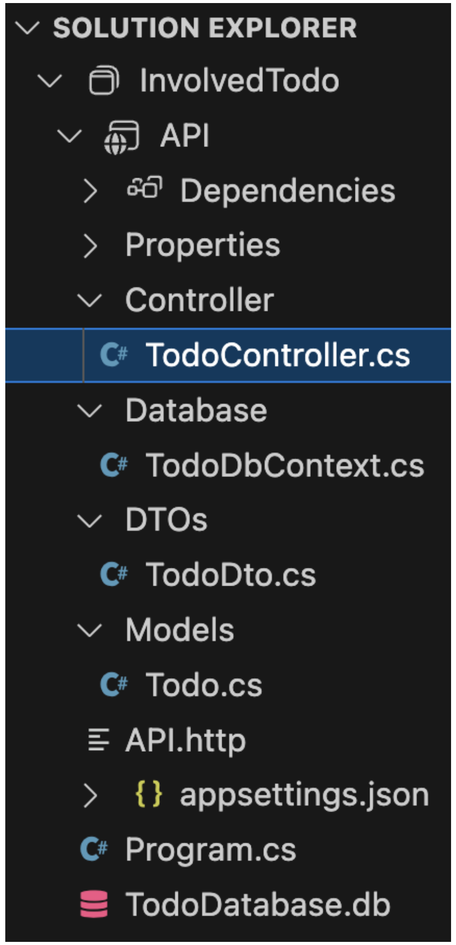
```
[HttpPost("[action]")]
0 references
public IActionResult Add([FromBody] TodoDto todoDto)
{
    var todo = new Todo() {
        Title = todoDto.Title,
        Description = todoDto.Description,
        Assignee = todoDto.Assignee
    };

    _todoDbContext.Todos.Add(todo);
    _todoDbContext.SaveChanges();

    todoDto.Id = todo.Id;

    return Ok(todo);
}
```



Querying the database

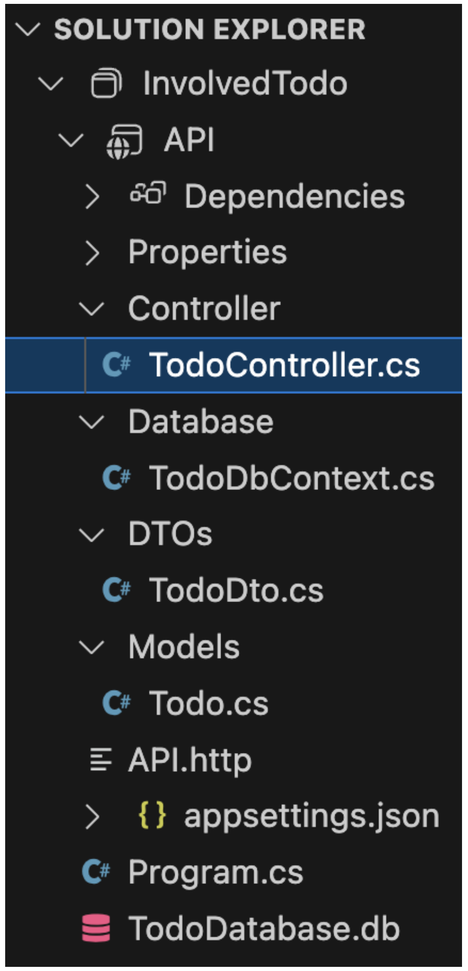
Get all TODOs

```
[HttpGet("[action]")]
0 references
public IActionResult GetAll()
{
    var todoDtos = _todoDbContext.Todos
        .Select(todo => new TodoDto() {
            Id = todo.Id,
            Title = todo.Title,
            Description = todo.Description,
            Assignee = todo.Assignee
        })
        .ToList();

    return Ok(todoDtos);
}
```



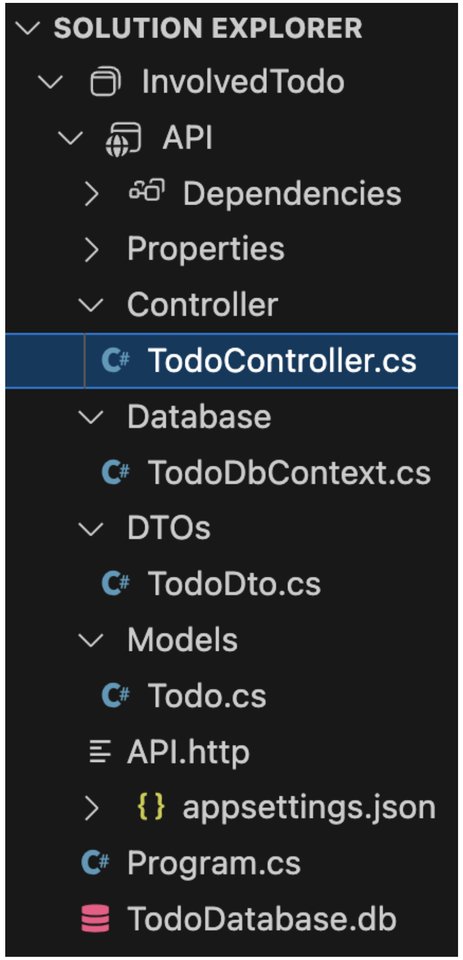
Get a single TODO

```
[HttpGet("[action]/{id}")]
0 references
public IActionResult Get([FromRoute] int id)
{
    var todo = _todoDbContext.Todos.Find(id);

    if(todo == null) {
        return NotFound();
    }

    var todoDto = new TodoDto() {
            Id = todo.Id,
            Title = todo.Title,
            Description = todo.Description,
            Assignee = todo.Assignee
        };

    return Ok(todoDto);
}
```



Searching TODOs

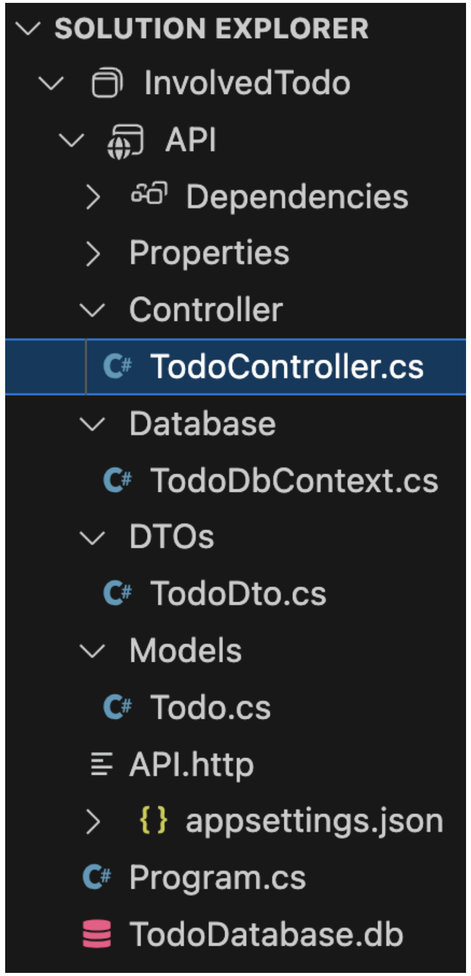
```
[HttpGet("[action]")]
0 references
public IActionResult Search([FromQuery] string? title, [FromQuery] string? assignee)
{
    var todos = _todoDbContext.Todos.AsQueryable();

    if (!string.IsNullOrWhiteSpace(title))
        todos = todos.Where(todo => todo.Title.Contains(title));

    if (!string.IsNullOrWhiteSpace(assignee))
        todos = todos.Where(todo => todo.Assignee.Contains(assignee));

    var todoDtos = todos
        .Select(todo => new TodoDto
        {
            Id = todo.Id,
            Title = todo.Title,
            Assignee = todo.Assignee,
            Description = todo.Description
        })
        .ToList();

    return Ok(todoDtos);
}
```



Manipulating the database

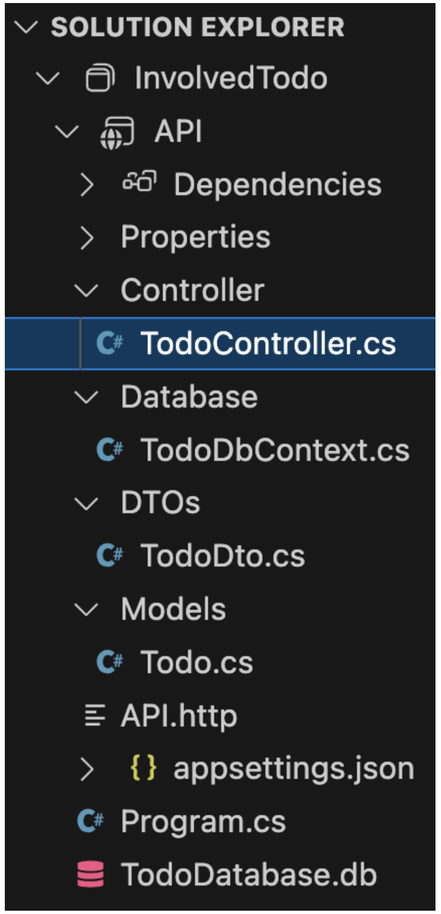
Update a TODO

```
[HttpPut("[action]")]
0 references
public IActionResult Update([FromBody] TodoDto todoDto)
{
    var todo = _todoDbContext.Todos.Find(todoDto.Id);

    if(todo == null) {
        return NotFound();
    }

    todo.Title = todoDto.Title;
    todo.Assignee = todoDto.Assignee;
    todo.Description = todoDto.Description;

    _todoDbContext.Todos.Update(todo);
    _todoDbContext.SaveChanges();

    return Ok();
}
```



Delete a TODO

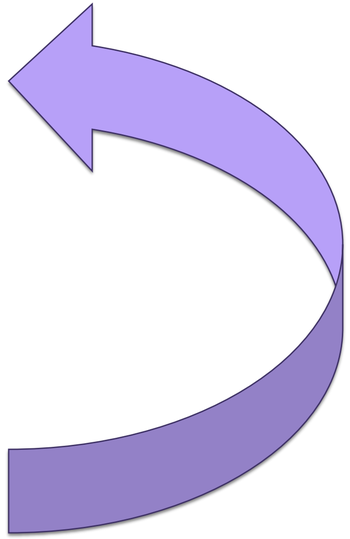
```
[HttpDelete("[action]/{id}")]
0 references
public IActionResult Delete([FromRoute] int id)
{
    var todo = _todoDbContext.Todos.Find(id);

    if(todo == null) {
        return NotFound();
    }

    _todoDbContext.Todos.Remove(todo);
    _todoDbContext.SaveChanges();

    return Ok();
}
```

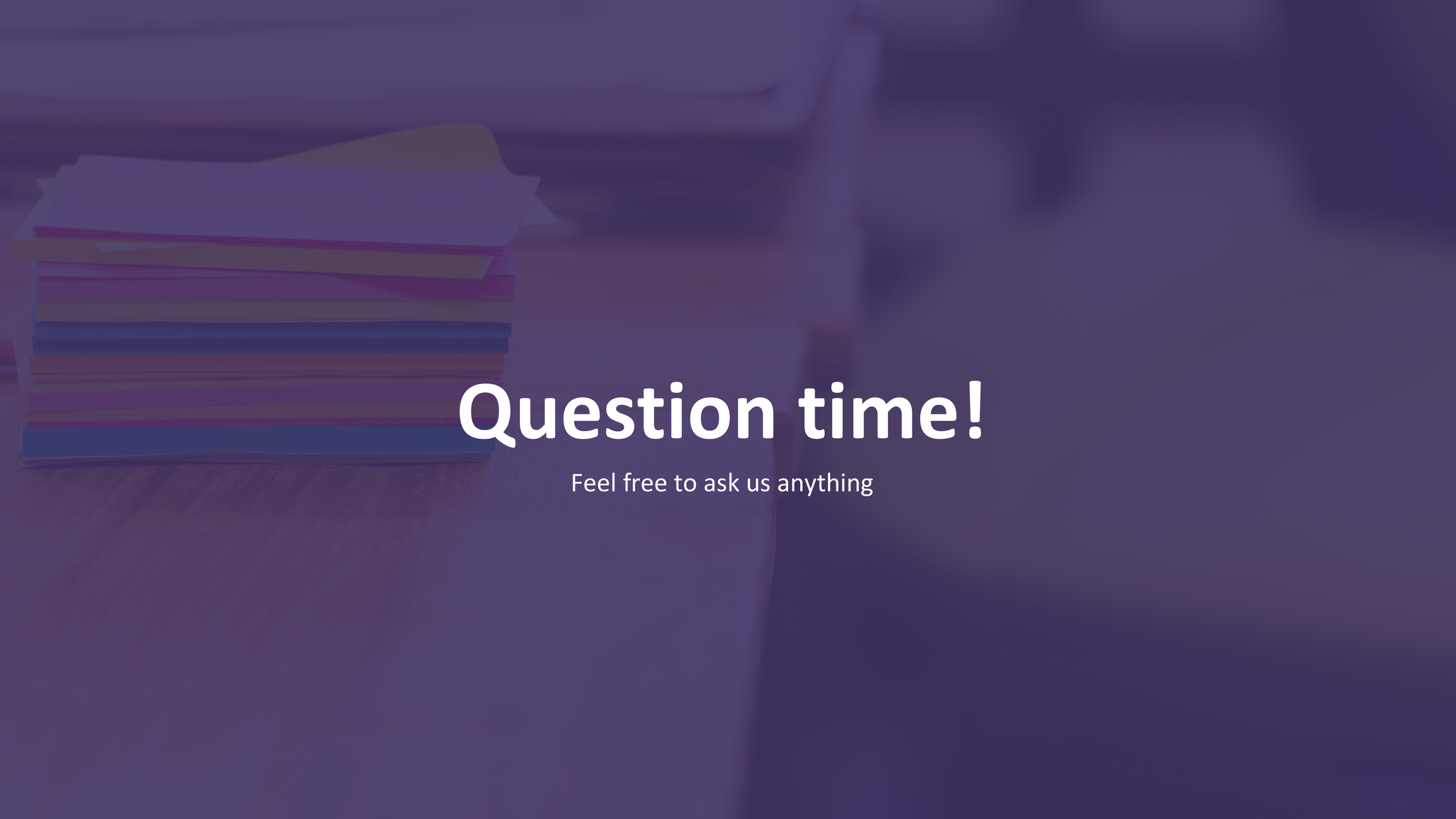



Congrats on your first Web API!

Summary

1. Define domain/database model(s)
2. Define DbContext
3. Review/adjust Program.cs
4. Define DTO(s) (Data Transfer Objects)
5. Implement endpoints
6. Test using Swagger





Question time!

Feel free to ask us anything

@involved_it

Veldkant 35C, 2550 Kontich