Leverage Github Copilot for Full Stack Development

# Scenario:

We have just retained a client that requires 2 reports be made to a Web based from International resellers. The client specifies the Web Portal will run using Angular and the API will use Dot Net connected to a SQLLite database (until they figure out what platform to port this to later)

Deliverables:

Dot Net API with REST interface to get Invoices and Sales reports as JSON.

Angular Application that queries DOT NET REST API for report data

SQL Lite Database

Queries

**select** i.billingcountry, sum(total) **as** 'TotalSales'  
**from** invoice **as** i  
**group** **by** billingcountry  
**order** **by** totalsales **desc**

A screenshot of a computer

Description automatically generated

**select** billingcountry, count(billingcountry) **as** 'InvoiceCount'  
**from** invoice  
**group** **by** billingcountry

A screenshot of a computer

Description automatically generated

## CoPilot Chat

Start setting your directory to your login root:

cd ~

Chat Prompt #1:

**Create** a C# Web API that uses ADO/Dapper **to** **access** a **SQL** Lite **database** at c:\users\johndohoney\reports.db, **Add** support **for** OpenAPI. This API will support a report controller that has 2 methods **to** getSales **and** getInvoices. Please **create** step **by** step instructions **to** **create** the dotnet api along **with** **all** necessary NuGet packages **and** a Program **to** run the API. **Create** Sales **and** Invoices Models **and** controllers **in** C#. **Create** a **database** context **to** **access** the **SQL** lite **database**. configure startup.cs **to** **enable** OpenAPI **and** run the API **for** **access** **from** browsers. The Invoices Model will have String BillingCountry **and** Integer InvliceCount The Sales Model will have a string BillingCountry **and** a Float TotalSales.

Response:

GitHub Copilot: Here are the step-by-step instructions to create a .NET Web API using ADO.NET/Dapper to access a SQLite database and support OpenAPI.

1. \*\*Create a new Web API project\*\*

Open a terminal and navigate to the directory where you want to create your project. Run the following command to create a new Web API project:

dotnet **new** webapi -n ReportsAPI

Navigate into the new project directory:

cd ReportsAPI

2. \*\*Add necessary NuGet packages\*\*

Run the following commands to install the necessary packages:

dotnet add **package** **Microsoft.EntityFrameworkCore.Sqlite**  
 dotnet add **package** **Dapper**  
 dotnet add **package** **Swashbuckle.AspNetCore**

3. \*\*Create Models\*\*

Create a new file `Models/Sales.cs`:

**namespace** **ReportsAPI.Models**  
 {  
 **public** **class** **Sales**  
 {  
 **public** **string** BillingCountry { **get**; **set**; }  
 **public** **float** TotalSales { **get**; **set**; }  
 }  
 }

Create a **new** file `Models/Invoices.cs`:

**namespace** **ReportsAPI.Models**  
 {  
 **public** **class** **Invoices**  
 {  
 **public** **string** BillingCountry { **get**; **set**; }  
 **public** **int** InvoiceCount { **get**; **set**; }  
 }  
 }

4. \*\*Create a Database Context\*\*

Create a new file `Data/ReportsContext.cs`:

**using** Microsoft.EntityFrameworkCore;  
 **using** ReportsAPI.Models;  
  
 **namespace** **ReportsAPI.Data**  
 {  
 **public** **class** **ReportsContext** : **DbContext**  
 {  
 **public** DbSet<Sales> Sales { **get**; **set**; }  
 **public** DbSet<Invoices> Invoices { **get**; **set**; }  
  
 **protected** **override** **void** **OnConfiguring**(DbContextOptionsBuilder optionsBuilder)  
 {  
 optionsBuilder.UseSqlite(@"Data Source=c:\users\johndohoney\reports.db");  
 }  
 }  
 }

\*\*Note that you will need to adjust the data source path for your own environment

5. \*\*Create a Reports Controller\*\*

Create a new file `Controllers/ReportsController.cs`:

**namespace** **ReportsAPI.Controllers**  
 {  
 [ApiController]  
 [Route("[controller]")]  
 **public** **class** **ReportsController** : **ControllerBase**  
 {  
 // Add getSales and getInvoices methods here  
 }  
 }

6. \*\*Create a Startup.cs file\*\*

Replace the existing `Startup.cs` file with the following:

using Microsoft.AspNetCore.Builder;  
 using Microsoft.AspNetCore.Hosting;  
 using Microsoft.Extensions.Configuration;  
 using Microsoft.Extensions.DependencyInjection;  
 using Microsoft.Extensions.Hosting;  
 using Microsoft.OpenApi.Models;  
 using ReportsAPI.Data;  
  
 namespace ReportsAPI  
 {  
 public **class** Startup  
 {  
 public Startup(IConfiguration configuration)  
 {  
 Configuration = configuration;  
 }  
  
 public IConfiguration Configuration { get; }  
  
 public void ConfigureServices(IServiceCollection services)  
 {  
 services.AddControllers();  
 services.AddDbContext<ReportsContext>();  
 services.AddSwaggerGen(c =>  
 {  
 c.SwaggerDoc("v1", new OpenApiInfo { Title = "ReportsAPI", Version = "v1" });  
 });  
 }  
  
 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
 {  
 **if** (env.IsDevelopment())  
 {  
 app.UseDeveloperExceptionPage();  
 }  
  
 app.UseHttpsRedirection();  
  
 app.UseRouting();  
  
 app.UseAuthorization();  
  
 app.UseEndpoints(endpoints =>  
 {  
 endpoints.MapControllers();  
 });  
  
 app.UseSwagger();  
 app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "ReportsAPI v1"));  
 }  
 }  
 }

7. \*\*Create a Program.cs file\*\*

Replace the existing `Program.cs` file with the following:

**using** Microsoft.AspNetCore.Hosting;  
 **using** Microsoft.Extensions.Hosting;  
  
 **namespace** **ReportsAPI**  
 {  
 **public** **class** **Program**  
 {  
 **public** **static** **void** **Main**(**string**[] args)  
 {  
 CreateHostBuilder(args).Build().Run();  
 }  
  
 **public** **static** IHostBuilder **CreateHostBuilder**(**string**[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .ConfigureWebHostDefaults(webBuilder =>  
 {  
 webBuilder.UseStartup<Startup>();  
 });  
 }  
 }

8. \*\*Run the API\*\*

dotnet **run**

A screen shot of a computer

Description automatically generated  
  
As it exists now, there are compile Issues, First, edit the csproj, and disable nullable.  
  
A screen shot of a computer program

Description automatically generated  
  
Next, most of the files have omitted “using” statement. Some code generation did add these; however, Co-Pilot can help with this task.   
  
Navigate to ReportsController.cs. Open up a blank line before the namespace directive, and use inline prompting by starting to type “using<tab>” In most cases there are a couple missing, so CoPilot will stop when all references are added.

Next, we have to add our queries.

Chat Prompt #2:

**Add** **sql** query **for** getsales **to** ReportsController. Use **SQL**: **select** i.billingcountry, sum(total) **as** 'TotalSales'  
**from** invoice **as** i  
**group** **by** billingcountry  
**order** **by** totalsales **desc**

Response:

GitHub Copilot: To add the `getSales` method to the `ReportsController`, you will need to use Dapper to execute the SQL query. Here's how you can do it:

First, you need to inject `IDbConnection` into your controller. This can be done in the constructor of your `ReportsController`.

Then, you can use this connection to execute your SQL query in the `getSales` method.

Here's how your `ReportsController` should look like:

**using** System.Collections.Generic;  
**using** System.Data;  
**using** System.Linq;  
**using** Dapper;  
**using** Microsoft.AspNetCore.Mvc;  
**using** ReportsAPI.Models;  
  
**namespace** **ReportsAPI.Controllers**  
{  
 [ApiController]  
 [Route("[controller]")]  
 **public** **class** **ReportsController** : **ControllerBase**  
 {  
 **private** **readonly** IDbConnection \_db;  
  
 **public** **ReportsController**(IDbConnection db)  
 {  
 \_db = db;  
 }  
  
 [HttpGet("getSales")]  
 **public** ActionResult<IEnumerable<Sales>> GetSales()  
 {  
 **var** sql = @"select i.billingcountry, sum(total) as 'TotalSales'  
 from invoice as i  
 group by billingcountry  
 order by totalsales desc";  
  
 **var** sales = \_db.Query<Sales>(sql).ToList();  
  
 **return** Ok(sales);  
 }  
  
 // Add getInvoices method here  
 }  
}

Please note that you need to configure the `IDbConnection` in your `Startup.cs` file to use SQLite and point to your database file. Here's how you can do it:

public void ConfigureServices(IServiceCollection services)  
{  
 services.AddControllers();  
 services.AddTransient<IDbConnection>(b => **new** SqliteConnection("Data Source=c:\\users\\johndohoney\\reports.db"));  
 services.AddSwaggerGen(c =>  
 {  
 c.SwaggerDoc("v1", new OpenApiInfo { Title = "ReportsAPI", Version = "v1" });  
 });  
}

Don't forget to add the necessary `using` statements for `Dapper` and `Microsoft.Data.Sqlite`.  
  
Chat Prompt #3:

**Add** **sql** query **for** getInvoices **to** ReportsController. Use **SQL**: **select** billingcountry, count(billingcountry) **as** 'InvoiceCount'  
**from** invoice  
**group** **by** billingcountry

Response:

GitHub Copilot: To add the `getInvoices` method to the `ReportsController`, you can use Dapper to execute the SQL query. Here's how you can do it:

[HttpGet("getInvoices")]  
public ActionResult<IEnumerable<Invoices>> GetInvoices()  
{  
 **var** sql = @"select billingcountry, **count**(billingcountry) **as** 'InvoiceCount'  
 from invoice  
 group **by** billingcountry";  
  
 **var** invoices = \_db.**Query**<Invoices>(sql).ToList();  
  
 **return** Ok(invoices);  
}

Now, your `ReportsController` should look like this:

**using** System.Collections.Generic;  
**using** System.Data;  
**using** System.Linq;  
**using** Dapper;  
**using** Microsoft.AspNetCore.Mvc;  
**using** ReportsAPI.Models;  
  
**namespace** **ReportsAPI.Controllers**  
{  
 [ApiController]  
 [Route("[controller]")]  
 **public** **class** **ReportsController** : **ControllerBase**  
 {  
 **private** **readonly** IDbConnection \_db;  
  
 **public** **ReportsController**(IDbConnection db)  
 {  
 \_db = db;  
 }  
  
 [HttpGet("getSales")]  
 **public** ActionResult<IEnumerable<Sales>> GetSales()  
 {  
 **var** sql = @"select i.billingcountry, sum(total) as 'TotalSales'  
 from invoice as i  
 group by billingcountry  
 order by totalsales desc";  
  
 **var** sales = \_db.Query<Sales>(sql).ToList();  
  
 **return** Ok(sales);  
 }  
  
 [HttpGet("getInvoices")]  
 **public** ActionResult<IEnumerable<Invoices>> GetInvoices()  
 {  
 **var** sql = @"select billingcountry, count(billingcountry) as 'InvoiceCount'  
 from invoice  
 group by billingcountry";  
  
 **var** invoices = \_db.Query<Invoices>(sql).ToList();  
  
 **return** Ok(invoices);  
 }  
 }  
}

This will add a new endpoint to your API that responds to `GET /reports/getInvoices` and returns a list of invoices grouped by `billingcountry`.

I tried to compile my code at this point, and got an error about SqliteConnection, so CoPilot to the rescue.

Chat Prompt #4:

What imports **do** I need **for** SqliteConnection

Response:

GitHub Copilot: To use `SqliteConnection`, you need to import the `Microsoft.Data.Sqlite` namespace. Here is the import statement:

**using** Microsoft.Data.Sqlite;

This namespace is part of the `Microsoft.Data.Sqlite` NuGet package. If you haven't already installed this package, you can do so with the following command:

dotnet add package Microsoft.Data.Sqlite

Your code is ready to run

dotnet **run**

Alternatively, you can run this in the VSC Debugger.

Select the Debugger (CTRL + SHIFT + D)

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

http or https C++ will work

Set some break points and walk through your code.

Take a note of the port that your app is listening on:  
  
A computer screen with text on it

Description automatically generated

If we navigate to localhost:5026/Reports/getSales, we should see:

A screenshot of a computer

Description automatically generated  
  
If we navigate to localhost:5026/Reports/ getInvoices, we should see:

A screenshot of a computer

Description automatically generated