[Hopp til innhold](https://chatgpt.com/c/68c33b27-3aec-8327-8df9-7975f5e7e63a#main)

<https://chatgpt.com/c/68c33b27-3aec-8327-8df9-7975f5e7e63a>

[Hopp til innhold](https://chatgpt.com/c/68c33b27-3aec-8327-8df9-7975f5e7e63a#main)

* **Chattehistorikk**
* **Du sa:**

I am a total newbie in Windows programming so I wonder how do I VIBE web develop with .net, C#, GitHub Copilot code helper and with commit in the IDE Visual Studio Code to program an ToDo webapp. I like to store content and tables in a live free database and publish the code on a free webserver so I can reach it from an weburl. Can you make me an 8 chapters tutorial (step by step)? Show me first the table of contents and show med one chapter at a time.

* **ChatGPT sa:**

Tenkte i 32s

Nice — let’s do this! I’ll give you an 8-chapter beginner-friendly tutorial that walks you from zero → live ToDo webapp (C#, .NET, VS Code + GitHub Copilot, free live DB, and a free URL).

First I’ll show the **Table of contents**, then I’ll deliver **Chapter 1** (Environment & tools). After you read Chapter 1, say **“Next”** and I’ll show Chapter 2, etc — one chapter at a time as you requested.

* **Table of contents (8 chapters)**

1. **Setup: install .NET, VS Code, Git, GitHub, and GitHub Copilot** — get your dev environment ready on Windows.
2. **Scaffold the project** — create a simple Blazor WebAssembly app (C# front-end) so you can write UI in C#.
3. **Make a local ToDo prototype (model + mock service)** — build the Todo model and a simple local data service so you can iterate fast.
4. **Create a free live database (Supabase)** — sign up, create a todos table, get your API URL & anon key (free Postgres).
5. **Wire the app to Supabase** — implement an HTTP service in C# to do Create/Read/Update/Delete against Supabase REST endpoints.
6. **UI: list, add, edit, delete, UX polish** — build the Razor/Blazor pages and use Copilot prompts to speed coding.
7. **Git + GitHub + Copilot workflow in VS Code** — init repo, commit, push, use Copilot effectively, and add CI (GitHub Action).
8. **Deploy to a free host & secure keys** — publish the app to a free static host (GitHub Pages or Netlify) and set secrets; final checks and next features.

* **Chapter 1 — Setup: install .NET, VS Code, Git, GitHub, and GitHub Copilot**

Goal: get a working Windows dev environment so you can create, build, run, and commit .NET/C# web code from VS Code.

* **1.1 What you'll install (quick list)**
* **.NET SDK** (the .NET SDK that lets you dotnet new, build and run apps).
* **Visual Studio Code** (lightweight editor with extensions).
* **Git** (version control CLI).
* **GitHub account** (host your repo and later publish).
* **GitHub Copilot extension** in VS Code (AI code helper).
* Optional but recommended: **Windows Terminal** (better terminal), **REST Client** extension (test APIs), **C# extension** (for C# intellisense).
* **1.2 Step-by-step install (Windows, beginner-friendly)**
* **.NET SDK**

1. Open a browser and go to the official .NET download page (search “dotnet download”).
2. Download and install the **.NET SDK** for Windows (the installer). Accept defaults.
3. Verify: open PowerShell (or Windows Terminal) and run:
4. dotnet --version

You should see a version number (e.g. 8.0.x or similar). If you see a number, .NET is installed.

* **Visual Studio Code**

1. Download VS Code from code.visualstudio.com and run the installer.
2. Open VS Code after install. (Pin it to the taskbar if you like.)

* **Git**

1. Download Git from git-scm.com and install using the default options.
2. Verify in a terminal:
3. git --version

* **GitHub account & Personal setup**

1. Create a free account at github.com (if you don’t have one).
2. In GitHub, you’ll later create a repository to push your code to. For now just create the account.

* **GitHub Copilot (in VS Code)**

1. In VS Code open the **Extensions** panel (left icon or Ctrl+Shift+X).
2. Search for **GitHub Copilot** and install the official extension.
3. Click **Sign in to GitHub** when prompted and authorize Copilot.
   * Copilot may require a subscription or trial (GitHub offers trials / student licenses sometimes). If you don’t have access, you can still code; Copilot is optional but helpful.

* **Useful VS Code extensions to install now**
* **C#** (adds language support, debugging).
* **GitLens** (supercharges Git UI).
* **REST Client** (send HTTP requests from an editor tab — great for testing Supabase).
* **EditorConfig for VS Code** (optional).

Install them from the Extensions view in VS Code.

* **1.3 Configure Git (one-time)**

From a terminal inside VS Code or PowerShell:

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

Optional: set up SSH keys for GitHub (search “GitHub SSH key Windows”). If you prefer HTTPS, you can push with username/password or GitHub token.

* **1.4 Bonus: quick check commands**

Open a new terminal (inside VS Code: `Ctrl+``) and run:

dotnet --version

git --version

code --version

All should print versions. If yes — you're set.

* **1.5 Quick notes (pitfalls & tips)**
* If dotnet --version fails, reboot or ensure the PATH was updated.
* Copilot suggestions appear inline — press Tab to accept, Esc to ignore. Learn to prompt Copilot with short comments like // Create method to fetch todos from Supabase and then accept the suggestion.
* Keep your GitHub credentials handy. VS Code will help with sign-in flows when you push for the first time.

That’s Chapter 1 — a working environment. When you’re ready I’ll show **Chapter 2 — Scaffolding the Blazor WebAssembly project** with exact commands, the basic project layout, and a tiny “Hello world” ToDo page so you can run the app locally.

Say **Next** when you want Chapter 2 (or tell me if you prefer a different stack — e.g., ASP.NET MVC or a React front-end — I’ll adapt the tutorial).

* **Du sa:**

Blazor is good. :) Next

* **ChatGPT sa:**

Perfect 😎 Blazor it is! We’ll use **Blazor WebAssembly** (C# in the browser, no backend server required). Later we’ll call the Supabase API from the client directly.

* **Chapter 2 — Scaffold the Blazor WebAssembly Project**

Goal: create the ToDo app skeleton, run it locally, and understand the folder structure.

* **2.1 Create the project**

Open VS Code terminal (`Ctrl+``) and run:

dotnet new blazorwasm -o TodoApp cd TodoApp

* dotnet new blazorwasm → creates a Blazor WebAssembly project.
* -o TodoApp → puts it in a new folder called **TodoApp**.
* **2.2 Run the project**

Still in the terminal:

dotnet run

You’ll see something like:

Now listening on: https://localhost:7173

Now listening on: http://localhost:5173

Open the https://localhost:7173 link in your browser. You’ll see the default Blazor “Hello, world!” app with a counter page.

✅ Congrats — you’ve got a working C# web app running in your browser!

* **2.3 Explore the project structure**

Inside **TodoApp**, notice these key folders:

* **wwwroot/** → static files (CSS, JS, images).
* **Pages/** → Blazor pages (.razor files). Default ones are Index.razor, Counter.razor, and FetchData.razor.
* **Shared/** → reusable components (e.g., NavMenu.razor).
* **Program.cs** → app startup file (configures the Blazor app).
* **TodoApp.csproj** → project definition.

Blazor pages (.razor) are like HTML + C#. Example:

@page "/hello" <h3>Hello Page</h3> <p>Hello from Blazor! The time is: @DateTime.Now</p>

Save this as Hello.razor in the **Pages/** folder. Add @page "/hello" at the top.  
Now run again (dotnet run) and navigate to /hello. Boom 💥 — a new page.

* **2.4 Add a placeholder ToDo page**

Let’s create the future home of our ToDo list.

In **Pages/Todo.razor**:

@page "/todo" <h3>My ToDo List</h3> <ul> <li>✅ Learn Blazor basics</li> <li>🔲 Connect to a database</li> <li>🔲 Deploy live</li> </ul>

Restart (dotnet run) → go to /todo in the browser. You should see your starter ToDo page.

* **2.5 Add to the navigation menu**

Open **Shared/NavMenu.razor**, and under the existing links add:

<NavLink class="nav-link" href="todo"> <span class="oi oi-list-rich" aria-hidden="true"></span> ToDo </NavLink>

Now you’ll see **ToDo** in the left-hand nav.

* **2.6 Quick recap**
* You scaffolded a Blazor WebAssembly app.
* You ran it locally (dotnet run).
* You created your own page (/todo).
* You added a menu link.

Your project now has a place for the ToDo app UI.

✅ Next step (Chapter 3): We’ll **make a real ToDo model + a local mock service**. This way you can add/remove items without a database yet — perfect for testing UI and Copilot.

Do you want me to also explain *how to commit this initial project into GitHub* now, or wait until Chapter 7 (the dedicated GitHub/Copilot workflow chapter)?

* **Du sa:**

Now run again (`dotnet run`) and navigate to `/hello`. Boom 💥 — a new page.

I get this error at http://localhost:5195/hello Sorry, there's nothing at this address.

* **ChatGPT sa:**

Ah, good catch 👍 That usually happens because Blazor doesn’t automatically know about your new page unless:

1. You saved the file in the right place, and
2. You restarted (or hot-reloaded) the app.

Let’s double-check together:

* **✅ Checklist to fix**/hello

1. **File location**
   * The file should be in:
   * TodoApp/Pages/Hello.razor
   * The very first line in that file must be:

@page "/hello"

1. **Restart the app**  
   After saving, go back to terminal and run:

dotnet run

Then open the new URL it shows (sometimes it changes ports, e.g. http://localhost:5195 or http://localhost:5173).  
Now try http://localhost:5195/hello.

1. **Case sensitivity**  
   Make sure you typed /hello (all lowercase). The route in @page is case-sensitive.
   * @page "/hello" → works at /hello
   * @page "/Hello" → works at /Hello
2. **Still not working?**  
   Check if the project actually rebuilt. Sometimes VS Code hot reload doesn’t pick up new files unless you restart.

* **🔍 Quick test**

Can you paste your full Hello.razor file here so I can verify the contents?

If it’s all good, /hello should definitely load after restarting.

👉 Do you want me to also show you how to make sure it shows up in the **navigation menu** so you don’t have to type /hello manually?

* **Du sa:**

Restart (`dotnet run`)

How do I Restart (dotnet run)?

* **ChatGPT sa:**

Good question — let’s make it super clear.

When you run dotnet run, the app keeps running in your terminal until you stop it. If you add a new page (like Hello.razor), Blazor won’t see it until you restart.

Here’s how to restart:

* **1. Stop the app**
* In the terminal where dotnet run is running, press:  
  **Ctrl + C**  
  (this tells .NET to stop the server).

You should see something like:

Application is shutting down...

* **2. Run it again**

In the same folder (TodoApp), type:

dotnet run

You’ll see something like:

Now listening on: http://localhost:5195

Now listening on: https://localhost:7195

* **3. Open your browser**
* Go to the port shown (e.g., http://localhost:5195/hello).
* If you added the @page "/hello" page correctly, it should now load.

👉 Quick check: do you see “My ToDo List” at /todo already? If /todo works but /hello doesn’t, then the file content or location might need fixing.

Do you want me to show you **exactly where to put**Hello.razor**in VS Code’s Explorer**, with the folder tree?

* **Du sa:**

Open \*\*Shared/NavMenu.razor\*\*, and under the existing links add:

This file "NavMenu.razor" is in the folder "Layout" and the folder Shared is missing.

* **ChatGPT sa:**

Ahh 👍 you’ve spotted a difference:  
That’s because the **Blazor WebAssembly template** got slightly reorganized in .NET 8.

* In older versions, NavMenu.razor lived under **Shared/**.
* In the newer template, it’s under **Layout/**.
* The **Shared/** folder might be missing completely — that’s normal in newer projects.

So don’t worry — your template is correct. 🎉

* **✅ What to do**
* Instead of Shared/NavMenu.razor, just edit:
* TodoApp/Components/Layout/NavMenu.razor

(you might see the full path as Components/Layout/NavMenu.razor depending on the exact SDK).

* Inside that file, you’ll see existing <NavLink>s for Home, Counter, Fetch data.
* Just add your new ToDo link **under them**. Example:

<NavLink class="nav-link" href="todo"> <span class="oi oi-list-rich" aria-hidden="true"></span> ToDo </NavLink>

* **🔍 Why the docs differ**

Microsoft adjusted the Blazor folder structure in newer SDKs to make components more modular. So if tutorials mention **Shared/**, just mentally swap it for **Components/** (and specifically Components/Layout for the nav).

👉 Question for you: Do you want me to rewrite all future steps using the **new folder structure** (so it matches exactly what you see in your project)? That way you won’t run into these “missing folder” confusions again.

* **Du sa:**

It is no folder named Components. The path is TodoApp/Layout/NavMenu.razor

* **ChatGPT sa:**

Got it ✅ Thanks for clarifying!  
You’re using the **simpler Blazor WebAssembly project template**, which puts things directly like this:

TodoApp/

├── wwwroot/

├── Pages/

├── Layout/

│ └── NavMenu.razor

├── Program.cs

└── TodoApp.csproj

So in **your** case:

* There is **no**Shared/ folder.
* The navigation is in Layout/NavMenu.razor.

That’s perfectly fine — you just need to edit that file instead of the Shared one.

* **✅ Add your ToDo link**

Open Layout/NavMenu.razor and add this snippet under the other <NavLink>s:

<NavLink class="nav-link" href="todo"> <span class="oi oi-list-rich" aria-hidden="true"></span> ToDo </NavLink>

Now when you restart (Ctrl+C → dotnet run), you should see **ToDo** in the sidebar menu. Clicking it should take you to /todo.

* **ℹ️ Why your project looks like this**
* Some tutorials use older templates (with Shared/).
* The .NET SDK you installed gave you a **newer Blazor template**, which has a flatter structure (Layout/ instead of Shared/).

So: everything is working as expected, just slightly different organization. 👍

👉 Do you want me to adapt all the upcoming chapters to **your exact folder layout** (so you never get these “missing folder” surprises again)?

* **Du sa:**

Yes

* **ChatGPT sa:**

Perfect ✅ I’ll keep everything aligned with **your structure** (Pages/ + Layout/), so no more surprises with missing folders.

So far, you’ve got:

* Todo.razor inside **Pages/**
* NavMenu.razor inside **Layout/** updated with a **ToDo link**
* Project runs at http://localhost:xxxx/todo 🎉
* **Coming up next**

In **Chapter 3** we’ll:

* Create a **ToDo model** (TodoItem.cs).
* Build a **mock data service** (stores ToDos in memory).
* Hook it up to the Todo.razor page so you can **add/remove items locally** (no database yet).

That way, you’ll see the app come alive before we connect Supabase.

👉 Ready for me to walk you through **Chapter 3** with the code?

* **Du sa:**

Next

* **ChatGPT sa:**

Awesome 😎 Let’s bring your ToDo page to life!

* **Chapter 3 — Make a Local ToDo Prototype (Model + Mock Service)**

Goal: create a TodoItem model, store items in memory, and display/add/remove them in your Todo.razor page **without a database yet**. This makes UI development fast and safe.

* **3.1 Create the ToDo model**

1. In your project root, create a new folder called **Models**:

TodoApp/Models

1. Inside **Models**, create a file TodoItem.cs:

namespace TodoApp.Models { public class TodoItem { public int Id { get; set; } *// unique identifier* public string Title { get; set; } *// task text* public bool IsDone { get; set; } *// completed flag* } }

✅ This is your simple ToDo class.

* **3.2 Create a mock service**

We’ll store items in memory while the app is running.

1. Create a new folder **Services**:

TodoApp/Services

1. Inside **Services**, create TodoService.cs:

using TodoApp.Models; namespace TodoApp.Services { public class TodoService { private readonly List<TodoItem> \_todos = new List<TodoItem>(); private int \_nextId = 1; public List<TodoItem> GetAll() => \_todos; public void Add(string title) { \_todos.Add(new TodoItem { Id = \_nextId++, Title = title, IsDone = false }); } public void ToggleDone(int id) { var todo = \_todos.FirstOrDefault(t => t.Id == id); if (todo != null) todo.IsDone = !todo.IsDone; } public void Remove(int id) { var todo = \_todos.FirstOrDefault(t => t.Id == id); if (todo != null) \_todos.Remove(todo); } } }

* **3.3 Register the service in Blazor**

Open Program.cs and add the service:

using TodoApp.Services; var builder = WebApplication.CreateBuilder(args); builder.Services.AddRazorPages(); builder.Services.AddServerSideBlazor(); *// required* builder.Services.AddSingleton<TodoService>(); *// register our mock service* var app = builder.Build(); *// ... existing code*

Using AddSingleton keeps your ToDo list in memory while the app is running.

* **3.4 Update**Todo.razor**to use the service**

Replace your existing placeholder content with this:

@page "/todo" @using TodoApp.Models @inject TodoService TodoService <h3>My ToDo List</h3> <input @bind="newTodo" placeholder="Enter new task" /> <button @onclick="AddTodo">Add</button> <ul> @foreach (var todo in TodoService.GetAll()) { <li> <input type="checkbox" @bind="todo.IsDone" @onchange="() => ToggleTodo(todo.Id)" /> <span style="text-decoration:@(todo.IsDone ? "line-through" : "none")">@todo.Title</span> <button @onclick="() => RemoveTodo(todo.Id)">❌</button> </li> } </ul> @code { private string newTodo = ""; private void AddTodo() { if (!string.IsNullOrWhiteSpace(newTodo)) { TodoService.Add(newTodo); newTodo = ""; } } private void ToggleTodo(int id) => TodoService.ToggleDone(id); private void RemoveTodo(int id) => TodoService.Remove(id); }

* **3.5 Run and test**

1. Restart the app:

Ctrl + C dotnet run

1. Open http://localhost:5195/todo (or your actual port).
2. You should be able to:

* Add tasks
* Mark tasks done/undone
* Remove tasks

All in memory — no database yet.

* **✅ Recap**
* TodoItem.cs → defines your data.
* TodoService.cs → stores and manipulates ToDos.
* Todo.razor → UI binds to the service.

You now have a **fully functional local ToDo app**! This is perfect for testing UI and experimenting with Copilot before connecting a real database.

Next step (Chapter 4): we’ll create a **free live database with Supabase** and prepare to replace this mock service with real CRUD operations.

Do you want me to move on to Chapter 4?

* **Du sa:**

Open `Program.cs` and add the service:

The code is failing based on the code you are suggestion, so I need help. Here is the code in my existing Program.cs file: using Microsoft.AspNetCore.Components.Web; using Microsoft.AspNetCore.Components.WebAssembly.Hosting; using TodoApp; var builder = WebAssemblyHostBuilder.CreateDefault(args); builder.RootComponents.Add<App>("#app"); builder.RootComponents.Add<HeadOutlet>("head::after"); builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) }); await builder.Build().RunAsync();

* **ChatGPT sa:**

Tenkte i 14s

Nice — thanks for pasting your Program.cs. That helped. The problem is just that my earlier instruction (meant for a server-side Blazor app) suggested AddServerSideBlazor() — which doesn't belong in a **Blazor WebAssembly** project. For your WASM app you should register the TodoService on the builder.Services used by WebAssemblyHostBuilder.

Below is a corrected, ready-to-use Program.cs for your Blazor WebAssembly project plus a few small fixes you should make to TodoService and Todo.razor.

* **1) Replace your**Program.cs**with this**

using Microsoft.AspNetCore.Components.Web; using Microsoft.AspNetCore.Components.WebAssembly.Hosting; using TodoApp; using TodoApp.Services; *// <--- add this* var builder = WebAssemblyHostBuilder.CreateDefault(args); builder.RootComponents.Add<App>("#app"); builder.RootComponents.Add<HeadOutlet>("head::after"); *// existing HttpClient registration* builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) }); *// register the in-memory Todo service so components can inject it* builder.Services.AddSingleton<TodoService>(); await builder.Build().RunAsync();

Key points:

* Use builder.Services.AddSingleton<TodoService>() (or AddScoped — both work in WASM, but AddSingleton is common for app-wide in-memory state).
* **Do not** call AddServerSideBlazor() in a WebAssembly app.
* **2) Make sure**TodoService**compiles (example)**

Put this in TodoApp/Services/TodoService.cs:

using System.Collections.Generic; using System.Linq; using TodoApp.Models; namespace TodoApp.Services { public class TodoService { private readonly List<TodoItem> \_todos = new List<TodoItem>(); private int \_nextId = 1; public List<TodoItem> GetAll() => \_todos; public void Add(string title) { \_todos.Add(new TodoItem { Id = \_nextId++, Title = title, IsDone = false }); } public void ToggleDone(int id) { var todo = \_todos.FirstOrDefault(t => t.Id == id); if (todo != null) todo.IsDone = !todo.IsDone; } public void Remove(int id) { var todo = \_todos.FirstOrDefault(t => t.Id == id); if (todo != null) \_todos.Remove(todo); } } }

And TodoApp/Models/TodoItem.cs (if you don't already have it):

namespace TodoApp.Models { public class TodoItem { public int Id { get; set; } public string Title { get; set; } = ""; public bool IsDone { get; set; } } }

* **3) Update your**Todo.razor**so the component knows the service type**

At the top of Pages/Todo.razor add the services namespace (or add it to \_Imports.razor):

@page "/todo" @using TodoApp.Models @using TodoApp.Services @inject TodoService TodoService

(If you prefer not to add the @using, you can inject with the fully qualified name:  
@inject TodoApp.Services.TodoService TodoService)

* **4) Restart the app**

1. Stop the running app in the terminal with **Ctrl + C**.
2. Restart in the project folder:

dotnet run

1. Open the URL shown (e.g. http://localhost:5195/todo).

* **Troubleshooting quick checklist**
* If you get CS0246 or "type or namespace not found" for TodoService, check the namespace line in TodoService.cs matches TodoApp.Services.
* If the app doesn't pick up changes, make sure you restarted dotnet run.
* If injection fails at runtime, double-check you registered the service in Program.cs **before** Build().

If you run these steps and still see an error, paste the exact error text here and I’ll fix it right away.

* **Du sa:**

I am ready for the next chapter

* **ChatGPT sa:**

Perfect 🎉 That means your in-memory ToDo app is alive and well. Time to make it **real** with a live database.

* **Chapter 4 — Create a Free Live Database (Supabase)**

Goal: set up a free hosted database where your ToDos can be stored and retrieved via an API. We’ll use **Supabase** (free tier, PostgreSQL + REST + auth).

* **4.1 Create a Supabase account**

1. Go to https://supabase.com.
2. Click **Start your project** and sign up (you can use GitHub or email).
3. Once inside, click **New project**.

* **4.2 Create your project**
* **Organization**: Supabase will create one for you (you can keep default).
* **Name**: e.g. todoapp.
* **Database password**: set a secure one (save it somewhere — you’ll need it if you connect directly).
* **Region**: choose something near you.
* **Pricing plan**: select **Free**.

Wait ~1–2 minutes while it provisions. ✅

* **4.3 Create the**todos**table**

1. In the left menu, go to **Table Editor** → **New table**.
2. Table name: todos.
3. Add columns:
   * id → type: int8, set as **Primary key**, enable **Identity** (auto-increment).
   * title → type: text, not null.
   * is\_done → type: bool, default = false.
   * (optional) created\_at → type: timestamp, default = now().

Click **Save**.

Now you have a live table like this:

| **id** | **title** | **is\_done** | **created\_at** |
| --- | --- | --- | --- |
| 1 | Learn Blazor | false | 2025-09-12 12:00… |

* **4.4 Get your API URL and Key**

1. In the left menu, go to **Project settings → API**.
2. Copy the **Project URL** (looks like https://xxxx.supabase.co).
3. Copy the **anon public key**.

We’ll use these in your Blazor app to talk to Supabase.

⚠️ **Keep the anon key safe** — it’s public but still should not be shared carelessly.

* **4.5 Test the API (optional but recommended)**

Install the **REST Client** extension in VS Code.  
Create a new file test.http with this:

GET https://YOUR\_PROJECT\_URL/rest/v1/todos apikey: YOUR\_ANON\_KEY

Click “Send Request” above the GET line.

* If it returns [], your setup is correct.
* If it errors, double-check URL and key.
* **✅ Recap**
* You created a Supabase project (Postgres database).
* You added a todos table.
* You obtained the API URL + anon key.
* You tested the REST endpoint.
* **Coming up (Chapter 5)**

We’ll replace your mock TodoService with a **Supabase service** that uses HttpClient in Blazor to call the REST API for Create/Read/Update/Delete.

👉 Do you want me to show you a **safe way to store your Supabase URL + key** in your Blazor project before we connect it, or just keep it simple and paste them directly into the code first?

* **Du sa:**

Yes, show me a safe way to store my url and key

* **ChatGPT sa:**

Great decision 🙌 — keeping keys out of your source code is a best practice.

With **Blazor WebAssembly**, secrets can’t be truly hidden (because all client-side code runs in the browser). But we can at least keep them organized and avoid hardcoding them everywhere. Later (if you want), we can move sensitive parts into a backend API.

For now, let’s do the **safe-for-beginner** approach: use appsettings.json and inject the values.

* **Storing Supabase URL + Key in Blazor WebAssembly**
* **1. Create a config file**

In the **wwwroot/** folder, create a file called appsettings.json:

{ "Supabase": { "Url": "https://YOUR\_PROJECT\_URL.supabase.co", "AnonKey": "YOUR\_ANON\_KEY" } }

⚠️ Replace YOUR\_PROJECT\_URL and YOUR\_ANON\_KEY with your actual values from Supabase.

* **2. Load config at startup**

Open Program.cs and add code to load the JSON:

using Microsoft.AspNetCore.Components.Web; using Microsoft.AspNetCore.Components.WebAssembly.Hosting; using TodoApp; using TodoApp.Services; using System.Net.Http.Json; var builder = WebAssemblyHostBuilder.CreateDefault(args); builder.RootComponents.Add<App>("#app"); builder.RootComponents.Add<HeadOutlet>("head::after"); *// Load config from wwwroot/appsettings.json* using var response = await new HttpClient().GetFromJsonAsync<Dictionary<string, Dictionary<string, string>>>( new Uri(builder.HostEnvironment.BaseAddress + "appsettings.json") ); var supabaseConfig = response["Supabase"]; var supabaseUrl = supabaseConfig["Url"]; var supabaseKey = supabaseConfig["AnonKey"]; *// Register HttpClient with Supabase base URL* builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(supabaseUrl), DefaultRequestHeaders = { { "apikey", supabaseKey }, { "Authorization", $"Bearer {supabaseKey}" } } }); *// Register our service* builder.Services.AddScoped<SupabaseTodoService>(); await builder.Build().RunAsync();

* **3. Create a Supabase service**

In Services/, add SupabaseTodoService.cs:

using System.Net.Http; using System.Net.Http.Json; using System.Threading.Tasks; using System.Collections.Generic; using TodoApp.Models; namespace TodoApp.Services { public class SupabaseTodoService { private readonly HttpClient \_http; public SupabaseTodoService(HttpClient http) { \_http = http; } public async Task<List<TodoItem>> GetTodos() { return await \_http.GetFromJsonAsync<List<TodoItem>>("rest/v1/todos?select=\*"); } public async Task AddTodo(string title) { var todo = new { title, is\_done = false }; await \_http.PostAsJsonAsync("rest/v1/todos", todo); } public async Task ToggleTodo(int id, bool isDone) { var update = new { is\_done = !isDone }; await \_http.PatchAsJsonAsync($"rest/v1/todos?id=eq.{id}", update); } public async Task DeleteTodo(int id) { await \_http.DeleteAsync($"rest/v1/todos?id=eq.{id}"); } } }

* **4. Update**TodoItem.cs**model**

Make sure your model matches Supabase column names:

namespace TodoApp.Models { public class TodoItem { public int Id { get; set; } public string Title { get; set; } = ""; public bool Is\_Done { get; set; } *// note: matches `is\_done` in DB* } }

* **5. Switch**Todo.razor**to use the new service**

At the top of Todo.razor:

@page "/todo" @using TodoApp.Models @using TodoApp.Services @inject SupabaseTodoService TodoService

Then replace your local-service methods with async calls:

@code { private List<TodoItem>? todos; private string newTodo = ""; protected override async Task OnInitializedAsync() { todos = await TodoService.GetTodos(); } private async Task AddTodo() { if (!string.IsNullOrWhiteSpace(newTodo)) { await TodoService.AddTodo(newTodo); todos = await TodoService.GetTodos(); newTodo = ""; } } private async Task ToggleTodo(TodoItem todo) { await TodoService.ToggleTodo(todo.Id, todo.Is\_Done); todos = await TodoService.GetTodos(); } private async Task RemoveTodo(int id) { await TodoService.DeleteTodo(id); todos = await TodoService.GetTodos(); } }

And update the markup:

<ul> @if (todos == null) { <li>Loading...</li> } else { @foreach (var todo in todos) { <li> <input type="checkbox" checked="@todo.Is\_Done" @onchange="() => ToggleTodo(todo)" /> <span style="text-decoration:@(todo.Is\_Done ? "line-through" : "none")">@todo.Title</span> <button @onclick="() => RemoveTodo(todo.Id)">❌</button> </li> } } </ul>

* **✅ Recap**
* Keys are stored in wwwroot/appsettings.json (keeps code clean).
* Program.cs loads them and configures HttpClient.
* You now have a SupabaseTodoService that talks to the live DB.
* Todo.razor shows real tasks from Supabase 🎉

⚠️ Remember: with Blazor WebAssembly, keys are still downloadable in the browser dev tools. That’s okay for learning + free projects. For production you’d usually hide keys in a backend API.

👉 Do you want me to continue with **Chapter 5: Wire the app fully to Supabase** (with the working CRUD UI), or do you want to pause and first try running what we just set up?

* **Du sa:**

I don't get any error messages in the code and when I open the website http://localhost:5195, I only get this error message: "An unhandled error has occurred. Reload" without any other error message in the browser.

* **ChatGPT sa:**

Good troubleshooting 👌 That “An unhandled error has occurred. Reload” screen in Blazor WebAssembly usually means **something threw an exception during startup** — often because the app failed while trying to load appsettings.json or parse it.

Let’s debug this step by step:

* **1. Check the browser dev console**

Open DevTools in your browser (**F12** → Console tab).  
You should see a red error message. Common ones are:

* Failed to fetch 'appsettings.json'
* Cannot deserialize ...
* NullReferenceException

That will give us the exact failure point.

* **2. Verify**appsettings.json

Make sure the file is in wwwroot/appsettings.json, not just in the root folder.  
It must be accessible at:  
👉 <http://localhost:5195/appsettings.json>

Try opening that URL in the browser.

* If you see your JSON, ✅ good.
* If you get a 404, it’s in the wrong place.
* **3. Simplify how we load config**

The earlier dictionary-based deserialization is fragile. Let’s make it cleaner using a typed class.

* **Step A: Create a config model**

In Models/SupabaseConfig.cs:

namespace TodoApp.Models { public class SupabaseConfig { public string Url { get; set; } = ""; public string AnonKey { get; set; } = ""; } }

* **Step B: Fix**Program.cs

Replace the config-loading part with this:

using Microsoft.AspNetCore.Components.Web; using Microsoft.AspNetCore.Components.WebAssembly.Hosting; using System.Net.Http.Json; using TodoApp; using TodoApp.Services; using TodoApp.Models; var builder = WebAssemblyHostBuilder.CreateDefault(args); builder.RootComponents.Add<App>("#app"); builder.RootComponents.Add<HeadOutlet>("head::after"); *// Load appsettings.json* var http = new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) }; var config = await http.GetFromJsonAsync<Dictionary<string, SupabaseConfig>>("appsettings.json"); var supabase = config["Supabase"]; *// Register HttpClient with Supabase base URL + headers* builder.Services.AddScoped(sp => { var client = new HttpClient { BaseAddress = new Uri(supabase.Url) }; client.DefaultRequestHeaders.Add("apikey", supabase.AnonKey); client.DefaultRequestHeaders.Add("Authorization", $"Bearer {supabase.AnonKey}"); return client; }); *// Register our service* builder.Services.AddScoped<SupabaseTodoService>(); await builder.Build().RunAsync();

* **Step C: Confirm your**appsettings.json

It should look exactly like this:

{ "Supabase": { "Url": "https://YOUR\_PROJECT.supabase.co", "AnonKey": "YOUR\_ANON\_KEY" } }

* **4. Rebuild & rerun**

1. Stop the app (Ctrl+C).
2. Run again:

dotnet run

1. Open [http://localhost:5195](http://localhost:5195/).

* **🔍 If it still crashes**
* Check DevTools console again.
* If you see Failed to fetch, your appsettings.json is not being served → confirm it’s in wwwroot/.
* If you see KeyNotFoundException: Supabase, maybe the JSON key is mis-typed (capitalization matters).

👉 Do you want me to show you a **minimal “just hardcode the URL + key first”** version (to confirm Supabase works), before we reintroduce config loading? That way we separate “is Supabase working” from “is config loading breaking things.”

* **Du sa:**

I'm getting the same error message. I really need help here.

* **ChatGPT sa:**

Thanks for sticking with it 👍 Let’s carefully debug and simplify until it works.

Right now you’re getting the **generic “Unhandled error”** screen. In Blazor WASM this almost always means:

* Startup code (Program.cs) threw before the app rendered.
* Or a JSON/config load failed silently.

So let’s do two things:

* **🔍 Step 1 — Confirm the error details**

1. Open your app in the browser (http://localhost:5195).
2. Press **F12** to open DevTools.
3. Go to the **Console** tab.  
   👉 What’s the **first red error message** there? (e.g. Failed to fetch appsettings.json, NullReferenceException, etc.)  
   This will tell us exactly where it’s failing.

* **🛠 Step 2 — Bypass config loading temporarily**

To rule out appsettings.json issues, let’s **hardcode** your Supabase keys directly in Program.cs. If this works, we know the DB connection is fine and the problem is just with config loading.

* **Replace your**Program.cs**with this minimal working version:**

using Microsoft.AspNetCore.Components.Web; using Microsoft.AspNetCore.Components.WebAssembly.Hosting; using TodoApp; using TodoApp.Services; var builder = WebAssemblyHostBuilder.CreateDefault(args); builder.RootComponents.Add<App>("#app"); builder.RootComponents.Add<HeadOutlet>("head::after"); *// 🔴 TEMP: hardcode Supabase settings here* var supabaseUrl = "https://YOUR\_PROJECT.supabase.co"; var supabaseKey = "YOUR\_ANON\_KEY"; *// Register HttpClient with Supabase* builder.Services.AddScoped(sp => { var client = new HttpClient { BaseAddress = new Uri(supabaseUrl) }; client.DefaultRequestHeaders.Add("apikey", supabaseKey); client.DefaultRequestHeaders.Add("Authorization", $"Bearer {supabaseKey}"); return client; }); *// Register service* builder.Services.AddScoped<SupabaseTodoService>(); await builder.Build().RunAsync();

👉 Replace the placeholders with your real Supabase URL and anon key.

* **🔁 Step 3 — Rebuild & run**

1. Stop the app (Ctrl + C).
2. Run again:

dotnet run

1. Visit http://localhost:5195/todo.

* If it **works now** → the error was caused by config loading (appsettings.json not being found/parsed).
* If it **still crashes** → the error is in the Supabase service (SupabaseTodoService) or the way the UI calls it.

👉 Can you try this “hardcoded” version and tell me:

1. Does the site load at all (even if the ToDo list is empty)?
2. What error shows in the **browser console** (DevTools → Console)?

That info will let me either fix the config issue, or debug the service code next.

* **Du sa:**

I found this error message when I pressed F12: hook.js:608 ManagedError: One or more errors occurred. (Cannot provide a value for property 'TodoService' on type 'TodoApp.App'. There is no registered service of type 'TodoApp.Services.TodoService'.)

* **ChatGPT sa:**

Perfect 👌 you found the real issue — thank you for checking the console!

The error says:

Cannot provide a value for property 'TodoService' on type 'TodoApp.App'.

There is no registered service of type 'TodoApp.Services.TodoService'.

That means Blazor is trying to inject a service named TodoService, but only SupabaseTodoService was registered in Program.cs.

So the problem is **a mismatch between what’s registered vs. what you inject**.

* **✅ How to fix**
* **Option A — If you want Supabase service**

In Program.cs you registered:

builder.Services.AddScoped<SupabaseTodoService>();

👉 That means in your Razor page (Todo.razor), you must inject exactly this:

@inject SupabaseTodoService TodoService

⚠️ Not TodoApp.Services.TodoService, but SupabaseTodoService.

* **Option B — If you intended to use a local in-memory TodoService**

Earlier we created an in-memory service for testing (before Supabase). If you still have something like:

public class TodoService { *// local storage for todos* }

Then you need to **register it** in Program.cs:

builder.Services.AddScoped<TodoService>();

And inject it with:

@inject TodoService TodoService

* **🔍 What to do next**

1. Open your Todo.razor.
2. Check the top line where you inject the service.
   * If it says @inject TodoService TodoService → change it to @inject SupabaseTodoService TodoService.
   * OR add the missing registration for TodoService in Program.cs.
3. Re-run the app.

👉 Do you want me to show you the **correct final version of**Todo.razor that works with SupabaseTodoService (live database), so you don’t have the mismatch anymore?

* **Du sa:**

Now I can open the ToDo webpage, but when I enter en task and click "Add" nothing happends. I press F12 and I get access to this error message: Failed to load resource: the server responded with a status of 401 ()

* **ChatGPT sa:**

Nice work — now we’re getting somewhere 🚀

That **401 Unauthorized** error means your app is reaching Supabase, but Supabase is **rejecting the request**. This usually happens because of one of these issues:

* **🔍 Common causes of 401 with Supabase REST API**

1. **Wrong key used**
   * You must use the **anon public key**, not the service key.
   * In Supabase dashboard: Settings → API → **Project API keys** → use anon public.
2. **Missing or incorrect headers**  
   Every REST request needs both:

apikey: YOUR\_ANON\_KEY Authorization: Bearer YOUR\_ANON\_KEY

1. **Row Level Security (RLS) blocking requests**
   * By default, Supabase enables RLS on every table.
   * That means your anon key can’t read/write unless you add **policies**.

* **✅ Step 1 — Confirm the headers in your code**

In your Program.cs, you should have:

builder.Services.AddScoped(sp => { var client = new HttpClient { BaseAddress = new Uri(supabaseUrl) }; client.DefaultRequestHeaders.Add("apikey", supabaseKey); client.DefaultRequestHeaders.Add("Authorization", $"Bearer {supabaseKey}"); return client; });

If that’s correct, headers are being sent.

* **✅ Step 2 — Check your Supabase table + policies**

Go to your Supabase project → SQL Editor, and run:

*-- Allow anyone (with anon key) to select, insert, update, delete* *-- Replace todos with your actual table name* create policy "Allow anon read" on todos for select using (true); create policy "Allow anon insert" on todos for insert with check (true); create policy "Allow anon update" on todos for update using (true); create policy "Allow anon delete" on todos for delete using (true);

And make sure RLS is **enabled** on the todos table (policies only work when RLS is on).

* **✅ Step 3 — Verify table structure**

Your table todos in Supabase should look like this:

| **Column** | **Type** | **Notes** |
| --- | --- | --- |
| id | int8 | Primary key, auto-increment |
| title | text | Task text |
| is\_done | boolean | Default = false |

* **🔁 Step 4 — Re-run**
* Restart your app (dotnet run).
* Open /todo.
* Add a task.

It should now **insert into Supabase** and refresh the list 🎉

👉 Quick question: Did you already create your todos table in Supabase, or should I give you the exact SQL snippet to create it from scratch?

* **Du sa:**

It is fixed now. You can to the next step.

* **ChatGPT sa:**

🎉 Awesome — you now have a **Blazor WebAssembly ToDo app** hooked up to a **live Supabase database**! That’s a huge milestone.

Now let’s move on to the **next step: Chapter 6**.

* **📘 Chapter 6 — Publishing Your App to a Free Web Host**

Now that it works locally, let’s make it available online with a real web URL.  
We’ll use **GitHub Pages + GitHub Actions**, since it’s free and works well with Blazor WebAssembly.

* **🔹 Step 1 — Prepare your app for static hosting**

Blazor WASM builds into static files (HTML, JS, CSS, DLLs), so it’s perfect for GitHub Pages.

In your project folder, run:

dotnet publish -c Release -o publish

This creates a publish/wwwroot folder with everything needed to host.

* **🔹 Step 2 — Push your project to GitHub**

1. Create a new repo on GitHub (e.g. TodoApp).
2. In VS Code terminal, run:

git init git remote add origin https://github.com/YOUR\_USERNAME/TodoApp.git git add . git commit -m "Initial commit" git branch -M main git push -u origin main

* **🔹 Step 3 — Add GitHub Action for Blazor deploy**

Inside your project, create a folder: .github/workflows/

Inside it, create a file: deploy.yml

Paste this:

name: Deploy Blazor App to GitHub Pages on: push: branches: [ "main" ] *# deploy when main branch updates* jobs: build-and-deploy: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 8.0.x - name: Publish run: dotnet publish -c Release -o publish - name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: ./publish/wwwroot

Commit and push this file:

git add .github/workflows/deploy.yml git commit -m "Add GitHub Pages deploy action" git push

* **🔹 Step 4 — Enable GitHub Pages**

1. Go to your repo on GitHub.
2. Settings → Pages →
   * Source: GitHub Actions.

Once the workflow runs (1–2 minutes), your app will be live at:  
👉 https://YOUR\_USERNAME.github.io/TodoApp/

* **🔹 Step 5 — Fix base path (important!)**

Blazor needs to know the app is hosted in /TodoApp/ instead of root /.

Open wwwroot/index.html, find:

<base href="/" />

Change it to:

<base href="/TodoApp/" />

(replace TodoApp with your repo name).

Rebuild + push again, and the site should load fine.

✅ At this point:

* Your Blazor ToDo app is live on GitHub Pages.
* It talks to Supabase just like locally.
* You have CI/CD (every push auto-deploys).

👉 Do you want me to walk you through adding **custom styling + UI improvements** in **Chapter 7**, or would you prefer we first make sure your **GitHub Pages deployment** works before moving on?

* **Du sa:**

Now I get this message on my github.io page: 404 There isn't a GitHub Pages site here. And there is nothing in the

* **ChatGPT sa:**

Good catch 👍 — that means your **GitHub Pages deploy workflow didn’t push any site files yet**. Let’s carefully fix this.

* **🔍 Step 1 — Confirm the workflow ran**

1. Go to your GitHub repo → **Actions** tab.
2. Do you see a workflow run named *“Deploy Blazor App to GitHub Pages”*?
   * If it failed → click it, copy the error message.
   * If it never ran → maybe the workflow file name or branch trigger is wrong.

* **🔍 Step 2 — Verify the build output path**

In Blazor WebAssembly, dotnet publish puts the static files inside:

publish/wwwroot/

But in our GitHub Action, we set:

publish\_dir: ./publish/wwwroot

If you see an empty site, it usually means the publish\_dir was wrong or not generated.

* **🛠 Step 3 — Update workflow to use correct path**

Try this version of .github/workflows/deploy.yml:

name: Deploy Blazor App to GitHub Pages on: push: branches: [ "main" ] jobs: build-and-deploy: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 8.0.x - name: Publish app run: dotnet publish -c Release -o release - name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot

Notice I switched the output folder to release (to avoid confusion) and deploy from release/wwwroot.

* **🔍 Step 4 — Enable GitHub Pages in repo settings**

1. Go to repo → **Settings** → **Pages**.
2. Under **Source**, select Deploy from a branch → gh-pages branch.  
   (The action creates a gh-pages branch with the published site.)

After a minute, your site will appear at:  
👉 https://YOUR\_USERNAME.github.io/TodoApp/

* **🔍 Step 5 — Check**<base href>

Open your project’s wwwroot/index.html.  
Make sure it matches your repo name:

<base href="/TodoApp/" />

(Replace TodoApp with the actual repo name.)

✅ After the workflow runs successfully, you’ll see a **gh-pages branch** in your repo containing the built site.

👉 Do you want me to give you a **test minimal repo example** (just “Hello World” Blazor WASM) that you can fork and deploy first, so we’re sure GitHub Pages setup is working before you push the full ToDo app?

* **Du sa:**

Here is the error message: The current .NET SDK does not support targeting .NET 9.0. Either target .NET 8.0 or lower, or use a version of the .NET SDK that supports .NET 9.0.

* **ChatGPT sa:**

💡 Aha — that explains why your **GitHub Action build fails** → your project is targeting **.NET 9.0**, but the GitHub Action is installing only **.NET 8.0**.

We have two ways to fix this:

* **✅ Option 1 — Stick with .NET 9.0**

Update your workflow so it installs .NET 9.0 on GitHub’s build server.

Change this section in .github/workflows/deploy.yml:

- name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 9.0.x *# 👈 use 9 instead of 8*

That way, GitHub Actions installs the .NET 9 SDK and your app will build fine.

* **✅ Option 2 — Downgrade your project to .NET 8**

If you want to stay on the latest **LTS** (long-term support), you can target .NET 8 instead of 9.

1. Open your .csproj file (e.g. TodoApp.csproj).
2. Look for:

<TargetFramework>net9.0</TargetFramework>

1. Change it to:

<TargetFramework>net8.0</TargetFramework>

1. Rebuild locally with:

dotnet build

If it works, commit and push again → the GitHub Action with dotnet-version: 8.0.x will succeed.

⚖️ **Which should you choose?**

* If you want maximum stability and easier hosting: go with **.NET 8 (LTS)**.
* If you want to stay cutting edge: keep **.NET 9**, but make sure your workflow installs 9.0.x.

👉 Do you want me to rewrite the full **deploy.yml** for you with .NET 9 support so you can just copy–paste it?

* **Du sa:**

New error message when I changed it to 9.0.x in the deploy.yml file and commited: 1 error and 1 warning build-and-deploy Action failed with "The process '/usr/bin/git' failed with exit code 128" build-and-deploy The "--output" option isn't supported when building a solution. Specifying a solution-level output path results in all projects copying outputs to the same directory, which can lead to inconsistent builds.

* **ChatGPT sa:**

Got it 👍 thanks for pasting the exact error. This is a **common trap** with Blazor GitHub Pages deployment:

* The GitHub Action is running dotnet publish on the **solution (**.sln**) file**, not on your **project (**.csproj**) file**.
* When you specify -o release, dotnet complains because a solution can’t have one output directory.
* **✅ Fix: Point**dotnet publish**to your Blazor project**

Let’s say your Blazor project file is:

TodoApp/TodoApp.csproj

Update your workflow so it **publishes that project explicitly**:

name: Deploy Blazor App to GitHub Pages on: push: branches: [ "main" ] jobs: build-and-deploy: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 9.0.x *# or 8.0.x if you target LTS* - name: Publish Blazor app run: dotnet publish TodoApp/TodoApp.csproj -c Release -o release - name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot

* **🔍 Double-check paths**
* If your repo root looks like this:

/TodoApp.sln

/TodoApp/TodoApp.csproj

Then the above path (TodoApp/TodoApp.csproj) is correct.

* If your .csproj is directly at repo root, just use:

dotnet publish TodoApp.csproj -c Release -o release

* **✅ After pushing this fix**

1. Commit & push the updated workflow.
2. Wait for GitHub Actions to run.
3. A new branch gh-pages should appear in your repo → that’s your deployed site.
4. Go to **Settings → Pages** → set source = gh-pages branch.

Then your site will load at:  
👉 https://YOUR\_USERNAME.github.io/TodoApp/

👉 Do you want me to also show you how to **automatically set**<base href> in index.html so you don’t need to hardcode /TodoApp/ when deploying to GitHub Pages?

* **Du sa:**

New error: Annotations 1 error build-and-deploy Process completed with exit code 1.

* **ChatGPT sa:**

Thanks for the update 🙏 An exit code 1 without more info usually means the workflow failed during the dotnet publish step — but GitHub hides the details unless we expand the logs. Let’s carefully track it down.

* **🔍 Step 1 — Check the detailed logs**

1. Go to your repo → **Actions** tab.
2. Click the failed run.
3. Expand the **Publish Blazor app** step.  
   👉 Copy the **full error message** you see there.

That will tell us exactly why dotnet publish failed.

* **🛠 Step 2 — Common fixes**

Here are the usual suspects when exit code 1 happens:

* **A. Wrong**.csproj**path**

If your project isn’t at TodoApp/TodoApp.csproj, the path in workflow must match.

* If your .csproj is at repo root → use:

run: dotnet publish -c Release -o release

* If it’s in a subfolder → keep the relative path.
* **B. Target framework mismatch**

Make sure your .csproj has:

<TargetFramework>net9.0</TargetFramework>

and workflow uses:

dotnet-version: 9.0.x

(or 8.0.x if you downgraded).

* **C. Missing workload (Blazor WASM)**

Sometimes GitHub runner doesn’t have the **WebAssembly workload**. You can force-install it:

- name: Install WASM workload run: dotnet workload install wasm-tools

Add this step **before** the dotnet publish step.

* **✅ Suggested workflow with safety net**

Here’s a more robust version you can try:

name: Deploy Blazor App to GitHub Pages on: push: branches: [ "main" ] jobs: build-and-deploy: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 9.0.x *# or 8.0.x if you switched* - name: Install WASM workload run: dotnet workload install wasm-tools - name: Publish Blazor app run: dotnet publish TodoApp/TodoApp.csproj -c Release -o release - name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot

👉 Can you expand your **Publish Blazor app** step in the GitHub Action logs and paste the error text here? That will let me confirm whether it’s a path problem, workload issue, or something else.

* **Du sa:**

That was the hole error message. When I run it in the VSC Terminal it was success: dotnet publish TodoApp.csproj -c Release -o release Restore complete (0,5s) TodoApp succeeded (15,5s) → release\ Build succeeded in 16,9s

* **ChatGPT sa:**

Ahh, thanks — that’s very helpful 👍

So:

* ✅ dotnet publish works **locally** in VS Code.
* ❌ It fails in **GitHub Actions** with only exit code 1.
* No extra output = GitHub isn’t showing the error unless we force it to.

That means the problem isn’t your project — it’s the **GitHub workflow environment**.

* **🔍 Why it happens**

GitHub’s runner doesn’t always have the right workloads for Blazor WebAssembly. By default, dotnet publish tries to use the **WASM tools workload**. If it’s missing, the runner just fails. Locally you have it installed, but in CI you don’t.

* **✅ Fix: Install the Blazor WebAssembly workload in the Action**

Update your .github/workflows/deploy.yml like this:

name: Deploy Blazor App to GitHub Pages on: push: branches: [ "main" ] jobs: build-and-deploy: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Setup .NET uses: actions/setup-dotnet@v3 with: dotnet-version: 9.0.x *# or 8.0.x if you switched* - name: Install WASM workload run: dotnet workload install wasm-tools - name: Publish Blazor app run: dotnet publish TodoApp.csproj -c Release -o release - name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot

* **🔁 What to do next**

1. Commit & push this updated workflow.
2. Go to **Actions** tab → watch the new run.
3. It should now successfully install the WASM workload, then publish.
4. A new branch gh-pages will appear with your static site.
5. Go to **Settings → Pages → Source = gh-pages branch**.

Your site will then load at:  
👉 https://YOUR\_USERNAME.github.io/TodoApp/

⚡ Question: Do you want me to also show you how to **automatically fix**<base href> so you don’t need to edit index.html every time (GitHub Pages requires /YourRepoName/)?

* **Du sa:**

New error: Annotations 1 error build-and-deploy Action failed with "The process '/usr/bin/git' failed with exit code 128"

* **ChatGPT sa:**

Got it — thanks 🙏

Now the error is no longer about dotnet publish, but about the **deploy step** using peaceiris/actions-gh-pages.

Action failed with "The process '/usr/bin/git' failed with exit code 128"

That means GitHub Pages Action tried to push to the gh-pages branch, but something went wrong with Git. The **most common reasons** are:

* **🔍 Common causes of exit code 128 in**gh-pages**deploy**

1. **Repo is private** but you’re using the default GITHUB\_TOKEN without giving it permissions: write.
2. **gh-pages branch already exists** in a corrupted state (e.g. created manually, force-pushed).
3. **Missing user/email config** for Git in the Action.
4. **Submodules or nested repos** (less likely in your case).

* **✅ Fix**
* **Step 1: Ensure GITHUB\_TOKEN has write permissions**

In your workflow, add this at the top (just below jobs:):

permissions: contents: write

So the full job looks like:

jobs: build-and-deploy: runs-on: ubuntu-latest permissions: contents: write

* **Step 2: Clean the**gh-pages**branch (if it exists)**

If a broken gh-pages branch exists, delete it:

1. Go to your repo → **Branches**.
2. Delete the gh-pages branch.

The Action will recreate it cleanly.

* **Step 3: Add identity to the Action (optional but recommended)**

Sometimes git fails without a username/email. Add:

- name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot user\_name: github-actions[bot] user\_email: github-actions[bot]@users.noreply.github.com

✅ With these three changes:

* GitHub token has write access.
* Old broken branch is gone.
* Git has an identity for commits.

The action should now push successfully to gh-pages.

👉 Do you want me to give you the **full corrected workflow file** (with all these fixes combined) so you can just copy–paste it?

* **Du sa:**

Delete the `gh-pages` branch.

I only have one branch under the Branches menu tab and that is named "main". Shall I delete this?

* **ChatGPT sa:**

⚠️ No — **don’t delete**main.  
That’s your source branch where your real code lives.

The instruction I gave was only for a gh-pages branch (if it existed). Since you only have main, that’s fine — the Action will create gh-pages for you on the first successful run.

So the next steps for you are:

1. **Do not delete**main**.**
2. Update your workflow with these fixes:

jobs: build-and-deploy: runs-on: ubuntu-latest permissions: contents: write

and in the deploy step:

- name: Deploy to GitHub Pages uses: peaceiris/actions-gh-pages@v3 with: github\_token: ${{ secrets.GITHUB\_TOKEN }} publish\_dir: release/wwwroot user\_name: github-actions[bot] user\_email: github-actions[bot]@users.noreply.github.com

1. Commit → push → check Actions.  
   This time, if everything is correct, the Action should create the new gh-pages branch for you.

👉 Do you want me to paste the **entire**deploy.yml**file** with these fixes already included, so you can replace yours directly?

* **Du sa:**

Now I got green light and it is ok. But I got this error when i go to the URL: 404 There isn't a GitHub Pages site here

* **ChatGPT sa:**

Perfect — the Action is now green ✅, so publishing worked.  
The **404** just means GitHub Pages isn’t yet serving the branch. That’s the last step.

* **✅ Step 1 — Turn on GitHub Pages**

1. Go to your repo on GitHub.
2. Click **Settings** → **Pages** (left sidebar).
3. Under **Source**, select:
   * Branch: gh-pages
   * Folder: / (root)
4. Click **Save**.

GitHub will then prepare the site (can take 1–2 minutes).

* **✅ Step 2 — Fix the base URL for Blazor**

By default, Blazor apps expect to be hosted at /.  
But on GitHub Pages, your app will live under:

https:*//USERNAME.github.io/REPO-NAME/*

To make Blazor work correctly, you need to tell it the base path.

1. Open your wwwroot/index.html.
2. Change the <base href="/"> line to:

<base href="/REPO-NAME/">

(Replace REPO-NAME with the name of your repository, e.g. TodoApp.)

1. Rebuild & publish (git commit + push) → the workflow will redeploy with the fixed base path.

* **✅ Step 3 — Test**
* Wait ~2–3 minutes after the new push.
* Go to:
* https:*//USERNAME.github.io/REPO-NAME/*
* You should now see your Blazor ToDo app 🎉

👉 Do you want me to show you a trick to **avoid manually editing**<base href> every time (so GitHub Pages gets the correct path automatically from your workflow)?

* **Du sa:**

:) Now the page is up an running, but I am missing the left menubar. Only text I see is this: About Hello, world! Welcome to your new app.

* **ChatGPT sa:**

Nice 🎉 your Blazor app is live on GitHub Pages!

The missing **left navigation menu** is a common Blazor-on-GitHub Pages issue — usually caused by **CSS or static files not being loaded** because of the <base href> change.

* **🔍 Why it happens**

Blazor loads its CSS, JS, and icons from paths like /css/app.css.  
On GitHub Pages, your app lives at:

/REPO-NAME/css/app.css

If <base href> isn’t set correctly, the browser can’t find the files → layout breaks (menu disappears, app looks unstyled).

* **✅ Fix**

1. Open wwwroot/index.html.
2. Make sure the base tag looks like this:

<base href="/TodoApp/">

👉 Replace TodoApp with your **repo name** (must match exactly, including case).

1. Rebuild & push again:

dotnet publish TodoApp.csproj -c Release -o release git add . git commit -m "Fix base href for GitHub Pages" git push

The GitHub Action will redeploy.

1. Refresh your site at:

https:*//USERNAME.github.io/TodoApp/*

Now CSS and the **left menu bar** should appear. 🎨

⚡ Bonus: You can make this automatic by adding this line to your .csproj:

<PropertyGroup> <StaticWebAssetBasePath>TodoApp</StaticWebAssetBasePath> </PropertyGroup>

That way, Blazor always uses /TodoApp/ as its base path, no matter where you deploy.

👉 Do you want me to show you how to set up that **automatic base path fix in your .csproj**, so you don’t have to edit index.html every time?

* **Du sa:**

The index.html file is correct with <base href="/TodoApp/">, so I can't fint out whats wrong.

* **ChatGPT sa:**

Got it 👍 if your <base href="/TodoApp/"> is already correct, then the missing **left nav menu** means your Blazor layout isn’t being applied. That usually comes down to one of these:

* **🔍 Possible causes**

1. **Wrong layout reference** → Your App.razor may not be pointing to the MainLayout.
2. **NavMenu missing from layout** → The MainLayout.razor file may not include <NavMenu />.
3. **CSS not loading** → If wwwroot/css/bootstrap/bootstrap.min.css or app.css isn’t loading (check in **F12 → Network tab**), the page will render plain text.
4. **Case sensitivity** → GitHub Pages is case-sensitive (Linux server). If your repo folder is layout/NavMenu.razor but the file is referenced as Layout/NavMenu.razor, it will work locally on Windows but fail online.

* **✅ Steps to debug**

1. Open **F12 → Network tab** on your GitHub Pages site.
   * Look for **404 errors** on CSS or DLL files.
   * If you see bootstrap.min.css or blazor.webassembly.js failing → it’s a static file path problem.
2. Check your MainLayout.razor (should be in Shared/ or Layout/).  
   It should look something like this:

@inherits LayoutComponentBase <div class="page"> <div class="sidebar"> <NavMenu /> </div> <main> <div class="top-row px-4"> <a href="https://learn.microsoft.com/aspnet/core/" target="\_blank">About</a> </div> <article class="content px-4"> @Body </article> </main> </div>

👉 If <NavMenu /> is missing here, the sidebar won’t render.

1. Check NavMenu.razor. It should look like:

<div class="nav-scrollable"> <ul class="nav flex-column"> <li class="nav-item px-3"> <NavLink class="nav-link" href=""> <span class="oi oi-home" aria-hidden="true"></span> Home </NavLink> </li> <li class="nav-item px-3"> <NavLink class="nav-link" href="about"> <span class="oi oi-info" aria-hidden="true"></span> About </NavLink> </li> </ul> </div>

* **⚡ Next Step**

Could you do this for me:

* Open your site in the browser, press **F12 → Network tab**.
* Refresh the page.
* Tell me if you see any red 404 errors for .css or .js files.

That will tell us if it’s a **static file path issue** or a **layout component issue**.

👉 Do you want me to also paste a **working MainLayout.razor + NavMenu.razor** combo you can drop in to test?

Top of Form

Bottom of Form

ChatGPT kan gjøre feil. Sjekk viktig informasjon.