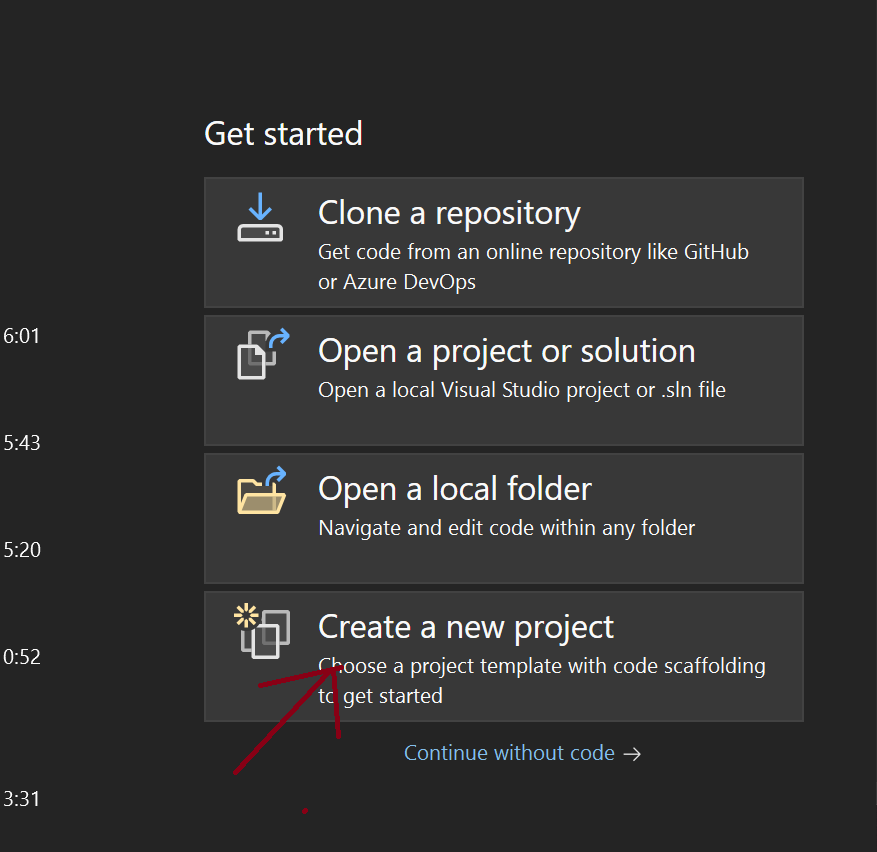
**Step 1**

Create new webapi project from visual studio



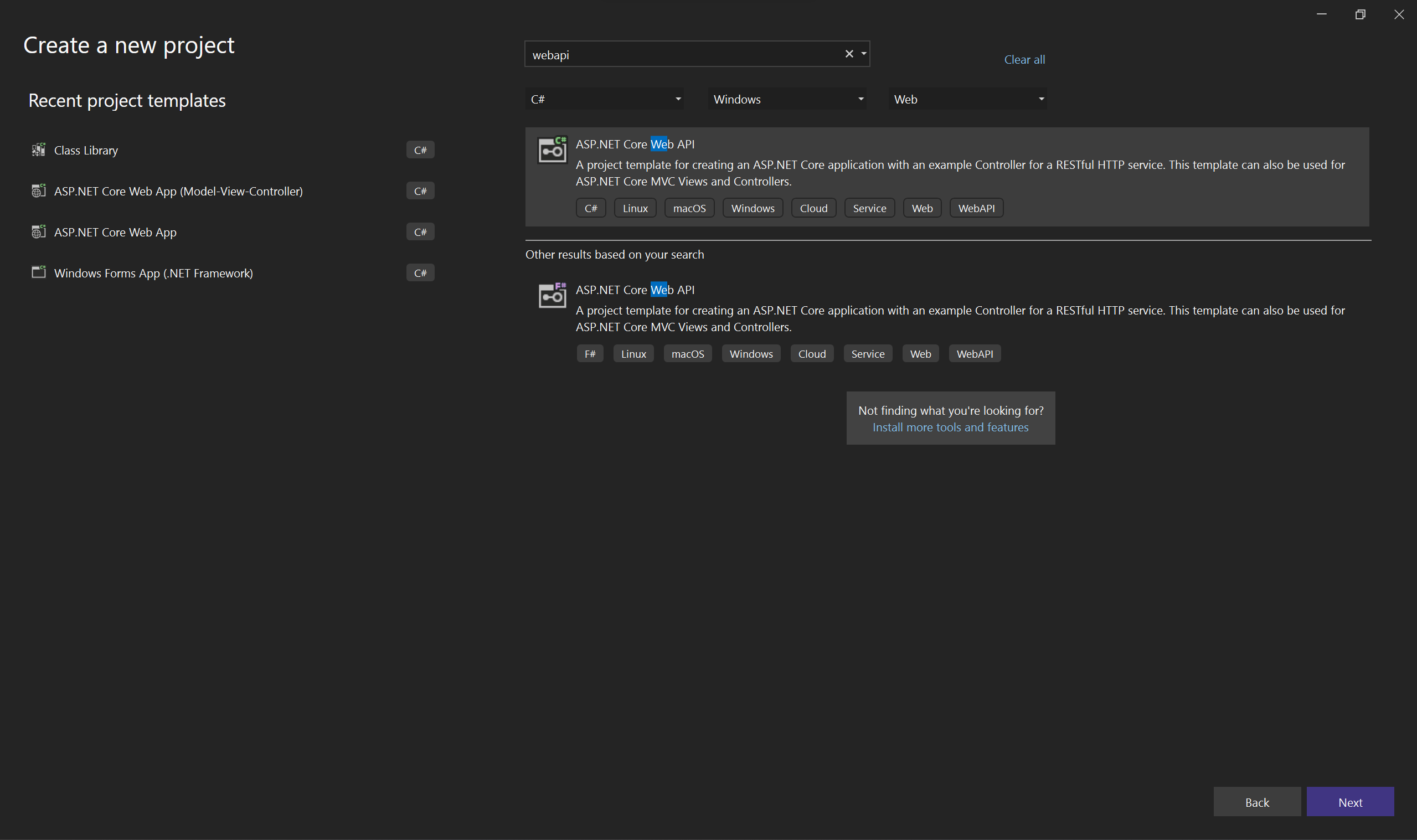


Figure 2 – Select Web API Template

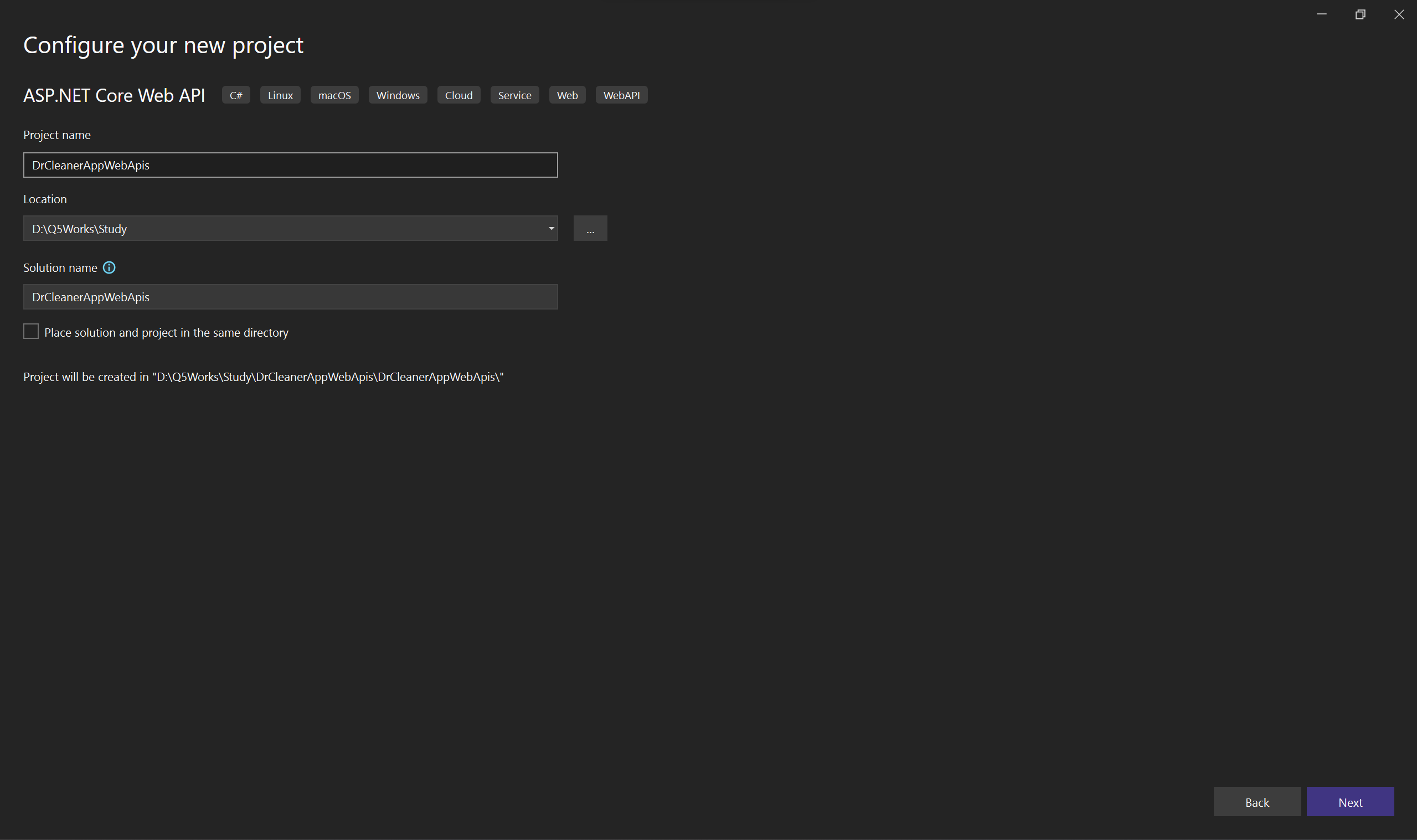


Figure 3 – Give project name and select location for saving project files

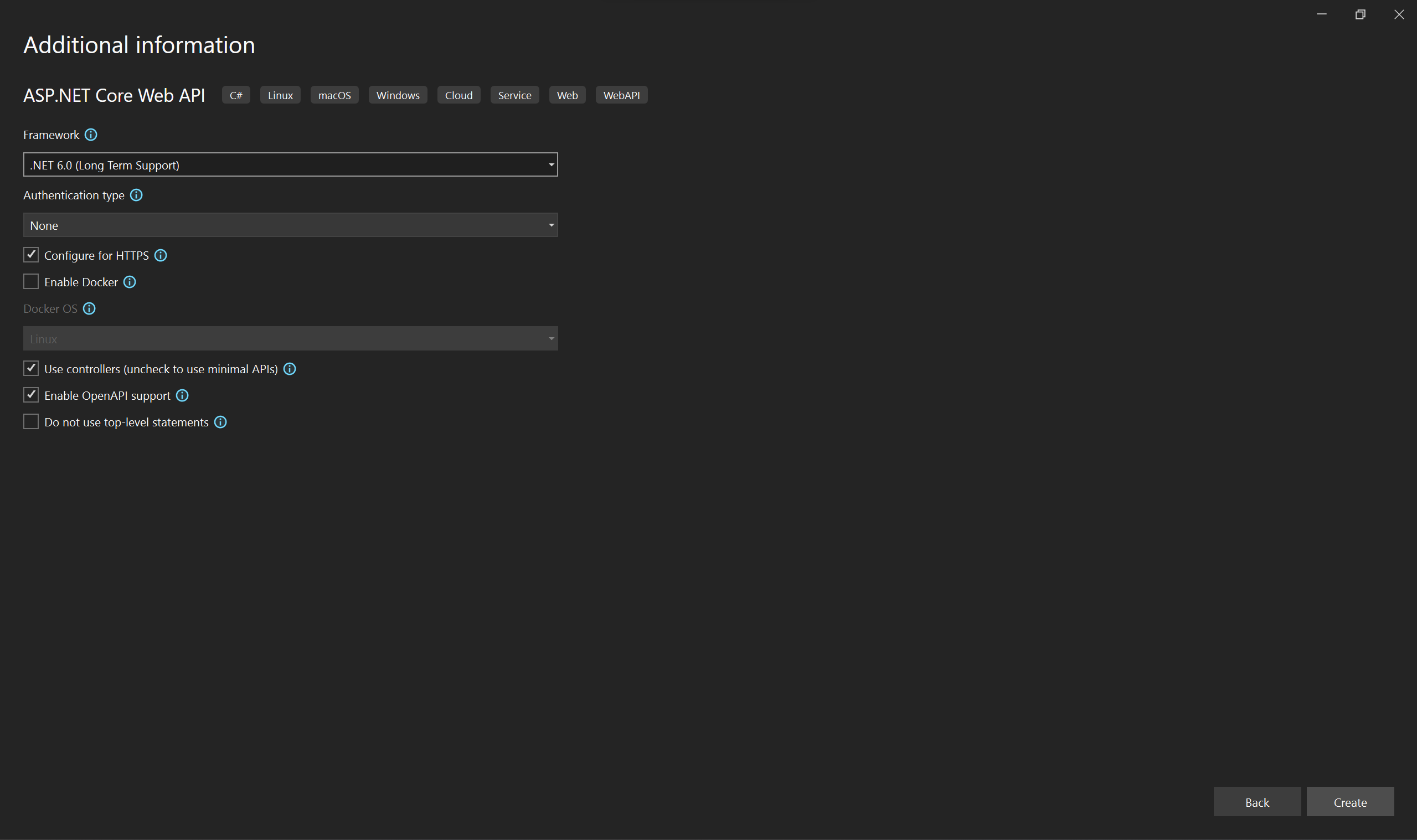


Figure 4 – Select Faremwork (.Net6)

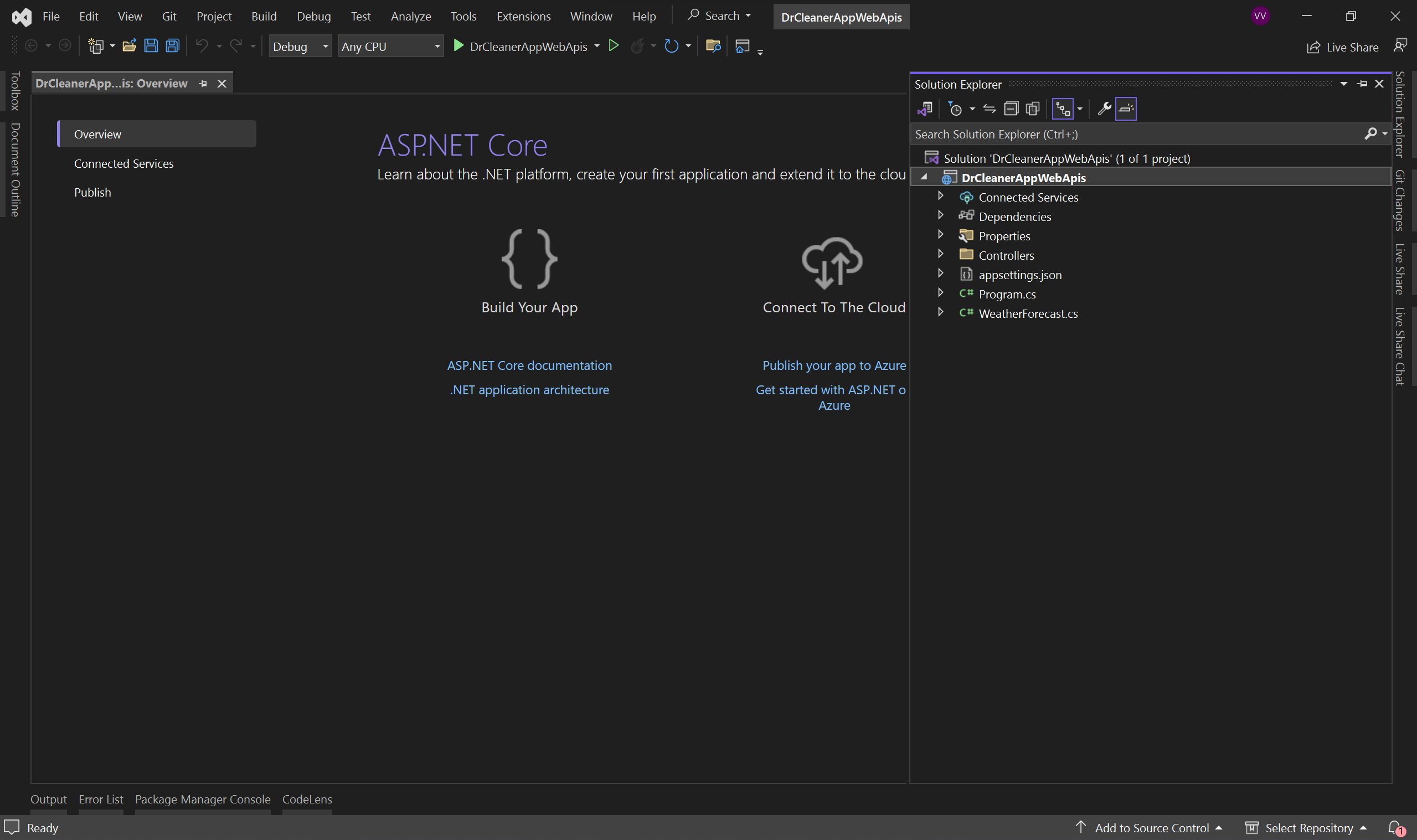


Figure 5 – Project created you can check on the Solution explorer Right Side

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

To enable cross-origin requests on the apis add the below code In the middleware on program.cs file

builder.Services.AddCors(options =>

{

options.AddPolicy("CORSPolicy", builder => builder

.AllowAnyMethod()

.AllowAnyHeader()

.AllowCredentials()

.SetIsOriginAllowed((hosts) => true));

});

Add Connection string details on the program.cs file

// Read the connection string from appsettings.

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

builder.Services.AddTransient<IDbConnection>(sql => new MySqlConnection(dbConnectionString));

Here we need to add the Mysql.Data in to the application to resolve the error.

Add Class Library for Business logic Layer

**Step 2**

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

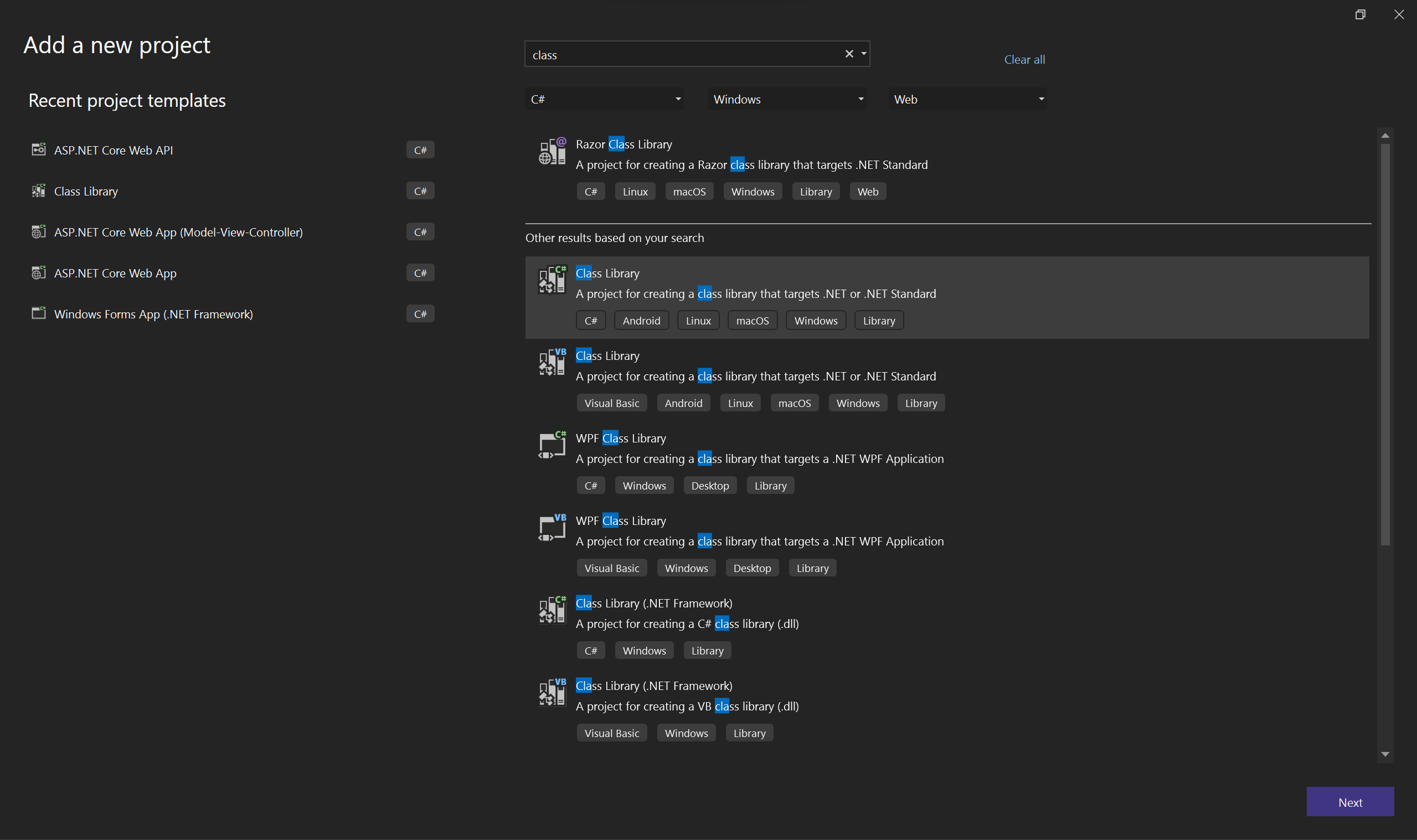


Figure 1 – Select class library from project template list

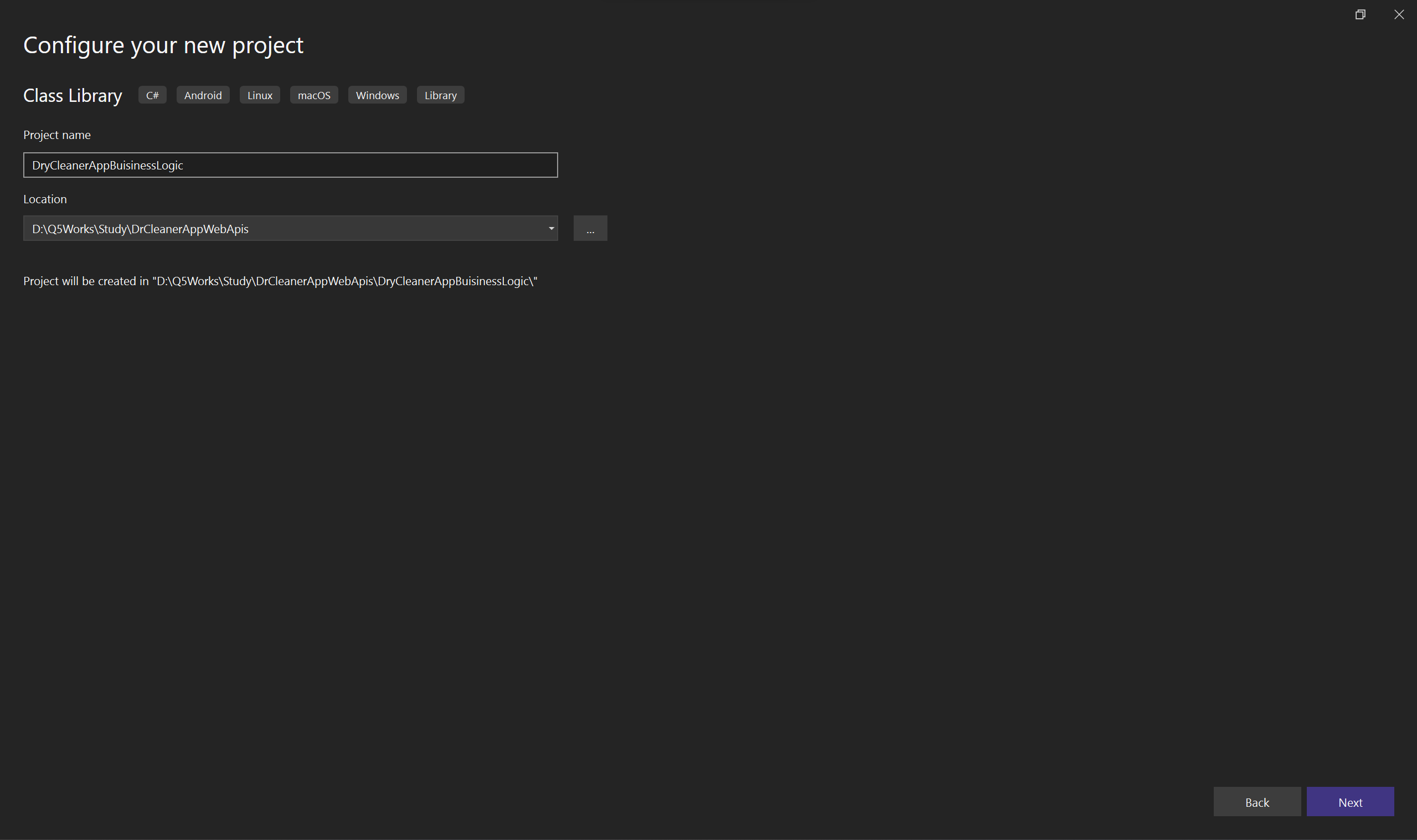


Figure 2 – Giving Proper name and selecting proper Folder for class library

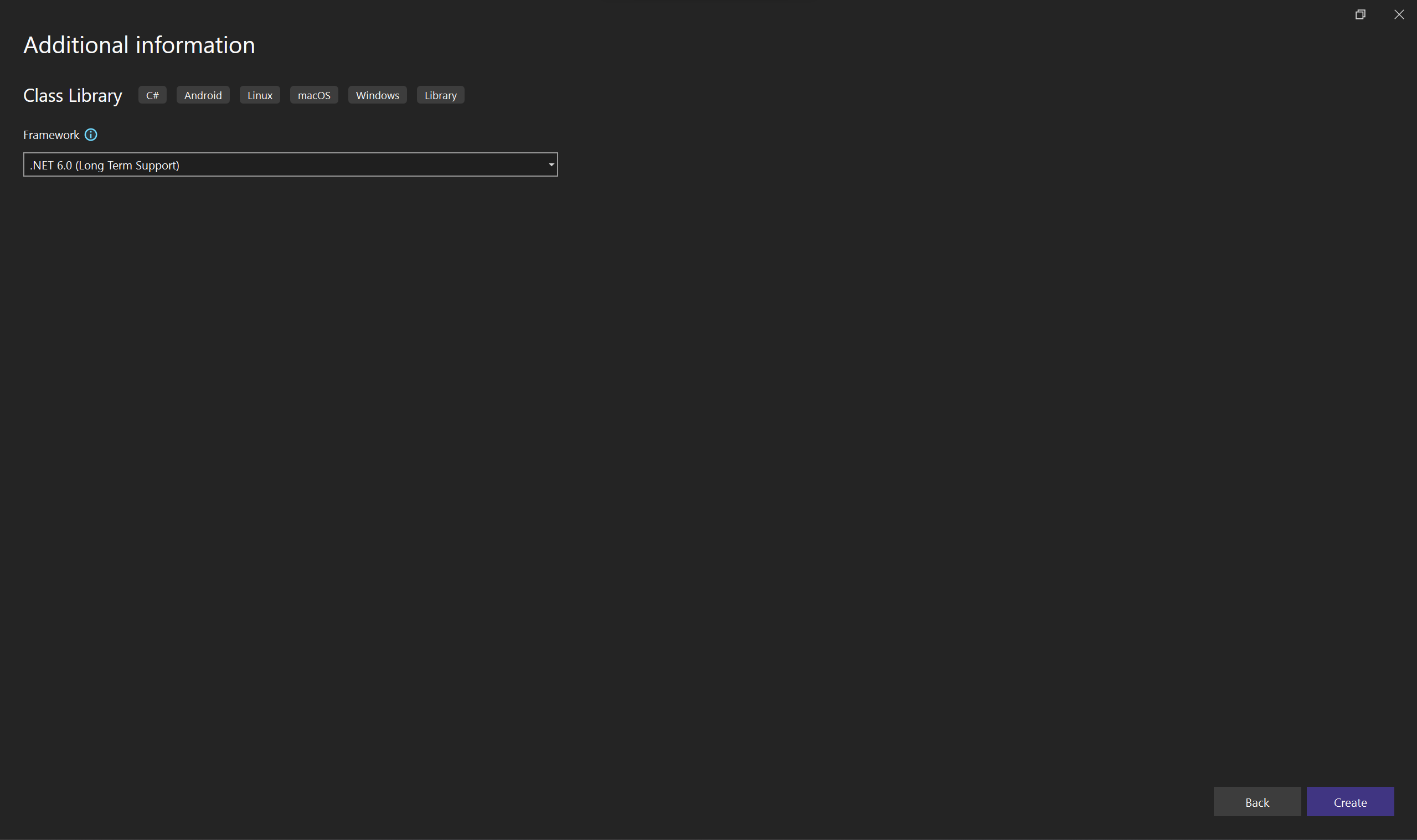
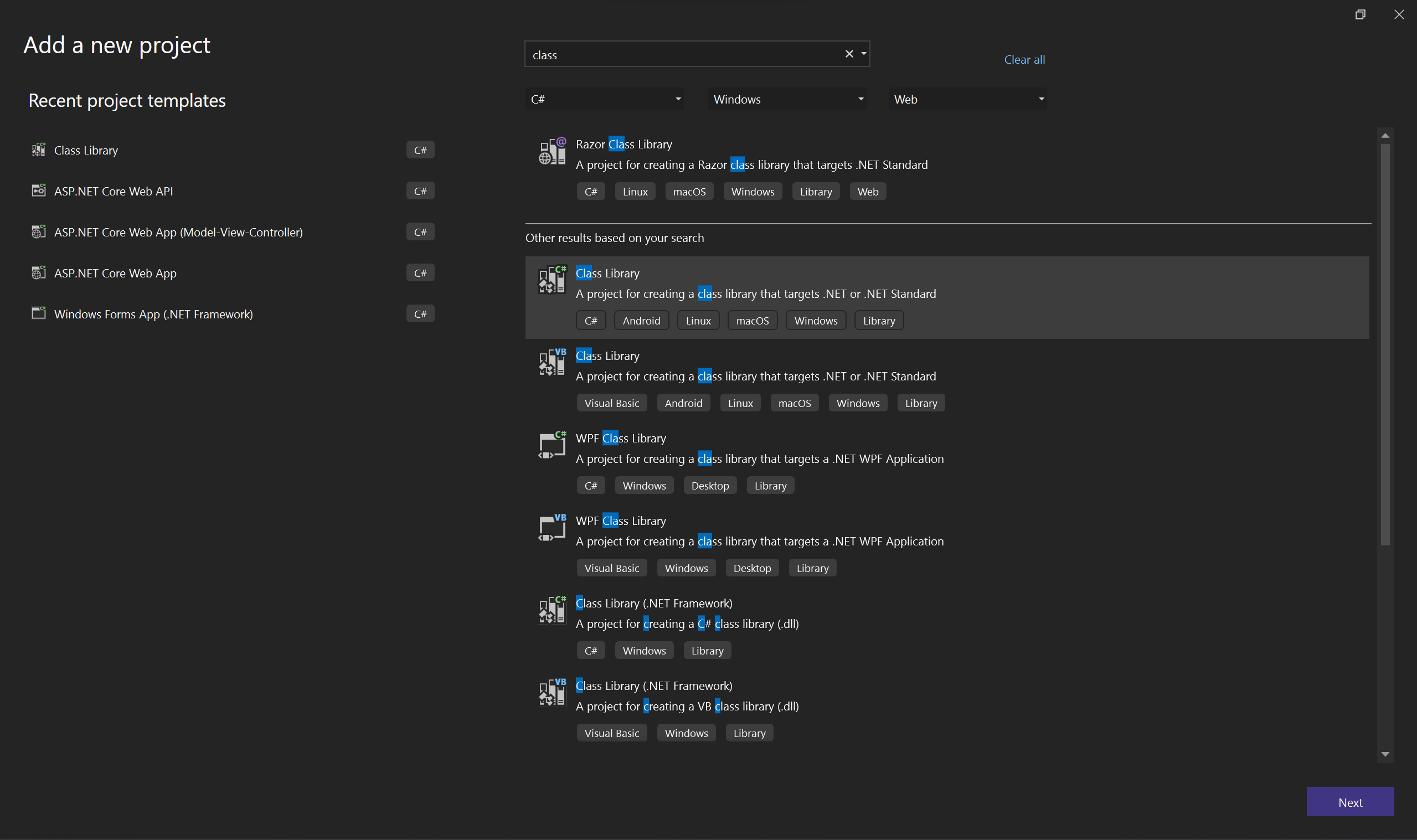


Figure 3 – Select Framework (.Net6)

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

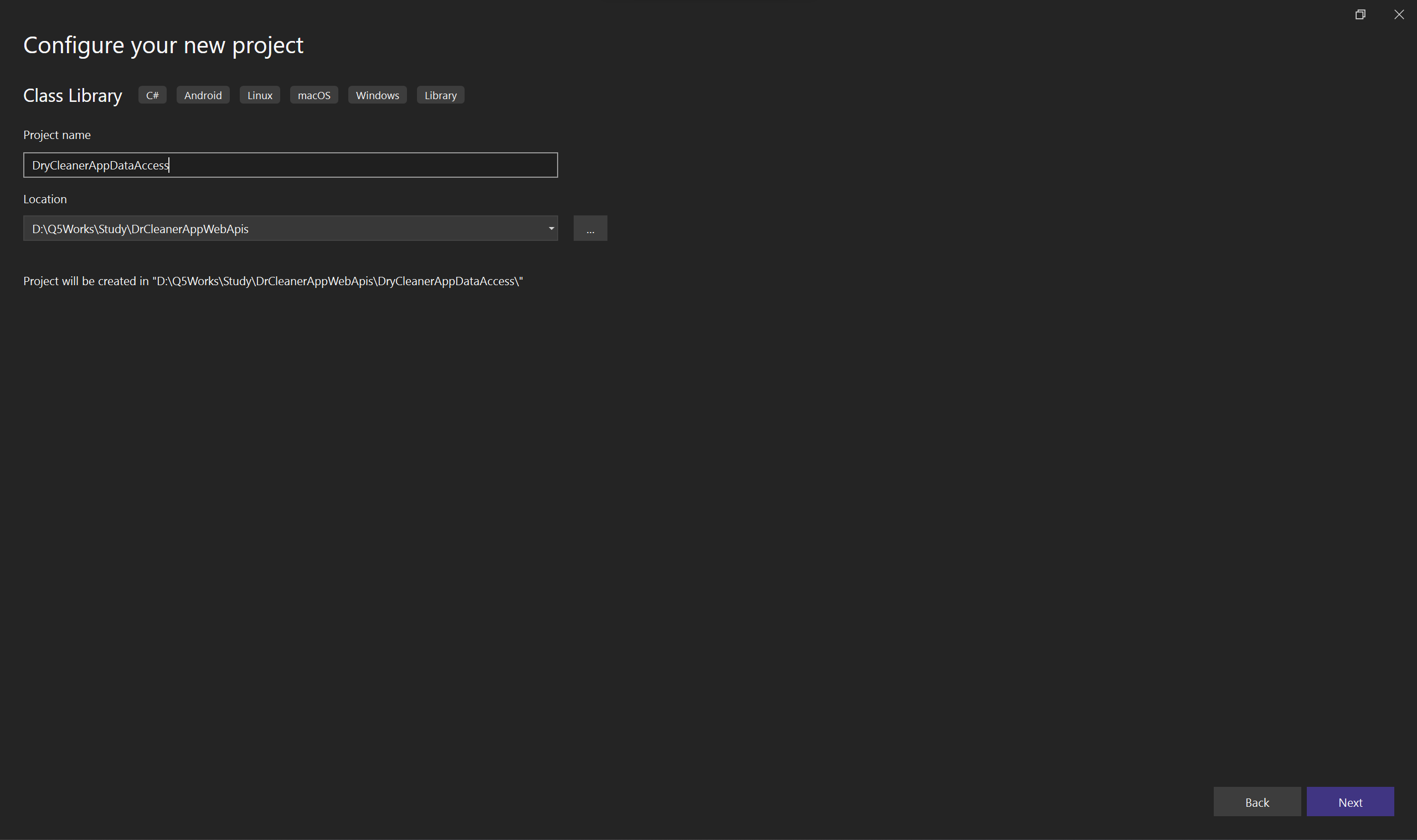


Figure 2 – Giving Proper name and selecting proper Folder for class library

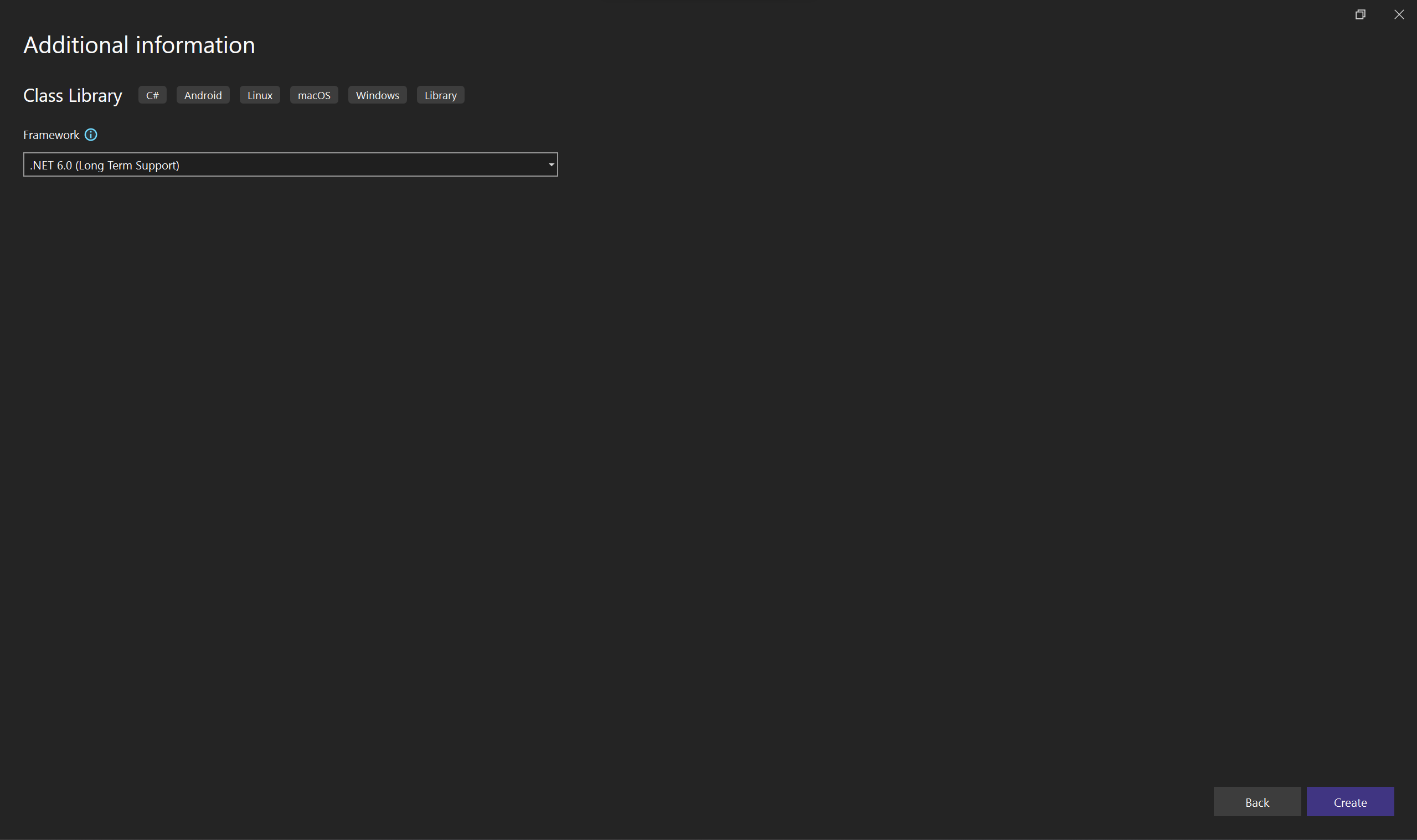
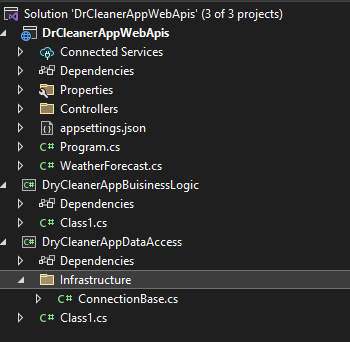


Figure 3 – Select Framework (.Net6)

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 Connectionstring**

Add New abstract class named ConnectionBase in the folder for database based functionalities using IDbConnection Interface.



using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DryCleanerAppDataAccess.Infrastructure

{

public abstract class ConnectionBase : IDbConnection

{

readonly string injectedConnectionString;

public ConnectionBase(string connectionString)

{

this.injectedConnectionString = connectionString;

}

protected ConnectionBase(IDbConnection connection)

{

Connection = connection;

}

private IDbConnection Connection { get; set; }

// Verbose but necessary implementation of IDbConnection:

#region "IDbConnection implementation"

public string ConnectionString

{

get

{

return Connection.ConnectionString;

}

set

{

Connection.ConnectionString = value;

}

}

public int ConnectionTimeout

{

get

{

return Connection.ConnectionTimeout;

}

}

public string Database

{

get

{

return Connection.Database;

}

}

public ConnectionState State

{

get

{

return Connection.State;

}

}

public IDbTransaction BeginTransaction()

{

return Connection.BeginTransaction();

}

public IDbTransaction BeginTransaction(IsolationLevel isolationLevel)

{

return Connection.BeginTransaction(isolationLevel);

}

public void ChangeDatabase(string databaseName)

{

Connection.ChangeDatabase(databaseName);

}

public void Close()

{

Connection.Close();

}

public IDbCommand CreateCommand()

{

return Connection.CreateCommand();

}

public void Dispose()

{

Connection.Dispose();

}

public void Open()

{

Connection.Open();

}

#endregion

protected IDbConnection GetConnection(string connectionString)

{

if (String.IsNullOrEmpty(connectionString))

throw new ArgumentNullException("conneciton string is null or empty");

return new MySqlConnection(connectionString);

}

/// <summary>

///

/// </summary>

/// <returns></returns>

protected IDbConnection GetConnection()

{

if (Connection.State == ConnectionState.Closed)

{

return GetConnection(Connection.ConnectionString);

}

else

{

if (Connection == null)

throw new ArgumentNullException("conneciton unable to inject");

return Connection;

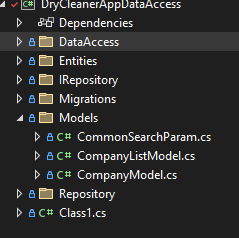
}

}

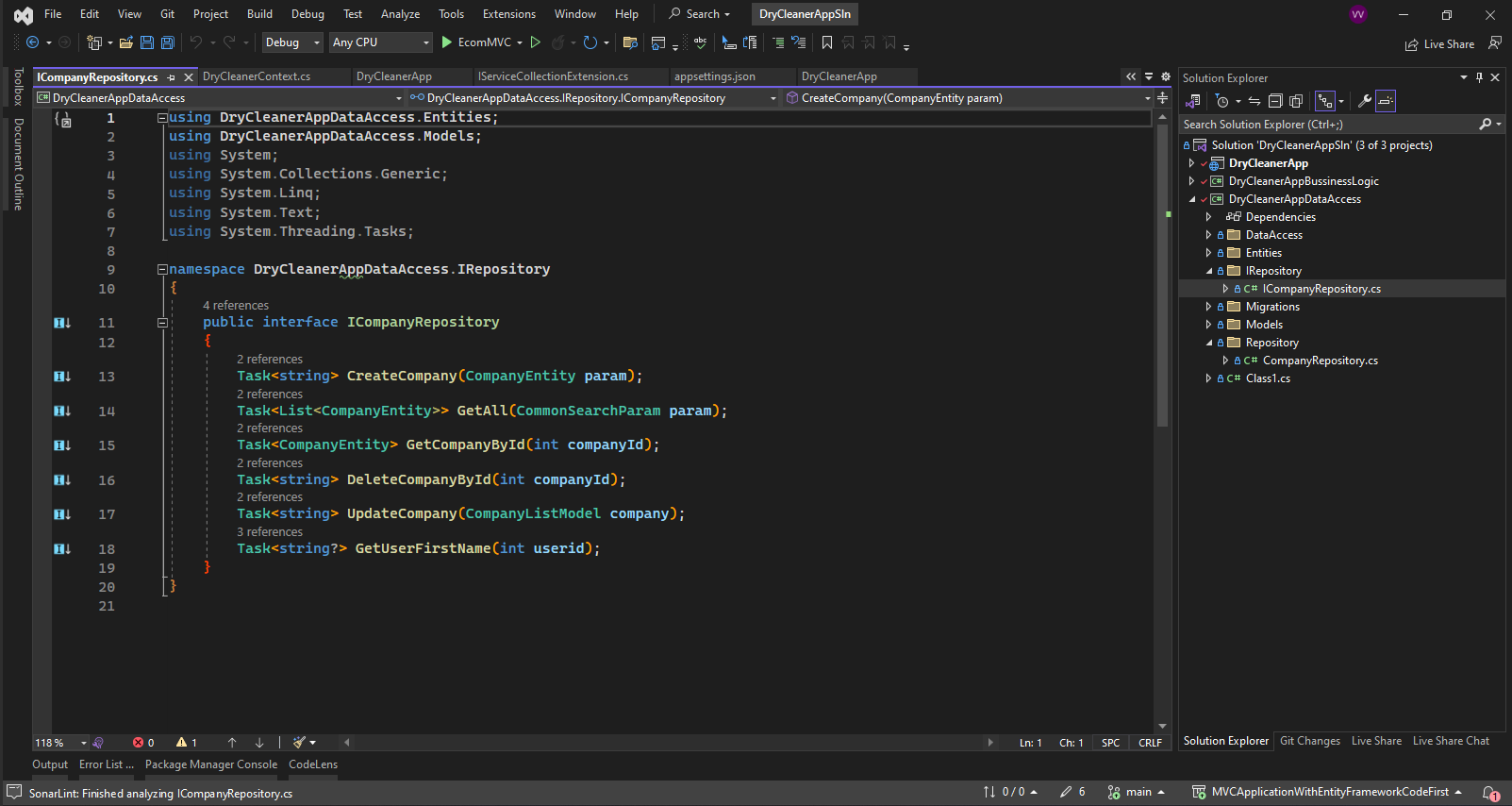
}

}

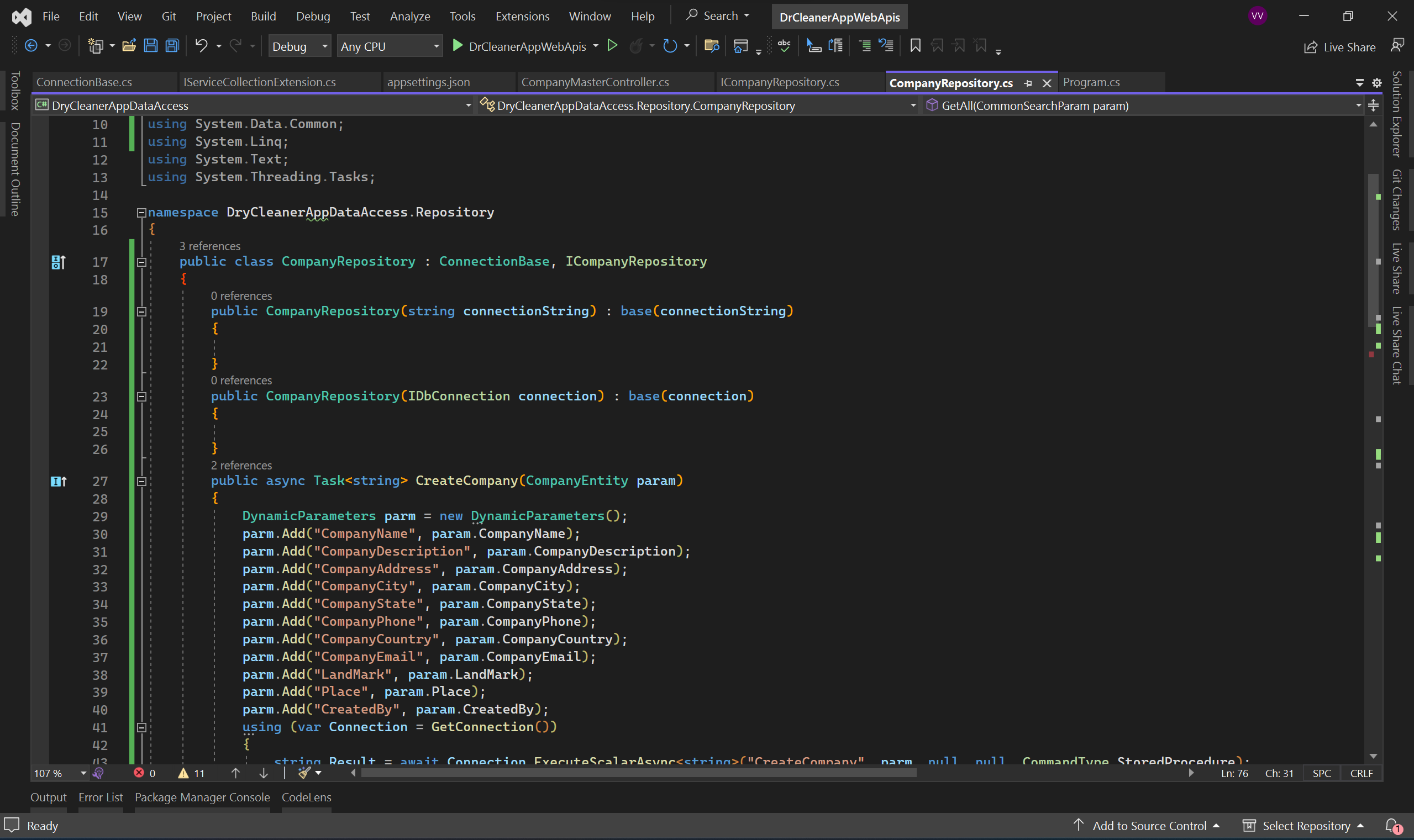
Here we need to add the Mysql.Data in to the application to resolve the error.



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

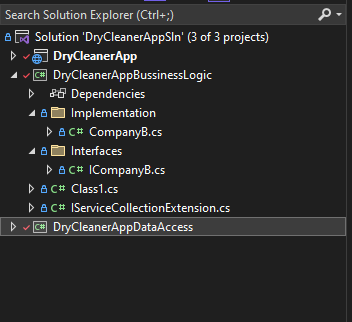


Then Create Interface for Repository files to access from bussinesslogic layer ICompanyrepositry.

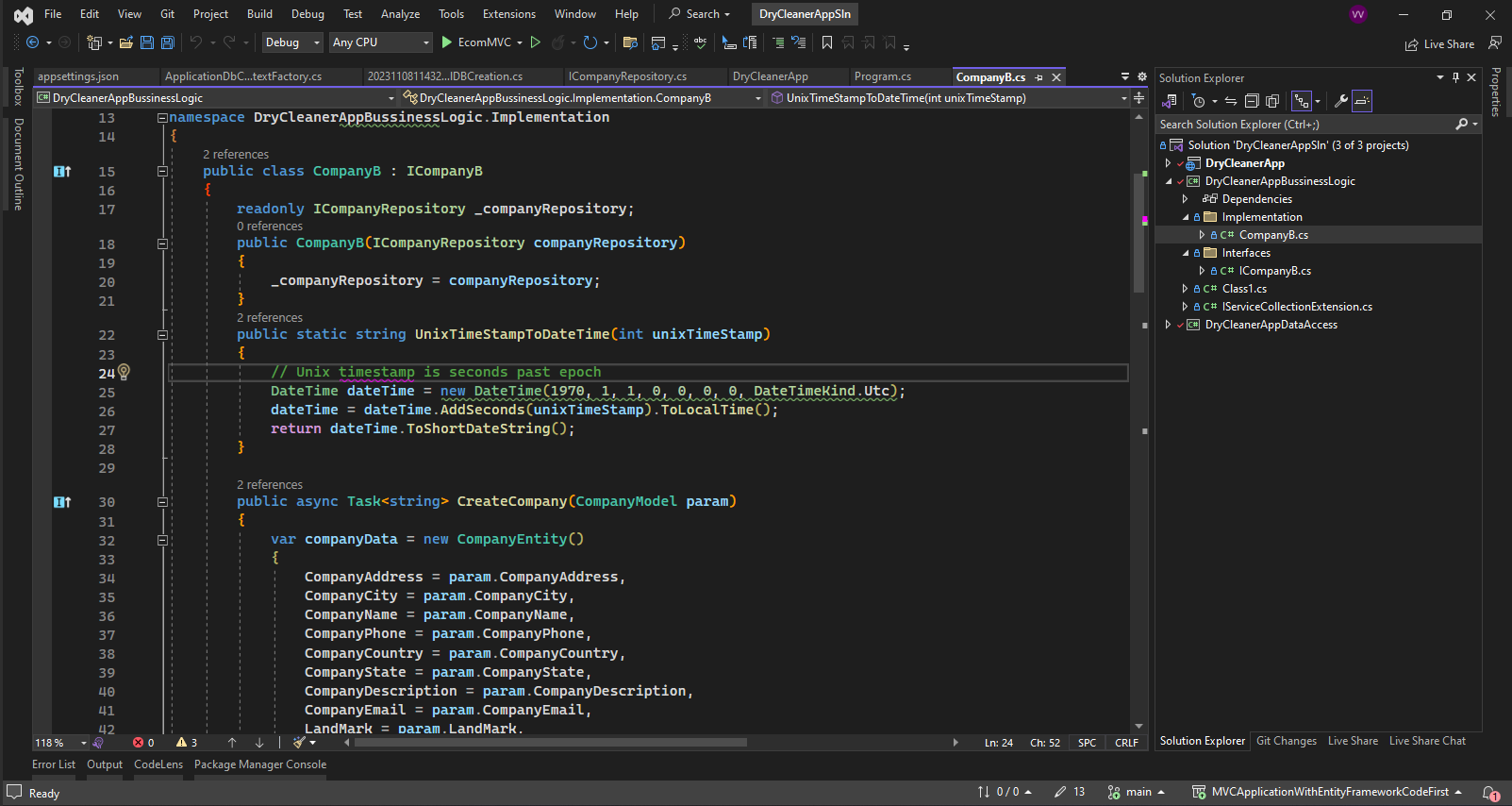


Then Create Repository files to access data from database using Entity Framework named Companyrepositry.

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,

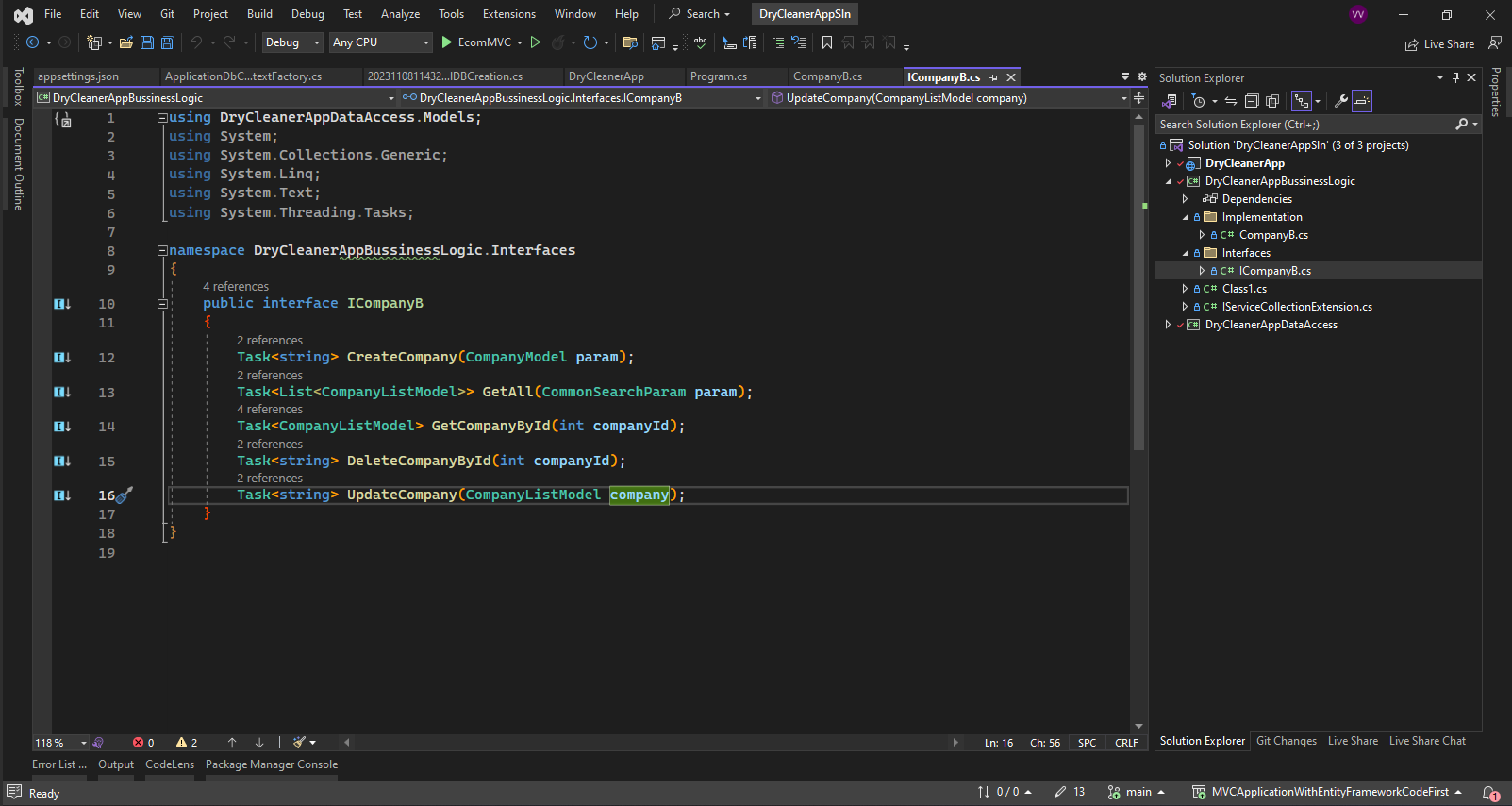
Place = param.Place,

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 api application middle ware for running the application.

Code in Program.cs

using DryCleanerAppBuisinessLogic;

using DryCleanerAppBuisinessLogic.Implementation;

using DryCleanerAppBuisinessLogic.Interfaces;

using MySql.Data.MySqlClient;

using System.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

//To allow cross-origin request.

builder.Services.AddCors(options =>

{

options.AddPolicy("CORSPolicy", builder => builder

.AllowAnyMethod()

.AllowAnyHeader()

.AllowCredentials()

.SetIsOriginAllowed((hosts) => true));

});

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Read the connection string from appsettings.

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

builder.Services.AddTransient<IDbConnection>(sql => new MySqlConnection(dbConnectionString));

builder.Services.AddScoped<ICompanyB, CompanyB>(); // Custom Buisiness layer injection

builder.Services.AddRepositoryDependencies(); //Repository Layer Injection

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();