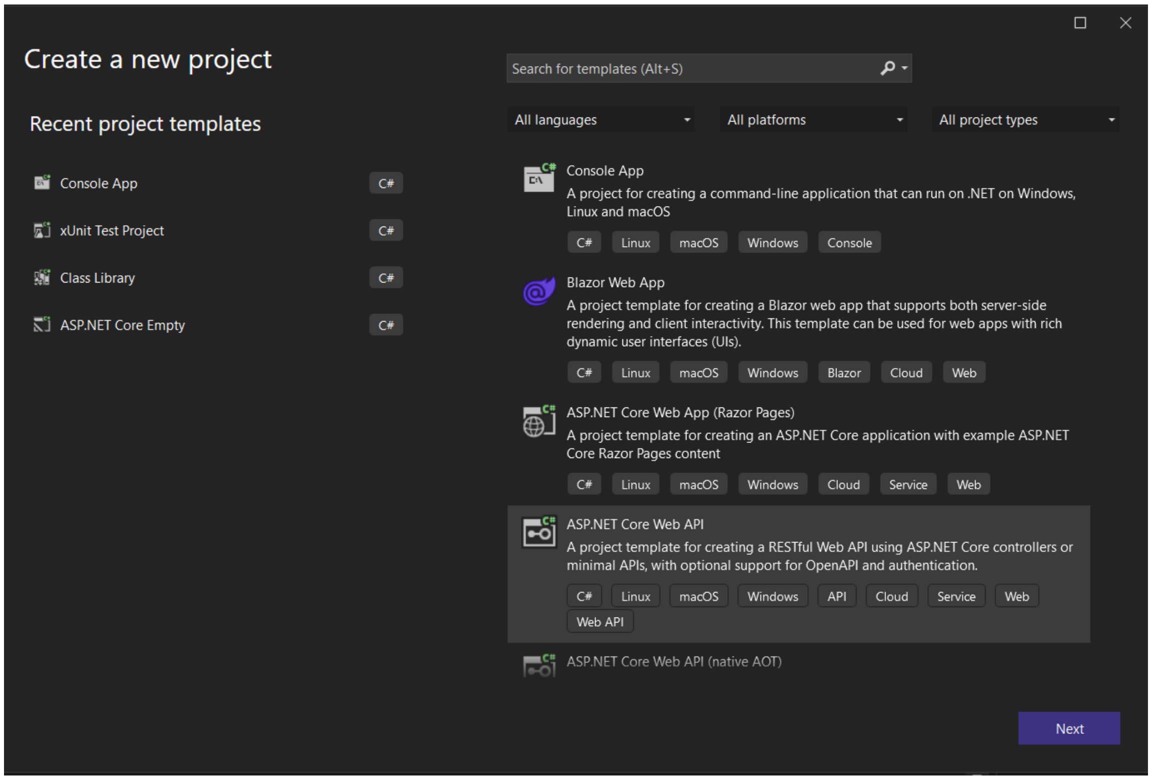
1. Create new backend service
   Create new project with **ASP.NET Core Web API** template



Create a new project

Search for templates (Alt+S)

All languages | All platforms | All project types

Recent project templates

- Console App C#
- xUnit Test Project C#
- Class Library C#
- ASP.NET Core Empty C#

Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

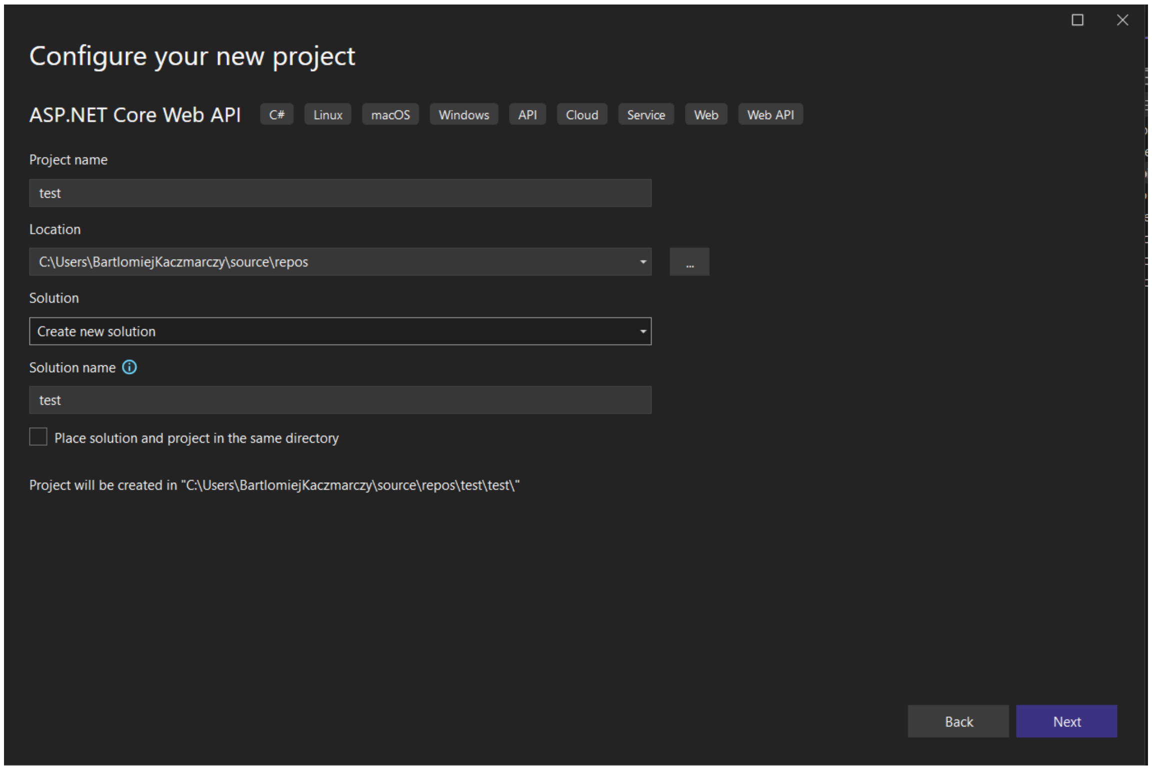
Blazor Web App
A project template for creating a Blazor web app that supports both server-side rendering and client interactivity. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

ASP.NET Core Web App (Razor Pages)
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# Linux macOS Windows API Cloud Service Web Web API

ASP.NET Core Web API (native AOT)

Next



Configure your new project

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

Project name: test

Location: C:\Users\BartlomiejKaczmarczy\source\repos

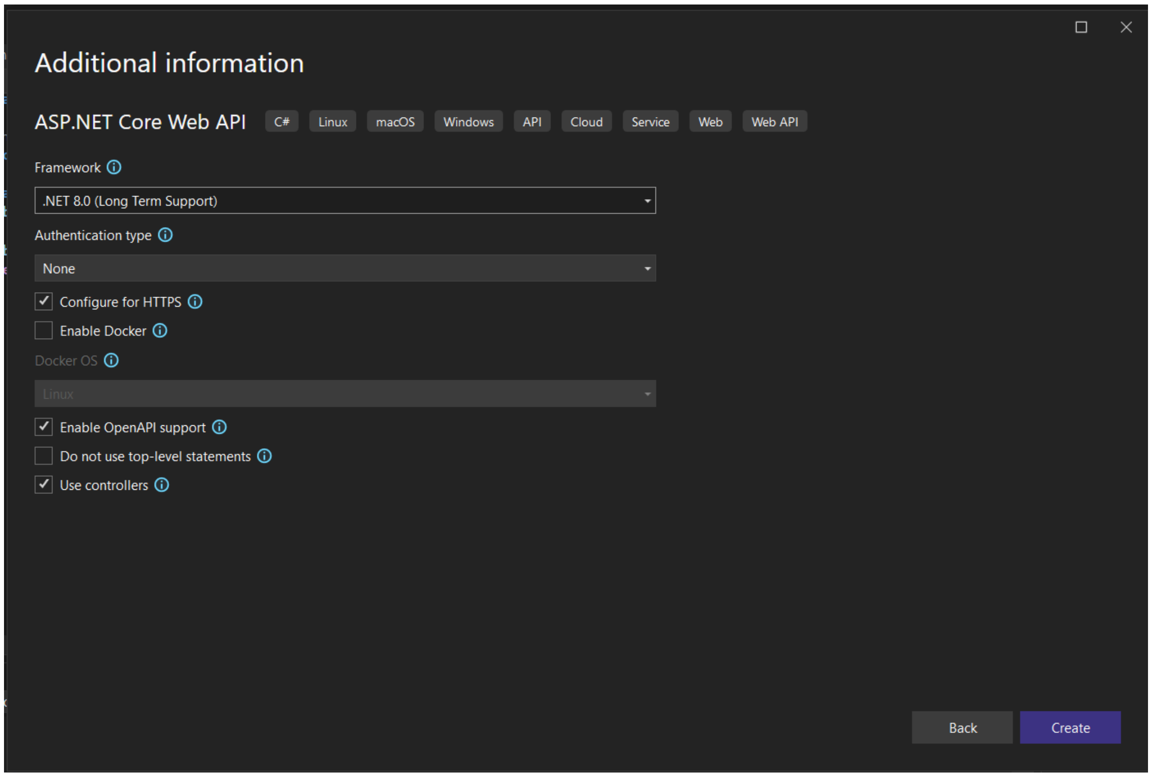
Solution: Create new solution

Solution name: test

☐ Place solution and project in the same directory

Project will be created in "C:\Users\BartlomiejKaczmarczy\source\repos\test\test\"

Back | Next


Additional information

ASP.NET Core Web API

Framework: .NET 8.0 (Long Term Support)

Authentication type: None

[✓] Configure for HTTPS

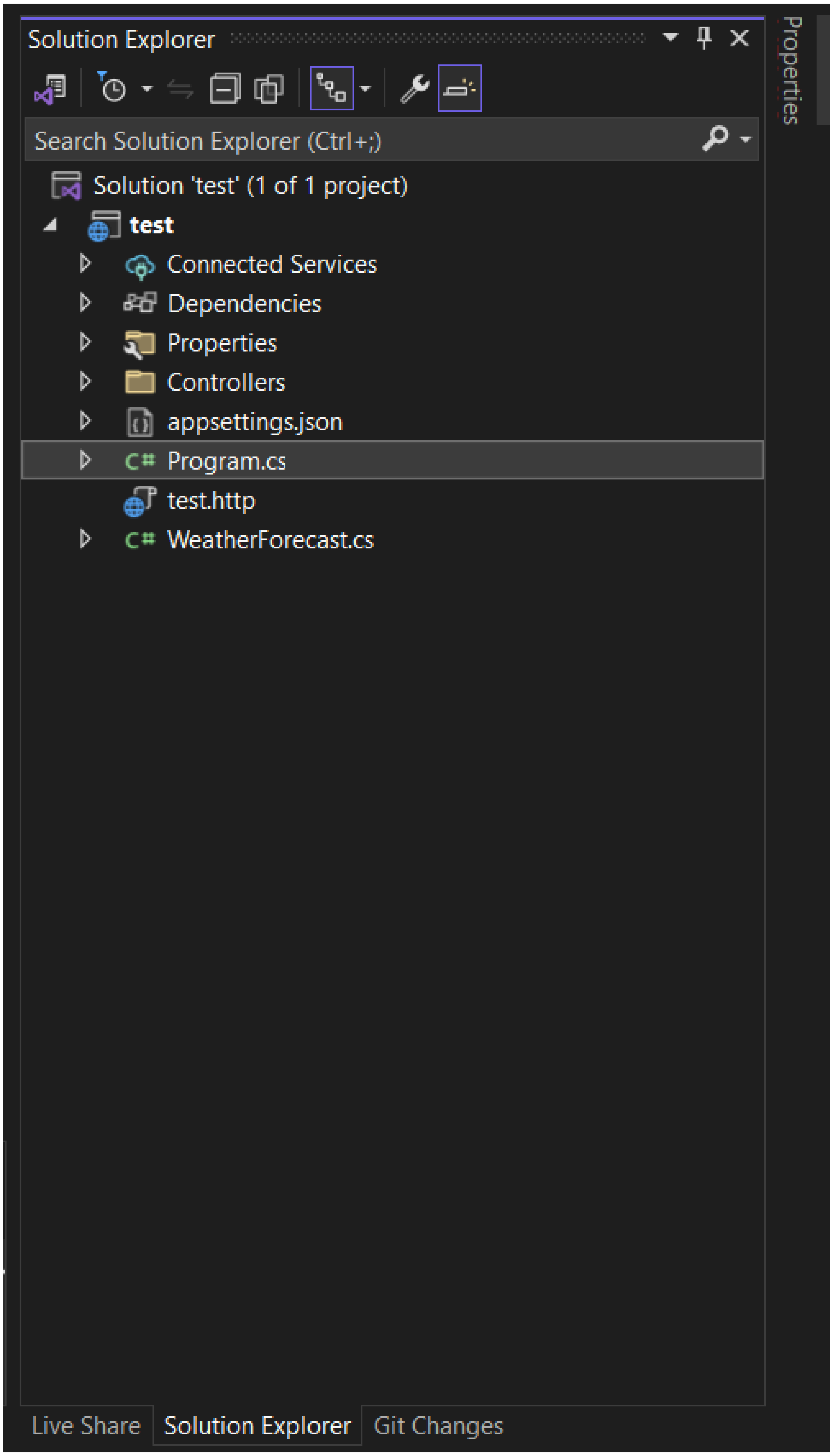
[ ] Enable Docker

Docker OS: Linux

[✓] Enable OpenAPI support

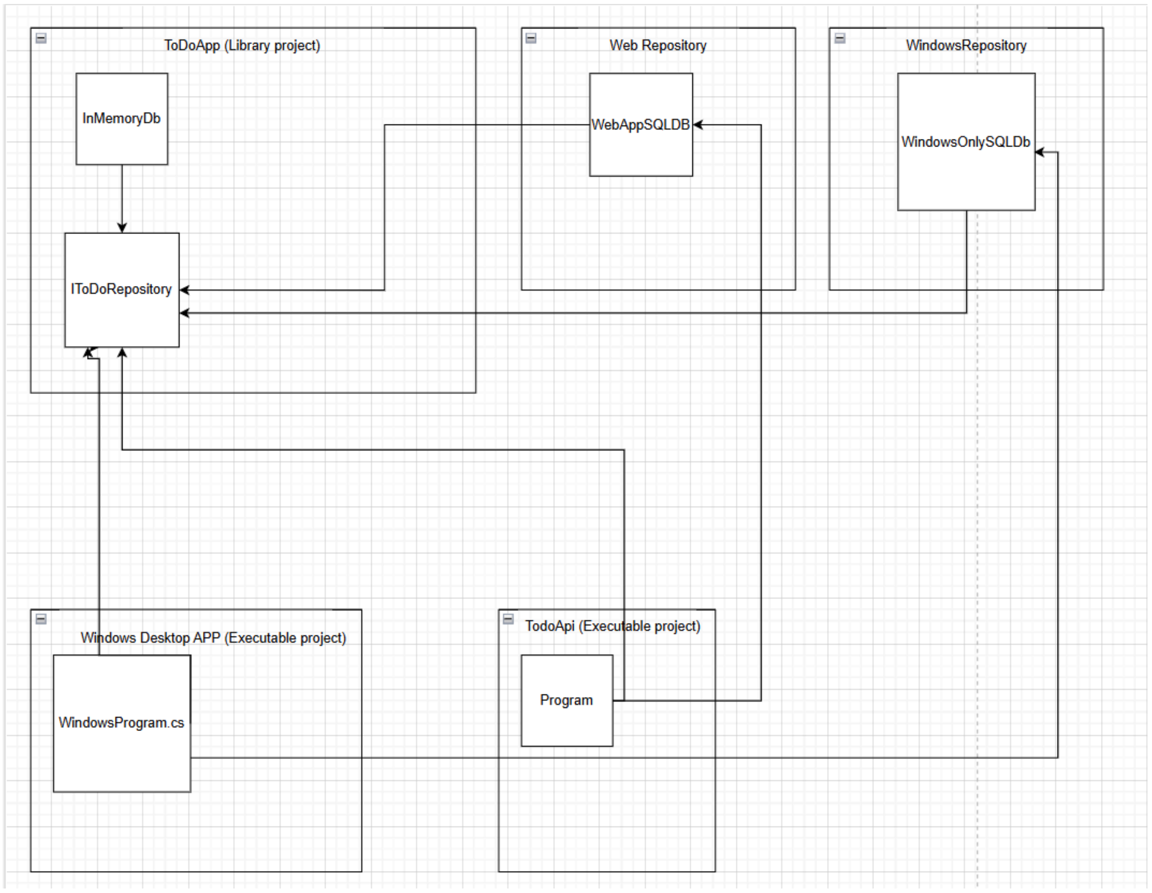
[ ] Do not use top-level statements

[✓] Use controllers


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'test' (1 of 1 project)
test
Connected Services
Dependencies
Properties
Controllers
appsettings.json
Program.cs
test.http
WeatherForecast.cs
Properties
Live Share
Solution Explorer
Git Changes

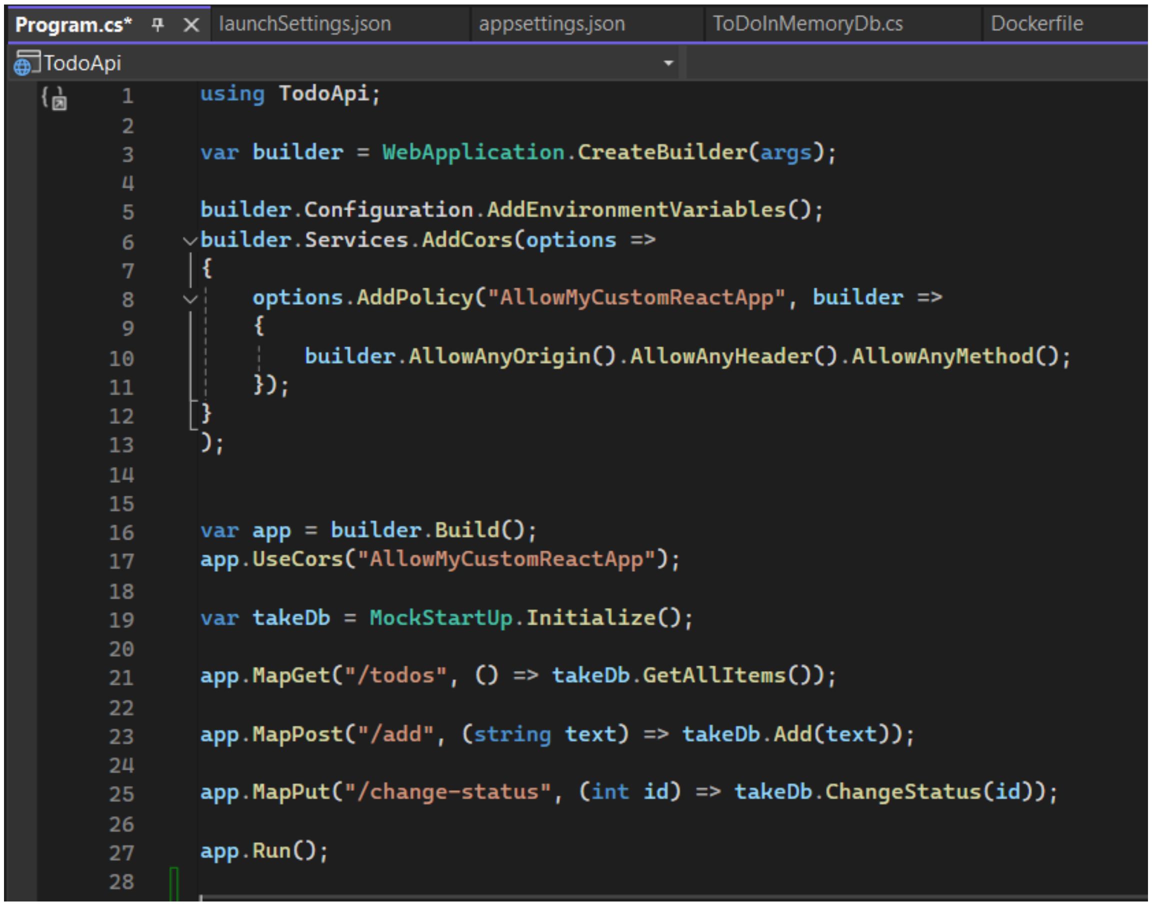
Empty project will look like this^

2. Project architecture



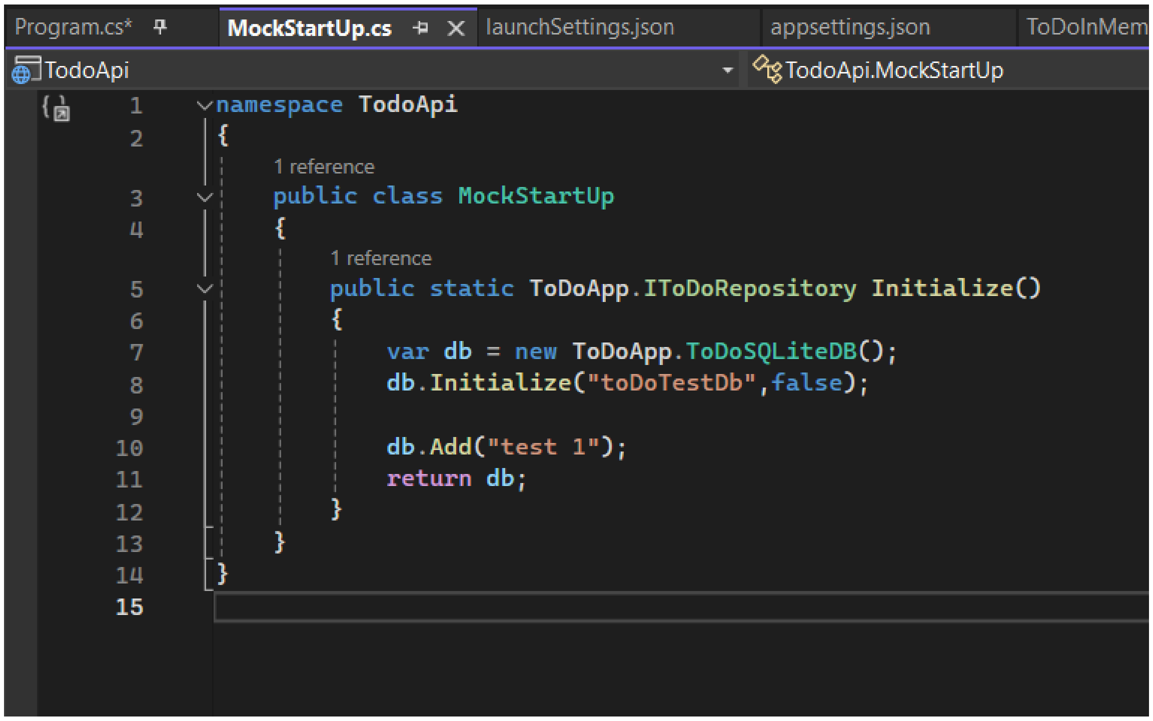
3. Add **ToDoApp** (Library project)
4. Add **TodoApi** (Executable project) (default)
5. Add TestProject (optiopnal)

6. Configure **Program.cs** in **TodoApi** (main file)



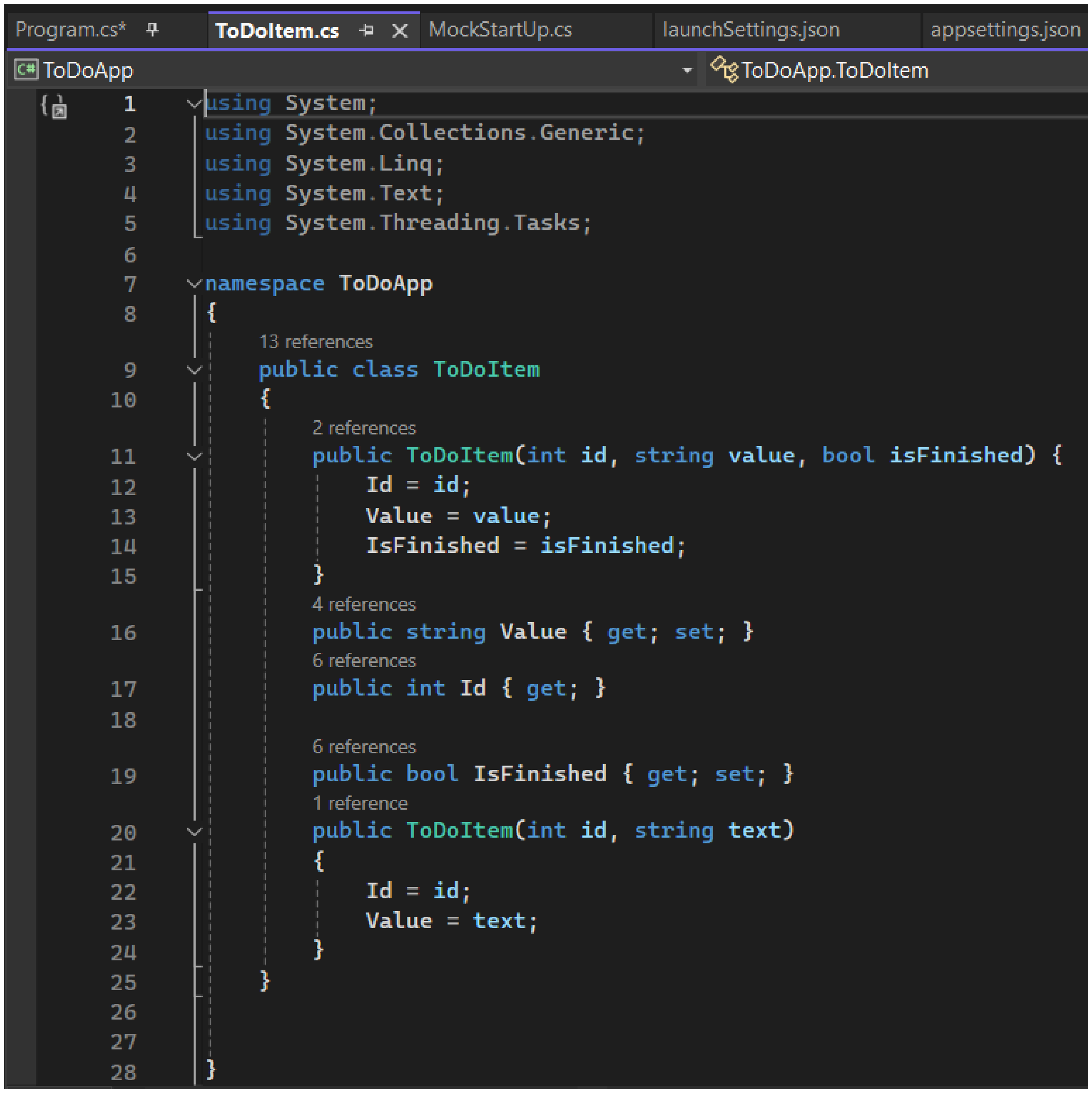
```
using TodoApi;

var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddEnvironmentVariables();
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowMyCustomReactApp", builder =>
    {
        builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod();
    });
}
);


var app = builder.Build();
app.UseCors("AllowMyCustomReactApp");

var takeDb = MockStartUp.Initialize();

app.MapGet("/todos", () => takeDb.GetAllItems());

app.MapPost("/add", (string text) => takeDb.Add(text));

app.MapPut("/change-status", (int id) => takeDb.ChangeStatus(id));

app.Run();

```

7. Add **MockStarUp.cs** to **TodoApi** (add first row row to database)



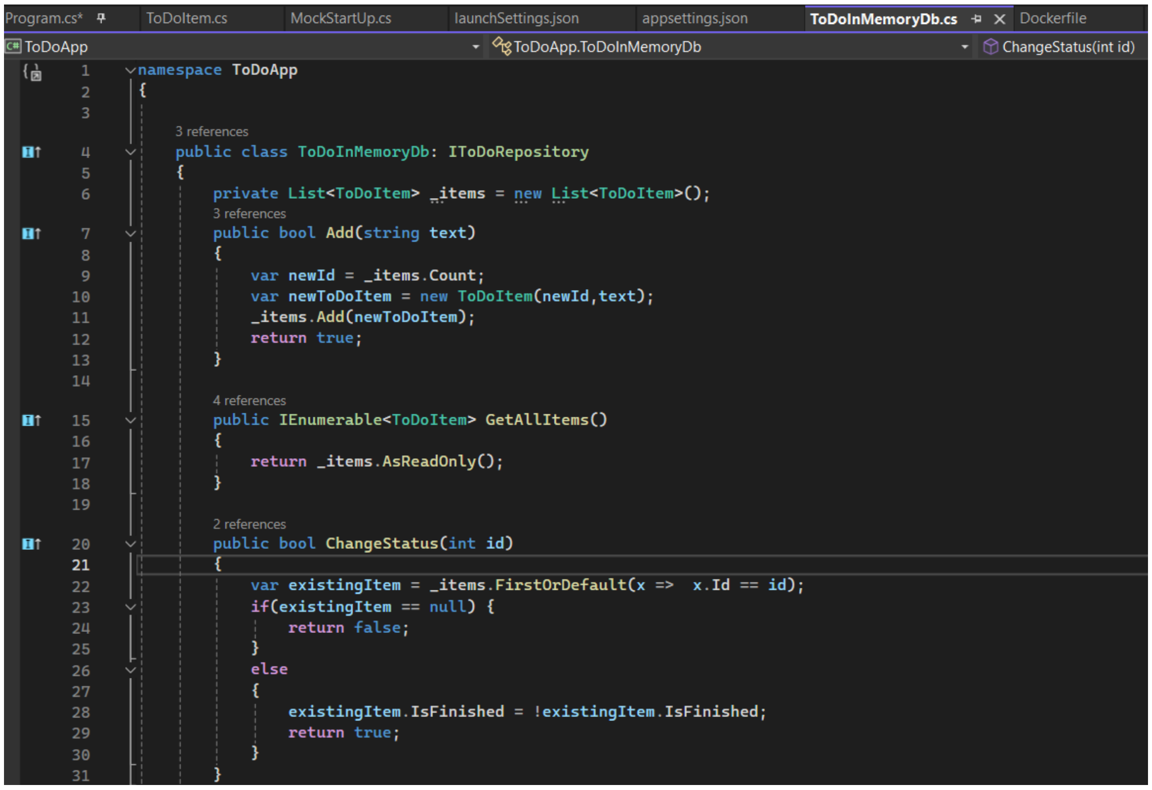
```
namespace TodoApi
{
    public class MockStartUp
    {
        public static ToDoApp.IToDoRepository Initialize()
        {
            var db = new ToDoApp.ToDoSQLiteDB();
            db.Initialize("toDoTestDb",false);

            db.Add("test 1");
            return db;
        }
    }
}

```

8. Add **ToDoItem.cs** to **TodoApp** (declare toDoItem schema)



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ToDoApp
{
    13 references
    public class ToDoItem
    {
        2 references
        public ToDoItem(int id, string value, bool isFinished) {
            Id = id;
            Value = value;
            IsFinished = isFinished;
        }
        4 references
        public string Value { get; set; }
        6 references
        public int Id { get; }

        6 references
        public bool IsFinished { get; set; }
        1 reference
        public ToDoItem(int id, string text)
        {
            Id = id;
            Value = text;
        }
    }


}
```

9. Add **ToDoInMemoryDb.cs** in **TodoApp** to save todos in application memory

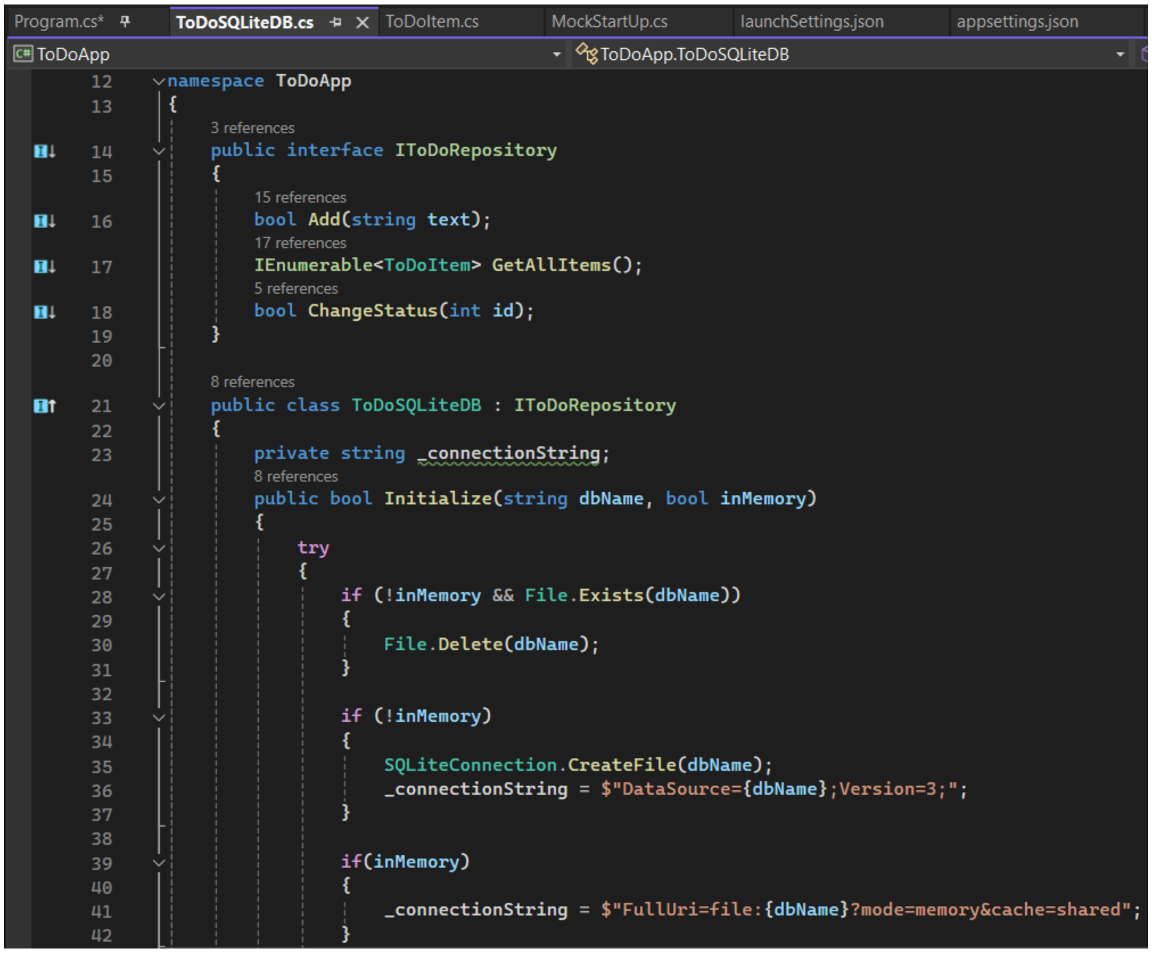


Program.cs* | ToDoItem.cs | MockStartUp.cs | launchSettings.json | appsettings.json | ToDoInMemoryDb.cs | Dockerfile

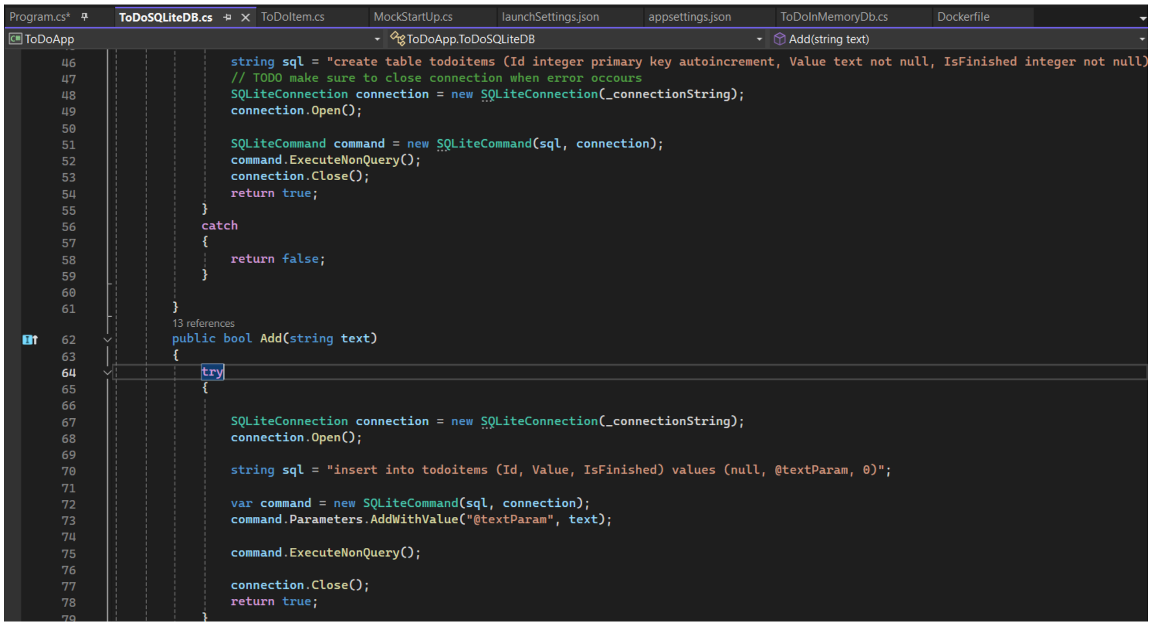
ToDoApp | ToDoApp.ToDoInMemoryDb | ChangeStatus(int id)

```
namespace ToDoApp
{

    3 references
    public class ToDoInMemoryDb: IToDoRepository
    {
        private List<ToDoItem> _items = new List<ToDoItem>();
        3 references
        public bool Add(string text)
        {
            var newId = _items.Count;
            var newToDoItem = new ToDoItem(newId,text);
            _items.Add(newToDoItem);
            return true;
        }

        4 references
        public IEnumerable<ToDoItem> GetAllItems()
        {
            return _items.AsReadOnly();
        }

        2 references
        public bool ChangeStatus(int id)
        {
            var existingItem = _items.FirstOrDefault(x =>  x.Id == id);
            if(existingItem == null) {
                return false;
            }
            else
            {
                existingItem.IsFinished = !existingItem.IsFinished;
                return true;
            }
        }
```

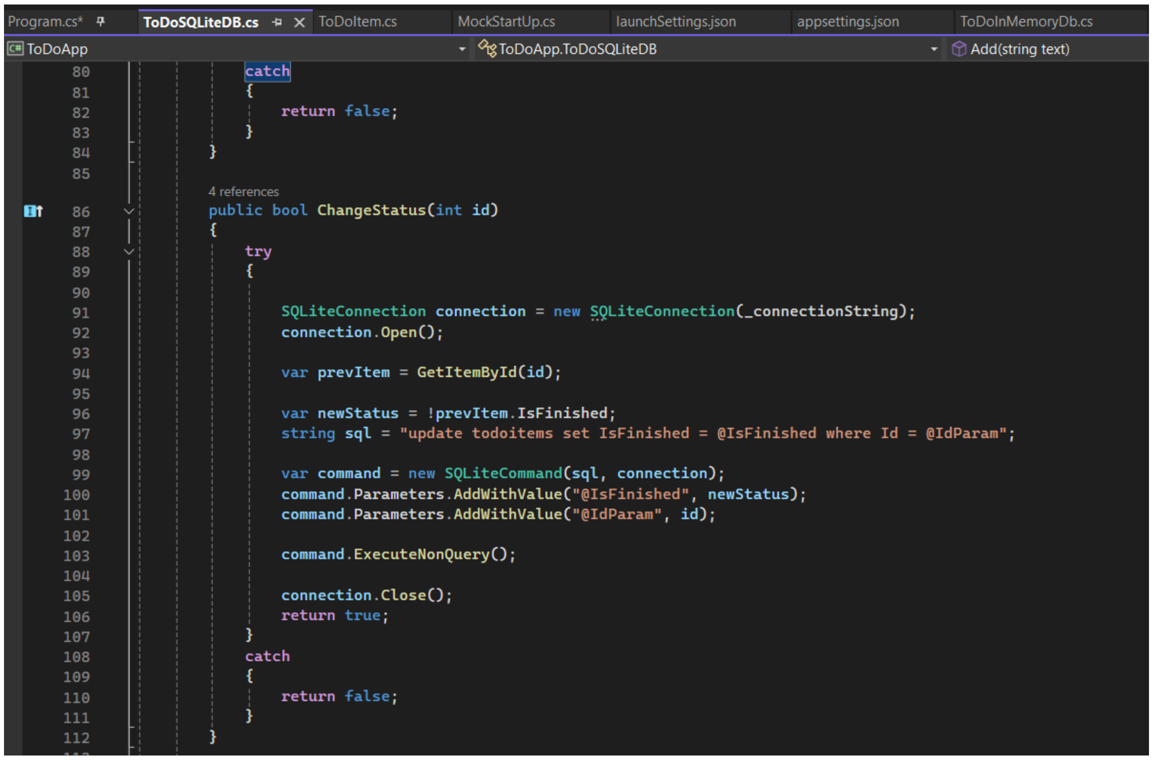
10. Add **ToDoSqliteDb.cs** to **TodoApp** (save todos in sqlite database)



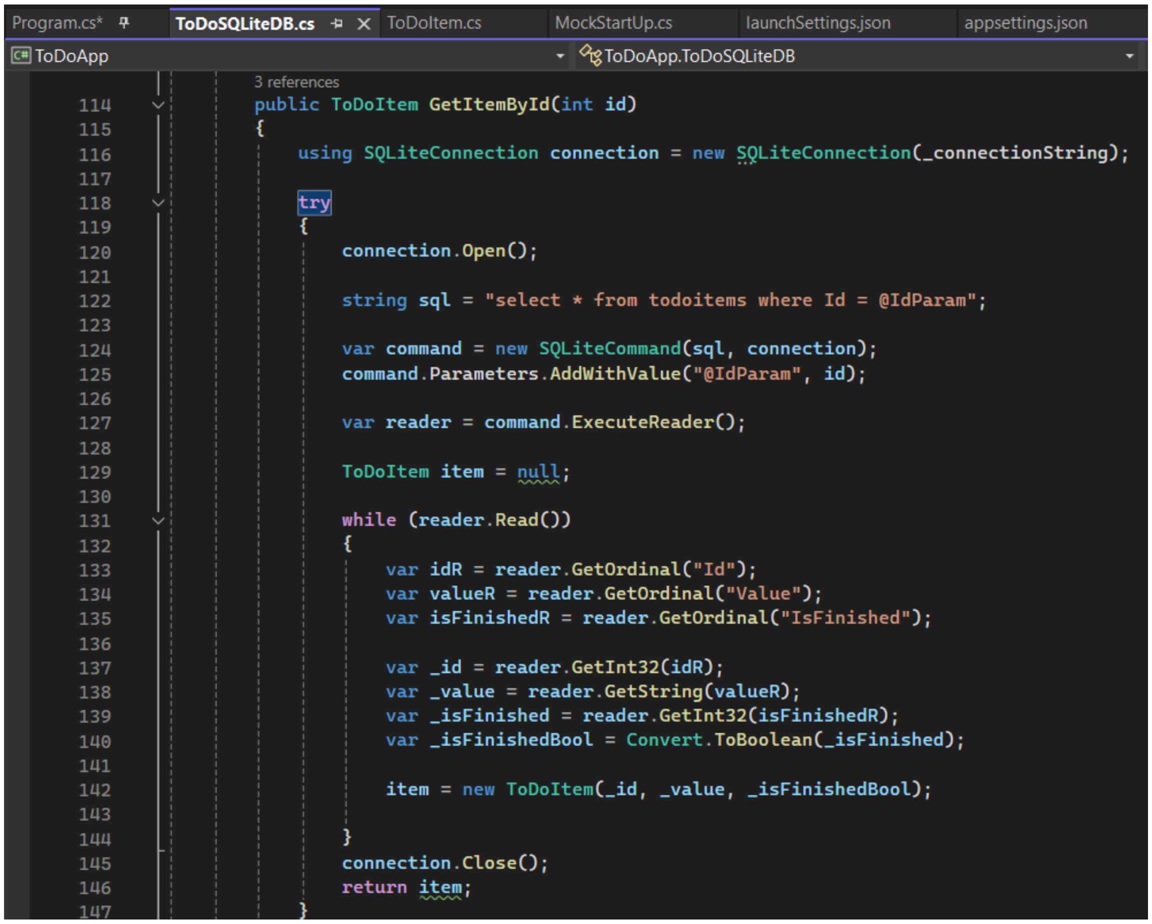
```csharp
namespace ToDoApp
{
    3 references
    public interface IToDoRepository
    {
        15 references
        bool Add(string text);
        17 references
        IEnumerable<ToDoItem> GetAllItems();
        5 references
        bool ChangeStatus(int id);
    }

    8 references
    public class ToDoSQLiteDB : IToDoRepository
    {
        private string _connectionString;
        8 references
        public bool Initialize(string dbName, bool inMemory)
        {
            try
            {
                if (!inMemory && File.Exists(dbName))
                {
                    File.Delete(dbName);
                }

                if (!inMemory)
                {
                    SQLiteConnection.CreateFile(dbName);
                    _connectionString = $"DataSource={dbName};Version=3;";
                }

                if(inMemory)
                {
                    _connectionString = $"FullUri=file:{dbName}?mode=memory&cache=shared";
                }
```



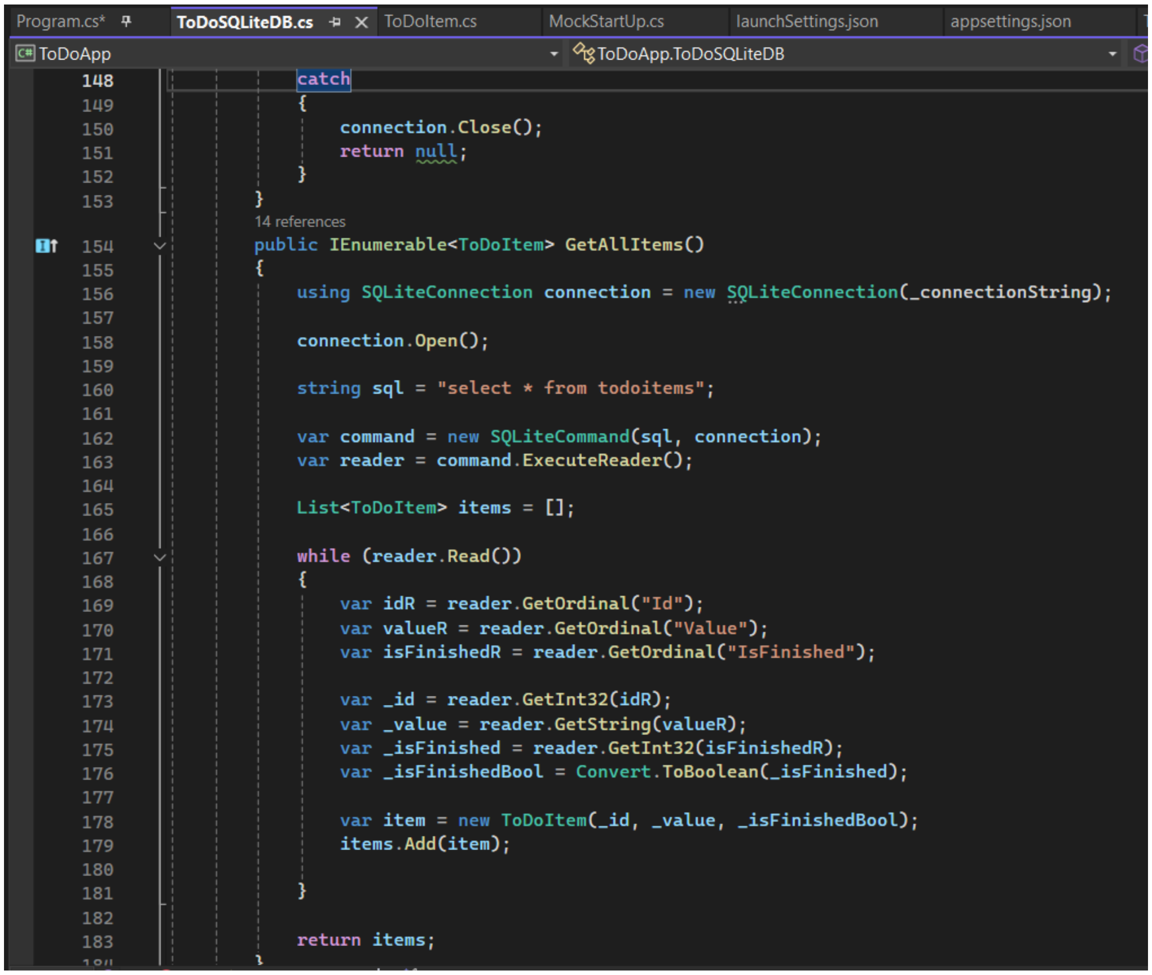
```csharp
                string sql = "create table todoitems (Id integer primary key autoincrement, Value text not null, IsFinished integer not null)
                // TODO make sure to close connection when error occours
                SQLiteConnection connection = new SQLiteConnection(_connectionString);
                connection.Open();

                SQLiteCommand command = new SQLiteCommand(sql, connection);
                command.ExecuteNonQuery();
                connection.Close();
                return true;
            }
            catch
            {
                return false;
            }

        }
        13 references
        public bool Add(string text)
        {
            try
            {

                SQLiteConnection connection = new SQLiteConnection(_connectionString);
                connection.Open();

                string sql = "insert into todoitems (Id, Value, IsFinished) values (null, @textParam, 0)";

                var command = new SQLiteCommand(sql, connection);
                command.Parameters.AddWithValue("@textParam", text);

                command.ExecuteNonQuery();

                connection.Close();
                return true;
            }
```



```
            catch
            {
                return false;
            }
        }

        4 references
        public bool ChangeStatus(int id)
        {
            try
            {

                SQLiteConnection connection = new SQLiteConnection(_connectionString);
                connection.Open();

                var prevItem = GetItemById(id);

                var newStatus = !prevItem.IsFinished;
                string sql = "update todoitems set IsFinished = @IsFinished where Id = @IdParam";

                var command = new SQLiteCommand(sql, connection);
                command.Parameters.AddWithValue("@IsFinished", newStatus);
                command.Parameters.AddWithValue("@IdParam", id);

                command.ExecuteNonQuery();

                connection.Close();
                return true;
            }
            catch
            {
                return false;
            }
        }
```

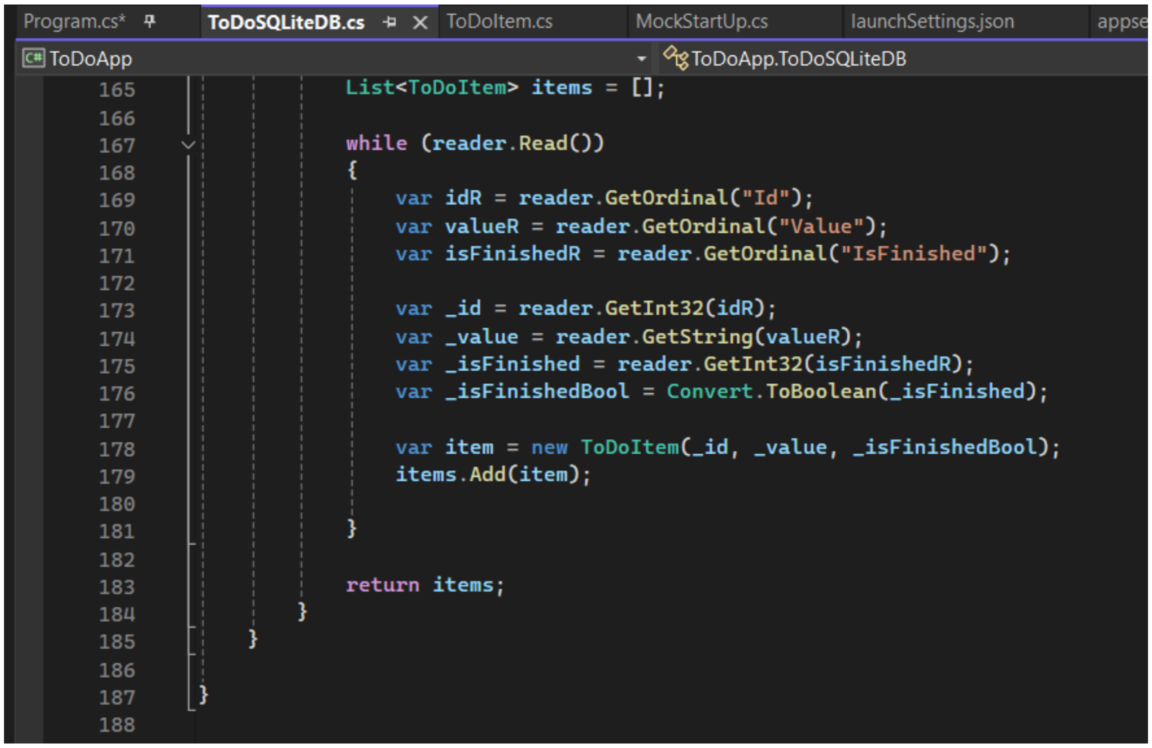


```
        3 references
        public ToDoItem GetItemById(int id)
        {
            using SQLiteConnection connection = new SQLiteConnection(_connectionString);

            try
            {
                connection.Open();

                string sql = "select * from todoitems where Id = @IdParam";

                var command = new SQLiteCommand(sql, connection);
                command.Parameters.AddWithValue("@IdParam", id);

                var reader = command.ExecuteReader();

                ToDoItem item = null;

                while (reader.Read())
                {
                    var idR = reader.GetOrdinal("Id");
                    var valueR = reader.GetOrdinal("Value");
                    var isFinishedR = reader.GetOrdinal("IsFinished");

                    var _id = reader.GetInt32(idR);
                    var _value = reader.GetString(valueR);
                    var _isFinished = reader.GetInt32(isFinishedR);
                    var _isFinishedBool = Convert.ToBoolean(_isFinished);

                    item = new ToDoItem(_id, _value, _isFinishedBool);

                }
                connection.Close();
                return item;
            }
```


Program.cs* ToDoSQLiteDB.cs ToDoItem.cs MockStartUp.cs launchSettings.json appsettings.json

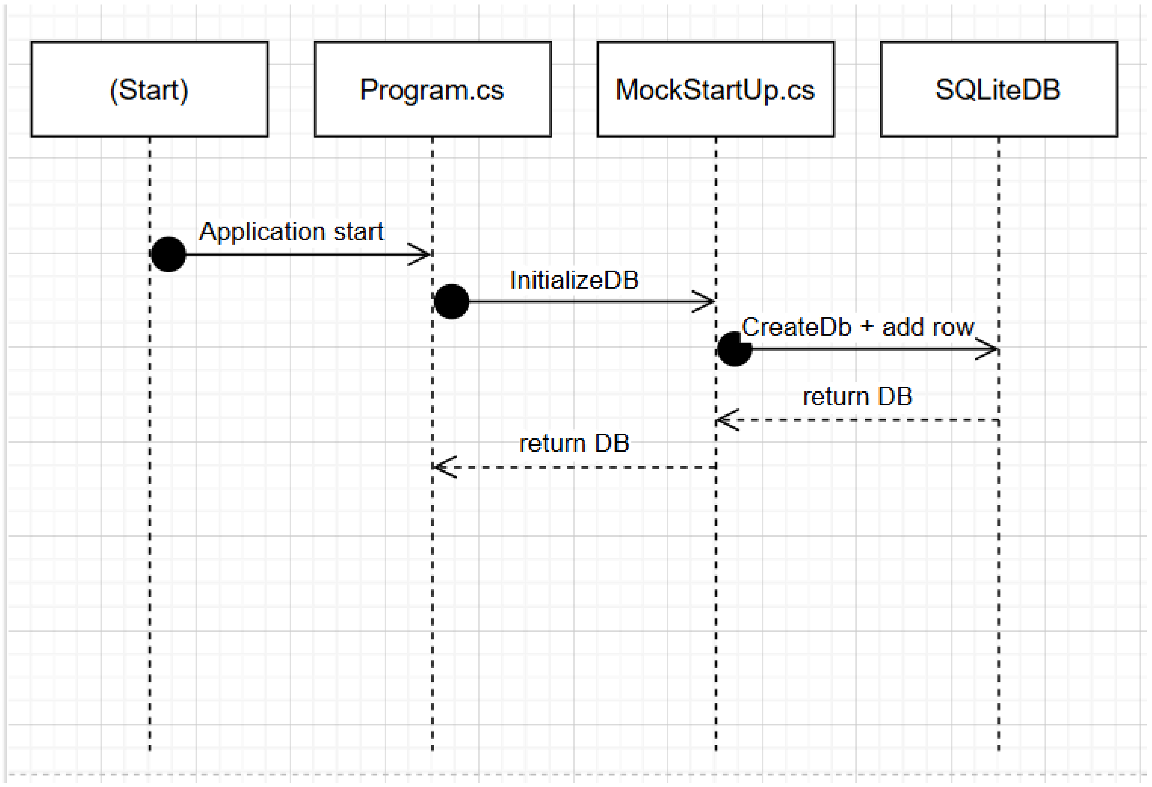
ToDoApp ToDoApp.ToDoSQLiteDB

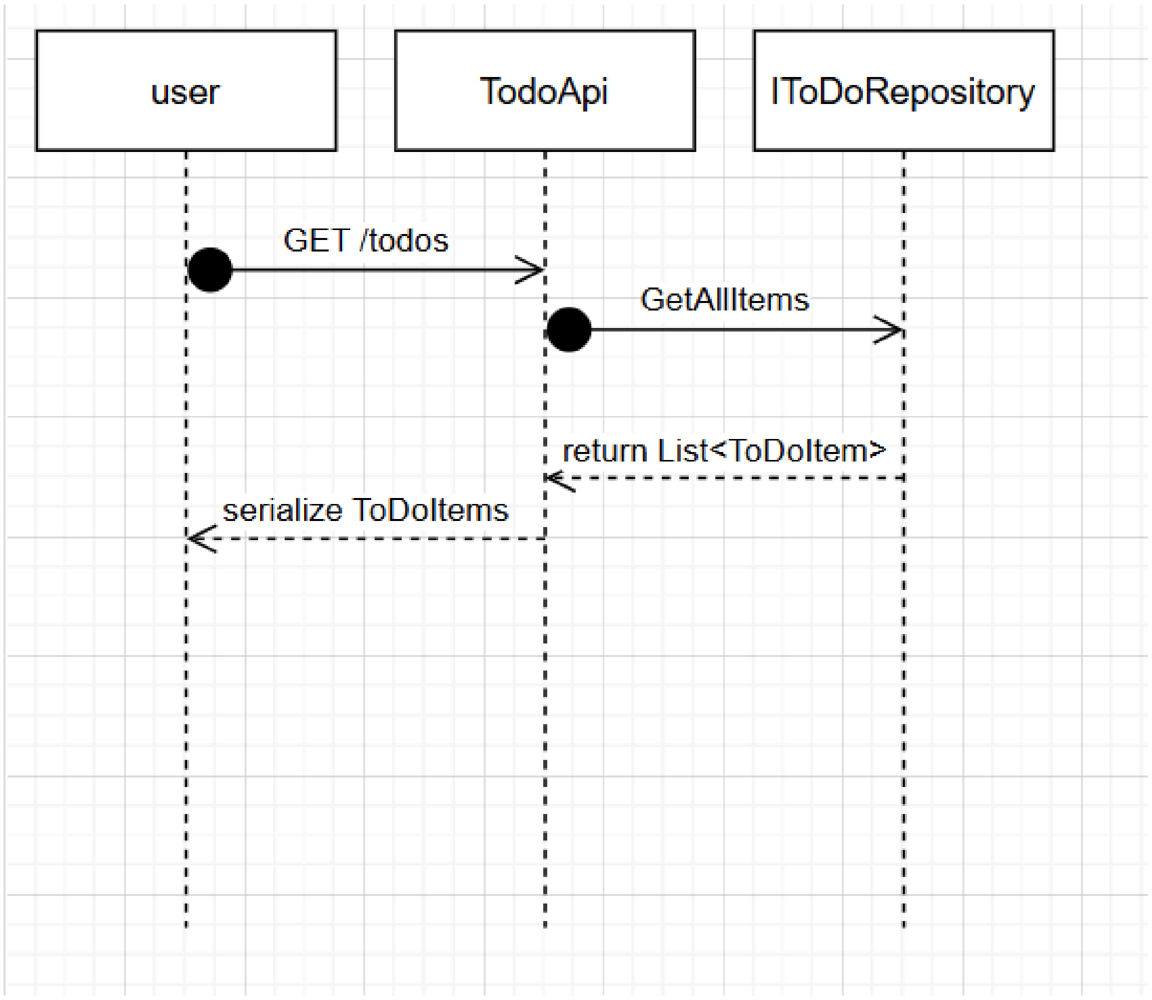
```
            catch
            {
                connection.Close();
                return null;
            }
        }
        14 references
        public IEnumerable<ToDoItem> GetAllItems()
        {
            using SQLiteConnection connection = new SQLiteConnection(_connectionString);

            connection.Open();

            string sql = "select * from todoitems";

            var command = new SQLiteCommand(sql, connection);
            var reader = command.ExecuteReader();

            List<ToDoItem> items = [];

            while (reader.Read())
            {
                var idR = reader.GetOrdinal("Id");
                var valueR = reader.GetOrdinal("Value");
                var isFinishedR = reader.GetOrdinal("IsFinished");

                var _id = reader.GetInt32(idR);
                var _value = reader.GetString(valueR);
                var _isFinished = reader.GetInt32(isFinishedR);
                var _isFinishedBool = Convert.ToBoolean(_isFinished);

                var item = new ToDoItem(_id, _value, _isFinishedBool);
                items.Add(item);

            }

            return items;
```



```
                List<ToDoItem> items = [];

                while (reader.Read())
                {
                    var idR = reader.GetOrdinal("Id");
                    var valueR = reader.GetOrdinal("Value");
                    var isFinishedR = reader.GetOrdinal("IsFinished");

                    var _id = reader.GetInt32(idR);
                    var _value = reader.GetString(valueR);
                    var _isFinished = reader.GetInt32(isFinishedR);
                    var _isFinishedBool = Convert.ToBoolean(_isFinished);

                    var item = new ToDoItem(_id, _value, _isFinishedBool);
                    items.Add(item);

                }

                return items;
            }
        }

}
```

11. How it works:




user
TodoApi
IToDoRepository
GET /todos
GetAllItems
return List<ToDoItem>
serialize ToDoItems

12.