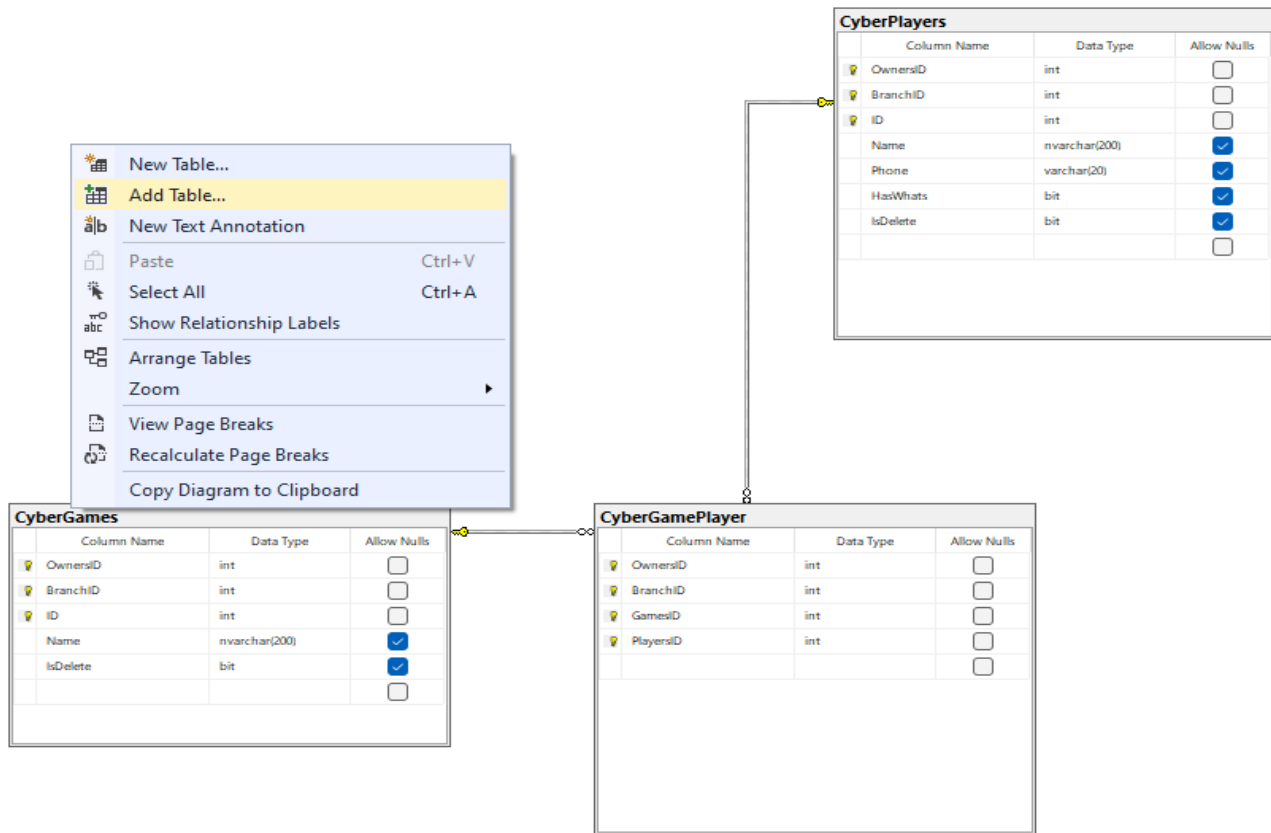


C# API By SWE Sharaf El Dean

Step One the Database

For this we will be using the MS SQL SERVER

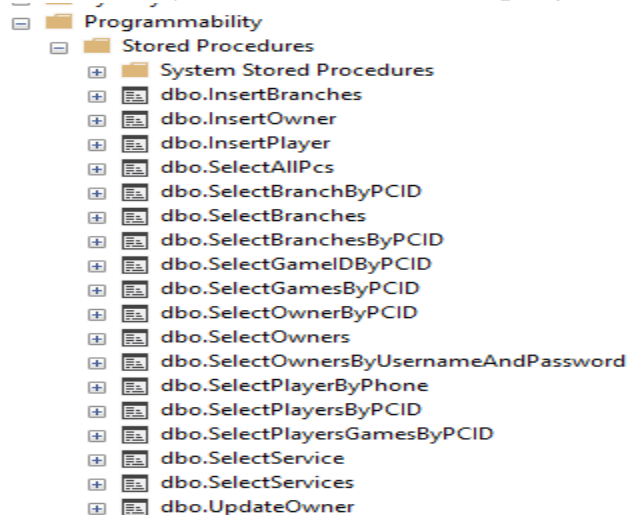
For time consumption and best practice create the database via Database Diagrams



Use New Table to create new one or Add table to add an already created table to the diagram.

You can set primary keys and everything from here it is very fast and helpful.

Then after you build them Go to programmability then stored Procedures



We create the proc by right click then new proc

It will show something like this

```
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =====
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
-- Add the parameters for the stored procedure here
<@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
<@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
SET NOCOUNT ON;

-- Insert statements for procedure here
SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

You can delete all of this and start fresh like this

```
create proc SelectGamesByID
@ID int
as
Select * from CyberGames where ID = @ID
```

Simple and clear.

NOTE: WHEN NAMING THE STORED PROC – Make sure it means what it do.


SQL insert Update delete and select then table name then the where

InsertStudent – UpdateTeacher – DeleteAll – DeleteGamerById – DeleteGamerByName

Make sure it says what it does follow the naming conventions above it will help you in the code

Step 2 the C#

We create a dotnet API application using Dotnet 6 and minimal API.

 **ASP.NET Core Web API**
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

C#LinuxmacOSWindowsCloudServiceWebWebAPI

Configure your new project

ASP.NET Core Web API C#LinuxmacOSWindowsCloudServiceWebWebAPI

Project name

Location

 ...

Solution name ⓘ

☒ Place solution and project in the same directory

Additional information

ASP.NET Core Web API C#LinuxmacOSWindowsCloudServiceWebWebAPI

Framework ⓘ

Authentication type ⓘ

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

☐ Use controllers (uncheck to use minimal APIs) ⓘ

☒ Enable OpenAPI support ⓘ

☐ Do not use top-level statements ⓘ

Make sure to De-Select the Use Controllers to use the minimal APIs

After that you will get a code like this

```
SKULLAPI WeatherForecast
1  var builder = WebApplication.CreateBuilder(args);
2
3  // Add services to the container.
4  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
5  builder.Services.AddEndpointsApiExplorer();
6  builder.Services.AddSwaggerGen();
7
8  var app = builder.Build();
9
10 // Configure the HTTP request pipeline.
11 if (app.Environment.IsDevelopment())
12 {
13     app.UseSwagger();
14     app.UseSwaggerUI();
15 }
16
17 app.UseHttpsRedirection();
18
19 var summaries = new[]
20 {
21     "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
22 };
23
24 app.MapGet("/weatherforecast", () =>
25 {
26     var forecast = Enumerable.Range(1, 5).Select(index =>
27         new WeatherForecast
28         (
29             DateTime.Now.AddDays(index),
30             Random.Shared.Next(-20, 55),
31             summaries[Random.Shared.Next(summaries.Length)]
32         )
33     ).ToArray();
34     return forecast;
35 })
36 .WithName("GetWeatherForecast");
37
38 app.Run();
39
40 1 reference
41 internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
42 {
43     0 references
44     public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
45 }
```

That is the default coding.

We will do a little bit of change

```
1  var builder = WebApplication.CreateBuilder(args);
2  builder.Services.AddEndpointsApiExplorer();
3  builder.Services.AddSwaggerGen();
4  var app = builder.Build();
5  if (app.Environment.IsDevelopment()) app.UseSwagger().UseSwaggerUI();
6  app.UseHttpsRedirection();
```

That is all.

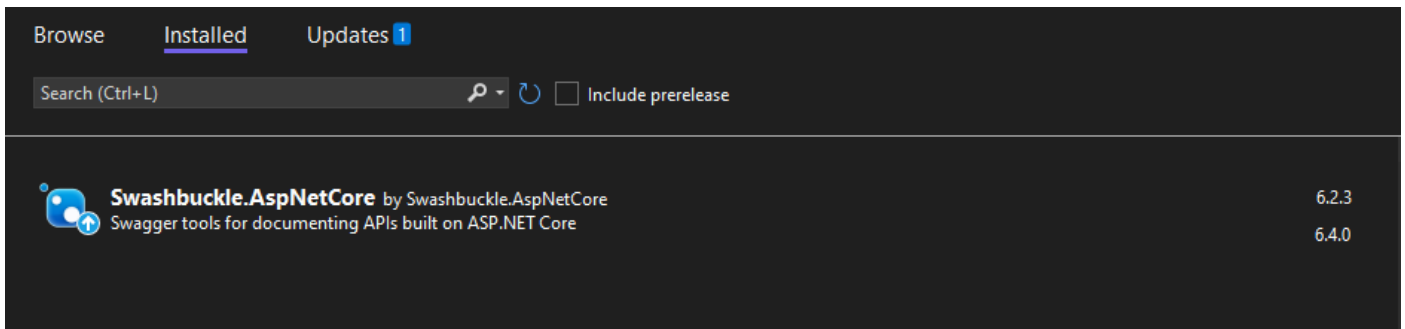
Now the real work starts

We will go to nuget and download 2 very important packages.

Microsoft.Data.SqlClient

Dapper

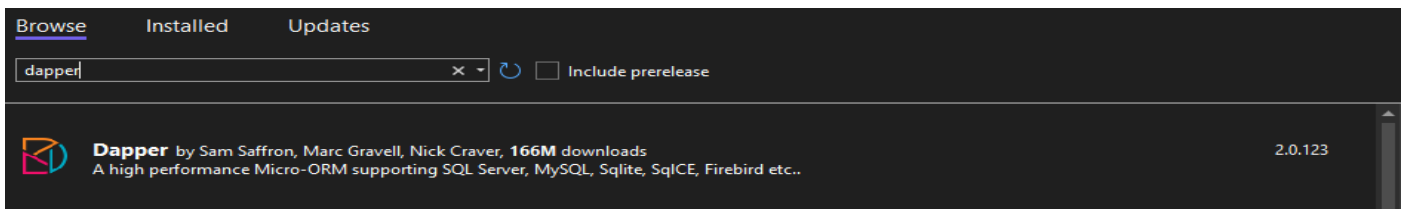
But before that lets update swagger



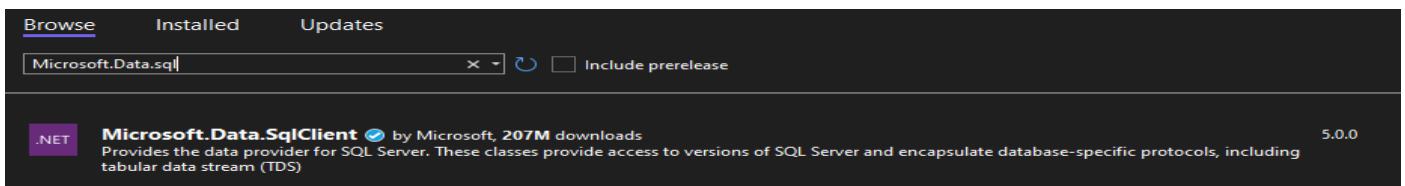
Swagger is an open API like postman that build UI for the API for testing and development phase very good and easy.

It comes with the API but not always up to date

Dapper



Sql Client



NOTE: DO NOT USE SYSTEM.DATA.SQLCLIENT ANYMORE – It is legacy and does not get updated and not compatible very well with the dotnet core also have missing features.

Now you look good.

Let's code

Step 3 The Code

Using...

```
1 using Microsoft.Data.SqlClient;  
2 using Dapper;
```

Then the Connection string

Go to Appsettings.json will look like this

```
1 {  
2   "Logging": {  
3     "LogLevel": {  
4       "Default": "Information",  
5       "Microsoft.AspNetCore": "Warning"  
6     }  
7   },  
8   "AllowedHosts": "*"   
9 }  
10
```

There is also another file for dev if you use local DB for testing and Public DB for production

```
Properties  
appsettings.json  
appsettings.Development.json
```

Use each one in its each place

```
1 {  
2   "ConnectionStrings": {  
3     "DB": "Data Source=YourServerName or IP;Initial Catalog=DatabaseName;User ID=sa {SA is server admin the default} ;password= The Password;TrustServerCertificate=True"  
4   },  
5   "Logging": {  
6     "LogLevel": {  
7       "Default": "Information",  
8       "Microsoft.AspNetCore": "Warning"  
9     }  
10  },  
11  "AllowedHosts": "*"   
12 }  
13
```

```
Data Source=YourServerName;Initial Catalog=DatabaseName;User ID=sa;  
password=123;TrustServerCertificate=True
```

NOTE: MAKE SURE THAT YOU HAVE **TrustServerCertificate = True**

Otherwise you will get errors that google wont help much with.

Now let's call it in the Code

```
1 using Microsoft.Data.SqlClient;
2 using Dapper;
3
4 var builder = WebApplication.CreateBuilder(args);
5 var constring = builder.Configuration.GetConnectionString("DB");
6
7
8 builder.Services.AddEndpointsApiExplorer();
9 builder.Services.AddSwaggerGen();
10 var app = builder.Build();
11 if (app.Environment.IsDevelopment()) app.UseSwagger().UseSwaggerUI();
12 app.UseHttpsRedirection();
```

Now that is done

Let's build the connection

```
5 var constring = builder.Configuration.GetConnectionString("DB");
6 var con = new SqlConnection(constring);
7
```

Then that is all

We query now.

We use mapGet Post Put And delete

Here is the APIs

Make sure to add one more import

```
1 using Microsoft.Data.SqlClient;
2 using Dapper;
3 using System.Data;
```

System.Data

```
14 app.MapGet("/", () => "Hello World!");
15 app.MapGet("/api/employees", () => con.Query("SELECT * FROM Employees"));
16 app.MapGet("/api/employees/{id}", (int id) => con.Query("SelectEmployeesByID", new { id }, commandType: CommandType.StoredProcedure));
17 app.Run();
```

This the simplest way to do API endpoints connected to database in C#

Step 4 Complicated scenarios

What if I want to make API that Post data and return a result if that is added or not

Let's see how it works

```
app.MapPost("/api/PostGame", ([FromBody] Game game) =>
{
    var BranchData = con.QuerySingleOrDefault<CyberBranch>("[SelectBranchByPCID]", new { game.PCID }, commandType: CommandType.StoredProcedure);
    if (BranchData is null)
        return new { message = "PCID is not valid" };
    var Game = con.ExecuteScalar("[SelectGameByGameName]", new { game.Name, BranchData.ID, BranchData.OwnersID }, commandType: CommandType.StoredProcedure);
    if (Game is null)
    {
        con.Execute("[InsertGame]", new { game.Name, BranchData.ID, BranchData.OwnersID }, commandType: CommandType.StoredProcedure);
        return new { message = "Game Added" };
    }
    else
    {
        return new { message = "Game Already Exists" };
    }
});
```

You can also return multiple arrays and use Linq

Something like this.

```
25 app.MapPost("/api/GetEverything", ([FromBody] string PCID) =>
26 {
27     var branches = con.Query("[SelectBranchesByPCID]", new { PCID }, commandType: CommandType.StoredProcedure);
28     var games = con.Query("[SelectGamesByPCID]", new { PCID }, commandType: CommandType.StoredProcedure);
29     var players = con.Query("[SelectPlayersByPCID]", new { PCID }, commandType: CommandType.StoredProcedure);
30     var playersGames = con.Query("[SelectGameIDByPCID]", new { PCID }, commandType: CommandType.StoredProcedure);
31
32     var players = Players.Select(_ => new { gamesid = playersGames.Where(x => x.id == _.id && x.branchid == _.branchid)
33     .Select(_ => _.gamesid), _.id, _.branchid, _.name, _.phone, _.Checked }).GroupBy(_ => _.id).Select(_ => _.FirstOrDefault());
34
35     return new { branches, games, players };
36 });
```

You can also do Data Manipulation in the code and filters and middleware's before and after you get the data from Database

```
app.MapPost("/api/PostPlayer", ([FromBody] Player player) =>
{
    dynamic result = new object();
    player.Phone = player.Phone.Replace(" ", "");
    player.Phone = player.Phone switch
    {
        _ when player.Phone.Any(char.IsLetter) || player.Phone.Length is < 11 or > 15 => result = player.Phone,
        _ when player.Phone.StartsWith("+2") => player.Phone.Replace("+2", "2"),
        _ when player.Phone.StartsWith("002") => player.Phone.Replace("002", "2"),
        _ when player.Phone.StartsWith("2") => player.Phone,
        _ when player.Phone.StartsWith("01") => player.Phone.Replace("01", "201"),
        _ when player.Phone.StartsWith("02") => player.Phone.Replace("02", "2"),
        _ => result = player.Phone
    };
    if (result is string)
        return result = new { message = "Phone Number is not valid" };
    var BranchData = con.QuerySingleOrDefault<CyberBranch>("[SelectBranchByPCID]", new { player.PCID }, commandType: CommandType.StoredProcedure);
    if (BranchData is null)
    {
        result = new { message = "PCID is not valid" };
    }
    else{
        var Player = con.ExecuteScalar("[SelectPlayerByPhone]", new { player.Phone, BranchData.ID, BranchData.OwnersID }, commandType: CommandType.StoredProcedure);
        if (Player is null)
        {
            con.Execute("[InsertPlayer]", new { player.Name, player.Phone, BranchData.ID, BranchData.OwnersID }, commandType: CommandType.StoredProcedure);
            result = new { message = "Player Added" };
        }
        else{
            result = new { message = "Player Already Exists" };
        }
    }

    return result;
});
```

NOTE, Apis docs can be found here [Minimal APIs overview | Microsoft Learn](#)

Also Dapper here [DapperLib/Dapper: Dapper - a simple object mapper for .Net \(github.com\)](#)

Read the MD file it has all the docs you need

Step 5 THE PUBLISH

There is a common theme that the code will work on your PC but not on the clients or servers or production

And in this case this code will not work at all

It will give error when fetching calling related to CORS

To fix that you can use a CORS Policy to add the domain name

Or just allow all headers Policy

```
builder.Services.AddCors(o => o.AddPolicy("AllowAll",
    P => P.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin()).AddEndpointsApiExplorer().AddSwaggerGen();

var app = builder.Build();
if (app.Environment.IsDevelopment())
    app.UseSwagger().UseSwaggerUI();
//app.UseHttpsRedirection();
app.UseCors("AllowAll");
```

Something like that

But NOTE HERE I Commented out UseHttpsRedirection Because if the domain I use does not have SSL or does not apply HTTPS. The application WON'T BE HOSTED AT ALL. And will always throw error.

So, if you publish it on HTTP just comment the code and apply it when they move to HTTPS or get SSL

THERE ARE MORE TO LEARN But this is the Simplest way and fastest way to create an API

The only limitation here is your imagination and your skills.

You can follow a standard but you truly shine when you create things your own way,

Otherwise why the hell are we get hired if we are told what to do.

We get hired to do what we are asked for.

Tell me what do you want, and I show you how to do it.

I am a brain before a hand.

The Source code for step 2 is Here

[Sharaf-Mansour/CsharpAPIwithDapper \(github.com\)](https://github.com/Sharaf-Mansour/CsharpAPIwithDapper)

The other source code is a real paid Project but for the sake of learning I can share it with screenshots because I can

Thank you for your time, Have a wonderful day.

Feel free to ask me anything on [LinkedIn](#) or [Facebook](#)

Happy Coding 😊❤️

Think More, Code Less