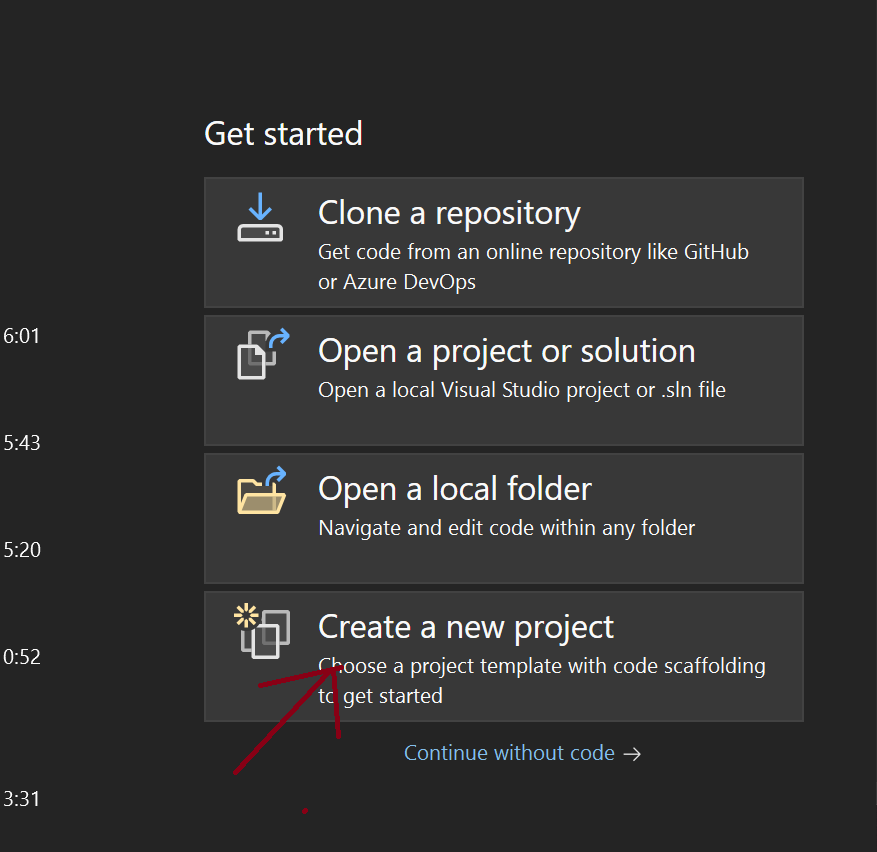
**Step 1**

Create new MVC Application from visual studio



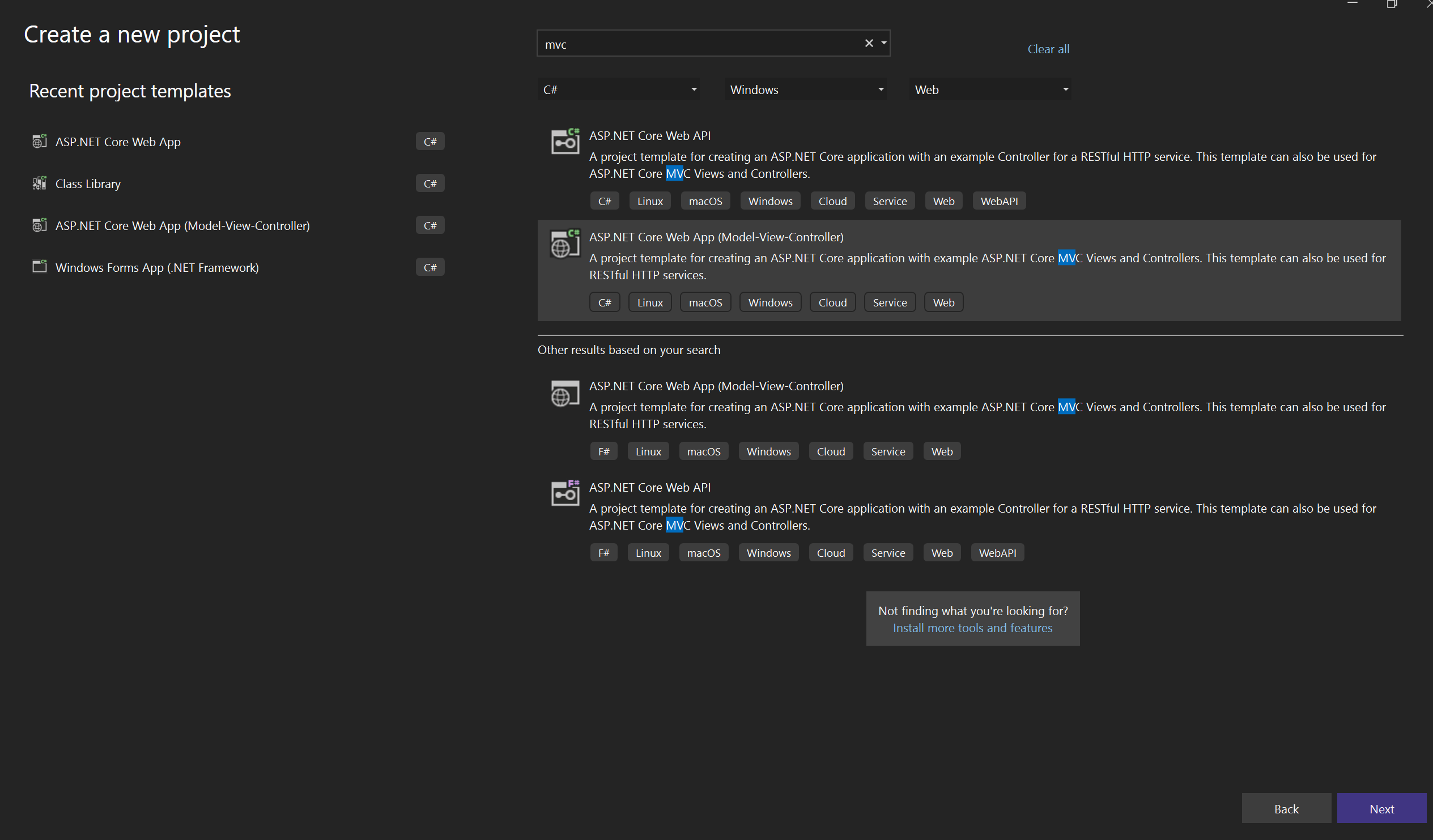


Figure 2 – Select MVC Template

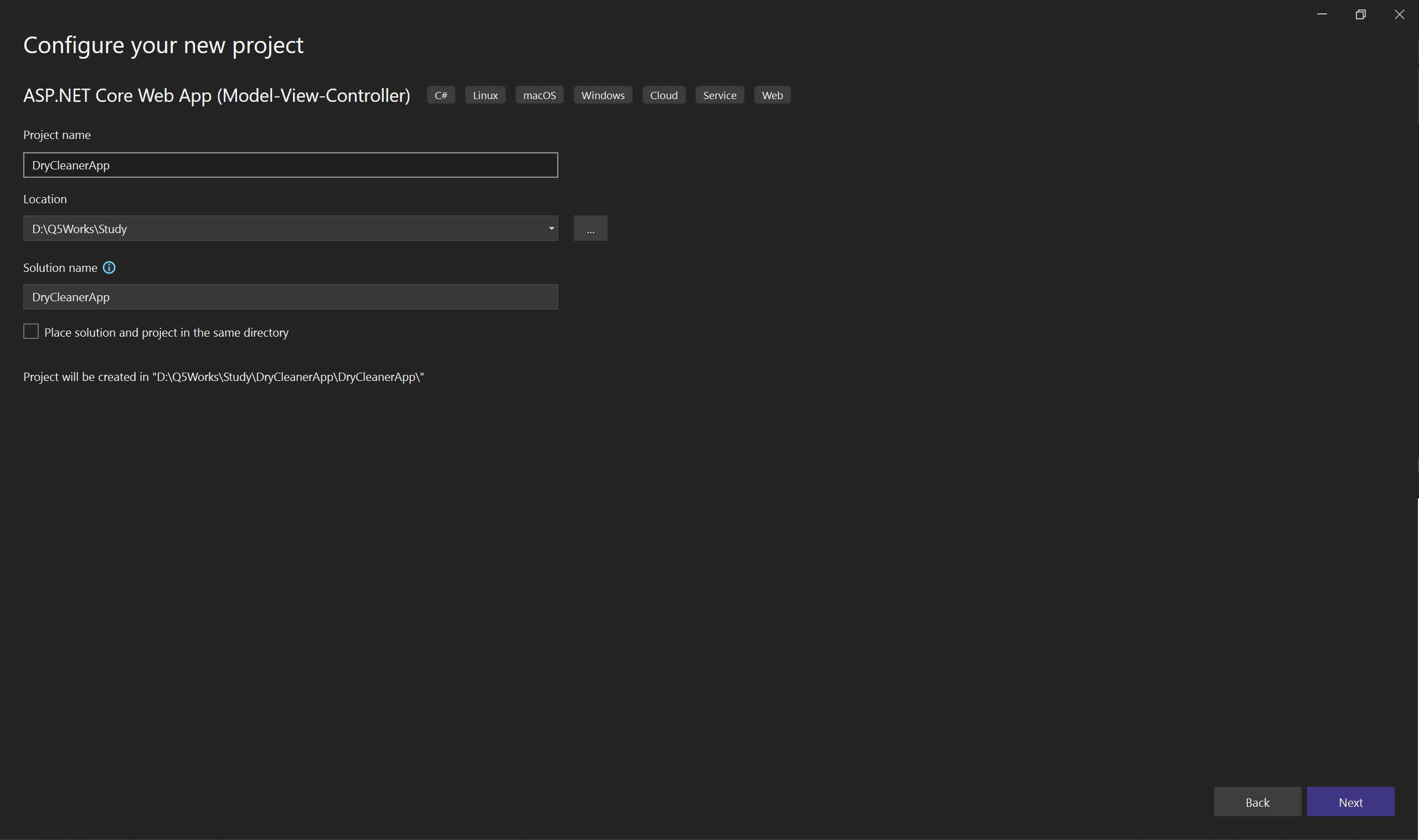


Figure 3 – Give project name and select location for saving project files

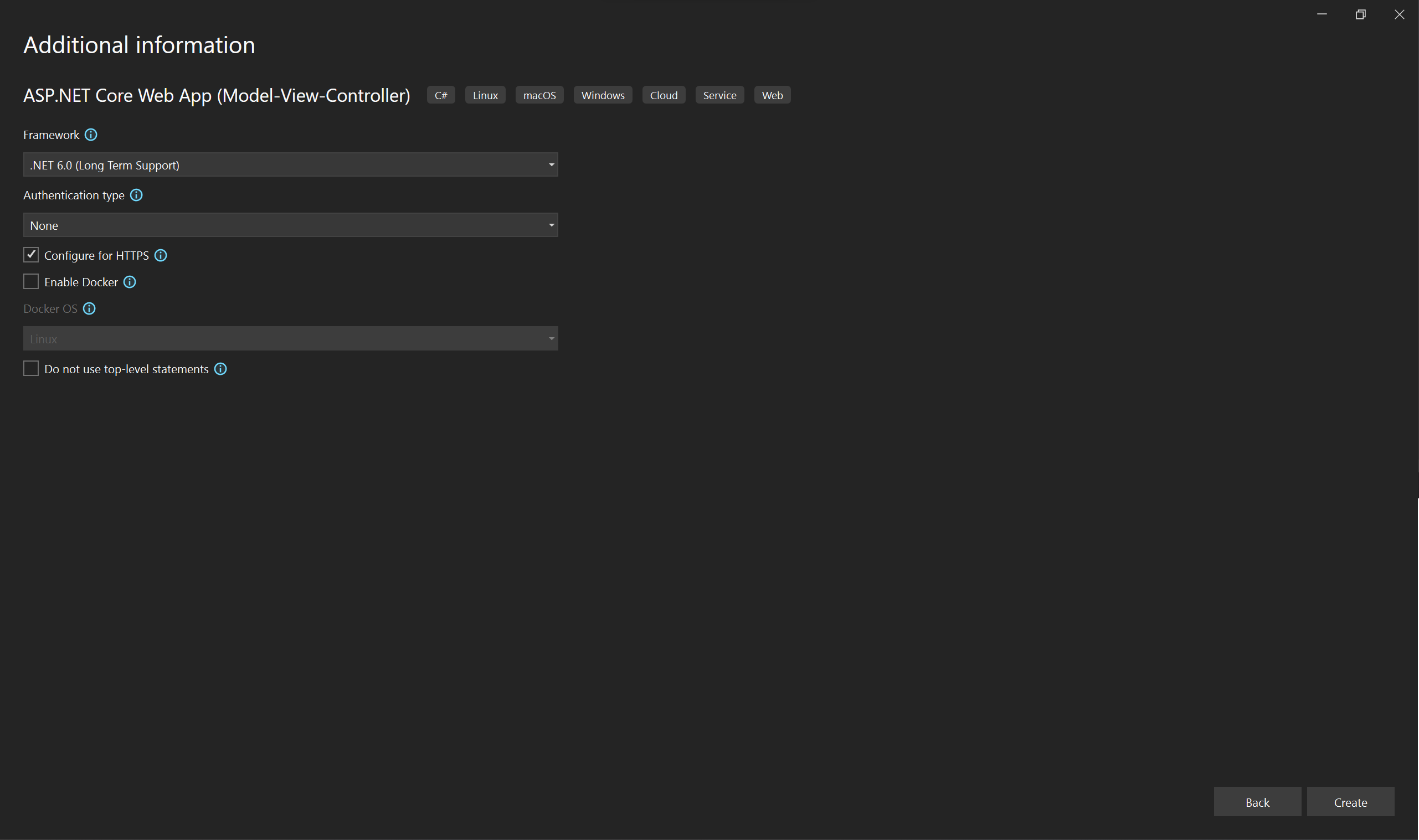


Figure 4 – Select Faremwork (.Net6)

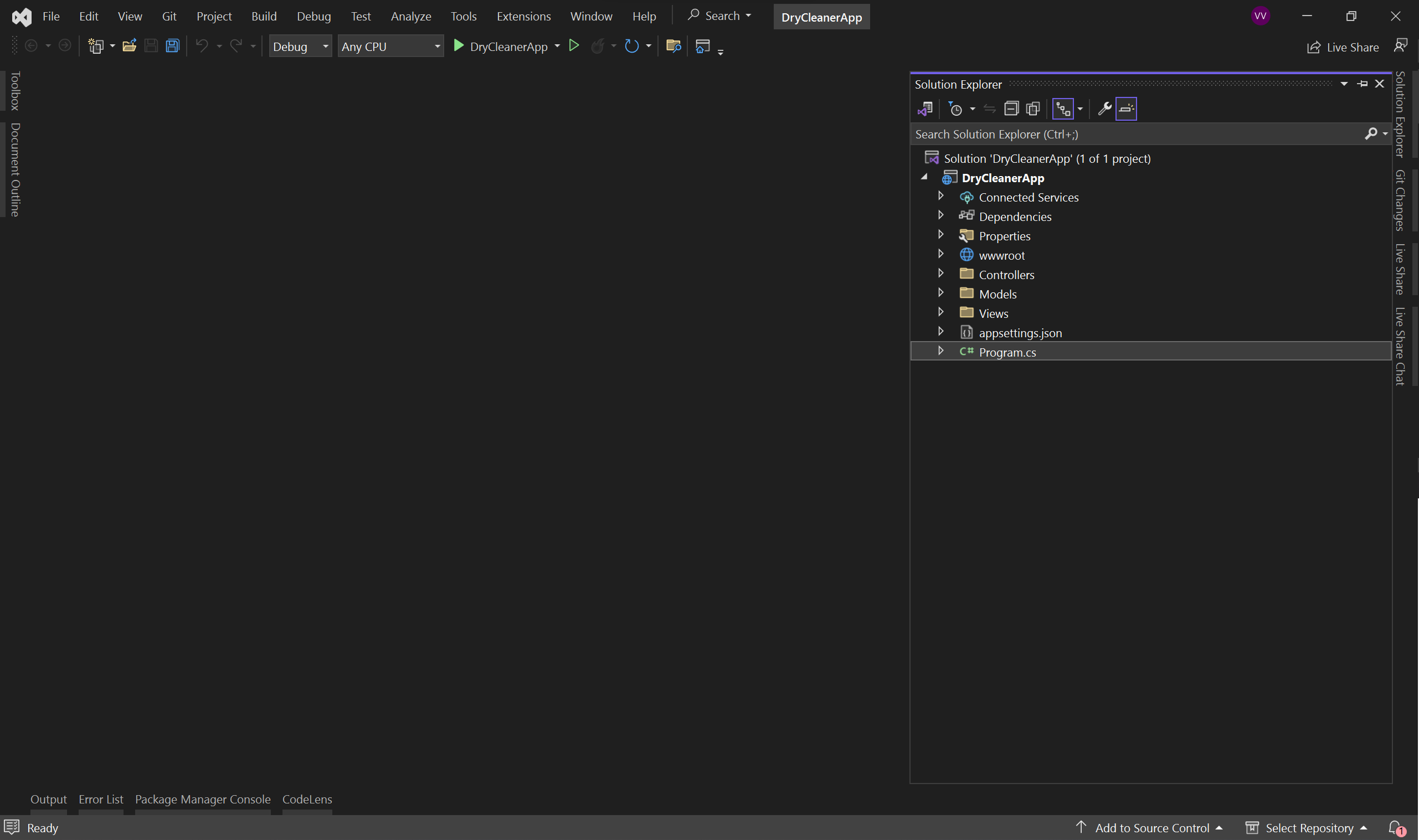


Figure 5 – Project created you can check on the Solution explorer Right Side

If getting build errors Install Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.Tools, MySql.Data Using nuget package manager

**Step 2**

Create New Class Library for business logic related files to save.

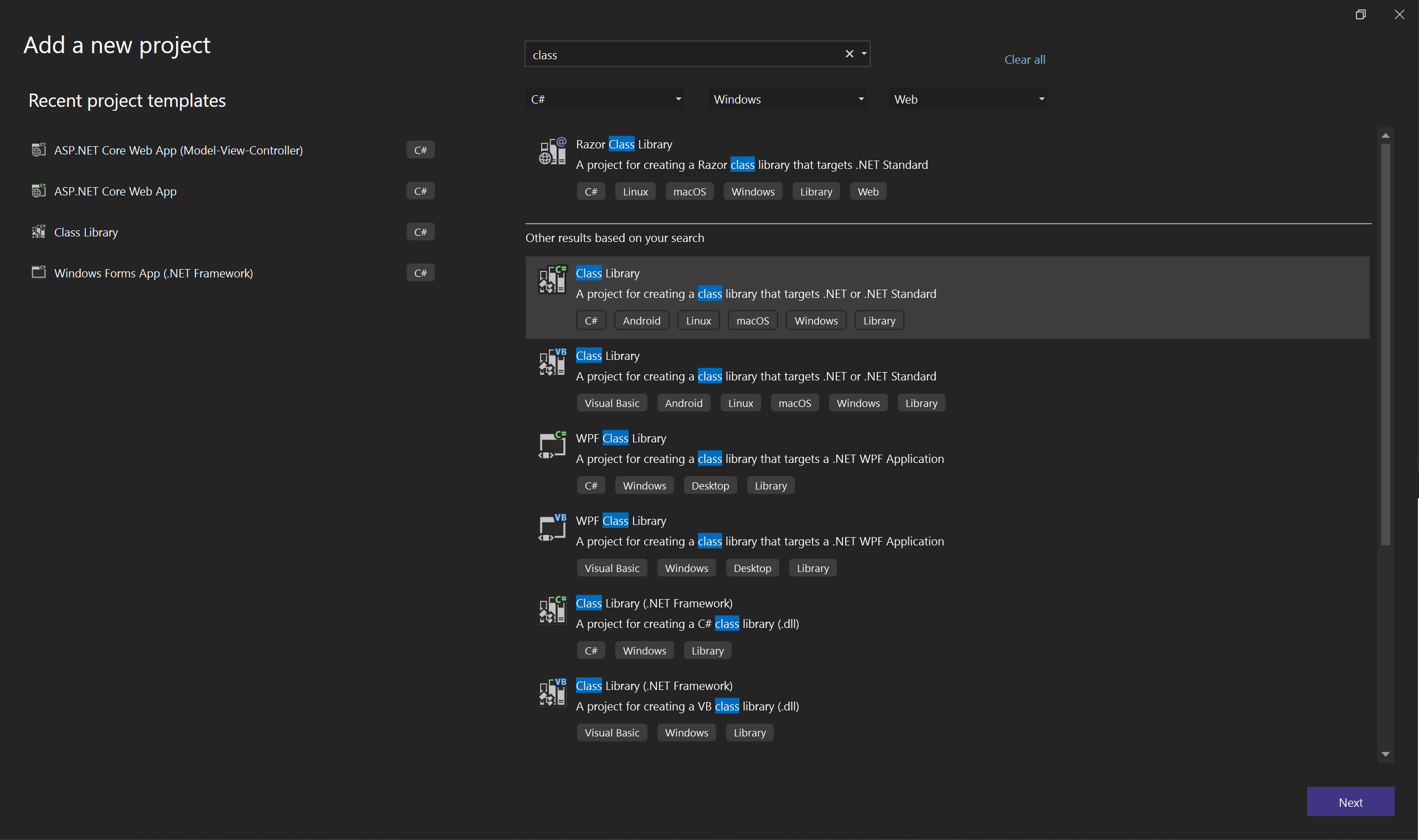


Figure 1 – Select class library from project template list

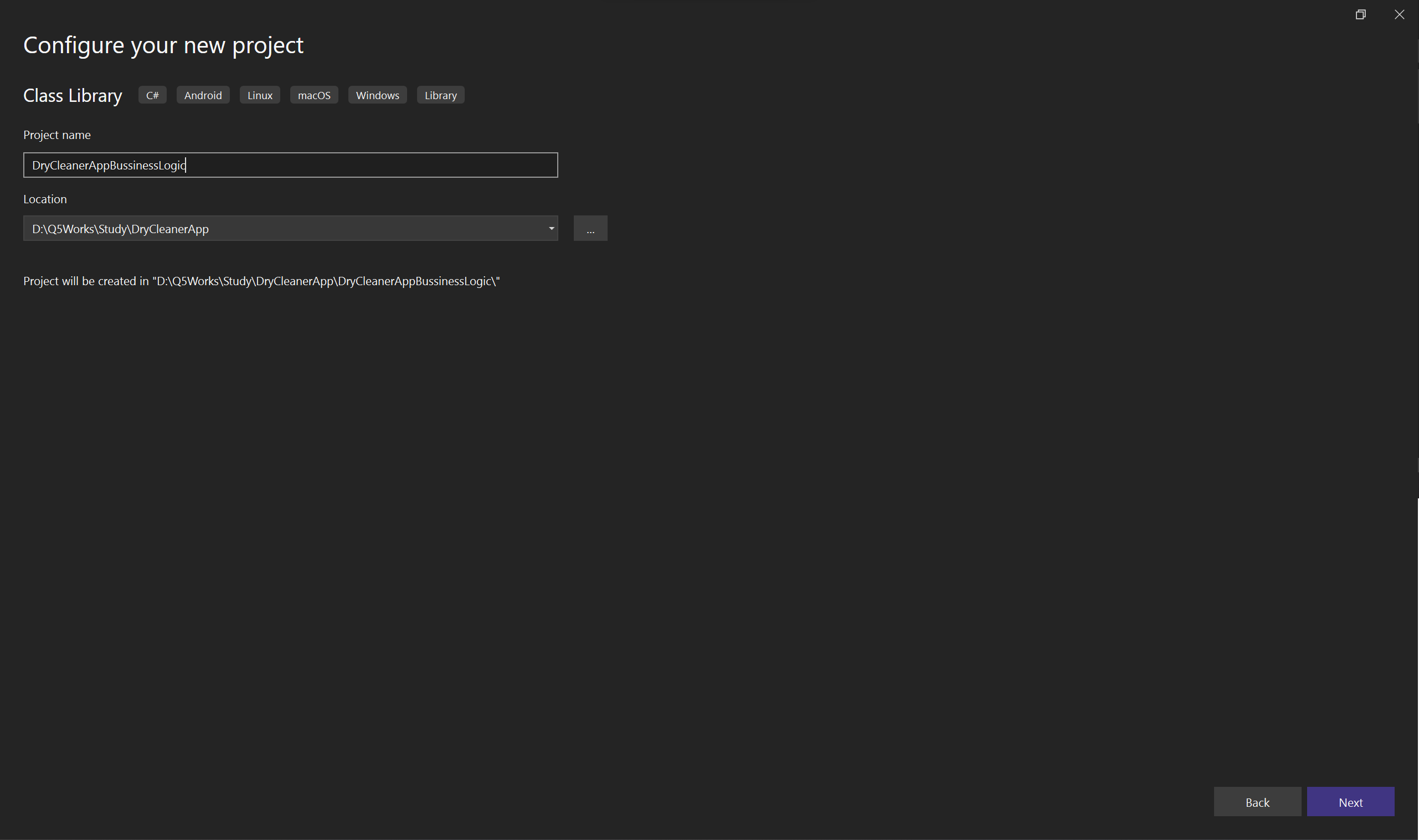


Figure 2 – Giving Proper name and selecting proper Folder for class library

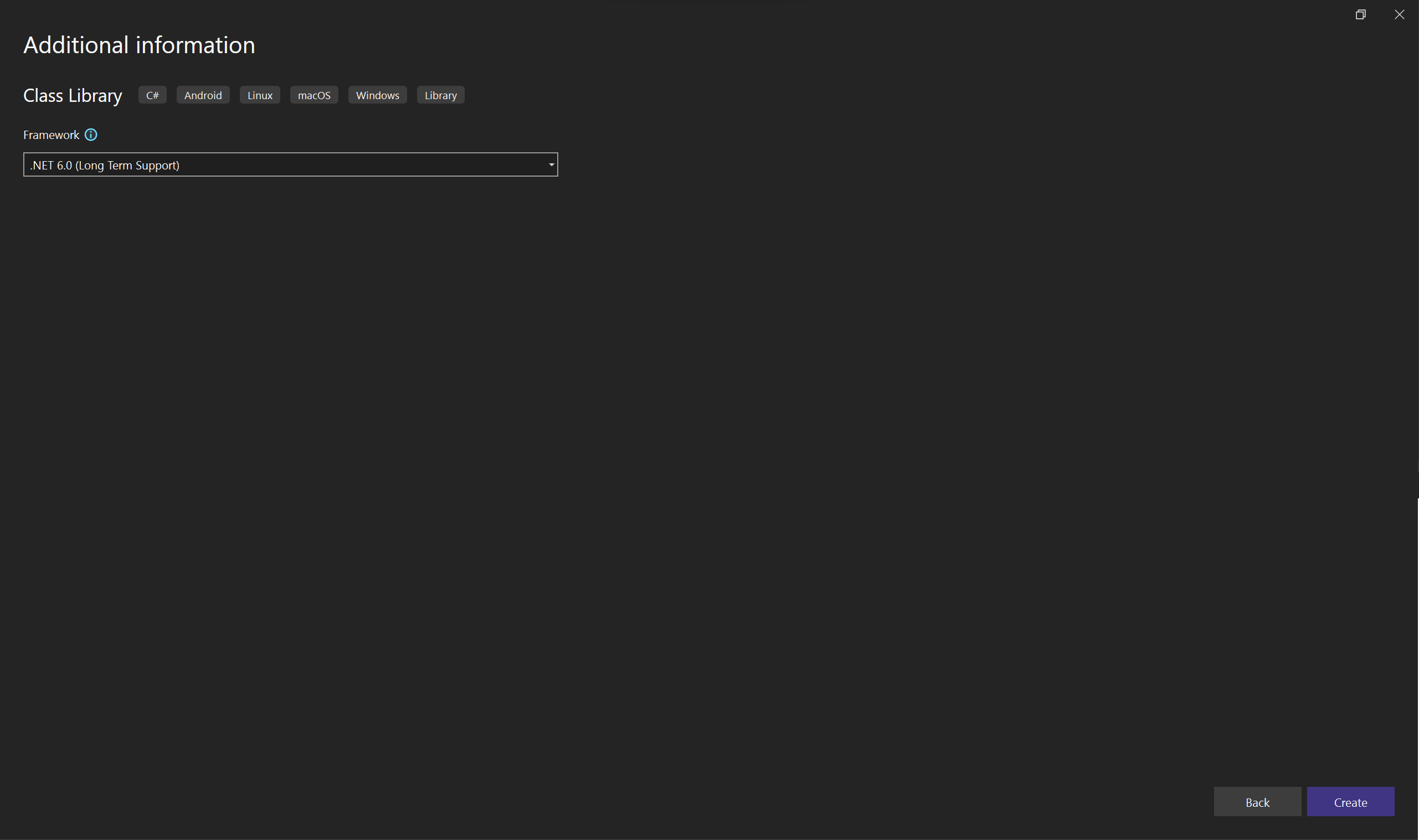


Figure 3 – Select Framework (.Net6)

If getting build errors Install Microsoft.Extensions.DependencyInjection.Abstractions using nuget package manager

**Step 3**

Create Class library for Data Access

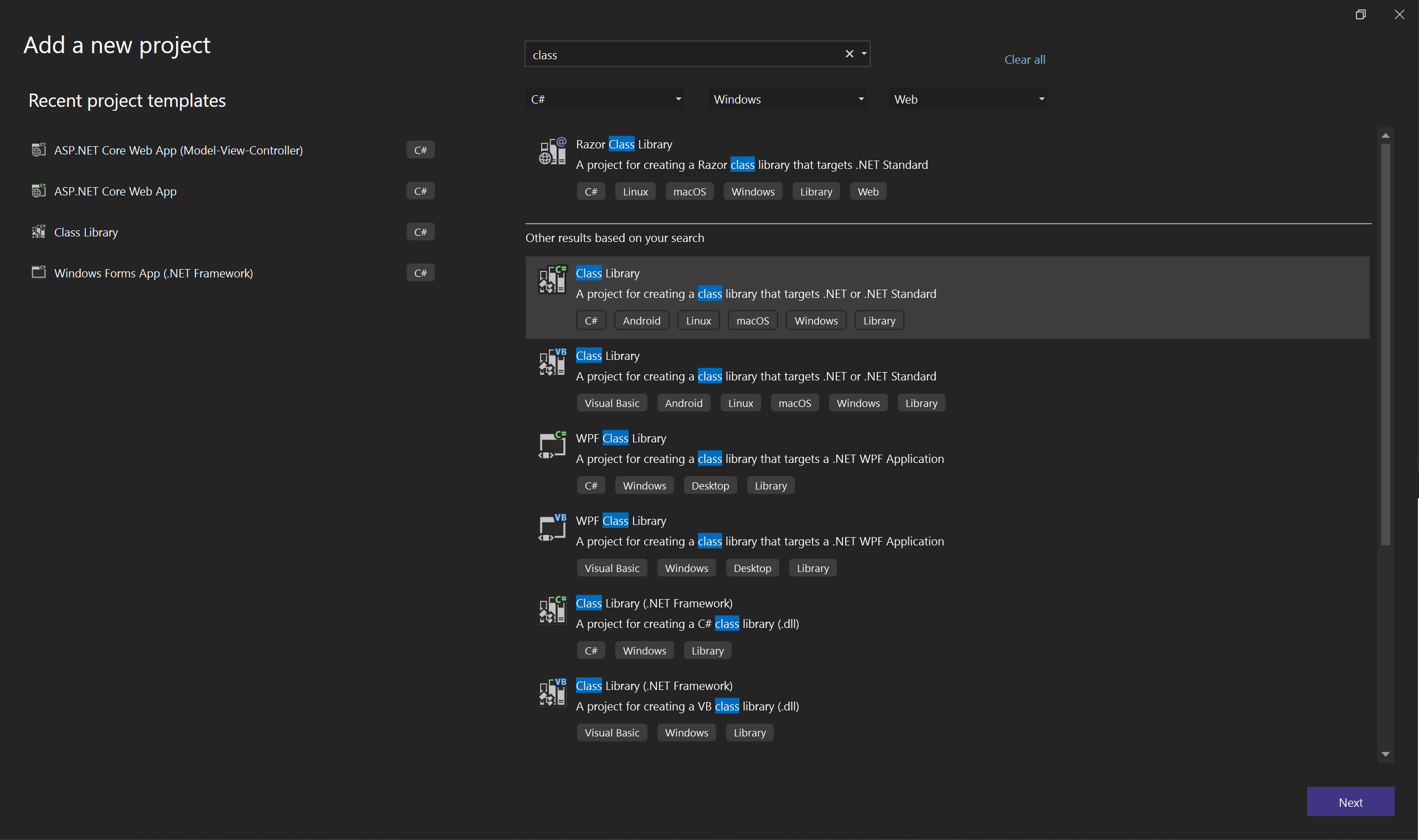


Figure 1 – Select class library from project template list

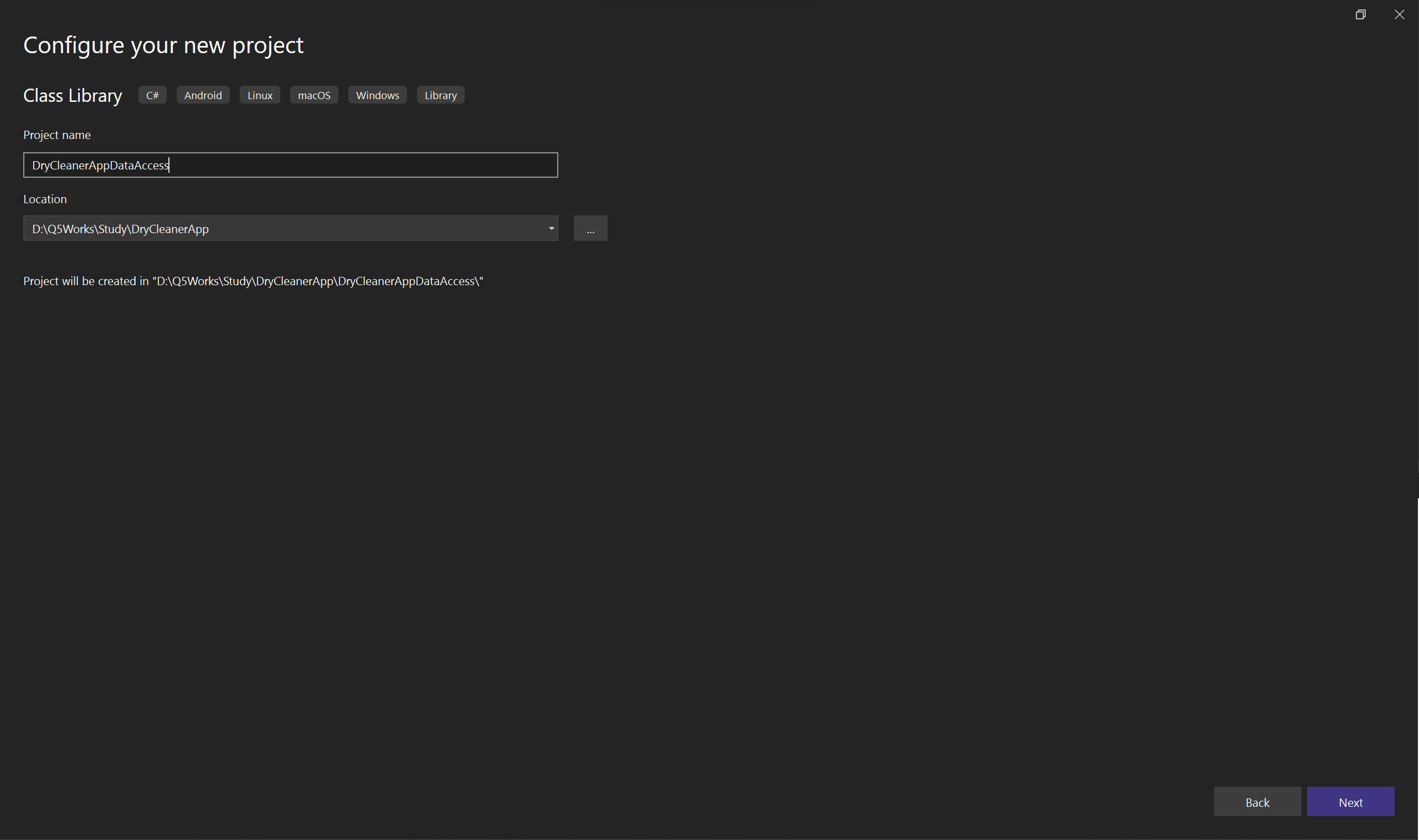


Figure 2 – Giving Proper name and selecting proper Folder for class library

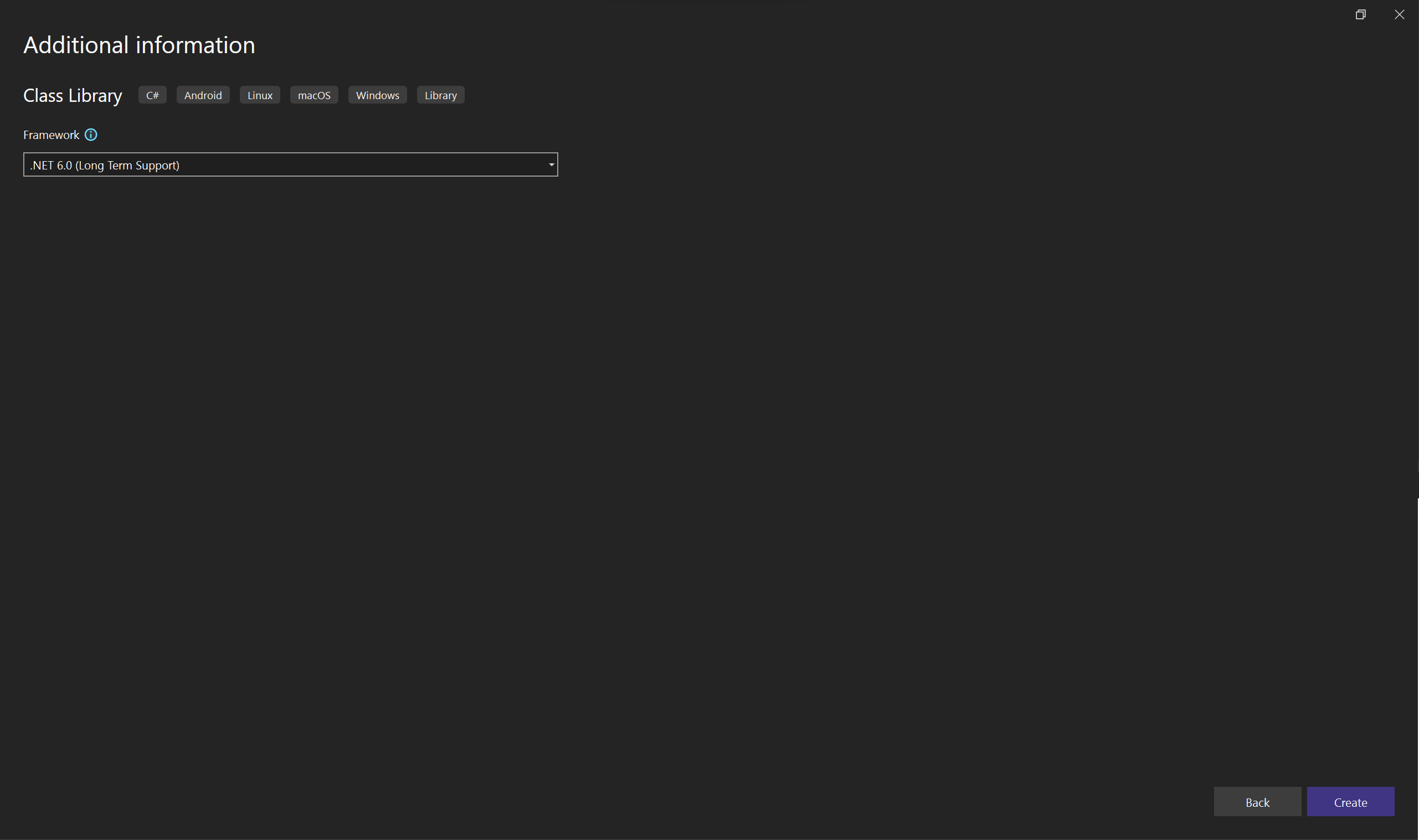
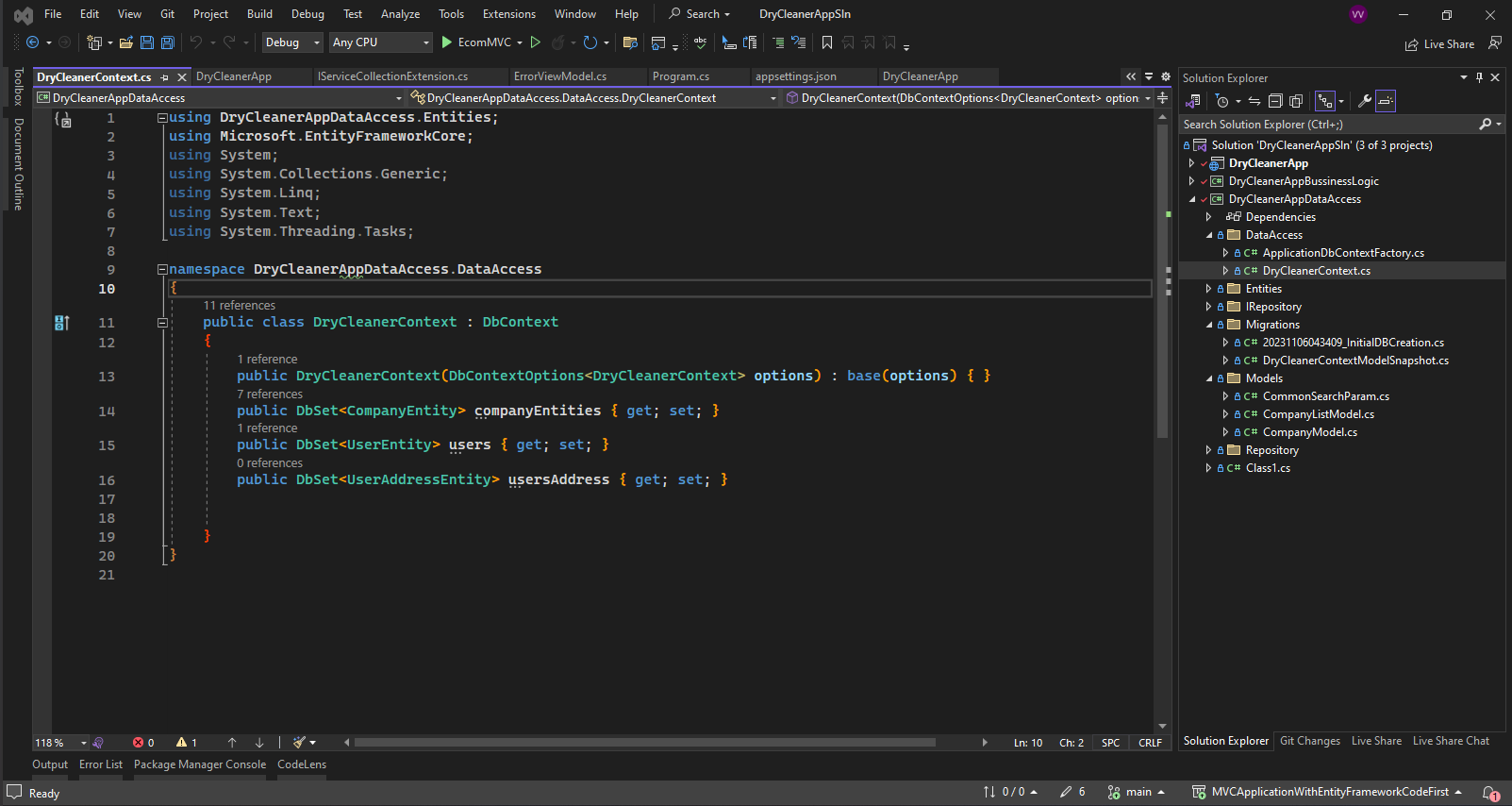


Figure 3 – Select Framework (.Net6)

If getting build errors Install Microsoft.EntityFrameworkCore ,Pomelo.EntityFrameworkCore.MySql, Microsoft.EntityFrameworkCore.Relational, on the dataccess layer using nuget package manager

**After creating projects we can start with creating Db Context**

For that create a new class named **DryCleanerContext** in new folder DataAccess



* Add Constrictor for DryCleanerContext

public DryCleanerContext(DbContextOptions<DryCleanerContext> options) : base(options) { }

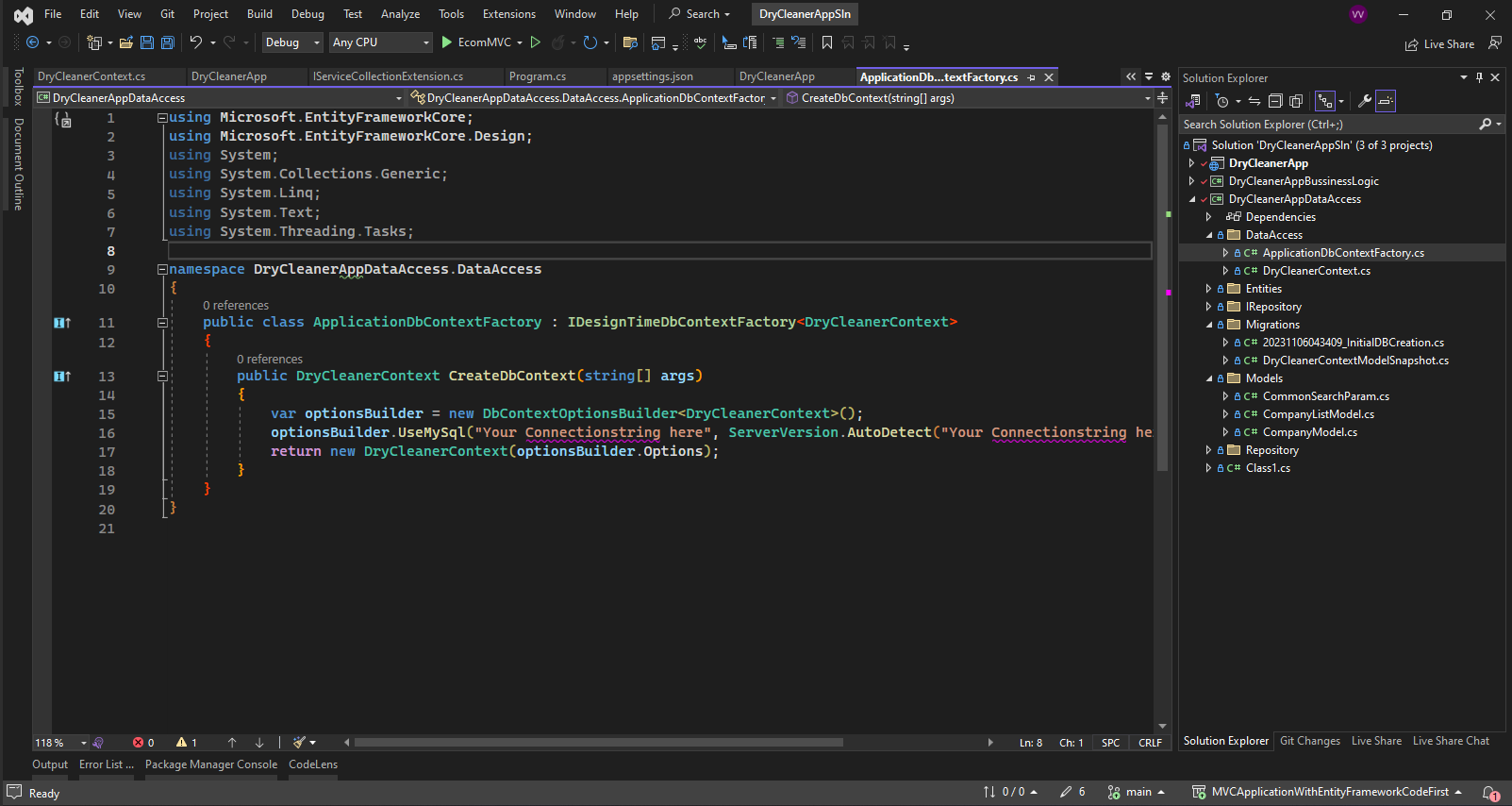
* Then add Dbset to create Table on the database

public DbSet<CompanyEntity> companyEntities { get; set; }

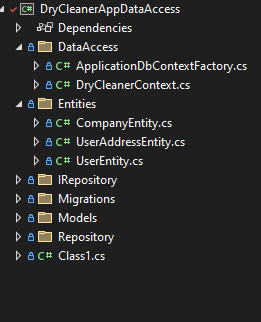
public DbSet<UserEntity> users { get; set; }

public DbSet<UserAddressEntity> usersAddress { get; set; }

Create new class named ApplicationDbContextFactory



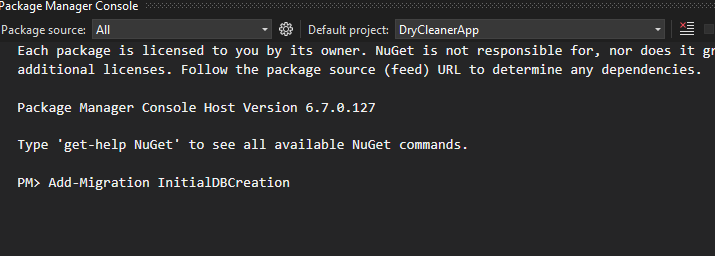
If you are trying to create scaffolded item or controller with read/write actions you need to this create the above class in DataAccess Folder.



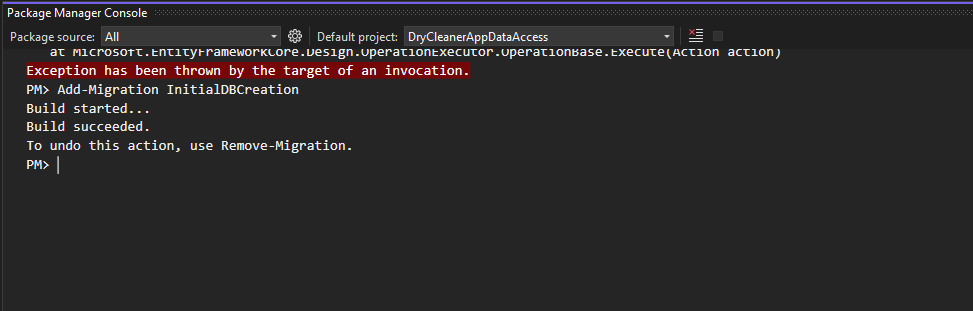
Add Entity classes to on entities folder as shown in figure.These classes in entity folder is used in DryCleanerContext class.

After creating Entity classes and dbcontext class we need to run some codes in Package Manager Console to effect the changes on the db context and entities in Database.

Step 1 – Run command “Add-Migration Nameforthemigration”

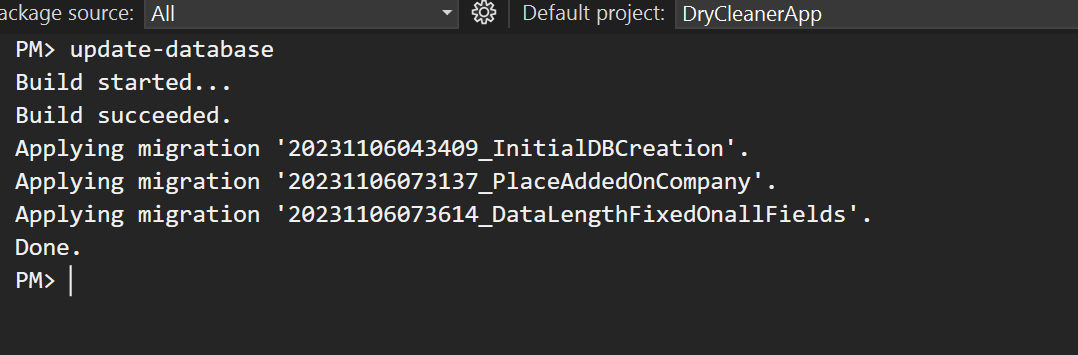


It will get the below image if success

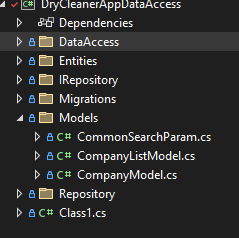


If getting error please check for the default project on the package manager console.Or check the connection string given are correct on both **ApplicationDbContextFactory** and  **appsettings.json** file.

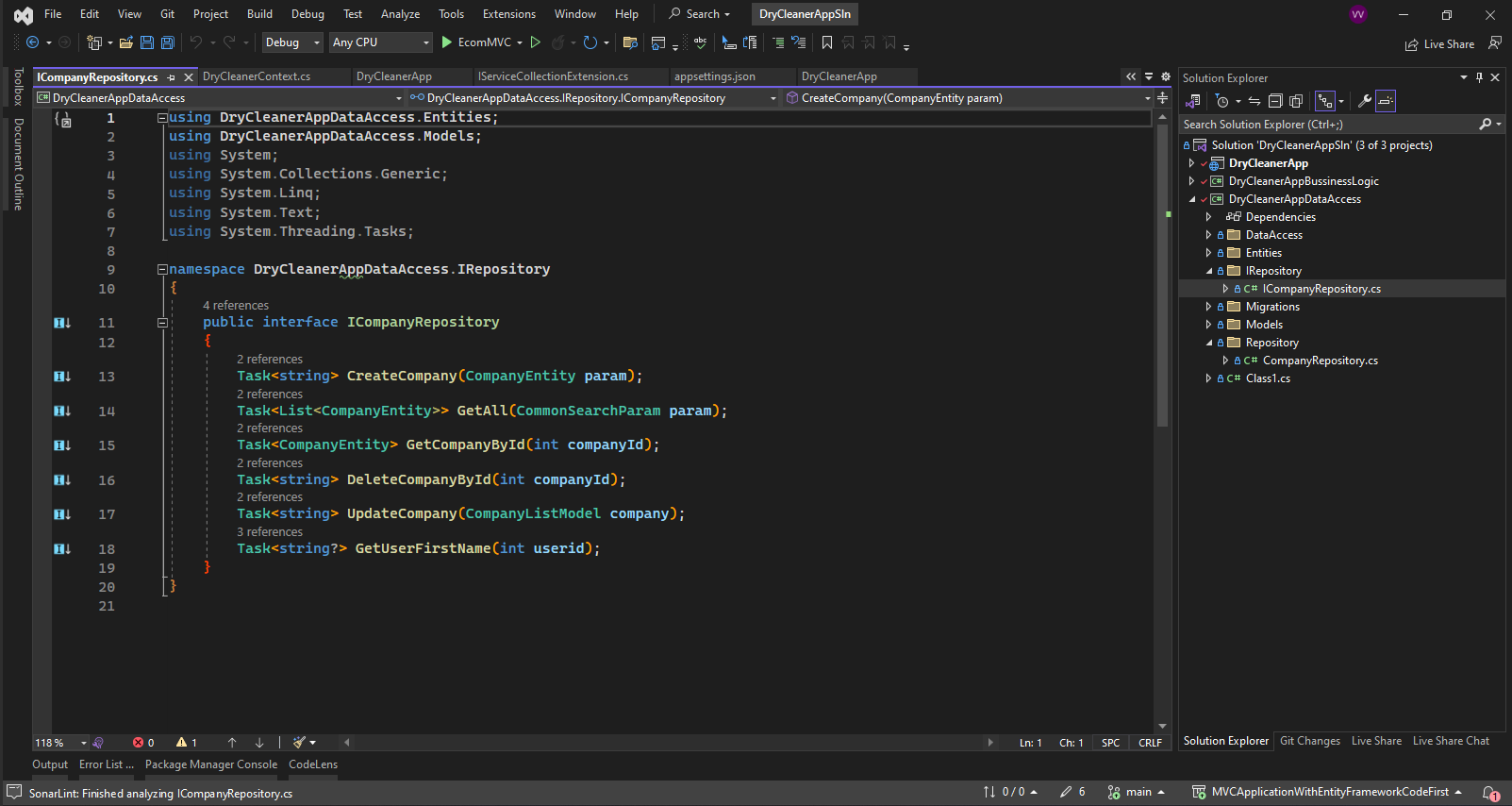
Step 2 – Run Command “database-update”



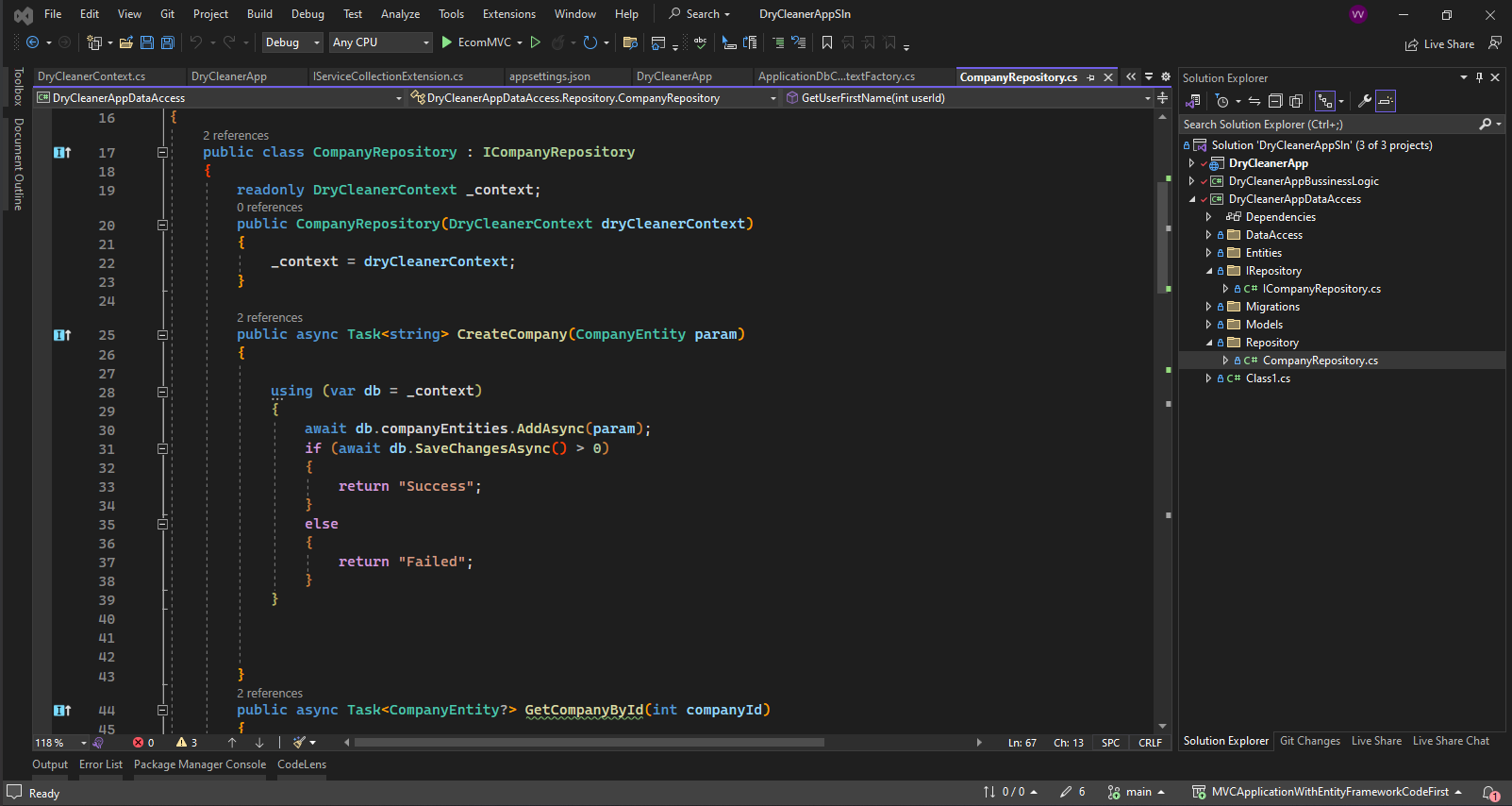
You will get message the the migration created effected to database. Here showing 3 migration as there already created 2 migration before doing this explanation.



You can create new classes for using in application controllers in Model folder

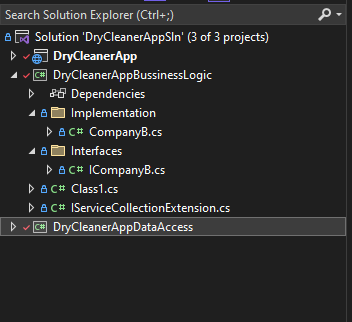


Then Create Interface for Repository files to access from bussinesslogic layer ICompanyrepositry.

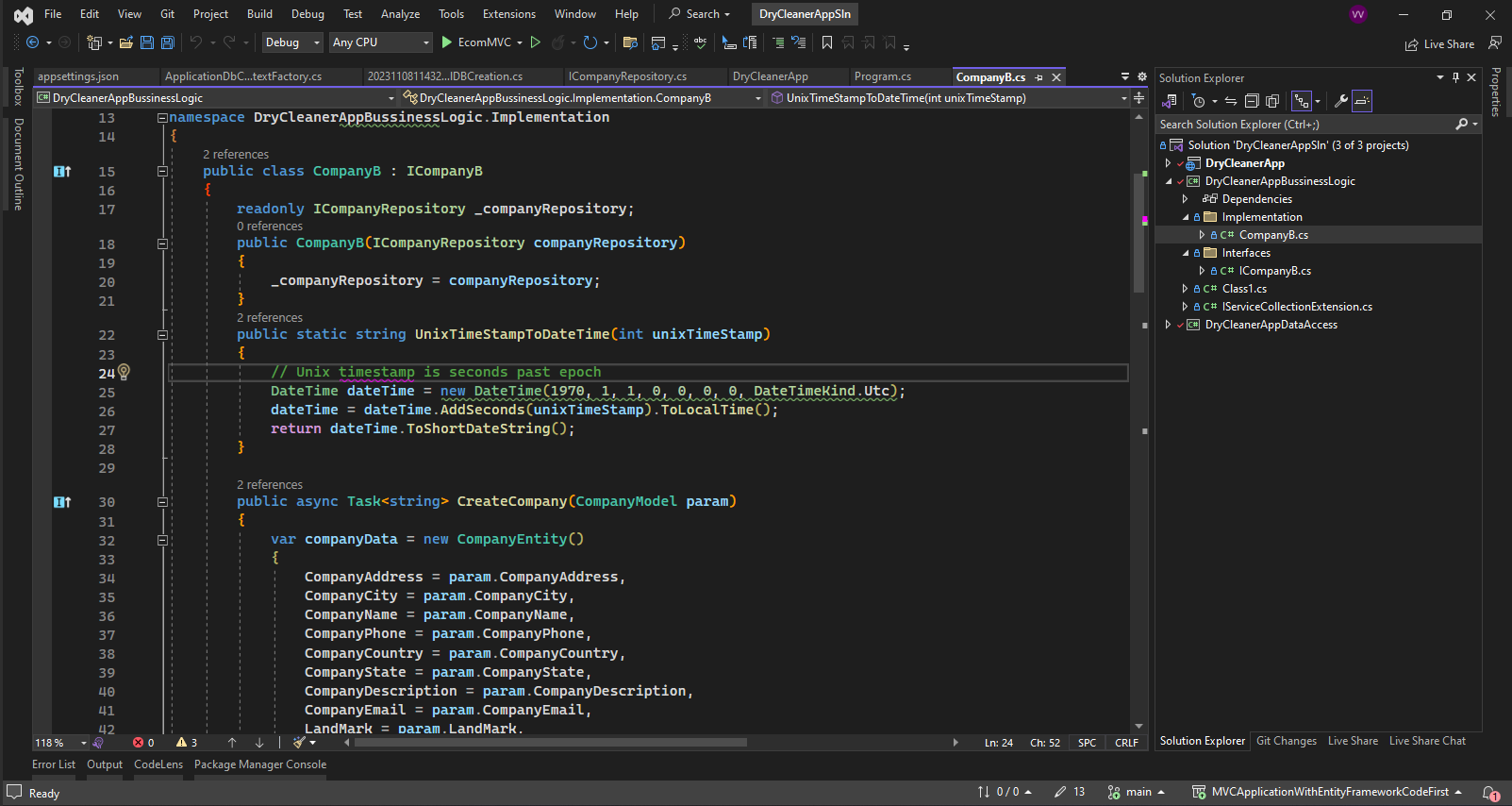


Then Create Repository files to access data from database using Entity Framework named Companyrepositry.

Then We need to create buissiness classes .



Create business files (Interface and its implementation) as show in the above figure.



Business login Implementation class.

Used IRepository interface for getting data from database to the business layer.

The below code is populating entity class from the company model getting from the controller method.

public async Task<string> CreateCompany(CompanyModel param)

{

var companyData = new CompanyEntity()

{

CompanyAddress = param.CompanyAddress,

CompanyCity = param.CompanyCity,

CompanyName = param.CompanyName,

CompanyPhone = param.CompanyPhone,

CompanyCountry = param.CompanyCountry,

CompanyState = param.CompanyState,

CompanyDescription = param.CompanyDescription,

CompanyEmail = param.CompanyEmail,

LandMark = param.LandMark,

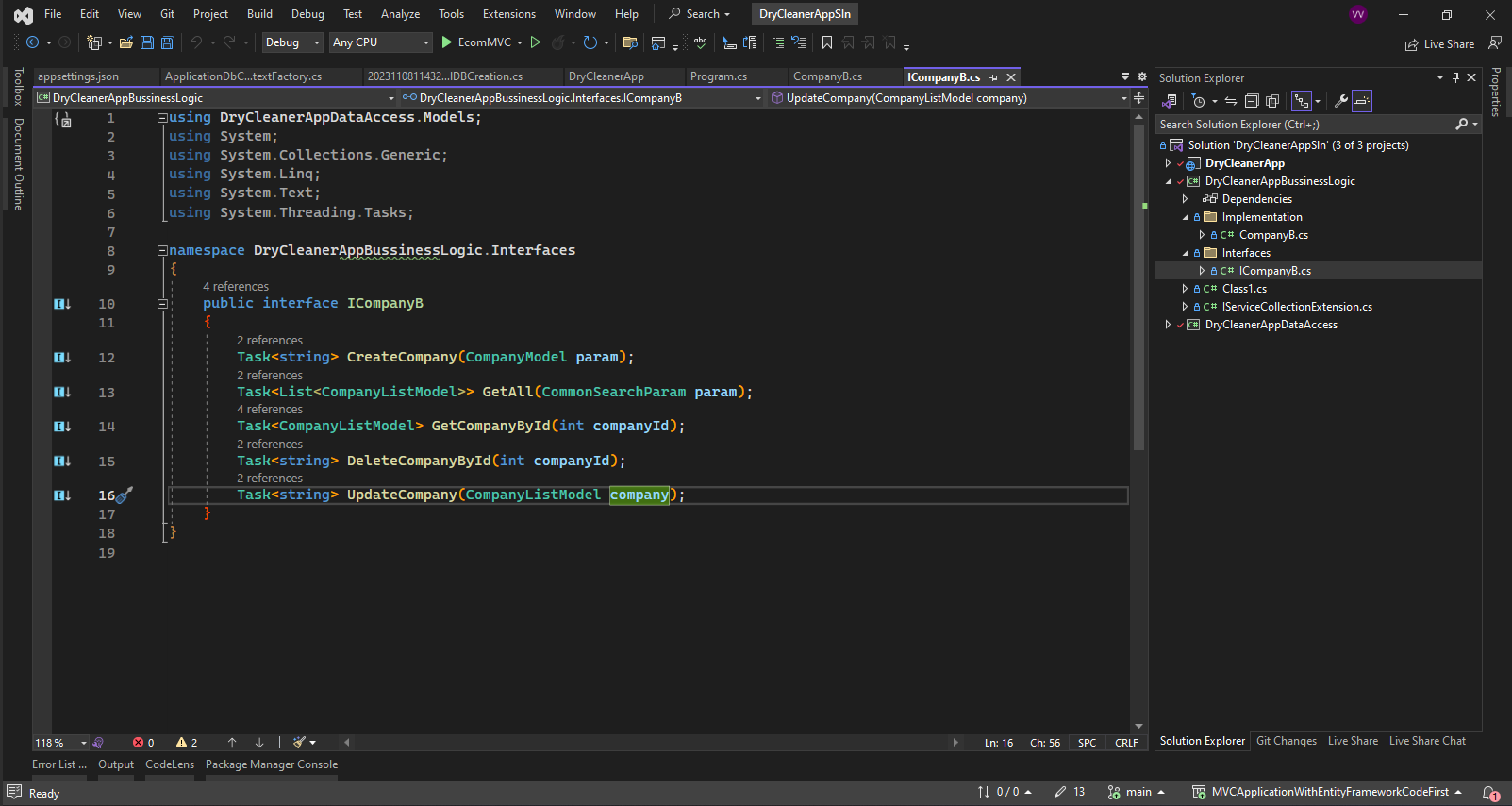
Place = param.Place,

CreatedBy = param.CreatedBy,

};

return await \_companyRepository.CreateCompany(companyData);

}



Interface class for the business logic layer.

Then we can inject repository layer and business layer to our mvc application middle ware for running the application.

Code in Program.cs

using DryCleanerAppBussinessLogic;

using DryCleanerAppBussinessLogic.Implementation;

using DryCleanerAppBussinessLogic.Interfaces;

using DryCleanerAppDataAccess.IRepository;

using DryCleanerAppDataAccess.Repository;

using DryCleanerAppDataAccess.DataAccess;

using Microsoft.Data.SqlClient;

using Microsoft.EntityFrameworkCore;

using MySql.Data.MySqlClient;

using System.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

//getting connection string from appsettings.json

string? dbConnectionString = builder.Configuration.GetConnectionString("AppConnection");

//adding DB Context for mysql connection

builder.Services.AddDbContext<DryCleanerContext>(options =>

options.UseMySql(dbConnectionString, ServerVersion.AutoDetect(dbConnectionString))

);

//Adding Repository for dependency injection

builder.Services.AddRepositoryDependecies();

//Adding Buissiness logic files

builder.Services.AddScoped<ICompanyB, CompanyB>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=CompanyMaster}/{action=CompanyList}/{id?}");

app.Run();