How to create a lab using Visual Studio code and Github CoPilot

A guide to create a Blazor page that calls a DOTNET backend

# Introduction

In this document, you will learn how to create a lab using Visual Studio code and Github CoPilot. Github CoPilot is an AI-powered code completion tool that helps you write code faster and with less errors. You will use Github CoPilot to create a Blazor page that calls a DOTNET backend. Blazor is a framework for building interactive web UIs using C# and HTML. You will also learn how to use Visual Studio code, a lightweight and powerful code editor, to write and debug your code.

# Prerequisites

Before you start, you will need the following:

* A Github account and a Github repository for your lab.
* Visual Studio code installed on your computer.
* The .NET 5.0 SDK installed on your computer.
* The Blazor WebAssembly template installed on your computer.
* The Github CoPilot extension installed on your Visual Studio code.

# Steps

Follow these steps to create your lab:

1. Open Visual Studio code and create a new folder for your lab.
2. In the terminal, run the command dotnet new blazorwasm -o Lab to create a new Blazor WebAssembly project in the Lab folder.
3. Open the Lab folder in Visual Studio code and select the Github CoPilot icon on the left sidebar.
4. Sign in to your Github account and authorize Github CoPilot to access your code.
5. Install the following extensions:

* ms-dotnettools.csharp
* ms-dotnettools.blazorwasm-companion.

1. Prompt CoPilot Chat:

**Create** a Blazor Web **Add** that will run **in** a browser. **Create** **all** **dotnet** commands **to** setup **project** **and** run it, **and** **install** NuGet Packages. The Web App will display 2 reports. The **GLobal** Sales report calls the https://localhost:7044/Reports/getSales **and** displays the **data** **in** a table. The **Global** Invoices report calls https://localhost:7044/Reports/getInvoices. The webapp must run **in** Edge, Safari, **and** Firefox. We want **to** develop **and** debug this **in** visual studio code. Please suggest best extensions

GitHub Copilot: To create a Blazor WebAssembly app that runs in a browser, you will need to follow these steps:

1. Install the necessary tools: .NET Core SDK, Visual Studio Code, and the C# extension for Visual Studio Code.

2. Create a new Blazor WebAssembly project.

3. Install necessary NuGet packages.

4. Create the pages for the Global Sales and Global Invoices reports.

5. Implement the logic to fetch and display the data from the provided endpoints.

6. Run and debug the application in Visual Studio Code.

Here are the detailed steps:

1. Install the .NET Core SDK from the official Microsoft website.

2. Install Visual Studio Code from the official website.

3. Install the C# extension for Visual Studio Code. You can do this by going to the Extensions view in Visual Studio Code (View -> Extensions) and searching for 'C#'.

4. Open the terminal in Visual Studio Code and run the following command to create a new Blazor WebAssembly project:

dotnet **new** blazorwasm -o MyBlazorApp

5. Navigate to the newly created project directory :

cd MyBlazorApp

6. Install the necessary NuGet packages. For example, if you need the `Newtonsoft.Json` package, you can install it with the following command, then start VSC:

dotnet add **package** **Newtonsoft.Json**

**code .**

7. Create the pages for the Global Sales and Global Invoices reports. In the `Pages` directory, create two new files: `Sales.razor` and `Invoices.razor`. In these files, you can use the `@page` directive to specify the route, and then implement the HTML and C# code to fetch and display the data.

8. To fetch the data, you can use the `HttpClient` class, which is injected into Blazor apps by default. Here's an example of how you can fetch and display data in the `Sales.razor` page:

DON’T Do this, it is far too general, we need to further prompt CoPilot to Get more precise code for calling and rendering the JSON response data

@page "/sales"  
@inject HttpClient Http  
  
<h1>Global Sales</h1>  
  
@**if** (salesData == null)  
{  
 <p><em>Loading...</em></p>  
}  
**else**  
{  
 // Display the data in a table  
}  
  
@code {  
 **private** SalesData[] salesData;  
  
 **protected** **override** **async** Task **OnInitializedAsync**()  
 {  
 salesData = **await** Http.GetFromJsonAsync<SalesData[]>("https://localhost:7044/Reports/getSales");  
 }  
}

Lets prompt for an Invoices Razor Page

Add the Invoice Report Blazor Page. This Page will **call** https://localhost:7044/Reports/getInvoices. The API will **return** the **following** **JSON** **data**:   
Response **body**  
Download  
[  
 {  
 "billingCountry": "Argentina",  
 "invoiceCount": 7  
 },  
 {  
 "billingCountry": "Australia",  
 "invoiceCount": 7  
 },  
 {  
 "billingCountry": "Austria",  
 "invoiceCount": 7  
 }] I need the Page **to** display this **data** **in** a **table** **with** a CSS that colors the **rows** **in** alternating colors

GitHub Copilot: To create the `Invoices.razor` page that fetches and displays the data in a table with alternating row colors, you can follow these steps:

1. Create a new file named `Invoices.razor` in the `Pages` directory.

2. Use the `@page` directive to specify the route.

3. Inject the `HttpClient` class to fetch the data from the API.

4. Implement the HTML and C# code to fetch and display the data.

Here's the code for the `Invoices.razor` page:

@page "/invoices"  
@inject HttpClient Http  
  
<h1>Global Invoices</h1>  
  
@**if** (invoiceData == null)  
{  
    <p><em>Loading...</em></p>  
}  
**else**  
{  
    <table **class**="table">  
        <thead>  
            <tr>  
                <th>Billing Country</th>  
                <th>Invoice Count</th>  
            </tr>  
        </thead>  
        <tbody>  
            @foreach (var invoice in invoiceData)  
            {  
                <tr>  
                    @using static Invoices  
                    <td>@invoice.BillingCountry</td>   
                    <td>@invoice.InvoiceCount</td>  
                </tr>  
            }  
        </tbody>  
    </table>  
}  
  
@code {  
    #nullable enable  
  
    **private** InvoiceData[]? invoiceData;  
  
    **protected** **override** async Task OnInitializedAsync()  
    {  
        // add exception handling  
        {  
            **try**  
            {  
                invoiceData = await Http.GetFromJsonAsync<InvoiceData[]>("https://localhost:7044/Reports/getInvoices");  
                InvoiceData[]? foo = invoiceData;  
                // refresh the page with data  
                StateHasChanged();            }  
            **catch** (Exception ex)  
            {  
                Console.WriteLine(ex.Message);  
            }  
        }  
    }  
  
    **public** **class** InvoiceData  
    {  
        **public** string? BillingCountry { **get**; **set**; }  
        **public** int InvoiceCount { **get**; **set**; }  
    }  
}

To style the table with alternating row colors, you can add the following CSS to the `wwwroot/css/app.css` file: (The file is already there, just add the rule)

.table **tr**:nth-child(even) {  
 **background-color**: #f2f2f2;  
}

This CSS rule will apply a light gray background color to even rows in the table. You can adjust the color as needed.

@@@@@@@@@@@@@@@@@

9. You can run the application with the following command:

dotnet **run**

10. To debug the application in Visual Studio Code, you can use the built-in debugger. First, you need to create a launch configuration file (`launch.json`). You can do this by going to the Run view (View -> Run), and then clicking on the 'create a launch.json file' link. In the launch configuration file, you can specify the project to run and the browser to use.

Here's an example of a launch configuration for a Blazor WebAssembly app:

{  
 "version": "0.2.0",  
 "configurations": [  
 {  
 "name": ".NET Core Launch (Blazor Standalone)",  
 "type": "coreclr",  
 "request": "launch",  
 "preLaunchTask": "build",  
 "program": "${workspaceFolder}/bin/Debug/netstandard2.1/MyBlazorApp.dll",  
 "args": [],  
 "cwd": "${workspaceFolder}",  
 "stopAtEntry": false,  
 "serverReadyAction": {  
 "action": "openExternally",  
 "pattern": "\\bNow listening on:\\s+(https?://\\S+)"  
 },  
 "env": {  
 "ASPNETCORE\_ENVIRONMENT": "Development"  
 },  
 "dotNetAttach": {  
 "logging": {  
 "moduleLoad": false,  
 "trace": false,  
 "engineLogging": false,  
 "browserStdOut": false  
 }  
 }  
 }  
 ]  
}

11. To ensure that your web app runs in Edge, Safari, and Firefox, you should test it in these browsers. You can specify the browser to use in the `serverReadyAction` section of the launch configuration file.

How **do** I add my Invoices Razor page **to** the nav menu?

**GitHub Copilot**

To add your Invoices Razor page to the navigation menu, you need to add a new div element with a NavLink component pointing to your Invoices page. The href attribute of the NavLink component should match the route of your Invoices page. Here's how you can do it:

<**div** class="nav-item px-3">  
    <**NavLink** class="nav-link" href="invoices">  
        <**span** class="bi bi-file-earmark-text-nav-menu" aria-hidden="true"></**span**> Invoices  
    </**NavLink**>  
</**div**>

You can replace the bi-file-earmark-text-nav-menu class with the appropriate Bootstrap icon class for your Invoices page. If you don't have a specific icon, you can just remove the span element.

Add this code to your NavMenu.razor file where you want the Invoices link to appear. For example, if you want it to appear after the Weather link, you would add it after the div containing the Weather NavLink.

## TotalSales Page

How do I add **my** Total Sales Razor page **to** process **that** JSON data, **it** **contains** BillingCountry **as** string **and** TotalSales **as** Float Display this **as** a bar chart **with** BillingCountry **on** **the** X-Axis **and** TotalSales **on** **the** Y-Axis

GitHub Copilot: To display the JSON data as a bar chart, you can use a JavaScript library like Chart.js. Here's a step-by-step guide:

1. Add Chart.js to your project. You can do this by adding the following script tag to your `\_Host.cshtml` or `index.html` file:

<**script** src="https://cdn.jsdelivr.net/npm/chart.js"></**script**>

2. Create a new Razor page named `TotalSales.razor`.

3. In the `TotalSales.razor` page, add a `canvas` element where the chart will be rendered:

<**canvas** id="totalSalesChart"></**canvas**>

4. Define a `TotalSalesData` class in the `@code` block:

**public** **class** **TotalSalesData**  
{  
 **public** **string** BillingCountry { **get**; **set**; }  
 **public** **float** TotalSales { **get**; **set**; }  
}

5. Fetch the data from the server in the `OnInitializedAsync` method:

**private** TotalSalesData[] totalSalesData;  
  
**protected** **override** **async** Task **OnInitializedAsync**()  
{  
 totalSalesData = **await** Http.GetFromJsonAsync<TotalSalesData[]>("https://localhost:7044/Reports/getTotalSales");  
}

6. Add a JavaScript interop method to create the chart:

protected override async Task OnAfterRenderAsync(bool firstRender)  
{  
 **if** (firstRender && totalSalesData != null)  
 {  
 var labels = totalSalesData.Select(data => data.BillingCountry).ToArray();  
 var data = totalSalesData.Select(data => data.TotalSales).ToArray();  
  
 await JSRuntime.InvokeVoidAsync("createChart", "totalSalesChart", labels, data);  
 }  
}

7. In your `wwwroot/index.html` or `\_Host.cshtml` file, add the `createChart` JavaScript function:

window.createChart = (canvasId, labels, data) => {  
 var ctx = document.getElementById(canvasId).getContext('2d');  
 **new** Chart(ctx, {  
 type: 'bar',  
 data: {  
 labels: labels,  
 datasets: [{  
 label: 'Total Sales',  
 data: data,  
 backgroundColor: 'rgba(75, 192, 192, 0.2)',  
 borderColor: 'rgba(75, 192, 192, 1)',  
 borderWidth: 1  
 }]  
 },  
 options: {  
 scales: {  
 y: {  
 beginAtZero: true  
 }  
 }  
 }  
 });  
};

8. Finally, add the `TotalSales` page to your navigation menu:

<**div** class="nav-item px-3">  
 <**NavLink** class="nav-link" href="totalsales">  
 <**span** class="bi bi-bar-chart-fill" aria-hidden="true"></**span**> Total Sales  
 </**NavLink**>  
</**div**>

This will create a bar chart with `BillingCountry` on the X-axis and `TotalSales` on the Y-axis.

Create the **code** **for** the totalsales.razor page

GitHub Copilot: Sure, here's the code for the `TotalSales.razor` page. This code assumes that you have a `TotalSalesData` endpoint that returns an array of objects, each with a `BillingCountry` string and a `TotalSales` float.

@page "/totalsales"  
@inject HttpClient Http  
@inject IJSRuntime JSRuntime  
  
<h1>Total Sales</h1>  
  
@if (totalSalesData == null)  
{  
 <p><em>Loading...</em></p>  
}  
**else**  
{  
 <canvas id="totalSalesChart"></canvas>  
}  
  
@code {  
 **private** TotalSalesData[]? totalSalesData;  
  
 **protected** **override** async Task OnInitializedAsync()  
 {  
 **try**  
 {  
 totalSalesData = await Http.GetFromJsonAsync<TotalSalesData[]>("https://localhost:7044/Reports/getTotalSales");  
 }  
 **catch** (Exception ex)  
 {  
 Console.WriteLine(ex.Message);  
 }  
 }  
  
 **protected** **override** async Task OnAfterRenderAsync(bool firstRender)  
 {  
 **if** (firstRender && totalSalesData != null)  
 {  
 **var** labels = totalSalesData.Select(**data** => **data**.BillingCountry).ToArray();  
 **var** **data** = totalSalesData.Select(**data** => **data**.TotalSales).ToArray();  
  
 await JSRuntime.InvokeVoidAsync("createChart", "totalSalesChart", labels, **data**);  
 }  
 }  
  
 **public** **class** **TotalSalesData**  
 {  
 **public** string? BillingCountry { **get**; **set**; }  
 **public** float TotalSales { **get**; **set**; }  
 }  
}

This code fetches the total sales data from the server when the component is initialized. After the component is rendered, it uses JavaScript interop to create a bar chart with Chart.js. The `BillingCountry` values are used as labels on the X-axis, and the `TotalSales` values are used as data on the Y-axis.