**Migrating a VB.NET Win Forms   
Application to C#/Blazor Web Application**

**Using ChatGTP 4.0  
Repeatable Steps**

**Prepared by Gregory Leake for GapMobilize**

**March 1, 2024**Note Please See Accompanying Presentation with Key Learnings/Timings Extracted

Contents

[Understanding Form Mapping from VB.NET WinForms App: To Target Blazor .NET 8/C# Version, Modern Web App 14](#_Toc160090051)

[I have not done any serious coding for 14 years, on .NET/C# or VB.NET. I am a good “proxy” for a large customer base with similar skills. 14](#_Toc160090052)

[My notes are detailed: As I am “new” on these latest MSFT technologies. 14](#_Toc160090053)

[VB.NET WinForms 14](#_Toc160090054)

[[MainNav] 14](#_Toc160090055)

[[Login] 15](#_Toc160090056)

[[Quotes] 15](#_Toc160090057)

[[GetQuotes/Button🡪VB DataGridView] 16](#_Toc160090058)

[Buy a Stock [Click DataGridView Row] (if logged in, validation not shown) 17](#_Toc160090059)

[Holdings, DataGridView (Logged In, Validation not shown) 18](#_Toc160090060)

[Sell a Stock 18](#_Toc160090061)

[AccountForm 19](#_Toc160090062)

[Phase 1: Get Basic Instructions and Steps for Migration 20](#_Toc160090063)

[ChatGTP Interaction 1 20](#_Toc160090064)

[Input 20](#_Toc160090065)

[Output 20](#_Toc160090066)

[Notes 22](#_Toc160090067)

[ChatGTP Interaction 2 22](#_Toc160090068)

[Input 22](#_Toc160090069)

[Output 22](#_Toc160090070)

[Notes 24](#_Toc160090071)

[ChatGTP Extra Interaction – What if Choose Node.JS/Angular as the Target, Not Blazor/C#? 25](#_Toc160090072)

[Input 25](#_Toc160090073)

[Output 25](#_Toc160090074)

[Notes 26](#_Toc160090075)

[Phase 2 – Starting the Migration with Business Logic 27](#_Toc160090076)

[ChatGTP Interaction 3 - Direct Code Migration Business Logic Tier 29](#_Toc160090077)

[Input 29](#_Toc160090078)

[Output 32](#_Toc160090079)

[Notes 35](#_Toc160090080)

[ChatGTP Interaction 4 - Direct Code Migration Business Logic Tier 36](#_Toc160090081)

[Input 36](#_Toc160090082)

[Output 36](#_Toc160090083)

[Notes 37](#_Toc160090084)

[Phase 3 – The First Mixed Business and Data Logic Classes 38](#_Toc160090085)

[ChatGTP Interaction 5 – Direct Code Migration Mixed Business and Data Tier 38](#_Toc160090086)

[Input 38](#_Toc160090087)

[Output 40](#_Toc160090088)

[Notes 45](#_Toc160090089)

[ChatGTP Interaction 6a – Output Code Optimization/Fixup 45](#_Toc160090090)

[Input 45](#_Toc160090091)

[Output 45](#_Toc160090092)

[Notes 46](#_Toc160090093)

[ChatGTP Interaction 6b - Output Code Optimization/Fixup 47](#_Toc160090094)

[Input 47](#_Toc160090095)

[Output 47](#_Toc160090096)

[Notes 49](#_Toc160090097)

[ChatGTP Interaction 7 - Output Code Optimization/Fixup 49](#_Toc160090098)

[Input 49](#_Toc160090099)

[Output 49](#_Toc160090100)

[Notes 51](#_Toc160090101)

[ChatGTP Interaction 8 - Output Code Optimization/Fixup 51](#_Toc160090102)

[Input 51](#_Toc160090103)

[Output 51](#_Toc160090104)

[Notes 52](#_Toc160090105)

[ChatGTP Interaction 9 – Output Code Optimization/Fixup 52](#_Toc160090106)

[Input 52](#_Toc160090107)

[Output 53](#_Toc160090108)

[Notes 56](#_Toc160090109)

[ChatGTP Interaction 10 56](#_Toc160090110)

[Input 56](#_Toc160090111)

[Output 56](#_Toc160090112)

[ChatGTP Interaction 11 56](#_Toc160090113)

[Input 56](#_Toc160090114)

[Output 56](#_Toc160090115)

[ChatGTP Interaction 12: - Output Code Optimization/Fixup 56](#_Toc160090116)

[Input 56](#_Toc160090117)

[Output 57](#_Toc160090118)

[Notes 59](#_Toc160090119)

[ChatGTP Interaction 13 – Direct Code Migration (Config) 64](#_Toc160090120)

[Input 64](#_Toc160090121)

[Output 65](#_Toc160090122)

[Notes 66](#_Toc160090123)

[ChatGTP Interaction 14 - Direct Code Migration (Config) 67](#_Toc160090124)

[Input 67](#_Toc160090125)

[Output 68](#_Toc160090126)

[Notes 70](#_Toc160090127)

[ChatGTP Interaction 15 - Output Code Optimization/Fixup 70](#_Toc160090128)

[Input 70](#_Toc160090129)

[Output 70](#_Toc160090130)

[Notes 71](#_Toc160090131)

[ChatGTP Interaction 16: Output Code Optimization/Fixup 72](#_Toc160090132)

[Input 72](#_Toc160090133)

[Output 72](#_Toc160090134)

[Notes 74](#_Toc160090135)

[ChatGTP Interaction 17: 75](#_Toc160090136)

[Input 75](#_Toc160090137)

[Output 75](#_Toc160090138)

[Notes 75](#_Toc160090139)

[ChatGTP Interaction 18 – Output Code Optimization/Fixup 75](#_Toc160090140)

[Input 75](#_Toc160090141)

[Output 76](#_Toc160090142)

[Notes 77](#_Toc160090143)

[ChatGTP Interaction 19: 77](#_Toc160090144)

[Input 77](#_Toc160090145)

[Output 78](#_Toc160090146)

[Notes 78](#_Toc160090147)

[ChatGTP Interaction 20 – Direct Code Migration Business Logic Tier 78](#_Toc160090148)

[Input 78](#_Toc160090149)

[Output 78](#_Toc160090150)

[Notes 83](#_Toc160090151)

[ChatGTP Interaction 21 - Direct Code Migration Business Logic Tier 84](#_Toc160090152)

[Input 84](#_Toc160090153)

[Output 85](#_Toc160090154)

[Notes 90](#_Toc160090155)

[ChatGTP Interaction 22 – Output Code Optimization/Fixup 90](#_Toc160090156)

[Input 91](#_Toc160090157)

[Output 91](#_Toc160090158)

[Notes 93](#_Toc160090159)

[Phase 4 – Revisiting the Data Tier and Migrating In a Better Way – Starting From the DBs Themselves 94](#_Toc160090160)

[ChatGTP Interaction 23 - Direct Code Generation Data Tier Classes from DB DDL 94](#_Toc160090161)

[Input 94](#_Toc160090162)

[Output 94](#_Toc160090163)

[Notes 96](#_Toc160090164)

[ChatGTP Interaction 24 – Output Code Optimization/Fixup 97](#_Toc160090165)

[Input 97](#_Toc160090166)

[Output 98](#_Toc160090167)

[Notes 100](#_Toc160090168)

[ChatGTP Interaction 25 - - Direct Code Generation Data Tier Classes from DB DDL 100](#_Toc160090169)

[Input 100](#_Toc160090170)

[Output 100](#_Toc160090171)

[Notes 102](#_Toc160090172)

[ChatGTP Interaction 26 – Output Code Optimization/Fixup 103](#_Toc160090173)

[Input 103](#_Toc160090174)

[Output 103](#_Toc160090175)

[Notes 105](#_Toc160090176)

[ChatGTP Interaction 27 - Direct Code Generation Data Tier Classes from DB DDL 105](#_Toc160090177)

[Input 105](#_Toc160090178)

[Output 105](#_Toc160090179)

[Notes 107](#_Toc160090180)

[ChatGTP Interaction 28 – Output Code Optimization/Fixup 108](#_Toc160090181)

[Input 108](#_Toc160090182)

[Output 108](#_Toc160090183)

[Notes 110](#_Toc160090184)

[ChatGTP Interaction 29 111](#_Toc160090185)

[Input 111](#_Toc160090186)

[Output 111](#_Toc160090187)

[Notes 113](#_Toc160090188)

[ChatGTP Interaction 30 - Output Code Optimization/Fixup 113](#_Toc160090189)

[Input 113](#_Toc160090190)

[Output 113](#_Toc160090191)

[Notes 115](#_Toc160090192)

[ChatGTP Interaction 31 – Output Code Optimization/Fixup 115](#_Toc160090193)

[Input 115](#_Toc160090194)

[Output 116](#_Toc160090195)

[Notes 117](#_Toc160090196)

[ChatGTP Interaction 32 - Direct Code Generation Data Tier Classes from DB DDL 117](#_Toc160090197)

[Input 117](#_Toc160090198)

[Output 117](#_Toc160090199)

[Notes 120](#_Toc160090200)

[ChatGTP Interaction 33 – Output Code Optimization/Fixup 120](#_Toc160090201)

[Input 120](#_Toc160090202)

[Output 120](#_Toc160090203)

[Notes 122](#_Toc160090204)

[Phase 5 – Migrating the UI/VB.NET Win Forms to Blazor UI 127](#_Toc160090205)

[ChatGTP Interaction 34 127](#_Toc160090206)

[Input 127](#_Toc160090207)

[Output 127](#_Toc160090208)

[Notes 127](#_Toc160090209)

[ChatGTP Interaction 35 - Direct Code Migration UI Tier 127](#_Toc160090210)

[Input 127](#_Toc160090211)

[Output 129](#_Toc160090212)

[Notes 132](#_Toc160090213)

[ChatGTP Interaction 36 – Output Code Optimization/Fixup 132](#_Toc160090214)

[Input 132](#_Toc160090215)

[Output 132](#_Toc160090216)

[Notes 135](#_Toc160090217)

[ChatGTP Interaction 37 – Output Code Optimization/Fixup 135](#_Toc160090218)

[Input 135](#_Toc160090219)

[Output 135](#_Toc160090220)

[Notes 137](#_Toc160090221)

[ChatGTP Interaction 38 – Output Code Optimization/Fixup 137](#_Toc160090222)

[Input 137](#_Toc160090223)

[Output 138](#_Toc160090224)

[Notes 139](#_Toc160090225)

[ChatGTP Interaction 39 – Output Code Optimization/Fixup 139](#_Toc160090226)

[Input 139](#_Toc160090227)

[Output 141](#_Toc160090228)

[Notes 143](#_Toc160090229)

[ChatGTP Interaction 40 - Output Code Optimization/Fixup 143](#_Toc160090230)

[Input 143](#_Toc160090231)

[Output 144](#_Toc160090232)

[Notes 145](#_Toc160090233)

[ChatGTP Interaction 41 – Direct Code Migration UI Tier 146](#_Toc160090234)

[Input 146](#_Toc160090235)

[Output 148](#_Toc160090236)

[Notes 151](#_Toc160090237)

[ChatGTP Interaction 42 - Output Code Optimization/Fixup 151](#_Toc160090238)

[Input 151](#_Toc160090239)

[Output 153](#_Toc160090240)

[Notes 155](#_Toc160090241)

[ChatGTP Interaction 43 155](#_Toc160090242)

[Input 155](#_Toc160090243)

[Output 155](#_Toc160090244)

[Notes 155](#_Toc160090245)

[ChatGTP Interaction 44 - Direct Code Migration UI Tier 155](#_Toc160090246)

[Input 155](#_Toc160090247)

[Output 156](#_Toc160090248)

[Notes 159](#_Toc160090249)

[ChatGTP Interaction 45 - Output Code Optimization/Fixup 159](#_Toc160090250)

[Input 159](#_Toc160090251)

[Output 159](#_Toc160090252)

[Notes 160](#_Toc160090253)

[ChatGTP Interaction 46 - Output Code Optimization/Fixup 161](#_Toc160090254)

[Input 161](#_Toc160090255)

[Output 163](#_Toc160090256)

[Notes 165](#_Toc160090257)

[ChatGTP Interaction 47 - Output Code Optimization/Fixup 166](#_Toc160090258)

[Input 166](#_Toc160090259)

[Output 168](#_Toc160090260)

[Notes 171](#_Toc160090261)

[ChatGTP Interaction 48 - Output Code Optimization/Fixup 172](#_Toc160090262)

[Input 172](#_Toc160090263)

[Output 172](#_Toc160090264)

[Notes 174](#_Toc160090265)

[ChatGTP Interaction 49 - Output Code Optimization/Fixup 175](#_Toc160090266)

[Input 175](#_Toc160090267)

[Output 177](#_Toc160090268)

[Notes 180](#_Toc160090269)

[ChatGTP Interaction 50 - Output Code Optimization/Fixup 180](#_Toc160090270)

[Input 180](#_Toc160090271)

[Output 180](#_Toc160090272)

[Notes 182](#_Toc160090273)

[ChatGTP Interaction 51 - Output Code Optimization/Fixup 183](#_Toc160090274)

[Input 183](#_Toc160090275)

[Output 184](#_Toc160090276)

[Notes 185](#_Toc160090277)

[ChatGTP Interaction 52 – Direct Code Migration UI Tier 185](#_Toc160090278)

[Input 185](#_Toc160090279)

[Output 188](#_Toc160090280)

[Notes 191](#_Toc160090281)

[ChatGTP Interaction 53 - Output Code Optimization/Fixup 191](#_Toc160090282)

[Input 191](#_Toc160090283)

[Output 191](#_Toc160090284)

[Notes 193](#_Toc160090285)

[ChatGTP Interaction 54 - Output Code Optimization/Fixup 194](#_Toc160090286)

[Input 194](#_Toc160090287)

[Output 194](#_Toc160090288)

[Notes 195](#_Toc160090289)

[ChatGTP Interaction 55 195](#_Toc160090290)

[Input 195](#_Toc160090291)

[Output 196](#_Toc160090292)

[Notes 196](#_Toc160090293)

[ChatGTP Interaction 56 - Output Code Optimization/Fixup 196](#_Toc160090294)

[Input 196](#_Toc160090295)

[Output 198](#_Toc160090296)

[Notes 199](#_Toc160090297)

[ChatGTP Interaction 57 200](#_Toc160090298)

[Input 200](#_Toc160090299)

[Output 200](#_Toc160090300)

[Notes 202](#_Toc160090301)

[ChatGTP Interaction 58 202](#_Toc160090302)

[Input 202](#_Toc160090303)

[Output 203](#_Toc160090304)

[Notes 203](#_Toc160090305)

[ChatGTP Interaction 59 – Direct Code Migration UI Tier 203](#_Toc160090306)

[Input 203](#_Toc160090307)

[Output 205](#_Toc160090308)

[Notes 208](#_Toc160090309)

[ChatGTP Interaction 60 - Output Code Optimization/Fixup 209](#_Toc160090310)

[Input 209](#_Toc160090311)

[Output 209](#_Toc160090312)

[Notes 210](#_Toc160090313)

[ChatGTP Interaction 61 210](#_Toc160090314)

[Input 210](#_Toc160090315)

[Output 210](#_Toc160090316)

[Notes 210](#_Toc160090317)

[ChatGTP Interaction 62 211](#_Toc160090318)

[Input 211](#_Toc160090319)

[Output 211](#_Toc160090320)

[Notes 212](#_Toc160090321)

[ChatGTP Interaction 63 212](#_Toc160090322)

[Input 212](#_Toc160090323)

[Output 212](#_Toc160090324)

[Notes 212](#_Toc160090325)

[ChatGTP Interaction 64 212](#_Toc160090326)

[Input 212](#_Toc160090327)

[Output 213](#_Toc160090328)

[Notes 213](#_Toc160090329)

[ChatGTP Interaction 65 - Output Code Optimization/Fixup 213](#_Toc160090330)

[Input 213](#_Toc160090331)

[Output 213](#_Toc160090332)

[Notes 214](#_Toc160090333)

[ChatGTP Interaction 66 214](#_Toc160090334)

[Input 214](#_Toc160090335)

[Output 214](#_Toc160090336)

[Notes 214](#_Toc160090337)

[ChatGTP Interaction 67 – Direct Code Migration UI Tier 215](#_Toc160090338)

[Input 215](#_Toc160090339)

[Output 218](#_Toc160090340)

[Notes 218](#_Toc160090341)

[ChatGTP Interaction 68 – Direct Code Migration UI Tier 219](#_Toc160090342)

[Input 219](#_Toc160090343)

[Output 223](#_Toc160090344)

[Notes 226](#_Toc160090345)

[ChatGTP Interaction 69 – Direct Code Migration UI Tier 227](#_Toc160090346)

[Input 227](#_Toc160090347)

[Output 229](#_Toc160090348)

[Notes 233](#_Toc160090349)

[ChatGTP Interaction 70 – Direct Code Migration UI Tier 233](#_Toc160090350)

[Input 233](#_Toc160090351)

[Output 237](#_Toc160090352)

[Notes 240](#_Toc160090353)

[Phase 6 – Test/Debug 241](#_Toc160090354)

[Final Testing Stage Notes 241](#_Toc160090355)

[Phase 7 Stylizing the UI Pages to Mimic Look and Feel of VB.NET Win Forms 244](#_Toc160090356)

[ChatGTP Interaction 71 – Formatting Generation HTML/CSS UI Tier 244](#_Toc160090357)

[Input 244](#_Toc160090358)

[Output 244](#_Toc160090359)

[Notes 245](#_Toc160090360)

[ChatGTP Interaction 72 - Formatting Generation HTML/CSS UI Tier 245](#_Toc160090361)

[Input 245](#_Toc160090362)

[Output 246](#_Toc160090363)

[Notes 248](#_Toc160090364)

[ChatGTP Interaction 73 - Formatting Generation HTML/CSS UI Tier 248](#_Toc160090365)

[Input 248](#_Toc160090366)

[Output 249](#_Toc160090367)

[Notes 251](#_Toc160090368)

[ChatGTP Interaction 74 - Formatting Generation HTML/CSS UI Tier 251](#_Toc160090369)

[Input 251](#_Toc160090370)

[Output 253](#_Toc160090371)

[Notes 255](#_Toc160090372)

[Appendix: Why You Cannot “Just Start” With ChatGTP to Understand Target Technologies 256](#_Toc160090373)

[PHASE 1: Simple VB.NET Win Form Code, with Databinding to SQL Server:  Port to Blazor and Learn Blazor Basics, Through ChatGTP with No Prior Knowledge of Blazor, no “External” Tutorials. 256](#_Toc160090374)

[ChatGPT:  Interaction 1 256](#_Toc160090375)

[Input: 256](#_Toc160090376)

[Output 268](#_Toc160090377)

[Notes 272](#_Toc160090378)

[ChatGTP Interaction 2 273](#_Toc160090379)

[Input 273](#_Toc160090380)

[Output 273](#_Toc160090381)

[Notes 274](#_Toc160090382)

[ChatGTP Interaction 3 275](#_Toc160090383)

[INPUT 275](#_Toc160090384)

[OUTPUT 275](#_Toc160090385)

[Notes 278](#_Toc160090386)

[ChatGTP Interaction 4 280](#_Toc160090387)

[INPUT 280](#_Toc160090388)

[OUTPUT 280](#_Toc160090389)

[ChatGTP INTERACTION 5 281](#_Toc160090390)

[INPUT 282](#_Toc160090391)

[OUTPUT 282](#_Toc160090392)

[Notes 283](#_Toc160090393)

[ChatGTP Interaction 6 283](#_Toc160090394)

[Input 283](#_Toc160090395)

[Output 283](#_Toc160090396)

[ChatGTP INTERACTION 7 285](#_Toc160090397)

[Input 285](#_Toc160090398)

[Output 285](#_Toc160090399)

[Notes 288](#_Toc160090400)

[ChatGTP INTERACTION 8 288](#_Toc160090401)

[INPUT: 288](#_Toc160090402)

[OUTPUT 288](#_Toc160090403)

[ChatGTP INTERACTION 9 290](#_Toc160090404)

[INPUT 290](#_Toc160090405)

[OUTPUT 290](#_Toc160090406)

[Notes 292](#_Toc160090407)

[ChatGTP INTERACTION 10 292](#_Toc160090408)

[INPUT 292](#_Toc160090409)

[OUTPUT 292](#_Toc160090410)

[NOTES 293](#_Toc160090411)

[ChatGTP INTERACTION 11 294](#_Toc160090412)

[INPUT 294](#_Toc160090413)

[OUTPUT 295](#_Toc160090414)

[NOTES 296](#_Toc160090415)

[CHATGPT INTERACTION 12: 296](#_Toc160090416)

[INPUT 296](#_Toc160090417)

[OUTPUT 297](#_Toc160090418)

[CHATGPT INTERACTION 13: 299](#_Toc160090419)

[INPUT 299](#_Toc160090420)

[OUTPUT 299](#_Toc160090421)

[ChatGTP Interaction 14: 300](#_Toc160090422)

[INPUT 300](#_Toc160090423)

[OUTPUT 301](#_Toc160090424)

[ChatGTP Interaction 15 302](#_Toc160090425)

[INPUT 302](#_Toc160090426)

[OUTPUT 303](#_Toc160090427)

[Notes 305](#_Toc160090428)

[ChatGTP INTERACTION 16: 306](#_Toc160090429)

[INPUT 306](#_Toc160090430)

[OUTPUT 306](#_Toc160090431)

[Notes 307](#_Toc160090432)

[ChatGTP INTERACTION 17: 308](#_Toc160090433)

[INPUT 308](#_Toc160090434)

[OUTPUT 308](#_Toc160090435)

[PHASE 2:  Need External Learning as Phase 1 Not Successful In Teaching me the “Basics” of Blazor. 310](#_Toc160090436)

[Notes: 310](#_Toc160090437)

# Understanding Form Mapping from VB.NET WinForms App: To Target Blazor .NET 8/C# Version, Modern Web App

## I have not done any serious coding for 14 years, on .NET/C# or VB.NET. I am a good “proxy” for a large customer base with similar skills.

### My notes are detailed: As I am “new” on these latest MSFT technologies.

Notes: While the “VB.NET WinForms” are simple: The code behind is not as simple. Includes salted hash logins, login/auth validation per form, validation of inputs on buy/sell forms, business logic to complete transactions, MS Older (6.4.4) Entity Framework and also Pure ADO.NET Logic for demonstration of migrations to EntityFrameworkCore 8.0.2, all while translating from VB to C#, as well from WinForms to a modern .NET 8 Blazor Web App, with suggested ChatGTP advice on steps to do, and re-factoring for the Web to make “modern.” Also, for the ChatGTP migrations, used ChatGTP to help translate from VB Property Sheets, to Web App/Style Sheets, to make UI true to VB.NET original VB WinForm app. Every step of this migration, I document each and every interaction I had with OpenAI, via $20/month subscription, premium, to be on ChatGTP 4.0 and their “latest.” All source databases, data loading, source VB.NET 6 and “Target/Output” Blazor .NET 8 provided via GitHub as key part of this project, along with these detailed notes on ChatGTP migration steps.

## VB.NET WinForms

### [MainNav]

A close-up of a stock trader menu

Description automatically generated

### [Login]

A screenshot of a computer screen

Description automatically generated

### [Quotes]

A screenshot of a computer screen

Description automatically generated

### [GetQuotes/Button🡪VB DataGridView]

A screenshot of a computer screen

Description automatically generated

### Buy a Stock [Click DataGridView Row] (if logged in, validation not shown)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Holdings, DataGridView (Logged In, Validation not shown)

A screenshot of a computer

Description automatically generated

### Sell a Stock

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

### AccountForm

A screenshot of a computer

Description automatically generated

# Phase 1: Get Basic Instructions and Steps for Migration

## ChatGTP Interaction 1

### Input

Can you tell me the basic steps to migrate a VB.NET WinForm Application to a Blazor Application?

### Output

Migrating a VB.NET WinForms application to a Blazor application involves several significant changes, primarily because you are moving from a desktop framework to a web framework. This migration involves not just a change in programming language syntax (from VB.NET to C# in most cases, as Blazor uses C#), but also a fundamental shift in how the application's architecture is designed, how the UI is rendered, and how data is managed and communicated.

Here are the basic steps for migrating a VB.NET WinForms application to a Blazor application:

**1. Analyze and Plan:**

* **Assess the WinForms Application:** Understand the structure, features, and complexities of your current application. Identify core functionalities, third-party dependencies, and custom controls used in the app.
* **Define the Scope:** Decide which features need to be migrated and which can be redesigned or omitted. This might be a good opportunity to refactor or improve parts of the application.
* **Plan the Architecture:** Blazor applications can be developed as Blazor Server or Blazor WebAssembly. Choose the one that best fits your needs. Blazor Server executes on the server and uses SignalR to communicate with the client, while Blazor WebAssembly runs directly in the browser using WebAssembly.

**2. Set Up the Blazor Project:**

* **Create a New Blazor Project:** Using Visual Studio, create a new Blazor App. Choose between Blazor Server App and Blazor WebAssembly App based on your earlier decision.
* **Organize the Solution:** If you plan to share code between your old VB.NET application and the new Blazor app (like business logic or data access layers), consider organizing these into separate class libraries that can be referenced by both projects.

**3. Migrate Business Logic:**

* **Convert VB.NET to C#:** Most Blazor development is done in C#, so you might need to convert your VB.NET code. Tools like Telerik’s Code Converter can help, but manual adjustments are often necessary, especially for complex code or to adhere to C# best practices.
* **Refactor Code:** Adapt the business logic to fit the web application model. This may involve changing how data access is handled, modifying application workflows, and ensuring that the code is asynchronous where appropriate.

**4. Develop the UI with Blazor Components:**

* **Understand Blazor Components:** Blazor apps are built using components. A component is a block of reusable web UI that can include HTML, CSS, and C# code.
* **Map WinForms Controls to Blazor Components:** Identify equivalents for WinForms controls in Blazor. Some controls might have direct equivalents, while others might need custom components.
* **Layouts and Routing:** Design the layout and navigation of your Blazor app. Blazor uses a routing system to navigate between pages/components.

**5. Handle Data Access and State Management:**

* **Data Access:** Adapt your data access layer to work with Blazor. This might involve using Entity Framework Core for database operations.
* **State Management:** Web applications often have different state management needs compared to desktop applications. Explore Blazor's state management features and decide how to manage application state, considering both client-side and server-side options.

**6. Implement Authentication and Authorization (if applicable):**

* **Review Security Needs:** Web applications often face different security threats compared to desktop applications. Ensure your Blazor app is secure.
* **Implement Authentication/Authorization:** Blazor supports various authentication mechanisms, including cookie-based, token-based, and third-party auth providers. Implement the one that fits your application's needs.

**7. Testing and Deployment:**

* **Test Thoroughly:** Web applications can be accessed from various devices and browsers, so comprehensive testing is crucial. Test for functionality, performance, security, and compatibility.
* **Deployment:** Deploy your Blazor application to a suitable hosting environment. Blazor Server apps can be hosted in environments similar to ASP.NET Core apps, while Blazor WebAssembly apps can be hosted on static file servers.

**8. Training and Documentation:**

* **User Training:** Users familiar with the desktop application might need training to adapt to the web interface.
* **Update Documentation:** Ensure that all documentation reflects the new application and its deployment and maintenance procedures.

**Considerations:**

* **Migration Complexity:** Depending on the complexity and size of your WinForms application, a full migration might be a significant undertaking. It's often not a straightforward "conversion" process but rather a reimagining of the application for the web.
* **Incremental Migration:** For large applications, consider an incremental approach, where parts of the application are migrated or rewritten over time.
* **Learning Curve:** If your team is more familiar with VB.NET and WinForms, there might be a learning curve involved in switching to Blazor and C#. Factor this into your planning.

This migration process is an opportunity to modernize your application, not only in terms of technology but also by possibly rethinking user workflows, improving performance, and enhancing security.

### Notes

Will ChatGTP help me decide between a client side or server Blazor application?

## ChatGTP Interaction 2

### Input

Can you help me decide whether my VB.NET Win Forms application to be migrated to Blazor should be a Blazor WebAssembly or a Blazor Server App?

### Output

Deciding between Blazor WebAssembly and Blazor Server for migrating your VB.NET WinForms application depends on various factors related to your application's requirements, the expected user experience, and hosting considerations. Here's a comparison of key aspects to help you make an informed decision:

**Blazor WebAssembly**

**Pros:**

* **Client-Side Processing:** The app runs directly in the user's browser, utilizing the client's resources. This can reduce server load and potentially lead to a more responsive application, especially for read-heavy applications with minimal server interactions.
* **Offline Support:** Blazor WebAssembly apps can be made to work offline, which is beneficial if your users need to access the application without an internet connection.
* **Client-Side Data:** Ideal for applications that primarily work with client-side data or need to function with limited server connectivity.

**Cons:**

* **Initial Load Time:** The first load time can be longer since the entire app, including the .NET runtime, needs to be downloaded to the client's browser.
* **Browser Capabilities:** The app's performance and capabilities are limited by the browser's capabilities and the client's hardware.
* **SEO Challenges:** Client-side rendering can present challenges for search engine optimization (SEO), though this might not be a concern for internal or enterprise applications.

**Blazor Server**

**Pros:**

* **Thin Clients:** The app's execution happens on the server, which means clients don't need powerful hardware. This can be beneficial if your users have older computers or limited hardware resources.
* **Reduced Download Size:** Only the UI diff is sent over the network, not the entire app, leading to potentially faster initial load times and more efficient real-time UI updates.
* **Server-Side Capabilities:** Direct access to server-side resources and capabilities, which can simplify data access and integration with server-side APIs or databases.
* **SEO Friendly:** Since the rendering happens on the server, it's easier to make the application SEO-friendly compared to client-side rendered applications.

**Cons:**

* **Scalability:** Each connected client requires an active server connection and state management, which can increase server memory usage and potentially impact scalability.
* **Latency Sensitivity:** The user experience can be affected by network latency, as every user interaction requires a round trip to the server.
* **No Offline Support:** The application requires a constant connection to the server, so it won't work offline.

**Decision Factors**

* **Application Complexity and Size:** If your application is large and complex, Blazor Server might be more suitable due to its lighter client-side requirements. For smaller, more interactive apps, WebAssembly could provide a better experience.
* **User Base and Hardware:** Consider the hardware capabilities of your target users. For users with limited hardware, Blazor Server might be more appropriate.
* **Interactivity:** If your application requires high interactivity with minimal server processing, Blazor WebAssembly might offer a smoother experience.
* **Connectivity:** For applications that must work offline or in environments with unreliable connectivity, Blazor WebAssembly is the clear choice.
* **Hosting and Scalability:** Blazor Server apps can be more demanding on server resources, especially with many concurrent users. Evaluate your hosting environment and scalability needs.
* **Development Skills:** While both models use Blazor and C#, Blazor Server applications are more akin to traditional server-side web applications (like ASP.NET MVC or Razor Pages). If your team is more familiar with these patterns, Blazor Server might have a gentler learning curve.

**Hybrid Approach**

In some cases, a hybrid approach might be suitable, where parts of your application use Blazor Server to leverage server-side capabilities and other parts use Blazor WebAssembly for client-side interactivity and offline support. This approach, however, increases complexity and should be carefully considered.

Ultimately, the choice depends on balancing these factors against your application's specific needs and your users' expectations. It's often beneficial to prototype small parts of your application in both models to better understand the implications of each choice.

### Notes

I am going to use a Blazor Server app as the target, based on my desire to be fully thin-client, and have rich access to server-side logic, and reduce the download size for performance reasons.

## ChatGTP Extra Interaction – What if Choose Node.JS/Angular as the Target, Not Blazor/C#?

### Input

Can you help me migrate a VB.NET WInForms application to Node.Js and Angular?

### Output

Migrating a VB.NET WinForms application to a Node.js and Angular stack involves several steps, as you're moving from a desktop-based application framework to a web-based architecture. This process will not only involve translating the VB.NET code to JavaScript (or TypeScript for Angular), but also re-architecting the application to fit the web model. Here's a high-level overview of the steps involved:

**1. Analyze and Plan**

* **Understand the Current Application**: Document the current VB.NET application's functionality, architecture, and dependencies.
* **Define the Target Architecture**: Decide on the architecture for your Node.js backend and Angular frontend. Consider RESTful or GraphQL APIs for communication between the frontend and backend.

**2. Set Up the Development Environment**

* **Node.js**: Install Node.js and npm (Node Package Manager) to manage backend dependencies.
* **Angular CLI**: Install Angular CLI to scaffold and manage the Angular project.
* **TypeScript**: Since Angular applications are typically written in TypeScript, ensure familiarity with it.

**3. Migrate the Backend to Node.js**

* **Recreate Business Logic**: Convert VB.NET business logic into JavaScript/TypeScript. This will be the most time-consuming part, as it may not be a direct translation.
* **Data Access Layer**: If your application interacts with a database, implement data access in Node.js using libraries like **sequelize** for SQL databases or **mongoose** for MongoDB.
* **APIs**: Design and implement RESTful or GraphQL APIs to expose your backend services to the Angular frontend.

**4. Migrate the Frontend to Angular**

* **UI Components**: Translate WinForms UI components into Angular components. This will involve rethinking the UI in terms of web standards and responsive design.
* **Services and State Management**: Implement services in Angular for API communication and manage application state, possibly using libraries like NgRx or Akita if needed.
* **Routing**: Implement web-based routing using Angular's Router to manage navigation within your application.

**5. Testing and Refinement**

* **Unit and Integration Testing**: Write tests for both backend and frontend. For Node.js, you might use Mocha or Jest. For Angular, Karma and Jasmine are commonly used.
* **Performance and Security**: Test the application's performance and address security concerns inherent in web applications, like XSS, CSRF, and SQL injection.

**6. Deployment**

* **Deployment Strategy**: Choose a deployment strategy and platform for both the Node.js backend (e.g., AWS, Heroku) and the Angular frontend (e.g., Netlify, Vercel).
* **CI/CD Pipelines**: Set up continuous integration and continuous deployment pipelines to automate testing and deployment processes.

**Tips**

* **Incremental Migration**: Consider migrating the application module by module or screen by screen, rather than all at once.
* **Leverage Libraries and Frameworks**: Utilize existing Node.js and Angular libraries to speed up the migration process.
* **Compatibility and Polyfills**: Ensure your new application is compatible across different browsers and environments. Angular and TypeScript have good support for this.

This migration is a significant undertaking and requires careful planning and execution. Depending on the complexity and size of your VB.NET WinForms application, this could be a lengthy project. It's essential to involve stakeholders and potentially end-users during the planning and testing phases to ensure the new application meets all requirements and expectations.

### Notes

This is one big strength of ChatGTP—I can migrate to/from different targets. The steps are similar, and while I am not pursuing targeting Node.JS/Angular, I believe it could achieve similar levels of success as this project, with a completely different Web App Target platform.

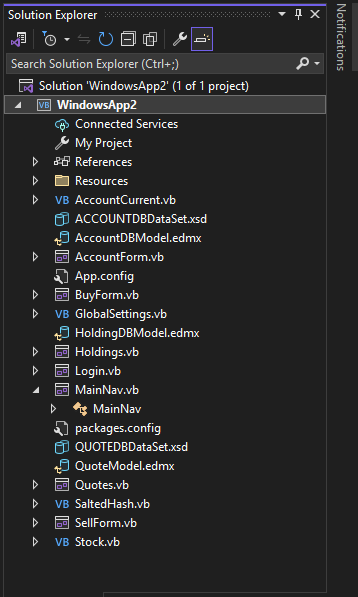
# Phase 2 – Starting the Migration with Business Logic

I am now ready to start with the basic steps outlined above in interaction 1.

I am choosing not to add sample pages in the new VS 2022 Solution. A clean starting point, empty project.

I am also adding folders for Business Classes and Data Classes, as this is recommended by ChatGTP in the Interaction 1. I will now start on migrating the business logic, as ChatGTP recommends **this as Step 3 in Interaction 1.**

First, the structure of my VB.NET WinForms App to migrate is represented as follows:



Forms: 7

Business Logic Classes: 4 (AccountCurrrent.vb, Stock.vb, GlobalSettings.vb and SaltedHash.vb)

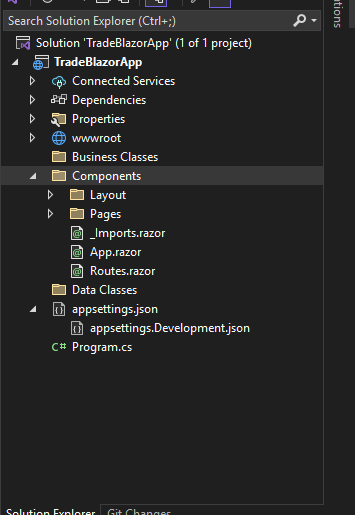
Config files: 1 (App.Config)

Model Classes: 3 (AccountDBModel, HoldingDBModel, QuoteModel)

.NET EntityFramework   
DataSets (v 6.4.4): 2 (ACCOUNTDBDataSet, QUOTEDBDataSet)

Databases used in the   
VB.NET application: 2 (ACCCOUNTDB and QUOTEDB)

Let’s start with SaltedHash.vb. I will place in the Business Logic folder in my Blazor App, vs. the root folder to organize better than the source VB Application.

My Blank Blazor Solution, created per ChatGTP instructions:  
  


## ChatGTP Interaction 3 - Direct Code Migration Business Logic Tier

### Input

Can you migrate this .NET class in VB.NET, to a .NET 8.0 Class in C# for use in my new Blazor application?

Imports System

Imports System.Security.Cryptography

Imports System.Text

Public NotInheritable Class SaltedHash

Private ReadOnly \_salt As String

Private ReadOnly \_hash As String

Private Const saltLength As Integer = 12

Public ReadOnly Property Salt As String

Get

Return \_salt

End Get

End Property

Public ReadOnly Property Hash As String

Get

Return \_hash

End Get

End Property

Private Sub New(s As String, h As String)

\_salt = s

\_hash = h

End Sub

Public Sub New()

End Sub

Public Shared Function Create(password As String) As SaltedHash

Dim salt As String = \_createSalt()

Dim hash As String = \_calculateHash(salt, password)

Return New SaltedHash(salt, hash)

End Function

Public Shared Function Create(salt As String, hash As String) As SaltedHash

Return New SaltedHash(salt, hash)

End Function

Public Function Verify(password As String) As Boolean

Dim h As String = \_calculateHash(\_salt, password)

Return \_hash.Equals(h)

End Function

Private Shared Function \_createSalt() As String

Dim r As Byte() = \_createRandomBytes(saltLength)

Return Convert.ToBase64String(r)

End Function

Private Shared Function \_createRandomBytes(len As Integer) As Byte()

Dim r As Byte() = New Byte(len - 1) {}

Using rng As New RNGCryptoServiceProvider()

rng.GetBytes(r)

End Using

Return r

End Function

Private Shared Function \_calculateHash(salt As String, password As String) As String

Dim data As Byte() = \_toByteArray(salt & password)

Dim hash As Byte() = \_calculateHash(data)

Return Convert.ToBase64String(hash)

End Function

Private Shared Function \_calculateHash(data As Byte()) As Byte()

Using sha1 As New SHA1CryptoServiceProvider()

Return sha1.ComputeHash(data)

End Using

End Function

Private Shared Function \_toByteArray(s As String) As Byte()

Return Encoding.UTF8.GetBytes(s)

End Function

End Class

### Output

To migrate your VB.NET class to a C# class for use in a .NET 8.0 Blazor application, you need to translate the VB.NET syntax to C# and adjust any language-specific constructs. Here's the converted C# class:

using System;

using System.Security.Cryptography;

using System.Text;

public sealed class SaltedHash

{

private readonly string \_salt;

private readonly string \_hash;

private const int SaltLength = 12;

public string Salt => \_salt;

public string Hash => \_hash;

private SaltedHash(string s, string h)

{

\_salt = s;

\_hash = h;

}

public SaltedHash()

{

}

public static SaltedHash Create(string password)

{

var salt = CreateSalt();

var hash = CalculateHash(salt, password);

return new SaltedHash(salt, hash);

}

public static SaltedHash Create(string salt, string hash)

{

return new SaltedHash(salt, hash);

}

public bool Verify(string password)

{

var h = CalculateHash(\_salt, password);

return \_hash.Equals(h);

}

private static string CreateSalt()

{

var r = CreateRandomBytes(SaltLength);

return Convert.ToBase64String(r);

}

private static byte[] CreateRandomBytes(int len)

{

var r = new byte[len];

using (var rng = new RNGCryptoServiceProvider())

{

rng.GetBytes(r);

}

return r;

}

private static string CalculateHash(string salt, string password)

{

var data = ToByteArray(salt + password);

var hash = CalculateHash(data);

return Convert.ToBase64String(hash);

}

private static byte[] CalculateHash(byte[] data)

{

using (var sha1 = new SHA1CryptoServiceProvider())

{

return sha1.ComputeHash(data);

}

}

private static byte[] ToByteArray(string s)

{

return Encoding.UTF8.GetBytes(s);

}

}

**Key Changes from VB.NET to C#:**

* **Property Syntax:** C# uses a different syntax for properties. The **Get** property in VB.NET is replaced with a lambda expression for read-only properties in C#.
* **Constructors:** C# uses a constructor with the same name as the class. In VB.NET, the **New** keyword is used for constructors.
* **Static Methods:** In C#, **Shared** methods in VB.NET are equivalent to **static** methods.
* **String Concatenation:** Both languages use the **+** operator for string concatenation, so this remains unchanged.
* **Array Initialization:** C# uses **new byte[len]** instead of VB.NET's **New Byte(len - 1) {}**.
* **Using Statements:** Both languages use **Using** statements for disposing of resources, but the syntax is slightly different. In C#, the **using** statement wraps the resource in parentheses.

This class should work within a .NET 8.0 Blazor application. Note that while SHA1 is used in this example for hashing, it is no longer considered secure for cryptographic purposes. For new applications, especially those requiring high security, consider using a more secure hashing algorithm like SHA256 or SHA512.

### Notes

Lets try it—nice it suggests a better/newer and more secure hashing algorithm!

The Build succeeds! A great start.  
  
Now, let’s do the next class, GlobalSettings, it’s very simple.

## ChatGTP Interaction 4 - Direct Code Migration Business Logic Tier

### Input

Can you migrate this VB.NET class to C# .NET 8.0 so it works in my Blazor app?

Public Class GlobalSettings

Public Shared LoggedIn As Boolean

Public Shared theLoginAccount As AccountCurrent

End Class

### Output

To migrate your VB.NET **GlobalSettings** class to a C# class suitable for a .NET 8.0 Blazor application, you would write it as follows:

public class GlobalSettings

{

public static bool LoggedIn { get; set; }

public static AccountCurrent TheLoginAccount { get; set; }

}

**Key Points in the Migration:**

* **Static Members:** In C#, **Shared** members in VB.NET are equivalent to **static** members. This means they belong to the class itself rather than any instance of the class.
* **Properties:** C# uses properties with get and set accessors for encapsulating fields. While your original VB.NET class directly exposed the fields as **Public Shared**, it's a common practice in C# to use properties. This approach provides more control over the data and allows for additional logic to be added to the getter and setter in the future if needed.
* **Naming Conventions:** C# typically uses PascalCase for property names (**TheLoginAccount** instead of **theLoginAccount**). It's a common convention in C# to start property names with an uppercase letter.
* **Type Declaration:** The **Boolean** type in VB.NET corresponds to **bool** in C#. Similarly, any custom or framework types (like **AccountCurrent** in your example) will keep their names but follow C# syntax rules.
* **Semicolons:** C# statements end with a semicolon (**;**).

**Considerations for Blazor Applications:**

* **State Management:** Static/global state as shown in your **GlobalSettings** class can be problematic in Blazor applications, especially in Blazor Server applications where the server maintains state for multiple clients. It's important to consider the scope of such state and whether it's appropriate for your application's architecture. In many cases, dependency injection with scoped or transient services might be a more suitable approach for managing user-specific data like login state and account information.
* **Security:** Ensure that storing user-specific data in a static context does not lead to security issues or data leaks between users, particularly in Blazor Server applications.

If **GlobalSettings** is intended to maintain state that is specific to an individual user session, especially in Blazor Server applications, you might want to explore other state management strategies that are scoped to the user session or the component lifecycle.

### Notes

Ok, I created the new class in my Blazor project. It will not compile yet, because it references a type: AccountCurrent, which is not migrated yet.   
  
So let’s try to migrate this class. I will continue to place these classes in my Business Classes Folder.

# Phase 3 – The First Mixed Business and Data Logic Classes

Note that the source VB application is not well factored, with various “business logic” classes and VB Forms containing data logic as well. So beginning here, business classes to be migrated will require also creating a well-factored database layer. Also note, the VB.NET application uses Entity Framework V6.4.4, but the latest Entity Framework for .NET 8/Blazor is EntityFrameworkCore v8.0.2, which is in an entirely different Microsoft namespace/package, and has modified constructs/concepts.

## ChatGTP Interaction 5 – Direct Code Migration Mixed Business and Data Tier

### Input

Can you migrate this VB.NET class to C# .NET 8.0 so it works in my Blazor app?

Imports System.Data.Entity

Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder

Imports System.Data.Entity.Migrations

Imports System.Data.SqlClient

Imports System.Runtime.ConstrainedExecution

Imports System.Runtime.Remoting.Contexts

Public Class AccountCurrent

Public Property Password As String

Public Property UserID As String

Public Property Name As String

Public Property AccountID As Decimal

Public Property Salt As String

Public Function Login(userid As String, password As String) As AccountCurrent

Dim theAccount As New AccountCurrent

theAccount.UserID = userid

Using accountcontext As New ACCOUNTDBEntities()

'START WHERE OK

'Dim result = accountcontext.ACCOUNTPROFILEs.SqlQuery("Select \* FROM dbo.ACCOUNTPROFILE WHERE USERID = '" + userid + "'")

' Assuming "userid" is a variable containing the user's ID you're interested in

Dim parameter = New SqlParameter("@userId", userid)

' Execute the SQL query using a parameterized query for safety

Dim result = accountcontext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter)

' Iterate over the results

For Each profile In result

' Access the properties/columns of each ACCOUNTPROFILE entity

theAccount.UserID = profile.USERID

theAccount.Password = profile.PASSWORD

theAccount.AccountID = profile.ACCOUNTID

theAccount.Salt = profile.SALT

theAccount.Name = profile.FULLNAME

' Continue accessing other columns/properties as needed

Next

If theAccount.Password Is Nothing Then

GlobalSettings.LoggedIn = False

Else

Dim ver = New SaltedHash().Create(theAccount.Salt, theAccount.Password)

GlobalSettings.LoggedIn = ver.Verify(password)

End If

End Using

Return theAccount

End Function

End Class

### Output

To migrate your VB.NET **AccountCurrent** class to a C# class for .NET 8.0, suitable for a Blazor application, the following C# code can be used. This code assumes that you have an Entity Framework context named **ACCOUNTDBEntities** and an entity **ACCOUNTPROFILE** that matches the structure of your database table.

using System.Data.Entity;

using System.Data.SqlClient;

public class AccountCurrent

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

public AccountCurrent Login(string userid, string password)

{

var theAccount = new AccountCurrent();

using (var accountContext = new ACCOUNTDBEntities())

{

var parameter = new SqlParameter("@userId", userid);

var result = accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter);

foreach (var profile in result)

{

theAccount.UserID = profile.USERID;

theAccount.Password = profile.PASSWORD;

theAccount.AccountID = profile.ACCOUNTID;

theAccount.Salt = profile.SALT;

theAccount.Name = profile.FULLNAME;

// Add more properties if needed

}

if (theAccount.Password == null)

{

GlobalSettings.LoggedIn = false;

}

else

{

var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password);

GlobalSettings.LoggedIn = verifier.Verify(password);

}

}

return theAccount;

}

}

To migrate your VB.NET **AccountCurrent** class to a C# class for .NET 8.0, suitable for a Blazor application, the following C# code can be used. This code assumes that you have an Entity Framework context named **ACCOUNTDBEntities** and an entity **ACCOUNTPROFILE** that matches the structure of your database table.

using System.Data.Entity;

using System.Data.SqlClient;

public class AccountCurrent

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

public AccountCurrent Login(string userid, string password)

{

var theAccount = new AccountCurrent();

using (var accountContext = new ACCOUNTDBEntities())

{

var parameter = new SqlParameter("@userId", userid);

var result = accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter);

foreach (var profile in result)

{

theAccount.UserID = profile.USERID;

theAccount.Password = profile.PASSWORD;

theAccount.AccountID = profile.ACCOUNTID;

theAccount.Salt = profile.SALT;

theAccount.Name = profile.FULLNAME;

// Add more properties if needed

}

if (theAccount.Password == null)

{

GlobalSettings.LoggedIn = false;

}

else

{

var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password);

GlobalSettings.LoggedIn = verifier.Verify(password);

}

}

return theAccount;

}

}

**Key Points and Changes:**

* **Entity Framework Context**: The **ACCOUNTDBEntities** context is used similarly to the VB.NET version. Ensure that this context exists in your C# project and is properly configured.
* **Parameterized Queries**: The SQL query uses a parameterized query to avoid SQL injection, similar to the VB.NET version.
* **Iteration Over Results**: The **foreach** loop in C# replaces the **For Each** loop in VB.NET for iterating over the query results.
* **Property Access**: Properties in C# use PascalCase by convention. Make sure that the properties in the **ACCOUNTPROFILE** entity match these names or adjust them accordingly.
* **GlobalSettings**: The **GlobalSettings** class usage remains the same. Ensure that this class is correctly migrated and available in your C# project.
* **SaltedHash**: The usage of **SaltedHash** assumes that you have a corresponding C# version of this class that includes a static **Create** method returning an instance which can be used to verify passwords.

**Additional Considerations:**

* **Security**: Be cautious with storing and handling passwords. Ensure that the **SaltedHash** class uses secure hashing algorithms and practices.
* **Entity Framework Version**: This example assumes Entity Framework 6.x due to the usage of **DbContext.SqlQuery**. If you're using EF Core, the approach might differ slightly, such as using **FromSqlRaw** for raw SQL queries.
* **Error Handling**: Consider adding error handling around database operations to manage exceptions gracefully.
* **DbContext Lifetime**: In Blazor Server applications, be mindful of the **DbContext** lifecycle. It's generally recommended to use a scoped lifetime for **DbContext** instances within Blazor applications to avoid issues with concurrent database operations.

### Notes

OK, this will take awhile to digest, as I have now hit a class that uses the Entity Framework 6.4.4 vs new 8.0.2 (.NET Core) and I am going to have to dive into creating the equivalent data classes in my Blazor app.

I will start by creating the class AccountCurrent, and pasting the code from ChatGTP.

Now, I need some manual work to fix some things in the new C# class. First, need add reference to the Entity Framework in my project.

## ChatGTP Interaction 6a – Output Code Optimization/Fixup

### Input

My Razor application in .NET 8 can not find this import: "using System.Data.Entity;" How to fix?

### Output

If your Razor application in .NET 8 (which I assume refers to a version of .NET Core 3.1, .NET 5, .NET 6, or .NET 7, as there isn't a .NET 8 at the time of writing) cannot find the **using System.Data.Entity;** namespace, it's likely because you're trying to use Entity Framework 6 (EF6) in a .NET Core/.NET 5+/ASP.NET Core application, where the **Entity Framework Core** is the recommended ORM (Object-Relational Mapper) framework.

To fix this issue, you have a couple of options:

**Migrate to Entity Framework Core (Recommended)**

Entity Framework Core (EF Core) is the modern version of Entity Framework designed for .NET Core applications, including ASP.NET Core and Blazor applications. Here's how to start migrating:

1. **Add EF Core to Your Project**: Install the EF Core package via NuGet. You can do this using the NuGet Package Manager in Visual Studio or via the command line in your project directory:

dotnet add package Microsoft.EntityFrameworkCore

If you're using SQL Server, you might also need:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

1. **Update Your Code**: You'll need to update your data access code to use EF Core. This includes adjusting your DbContext and entity classes. The namespace changes from **System.Data.Entity** to **Microsoft.EntityFrameworkCore**.
2. **Migrate Your Data Access Code**: EF Core has different APIs and capabilities, so some code changes will be necessary. Refer to the EF Core documentation for guidance on performing these migrations.

**Use Entity Framework 6 with .NET Core/.NET 5+/ASP.NET Core**

If you must use EF6 in your application, you should ensure that you're using EF6.3 or later, as these versions include support for .NET Core. However, note that some features may not be fully compatible or perform as well as they would in a .NET Framework application.

1. **Install EF6 Package**: Make sure you have installed the Entity Framework 6 NuGet package that supports .NET Core:

dotnet add package EntityFramework --version 6.4.4

1. **Correct the Namespace**: The namespace **System.Data.Entity** should be available if you have correctly installed EF6. If it's still not recognized, try rebuilding your solution or ensuring that the EF6 package is properly installed and referenced in your project.
2. **Compatibility Considerations**: Keep in mind that while EF6 can work with .NET Core and ASP.NET Core applications, it might not integrate as seamlessly as EF Core, especially for newer ASP.NET Core features and functionalities.

**Final Note**

For most new development, especially with .NET Core, .NET 5+, and ASP.NET Core applications, it's recommended to use Entity Framework Core due to its active development, performance improvements, and better integration with the .NET ecosystem.

### Notes

First, its interesting it says there “Is not .NET 8.0 at the time of this writing.” That is wrong. I am using .NET 8 in my target Blazor app.

I will use VS 2022 NuGet package manager to add the two suggested packages.

I found them easily, and added to my Project via VS 2022. However, I notice there are 3 other Entity Framework packages I could add, but will wait for now.

Also, it did not tell me to add using statements in my AccountCurrent class, for the two packages. So I will do manually.

I manually copied all of my “Imports” from the AccountCurrent.VB class, into my AccountCurrent.cs class, translating to using statements:  
  
 using System;

using System.Data.Entity;

using System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder;

using System.Data.Entity.Migrations;

using System.Data.SqlClient;

using System.Runtime.ConstrainedExecution;

using System.Runtime.Remoting.Contexts;

But VS cannot resolve most of them! Why?

## ChatGTP Interaction 6b - Output Code Optimization/Fixup

### Input

I have the following statement in my .NET 8 application: using System.Data.Entity; , and I get this error: "The type or namespace name Entity does not exist in the namespace System.Data. Are you missing an assembly reference?" How to fix?

### Output

The error message you're encountering, "The type or namespace name 'Entity' does not exist in the namespace 'System.Data'...," typically occurs when your project is targeting .NET Core, .NET 5, .NET 6, or a later version, but you're trying to use Entity Framework 6 (or earlier) which relies on the **System.Data.Entity** namespace. In .NET Core and later versions, Entity Framework Core (EF Core) is used instead, which utilizes different namespaces.

To resolve this issue, you'll need to transition to using EF Core or ensure your project is correctly set up to use EF 6.3 or later, which is compatible with .NET Core projects. Here's how you can approach this:

**Option 1: Migrate to Entity Framework Core (Recommended for New Projects)**

1. **Remove the Old Entity Framework Reference**: If you have a reference to the traditional Entity Framework, remove it from your project.
2. **Install Entity Framework Core**: Install the EF Core package suitable for your database provider. For example, for SQL Server, use the following command in your project directory:

shellCopy code

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

You might also need the EF Core Tools package for migrations:

shellCopy code

dotnet add package Microsoft.EntityFrameworkCore.Tools

1. **Update Your Using Statements**: Replace **using System.Data.Entity;** with **using Microsoft.EntityFrameworkCore;**.
2. **Refactor Your Code**: Adjust your data models, DbContext, and data access code to use EF Core. This may involve significant changes, depending on your existing codebase.

**Option 2: Ensure Compatibility with EF 6 in .NET Core**

If you must use Entity Framework 6 due to legacy reasons or specific requirements, ensure you're using version 6.3 or later, as these versions have compatibility improvements for .NET Core:

1. **Install the Correct EF 6 Version**: Make sure you have Entity Framework 6.3 or later installed. You can install or update it via NuGet Package Manager or the command line:

shellCopy code

dotnet add package EntityFramework --version 6.4.4

1. **Verify Your Project File**: Ensure your project file (**.csproj**) doesn't have conflicting references and is targeting a compatible framework version.
2. **Check Your Code**: Even with EF 6.3 or later, some features or usage patterns from .NET Framework may not work seamlessly in .NET Core or .NET 5/6/7/8. You may need to adjust certain parts of your code.

**Additional Considerations**

* **Namespace Issues**: If you've confirmed that the correct packages are installed and the project still doesn't recognize the namespace, try cleaning and rebuilding the solution. If the issue persists, check the **.csproj** file for any inconsistencies or errors.
* **Migration to EF Core**: Given that EF Core is the recommended approach for new .NET Core and .NET 5+ applications, consider migrating to EF Core unless there's a compelling reason to stick with EF 6. EF Core has many improvements and features not available in EF 6.

Choose the option that best fits your project's requirements and long-term maintenance plans. For most new development, migrating to EF Core is the recommended path.

### Notes

OK. I see now, the newer Entity Framework (8.0.2) Microsoft has changed namespaces in the package. Such as System.Data.Entity is now:  
  
using Microsoft.EntityFrameworkCore. I changed it and it seems ok.

I will try to get ChatGTP to fix my other references to new namespaces.

## ChatGTP Interaction 7 - Output Code Optimization/Fixup

## Input

OK, I am using .NET Entity Framework 8.0.2, and the following using statements will not resolve in my Blazor app, I have already followed steps to add the latest EntityFrameworkCore to my project. Can you fix these:

using System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder;

using System.Data.Entity.Migrations;

### Output

If you're using Entity Framework Core 8.0.2 in your Blazor application and encountering issues with resolving namespaces similar to **System.Data.Entity**, it's because Entity Framework Core uses different namespaces compared to the older Entity Framework 6 and earlier versions.

The namespaces and APIs you mentioned (**System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder** and **System.Data.Entity.Migrations**) do not directly correspond to Entity Framework Core's structure, as EF Core has been significantly redesigned and does not use the same namespaces or API designs as EF 6.

**Migrating the Namespaces and APIs**

1. **Expression Building**:
   * In EF Core, expression building is typically done using LINQ and doesn't require direct use of expression builder namespaces as in EF 6. If you need to build expressions dynamically, you would typically use the **System.Linq.Expressions** namespace.
   * If you're doing something more complex that you believe requires direct manipulation of command trees, you might want to reconsider the approach or look for an EF Core-compatible way to achieve the same result.
2. **Migrations**:
   * EF Core handles migrations differently than EF 6. The namespace for migrations in EF Core is **Microsoft.EntityFrameworkCore.Migrations**.
   * To add, remove, or manage migrations, you would typically use the EF Core CLI tools or the Package Manager Console commands in Visual Studio, such as **Add-Migration**, **Update-Database**, etc.

**Adjusting Your Using Statements**

Replace the non-resolving **using** statements with ones that are relevant to EF Core. For most scenarios, you might not need direct replacements for these namespaces. Instead, focus on what you're trying to achieve (e.g., querying the database, managing migrations) and use the corresponding EF Core APIs.

For migrations, make sure you have the EF Core tools installed, and use the following namespace if you're writing custom migration code:

csharpCopy code

using Microsoft.EntityFrameworkCore.Migrations;

For expression building or any advanced query scenarios, you might only need:

csharpCopy code

using System.Linq.Expressions;

And for standard EF Core operations (querying, updating, etc.), ensure you have:

csharpCopy code

using Microsoft.EntityFrameworkCore;

**Additional Steps**

* Review the EF Core documentation to understand the new patterns and APIs. The documentation provides a comprehensive guide on performing various tasks with EF Core.
* If you have specific code segments or functionalities that you're having trouble migrating from EF 6 to EF Core, consider posting those specific instances for more targeted guidance.
* Remember that EF Core uses a different approach and philosophy in some areas compared to EF 6, so some direct one-to-one mappings of classes or methods might not exist. Often, you'll find that EF Core provides a more streamlined or efficient way to accomplish your objectives.

### Notes

OK: I changed my using statements per instructions, to the new namespaces Microsoft created for .NET 8.  
  
I have one lingering issue, it cannot find:  
  
System.Runtime.Remoting.Contexts  
  
Will use ChatGTP to fix? Or do I need it?

## ChatGTP Interaction 8 - Output Code Optimization/Fixup

### Input

My Blazor application in .NET 8 can not find this import: "using System.Data.Entity;" How to fix?

### Output

In .NET Core, .NET 5, .NET 6, and later versions (including what you're referring to as .NET 8.0, which could be .NET 8 in the future or .NET 7 as the latest at the time of writing), the **System.Runtime.Remoting** namespace, including **System.Runtime.Remoting.Contexts**, is not supported. Remoting was a feature in .NET Framework used for inter-application communication, but it's not compatible with the newer, cross-platform .NET implementations due to its tight coupling with the Windows platform and the AppDomain isolation model.

For Blazor and other modern .NET applications, you'll need to use alternative approaches for inter-process or inter-service communication, such as:

1. **HTTP/HTTPS Requests**: For web-based or networked communication, using HttpClient or a similar web client to make HTTP requests is standard practice.
2. **SignalR**: For real-time, bi-directional communication between client and server over websockets, SignalR is a good choice, especially within Blazor applications.
3. **gRPC**: For high-performance, contract-based RPC (Remote Procedure Call) communication, gRPC is a modern, open-source framework that works well in the .NET ecosystem.
4. **Messaging Queues and Event Brokers**: Technologies like RabbitMQ, Azure Service Bus, or Kafka can be used for asynchronous communication and event-driven architectures.

If you're migrating a .NET Framework application that used remoting to .NET Core or .NET 5/6/7/8, you'll need to replace the remoting parts of your application with one of these modern communication mechanisms.

For specific use cases within the same application (like accessing shared data or services), consider using dependency injection (DI), which is a first-class concept in .NET Core and Blazor applications. DI can be used to share instances of services or data contexts across components and services within your app.

If you have a specific scenario in mind where you were considering using remoting or need to replace existing remoting functionality, please provide more details so that a more targeted solution can be suggested.

### Notes

I will simply remove the reference for now.  
  
It seems the final step to get Account.CS to migrate is to add the ACCOUNTDBEntities as an EF Data model to my project.

I will again use ChatGTP to make sure I know the steps.

## ChatGTP Interaction 9 – Output Code Optimization/Fixup

### Input

I am using .NET and the EntityFrameworkCore. I have this code that cannot resolve "ACCOUNTDBEntities", I have not yet added and EntityFramework database models/entities to my project. How to fix? Here is the code:

using (var accountContext = new ACCOUNTDBEntities()) { var parameter = new SqlParameter("@userId", userid); var result = accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter); foreach (var profile in result) { theAccount.UserID = profile.USERID; theAccount.Password = profile.PASSWORD; theAccount.AccountID = profile.ACCOUNTID; theAccount.Salt = profile.SALT; theAccount.Name = profile.FULLNAME; // Add more properties if needed } if (theAccount.Password == null) { GlobalSettings.LoggedIn = false; } else { var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password); GlobalSettings.LoggedIn = verifier.Verify(password); } }

### Output

To resolve the **ACCOUNTDBEntities** issue in your .NET application using Entity Framework Core, you'll need to add EF Core data models and a DbContext to your project. The **DbContext** acts as a session with the database, allowing you to query and save data.

Here's a step-by-step guide to fix the issue:

**Step 1: Define Your Models**

Based on your code snippet, you'll likely need a model for your **ACCOUNTPROFILE** table. Create a C# class that maps to the structure of your **ACCOUNTPROFILE** table:

public class AccountProfile

{

public string UserID { get; set; }

public string Password { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

public string FullName { get; set; }

}

**Step 2: Create Your DbContext**

Create a class that derives from **DbContext** and represents a session with your database. This is where you'll define properties for each table you want to interact with:

using Microsoft.EntityFrameworkCore;

public class ACCOUNTDBEntities : DbContext

{

public ACCOUNTDBEntities(DbContextOptions<ACCOUNTDBEntities> options)

: base(options)

{

}

public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; }

}

**Step 3: Configure Your DbContext**

In your startup configuration (e.g., **Startup.cs** or **Program.cs** depending on your .NET version), configure the **ACCOUNTDBEntities** context to use a specific database provider like SQL Server, PostgreSQL, etc.:

For ASP.NET Core 6.0+ (in **Program.cs**):

var connectionString = builder.Configuration.GetConnectionString("YourConnectionStringName");

builder.Services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(connectionString)); // Adjust for your database provider

Make sure you have the appropriate EF Core provider package installed, e.g., **Microsoft.EntityFrameworkCore.SqlServer** for SQL Server.

**Step 4: Update Your Code for EF Core**

EF Core uses **LINQ** to query the database, so you'll need to adjust your querying code. The **SqlQuery** method you used is not available in EF Core like it was in EF 6. Instead, use LINQ queries:

using (var accountContext = new ACCOUNTDBEntities())

{

var profiles = accountContext.ACCOUNTPROFILEs

.Where(p => p.UserID == userid)

.ToList();

foreach (var profile in profiles)

{

theAccount.UserID = profile.UserID;

theAccount.Password = profile.Password;

theAccount.AccountID = profile.AccountID;

theAccount.Salt = profile.Salt;

theAccount.Name = profile.FullName;

}

// Rest of your code...

}

**Step 5: Add Connection String**

Ensure your connection string is defined in your **appsettings.json** or wherever you manage your application's configuration settings:

{

"ConnectionStrings": {

"YourConnectionStringName": "Server=your\_server;Database=your\_database;User Id=your\_user;Password=your\_password;"

}

}

**Final Notes**

* Replace **"YourConnectionStringName"** with a key that represents your connection string in the configuration file.
* Adjust property names and types in the **AccountProfile** model to match your database schema exactly.
* If you're new to EF Core, consider reviewing the official Entity Framework Core documentation for more detailed guidance on getting started, querying data, and managing migrations.

### Notes

I am going to ask ChatGTP to narrow future responses to .NET 8.0 and Blazor

## ChatGTP Interaction 10

### Input

For future responses, can you assume I am building a .NET 8.0 Blazor Server app?

### Output

Understood, I'll assume you're working on a .NET 8.0 Blazor Server application for future responses. If you have more questions or need further assistance, feel free to ask!

I will ask again, the same question as Interaction 9:

## ChatGTP Interaction 11

### Input

For future responses, can you assume I am building a .NET 8.0 Blazor Server app?

### Output

Understood, I'll assume you're working on a .NET 8.0 Blazor Server application for future responses. If you have more questions or need further assistance, feel free to ask!

I will ask again, the same question as Interaction 8:

## ChatGTP Interaction 12: - Output Code Optimization/Fixup

### Input

I am using .NET and the EntityFrameworkCore. I have this code that cannot resolve "ACCOUNTDBEntities", I have not yet added and EntityFramework database models/entities to my project. How to fix? Here is the code: using (var accountContext = new ACCOUNTDBEntities()) { var parameter = new SqlParameter("@userId", userid); var result = accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter); foreach (var profile in result) { theAccount.UserID = profile.USERID;

theAccount.Password = profile.PASSWORD; theAccount.AccountID = profile.ACCOUNTID; theAccount.Salt = profile.SALT; theAccount.Name = profile.FULLNAME; // Add more properties if needed } if (theAccount.Password == null) { GlobalSettings.LoggedIn = false; } else { var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password); GlobalSettings.LoggedIn = verifier.Verify(password); } }

### Output

To resolve the issue with **ACCOUNTDBEntities** not being recognized in your .NET 8.0 Blazor Server application using Entity Framework Core, you'll need to define your database context and entity models. Here's a step-by-step guide to setting up your Entity Framework Core models and DbContext:

**Step 1: Define Your Entity Models**

First, you need to define a class for your **ACCOUNTPROFILE** table. This class should mirror the structure of the table in your database:

public class AccountProfile

{

public string UserID { get; set; }

public string Password { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

public string FullName { get; set; }

// Add more properties as needed to match your table's structure

}

**Step 2: Create Your DbContext**

The **DbContext** serves as the primary class that coordinates Entity Framework functionality for a given data model. Create a class named **ACCOUNTDBEntities** that inherits from **DbContext**:

using Microsoft.EntityFrameworkCore;

public class ACCOUNTDBEntities : DbContext

{

public ACCOUNTDBEntities(DbContextOptions<ACCOUNTDBEntities> options)

: base(options)

{

}

public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; }

}

**Step 3: Register Your DbContext in Startup.cs**

In your **Startup.cs** or **Program.cs** (depending on your project setup), you need to register your **DbContext** with dependency injection:

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

// Add other services like AddRazorPages(), AddServerSideBlazor(), etc.

}

Make sure you have a connection string defined in your **appsettings.json**:

"ConnectionStrings": {

"YourConnectionStringName": "Server=your\_server\_name;Database=your\_database\_name;User Id=your\_username;Password=your\_password;"

}

**Step 4: Update Your Code to Use EF Core**

Entity Framework Core does not support the **SqlQuery** method for executing raw SQL queries directly on the **DbSet**. Instead, you can use the **FromSqlRaw** method on the **DbSet** to execute raw SQL queries that return entities. However, this method should be used on a **DbContext** instance rather than directly on a **DbSet<AccountProfile>**:

using (var accountContext = new ACCOUNTDBEntities())

{

var profiles = accountContext.ACCOUNTPROFILEs

.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = {0}", userid)

.ToList();

foreach (var profile in profiles)

{

// Your code to process each profile

}

}

**Step 5: Adjust for EF Core Practices**

* Entity Framework Core encourages the use of LINQ to query data instead of raw SQL where possible. Consider refactoring your raw SQL queries to LINQ if feasible.
* Ensure that **GlobalSettings** and **SaltedHash** are appropriately implemented in your project, as these seem to be custom classes/utilities.

By following these steps, you should be able to integrate Entity Framework Core into your Blazor Server application and resolve the issue with **ACCOUNTDBEntities**.

### Notes

Learning: Make sure to adjust ChatGTP queries with as much qualifying info as possible, for context in future queries. I do not know how long a ChatGTP session lasts, but in a nutshell, this was key to avoid getting instructions out of date:

**Input from ChatGTP Interaction 11**

For future responses, can you assume I am building a .NET 8.0 Blazor Server app?

**Output from ChatGTP Interaction 11**

Understood, I'll assume you're working on a .NET 8.0 Blazor Server application for future responses. If you have more questions or need further assistance, feel free to ask!

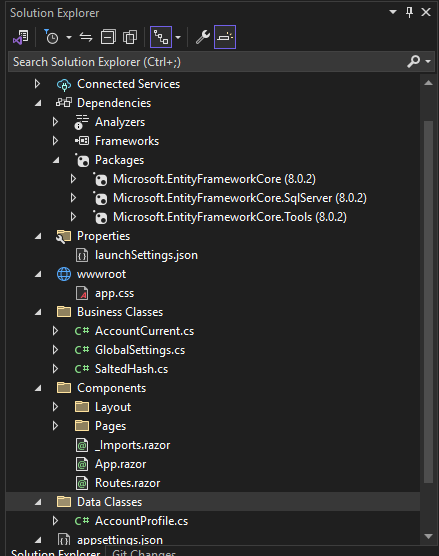
Key question I have: how long does this context last? I of course take time to follow the steps, breaks, etc. So, would be good to know:  
  
1) How many questions in a row will this ChatGTP context setting last for?

2) And is there and/or a time expire on this? (session timeouts?)  
  
Of course, I could qualify every question, to target responses to .NET 8 and Blazor. But wonder how this context setting works (ala Robert’s experimentation)? Especially not only interactive responses via OpenAI web site; but when you start using their API? Something to try to find out, if they publish this info.

But anyway, I am totally impressed so far—with quality of responses from OpenAI/ChatGTP 4.0, and the “additional considerations” they provide, re re-factoring, other issues to consider. IMPRESSIVE.

Now on to next step, Lets get my DBEntities defined. Following ChatGTP advice—start with Business Logic/Layer/Classes. Then move on. The right advice. (note: later figured out, no, need start with DB layer, not bus logic layer,)

Continue to follow the advice, and code migrated/project/VS Solution setup, etc.  
  
So far so good! With some caveats noted: initially giving me antiquated MS Namespaces, references, esp in C# using statements, for “older” versions of the MS Entity Framework. I want to target 8.0.xx, the latest, per NuGet package management and my target of .NET 8.0 for this Blazor app.

Also, my VS 2022 Blazor Solution Now Looks as Follows:  
  


Continued notes:

From ChatGTP Interaction 12, immediately above, I am on to:  
  
**Step 3: Register Your DbContext in Startup.cs**

In your **Startup.cs** or **Program.cs** (depending on your project setup), you need to register your **DbContext** with dependency injection:

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

// Add other services like AddRazorPages(), AddServerSideBlazor(), etc.

}

Make sure you have a connection string defined in your **appsettings.json**:

"ConnectionStrings": {

"YourConnectionStringName": "Server=your\_server\_name;Database=your\_database\_name;User Id=your\_username;Password=your\_password;"

}

**FAIL!**

First, I asked target .NET 8: Program.cs replaced Startup.cs, for awhile now. So still giving me non-targeted .NET version for Blazor info. Startup.cs is an “old” construct for Blazor apps, prior .NET 6.  
  
Second, It asks I paste a method definition into Program.cs.   
This does not work (yes, I know Program.cs is the “bootstrap” for Blazor .NET 8.x apps.

The above, is a ton of info for me to grock, try to learn why copy paste of method above (public void ConfigureServices(IServiceCollection services)

into Program.cs not working. Working on.

I expect, as just a bootstrap/startup process, the Program.cs cannot take method declarations.   
  
So ChatGTP fail here, on location to place. Yes, I know I need to “Register Services”. But confused now. Was going well.

Continued Notes:  
  
Obviously, a method here in Program. Cs (not a class), not right advice. But the generated code, within that method, seems right?  
  
So, just pasting that into Program.cs, as ChatGTP generated: pasting:

“services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

// Add other services like AddRazorPages(), AddServerSideBlazor(), etc.  
----------------------  
  
Into Program.cs, in place I think it goes, BEFORE Builder.Build()

This seems better:  
  
 **I deleted method declaration in favor of “using my brain” on where this line belongs, vs ChatGTP advice.**

My Program.cs now reads as:  
  
using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using TradeBlazorApp.Components;

using TradeBlazorApp.Data\_Classes;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

// Add services to the container.

builder.Services.AddRazorComponents()

.AddInteractiveServerComponents();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error", createScopeForErrors: true);

}

app.UseStaticFiles();

app.UseAntiforgery();

app.MapRazorComponents<App>()

.AddInteractiveServerRenderMode();

app.Run();

Now, on to figuring out Config, to get my Connection string right. Of course, there will be some very interesting steps to come (not only Config, from App.Config to JSON):

* 1. Building Data Model classes out of XSD Entity Framework.
  2. What help I get in UI? Considering, look/feel of App is all in VB Properties, with ChatGTP cannot read.
  3. Code optimizations.
  4. Using pure client-side Blazor, vs. Server Side
  5. The repeatable steps.
  6. What about building re-usable Razor client side components, ala what VB developers know as controls/toolbox elements?
  7. Database migration steps; to create Schemas, load data?
  8. Other, like code and comments optimizations?

## ChatGTP Interaction 13 – Direct Code Migration (Config)

### Input

I am migrating an application, from VB.NET, within VS 2022. I am making great progress, but, the latest issue is reading from my migrated Config settings, from older App.Config construct. Here is the code:   
  
"using Microsoft.EntityFrameworkCore; using Microsoft.Extensions.Configuration; using TradeBlazorApp.Components; using TradeBlazorApp.Data\_Classes; var builder = WebApplication.CreateBuilder(args); //ADDED this line of code, per ChatGTP instructions, re Interaction 12. Tho ChatGTP wanted me to define a method within Program.cs, that did not fly. //So just added this line, in line, to configure the service. Next onto Config! Need connection string to be resolved. //1. builder.Services.AddDbContext<ACCOUNTDBEntities>(options => options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName"))); // Add services to the container. builder.Services.AddRazorComponents() .AddInteractiveServerComponents(); var app = builder.Build(); // Configure the HTTP request pipeline. if (!app.Environment.IsDevelopment()) { app.UseExceptionHandler("/Error", createScopeForErrors: true); } app.UseStaticFiles(); app.UseAntiforgery(); app.MapRazorComponents<App>() .AddInteractiveServerRenderMode(); app.Run();" My issue is: "The Name Configuration does not exist in the current context": How to fix?

### Output

To resolve the issue where "The name 'Configuration' does not exist in the current context" in your Blazor Server Application within .NET 8, you need to access the configuration from the **builder** object that you have created using **WebApplication.CreateBuilder(args)**. The **builder** object has a **Configuration** property that you should use to access your configuration settings, including connection strings.

Here is how you can modify your code snippet to use the **builder.Configuration** properly:

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using TradeBlazorApp.Components;

using TradeBlazorApp.Data\_Classes;

var builder = WebApplication.CreateBuilder(args);

// Use builder.Configuration to access the Configuration properties

builder.Services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("YourConnectionStringName")));

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddServerSideBlazor();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

In this revised code:

* I've replaced **builder.Services.AddRazorComponents().AddInteractiveServerComponents()** with **builder.Services.AddRazorPages()** and **builder.Services.AddServerSideBlazor()**, which are the standard way to add Razor Pages and Server-Side Blazor services to your application in .NET Core 3 onwards, including .NET 8. If **AddRazorComponents()** and **AddInteractiveServerComponents()** are specific methods you've defined for your application, ensure they're correctly implemented.
* The **builder.Configuration.GetConnectionString("YourConnectionStringName")** is used to access the connection string from your configuration. Make sure to replace **"YourConnectionStringName"** with the actual name of your connection string key as defined in your **appsettings.json** or other configuration source.
* I've adjusted the pipeline configuration to include **app.UseRouting()**, **app.MapBlazorHub()**, and **app.MapFallbackToPage("/\_Host")**, which are typical for a Blazor Server application. Adjust these as necessary for your application's specific routing and page structure.

Ensure your connection strings are properly configured in your **appsettings.json** or other configuration files, and replace **"YourConnectionStringName"** with the actual key you have used in your configuration.

### Notes

Pasted entire new Program.cs into solution. Seems to be better! Can understand “Configuration” namespace now. It understands I am using new construct/file for app config:  
  
appsettings.json  
  
vs:  
  
VB.NET = app.config.

Seems to have corrected my using statements/imports, to the new .NET 8 way of doing things for a Blazor app.

## ChatGTP Interaction 14 - Direct Code Migration (Config)

### Input

Can you migrate this VB.NET app.config file to a Blazor appsettings.json file:  
  
<?xml version="1.0" encoding="utf-8"?>

<configuration>

<configSections>

<!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->

<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

</configSections>

<connectionStrings>

<add name="WindowsApp2.My.MySettings.QUOTEDBConnectionString" connectionString="Data Source=GREG-SURFACE4\SQLEXPRESS;Initial Catalog=QUOTEDB;Integrated Security=True;TrustServerCertificate=True" providerName="System.Data.SqlClient" />

<add name="WindowsApp2.My.MySettings.ACCOUNTDBConnectionString" connectionString="Data Source=GREG-SURFACE4\SQLEXPRESS;Initial Catalog=ACCOUNTDB;Integrated Security=True;TrustServerCertificate=True" providerName="System.Data.SqlClient" />

<add name="QUOTEDBEntities" connectionString="metadata=res://\*/QuoteModel.csdl|res://\*/QuoteModel.ssdl|res://\*/QuoteModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=GREG-SURFACE4\SQLEXPRESS;initial catalog=QUOTEDB;integrated security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

<add name="ACCOUNTDBEntities" connectionString="metadata=res://\*/AccountDBModel.csdl|res://\*/AccountDBModel.ssdl|res://\*/AccountDBModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=GREG-SURFACE4\SQLEXPRESS;initial catalog=ACCOUNTDB;integrated security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

<add name="ACCOUNTDBEntities1" connectionString="metadata=res://\*/HoldingDBModel.csdl|res://\*/HoldingDBModel.ssdl|res://\*/HoldingDBModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=GREG-SURFACE4\SQLEXPRESS;initial catalog=ACCOUNTDB;integrated security=True;encrypt=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />

</startup>

<entityFramework>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

</providers>

</entityFramework>

</configuration>

### Output

To migrate your **app.config** file from a VB.NET project to an **appsettings.json** file for a .NET 8.0 Blazor Server application, you'll primarily need to convert the connection strings to JSON format. The **entityFramework** configuration section from your **app.config** typically does not need to be directly migrated to **appsettings.json**, as EF Core uses a different configuration approach compared to Entity Framework 6.

Here's how your connection strings can be migrated to **appsettings.json**:

{

"ConnectionStrings": {

"QUOTEDBConnectionString": "Data Source=GREG-SURFACE4\\SQLEXPRESS;Initial Catalog=QUOTEDB;Integrated Security=True;TrustServerCertificate=True",

"ACCOUNTDBConnectionString": "Data Source=GREG-SURFACE4\\SQLEXPRESS;Initial Catalog=ACCOUNTDB;Integrated Security=True;TrustServerCertificate=True",

"QUOTEDBEntities": "metadata=res://\*/QuoteModel.csdl|res://\*/QuoteModel.ssdl|res://\*/QuoteModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=GREG-SURFACE4\\SQLEXPRESS;initial catalog=QUOTEDB;integrated security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework\"",

"ACCOUNTDBEntities": "metadata=res://\*/AccountDBModel.csdl|res://\*/AccountDBModel.ssdl|res://\*/AccountDBModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=GREG-SURFACE4\\SQLEXPRESS;initial catalog=ACCOUNTDB;integrated security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework\"",

"ACCOUNTDBEntities1": "metadata=res://\*/HoldingDBModel.csdl|res://\*/HoldingDBModel.ssdl|res://\*/HoldingDBModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=GREG-SURFACE4\\SQLEXPRESS;initial catalog=ACCOUNTDB;integrated security=True;encrypt=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework\""

}

}

In the JSON snippet:

* Each connection string from your **app.config** is transformed into a JSON key-value pair within the **"ConnectionStrings"** object.
* Note that backslashes (**\**) in connection strings are escaped with an additional backslash (**\\**) in JSON format.
* The **metadata** parts of the connection strings for **QUOTEDBEntities**, **ACCOUNTDBEntities**, and **ACCOUNTDBEntities1** are kept as-is. However, it's important to note that if you are migrating to EF Core, you might not need these metadata parts, as EF Core handles model configuration differently compared to Entity Framework 6.

After adding these connection strings to **appsettings.json**, you can access them in your Blazor Server application using the **IConfiguration** interface, typically in your **Startup.cs** or wherever you configure services:

builder.Services.AddDbContext<YourDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("QUOTEDBConnectionString")));

Replace **YourDbContext** with the actual name of your DbContext class, and use the appropriate connection string key as needed.

### Notes

So, I edited my appsettings.json file, and added all the connection strings.  
  
I now need to work on my code fixup in my accountCurrent.cs class.  
  
First, VS tells me it cannot find SqlParamater in System.Data.SqlClient.

## ChatGTP Interaction 15 - Output Code Optimization/Fixup

## Input

In my Blazor class, I am using the following line of code: var parameter = new SqlParameter("@userId", userid); But it says "SqlParameter could not be found in the namespace System.Data.SqlClient" How to fix?

### Output

The error you're encountering suggests that the necessary namespace or assembly for **SqlParameter** is not being recognized in your Blazor Server project. In .NET 8.0 and other .NET Core or .NET 5/6/7/8 applications, **SqlParameter** is available within the **System.Data.SqlClient** namespace, which is part of the **System.Data.SqlClient** package.

To resolve this issue, you should ensure that the **System.Data.SqlClient** package is installed and properly referenced in your project. Here's how you can fix this:

1. **Install the System.Data.SqlClient Package**: If you haven't already, you need to add the **System.Data.SqlClient** NuGet package to your project. This can be done via the NuGet Package Manager in Visual Studio or by using the following .NET CLI command:

dotnet add package System.Data.SqlClient

**Add the Using Directive**: Ensure that your code file includes a using directive for the **System.Data.SqlClient** namespace at the top:

using System.Data.SqlClient;

After installing the package and adding the using directive, your code snippet should work as expected:

var parameter = new SqlParameter("@userId", userid);

If you've already done these steps and the issue persists, consider the following additional troubleshooting steps:

* **Ensure Correct Target Framework**: Make sure your project targets a framework that is compatible with **System.Data.SqlClient**. For .NET 8.0 Blazor Server applications, this should not be an issue, but it's good to double-check.
* **Check for Typos**: Ensure there are no typos in the using directive or in the package reference.
* **Restore NuGet Packages**: Sometimes, restoring NuGet packages can resolve reference issues. You can do this via Visual Studio or by running **dotnet restore** in the terminal within your project directory.

If the problem continues, you might want to check your project's **.csproj** file to ensure the **System.Data.SqlClient** package reference is correctly listed there.

### Notes

I found “Microsoft.Data.SqlClient” in nuGet package manager. I also found the recommended “System.Data.SqlClient:, but the Microsoft.Data.SqlClient seems newer, so I downloaded this package within my project.

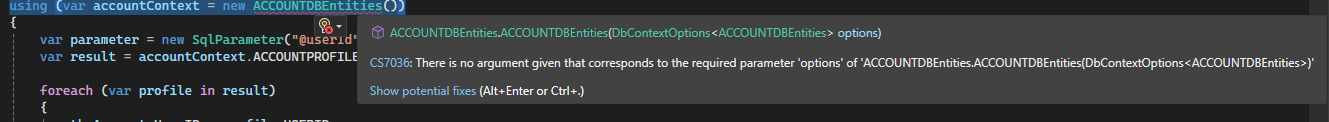
This fixed the issue. Now onto continued code fixup in accountCurrent.cs.  
  
Next issue is it cannot find ACCOUNTDBEntities class. Missing assembly reference?

I believe this is because I separated Data classes into a Data folder, and it exists in a different namespace than the business classes. So will add a using in to TradeBlazorApp.Data\_Classes in my accountCurrent.cs class.

This fixed that error, but now on this line in accountCurrent.cs

using (var accountContext = new ACCOUNTDBEntities())

I get:



I am finding I need to go back to ChatGTP Interaction 12, and re-read the output. This seems to contain key information on my DBEntities.

## ChatGTP Interaction 16: Output Code Optimization/Fixup

### Input

I have the following in .NET EntityFrameworkcore Blazor Server app:   
"public class ACCOUNTDBEntities : DbContext { public ACCOUNTDBEntities(DbContextOptions<ACCOUNTDBEntities> options) : base(options) { } public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; } }"

generated by ChatGTP. However, when creating a new instance of this class with this line of code in another class (accountCurrent):

"using (var accountContext = new ACCOUNTDBEntities())"

I get this error in Visual Studio 2022: "There is no argument given that corresponds to the required paramater options..." How to fix?

### Output

The error you're encountering is because the **ACCOUNTDBEntities** class constructor expects a parameter of type **DbContextOptions<ACCOUNTDBEntities>**, but you're trying to instantiate it without providing any arguments. In ASP.NET Core and Blazor Server applications, the recommended way to work with **DbContext** instances is through dependency injection (DI) rather than manually instantiating them with the **new** keyword.

Here's how you can fix the issue:

**1. Register ACCOUNTDBEntities with Dependency Injection**

Ensure that your **ACCOUNTDBEntities** context is registered with the DI container in the **Startup.cs** or **Program.cs** file of your Blazor Server project. This is typically done in the **ConfigureServices** method:

public void ConfigureServices(IServiceCollection services)

{

// Other service registrations...

// Register your DbContext

services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

}

Make sure to replace **"YourConnectionStringName"** with the actual name of your connection string defined in **appsettings.json**.

**2. Use Dependency Injection to Obtain an Instance of ACCOUNTDBEntities**

Instead of manually creating a new instance of **ACCOUNTDBEntities**, you should use dependency injection to obtain an instance. You can do this by injecting **ACCOUNTDBEntities** into your components or services where you need to use it.

In a Razor Component:

@inject ACCOUNTDBEntities AccountContext

@code {

// Use AccountContext here

}

In a Class:

For a class, you would typically pass the **ACCOUNTDBEntities** instance through the constructor:

public class AccountCurrent

{

private readonly ACCOUNTDBEntities \_accountContext;

public AccountCurrent(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

// Use \_accountContext here

}

And ensure that **AccountCurrent** is also resolved via dependency injection where it's used, so that an instance of **ACCOUNTDBEntities** is appropriately injected.

**Summary**

By using dependency injection, you not only adhere to the best practices of ASP.NET Core and Blazor Server applications but also avoid issues related to the lifecycle management of your **DbContext** instances. This approach enhances the maintainability and scalability of your application.

### Notes

Step 2 is key—I need to use **dependency injection!**  
Copying code from “In a class” section to do this.  
  
Ok—did it. I had to then manually (though suppose ChatGTP would have helped):  
  
var theAccount = new AccountCurrent(\_accountContext);

//I needed to figure out, after ChatGTP interaction 16, to pass \_accountContext here in constructor.

And also change to the following, in this using statement:  
  
using (\_accountContext) //I needed to manually change this line of code to use \_accountContext, after reading ChatGTP interaction 16 (and pasting code at top here)

//as before it read: using (var accountContext = new ACCOUNTDBEntities()). ChatGTP is having me use autocontext via dependency injection instead

Next issue, is new SqlClient from Microsoft (vs what was in my VB.NET app), does not support SqlQuery method per below:  
  
var result = \_accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter); //

## ChatGTP Interaction 17:

### Input

For questions that follow, assume I am targeting a .NET 8 Blazor server app, using the latest EntityFramework core, and using Microsoft.Data.SqlClient when migrating old code from a VB.NET app using System.Data.SqlClient.

### Output

Understood. I'll keep in mind that your questions are in the context of a .NET 8 Blazor Server application, using the latest Entity Framework Core, and transitioning from **System.Data.SqlClient** in a VB.NET application to **Microsoft.Data.SqlClient**. Feel free to proceed with your questions!

### Notes

## ChatGTP Interaction 18 – Output Code Optimization/Fixup

### Input

How do I fix the error message "No overload for method SqlQuery takes two arguments" in this line of migrated code: var result = \_accountContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter); //

### Output

In Entity Framework Core, the **SqlQuery** method you might be familiar with from previous versions of Entity Framework (prior to Core) does not exist. Instead, EF Core uses different methods to execute raw SQL queries, particularly **FromSqlRaw** or **FromSqlInterpolated** for querying data into entities. Additionally, when using **Microsoft.Data.SqlClient**, you should ensure that parameters are passed correctly to these methods.

Given the code snippet you've provided, you can achieve a similar result using the **FromSqlRaw** method along with **SqlParameter**. Here's how you can modify your code:

1. First, ensure you have the **Microsoft.Data.SqlClient** package installed, as you're migrating from using **System.Data.SqlClient**.
2. Use **FromSqlRaw** or **FromSqlInterpolated** to execute your raw SQL query.

Here's an example using **FromSqlRaw**:

using Microsoft.Data.SqlClient;

// Other using directives...

var userIdParameter = new SqlParameter("@userId", userId);

var result = \_accountContext.ACCOUNTPROFILEs

.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", userIdParameter)

.ToList();

Or, if you prefer using string interpolation which provides a more readable syntax, you can use **FromSqlInterpolated**:

var result = \_accountContext.ACCOUNTPROFILEs

.FromSqlInterpolated($"SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = {userId}")

.ToList();

The **FromSqlInterpolated** method automatically parameterizes the input variables, reducing the risk of SQL injection.

**Important Notes**:

* Ensure that your query returns all columns of the **ACCOUNTPROFILE** entity because EF Core needs to map all properties of the entity. If the query does not return all columns, you might run into issues with missing data in your entity instances.
* These methods should be used carefully to avoid SQL injection vulnerabilities, especially when incorporating user input into the SQL query. The **FromSqlInterpolated** method helps mitigate this risk by parameterizing the input.
* Using raw SQL queries bypasses EF Core's ability to generate and optimize SQL based on your LINQ queries, so it's generally recommended to use LINQ wherever possible. Raw SQL queries should be reserved for complex scenarios where LINQ is not sufficient.

### Notes

OK: I am at a crossroads here. Clearly, when ChatGTP migrated my VB.NET AccountCurrent class, it did not also migrate to the newer namespaces and new version of the EntityFramework packages from Microsoft (8.0.2) which have different namespace, concepts, syntax and usage. I could:

1. Continue to fixup offended lines in generated class, line by line. Or
2. Attempt to re-generate the entire class again, making sure to tell ChatGTP to use the new version of EntityFrameworkCore, 8.0.2.

Which approach?  
  
I will first try to regenerate the entire class, with a more targeted ChatGTP query.

Depending on success, I might use this, or go back to line-by-line approach. It also wants me to use LINQ queries, vs my embedded queries from VB, to be more “modern” and in keeping with latest EntityFrameworkCore.

## ChatGTP Interaction 19:

### Input

Assume I am migrating a VB.NET 8.0 app that uses Microsoft Entity Framework 6.4.4 to a .NET 8.0 Blazor app that uses newer Microsoft Entity FrameworkCore 8.02, in the following questions

### Output

Understood. As you migrate from a VB.NET application using Entity Framework 6.4.4 to a .NET 8.0 Blazor application with Entity Framework Core 8.0.2, feel free to ask any questions related to this migration process, including differences in EF Core, code conversion, best practices, or any specific issues you encounter along the way.

### Notes

## ChatGTP Interaction 20 – Direct Code Migration Business Logic Tier

### Input

Can you migrate this VB.NET class to a class in Blazor Server app? Imports System.Data.Entity Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder Imports System.Data.Entity.Migrations Imports System.Data.SqlClient Imports System.Runtime.ConstrainedExecution Imports System.Runtime.Remoting.Contexts Public Class AccountCurrent Public Property Password As String Public Property UserID As String Public Property Name As String Public Property AccountID As Decimal Public Property Salt As String Public Function Login(userid As String, password As String) As AccountCurrent Dim theAccount As New AccountCurrent theAccount.UserID = userid Using accountcontext As New ACCOUNTDBEntities() 'START WHERE OK 'Dim result = accountcontext.ACCOUNTPROFILEs.SqlQuery("Select \* FROM dbo.ACCOUNTPROFILE WHERE USERID = '" + userid + "'") ' Assuming "userid" is a variable containing the user's ID you're interested in Dim parameter = New SqlParameter("@userId", userid) ' Execute the SQL query using a parameterized query for safety Dim result = accountcontext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter) ' Iterate over the results For Each profile In result ' Access the properties/columns of each ACCOUNTPROFILE entity theAccount.UserID = profile.USERID theAccount.Password = profile.PASSWORD theAccount.AccountID = profile.ACCOUNTID theAccount.Salt = profile.SALT theAccount.Name = profile.FULLNAME ' Continue accessing other columns/properties as needed Next If theAccount.Password Is Nothing Then GlobalSettings.LoggedIn = False Else Dim ver = New SaltedHash().Create(theAccount.Salt, theAccount.Password) GlobalSettings.LoggedIn = ver.Verify(password) End If End Using Return theAccount End Function

### Output

To migrate your VB.NET class to a C# class for a .NET 8.0 Blazor Server application using Entity Framework Core 8.0.2, you'll need to make several adjustments. These include language syntax changes, modifications to accommodate EF Core's API differences from EF 6, and best practices for handling database contexts and queries in a Blazor Server app.

Here's a converted and updated version of your class:

using Microsoft.EntityFrameworkCore;

using Microsoft.Data.SqlClient;

using System;

public class AccountCurrent

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

// Assuming ACCOUNTDBEntities is your DbContext and ACCOUNTPROFILEs is a DbSet<AccountProfile>

private readonly ACCOUNTDBEntities \_accountContext;

// Constructor injection for the DbContext

public AccountCurrent(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public AccountCurrent Login(string userid, string password)

{

var theAccount = new AccountCurrent(\_accountContext);

// Use EF Core's method to execute SQL directly, ensure parameterization for security

var parameter = new SqlParameter("@userId", userid);

var accountProfiles = \_accountContext.ACCOUNTPROFILEs

.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter)

.ToList(); // Execute the query and get the result

foreach (var profile in accountProfiles)

{

theAccount.UserID = profile.UserID;

theAccount.Password = profile.Password;

theAccount.AccountID = profile.AccountID;

theAccount.Salt = profile.Salt;

theAccount.Name = profile.FullName;

// Continue mapping other properties as needed

}

if (theAccount.Password == null)

{

// Assuming GlobalSettings is a static class you use for global settings

GlobalSettings.LoggedIn = false;

}

else

{

// Assuming SaltedHash is a class you have for hashing and verifying passwords

var ver = new SaltedHash().Create(theAccount.Salt, theAccount.Password);

GlobalSettings.LoggedIn = ver.Verify(password);

}

return theAccount;

}

}

To migrate your VB.NET class to a C# class for a .NET 8.0 Blazor Server application using Entity Framework Core 8.0.2, you'll need to make several adjustments. These include language syntax changes, modifications to accommodate EF Core's API differences from EF 6, and best practices for handling database contexts and queries in a Blazor Server app.

Here's a converted and updated version of your class:

using Microsoft.EntityFrameworkCore;

using Microsoft.Data.SqlClient;

using System;

public class AccountCurrent

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

// Assuming ACCOUNTDBEntities is your DbContext and ACCOUNTPROFILEs is a DbSet<AccountProfile>

private readonly ACCOUNTDBEntities \_accountContext;

// Constructor injection for the DbContext

public AccountCurrent(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public AccountCurrent Login(string userid, string password)

{

var theAccount = new AccountCurrent(\_accountContext);

// Use EF Core's method to execute SQL directly, ensure parameterization for security

var parameter = new SqlParameter("@userId", userid);

var accountProfiles = \_accountContext.ACCOUNTPROFILEs

.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter)

.ToList(); // Execute the query and get the result

foreach (var profile in accountProfiles)

{

theAccount.UserID = profile.UserID;

theAccount.Password = profile.Password;

theAccount.AccountID = profile.AccountID;

theAccount.Salt = profile.Salt;

theAccount.Name = profile.FullName;

// Continue mapping other properties as needed

}

if (theAccount.Password == null)

{

// Assuming GlobalSettings is a static class you use for global settings

GlobalSettings.LoggedIn = false;

}

else

{

// Assuming SaltedHash is a class you have for hashing and verifying passwords

var ver = new SaltedHash().Create(theAccount.Salt, theAccount.Password);

GlobalSettings.LoggedIn = ver.Verify(password);

}

return theAccount;

}

}

**Key Changes and Considerations:**

1. **DbContext Injection:** Instead of creating a new **DbContext** instance inside the method, it's injected through the constructor, which is a recommended practice in ASP.NET Core for better performance and testability.
2. **Raw SQL Queries:** The **SqlQuery** method does not exist in EF Core. Instead, use **FromSqlRaw** for executing raw SQL queries. Ensure that any raw SQL is parameterized to avoid SQL injection attacks.
3. **Error Handling:** Consider adding error handling (e.g., try-catch blocks) around database access to manage exceptions gracefully.
4. **GlobalSettings and SaltedHash:** Ensure that classes like **GlobalSettings** and **SaltedHash** are correctly migrated and accessible in your Blazor Server application context.
5. **DbContext Lifetime:** Ensure the lifetime of **ACCOUNTDBEntities** is correctly scoped in your dependency injection setup (typically scoped to a request in web applications).
6. **Security Considerations:** Be very careful with raw SQL queries to prevent SQL injection. The parameterization in the example helps, but always validate and sanitize inputs.

This code assumes the existence of certain classes (**GlobalSettings**, **SaltedHash**, **AccountProfile**) and properties based on your original VB.NET code. You might need to adjust the code further based on the actual structure of these classes and the overall architecture of your Blazor Server application.

### Notes

The new AccountCurrent class Builds successfully! So throwing away my first attempt at migration, key is to give ChatGTP this BEFORE migrating the class:  
 ***“Assume I am migrating a VB.NET 8.0 app that uses Microsoft Entity Framework 6.4.4 to a .NET 8.0 Blazor app that uses newer Microsoft Entity FrameworkCore 8.02, in the following questions”***

ON to migrating next and final Business Logic class from VB.NET: Stock.cs

## ChatGTP Interaction 21 - Direct Code Migration Business Logic Tier

### Input

Can you migrate the following VB.NET class to a Blazor class for my new Blazor Server app? Imports System.Data.Entity Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder Imports System.Data.Entity.Migrations Imports System.Data.SqlClient Public Class Stock ' Constructor for the Stock class Public Sub New() ' Initialization code can go here End Sub ' Method to buy stock Public Sub Buy(row As DataGridViewRow, stockSymbol As String, quantity As Integer) Dim orderdate As DateTime = DateAndTime.Now Try 'First, lets lookup the account via the combined primary key Using accountcontext As New ACCOUNTDBEntities() Dim theAccount As New ACCOUNT() theAccount.PROFILE\_USERID = GlobalSettings.theLoginAccount.UserID theAccount.ACCOUNTID = GlobalSettings.theLoginAccount.AccountID theAccount = accountcontext.ACCOUNTs.Find(theAccount.ACCOUNTID, theAccount.PROFILE\_USERID) 'Console.WriteLine("Account is " + theAccount.CREATIONDATE.ToString) ' Now let's create a new Order object for the Orders table to insert as new row Dim newOrder As New ORDER() With { .ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID, .COMPLETIONDATE = orderdate, .HOLDING\_HOLDINGID = Guid.NewGuid(), .OPENDATE = orderdate, .ORDERFEE = 1.0, .ORDERSTATUS = "open", .ORDERTYPE = "buy", .QUOTE\_SYMBOL = row.Cells("Symbol").Value, .PRICE = row.Cells("Price").Value, .QUANTITY = quantity, .USERID = theAccount.PROFILE\_USERID, .ORDERID = Guid.NewGuid()} 'Insert and Commit! ' This will be a distributed transaction, must also insert into Holdings Table Dim newHolding As New HOLDING() With { .ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID, .PURCHASEDATE = orderdate, .PURCHASEPRICE = newOrder.PRICE, .QUANTITY = quantity, .QUOTE\_SYMBOL = newOrder.QUOTE\_SYMBOL, .USERID = newOrder.USERID, .HOLDINGID = newOrder.HOLDING\_HOLDINGID} accountcontext.ORDERS.Add(newOrder) accountcontext.HOLDINGs.Add(newHolding) 'Commit! accountcontext.SaveChanges() 'Now close the order newOrder = accountcontext.ORDERS.Find(newOrder.ORDERID, newOrder.ACCOUNT\_ACCOUNTID) newOrder.ORDERSTATUS = "closed" 'Commit accountcontext.SaveChanges() MessageBox.Show("You Just Bought " + quantity.ToString() + " shares of " + row.Cells("Symbol").Value) ' Set other properties as needed End Using Catch ex As DataException ' Handle database-related exceptions MessageBox.Show("Database error: " & ex.Message) Catch ex As ApplicationException ' Handle application-specific exceptions MessageBox.Show("Application error: " & ex.Message) Catch ex As Exception ' Handle unexpected exceptions MessageBox.Show("An unexpected error occurred: " & ex.Message) End Try End Sub Public Sub Sell(row As DataGridViewRow, stockSymbol As String, quantity As Integer) Dim orderdate As DateTime = DateAndTime.Now Dim holdingID As Guid holdingID = row.Cells("HOLDINGID").Value Dim theStock As New QUOTE Try Using quoteContext As New QUOTEDBEntities() Dim parameter = New SqlParameter("@quotesymbol", stockSymbol) Dim query = quoteContext.QUOTEs.SqlQuery("SELECT \* FROM dbo.QUOTE WHERE SYMBOL = @quotesymbol", parameter) ' Materialize the results into a List (or another suitable collection) Dim result As List(Of QUOTE) = query.ToList() theStock = result(0) End Using 'First, lets lookup the account via the combined primary key Using accountcontext As New ACCOUNTDBEntities() Dim theAccount As New ACCOUNT() theAccount.PROFILE\_USERID = GlobalSettings.theLoginAccount.UserID theAccount.ACCOUNTID = GlobalSettings.theLoginAccount.AccountID theAccount = accountcontext.ACCOUNTs.Find(theAccount.ACCOUNTID, theAccount.PROFILE\_USERID) 'Console.WriteLine("Account is " + theAccount.CREATIONDATE.ToString) ' Now let's create a new Order object for the Orders table to insert as new row Dim newOrder As New ORDER() With { .ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID, .COMPLETIONDATE = orderdate, .HOLDING\_HOLDINGID = Guid.NewGuid(), .OPENDATE = orderdate, .ORDERFEE = 1.0, .ORDERSTATUS = "open", .ORDERTYPE = "sell", .QUOTE\_SYMBOL = row.Cells("Quote\_Symbol").Value, .PRICE = theStock.PRICE, .QUANTITY = quantity, .USERID = theAccount.PROFILE\_USERID, .ORDERID = Guid.NewGuid()} 'Insert and Commit! ' This will be a distributed transaction, must also update the Holdings Table Dim theHolding As New HOLDING() theHolding = accountcontext.HOLDINGs.Find(holdingID, theAccount.ACCOUNTID) 'Update Holding Quantity theHolding.QUANTITY = theHolding.QUANTITY - quantity 'If new quantity is zero, delete the holding from the DB If theHolding.QUANTITY = 0 Then accountcontext.HOLDINGs.Remove(theHolding) End If 'Add the Order to the Orders Table accountcontext.ORDERS.Add(newOrder) 'Commit! accountcontext.SaveChanges() 'Now close the order newOrder = accountcontext.ORDERS.Find(newOrder.ORDERID, newOrder.ACCOUNT\_ACCOUNTID) newOrder.ORDERSTATUS = "closed" 'Commit accountcontext.SaveChanges() MessageBox.Show("You Just Sold " + quantity.ToString() + " shares of " + row.Cells("Quote\_Symbol").Value) ' Set other properties as needed End Using Catch ex As DataException ' Handle database-related exceptions MessageBox.Show("Database error: " & ex.Message) Catch ex As ApplicationException ' Handle application-specific exceptions MessageBox.Show("Application error: " & ex.Message) Catch ex As Exception ' Handle unexpected exceptions MessageBox.Show("An unexpected error occurred: " & ex.Message) End Try End Sub End Class

### Output

Migrating your VB.NET class to a C# class for a Blazor Server application involves several adjustments. These include converting VB.NET syntax to C#, adapting Entity Framework usage to EF Core, and replacing Windows Forms components (like **DataGridViewRow** and **MessageBox**) with equivalents suitable for a web application context.

Here's a starting point for migrating your **Stock** class to C# for use in a Blazor Server app. Note that some modifications are necessary to fit the web context, such as handling user feedback through the UI rather than using **MessageBox**.

using Microsoft.EntityFrameworkCore;

using Microsoft.Data.SqlClient;

using System;

using System.Linq;

using System.Threading.Tasks;

public class Stock

{

private readonly ACCOUNTDBEntities \_accountContext;

private readonly QUOTEDBEntities \_quoteContext;

public Stock(ACCOUNTDBEntities accountContext, QUOTEDBEntities quoteContext)

{

\_accountContext = accountContext;

\_quoteContext = quoteContext;

}

public async Task<string> Buy(string stockSymbol, int quantity)

{

var orderDate = DateTime.Now;

var message = string.Empty;

try

{

var theAccount = await \_accountContext.ACCOUNTs

.FindAsync(GlobalSettings.TheLoginAccount.AccountID, GlobalSettings.TheLoginAccount.UserID);

var newOrder = new ORDER

{

ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID,

COMPLETIONDATE = orderDate,

HOLDING\_HOLDINGID = Guid.NewGuid(),

OPENDATE = orderDate,

ORDERFEE = 1.0M,

ORDERSTATUS = "open",

ORDERTYPE = "buy",

QUOTE\_SYMBOL = stockSymbol,

// Assuming PRICE is a decimal type

PRICE = decimal.Parse(stockSymbol), // You need to replace this with actual price retrieval logic

QUANTITY = quantity,

USERID = theAccount.PROFILE\_USERID,

ORDERID = Guid.NewGuid()

};

var newHolding = new HOLDING

{

ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID,

PURCHASEDATE = orderDate,

PURCHASEPRICE = newOrder.PRICE,

QUANTITY = quantity,

QUOTE\_SYMBOL = newOrder.QUOTE\_SYMBOL,

USERID = newOrder.USERID,

HOLDINGID = newOrder.HOLDING\_HOLDINGID

};

\_accountContext.ORDERS.Add(newOrder);

\_accountContext.HOLDINGs.Add(newHolding);

await \_accountContext.SaveChangesAsync();

newOrder.ORDERSTATUS = "closed";

await \_accountContext.SaveChangesAsync();

message = $"You just bought {quantity} shares of {stockSymbol}.";

}

catch (Exception ex)

{

// Handle exceptions

message = $"An error occurred: {ex.Message}";

}

return message;

}

public async Task<string> Sell(string stockSymbol, int quantity)

{

var orderDate = DateTime.Now;

var message = string.Empty;

try

{

var theStock = await \_quoteContext.QUOTEs

.FromSqlInterpolated($"SELECT \* FROM dbo.QUOTE WHERE SYMBOL = {stockSymbol}")

.FirstOrDefaultAsync();

if (theStock == null)

{

return "Stock not found.";

}

var theAccount = await \_accountContext.ACCOUNTs

.FindAsync(GlobalSettings.TheLoginAccount.AccountID, GlobalSettings.TheLoginAccount.UserID);

var newOrder = new ORDER

{

ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID,

COMPLETIONDATE = orderDate,

HOLDING\_HOLDINGID = Guid.NewGuid(),

OPENDATE = orderDate,

ORDERFEE = 1.0M,

ORDERSTATUS = "open",

ORDERTYPE = "sell",

QUOTE\_SYMBOL = stockSymbol,

PRICE = theStock.PRICE,

QUANTITY = quantity,

USERID = theAccount.PROFILE\_USERID,

ORDERID = Guid.NewGuid()

};

\_accountContext.ORDERS.Add(newOrder);

await \_accountContext.SaveChangesAsync();

newOrder.ORDERSTATUS = "closed";

await \_accountContext.SaveChangesAsync();

message = $"You just sold {quantity} shares of {stockSymbol}.";

}

catch (Exception ex)

{

// Handle exceptions

message = $"An error occurred: {ex.Message}";

}

return message;

}

}

**Key Considerations:**

1. **Dependency Injection:** The **ACCOUNTDBEntities** and **QUOTEDBEntities** contexts are injected into the **Stock** class, aligning with ASP.NET Core's dependency injection best practices.
2. **Async/Await:** Methods are made asynchronous (**async Task<string>**) to better align with scalable web application practices, improving performance and scalability.
3. **User Feedback:** Instead of **MessageBox**, which is not available in Blazor

### Notes

This class is going to be a challenge, cause it mixes VB.NET WInforms UI with logic code. It is not well-factored from the source VB.NET app!

It also uses many more EntityFramework data classes.

## ChatGTP Interaction 22 – Output Code Optimization/Fixup

### Input

Visual Studio in my new Blazor app cannot find references to QUOTEDBEntities, in this code, how to fix? " private readonly QUOTEDBEntities \_quoteContext; public Stock(ACCOUNTDBEntities accountContext, QUOTEDBEntities quoteContext) { \_accountContext = accountContext; \_quoteContext = quoteContext; }

### Output

If Visual Studio cannot find references to **QUOTEDBEntities** in your Blazor app, there are several steps you can take to troubleshoot and fix the issue:

1. **Ensure DbContext is Defined:**
   * Make sure that **QUOTEDBEntities** is defined in your project and that it inherits from **DbContext**. It should be something like this:

public class QUOTEDBEntities : DbContext

{

public QUOTEDBEntities(DbContextOptions<QUOTEDBEntities> options) : base(options)

{

}

// DbSets for your entities

public DbSet<Quote> Quotes { get; set; }

// Other DbSets...

}

**Check Namespace:**

* Ensure that the namespace where **QUOTEDBEntities** is defined is correctly used in the file where you're trying to use it. If it's in a different namespace, you'll need to import it with a **using** statement at the top of your C# file:

using YourNamespaceWhereQUOTEDBEntitiesIsDefined;

1. **Add Required NuGet Packages:**
   * Make sure you have all necessary Entity Framework Core NuGet packages installed in your project, especially **Microsoft.EntityFrameworkCore** and any provider-specific package (like **Microsoft.EntityFrameworkCore.SqlServer** if you're using SQL Server).
2. **Register DbContext in Startup:**
   * In your Blazor Server app's **Startup.cs** or **Program.cs** (depending on the template and .NET version), ensure that **QUOTEDBEntities** is registered with the dependency injection container:

builder.Services.AddDbContext<QUOTEDBEntities>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("YourQuoteDbConnectionString")));

* + Make sure to replace **"YourQuoteDbConnectionString"** with the actual key to your connection string defined in **appsettings.json**.

1. **Clean and Rebuild Solution:**
   * Sometimes, stale build artifacts can cause issues. Try cleaning your solution (**Build > Clean Solution**) and then rebuilding it (**Build > Rebuild Solution**).
2. **Restore NuGet Packages:**
   * Ensure all NuGet packages are correctly restored. You can right-click on the solution in Solution Explorer and select "Restore NuGet Packages".
3. **Check for Typos and Errors:**
   * A simple typo in the class name, namespace, or file structure can lead to such issues. Double-check for any such mistakes.
4. **Close and Reopen Visual Studio:**
   * Occasionally, Visual Studio might get into a state where it doesn't recognize certain references correctly. Closing and reopening Visual Studio can sometimes resolve these issues.

If, after trying these steps, you still face issues, it might be helpful to examine the exact error messages or warnings Visual Studio provides, as they can offer more specific guidance on what's going wrong.

### Notes

OK—this is similar/same process I used in interaction to migrate ACCOUNTDBEntities. I remember I am placing database model classes in my new Blazor app in a separate namespace/folder: “Data Classes”.   
  
So will being there, creating a new class QUOTEDBEntities per instructions above.

I generated the new class per instructions. However, I now need to define a model class for Quote:  
  
public DbSet<Quote> Quotes { get; set; }

// Other DbSets...

I am going to try to do this from the database DDL, generated by SQL Server Management Studio:

# Phase 4 – Revisiting the Data Tier and Migrating In a Better Way – Starting From the DBs Themselves

## ChatGTP Interaction 23 - Direct Code Generation Data Tier Classes from DB DDL

Need to start from Database Layer!

1) Understanding (scanning?):  All the databases/tables the source app touches

2) Now, use DDL (generated by SQL Server Management Studio) “As Create Table)” to build Entity FrameworkCore 8.0.2 model classes.

3) As FIRST STEP!  Then, go onto business/logic classes (and later UI classes).  Need a clean starting point here, ChatGTP cannot understand, without being able to look at holistic solution–just piecemeal.

So, kinda new start, but not all wasted work above.

### Input

I want to create an Entity Model class called Quote for Microsoft Entity Framework 8.0.2 from the following table definition, can you generate the class? CREATE TABLE [dbo].[QUOTE]( [LOW] [decimal](12, 2) NULL, [OPEN1] [decimal](12, 2) NULL, [VOLUME] [float] NOT NULL, [PRICE] [decimal](12, 2) NULL, [HIGH] [decimal](12, 2) NULL, [COMPANYNAME] [varchar](250) NULL, [SYMBOL] [varchar](10) NOT NULL, [CHANGE1] [float] NOT NULL, [QUOTEID] [bigint] NOT NULL, CONSTRAINT [PK\_QUOTE] PRIMARY KEY CLUSTERED ( [SYMBOL] ASC, [QUOTEID] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY]

### Output

To create an Entity Model class for Entity Framework Core 8.0.2 based on your table definition, you can define a C# class that represents the **QUOTE** table. Each column in your table will correspond to a property in the class. For the **QUOTEID** and **SYMBOL** columns, which are part of the primary key, you'll use the **[Key]** attribute to denote them as key properties. Since your primary key is clustered and consists of two columns, you'll also use the **[Column(Order = n)]** attribute to specify the order of the key columns.

Here's how you could define the **Quote** class:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("QUOTE", Schema = "dbo")]

public class Quote

{

[Column("LOW", TypeName = "decimal(12, 2)")]

public decimal? Low { get; set; }

[Column("OPEN1", TypeName = "decimal(12, 2)")]

public decimal? Open1 { get; set; }

[Column("VOLUME", TypeName = "float")]

[Required]

public float Volume { get; set; }

[Column("PRICE", TypeName = "decimal(12, 2)")]

public decimal? Price { get; set; }

[Column("HIGH", TypeName = "decimal(12, 2)")]

public decimal? High { get; set; }

[Column("COMPANYNAME", TypeName = "varchar(250)")]

public string CompanyName { get; set; }

[Key]

[Column("SYMBOL", TypeName = "varchar(10)", Order = 0)]

[Required]

public string Symbol { get; set; }

[Column("CHANGE1", TypeName = "float")]

[Required]

public float Change1 { get; set; }

[Key]

[Column("QUOTEID", TypeName = "bigint", Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)] // Assuming QUOTEID is not auto-generated

public long QuoteId { get; set; }

}

**Key Points:**

* **Nullable Types:** Columns that are allowed to be **NULL** in the database are represented by nullable types in C# (e.g., **decimal?** for the **LOW**, **OPEN1**, **PRICE**, and **HIGH** columns).
* **Data Annotations:** The **[Table]** attribute specifies the database table name, and the **[Column]** attribute specifies the column name and type. The **[Required]** attribute is used for non-nullable columns.
* **Composite Key:** The **[Key]** attribute and **[Column(Order = n)]** are used together to define a composite primary key. The **Order** property specifies the position of the column in the composite key.
* **DatabaseGenerated Attribute:** The **[DatabaseGenerated(DatabaseGeneratedOption.None)]** attribute is used for **QUOTEID** to indicate that its value is not auto-generated by the database. This can be changed to **DatabaseGeneratedOption.Identity** or **DatabaseGeneratedOption.Computed** if the database generates this value.

This class structure should align well with your database schema and allow Entity Framework Core to correctly map between your database and your .NET objects.

### Notes

Ok, implemented my Quote class in my Data Classes folder.  
  
Now on to ORDER object, which needs an Entity Model.

## ChatGTP Interaction 24 – Output Code Optimization/Fixup

### Input

Visual STudio in my new Blazor app cannot find references to ORDER object, which I want to define as a model class for .NET Entity FrameworkCore 8.0.2 from my SQL Server Orders table, how to fix:

var newOrder = new ORDER

{

ACCOUNT\_ACCOUNTID = theAccount.AccountID, //Manually fixed to point to AccountID vs. what ChatGTP generated ("ACCOUNTID") interaction 23

COMPLETIONDATE = orderDate,

HOLDING\_HOLDINGID = Guid.NewGuid(),

OPENDATE = orderDate,

ORDERFEE = 1.0M,

ORDERSTATUS = "open",

ORDERTYPE = "buy",

QUOTE\_SYMBOL = stockSymbol,

// Assuming PRICE is a decimal type

PRICE = decimal.Parse(stockSymbol), // You need to replace this with actual price retrieval logic

QUANTITY = quantity,

USERID = theAccount.UserID, //Manually fixed to point to UserID vs. what ChatGTP generated (".PROFILE\_USERID") interaction 23

ORDERID = Guid.NewGuid()

};

### Output

If Visual Studio cannot find references to the **ORDER** object in your Blazor app, it likely means that the **ORDER** class hasn't been defined yet in your project, or there's an issue with namespaces or project references. Here's how you can address this:

**1. Define the ORDER Class**

First, ensure that you have an **ORDER** class defined in your project that corresponds to your **Orders** table in SQL Server. This class should be defined in a way that mirrors the structure of the table, with properties for each column. Here's an example of what the **ORDER** class might look like based on the properties you've used:

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("Orders")] // Adjust the table name as per your database schema

public class ORDER

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)] // Assuming ORDERID is auto-generated. Adjust as necessary.

public Guid ORDERID { get; set; }

public Guid ACCOUNT\_ACCOUNTID { get; set; } // Foreign key relation, adjust the type as necessary

public DateTime COMPLETIONDATE { get; set; }

public Guid HOLDING\_HOLDINGID { get; set; } // Foreign key relation, adjust as necessary

public DateTime OPENDATE { get; set; }

public decimal ORDERFEE { get; set; }

[MaxLength(10)]

public string ORDERSTATUS { get; set; }

[MaxLength(10)]

public string ORDERTYPE { get; set; }

[MaxLength(10)]

public string QUOTE\_SYMBOL { get; set; } // Adjust the MaxLength as per your column definition

public decimal PRICE { get; set; }

public int QUANTITY { get; set; }

public string USERID { get; set; } // Adjust the type as necessary

// Navigation properties and any additional columns go here

}

**2. Check Namespace**

If the **ORDER** class is defined in a different namespace than the one where you're trying to use it, you'll need to import that namespace with a **using** statement at the top of your C# file:

using YourNamespaceWhereOrderIsDefined;

**3. Add Entity Framework Core References**

Ensure that your project has the necessary Entity Framework Core package references, such as **Microsoft.EntityFrameworkCore** and a provider package like **Microsoft.EntityFrameworkCore.SqlServer** for SQL Server.

**4. Include ORDER in Your DbContext**

Make sure your **DbContext** (e.g., **ACCOUNTDBEntities**) includes a **DbSet<ORDER>** property for the **ORDER** entity:

public DbSet<ORDER> Orders { get; set; }

**5. Clean and Rebuild**

Sometimes, the issue can be resolved by cleaning and rebuilding your project. Right-click on your solution in Visual Studio and select "Clean Solution", then "Rebuild Solution".

**6. Check for Typos**

Ensure there are no typos in your class name, file names, or references.

If you've checked all the above and the issue persists, you may want to close and reopen Visual Studio or check the build output and Error List windows for more specific error messages that could give additional clues.

### Notes

Pasted code into a new “Orders.cs” class I created in Data Classes Folder/namespace in Blazor app

Minor fixup on caps for ORDERS required from Orders.  
  
I am now getting type mismatches in AccountProfile class (decimal to Guid).  
AccountPorfile was generated in Interaction 5, from VB.NET code for the class.

I realize, in VB.NET I manually created this class. I am going to regenerate AccountProfile by using DDL from the database, to ensure it is complete set of properties, with right datatypes

## ChatGTP Interaction 25 - - Direct Code Generation Data Tier Classes from DB DDL

### Input

For .NET Entity FrameworkCore version 8.0.2, can you generate me a model class named AccountProfile from this SQL Server table? CREATE TABLE [dbo].[ACCOUNTPROFILE]( [ADDRESS] [varchar](250) NULL, [PASSWORD] [varchar](50) NULL, [USERID] [varchar](50) NOT NULL, [ACCOUNTID] [bigint] NOT NULL, [EMAIL] [varchar](250) NULL, [CREDITCARD] [varchar](250) NULL, [FULLNAME] [varchar](250) NULL, [SALT] [varchar](20) NULL, CONSTRAINT [PK\_ACCOUNTPROFILE] PRIMARY KEY CLUSTERED ( [USERID] ASC, [ACCOUNTID] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY]

### Output

To create a model class named **AccountProfile** for Entity Framework Core 8.0.2 based on your SQL Server table definition, you can define a C# class that mirrors the structure of your **ACCOUNTPROFILE** table. Each column in the table will correspond to a property in the class. Since your primary key is clustered and consists of two columns (**USERID** and **ACCOUNTID**), you'll use the **[Key]**, **[Column(Order = n)]**, and **[DatabaseGenerated(DatabaseGeneratedOption.None)]** attributes to define the composite key.

Here's how you could define the **AccountProfile** class:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("ACCOUNTPROFILE", Schema = "dbo")]

public class AccountProfile

{

[Column("ADDRESS", TypeName = "varchar(250)")]

public string Address { get; set; }

[Column("PASSWORD", TypeName = "varchar(50)")]

public string Password { get; set; }

[Key]

[Column("USERID", TypeName = "varchar(50)", Order = 0)]

[Required]

public string UserId { get; set; }

[Key]

[Column("ACCOUNTID", TypeName = "bigint", Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)] // Assuming ACCOUNTID is provided and not auto-generated

public long AccountId { get; set; }

[Column("EMAIL", TypeName = "varchar(250)")]

public string Email { get; set; }

[Column("CREDITCARD", TypeName = "varchar(250)")]

public string CreditCard { get; set; }

[Column("FULLNAME", TypeName = "varchar(250)")]

public string FullName { get; set; }

[Column("SALT", TypeName = "varchar(20)")]

public string Salt { get; set; }

}

**Key Points:**

* **Data Annotations:** The **[Table]** attribute specifies the database table name, and the **[Column]** attribute specifies the column name and type. The **[Required]** attribute is used for non-nullable columns.
* **Composite Key:** The **[Key]** attribute along with **[Column(Order = n)]** are used together to define a composite primary key. The **Order** property specifies the position of the column in the composite key. The **[DatabaseGenerated(DatabaseGeneratedOption.None)]** attribute indicates that the value of **AccountId** is not auto-generated by the database.
* **String Properties:** The **TypeName** in the **[Column]** attribute is used to match the SQL Server data types with the C# data types. For string properties mapped to **varchar** columns, you might not need to specify the **TypeName** if the default mapping suits your needs, but it can be helpful for clarity and in cases where precision is important.

This class structure aligns with your database schema, allowing Entity Framework Core to map between the database and your .NET objects effectively.

### Notes

I am re-creating my AccountProfile class, in this more complete way.

After doing so, I had to fixup my Orders class, to use type long for ACCOUNTID. It was defined as long.  
  
Now on to next Stock.cs issue: My HOLDINGs table does not have a model class or entity. Will follow now-learned process to create.

## ChatGTP Interaction 26 – Output Code Optimization/Fixup

### Input

in Visual Studio 2022 in my new Blazor app cannot find references to HOLDING object, which I want to define as a model class for .NET Entity FrameworkCore 8.0.2 from my SQL Server var newHolding = new HOLDING { ACCOUNT\_ACCOUNTID = theAccount.ACCOUNTID, PURCHASEDATE = orderDate, PURCHASEPRICE = newOrder.PRICE, QUANTITY = quantity, QUOTE\_SYMBOL = newOrder.QUOTE\_SYMBOL, USERID = newOrder.USERID, HOLDINGID = newOrder.HOLDING\_HOLDINGID }; Fix?

### Output

If Visual Studio 2022 cannot find references to the **HOLDING** object in your Blazor app, it suggests that the **HOLDING** class has not been defined in your project, or there might be an issue with namespaces or project references. Here's how you can address and fix the issue:

**1. Define the HOLDING Class**

You need to ensure that a **HOLDING** class is defined in your project, corresponding to your **Holding** table in SQL Server. This class should mirror the structure of the table, with properties for each column. Based on the properties you've used in the **newHolding** object, here's an example of what the **HOLDING** class might look like:

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("Holding", Schema = "dbo")] // Adjust the table name and schema as necessary

public class HOLDING

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)] // Adjust if HOLDINGID is not auto-generated

public Guid HOLDINGID { get; set; }

[ForeignKey("Account")] // Adjust based on your actual foreign key relationship

public Guid ACCOUNT\_ACCOUNTID { get; set; }

public DateTime PURCHASEDATE { get; set; }

public decimal PURCHASEPRICE { get; set; }

public int QUANTITY { get; set; }

[MaxLength(10)] // Adjust based on your column definition

public string QUOTE\_SYMBOL { get; set; }

public string USERID { get; set; } // Adjust the type as necessary

// Navigation property to Account

public virtual Account Account { get; set; }

// Additional properties or navigation properties can go here

}

**2. Check Namespace**

If the **HOLDING** class is defined in a different namespace than where you're trying to use it, you'll need to import that namespace with a **using** directive at the top of your C# file:

using YourNamespaceWhereHoldingIsDefined;

**3. Add Entity Framework Core References**

Make sure your project has the necessary Entity Framework Core package references, such as **Microsoft.EntityFrameworkCore** and a provider package like **Microsoft.EntityFrameworkCore.SqlServer** for SQL Server.

**4. Include HOLDING in Your DbContext**

Ensure your **DbContext** includes a **DbSet<HOLDING>** property for the **HOLDING** entity:

public DbSet<HOLDING> Holdings { get; set; }

**5. Clean and Rebuild**

Sometimes, issues can be resolved by cleaning and rebuilding your project. Right-click on your solution in Visual Studio and select "Clean Solution", then "Rebuild Solution".

**6. Check for Typos**

Ensure there are no typos in your class name, file names, or references.

After checking these steps, if the issue persists, consider looking at the build output and Error List windows in Visual Studio for more specific error messages that could provide additional clues.

### Notes

I know now from experience, I need to generate complete model classes from the source DB table DDL (SQLServer). Not use this output which generates only those fields referenced in the code I supplied. So will start there.

## ChatGTP Interaction 27 - Direct Code Generation Data Tier Classes from DB DDL

### Input

I want to generate a new Holdings class, for Microsoft Entity FrameworkCore 8.0.2 for my Blazor .NET 8 Server app, from this table definition, please generate for me: CREATE TABLE [dbo].[HOLDING]( [PURCHASEPRICE] [decimal](12, 2) NULL, [HOLDINGID] [uniqueidentifier] NOT NULL, [QUANTITY] [float] NOT NULL, [PURCHASEDATE] [datetime] NULL, [ACCOUNT\_ACCOUNTID] [bigint] NOT NULL, [QUOTE\_SYMBOL] [varchar](10) NULL, [USERID] [varchar](50) NOT NULL, CONSTRAINT [PK\_HOLDING] PRIMARY KEY CLUSTERED ( [HOLDINGID] ASC, [ACCOUNT\_ACCOUNTID] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY]

### Output

To generate a new **Holding** class for Entity Framework Core 8.0.2 in your .NET 8 Blazor Server application based on the provided SQL Server table definition, you can follow this model structure. This class will represent your **HOLDING** table, with properties corresponding to each column in the table. Since **HOLDINGID** and **ACCOUNT\_ACCOUNTID** are part of your primary key, you'll need to use data annotations to configure them properly.

Here's the **Holding** class:

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("HOLDING", Schema = "dbo")]

public class Holding

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.None)] // Use None if HOLDINGID is provided by the application

[Column("HOLDINGID", TypeName = "uniqueidentifier")]

public Guid HoldingId { get; set; }

[Column("PURCHASEPRICE", TypeName = "decimal(12, 2)")]

public decimal? PurchasePrice { get; set; }

[Column("QUANTITY", TypeName = "float")]

[Required]

public double Quantity { get; set; } // float in SQL Server is mapped to double in C#

[Column("PURCHASEDATE", TypeName = "datetime")]

public DateTime? PurchaseDate { get; set; }

[Column("ACCOUNT\_ACCOUNTID", TypeName = "bigint")]

[Required]

public long AccountAccountId { get; set; } // Assuming this is a foreign key to another table

[Column("QUOTE\_SYMBOL", TypeName = "varchar(10)")]

public string QuoteSymbol { get; set; }

[Column("USERID", TypeName = "varchar(50)")]

[Required]

public string UserId { get; set; }

// Navigation properties and any additional columns go here

}

**Key Points:**

* **Primary Key**: The **HOLDINGID** is marked with the **[Key]** attribute to denote it as the primary key field. Since your primary key is clustered and consists of two columns, consider using Fluent API to configure the composite key if needed, as data annotations do not support composite primary keys directly.
* **DatabaseGenerated**: The **[DatabaseGenerated(DatabaseGeneratedOption.None)]** attribute indicates that the **HOLDINGID** value is provided by the application or database and is not auto-generated. If **HOLDINGID** is auto-generated by the database, use **DatabaseGeneratedOption.Identity**.
* **Column Types**: The **[Column]** attribute specifies the column name and type in the database. Adjust these as necessary to match your database schema.
* **Nullable Properties**: Properties like **PurchasePrice** and **PurchaseDate** are nullable (**decimal?**, **DateTime?**) to match the **NULL** setting in the SQL definition.
* **Required Attributes**: The **[Required]** attribute is used for non-nullable columns in your database table.
* **Navigation Properties**: If there are relationships with other tables, you should include navigation properties in your class to represent these relationships. The example does not include navigation properties because the relationships are not defined in the table structure you provided.

This class should align well with your database schema and allow Entity Framework Core to correctly map between the database and your .NET objects.

### Notes

Ok. I created the new Holdings class from generated code above,

Then back to the stock.cs class, I had to re-map the previously generated code to match the property definitions, vs before generated code used “All Caps” as property names.

This worked, however on this line I now see I have created model classes for Orders and Holdings, but I have not dbcontext/Entities defined. Need to fix for both, to match what I did for QUOTEDBEntities and ACCOUNTDBEntities.

## ChatGTP Interaction 28 – Output Code Optimization/Fixup

### Input

Visual Studio 2022 in my new Blazor app cannot find references to ORDERS in this line  
  
“\_accountContext.ORDERS.Add(newOrder);”

### Output

If Visual Studio 2022 cannot find references to **ORDERS** in the line **\_accountContext.ORDERS.Add(newOrder);**, it suggests a few possible issues that you can check and correct:

**1. DbSet Naming**

Ensure that your **DbContext** class (**\_accountContext** in your case) has a **DbSet** property named **ORDERS** that corresponds to your **ORDER** entity. The naming must match exactly. For example:

public class ACCOUNTDBEntities : DbContext

{

public DbSet<ORDER> ORDERS { get; set; }

// Other DbSet properties...

}

If Visual Studio 2022 cannot find references to **ORDERS** in the line **\_accountContext.ORDERS.Add(newOrder);**, it suggests a few possible issues that you can check and correct:

**1. DbSet Naming**

Ensure that your **DbContext** class (**\_accountContext** in your case) has a **DbSet** property named **ORDERS** that corresponds to your **ORDER** entity. The naming must match exactly. For example:

csharpCopy code

public class ACCOUNTDBEntities : DbContext { public DbSet<ORDER> ORDERS { get; set; } // Other DbSet properties... }

If the **DbSet** for your **ORDER** entity is named differently in your **DbContext**, you should use that name instead of **ORDERS**.

**2. Class and Property Naming Convention**

In C#, the naming convention typically follows PascalCase for classes and properties. Ensure your **ORDER** class is correctly defined and follows C# naming conventions, and consider renaming it to **Order** if it does not:

public class Order

{

// Order properties...

}

And your **DbContext** would have:

public DbSet<Order> Orders { get; set; }

If Visual Studio 2022 cannot find references to **ORDERS** in the line **\_accountContext.ORDERS.Add(newOrder);**, it suggests a few possible issues that you can check and correct:

**1. DbSet Naming**

Ensure that your **DbContext** class (**\_accountContext** in your case) has a **DbSet** property named **ORDERS** that corresponds to your **ORDER** entity. The naming must match exactly. For example:

csharpCopy code

public class ACCOUNTDBEntities : DbContext { public DbSet<ORDER> ORDERS { get; set; } // Other DbSet properties... }

If the **DbSet** for your **ORDER** entity is named differently in your **DbContext**, you should use that name instead of **ORDERS**.

**2. Class and Property Naming Convention**

In C#, the naming convention typically follows PascalCase for classes and properties. Ensure your **ORDER** class is correctly defined and follows C# naming conventions, and consider renaming it to **Order** if it does not:

csharpCopy code

public class Order { // Order properties... }

And your **DbContext** would have:

csharpCopy code

public DbSet<Order> Orders { get; set; }

**3. Using Statement**

Check if the namespace where your **DbContext** (e.g., **ACCOUNTDBEntities**) and **ORDER** (or **Order**) entity classes are defined is correctly imported with a **using** statement in the file where you're trying to add the order to the context.

using YourNamespaceWhereDbContextAndOrderAreDefined;

**4. Build and Namespace Issues**

* Try **rebuilding** your solution to make sure all references are correctly updated.
* If you have recently moved classes between projects or changed namespaces, make sure those changes are reflected everywhere the classes are used.

**5. Check for Typos**

Ensure there are no typos in the entity class name or the **DbSet** property name in your **DbContext**.

If you've verified all the above and the issue persists, it might be helpful to look at the specific error message provided by Visual Studio for additional clues. Sometimes, closing and reopening Visual Studio or clearing the bin and obj folders manually and then rebuilding the solution can also resolve strange build issues.

### Notes

OK. I have come to a realization, that I have neen led down two alternate paths (and hence have a mismatch in migrated code so far)

1. Path 1: In VB.NET app, I defined an ACCOUNTDBEntities using a DATABASE definition that contains all table: ACCOUNT, ACCOUNTPROFILE, HOLDING, ORDERS. One DB Context to get to all.
2. Path 2: I did use this for my first DBEntity, at Database level, but then was led by ChatGTP to start creating classes on a PER TABLE basis.

I am exploring code, and do not know yet whether better to define DB Sets under an EntityFramework context at **DATABASE** level, or create separate EntityFramework Entities at the **TABLE** level.  
  
I will ask ChatGTP before proceeding.

## ChatGTP Interaction 29

### Input

OK. I have come to a realization, that I have neen led down two alternate paths (and hence have a mismatch in migrated code so far) 1. Path 1: In VB.NET app, I defined an ACCOUNTDBEntities using a DATABASE definition that contains all table: ACCOUNT, ACCOUNTPROFILE, HOLDING, ORDERS. One DB Context to get to all. 2. Path 2: I did use this for my first DBEntity, at Database level, but then was led by ChatGTP to start creating classes on a PER TABLE basis. I am exploring code, and do not know yet whether better to define DB Sets under an EntityFramework context at DATABASE level, or create separate EntityFramework Entities at the TABLE level. Which do you advise?

### Output

In Entity Framework Core (EF Core), the recommended approach is to use a single **DbContext** to represent the database context, within which you define **DbSet<T>** properties for each table in the database you wish to interact with. This approach provides a centralized configuration for your database schema, relationships, and operations like queries, updates, and migrations.

Here are the key reasons for using a single **DbContext** with multiple **DbSet<T>** properties, each representing a table:

**1. Simplicity and Maintainability:**

* A single **DbContext** provides a clear overview of the database structure and its tables within your application. It simplifies the management of database operations and migrations.

**2. Data Relationships:**

* Defining all your tables within one **DbContext** makes it easier to configure and manage relationships between tables (e.g., one-to-many, many-to-many). EF Core uses navigation properties to load related data, which are easier to set up and use within a single context.

**3. Performance:**

* Using one **DbContext** helps optimize connection management and caching. EF Core is designed to work efficiently with a single context managing multiple tables.

**4. Transactions:**

* A single **DbContext** ensures that operations involving multiple tables can be easily wrapped in transactions to maintain data integrity.

**5. Migrations:**

* With one **DbContext**, managing migrations becomes straightforward. EF Core migrations track changes across all tables in the context and generate migration scripts accordingly.

**Example Structure:**

public class ACCOUNTDBEntities : DbContext

{

public DbSet<Account> Accounts { get; set; }

public DbSet<AccountProfile> AccountProfiles { get; set; }

public DbSet<Holding> Holdings { get; set; }

public DbSet<Order> Orders { get; set; }

// Configuration and model building goes here

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Fluent API configurations

}

}

In this structure, **Account**, **AccountProfile**, **Holding**, and **Order** are model classes representing individual tables, and **ACCOUNTDBEntities** is the database context managing them all.

**Conclusion:**

For most applications, especially when starting new projects or migrating existing ones to EF Core, the single **DbContext** approach is recommended for its simplicity, maintainability, and the benefits listed above. However, in very large and complex systems where different parts of the database are completely isolated or belong to different bounded contexts (in terms of Domain-Driven Design), it might make sense to use separate **DbContexts**. But even in complex scenarios, it's often still preferable to start with a single context and only split it if there's a clear and justified need.

### Notes

OK! Great advice. My app uses two databases, QUOTEDB and ACCOUNTDB.

I have model classes defined already, from previous interactions with ChatGTP, for the accountprofile, orders and holdings tables. I may just need to paste these into the master class ACCOUNTDBEntities as individual datasets. Not sure, will find out by exploring more/thinking.

## ChatGTP Interaction 30 - Output Code Optimization/Fixup

### Input

For .NET Entity Framework 8.0.2, I have a problem in my code in that I have been defining separate model classes for each table, instead of defining within DbSets. How can I move this model class, for example to be a DbSet Quotes within this code: "public class QUOTEDBEntities : DbContext { public QUOTEDBEntities(DbContextOptions<QUOTEDBEntities> options) : base(options) { } // DbSets for your entities public DbSet<Quote> Quotes { get; set; } // Other DbSets... }" From this model class (Quotes): "//This entire class generated by ChatGTP in interaction 23 using System.ComponentModel.DataAnnotations; using System.ComponentModel.DataAnnotations.Schema; [Table("QUOTE", Schema = "dbo")] public class Quote { [Column("LOW", TypeName = "decimal(12, 2)")] public decimal? Low { get; set; } [Column("OPEN1", TypeName = "decimal(12, 2)")] public decimal? Open1 { get; set; } [Column("VOLUME", TypeName = "float")] [Required] public float Volume { get; set; } [Column("PRICE", TypeName = "decimal(12, 2)")] public decimal? Price { get; set; } [Column("HIGH", TypeName = "decimal(12, 2)")] public decimal? High { get; set; } [Column("COMPANYNAME", TypeName = "varchar(250)")] public string CompanyName { get; set; } [Key] [Column("SYMBOL", TypeName = "varchar(10)", Order = 0)] [Required] public string Symbol { get; set; } [Column("CHANGE1", TypeName = "float")] [Required] public float Change1 { get; set; } [Key] [Column("QUOTEID", TypeName = "bigint", Order = 1)] [DatabaseGenerated(DatabaseGeneratedOption.None)] // Assuming QUOTEID is not auto-generated public long QuoteId { get; set; } }"

### Output

To integrate your **Quote** model class into the **QUOTEDBEntities** DbContext as a **DbSet**, you've actually already done most of the work needed. Your **Quote** class is defined with all the necessary annotations to map it to the **QUOTE** table in your database, and you've declared a **DbSet<Quote>** property within your **QUOTEDBEntities** DbContext. This setup is exactly what's required to work with the **Quote** entities using Entity Framework Core.

Here's a brief overview of how these parts work together:

**Quote Model Class:**

Your **Quote** class is an Entity Framework model that maps to the **QUOTE** table in your database. Each property in the **Quote** class is mapped to a column in the **QUOTE** table, with data annotations like **[Column]** and **[Key]** specifying the column names, types, and key configuration. This class doesn't need to be changed or moved; it's correctly set up to represent rows in your **QUOTE** table.

**QUOTEDBEntities DbContext Class:**

The **QUOTEDBEntities** class is your Entity Framework DbContext. It serves as the primary class that coordinates Entity Framework functionality for a given data model. You've correctly added a **DbSet<Quote>** property to this class:

public DbSet<Quote> Quotes { get; set; }

This **DbSet<Quote>** property represents a collection of all **Quote** entities in the context and is what you'll use to query and save instances of **Quote**.

**Using the DbSet:**

You can perform CRUD operations on the **Quote** table using the **Quotes** **DbSet**. Here are some examples:

* **Querying:**

var quotes = await \_quotedbEntities.Quotes.Where(q => q.Symbol == "AAPL").ToListAsync();

**Adding a new quote:**

var newQuote = new Quote { /\* set properties \*/ };

\_quotedbEntities.Quotes.Add(newQuote);

await \_quotedbEntities.SaveChangesAsync();

**Updating an existing quote:**

var quote = await \_quotedbEntities.Quotes.FindAsync(someQuoteId);

quote.Price = newPrice;

await \_quotedbEntities.SaveChangesAsync();

**Removing a quote:**

var quote = await \_quotedbEntities.Quotes.FindAsync(someQuoteId);

\_quotedbEntities.Quotes.Remove(quote);

await \_quotedbEntities.SaveChangesAsync();

**Summary:**

Your current setup with separate model classes for each table and a single DbContext (**QUOTEDBEntities**) that includes **DbSet<T>** properties for each model is the correct approach with Entity Framework Core. You don't need to move your model classes into the DbContext; they should remain separate, with the DbContext referencing them through **DbSet<T>** properties as you have done with **Quotes**. This structure is clear, maintainable, and adheres to best practices.

### Notes

**I LOVE IT!** What a great response, that gives me good confidence my previous work is not wasted:  
  
“To integrate your **Quote** model class into the **QUOTEDBEntities** DbContext as a **DbSet**, you've actually already done most of the work needed. Your **Quote** class is defined with all the necessary annotations to map it to the **QUOTE** table in your database, and you've declared a **DbSet<Quote>** property within your **QUOTEDBEntities** DbContext. This setup is exactly what's required to work with the **Quote** entities using Entity Framework Core.”

So, now on to following these instructions, seems straightforward and I grock where this is headed.

One more ChatGTP clarification I want:

## ChatGTP Interaction 31 – Output Code Optimization/Fixup

### Input

Assuming in .NET 8 EntityFrameworkCore 8.0.2, I have the following DB Entity setup: //Al of this code was auto-genrated for this class, below, from ChatGTP interaction 12 using Microsoft.EntityFrameworkCore; "public class ACCOUNTDBEntities : DbContext { public ACCOUNTDBEntities(DbContextOptions<ACCOUNTDBEntities> options) : base(options) { } public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; } }" And I have also separately defined a model class, AccountProfile: "//The following is cute and paste, directly from ChatGTP interaction 25 using System.ComponentModel.DataAnnotations; using System.ComponentModel.DataAnnotations.Schema; [Table("ACCOUNTPROFILE", Schema = "dbo")] public class AccountProfile { [Column("ADDRESS", TypeName = "varchar(250)")] public string Address { get; set; } [Column("PASSWORD", TypeName = "varchar(50)")] public string Password { get; set; } [Key] [Column("USERID", TypeName = "varchar(50)", Order = 0)] [Required] public string UserId { get; set; } [Key] [Column("ACCOUNTID", TypeName = "bigint", Order = 1)] [DatabaseGenerated(DatabaseGeneratedOption.None)] // Assuming ACCOUNTID is provided and not auto-generated public long AccountId { get; set; } [Column("EMAIL", TypeName = "varchar(250)")] public string Email { get; set; } [Column("CREDITCARD", TypeName = "varchar(250)")] public string CreditCard { get; set; } [Column("FULLNAME", TypeName = "varchar(250)")] public string FullName { get; set; } [Column("SALT", TypeName = "varchar(20)")] public string Salt { get; set; } }" Do I need to map the AccountProfile model class directly into my : "public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; }" And if so, how to do?

### Output

Based on your setup in a .NET 8 Blazor Server application using Entity Framework Core 8.0.2, you have correctly defined both the **DbContext** (**ACCOUNTDBEntities**) and the entity model class (**AccountProfile**). Your **AccountProfile** model class is well-annotated to map to the **ACCOUNTPROFILE** table in your database, and you have declared a **DbSet<AccountProfile>** within your **DbContext**. This setup is essentially what you need to work with **AccountProfile** entities using Entity Framework Core.

**Do you need to map AccountProfile to DbSet<AccountProfile>?**

The short answer is no, not beyond what you've already done. By declaring **public DbSet<AccountProfile> ACCOUNTPROFILEs { get; set; }** within your **DbContext**, you have already mapped the **AccountProfile** model to the corresponding table in the database (**ACCOUNTPROFILE**), as indicated by the **[Table("ACCOUNTPROFILE", Schema = "dbo")]** attribute on the **AccountProfile** class.

**How does it work?**

* **DbContext (ACCOUNTDBEntities)**: Serves as the primary class that coordinates Entity Framework functionality for a given data model. It includes **DbSet** properties that represent collections of the specified entities.
* **DbSet<AccountProfile> Property (ACCOUNTPROFILEs)**: Represents the collection of all **AccountProfile** entities in the database context, allowing you to query and save instances of **AccountProfile**.
* **Entity Class (AccountProfile)**: Represents the schema of a table (**ACCOUNTPROFILE**) in the database. Attributes like **[Table]**, **[Column]**, and **[Key]** are used to configure the mapping between the class properties and the table columns.

**Using DbSet<AccountProfile>**

You can perform CRUD operations on the **ACCOUNTPROFILE** table using the **ACCOUNTPROFILEs** **DbSet**. For example:

[redacted here, as repeat from Interaction 30]  
  
**Summary**

Your current setup is correct for Entity Framework Core usage. The **DbSet<AccountProfile>** property in your **DbContext** (**ACCOUNTDBEntities**) directly corresponds to your **AccountProfile** model class, and no further mapping is required. You can use this **DbSet** to perform database operations related to **AccountProfile** entities.

### Notes

So at least I have AccountProfile setup correctly, as a table/DbSet for ACCOUNTDBEntities. I think all I need to do is add Datasets for Orders, Holdings, and Account Classes within this DB/Entity (ACCOUNTDBEntities)

I see I have one Table model class missing, ACCOUNT table. So I will generate the model class via ChatGTP:

## ChatGTP Interaction 32 - Direct Code Generation Data Tier Classes from DB DDL

### Input

Can you create an EntityFramework model class from the following SQLServer DDL? "CREATE TABLE [dbo].[ACCOUNT]( [CREATIONDATE] [datetime] NULL, [OPENBALANCE] [decimal](12, 2) NULL, [LOGOUTCOUNT] [int] NOT NULL, [BALANCE] [decimal](12, 2) NULL, [ACCOUNTID] [bigint] NOT NULL, [LASTLOGIN] [datetime] NULL, [LOGINCOUNT] [int] NOT NULL, [PROFILE\_USERID] [varchar](50) NOT NULL, CONSTRAINT [PK\_ACCOUNT] PRIMARY KEY CLUSTERED ( [PROFILE\_USERID] ASC, [ACCOUNTID] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY]"

### Output

To create an Entity Framework model class for the **ACCOUNT** table defined by your SQL Server DDL in a .NET 8 Entity Framework Core 8.0.2 context, you would define a C# class with properties corresponding to the columns in the table. Given that **PROFILE\_USERID** and **ACCOUNTID** form a composite primary key, you'll need to use data annotations or the Fluent API to configure this in Entity Framework Core.

Here's an example of how you could define the **Account** model class using data annotations:

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Table("ACCOUNT", Schema = "dbo")]

public class Account

{

[Column("CREATIONDATE", TypeName = "datetime")]

public DateTime? CreationDate { get; set; }

[Column("OPENBALANCE", TypeName = "decimal(12, 2)")]

public decimal? OpenBalance { get; set; }

[Column("LOGOUTCOUNT", TypeName = "int")]

[Required]

public int LogoutCount { get; set; }

[Column("BALANCE", TypeName = "decimal(12, 2)")]

public decimal? Balance { get; set; }

[Key]

[Column("ACCOUNTID", TypeName = "bigint", Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)] // Assuming ACCOUNTID is provided and not auto-generated

public long AccountId { get; set; }

[Column("LASTLOGIN", TypeName = "datetime")]

public DateTime? LastLogin { get; set; }

[Column("LOGINCOUNT", TypeName = "int")]

[Required]

public int LoginCount { get; set; }

[Key]

[Column("PROFILE\_USERID", TypeName = "varchar(50)", Order = 0)]

[Required]

public string ProfileUserId { get; set; }

// Navigation properties and any additional configuration can go here

}

**Key Points:**

* **Nullable Properties**: The **CreationDate**, **OpenBalance**, **Balance**, and **LastLogin** properties are nullable (**DateTime?**, **decimal?**) to match the **NULL** setting in the SQL definition.
* **Data Annotations**: The **[Table]** attribute specifies the database table name, and the **[Column]** attribute specifies the column name and type. The **[Key]** attribute denotes the properties that make up the composite primary key, and **[Required]** is used for non-nullable columns.
* **Composite Primary Key**: The **ProfileUserId** and **AccountId** properties are marked with the **[Key]** attribute and an **[Column(Order = n)]** to denote their order in the composite key. EF Core requires the use of Fluent API for composite keys, but this annotation approach can be used for simpler scenarios or in combination with Fluent API for more complex configurations.
* **DatabaseGeneratedOption**: The **[DatabaseGenerated(DatabaseGeneratedOption.None)]** attribute is used for **AccountId** to indicate that its value is not auto-generated by the database. Adjust this as necessary based on your actual database schema.

This class structure aligns with your **ACCOUNT** table schema and allows Entity Framework Core to correctly map between the database and your .NET objects. Remember, for more complex scenarios, especially those involving composite keys, Fluent API provides more flexibility and is recommended by EF Core documentation.

### Notes

Ok—now creating a new class “Account” in my Data Classes folder/namespace, for this table via cut paste from above.

OK: Spending time now on basic fixup of originally generated Entity framework class names/DBset property names, for example, from all CAPs to proper cases (ACCOUNTID is now AccountId, for example).  
  
This problem, I think, stems from my starting with VB.NET Business class code, and not starting by using ChatGTP to first build my Data Model Classes. This in turn stems from fact my VB.NET app was not factored into a Data <Model layer, properly used. NOTE:  
  
START BY DEFINING DATA MODEL CLASSES. This, I found in my case, was to start with the DDL for each table (generated in SQL Server Management Studio), after defining an ENTITY for each of the two databases.

TEHN DO BUSINESS CLASS TRANSLATIONS from code

THEN, presumably, DO UI (WinForms to Blazor Pages) (I am not at this step yet).

OK: in this line of code: Stock.cs (business layer class); I am down to this final error in VS 2022:  
  
PRICE = theStock.Price “cannot implicity convert type decimal? To type decimal. An explicit conversion exists, are you missing a cast?”

## ChatGTP Interaction 33 – Output Code Optimization/Fixup

### Input

How to fix in .NET 8 C#?

PRICE = theStock.Price “cannot implicity convert type decimal? To type decimal. An explicit conversion exists, are you missing a cast?”

### Output

The error message you're encountering indicates that you're trying to assign a nullable decimal (**decimal?**) to a non-nullable decimal (**decimal**) variable. In C#, nullable types can represent all the values of an underlying type, and an additional null value. When you try to assign a nullable type to a non-nullable type, you must handle the possibility of a null value explicitly.

To fix this error, you have a few options depending on your specific requirements and what you want to do when **theStock.Price** is null:

**Option 1: Use the Value Property (if you're sure it's not null)**

If you are certain that **theStock.Price** is not null at this point in your code, you can use the **Value** property of the nullable type:

PRICE = theStock.Price.Value;

**Option 2: Provide a Default Value with the Null-Coalescing Operator**

If **theStock.Price** could be null and you want to provide a default value in that case (e.g., **0**), you can use the null-coalescing operator (**??**):

PRICE = theStock.Price ?? 0;

**Option 3: Use Conditional Logic**

If you need to perform some specific logic when **theStock.Price** is null (e.g., logging, throwing an exception, etc.), you can use an **if** statement or the ternary conditional operator:

if (theStock.Price.HasValue)

{

PRICE = theStock.Price.Value;

}

else

{

// Handle the null case, e.g., set a default value, log a message, throw an exception, etc.

PRICE = 0; // Default value or handle differently

}

Or using the ternary operator:

PRICE = theStock.Price.HasValue ? theStock.Price.Value : 0; // Replace 0 with your desired default value

**Option 4: Change the Target Variable to Nullable**

If it's acceptable for the target variable (**PRICE**) to also be nullable, and it's defined in a context you can change, you might consider changing its type to **decimal?**:

decimal? PRICE = theStock.Price;

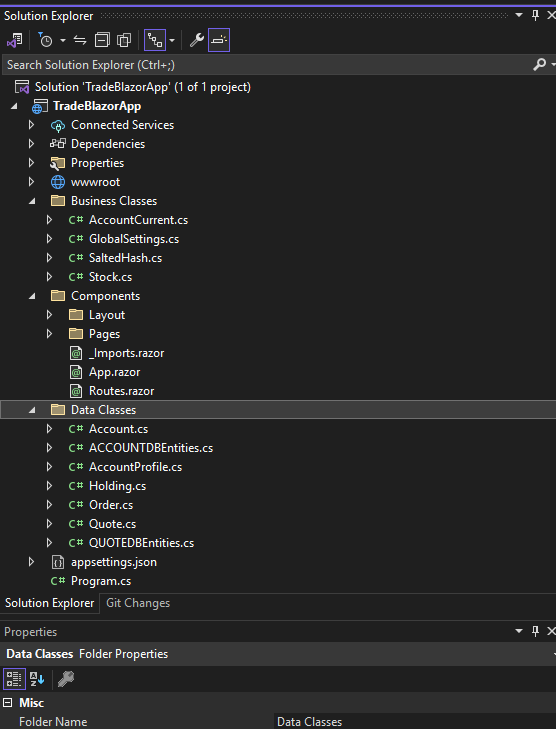
This way, **PRICE** can hold null values, and no conversion is necessary.

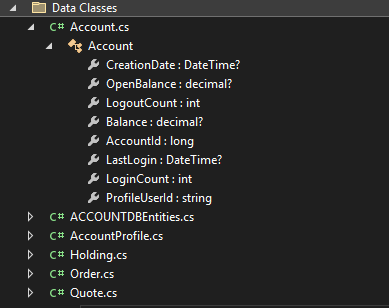
Choose the option that best fits your scenario. Each approach has different implications for how null values are handled in your application's logic.

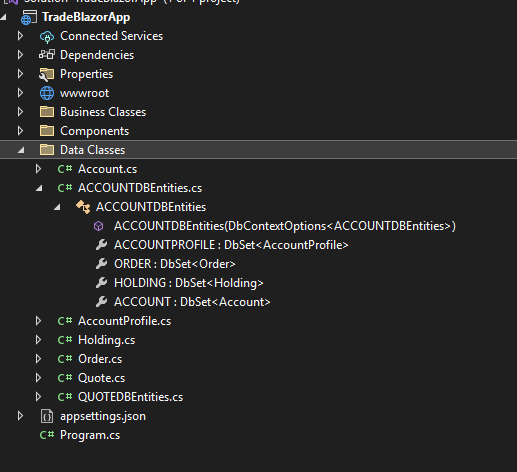
### Notes

Cool! Going with option 1.

Project BUILDS!  
  
My VS 2022 Blazor App Solution now Looks Like This:

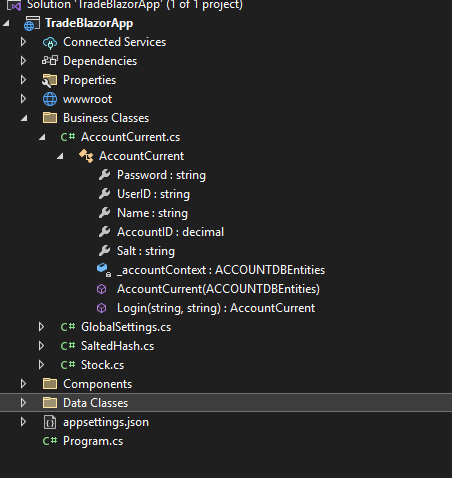


Exploding Out Account Data Model Class:  
  


Exploding Out ACCOUNTDBEntities:  
  


Exploding Out Stock.cs:  
  
A screenshot of a computer program

Description automatically generated

Exploding Out AccountCurrent:  
  


# Phase 5 – Migrating the UI/VB.NET Win Forms to Blazor UI

## ChatGTP Interaction 34

### Input

For the following quetions/session, please assume I am migrating a VB.NET WinForms application to a new .NET 8.0 Blazor Application, targeting .NET EntityFrameworkCore 8.0.2

### Output

Understood, you're migrating a VB.NET WinForms application to a new .NET 8.0 Blazor Application, and you plan to use EntityFrameworkCore 8.0.2 for data access. Please go ahead with your questions or any specific aspects of the migration process you need assistance with.

### Notes

I will now start with the main navigation form, the first for the UI Tier

## ChatGTP Interaction 35 - Direct Code Migration UI Tier

### Input

I have the following MainNav form in VB.NET WinForms, which is the main home form for the app. Can you migrate the following VB.NET WinForms code to Blazor?

Public Class MainNav

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

GlobalSettings.LoggedIn = False

End Sub

Private Sub Label3\_Click(sender As Object, e As EventArgs) Handles Label3.Click

Form1.Show()

End Sub

Private Sub Label2\_Click(sender As Object, e As EventArgs) Handles Label2.Click

If GlobalSettings.LoggedIn Then

MsgBox("You are now Logged Out!")

GlobalSettings.LoggedIn = False

Login.PasswordTextBox.Text = ""

Login.UserIDTextBox.Text = ""

Else

Login.Show()

End If

End Sub

Private Sub Label4\_Click(sender As Object, e As EventArgs) Handles Label4.Click

If GlobalSettings.LoggedIn Then

Dim holdingsForm As New Holdings()

holdingsForm.Show()

Else

MsgBox("You Must Login First!")

End If

End Sub

Private Sub Label5\_Click(sender As Object, e As EventArgs) Handles Label5.Click

If GlobalSettings.LoggedIn Then

Dim accountForm As New AccountForm()

accountForm.Show()

Else

MsgBox("You Must Login First!")

End If

End Sub

End Class

### Output

Migrating from a VB.NET WinForms application to a Blazor application involves significant changes, as Blazor is a component-based framework that uses Razor components (**.razor** files) for its UI, and it typically runs either on the server (Blazor Server) or in the client's web browser (Blazor WebAssembly). Since your original code uses event handlers and direct manipulation of form and control state, we'll need to adapt these concepts to Blazor's component model and data binding.

Below is an example of how you might begin to migrate your **MainNav** form to a Blazor component. This example assumes you are using Blazor Server, but the approach for Blazor WebAssembly would be similar.

@page "/mainnav"

@if (!GlobalSettings.LoggedIn)

{

<p>Please log in to continue.</p>

}

else

{

<div>

<button @onclick="ShowForm1">Form 1</button>

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

}

@code {

// Assuming GlobalSettings is a static class you have elsewhere in your project

// and Login, Holdings, and AccountForm are components you've migrated or will migrate.

private void ShowForm1()

{

// Navigate to Form1 component/page

NavigationManager.NavigateTo("/form1");

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Reset login information

// Assuming you have a Login component with parameters or a service to handle login state

// You might need to call a method on a service here or set some state to represent this

}

else

{

// Navigate to Login component/page

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

// Navigate to Holdings component/page

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

// Navigate to AccountForm component/page

NavigationManager.NavigateTo("/accountform");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

}

In this Blazor component:

* The **@page** directive at the top specifies the route for this component.
* The **@if** block conditionally renders content based on whether the user is logged in, similar to your VB.NET form's logic.
* Buttons are used in place of labels for actions, with **@onclick** event handlers bound to methods defined in the **@code** block.
* The **NavigationManager.NavigateTo** method is used to navigate to other components/pages. You'll need to ensure you have corresponding Blazor components for **Form1**, **Login**, **Holdings**, and **AccountForm** that these navigation calls reference.
* This example assumes **GlobalSettings** is accessible and its **LoggedIn** property is a static property that can be checked. Depending on your actual app structure, you might need a different approach to manage and access global settings and authentication state, such as using a Blazor service.

Remember, this is a starting point. Blazor offers a lot more in terms of component interaction, routing, and state management that you'll likely want to take advantage of as you migrate your application.

### Notes

The code generated uses NavigationManager as follows (ex):  
  
NavigationManager.NavigateTo("/login");

However, VS state that I “must use an object reference for a non-static class. So need to investigate.

## ChatGTP Interaction 36 – Output Code Optimization/Fixup

### Input

In the code you just generated, Visual Studio tells me “an object reference is required for the non-static field NavigationManager.NavigateTo method calls. Remember I am targeting a .NET 8.0 Blazor Server App. Can you fix the previously generated code to correct this issue?

### Output

In Blazor, the **NavigationManager** is available as a service that should be injected into your component to manage navigation. The error you're encountering suggests that **NavigationManager** is being called like a static class, whereas it actually needs to be accessed through an instance provided by the Blazor dependency injection system.

Here's the corrected version of the code with **NavigationManager** properly injected:

@page "/mainnav"

@inject NavigationManager NavigationManager

@if (!GlobalSettings.LoggedIn)

{

<p>Please log in to continue.</p>

}

else

{

<div>

<button @onclick="ShowForm1">Form 1</button>

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

}

@code {

// Your code logic remains the same here

private void ShowForm1()

{

NavigationManager.NavigateTo("/form1");

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Add logic to reset login state as needed

}

else

{

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/accountform");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

}

In this version, the **@inject** directive is used at the top of the component to inject the **NavigationManager** service. This service is then used within the component methods to navigate to different pages. This approach should resolve the issue you're encountering in Visual Studio.

### Notes

Aha! An example of using ChatGTP to correct previously generated code without throwing the whole code block at it again.

It works this time, no errors!

But when running the page (the “/” root page), I get a runtime error

## ChatGTP Interaction 37 – Output Code Optimization/Fixup

### Input

When running this page you generated for Blazor Server app I get the following error on load, how to fix: An unhandled exception occurred while processing the request. InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }. Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.DynamicPageEndpointMatcherPolicy.ApplyAsync(HttpContext httpContext, CandidateSet candidates)

### Output

For a .NET 8 Blazor Server application, encountering the error "Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }" typically indicates an issue with the setup of the **\_Host.cshtml** file or the routing configuration in your **Program.cs** file. Here's a focused approach to address this issue in a .NET 8 environment:

**1. \_Host.cshtml File**

Ensure that the **\_Host.cshtml** file is present in the **Pages** directory of your project. This file serves as the entry point for the Blazor Server application and should include the necessary directives and markup to host Blazor components. The file should look something like this:

@page "/"

@namespace YourAppName.Pages

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>

<html lang="en">

<head>

...

</head>

<body>

<app>

<component type="typeof(App)" render-mode="Server" />

</app>

<script src="\_framework/blazor.server.js"></script>

</body>

</html>

**2. Program.cs Configuration**

In .NET 8, the configuration previously done in **Startup.cs** is typically moved to **Program.cs**. Verify that your **Program.cs** includes the correct setup for Blazor Server. You should have something like the following:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddServerSideBlazor();

// Add any other necessary services

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

Ensure that the **MapFallbackToPage("/\_Host")** line is included, which specifies the fallback route to the **\_Host.cshtml** file, allowing Blazor to handle routing appropriately.

**3. Check for Typos or Misconfigurations**

Typos or small errors in either the **\_Host.cshtml** file or the **Program.cs** setup can lead to routing issues. Double-check these files for any discrepancies or omissions.

By ensuring that the **\_Host.cshtml** file is correctly placed and contains the necessary markup, and that your **Program.cs** is accurately configured for Blazor Server, you should be able to resolve the routing error you're encountering.

### Notes

I suspect again, that ChatGTP is targeting answers for an older version of Blazor (not .NET 8/VS 2022). I do not think I need a \_Hosts.cshtml file, think this was retired. Lets ask.

## ChatGTP Interaction 38 – Output Code Optimization/Fixup

### Input

For the latest .NET 8 Blazor Server app in Visual Studio 2022, are you sure a \_Host.cshtml is necessary? Visual Studio standard template does not have one in my project, but does have a Routes.razor, App.Razor and \_Imports.Razor files?

### Output

In the latest .NET 8 Blazor Server applications, the traditional **\_Host.cshtml** page is indeed not a requirement, especially with the newer project templates provided by Visual Studio 2022. These templates have evolved to use a more simplified structure for Blazor Server apps, which might not include the **\_Host.cshtml** file, especially in projects that have been created with the latest templates or .NET versions.

Instead of **\_Host.cshtml**, the routing and app component composition are handled within Razor component files such as **App.razor** and **Routes.razor**. Here's a brief overview of these files:

* **App.razor**: This is the root component of a Blazor application. It typically contains the **<Router>** component that handles routing within the app.
* **Routes.razor** (if present): This file is used for configuring routes in a more centralized manner, especially in larger applications. It's not always included by default in all project templates.
* **\_Imports.razor**: This file is used to import namespaces and Razor components that are available globally across the application.

Given this structure, your Blazor Server application does not rely on **\_Host.cshtml** for routing or component hosting. Instead, it leverages the **App.razor** and potentially **Routes.razor** for these purposes. Here's an example of what the **App.razor** file might look like:

<Router AppAssembly="@typeof(Program).Assembly">

<Found Context="routeData">

<RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

</Found>

<NotFound>

<LayoutView Layout="@typeof(MainLayout)">

<p>Sorry, there's nothing at this address.</p>

</LayoutView>

</NotFound>

</Router>

In this setup, navigation and routing are handled entirely within the Razor components, and there's no need for a separate **\_Host.cshtml** file. This approach simplifies the structure of Blazor Server applications and aligns more closely with typical component-based architectures. If you're following the standard Visual Studio 2022 template for a .NET 8 Blazor Server app, you should configure your routes and components within these **.razor** files.

### Notes

Not all that helpful, after looking at project files. Perhaps I just need to add blank pages for the other forms NavigationManager is pointing to?

## ChatGTP Interaction 39 – Output Code Optimization/Fixup

### Input

considering above, When running my newly created root page, defined here:

"@page "/"

@using TradeBlazorApp.Business\_Classes

@inject NavigationManager NavigationManager

<PageTitle>Trade Home</PageTitle>

@if (!GlobalSettings.LoggedIn)

{

<p>Please log in to continue.</p>

}

else

{

<div>

<button @onclick="ShowQuotes">Quotes</button> <!--I Changed the name here from Form1 to Quotes-->

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

}

@code {

// Your code logic remains the same here

private void ShowQuotes() //I changed from ShowForm1 to ShowQuotes.

{

NavigationManager.NavigateTo("/quotes");

//I Changed the name here from form1 to quotes

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Add logic to reset login state as needed

}

else

{

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/accountform");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

}

"

I get runtime error "InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }."

Can you build me a proper App.Razor file to fix this?

### Output

The error you're encountering, "InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }," suggests there's a routing issue in your Blazor Server application. Since you've confirmed that your project structure follows the newer format without a **\_Host.cshtml** file and includes **App.razor**, we can focus on ensuring the **App.razor** file is set up correctly to handle routing for your application.

The **App.razor** file is the root component for a Blazor app and is responsible for setting up routing. Here's an example of how your **App.razor** file might look to ensure proper routing within your application:

<Router AppAssembly="@typeof(Program).Assembly">

<Found Context="routeData">

<RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

</Found>

<NotFound>

<LayoutView Layout="@typeof(MainLayout)">

<p>Sorry, there's nothing at this address.</p>

</LayoutView>

</NotFound>

</Router>

In this setup:

* **<Router>** is responsible for intercepting browser navigation and rendering the component that matches the current URL.
* **AppAssembly="@typeof(Program).Assembly"** specifies the assembly to search for pages and components.
* **<Found>** is displayed when a matching route is found, and **<RouteView>** renders the component for the matched route.
* **DefaultLayout="@typeof(MainLayout)"** specifies the layout component that's used by default for pages in the application. You might need to adjust this to match the name of your layout component if it's different.
* **<NotFound>** is displayed when no matching route is found, providing a basic "not found" message. You can customize this section as needed.

Make sure you have a **MainLayout.razor** file in your **Shared** folder or specify a different layout if you're using one. The **MainLayout.razor** file usually contains the layout for your app, including headers, footers, and other common elements.

Additionally, ensure that each of your destination components (e.g., **Quotes.razor**, **Login.razor**, **Holdings.razor**, **AccountForm.razor**) includes a **@page** directive at the top with the correct route. For example, your **Quotes.razor** file should start with **@page "/quotes"** to be accessible via the **/quotes** route.

If you've followed these steps and are still encountering issues, it might be helpful to double-check the routes specified in your **@page** directives and ensure they match the paths used in **NavigationManager.NavigateTo** calls. Also, verify that all components referenced in navigation exist and are correctly named.

### Notes

Ok, I am going to simply create blank pages for: **Quotes.razor**, **Login.razor**, **Holdings.razor**, **AccountForm.razor** to see if this fixes it.

It does not, same error.

## ChatGTP Interaction 40 - Output Code Optimization/Fixup

### Input

If my existing App.Razor component looks like this, where do I add the code you generated me for App.Razor in above answer: my existing App.Razor looks like:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<base href="/" />

<link rel="stylesheet" href="app.css" />

<link rel="stylesheet" href="TradeBlazorApp.styles.css" />

<HeadOutlet />

</head>

<body>

<Routes />

<script src="\_framework/blazor.web.js"></script>

</body>

</html>

### Output

The content you've provided for your existing **App.razor** looks like it's actually intended for a layout file that's typically used in Blazor WebAssembly applications, not Blazor Server applications. In a Blazor Server application, the layout and root component setup are usually quite different, and the **App.razor** file doesn't contain HTML **<head>** and **<body>** tags directly.

For a Blazor Server application, the **App.razor** file should primarily contain the **<Router>** component and related routing logic, rather than full HTML document structure. The HTML structure is usually defined in **\_Layout.cshtml** (for pages and MVC views) and **\_Host.cshtml** (specific to Blazor Server), but as you mentioned, the newer template doesn't use **\_Host.cshtml** and might have a slightly different structure.

Given this, your **App.razor** file should focus on the routing aspect of your Blazor application. Here's a corrected version that aligns with typical Blazor Server applications:

<Router AppAssembly="@typeof(Program).Assembly">

<Found Context="routeData">

<RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

</Found>

<NotFound>

<LayoutView Layout="@typeof(MainLayout)">

<p>Sorry, there's nothing at this address.</p>

</LayoutView>

</NotFound>

</Router>

In this setup:

* **<Router>** handles the routing within your Blazor app.
* **<Found>** displays when a route matches, with **<RouteView>** rendering the matched component.
* **<NotFound>** provides content when no routes match.

Since your **App.razor** file contains HTML structure that's not typical for **App.razor**, you should:

1. Remove the HTML, **<head>**, and **<body>** tags from **App.razor**.
2. Ensure the above routing logic is present in **App.razor**.

For styling and scripts, use the **\_Layout.cshtml** file or include them in individual component files. The Blazor framework will inject the components into the layout at runtime.

If you're working with a Blazor WebAssembly app instead, the structure might indeed include HTML tags directly in the **App.razor** or another root component file, but the presence of **<script src="\_framework/blazor.web.js"></script>** suggests a Blazor Server application, which typically uses **\_framework/blazor.server.js**.

Double-check your project setup to ensure it aligns with the Blazor Server model and adjust your files accordingly. If your project is set up differently or uses a custom structure, you might need to adapt these instructions to fit your specific setup.

### Notes

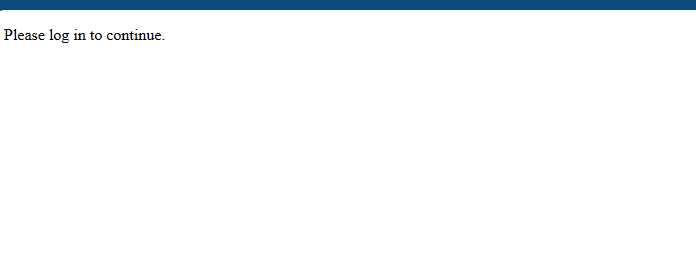
Ok—after an hour, looking carefully at my project files, I found I needed to modify Program.cs. I basically delegted the Maptofallback lines, when to basic template structure after Builder.Build() call. Then found I needed this line as well, before Builder.Build:  
  
// Add services to the container.

builder.Services.AddRazorComponents()

.AddInteractiveServerComponents();

ChatGTP did not really help me here, to get my root page at “/” (Home.Razor) to load.  
  
I figured out on my own, though just slight modifications were required. I hope I did not remove necessary elements from Program.cs when I simplified it.

My first rendered (Home) Page looks like this:



Not exactly what I expected!

I see the code ChatGTP generated is simply, well wrong. I will try again, see what I get. I may end up having to do some manual fixup of generated logic, I see what is happening here.

## ChatGTP Interaction 41 – Direct Code Migration UI Tier

### Input

Can you migrate this VB Winform to a new Razor Component, named "Home.Razor"?

Public Class MainNav

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

GlobalSettings.LoggedIn = False

End Sub

Private Sub Label3\_Click(sender As Object, e As EventArgs) Handles Label3.Click

Form1.Show()

End Sub

Private Sub Label2\_Click(sender As Object, e As EventArgs) Handles Label2.Click

If GlobalSettings.LoggedIn Then

MsgBox("You are now Logged Out!")

GlobalSettings.LoggedIn = False

Login.PasswordTextBox.Text = ""

Login.UserIDTextBox.Text = ""

Else

Login.Show()

End If

End Sub

Private Sub Label4\_Click(sender As Object, e As EventArgs) Handles Label4.Click

If GlobalSettings.LoggedIn Then

Dim holdingsForm As New Holdings()

holdingsForm.Show()

Else

MsgBox("You Must Login First!")

End If

End Sub

Private Sub Label5\_Click(sender As Object, e As EventArgs) Handles Label5.Click

If GlobalSettings.LoggedIn Then

Dim accountForm As New AccountForm()

accountForm.Show()

Else

MsgBox("You Must Login First!")

End If

End Sub

End Class

### Output

Migrating a VB.NET WinForms application to a Blazor component involves adapting the imperative event-driven model of WinForms to the declarative and component-based model of Blazor. For your **MainNav** WinForms class, we'll create a new Razor component named **Home.razor** that reflects the functionality of your original form.

This example assumes you have a mechanism in place to manage global settings and user authentication status, similar to your **GlobalSettings.LoggedIn** flag. In a Blazor app, this is often handled through services that use dependency injection. Also, the user interface interactions like showing forms and displaying messages will be converted to navigation and modal or inline messages in Blazor.

Here's an example of what the **Home.razor** component might look like:

@page "/home"

@inject NavigationManager NavigationManager

@inject IAuthenticationService AuthenticationService

<PageTitle>Home</PageTitle>

@if (!AuthenticationService.IsLoggedIn)

{

<p>Please log in to continue.</p>

}

else

{

<div>

<button @onclick="NavigateToForm1">Go to Form 1</button>

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="NavigateToHoldings">Holdings</button>

<button @onclick="NavigateToAccountForm">Account</button>

</div>

}

@code {

private void NavigateToForm1()

{

// Assuming you have a corresponding Razor component for Form1

NavigationManager.NavigateTo("/form1");

}

private void ToggleLogin()

{

if (AuthenticationService.IsLoggedIn)

{

AuthenticationService.Logout();

// Optionally display a message indicating logout

}

else

{

// Navigate to the login page/component

NavigationManager.NavigateTo("/login");

}

}

private void NavigateToHoldings()

{

if (AuthenticationService.IsLoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show a message or modal indicating the need to log in

}

}

private void NavigateToAccountForm()

{

if (AuthenticationService.IsLoggedIn)

{

NavigationManager.NavigateTo("/account");

}

else

{

// Show a message or modal indicating the need to log in

}

}

}

In this component:

* The **@page** directive specifies the URL at which this component is accessible.
* Dependency injection is used to inject services like **NavigationManager** for navigation and an example **IAuthenticationService** to manage login state. You'll need to implement **IAuthenticationService** based on your app's authentication logic.
* Conditional rendering (**@if** blocks) is used to display content based on the user's login state.
* The **@onclick** event handlers are bound to methods that either navigate to other components or perform actions like logging in/out.

Remember to create corresponding Razor components for each of the pages you're navigating to (**/form1**, **/login**, **/holdings**, **/account**) and implement the **IAuthenticationService** or equivalent service to manage the authentication state across the app.

### Notes

It generated the same, wrong logic. I will fix manually. OK, I now get buttons for navigation, but code from button clicks is not getting triggered. Why?

## ChatGTP Interaction 42 - Output Code Optimization/Fixup

### Input

In this .NET 8 Blazor hyrbid application, when I click on any button, the corresponding methods in the @code block are not getting triggered, why?

@page "/"

@using TradeBlazorApp.Business\_Classes

@inject NavigationManager NavigationManager

<PageTitle>Trade Home</PageTitle>

<!--This entire page was first generated by ChatGTP by migrating the MainNav form from VB.NET WIn Forms, then manually fixed up bellow where noted-->

<div>

<button @onclick="ShowQuotes">Quotes</button> <!--I Changed the name here from Form1 to Quotes-->

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

@code {

// Your code logic remains the same here

private void ShowQuotes() //I changed from ShowForm1 to ShowQuotes.

{

NavigationManager.NavigateTo("/quotes");

//I Changed the name here from form1 to quotes

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Add logic to reset login state as needed

}

else

{

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/accountform");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

}

### Output

If the methods in the **@code** block are not getting triggered when you click on the buttons in your Blazor component, it could be due to several reasons. Let's troubleshoot the most common ones:

**1. Method Accessibility**

Ensure that the methods you're calling from the **@onclick** directive are **public**. In your case, the methods are defined as **private**, which should generally work in Blazor components, but changing them to **public** is worth a try to rule out accessibility issues.

public void ShowQuotes() { ... }

public void ToggleLogin() { ... }

public void ShowHoldings() { ... }

public void ShowAccountForm() { ... }

**2. Event Handler Delegate**

The **@onclick** directive expects a delegate. While Blazor usually handles method group conversions implicitly, explicitly defining a lambda expression can sometimes resolve issues where methods aren't being invoked.

<button @onclick="(() => ShowQuotes())">Quotes</button>

<button @onclick="(() => ToggleLogin())">Toggle Login</button>

<button @onclick="(() => ShowHoldings())">Holdings</button>

<button @onclick="(() => ShowAccountForm())">Account</button>

**3. Component Rendering Issues**

If the component isn't re-rendering as expected, it might not detect clicks. This is less likely the issue if the component renders correctly but is something to consider if there are other rendering problems.

**4. JavaScript Errors**

Check your browser's developer console for JavaScript errors. JavaScript errors can interfere with Blazor's interop and prevent event handlers from firing.

**5. Blazor Framework Issues**

Ensure you're using the latest stable version of Blazor and that there are no known bugs with event handling in your particular version. Consider updating your project's NuGet packages if updates are available.

**6. GlobalSettings Access**

Your code references **GlobalSettings.LoggedIn** directly. Ensure that **GlobalSettings** is accessible and correctly configured to be used within your Blazor component. If **GlobalSettings** is a static class from your migrated WinForms app, consider using a Blazor service or a cascading parameter to manage and access global settings.

If after trying these suggestions the issue persists, you might want to create a minimal reproducible example and seek further assistance, potentially by raising an issue in the Blazor GitHub repository or a Blazor community forum.

### Notes

OK. This is very basic stuff. Why not work?   
I tried both suggestions 1 and 2, still does not work—no event is firing on button clicks.

I am going back to my MSLearnings sample app, which clearly handled button clicks just fine!  
  
AHA! I need to add this line to my pages: ChatGTP completely failed here, had to figure it out on my own. An hour or so wasted.

SUCH A SIMPLE FIX.

I now have a couple of paths I could first make this page pretty, or else, continue building next page, which will be login. Lets go for building the login page.

## ChatGTP Interaction 43

### Input

Assume for the next questions I am migrating a VB.NET Winforms App to a .NET 8 BlazorApp, targeting Microsoft EntityFrameworkcore 8.02 and using rendermode = InteractiveServer in my Blazor Components.

### Output

Understood, you're migrating a VB.NET WinForms application to a .NET 8 Blazor App, and you're using Entity Framework Core 8.0.2 for data access. Additionally, you're using **InteractiveServer** as the render mode for your Blazor components. Please go ahead with your questions or specify any particular aspects of the migration process where you need assistance.

### Notes

## ChatGTP Interaction 44 - Direct Code Migration UI Tier

### Input

Can you migrate this VB.NET WinForm to a Blazor Component please?  
  
'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

Public Class Login

Private Sub Login\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

End Sub

Private Sub LogInButton\_Click(sender As Object, e As EventArgs) Handles LogInButton.Click

Dim thisAccount As New AccountCurrent

thisAccount = thisAccount.Login(UserIDTextBox.Text, PasswordTextBox.Text)

If GlobalSettings.LoggedIn = False Then

MsgBox("Login Failed for " + UserIDTextBox.Text + ". Try Again")

GlobalSettings.theLoginAccount = Nothing

Else

MsgBox("Login Success for " + UserIDTextBox.Text + ". AccountID is " + thisAccount.AccountID.ToString)

GlobalSettings.theLoginAccount = thisAccount

End If

Me.Hide()

End Sub

End Class

### Output

To migrate your VB.NET WinForms **Login** form to a .NET 8 Blazor component, we need to adapt the form's UI and event-driven model to Blazor's component model. In Blazor, UI rendering is handled through Razor syntax and component interactions are managed via data binding and event callbacks.

The Blazor version will use a form with two-way data binding for the user ID and password, and will handle the login logic in an event callback method. Given that you're using Entity Framework Core for data access, we'll assume you have a similar **AccountCurrent** class and a **Login** method in your Blazor application.

Here's an example of how the **Login** component might look in Blazor:

@page "/login"

@inject IAccountService AccountService

@inject NavigationManager NavigationManager

<PageTitle>Login</PageTitle>

@if (!String.IsNullOrEmpty(loginMessage))

{

<p>@loginMessage</p>

}

<EditForm Model="@loginModel" OnValidSubmit="HandleLogin">

<DataAnnotationsValidator />

<ValidationSummary />

<div>

<label for="userId">User ID:</label>

<InputText id="userId" @bind-Value="loginModel.UserId" />

</div>

<div>

<label for="password">Password:</label>

<InputText id="password" @bind-Value="loginModel.Password" type="password" />

</div>

<button type="submit">Log In</button>

</EditForm>

@code {

private LoginModel loginModel = new LoginModel();

private string loginMessage;

private async Task HandleLogin()

{

var result = await AccountService.Login(loginModel.UserId, loginModel.Password);

if (result == null)

{

loginMessage = $"Login Failed for {loginModel.UserId}. Try Again";

}

else

{

loginMessage = $"Login Success for {loginModel.UserId}. AccountID is {result.AccountID}";

// Navigate to another page or update global settings as needed

NavigationManager.NavigateTo("/");

}

}

public class LoginModel

{

public string UserId { get; set; }

public string Password { get; set; }

}

}

**Explanation:**

* **@inject IAccountService AccountService**: This line assumes you have a service (**IAccountService**) that handles account-related operations, including login. This service should be registered in your **Startup.cs** or **Program.cs** file for dependency injection.
* **@inject NavigationManager NavigationManager**: Used for navigating to other pages upon successful login.
* **<EditForm>**: A Blazor component for handling form submissions. **OnValidSubmit** is triggered when the form is submitted and is bound to the **HandleLogin** method.
* **@bind-Value**: This directive creates a two-way binding between the input fields and the properties of **loginModel**.
* **HandleLogin**: This method performs the login operation using **AccountService**. Upon success or failure, it updates the **loginMessage** which is displayed to the user.
* **LoginModel**: A simple class used to bind values from the form.

**Notes:**

* Ensure your **IAccountService** and its implementation are properly set up to interact with your database using Entity Framework Core.
* This example uses client-side validation with **<DataAnnotationsValidator />** and **<ValidationSummary />**. Make sure your **LoginModel** class has the appropriate validation attributes if needed.
* You'll need to adjust the navigation and global settings handling (**GlobalSettings.theLoginAccount**) to fit the Blazor app model, potentially using services or cascading parameters.

### Notes

I had to add a using statement for Business\_Classes namespace. Then I had to change IAccountServices to AccountCurrent class in biz layer.

I now have an issue with the method login call, where VS tells me that AccountService.login does not contain a definition for GetAwaiter

## ChatGTP Interaction 45 - Output Code Optimization/Fixup

### Input

From above code, Visual Studio tells me: "AccountService.login does not contain a definition for GetAwaiter" Fix?

### Output

The error message "AccountService.Login does not contain a definition for GetAwaiter" suggests that the **Login** method in your **AccountService** is not asynchronous but is being awaited in the **HandleLogin** method. In C#, the **await** keyword can only be used with methods that return a **Task** or **Task<T>**.

To fix this issue, you have a couple of options:

**1. Make the Login Method Asynchronous**

If your **Login** method performs I/O operations like database access, especially using Entity Framework, it's recommended to make it asynchronous. Modify the **Login** method in your **AccountService** to be **async** and return a **Task** or **Task<T>**. For example:

public async Task<AccountCurrent> Login(string userId, string password)

{

// Your login logic here, using async EF Core methods like ToListAsync, FirstOrDefaultAsync, etc.

}

Then, your call to **AccountService.Login** in the Blazor component remains the same:

var result = await AccountService.Login(loginModel.UserId, loginModel.Password);

**2. Remove await if the Login Method is Synchronous**

If for some reason you need to keep the **Login** method synchronous, you should remove the **await** keyword when calling it:

private async Task HandleLogin()

{

var result = AccountService.Login(loginModel.UserId, loginModel.Password);

// Rest of your logic remains the same

}

However, be cautious with this approach, as blocking calls in a Blazor Server application can lead to scalability issues. It's generally best practice to use asynchronous methods for I/O operations.

**Additional Note**

If you modify the **Login** method to be asynchronous, make sure to use asynchronous versions of any Entity Framework methods you're calling within it, such as **FirstOrDefaultAsync** instead of **FirstOrDefault**, to avoid blocking calls.

### Notes

Well, I want this to be synchronous, I guess, for now. Will re-evalute though.

Runtime Error: **An unhandled exception occurred while processing the request.**

InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'.

## ChatGTP Interaction 46 - Output Code Optimization/Fixup

### Input

In this .NET 8 Blazor component:

@page "/"

@rendermode InteractiveServer

@using TradeBlazorApp.Business\_Classes

@inject NavigationManager NavigationManager

<PageTitle>Trade Home</PageTitle>

<!--I had to add rendermode InteractiveServer line manually, else no events fire-->

<!--This entire page was first generated by ChatGTP by migrating the MainNav form from VB.NET WIn Forms, then manually fixed up bellow where noted-->

<div>

<button @onclick="ShowQuotes">Quotes</button> <!--I Changed the name here from Form1 to Quotes-->

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

@code {

// Your code logic remains the same here

public void ShowQuotes() //I changed from ShowForm1 to ShowQuotes.

{

NavigationManager.NavigateTo("/quotes");

//I Changed the name here from form1 to quotes

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Add logic to reset login state as needed

}

else

{

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/accountform");

}

else

{

// Show message - consider using a modal or a toast notification component

}

}

}"

I get the following runtime error, how to fix?

InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'.

### Output

The error message you're encountering, **InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'.**, indicates that your Blazor app is trying to inject a service **AccountService** (or directly an instance of **AccountCurrent** based on the error message) into one of your components, but the service hasn't been registered in the app's dependency injection (DI) container.

To fix this error, ensure that the service you're trying to inject is properly registered in the DI container within your application startup code. Since you're using .NET 8, this code is likely in your **Program.cs** file.

Here's a general approach to registering a service:

**Step 1: Define the Service Interface**

First, ensure you have an interface that defines the contract for your **AccountService**. For example:

public interface IAccountService

{

Task<AccountCurrent> Login(string userId, string password);

// Other methods...

}

**Step 2: Implement the Service**

Ensure you have an implementation of the **IAccountService**. For example:

public class AccountService : IAccountService

{

public async Task<AccountCurrent> Login(string userId, string password)

{

// Implementation...

}

// Other method implementations...

}

**Step 3: Register the Service in DI Container**

In your **Program.cs**, register the **IAccountService** with its implementation **AccountService** in the DI container:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddScoped<IAccountService, AccountService>();

// Other service registrations...

var app = builder.Build();

// Configure the HTTP request pipeline and other app settings...

app.Run();

**Important Notes:**

* The service registration type (**AddScoped**, **AddSingleton**, **AddTransient**) depends on the desired service lifetime. **AddScoped** is commonly used in Blazor Server applications.
* Ensure that you're injecting the interface **IAccountService** into your components, not the implementation **AccountService**. This promotes loose coupling and makes your application more maintainable and testable.
* If the **AccountCurrent** class is being directly injected (as suggested by the error message), consider wrapping its functionality in a service like **IAccountService** as shown above. Directly injecting data models as services is not a common practice and can lead to design issues.

By following these steps, you should be able to resolve the **InvalidOperationException** and successfully use your **AccountService** within your Blazor components.

### Notes

OK. Another crossroads. My migrated business logic classes have working implementations within my Blazor app, and compile . But the methods are not setup for Async (or sync) Blazor Service calls. I think I will need to wrap them, including interfaces as suggested above.

For example:

public async Task<AccountCurrent> Login(string userId, string password)

Need to figure out how to do, so return type is Task<AccountCurrent>, for example in this method.

## ChatGTP Interaction 47 - Output Code Optimization/Fixup

### Input

Assume I have the following class, and need to make it's method Login callable from a Blazor Component. I am confused, because you earlier indicated I needed to first Define a Service Interface, then register that in the DI Container (Program.cs). But my method Login does not return a type of:

public class AccountService : IAccountService

{

public async Task<AccountCurrent> Login(string userId, string password)

{

// Implementation...

}

// Other method implementations...

}

Rather, my Login Method within the AccountCurrent class, looks as follows:

"public class AccountCurrent

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

// Assuming ACCOUNTDBEntities is your DbContext and ACCOUNTPROFILEs is a DbSet<AccountProfile>

private readonly ACCOUNTDBEntities \_accountContext;

// Constructor injection for the DbContext

public AccountCurrent(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public AccountCurrent Login(string userid, string password)

{

var theAccount = new AccountCurrent(\_accountContext);

// Use EF Core's method to execute SQL directly, ensure parameterization for security

var parameter = new SqlParameter("@userId", userid);

var accountProfiles = \_accountContext.ACCOUNTPROFILE // Manual Fixup of DBSet name here, required. (All Caps, name generated no 's'

.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter)

.ToList(); // Execute the query and get the result

foreach (var profile in accountProfiles)

{

theAccount.UserID = profile.UserId; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.Password = profile.Password; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.AccountID = profile.AccountId; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.Salt = profile.Salt;

theAccount.Name = profile.FullName;

// Continue mapping other properties as needed

}

if (theAccount.Password == null)

{

// Assuming GlobalSettings is a static class you use for global settings

GlobalSettings.LoggedIn = false;

}

else

{

// Assuming SaltedHash is a class you have for hashing and verifying passwords

var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password); //I had to manually fixup this line of code generated by Interaction 20, to use a typename and not an instance

GlobalSettings.LoggedIn = verifier.Verify(password);

}

return theAccount;

}

}"

How do I adapt this to what Blazor expects, I want to call the Login method within my Blazor Component?

### Output

To adapt your **AccountCurrent** class and its **Login** method to be used within a Blazor component following best practices, you'll want to encapsulate the data access logic within a service that implements an interface. This service can then be injected into your Blazor components.

Given your **AccountCurrent** class, let's define an interface and a service that wraps the login functionality:

**Step 1: Define the Service Interface**

Define an interface that describes the contract for your account-related operations, including login:

public interface IAccountService

{

Task<AccountCurrent> LoginAsync(string userId, string password);

}

**Step 2: Implement the Service**

Create a service that implements this interface. This service will use an instance of **ACCOUNTDBEntities** (your DbContext) to perform the login operation:

public class AccountService : IAccountService

{

private readonly ACCOUNTDBEntities \_accountContext;

public AccountService(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public async Task<AccountCurrent> LoginAsync(string userId, string password)

{

// Your existing login logic here, adapted for async execution

var theAccount = new AccountCurrent();

// Adapt this part to async if possible, for example by using ToListAsync()

var accountProfiles = await \_accountContext.ACCOUNTPROFILE

.FromSqlInterpolated($"SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = {userId}")

.ToListAsync();

// The rest of your login logic...

return theAccount;

}

}

**Step 3: Register the Service in DI Container**

In your **Program.cs**, register the **IAccountService** with its implementation **AccountService** in the DI container, ensuring that **ACCOUNTDBEntities** is also registered if it's not already:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("YourConnectionStringName")));

builder.Services.AddScoped<IAccountService, AccountService>();

var app = builder.Build();

// Configure the HTTP request pipeline and other app settings...

app.Run();

**Step 4: Inject and Use the Service in Your Blazor Component**

Now you can inject **IAccountService** into your Blazor components and use it to perform login operations:

@inject IAccountService AccountService

<!-- Your Blazor component's HTML and Razor markup -->

@code {

private string userId;

private string password;

private async Task HandleLogin()

{

var account = await AccountService.LoginAsync(userId, password);

if (account != null && GlobalSettings.LoggedIn)

{

// Login successful, proceed accordingly

}

else

{

// Login failed, handle accordingly

}

}

}

**Notes:**

* Ensure your **LoginAsync** method and the **IAccountService** interface are asynchronous to make the best use of Entity Framework's capabilities and to avoid blocking calls in your Blazor app.
* Consider security best practices, especially when dealing with user passwords and direct SQL queries. Ensure your queries are parameterized to prevent SQL injection, and never store plain-text passwords. Use hash and salt for storing and verifying passwords.
* The **GlobalSettings.LoggedIn** flag usage in a Blazor application might not be the best approach, especially in server-side Blazor due to its stateful nature. Consider using Blazor's authentication and authorization mechanisms or a scoped service to manage user sessions.

### Notes

I did the fixup. Compiles, but get this runtime error on Login page:  
  
InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business**\_Classes.AccountCurrent'.**

## ChatGTP Interaction 48 - Output Code Optimization/Fixup

### Input

I did as you instructed, but when running the page/navigating to, I get this error now: InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'." Fix?

### Output

The error message you're encountering, **InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'.**, suggests that there's a mismatch between what's being injected into your Blazor component and the services registered in the dependency injection (DI) container. Specifically, it seems the Blazor component might be trying to inject an **AccountCurrent** instance directly, rather than the **IAccountService** interface or the **AccountService** implementation.

Here's how to fix this issue:

**1. Ensure Correct Service Injection in Blazor Component**

First, confirm that your Blazor component is injecting the **IAccountService**, not **AccountCurrent**. The **@inject** directive should look like this:

@inject IAccountService AccountService

The error message you're encountering, **InvalidOperationException: Cannot provide a value for property 'AccountService' on type 'TradeBlazorApp.Components.Pages.Login'. There is no registered service of type 'TradeBlazorApp.Business\_Classes.AccountCurrent'.**, suggests that there's a mismatch between what's being injected into your Blazor component and the services registered in the dependency injection (DI) container. Specifically, it seems the Blazor component might be trying to inject an **AccountCurrent** instance directly, rather than the **IAccountService** interface or the **AccountService** implementation.

Here's how to fix this issue:

**1. Ensure Correct Service Injection in Blazor Component**

First, confirm that your Blazor component is injecting the **IAccountService**, not **AccountCurrent**. The **@inject** directive should look like this:

razorCopy code

@inject IAccountService AccountService

Ensure there's no line trying to inject **AccountCurrent** directly, as it's a model and not a service.

**2. Verify Service Registration in Program.cs**

Next, double-check your **Program.cs** file to ensure that the **IAccountService** is registered correctly with its implementation (**AccountService**). It should look something like this:

builder.Services.AddScoped<IAccountService, AccountService>();

This line tells the DI container that whenever an **IAccountService** is requested, it should provide an instance of **AccountService**.

**3. Check Constructor Injection in AccountService**

Ensure that the **AccountService** implementation is correctly expecting an **ACCOUNTDBEntities** instance (your DbContext) and not an **AccountCurrent**:

public class AccountService : IAccountService

{

private readonly ACCOUNTDBEntities \_accountContext;

public AccountService(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

// Implementation...

}

**4. Review Other Possible Injection Points**

If you have other components or services, review them to ensure there's no place where **AccountCurrent** is mistakenly being injected instead of **AccountService** or **IAccountService**.

**5. Rebuild and Run**

After making the necessary adjustments, rebuild your application and run it again to see if the issue is resolved.

If the error persists, it might be helpful to search your entire project for any **@inject AccountCurrent** directives or constructor parameters expecting **AccountCurrent** directly and replace them with the correct service interface (**IAccountService**) and its implementation (**AccountService**).

### Notes

It works now (at least displays, triggers Login properly. Key was doing this in the Login.Razor page, modified from original instructions I got:  
  
@inject IAccountService AccountService

Compile and run Debug!  
  
I simply get “An Unhandled Exception Occurred”, no exception details when trying to login.

AHA MOMENT! I need better exception handling in my Blazor application, in key method calls, etc.  
I need to add try/catch blocks, and dispay actual error in a label, for this and all other pages. I will use ChatGTP, to request it adds this automatically going forward, but first I will manually add in the Login.Blazor component/page, to see if works.  
  
Yes it does, my exception is, now displayed in a “debug” label:  
  
System.InvalidOperationException: The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'. at Microsoft.EntityFrameworkCore.Metadata.Conventions.KeyAttributeConvention.ProcessModelFinalizing(IConventionModelBuilder modelBuilder, IConventionContext`1 context) at Microsoft.EntityFrameworkCore.Metadata.Conventions.Internal.ConventionDispatcher.ImmediateConventionScope.OnModelFinalizing(IConventionModelBuilder modelBuilder)

Since my starting VB.NET WinForms app had at best sketchy exception handling, I will also need likely try/catch/throw blocks in my Data Classes and Business Classes. I likely could have had ChatGTP add these for my, when migrating/creating those classes from VB.NET sources. But I did not. Will attempt to “optimize” those later, towards end. But will have new Blazor Components added/generated by ChatGTP qualifying I want proper exception handling and display.

For now, on to fixing this exception in Login.Blazor.

## ChatGTP Interaction 49 - Output Code Optimization/Fixup

### Input

On the following call in my .NET Blazor app business service, using Entity Framework 8.0.2, I get this runtime exception, how to fix:

**“System.InvalidOperationException:** 'The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'.' “  
  
On the call to

var accountProfiles = \_accountContext.ACCOUNTPROFILE.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter).ToList();

In this class:  
  
//All of this code generated and pasted from ChatGTP Interaction 20 -- for the entire class!

using Microsoft.EntityFrameworkCore;

using Microsoft.Data.SqlClient;

using System;

using TradeBlazorApp.Data\_Classes;

public interface IAccountService

{

Task<AccountCurrent> LoginAsync(string userId, string password);

// Other methods...

}

public class AccountCurrent : IAccountService //Via ChatGTP Interaction 47: I need to implement Blazor-Style Interfaces for services, and Async logic. Added this line.

{

public string Password { get; set; }

public string UserID { get; set; }

public string Name { get; set; }

public decimal AccountID { get; set; }

public string Salt { get; set; }

// Assuming ACCOUNTDBEntities is your DbContext and ACCOUNTPROFILEs is a DbSet<AccountProfile>

private readonly ACCOUNTDBEntities \_accountContext;

// Constructor injection for the DbContext

public AccountCurrent(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public async Task<AccountCurrent> LoginAsync(string userId, string password)

{

var theAccount = new AccountCurrent(\_accountContext);

// Use EF Core's method to execute SQL directly, ensure parameterization for security

var parameter = new SqlParameter("@userId", userId);

var accountProfiles = \_accountContext.ACCOUNTPROFILE.FromSqlRaw("SELECT \* FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter).ToList(); foreach (var profile in accountProfiles)

{

theAccount.UserID = profile.UserId; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.Password = profile.Password; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.AccountID = profile.AccountId; // Manual Fixup of property name here, required. ( from all Caps, as originally generated)

theAccount.Salt = profile.Salt;

theAccount.Name = profile.FullName;

// Continue mapping other properties as needed

}

if (theAccount.Password == null)

{

// Assuming GlobalSettings is a static class you use for global settings

GlobalSettings.LoggedIn = false;

}

else

{

// Assuming SaltedHash is a class you have for hashing and verifying passwords

var verifier = SaltedHash.Create(theAccount.Salt, theAccount.Password); //I had to manually fixup this line of code generated by Interaction 20, to use a typename and not an instance

GlobalSettings.LoggedIn = verifier.Verify(password);

}

return theAccount;

}

}”

### Output

The error you're encountering, **System.InvalidOperationException: 'The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'.'**, indicates an issue with how the primary key is defined on your **Account** entity (or possibly on **ACCOUNTPROFILE** if that's the entity in question).

In Entity Framework Core, an entity should have a single property marked with the **[Key]** attribute to denote it as the primary key. If an entity is intended to have a composite primary key (a primary key made up of more than one property), you should not use the **[Key]** attribute. Instead, you must configure it using the Fluent API in your **DbContext**'s **OnModelCreating** method.

**Fixing the Model**

First, ensure that your entity class has at most one **[Key]** attribute if it's intended to have a simple primary key:

public class AccountProfile

{

[Key]

public decimal AccountID { get; set; }

// Other properties...

}

If your entity is supposed to have a composite key, remove any **[Key]** attributes from the properties and configure the composite key in the **OnModelCreating** method:

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<AccountProfile>().HasKey(e => new { e.Property1, e.Property2 });

// Replace 'Property1' and 'Property2' with the actual property names that make up the composite key

}

**Adjusting the Service Implementation**

Additionally, there are a few adjustments needed in your **AccountCurrent** service class:

* The **LoginAsync** method should not create a new instance of **AccountCurrent** (**var theAccount = new AccountCurrent(\_accountContext);**). This is unusual and might lead to issues since **AccountCurrent** is being used both as an entity and a service, which is not a standard practice.
* You should separate the concerns: keep **AccountCurrent** as an entity and create a separate service class (e.g., **AccountService**) that implements **IAccountService**.
* Avoid executing raw SQL queries directly unless necessary. Entity Framework Core provides LINQ methods that are safer and integrate better with the framework's change tracking and identity resolution features.

Here's a simplified example of what the **AccountService** might look like:

public class AccountService : IAccountService

{

private readonly ACCOUNTDBEntities \_accountContext;

public AccountService(ACCOUNTDBEntities accountContext)

{

\_accountContext = accountContext;

}

public async Task<AccountCurrent> LoginAsync(string userId, string password)

{

var accountProfile = await \_accountContext.ACCOUNTPROFILE

.FirstOrDefaultAsync(p => p.UserId == userId);

// Your login logic here...

return new AccountCurrent

{

// Map properties from accountProfile to AccountCurrent

};

}

}

Remember to register **AccountService** (not **AccountCurrent**) in the DI container, and adjust your Blazor components to inject **IAccountService**. This approach keeps your entity models clean and separates the data access logic into a dedicated service layer, which is a good practice in software architecture.

### Notes

Oh boy: I will work on these to start, lots to do here:  
  
**Adjusting the Service Implementation**

Additionally, there are a few adjustments needed in your **AccountCurrent** service class:

* The **LoginAsync** method should not create a new instance of **AccountCurrent** (**var theAccount = new AccountCurrent(\_accountContext);**). This is unusual and might lead to issues since **AccountCurrent** is being used both as an entity and a service, which is not a standard practice.
* You should separate the concerns: keep **AccountCurrent** as an entity and create a separate service class (e.g., **AccountService**) that implements **IAccountService**.
* Avoid executing raw SQL queries directly unless necessary. Entity Framework Core provides LINQ methods that are safer and integrate better with the framework's change tracking and identity resolution features.

## ChatGTP Interaction 50 - Output Code Optimization/Fixup

### Input

How do I "map" properties from AccountProfile to AccountCurrent in above: return new AccountCurrent { // Map properties from accountProfile to AccountCurrent };

### Output

To map properties from **AccountProfile** to **AccountCurrent** in the context of your service method, you would assign values from an **AccountProfile** instance to a new **AccountCurrent** instance property by property. This is a straightforward manual mapping approach.

Assuming **AccountProfile** is your entity with properties that correspond to the database fields, and **AccountCurrent** is a model you want to use within your application (perhaps to shape data specifically for use in the UI or business logic), you would do something like this:

public async Task<AccountCurrent> LoginAsync(string userId, string password)

{

var accountProfile = await \_accountContext.ACCOUNTPROFILE

.FirstOrDefaultAsync(p => p.UserId == userId);

if (accountProfile != null)

{

var accountCurrent = new AccountCurrent

{

UserID = accountProfile.UserId,

Password = accountProfile.Password,

Name = accountProfile.FullName, // Assuming FullName is the property in AccountProfile

AccountID = accountProfile.AccountId,

Salt = accountProfile.Salt,

// Map other necessary properties

};

// Your login verification logic...

return accountCurrent;

}

// Handle the case where the account is not found

return null;

}

In this example:

* An **AccountProfile** instance named **accountProfile** is retrieved from the database using Entity Framework Core's **FirstOrDefaultAsync** method, filtered by **UserId**.
* A new **AccountCurrent** instance named **accountCurrent** is created.
* Each property of **accountCurrent** is set to the corresponding value from **accountProfile**. The exact property names may differ between **AccountProfile** and **AccountCurrent**, and you might not map every property if some aren't needed in the specific context where **AccountCurrent** is used.
* The **accountCurrent** instance is returned from the method, now populated with the values from **accountProfile**.

**Notes:**

* **Mapping Complex Properties:** If there are complex properties or relationships, you might need to perform additional logic to map them correctly.
* **Automapper:** For more complex scenarios or to reduce boilerplate code, consider using a library like [AutoMapper](https://automapper.org/) to handle the mappings between objects. AutoMapper can significantly simplify the mapping process, especially when dealing with many entities and models.
* **Security:** Be cautious with password handling. In most cases, you should not be transferring hashed passwords around or exposing them to the client side. Ensure that any sensitive data is handled securely.

### Notes

I am having to piece together a bunch of advice here, to completely fixup my login method. FYI, the code I am fixing up was generated, upstream, by ChatGTP. This speaks to rule, MUST QUALIFY questions with EXACTLY what you want, when possible.

Lots of thinking here. I believe the core issue, is how, in early migration phase, I took VB.NET business classes, that had EntityFramework embedded within, and tried to separate out into Business Classes and Data Classes. But really, EntityFramework is doing this for me, and I need to stick with my entities defined in Data Classes, and only reference those within my Business Services tier, not create new entities like “AccountCurrent” which does not \*directly\* map to a DB Table. It should just be using Data Classes 🡪AccountProfile. Which maps to that ACCOUNTPROFILE table in SQL Server.  
  
So I have done massive fixup here, in this step, to just return type of Database\_Classes AccountProfile, not a new instance of a Business\_Classes “AccountCurrent”.  
  
After doing this fixup/learning (a couple of hours): I feel better but still at point of Exception, on Login button click:  
  
System.InvalidOperationException: The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'. At

I was expecting this, as per earlier ChatGTP interaction it also told me I would need to fix this, and I recall the issue is when I generated AccountProfile DB class from DDL (SQL Server): its clear AccountProfile has a composite key, not a single primary key. So I need to go back to that output, ChatGTP Interaction 49, and also fix this.

Working on now. Hope I do not have such issues with rest of my code/forms. Ha. I think just this AccountCurrentClass is the issue; as I learned later, a better way to do the re-factoring with ChatGTP and EntityFrameworkCore latest 8.0.2, by just going ahead and building Database Classes that directly mirror my database (2 databases) tables! Instead of mixing metaphors, so to speak, by having EntityFramework model classes directly in my business service tier. But, that is valuable learning, and based on fact I was starting with not-well-factored VB.NET WinForms code to begin with; and also trying to move from EntityFramework 6.4.4 (VB.NET/WinForms): to latest Entity FrameworkCore 8.0.2. To make “modern”. And remember:  
  
When I started I had:  
  
1) No knowledge of anything since .NET 4.5, no coding for last 14 years.  
2) No knowledge of EntityFramework, any version.  
3) No knowledge of latest VB.NET WinForms (changed a lot in 14 years).  
4) Zero knowledge of Blazor/Razor.  
5) Had not used Visual Studio since VS 2012!  
  
**Key lesson: ChatGTP really is HELPING ME A TON! I feel like this exercise scoped to 3 weeks, within 2 weeks has made me much better/serious skills update for me.**

## ChatGTP Interaction 51 - Output Code Optimization/Fixup

### Input

If my model class AccountProfile has a composite key, where do I place this code:

"protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<AccountProfile>().HasKey(e => new { e.Property1, e.Property2 });

// Replace 'Property1' and 'Property2' with the actual property names that make up the composite key

}"

to avoid this exception at runtime:

"System.InvalidOperationException: The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'. "

### Output

When migrating to a .NET Blazor Server application and using Entity Framework Core 8.0.2, handling composite keys requires careful configuration in your data context class, especially if you're encountering a **System.InvalidOperationException** related to multiple **[Key]** attributes.

The error message you're seeing suggests that your **AccountProfile** entity might have multiple properties decorated with the **[Key]** attribute, which is conflicting with the definition of a composite key using the **HasKey** method in the **OnModelCreating** method. In EF Core, a composite key should be defined either by using the **HasKey** method inside **OnModelCreating** or by applying the **[Key]** attribute on the properties in the model, but not both.

To resolve this issue, ensure that your **AccountProfile** class does not use the **[Key]** attribute on the properties that are part of the composite key. Instead, you should define the composite key solely in the **OnModelCreating** method of your **DbContext** subclass.

Here's a step-by-step guide on how to properly configure this:

1. **Remove [Key] Attributes**: First, ensure that **Property1** and **Property2** in your **AccountProfile** class do not have the **[Key]** attribute.
2. **Define Composite Key in OnModelCreating**: Place the **OnModelCreating** method override in your **DbContext** subclass. This class manages the database context and is where you configure mappings between your entity classes and database tables. If you haven't already, create a subclass of **DbContext** if you don't have one.
3. **Configure Composite Key**: Inside the **OnModelCreating** method of your **DbContext** subclass, configure the composite key as you've shown in your example. This method is called by the framework when your model is being created, and it's the right place to configure composite keys, relationships, and other configurations.

Here is an example of how your **DbContext** subclass might look:

public class YourDbContext : DbContext

{

public DbSet<AccountProfile> AccountProfiles { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// Configure the composite key for AccountProfile

modelBuilder.Entity<AccountProfile>().HasKey(e => new { e.Property1, e.Property2 });

}

}

Replace **YourDbContext** with the actual name of your **DbContext** subclass. Also, ensure that **Property1** and **Property2** are replaced with the actual property names in your **AccountProfile** class that constitute the composite key.

By following these steps, you should be able to avoid the **System.InvalidOperationException** and correctly configure the composite key for your **AccountProfile** entity.

### Notes

This worked, but now get runtime exception:  
  
System.InvalidOperationException: The entity type 'Account' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'. at Microsoft.EntityFrameworkCore.Metadata.Conventions.KeyAttributeConvention.ProcessModelFinalizing(IConventionModelBuilder modelBuilder, IConventionContext`1 context) at

So this is telling me I need to fixup all my DB model classes that have composite primary keys (I think all of them). Will start with Account. Success, Login now works.

Will build Quotes page next, from form.

## ChatGTP Interaction 52 – Direct Code Migration UI Tier

### Input

Can you migrate this VB.NET WinForms app that uses EntityFramework 6.4.4 to my .NET 8 Blazor Server application that uses EntityFrameworkCore 8.0.2?

"Imports System.Data.Entity ' Import Entity Framework namespace

'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

'It uses a DatagridView VB.NET Control

Public Class Form1

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

If GlobalSettings.LoggedIn Then

Label2.Text = "You Are Logged in as " + GlobalSettings.theLoginAccount.UserID

Label2.Visible = True

Label1.Text = "Click a Row to Buy a Stock"

Else

Label1.Text = "You are Not Logged In, So Cannot Buy"

Label2.Text = "Not Logged In, Please Login to Be Able to Place a Trade"

End If

End Sub

Private Sub DataGridView1\_CellClick(sender As Object, e As DataGridViewCellEventArgs) Handles DataGridView1.CellClick

If GlobalSettings.LoggedIn Then

' Check if the click is on a row, not the column header

If e.RowIndex >= 0 Then

' Access the clicked row

Dim row As DataGridViewRow = DataGridView1.Rows(e.RowIndex)

BuyForm.Show()

BuyForm.Form2\_Buy(row, e)

' Example: Display the value of the first cell in the row

' MessageBox.Show("Buy Stock: " & row.Cells("Symbol").Value.ToString())

End If

Else

MsgBox("You Must Login Before Selecting a Stock to Buy")

End If

End Sub

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click

' Initialize the DbContext

Using context As New QUOTEDBEntities()

' Query the database for all employees

context.QUOTEs.Load()

' Set the DataGridView's data source to the DbSet of Employees

DataGridView1.DataSource = context.QUOTEs.Local.ToBindingList()

DataGridView1.Columns("Symbol").DisplayIndex = 0

DataGridView1.Columns("CompanyName").DisplayIndex = 1

End Using

End Sub

Private Sub Button2\_Click(sender As Object, e As EventArgs) Handles Button2.Click

Me.Hide()

End Sub

End Class"

### Output

Migrating a VB.NET WinForms application that uses Entity Framework 6.4.4 to a .NET 8 Blazor Server application with EntityFrameworkCore 8.0.2 involves several steps and considerations. Given the complexity and the differences between WinForms and Blazor, as well as between EF 6 and EF Core, this will be a high-level overview rather than a line-by-line conversion.

**Step 1: Project Setup**

1. **Create a Blazor Server Application**: Start by creating a new Blazor Server project in Visual Studio or your preferred development environment.
2. **Add EntityFrameworkCore**: Ensure EntityFrameworkCore 8.0.2 is added to your project. You can do this by using the NuGet package manager.

**Step 2: Data Model Migration**

1. **Migrate Entity Models**: Your entity models need to be updated or rewritten to comply with EF Core conventions. This includes adjustments for any features that differ between EF 6 and EF Core.
2. **DbContext Configuration**: Rewrite your **DbContext** to use EF Core. This includes adjusting the **OnModelCreating** method and other configurations specific to EF Core.

**Step 3: Updating Data Access**

* **Data Loading**: Rewrite data access logic using EF Core's methods. Replace **.Load()** with **.ToList()**, **.ToBindingList()**, or async equivalents like **.ToListAsync()** for loading data.

**Step 4: UI Conversion to Blazor**

1. **Replace WinForms UI with Blazor Components**: Convert each WinForms control (like **DataGridView**, **Labels**, and **Buttons**) to Blazor components.
   * **DataGridView**: Use a Blazor component like a table or a third-party grid component that supports Blazor.
   * **Labels and Buttons**: Convert to equivalent HTML elements managed by Blazor components.
2. **Event Handling**: Convert event handlers (**Form1\_Load**, **DataGridView1\_CellClick**, **Button1\_Click**, etc.) to Blazor event handling mechanisms. This includes using **@onclick** for button clicks and other Blazor event binding techniques.

**Step 5: Implementing Business Logic**

* **Form Logic**: Adapt the form's logic to work within Blazor's component lifecycle. This includes managing user state, handling button clicks, and interacting with the database.

**Example Conversion Snippets**

Data Access in a Blazor Component

@code {

private List<QUOTE> quotes;

protected override async Task OnInitializedAsync()

{

using (var context = new QUOTEDBEntities())

{

quotes = await context.QUOTEs.ToListAsync();

}

}

}

UI Representation in Blazor

For the grid and buttons, you would have something like this in your Razor component:

@if (quotes != null)

{

<table class="table">

<thead>

<tr>

<th>Symbol</th>

<th>Company Name</th>

</tr>

</thead>

<tbody>

@foreach (var quote in quotes)

{

<tr @onclick="@(() => BuyStock(quote))">

<td>@quote.Symbol</td>

<td>@quote.CompanyName</td>

</tr>

}

</tbody>

</table>

}

<button @onclick="LoadQuotes">Load Quotes</button>

@code {

void LoadQuotes()

{

// Your logic to load quotes

}

void BuyStock(QUOTE quote)

{

// Your logic to handle stock purchase

}

}

**Step 6: Authentication and Authorization**

* **Authentication**: Since your WinForms app seems to use a global login state (**GlobalSettings.LoggedIn**), you'll need to implement authentication in Blazor using ASP.NET Core Identity or another authentication mechanism.

**Final Considerations**

* **Testing**: Thoroughly test your application to ensure all functionality has been migrated correctly.
* **Learning Blazor and EF Core**: If you're not already familiar, you'll need to spend some time learning Blazor and EF Core's paradigms and best practices.

This migration involves significant changes, both in terms of technology and architecture. It's essential to approach it incrementally, testing each part of the application as you go.

### Notes

I added the page and code logic. Fixup necessary. First:  
  
There is no argument given that corresponds to the required parameter 'options' of 'QUOTEDBEntities.QUOTEDBEntities(DbContextOptions<QUOTEDBEntities>)'

## ChatGTP Interaction 53 - Output Code Optimization/Fixup

### Input

In your suggestion above, this line: " using (var context = new QUOTEDBEntities())" produces an error on compile: "There is no argument given that corresponds to the required parameter 'options' of 'QUOTEDBEntities.QUOTEDBEntities(DbContextOptions<QUOTEDBEntities>)' TradeBlazorApp " How to fix?

### Output

The error you're encountering indicates that your **DbContext** (**QUOTEDBEntities** in your case) is expecting a **DbContextOptions<QUOTEDBEntities>** parameter in its constructor, which is a common requirement in Entity Framework Core to configure the context, such as specifying the database provider and connection string.

To fix this, you need to properly instantiate your **DbContext** with the required **DbContextOptions**. In a Blazor Server application, **DbContext** instances are typically configured and injected through the dependency injection (DI) container provided by ASP.NET Core. Here’s how you can do it:

**Step 1: Configure DbContext in Startup.cs**

In your **Startup.cs** (or **Program.cs** in .NET 6 and later), you need to add your **DbContext** to the DI container and specify its options, like the connection string:

public void ConfigureServices(IServiceCollection services)

{

services.AddRazorPages();

services.AddServerSideBlazor();

// Add DbContext with configuration

services.AddDbContext<QUOTEDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

}

public void ConfigureServices(IServiceCollection services)

{

services.AddRazorPages();

services.AddServerSideBlazor();

// Add DbContext with configuration

services.AddDbContext<QUOTEDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionStringName")));

}

Ensure you have the connection string defined in your **appsettings.json**:

{

"ConnectionStrings": {

"YourConnectionStringName": "Your actual database connection string here"

}

}

**Step 2: Inject DbContext into Blazor Component**

Instead of creating a new instance of your **DbContext** with **new**, you should inject it into your Blazor component:

@inject QUOTEDBEntities context

**Step 3: Use the Injected DbContext**

You can then use the injected **DbContext** directly in your component methods:

@code {

private List<QUOTE> quotes;

protected override async Task OnInitializedAsync()

{

quotes = await context.QUOTEs.ToListAsync();

}

}

With this setup, your **DbContext** is properly configured and instantiated by the DI container, and you won't need to manually create an instance with **new**, thus resolving the error.

**Additional Notes**

* Ensure you have the necessary using directives for EF Core and your data models.
* If your **DbContext** constructor has custom logic or configurations, make sure it also includes a constructor that accepts **DbContextOptions<QUOTEDBEntities>**.
* Be mindful of the lifecycle of **DbContext** instances in a Blazor application. They should be scoped appropriately to avoid issues such as memory leaks or concurrent access problems. The default service lifetime for **DbContext** in ASP.NET Core applications is scoped, which is generally appropriate for web applications, including Blazor Server apps.

### Notes

Starting with Step 2, Step 1 was already completed up the stream.

'DbSet<Quote>' does not contain a definition for 'ToListAsync' and no accessible extension method 'ToListAsync' accepting a first argument of type 'DbSet<Quote>' could be found (are you missing a using directive or an assembly reference?)

## ChatGTP Interaction 54 - Output Code Optimization/Fixup

### Input

I am targeting EntityFrameworkCore 8.0.2, and in the above line you generated: "quotes = await context.QUOTEs.ToListAsync();" Visual Studio tells me: 'DbSet<Quote>' does not contain a definition for 'ToListAsync' and no accessible extension method 'ToListAsync' accepting a first argument of type 'DbSet<Quote>' could be found (are you missing a using directive or an assembly reference?) Fix?

### Output

The error you're encountering typically occurs when the necessary namespace for the **ToListAsync** extension method is not included in your file, or the required package is not referenced in your project. The **ToListAsync** method is an extension method provided by the Entity Framework Core and is part of the **Microsoft.EntityFrameworkCore** namespace.

To resolve this issue, ensure the following:

**1. Add the necessary using directive**

Make sure you have the following **using** directive at the top of your C# file where you're attempting to use **ToListAsync**:

using Microsoft.EntityFrameworkCore;

This directive imports the namespace that contains the extension methods for **IQueryable**, including **ToListAsync**.

**2. Ensure EF Core package is installed**

Since you're targeting EntityFrameworkCore 8.0.2, verify that the **Microsoft.EntityFrameworkCore** package is installed and properly referenced in your project. Additionally, for asynchronous operations, you might need the **Microsoft.EntityFrameworkCore.SqlServer** package if you're using SQL Server, or the corresponding package for your specific database provider.

You can check and install these packages via the NuGet Package Manager in Visual Studio or by using the Package Manager Console with the following commands:

Install-Package Microsoft.EntityFrameworkCore -Version 8.0.2

And if you're using SQL Server:

powershell

Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 8.0.2

After ensuring the necessary using directive is added and the correct packages are installed, the **ToListAsync** method should be recognized without errors. If the issue persists, try cleaning and rebuilding your solution.

### Notes

So I am just missing a using statement. Adding to Quotes.Blazor.

Seems better now. Running.

Aha this again:  
  
InvalidOperationException: The entity type 'Quote' has multiple properties with the [Key] attribute. Composite primary keys configured by placing the [PrimaryKey] attribute on the entity type class, or by using 'HasKey' in 'OnModelCreating'.

Fixing—I know how to do now from previous interactions.

Success, my Quotes.Razor page loads, and displays on quotes on load.

However, two things:

1. Quotes are loading on page load, I want them to load on Load Quotes button click.
2. The table rows, now representing my VB DataGridView, are not clickable/fire event to trigger Buy.

## ChatGTP Interaction 55

### Input

Assume I am migrating a VB.NET WinForms app to .NET 8 Blazor Server app, targeting the Microsoft EntityFrameworkCore 8.0.2 for all questions.

### Output

Understood, your context for all questions is the migration of a VB.NET WinForms application to a .NET 8 Blazor Server application, with a focus on utilizing Microsoft EntityFrameworkCore 8.0.2. Please go ahead with your specific questions or concerns regarding this migration process.

### Notes

## ChatGTP Interaction 56 - Output Code Optimization/Fixup

### Input

I have the following Blazor Page, that works well, but I do not want to display all quotes on the page on page load, rather I want to have the quotes display in the table only after users presses the button “<button @onclick="LoadQuotes">Load Quotes</button>”  
  
How to fix?

@page "/quotes"

@using TradeBlazorApp.Business\_Classes

@using TradeBlazorApp.Data\_Classes

@using Microsoft.EntityFrameworkCore

@inject NavigationManager NavigationManager

@inject QUOTEDBEntities context //Added per ChatGTP INteraction 53

<**PageTitle**>Trade Quotes</**PageTitle**>

<h3>Quotes</h3>

@if (quotes != null)

{

<table class="table">

<thead>

<tr>

<th>Symbol</th>

<th>Company Name</th>

</tr>

</thead>

<tbody>

@foreach (var quote in quotes)

{

<tr @onclick="@(() => BuyStock(quote))">

<td>@quote.Symbol</td>

<td>@quote.CompanyName</td>

</tr>

}

</tbody>

</table>

}

<button @onclick="LoadQuotes">Load Quotes</button>

@code {

private List<Quote> quotes; // This line modified from ChatGTP codegen, simply to replace <QUOTE> with <Quote>

//The rest of this generated by ChatGTP Interaction 52

protected override async Task OnInitializedAsync()

{

{

quotes = await context.Quotes.ToListAsync();

}

}

void LoadQuotes()

{

// Your logic to load quotes

}

void BuyStock(Quote quote) //I manually Fixed case from QUOTE to Quote

{

// Your logic to handle stock purchase

Quote mytestquote;

mytestquote = quote;

}

}

### Output

To modify your Blazor page so that quotes are displayed only after the user presses the "Load Quotes" button, you need to move the logic that loads the quotes from the **OnInitializedAsync** method to the **LoadQuotes** method. The **OnInitializedAsync** method is executed as part of the component's initialization lifecycle, which is why your quotes are being loaded when the page is first loaded.

Here's how you can adjust your code:

1. Remove or comment out the **quotes = await context.Quotes.ToListAsync();** line from the **OnInitializedAsync** method, since you no longer want to load quotes during initialization.
2. Implement the logic to load quotes within the **LoadQuotes** method. This method should be asynchronous to await the loading of quotes from the database, so you'll need to change its signature to **async Task** and use **await** when calling **ToListAsync**.

Here is the adjusted **@code** block for your Blazor page:

@code {

private List<Quote> quotes;

// Removed the OnInitializedAsync method

private async Task LoadQuotes()

{

quotes = await context.Quotes.ToListAsync();

StateHasChanged(); // This may be necessary to inform Blazor that the component's state has changed and it needs to re-render.

}

void BuyStock(Quote quote)

{

// Your logic to handle stock purchase

// This is just a placeholder for whatever you want to do when a stock is clicked

}

}

With these changes, your Blazor page will initially load without displaying any quotes. The quotes will only be loaded and displayed after the user clicks the "Load Quotes" button, which triggers the **LoadQuotes** method, loads the quotes from the database, and updates the UI accordingly.

The **StateHasChanged** method call after loading the quotes is sometimes necessary to tell Blazor to re-check the component's state and update the UI if needed. It's particularly useful in event callbacks that might run after some asynchronous operations, like data loading in this case. However, Blazor usually manages this automatically for events like button clicks, so you might not need it here. If your UI updates as expected after loading quotes, you can omit the **StateHasChanged** call.

### Notes

Looks easy.

All good, but I always have to add “@rendermode InteractiveServer “

At top of every page, to get events to fire. Perhaps I could be qualifying my queries, so ChatGTP generates this line always/automatically. Like add “Please make sure any generated Razor .NET 8 components are using “@rendermode InteractiveServer “ in your generated Razor Components.

Anyway—this page now works as per the VB App. Nice.

## ChatGTP Interaction 57

### Input

Please make sure any generated Razor .NET 8 components are using “@rendermode InteractiveServer “ in your generated Razor Components.

### Output

To ensure that your Razor components in a .NET 8 Blazor Server application operate with interactive server-side rendering, you can specify the **@renderMode** directive with the value **InteractiveServer** in your Razor components. This directive controls how the component is initially rendered and ensures that the component is interactive once the initial rendering is completed and the browser has established a SignalR connection back to the server.

Here's how you can apply this in a Razor component:

@page "/quotes"

@using TradeBlazorApp.Business\_Classes

@using TradeBlazorApp.Data\_Classes

@using Microsoft.EntityFrameworkCore

@inject NavigationManager NavigationManager

@inject QUOTEDBEntities context

@renderMode InteractiveServer

<PageTitle>Trade Quotes</PageTitle>

<h3>Quotes</h3>

@if (quotes != null)

{

<table class="table">

<thead>

<tr>

<th>Symbol</th>

<th>Company Name</th>

</tr>

</thead>

<tbody>

@foreach (var quote in quotes)

{

<tr @onclick="@(() => BuyStock(quote))">

<td>@quote.Symbol</td>

<td>@quote.CompanyName</td>

</tr>

}

</tbody>

</table>

}

<button @onclick="LoadQuotes">Load Quotes</button>

@code {

private List<Quote> quotes;

private async Task LoadQuotes()

{

quotes = await context.Quotes.ToListAsync();

// StateHasChanged(); // May not be needed in most cases as Blazor handles this for events like button clicks.

}

void BuyStock(Quote quote)

{

// Logic to handle stock purchase

}

}

In this component, **@renderMode InteractiveServer** is specified at the top, instructing Blazor to render the component interactively on the server. This means that the component is initially rendered as static HTML, and once the browser establishes a connection back to the server, the component becomes interactive, allowing for dynamic updates and event handling.

It's important to note that **@renderMode InteractiveServer** is particularly useful for Blazor Server applications where you want to ensure that components are interactive after the initial static render and the establishment of the SignalR connection. This approach optimizes the initial load performance while ensuring interactivity is maintained.

### Notes

Well-it does not say it will continue to do this for me in future requests. Oh well. It just tells me how to do.

Bonus though, my list of quotes is now clickable—key is that line at top:  
  
**@renderMode InteractiveServer**

I am now ready to migrate the BuyForm from VB.NET to Blazor.

## ChatGTP Interaction 58

### Input

Assume for the following questions I am migrating a VB.NET WInForms app that uses Microsoft Entity Framework 6.4.4 to a new .NET 8 Blazor Server application using Microsoft EntityFrameworkCore 8.0.2. Please also pay attention to generating the right using statements in the generated Blazor components, and up top directives for an Interactive RenderMode for the server Blazor app.

### Output

Understood, you're migrating from a VB.NET WinForms application that uses Microsoft Entity Framework 6.4.4 to a .NET 8 Blazor Server application with Microsoft EntityFrameworkCore 8.0.2. I'll make sure to include the appropriate **using** statements in the generated Blazor components and the directive for an Interactive RenderMode (**@renderMode InteractiveServer**) for your server-side Blazor application. If you have any specific questions or need guidance on certain aspects of this migration, please let me know!

### Notes

I have added some more precise instructions for ChatGTP to follow, to try to avoid some annoyances in past interactions. Do not know how long this “context” will last (number of questions? Length of session? Session timeout with ChatGTP?).

## ChatGTP Interaction 59 – Direct Code Migration UI Tier

### Input

Given above, please migrate the following VB.NET Winform to a Blazor component:

"Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder

Imports System.Reflection.Emit

Imports System.Windows.Forms.VisualStyles.VisualStyleElement

'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

Public Class BuyForm

Dim symbol As String

Dim row As DataGridViewRow

Private Sub Form2\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Me.Visible = False

End Sub

Private Sub Button2\_Click(sender As Object, e As EventArgs) Handles Button2.Click

Me.Hide()

End Sub

Public Sub Form2\_Buy(theRow As DataGridViewRow, e As EventArgs)

Me.Show()

Me.BringToFront()

row = theRow

symbol = row.Cells("Symbol").Value

Label1.Text = "Select the Number of Shares to Buy for Symbol " + symbol

End Sub

Private Sub NumShares\_Validating(sender As Object, e As System.ComponentModel.CancelEventArgs) Handles NumShares.Validating

Dim input As String = NumShares.Text ' Get the input from the text box

Dim number As Integer ' Declare a variable to store the parsed integer

If Not Integer.TryParse(input, number) Then ' Try to parse the input as an integer

e.Cancel = True ' Cancel the validation if the input is not an integer

NumShares.BackColor = Color.Red ' Change the background color of the text box to red

MessageBox.Show("Please enter a valid integer.") ' Show an error message

Else

NumShares.BackColor = SystemColors.Window ' Restore the default background color of the text box

End If

End Sub

Private Sub Buy\_Click(sender As Object, e As EventArgs) Handles Button1.Click

Dim stock As New Stock

Dim sharestobuy As String = NumShares.Text ' Assume TextBox1 is your input control

stock.Buy(row, symbol, Integer.Parse(sharestobuy))

Me.NumShares.Text = ""

Me.Hide()

End Sub

Private Sub Label1\_Click(sender As Object, e As EventArgs) Handles Label1.Click

End Sub

End Class"

### Output

To migrate your VB.NET WinForms **BuyForm** to a Blazor component, we'll need to adapt the form's functionality to fit the Blazor component model. This includes converting event handlers to Blazor event callbacks, changing the way UI updates are handled, and managing component visibility. Here's how you can structure your Blazor component:

**BuyComponent.razor**

@page "/buy"

@using TradeBlazorApp.Data\_Classes

@using Microsoft.EntityFrameworkCore

@inject NavigationManager NavigationManager

@inject QUOTEDBEntities context

@renderMode InteractiveServer

<PageTitle>Buy Stock</PageTitle>

@if (isVisible)

{

<div>

<h3>Select the Number of Shares to Buy for Symbol @symbol</h3>

<EditForm Model="@this" OnValidSubmit="@BuyStock">

<DataAnnotationsValidator />

<ValidationSummary />

<div class="form-group">

<label for="numShares">Number of Shares:</label>

<InputNumber @bind-Value="numShares" class="form-control" id="numShares" />

</div>

<button type="submit" class="btn btn-primary">Buy</button>

</EditForm>

<button class="btn btn-secondary" @onclick="Hide">Cancel</button>

</div>

}

@code {

private string symbol;

private bool isVisible = false;

private int? numShares;

// Method to show the component with the selected stock symbol

public void Show(string selectedSymbol)

{

symbol = selectedSymbol;

isVisible = true;

}

// Method to hide the component

private void Hide()

{

isVisible = false;

}

// Method to handle stock purchase

private async Task BuyStock()

{

if (numShares.HasValue)

{

// Assume StockService is a service class you have to handle stock transactions

await StockService.BuyStock(symbol, numShares.Value);

numShares = null; // Reset after purchase

Hide(); // Hide the component after purchase

NavigationManager.NavigateTo("/"); // Optionally navigate to a different page

}

}

}

**Key Points:**

* **Page Directive**: **@page "/buy"** defines the route for this component.
* **Visibility**: The component's visibility is managed by the **isVisible** boolean variable. The form is only displayed when **isVisible** is **true**.
* **Data Binding**: The **<InputNumber>** component is used for binding the number of shares (**numShares**) to ensure that only numeric input is accepted.
* **Form Submission**: The form uses the **OnValidSubmit** event to handle stock purchases. The **BuyStock** method is called when the form is submitted with valid data.
* **Validation**: **DataAnnotationsValidator** and **ValidationSummary** are used for input validation. You would need to define validation rules in your model or use FluentValidation.
* **Navigation**: **NavigationManager** is used to navigate to other pages if needed after the purchase is made.

**Additional Considerations:**

* **StockService**: This example assumes you have a service class (**StockService**) to handle the logic for buying stocks. You would need to implement this service and register it in your DI container.
* **Component Invocation**: This component needs to be invoked from another part of your application, likely with a method to set the **symbol** and make the component visible. This could be from a list of stocks, where selecting a stock sets the **symbol** and calls **Show**.
* **Styling and Layout**: Add CSS classes as needed to style the component according to your application's design.

This component should give you a starting point for migrating your **BuyForm** functionality from WinForms to a Blazor component. Adjust the logic and UI as necessary to fit your application's requirements.

### Notes

First, notice with the targeted context in Interaction 58 above, I am now getting better migration of appropriate @using imports and page directive such as rendermode. That’s good!

Now creating Buy.Razor and pasting generated code into my new Blazor component.

Curious to see how easy/hard to integrate with my previously migrated Stock component (business layer class).

OK: I did some manual fixup in Stock.cs (my biz logic class):

1. Added an Interface IStockStockService
2. Changed the name from Stock.cs to StockService for implementation (to be clear, and not confuse with a possible DBEntity –it’s a SERVICE.
3. Fixup method declaration for Buy method implementation to:  
    public async Task<string> Buy(string stockSymbol, int quantity) (matching how Login is handled when calling a Biz Logic Service)
4. Added to Program.cs to Add the IStock services in builder.

I am also making some manual changes around isVisible and .Show() and .Hide() usage, per the BuyForm to Buy.Razor component. This is easy for me to do manually, now that ChatGTP has shown me so much/learning. It tried to carry these over in VB.NET WinForms app—but they are not proper for a Blazor/Web App. Easy for me to fix, so doing manually to get proper behavior (as ChatGTP above told me I would need to do):  
  
If LoggedIn: Can buy, view holdings, and sell. Else, need give user a message—MUST LOGIN FIRST (per Login.Razor).

## ChatGTP Interaction 60 - Output Code Optimization/Fixup

### Input

I want this to have a font color of Red: <label>@labelText</label>

### Output

To set the font color of the **<label>** to red in your Blazor component, you can use the **style** attribute directly within the **<label>** tag. Here's how you can do it:

<label style="color: red;">@labelText</label>

This inline style will apply the red color specifically to this label. If you're going to use this styling frequently throughout your application, consider defining a CSS class instead. For example, you might add the following class to your site's CSS file:

.red-label {

color: red;

}

Then, you can apply this class to your **<label>** like this:

<label class="red-label">@labelText</label>

This approach is cleaner and more maintainable, especially if you plan to reuse the same style in multiple places.

### Notes

Ok: I am now starting to adjust UI formatting. Basically, here, I used VB.NET WinForms MessageBoxes. I need to adjust to pure HTML. Using HTML “labels” in Blazor app, to represent those (modal) Message Boxes from VB.NET/WinForms.

I will take advice: use Styles to be “is cleaner and more maintainable, especially if you plan to reuse the same style in multiple places.”

Issue: I added to **MainLayout.Razor.css**. It did not work.  
  
I see in WWROOT: I have **another Style Sheet, app.css**. Will add there as well and retry.   
I wonder the difference between the two, and how they interact. NOT RESOLVED.  
  
Session Break, just for knowledge (do I need reset context with ChatGTP, after a 1 hour break)?

## ChatGTP Interaction 61

### Input

Do you have my context/instructions from last interaction?

### Output

I don't have access to our previous interactions or any context from them once the conversation ends or if you start a new session. Each session with me is stateless, meaning I don't retain memory of past conversations. How can I assist you today?

### Notes

This is key. Will continue to reset context. Choices:

* 1. Once per session: ChatGTP:  
       
     Further: A "session" with me lasts as long as our continuous interaction without a significant pause. If you close the chat window, refresh the page, or there's a long period of inactivity (usually around 30 minutes), the session will end, and any context from that session will be lost. If you start a new conversation, it will be considered a new session, and I won't have access to any information or context from previous sessions.
  2. With every question.

So—not sure how the **OpenAI API** handles this. But certainly useful to know, though it’s not precise.

I will move on and style my Pages later, and stick to functionality migrating forms, until then.

I am at a point now, where I need the “Buy” page to pickup on a **parameter** for the symbol to buy, from the Quotes.Razor redirect.

## ChatGTP Interaction 62

### Input

in Blazor, how to I pass a value between two pages?

### Output

In Blazor, there are multiple ways to pass data between pages:

1. **Using Singleton Services**: You can create a service class to hold the data you want to share between pages and register it as a Singleton in your application's service collection. This method ensures that the same instance of the service class is used throughout the application, allowing different components to access and modify the shared data. Inject this service into the components where you need access to the shared data​​.
2. **Component Parameters**: This method involves passing data directly to a component through parameters. You define public properties in the receiving component with the **[Parameter]** attribute and pass the data to these properties when using the component​​.
3. **Cascading Parameters**: For nested components or when you have a hierarchy of components, cascading parameters allow you to pass data down the component tree without having to pass the parameter through every level. You use the **<CascadingValue>** tag to provide the value at a higher level, and components further down can capture the value using the **[CascadingParameter]** attribute​​.
4. **Using Route Parameters**: Blazor allows you to pass parameters as part of the URL route. When configuring the **@page** directive, you can specify parameters within the route template. You can capture these parameters in the target component using properties with the **[Parameter]** attribute. Use the **NavigationManager** to navigate to the route with the parameter​​.

Each of these methods has its use cases depending on the structure of your application and the type of data you're passing. Singleton services and state containers are suitable for more complex objects or when multiple components, not necessarily in a direct parent-child relationship, need access to the same data. Component parameters and cascading parameters are more suited for direct parent-child component relationships. Route parameters are useful for passing data through navigation actions, especially when the data can be represented as simple types like strings or integers.

### Notes

## ChatGTP Interaction 63

### Input

Can you show me an example for Blazor .NET 8 server app, for passing parameters using Component Parameters when navigating to a new Blazor page?

### Output

To pass parameters between pages in a Blazor Server app using Component Parameters, you define public properties in your component with the **[Parameter]** attribute. Then, when you navigate to this component, you can pass the values as attributes in the component's tag. For example, if you have a **Title** parameter in your child component, you can set it from a parent component like so: **<ChildComponent Title="Your Title" />**. This allows you to pass data directly to the child component​​.

### Notes

Note enough info, need an example in code.

## ChatGTP Interaction 64

### Input

give me a code example?

### Output

For passing parameters between pages in a Blazor Server app using component parameters, define a property with the **[Parameter]** attribute in the receiving component, and pass the value when navigating. Here's a simplified example:

In the receiving component (**DetailPage.razor**):

@page "/detailpage/{Parameter}"

@code {

[Parameter]

public string Parameter { get; set; }

}

In the navigating component, use **NavigationManager** to navigate with a parameter:

@inject NavigationManager NavManager

<button @onclick="() => NavManager.NavigateTo($"/detailpage/{value}")">Go to Detail Page</button>

Replace **{value}** with the actual value you want to pass. For more details, visit the official documentation.

### Notes

Ok, am getting a 404 error, not sure my syntax is right.

## ChatGTP Interaction 65 - Output Code Optimization/Fixup

### Input

I get Error 404 page not found, when executing this and trying to pass a parameter named quote: NavigationManager.NavigateTo("/quotes/{quote}"); Fix?

### Output

To fix the Error 404, ensure your Blazor app has a corresponding route for **/quotes/{quote}**. If **quote** is a variable parameter, your route should be defined like **@page "/quotes/{parameter}"** in the Razor component handling this route. Also, when using **NavigationManager.NavigateTo**, do not include braces around the parameter. Instead, directly concatenate or interpolate the variable like so: **NavigationManager.NavigateTo($"/quotes/{quote}")**. Ensure **quote** has a value before navigating. For more details on routing in Blazor, you might find the official Blazor documentation helpful.

### Notes

Ok, changed the way I am calling to NavManager, to above. Let’s see when I run it.

I get to the receiving page without a 404, but parameter {quote} is always null.

After an hour here, and never getting anything but null in the receiving page for my complex object quote being passed as a parameter, I ran this:

## ChatGTP Interaction 66

### Input

show me an example with code in .NET 8 Blazor Server app of passing an object (not a string) as a parameter from source page to target page

### Output

In a .NET 8 Blazor Server app, to pass an object as a parameter between pages, you can use a cascading parameter or a service. For direct page-to-page navigation with complex objects, it's common to use a state management pattern or a scoped service to hold the object. The source page sets the object in the service, and the target page retrieves it. This approach avoids serializing complex objects in URLs, which is not recommended. For more details, refer to the [Blazor documentation](https://learn.microsoft.com/en-us/aspnet/core/blazor/).

### Notes

OK. Bad practice to try to pass complex objects as parameters. I will simply use my GlobalSettings class to set a public field theCurrentQuote and then retrieve it in the called page Buy.

It works, after some fixup per above, thru both the BusinessClass Stock, and GlobalSettings.

Very happy again. Failed experiment was trying to pass parameters between pages (complex object).

I also did some fixup on the Buy page, to handle “Cancel” button to return to main Home page.

Ready for the next Form – Holdings.

## ChatGTP Interaction 67 – Direct Code Migration UI Tier

### Input

Can you migrate this VB.NET WinForm to a .NET 8 Blazor Component please?

Target EntityFrameWorkCore 8.0.2

Imports System.Data.Entity ' Import Entity Framework namespace

Imports System.Data.Entity.Core

Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder

Imports System.Data.Entity.Migrations

Imports System.Data.SqlClient

Imports System.Runtime.ConstrainedExecution

Imports System.Runtime.Remoting.Contexts

'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

Public Class Holdings

Public Sub LoadGrid()

' Initialize the DbContext

Using holdingContext As New ACCOUNTDBEntities

' Query the database for all holdings

' Assuming "userid" is a variable containing the user's ID you're interested in

Dim parameter = New SqlParameter("@userId", GlobalSettings.theLoginAccount.UserID)

' Correct the DbSet if necessary. Here, it's assumed to be HOLDINGs

' Execute the SQL query using a parameterized query for safety

Dim query = holdingContext.HOLDINGs.SqlQuery("SELECT \* FROM dbo.HOLDING WHERE USERID = @userId", parameter)

' Materialize the results into a List (or another suitable collection)

Dim result As List(Of HOLDING) = query.ToList()

DataGridView1.DataSource = result

' Set the DataGridView's data source to the DbSet of Employees

'DataGridView1.DataSource = (holdingContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.HOLDING WHERE USERID = @userId", parameter).ToList())

'DataGridView1.Columns("Symbol").DisplayIndex = 0

' DataGridView1.Columns("CompanyName").DisplayIndex = 1

End Using

End Sub

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

If GlobalSettings.LoggedIn Then

LoadGrid()

Me.Label1.Text = "Hello " + GlobalSettings.theLoginAccount.Name + "! These are Your Current Holdings, Select one to Sell."

Else

MsgBox("You Must Login First, to see your Holdings and Sell Stock!")

End If

End Sub

Public Sub New()

' This call is required by the designer.

InitializeComponent()

' Add any initialization after the InitializeComponent() call.

AddHandler Me.FormClosing, AddressOf Form1\_FormClosing

End Sub

Private Sub DataGridView1\_CellClick(sender As Object, e As DataGridViewCellEventArgs) Handles DataGridView1.CellClick

If GlobalSettings.LoggedIn Then

' Check if the click is on a row, not the column header

If e.RowIndex >= 0 Then

' Access the clicked row

Dim row As DataGridViewRow = DataGridView1.Rows(e.RowIndex)

SellForm.Show()

SellForm.SellForm\_Sell(Me, row, e)

End If

Else

MsgBox("You Must Login Before Selecting a Stock to Buy")

End If

End Sub

Private Sub Form1\_FormClosing(sender As Object, e As FormClosingEventArgs)

' Check your condition here. This is just an example condition.

Dim allowClose As Boolean = GlobalSettings.LoggedIn ' Assume this is your condition

If allowClose Then

MessageBox.Show("Condition not met, form will not close.")

e.Cancel = True ' Prevent the form from closing

End If

End Sub

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click

Me.Hide()

End Sub

End Class

### Output

To migrate the VB.NET WinForms code to a .NET 8 Blazor component, you'll need to adapt the data loading logic to Blazor's component lifecycle methods, like **OnInitializedAsync**, and use EF Core for database operations. The UI will be constructed with Razor syntax, replacing WinForms controls like **DataGridView** with HTML tables and data binding. User interactions, like button clicks and cell clicks, will be handled with event bindings in Razor. For authentication checks, Blazor's built-in authorization mechanisms can be utilized. For a detailed guide, consider the [Blazor documentation](https://learn.microsoft.com/en-us/aspnet/core/blazor/).

### Notes

**WOW! ChatGTP 4.0 has stopped doing any code migrations for me! As of Thursday 2/22, 8:00 AM!**

**I have tried and tried, even past queries that worked, now I just get this boilerplate!  
  
Switching to Bing which is using ChatGTP 4/OpenAI**

**Bing.com only allows for 2000 characters at a time! Not enough for my full classes.**

**OK: ChatGTP 4.0 is Working again, and generating code. Mystery as to why it stopped working, then started again (after 8 tries).**

## ChatGTP Interaction 68 – Direct Code Migration UI Tier

### Input

Please migrate this VB.NET Winforms Code to Blazor .NET Server app:

Imports System.Data.Entity ' Import Entity Framework namespace

Imports System.Data.Entity.Core

Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder

Imports System.Data.Entity.Migrations

Imports System.Data.SqlClient

Imports System.Runtime.ConstrainedExecution

Imports System.Runtime.Remoting.Contexts

'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

Public Class Holdings

Public Sub LoadGrid()

' Initialize the DbContext

Using holdingContext As New ACCOUNTDBEntities

' Query the database for all holdings

' Assuming "userid" is a variable containing the user's ID you're interested in

Dim parameter = New SqlParameter("@userId", GlobalSettings.theLoginAccount.UserID)

' Correct the DbSet if necessary. Here, it's assumed to be HOLDINGs

' Execute the SQL query using a parameterized query for safety

Dim query = holdingContext.HOLDINGs.SqlQuery("SELECT \* FROM dbo.HOLDING WHERE USERID = @userId", parameter)

' Materialize the results into a List (or another suitable collection)

Dim result As List(Of HOLDING) = query.ToList()

DataGridView1.DataSource = result

' Set the DataGridView's data source to the DbSet of Employees

'DataGridView1.DataSource = (holdingContext.ACCOUNTPROFILEs.SqlQuery("SELECT \* FROM dbo.HOLDING WHERE USERID = @userId", parameter).ToList())

'DataGridView1.Columns("Symbol").DisplayIndex = 0

' DataGridView1.Columns("CompanyName").DisplayIndex = 1

End Using

End Sub

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

If GlobalSettings.LoggedIn Then

LoadGrid()

Me.Label1.Text = "Hello " + GlobalSettings.theLoginAccount.Name + "! These are Your Current Holdings, Select one to Sell."

Else

MsgBox("You Must Login First, to see your Holdings and Sell Stock!")

End If

End Sub

Public Sub New()

' This call is required by the designer.

InitializeComponent()

' Add any initialization after the InitializeComponent() call.

AddHandler Me.FormClosing, AddressOf Form1\_FormClosing

End Sub

Private Sub DataGridView1\_CellClick(sender As Object, e As DataGridViewCellEventArgs) Handles DataGridView1.CellClick

If GlobalSettings.LoggedIn Then

' Check if the click is on a row, not the column header

If e.RowIndex >= 0 Then

' Access the clicked row

Dim row As DataGridViewRow = DataGridView1.Rows(e.RowIndex)

SellForm.Show()

SellForm.SellForm\_Sell(Me, row, e)

End If

Else

MsgBox("You Must Login Before Selecting a Stock to Buy")

End If

End Sub

Private Sub Form1\_FormClosing(sender As Object, e As FormClosingEventArgs)

' Check your condition here. This is just an example condition.

Dim allowClose As Boolean = GlobalSettings.LoggedIn ' Assume this is your condition

If allowClose Then

MessageBox.Show("Condition not met, form will not close.")

e.Cancel = True ' Prevent the form from closing

End If

End Sub

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click

Me.Hide()

End Sub

End Class

### Output

Migrating your VB.NET WinForms code, which interacts with a database using Entity Framework, to a Blazor Server application involves several key steps. We'll need to adapt your database access logic, event handlers, and UI components to the Blazor framework. Below is a guide to help you through this process:

**1. Setup the Blazor Server Project**

First, ensure you have a Blazor Server project set up in Visual Studio. In your project, you'll need to include the Entity Framework package that corresponds to the type of database you are using (e.g., **Microsoft.EntityFrameworkCore.SqlServer** for SQL Server).

**2. Data Context and Models**

Assuming you have an existing Entity Framework data context (**ACCOUNTDBEntities**) and entity models (e.g., **HOLDING**), ensure they are included in your Blazor project. If you're using code-first migrations, your models and data context should be compatible with Blazor with little to no change.

**3. Service for Database Operations**

Create a service in your Blazor project to handle database operations. This service will encapsulate the logic for fetching holdings from the database.

public class HoldingsService

{

private readonly ACCOUNTDBEntities \_context;

public HoldingsService(ACCOUNTDBEntities context)

{

\_context = context;

}

public async Task<List<HOLDING>> GetHoldingsAsync(int userId)

{

return await \_context.HOLDINGs

.Where(h => h.UserID == userId)

.ToListAsync();

}

}

Register this service in **Startup.cs**:

services.AddDbContext<ACCOUNTDBEntities>(options =>

options.UseSqlServer(Configuration.GetConnectionString("YourConnectionString")));

services.AddScoped<HoldingsService>();

**4. Holdings Page**

Create a Razor component or page for displaying holdings. This component will retrieve and display the holdings using the **HoldingsService**.

@page "/holdings"

@inject HoldingsService HoldingsService

@inject GlobalStateService GlobalState // Assuming you have a service like in the previous example

@if (holdings == null)

{

<p><em>Loading...</em></p>

}

else

{

<table class="table">

<thead>

<tr>

<th>Symbol</th>

<th>CompanyName</th>

<!-- Add other columns as needed -->

</tr>

</thead>

<tbody>

@foreach (var holding in holdings)

{

<tr @onclick="@(() => SelectHolding(holding))">

<td>@holding.Symbol</td>

<td>@holding.CompanyName</td>

<!-- Add other cells as needed -->

</tr>

}

</tbody>

</table>

}

@code {

private List<HOLDING> holdings;

protected override async Task OnInitializedAsync()

{

if (GlobalState.LoggedIn)

{

holdings = await HoldingsService.GetHoldingsAsync(GlobalState.theLoginAccount.UserID);

}

else

{

// Handle not logged in state, possibly redirect to login

}

}

void SelectHolding(HOLDING holding)

{

// Handle the selection, possibly navigate to another page or open a modal

}

}

**5. Handling Events and Navigation**

Blazor handles user interactions through event bindings. In the example above, the **@onclick** directive on the **<tr>** element is used to handle row clicks. You can adapt other event handlers in a similar manner.

**6. Global State and Authentication**

Your original code uses a **GlobalSettings** static class to manage global state. In Blazor, global state can be managed using services (like **GlobalStateService** mentioned in the previous example) or Blazor's built-in authentication and authorization mechanisms.

**7. UI Feedback and Messages**

Instead of **MsgBox** for feedback, consider using Blazor components, JavaScript interop, or a Blazor library for toast notifications/modal dialogs to provide user feedback.

**8. Form Closing Logic**

Blazor applications are single-page applications (SPAs), so the concept of form closing doesn't directly apply. However, you can handle component disposal with the **@implements IDisposable** interface and the **Dispose** method if you need to perform any cleanup.

This guide should help you get started with migrating your WinForms code to Blazor. Remember, Blazor uses a different lifecycle and event handling model, so some adaptation of your logic will be necessary.

### Notes

Creating new Holdings Page now, and copying code.

With some fixup, it is working! But displaying ALL Holdings, as generated by ChatGTP. Need to fix lookup to retrieve JUST holdings for logged in customer.

Fixed now. Onto Next-to-Last UI/Form: the SellForm, from VB.NET WinForms SellForm.

## ChatGTP Interaction 69 – Direct Code Migration UI Tier

### Input

I am migrating this VB.NET WinForm to .NET 8 Blazor Application. Do note, my VB.NET Winform code below, uses Microsoft EntityFramework 6.4.4, and my new Blazor Server App uses Microsoft EntityFrameworkCore 8.0.2:

Please generate my Blazor code from the following VB.NET WinForms code:

Imports System.Data.Entity.Core.Common.CommandTrees.ExpressionBuilder

Imports System.Reflection.Emit

Imports System.Windows.Forms.VisualStyles.VisualStyleElement

'This Form Uses ADO.NET Entity Framework, a more "factored" approach toi database access with auto-built model classes

Public Class SellForm

Dim quantityHeld As Double

Dim symbol As String

Dim row As DataGridViewRow

Dim theHoldingForm As Holdings

Private Sub SellForm\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Me.Visible = False

End Sub

Public Sub SellForm\_Sell(thisHoldingForm As Holdings, theRow As DataGridViewRow, e As EventArgs)

Me.Show()

Me.BringToFront()

row = theRow

symbol = row.Cells("Quote\_Symbol").Value

quantityHeld = row.Cells("Quantity").Value

Label1.Text = "Select the Number of Shares to Sell for Symbol " + symbol

theHoldingForm = thisHoldingForm

End Sub

Private Sub CancelButton\_Click(sender As Object, e As EventArgs) Handles CancelButton.Click

Me.Hide()

End Sub

Private Sub SharesSell\_Validating(sender As Object, e As System.ComponentModel.CancelEventArgs) Handles SharesSell.Validating

Dim input As String = SharesSell.Text ' Get the input from the text box

Dim number As Integer ' Declare a variable to store the parsed integer

If Not Integer.TryParse(input, number) Then ' Try to parse the input as an integer

e.Cancel = True ' Cancel the validation if the input is not an integer

SharesSell.BackColor = Color.Red ' Change the background color of the text box to red

MessageBox.Show("Please enter a valid integer.") ' Show an error message

Else

If number > quantityHeld Then

e.Cancel = True ' Cancel the validation if the input is not an integer

SharesSell.BackColor = Color.Red ' Change the background color of the text box to red

MessageBox.Show("You only Hold " + quantityHeld.ToString())

Else

SharesSell.BackColor = SystemColors.Window ' Restore the default background color of the text box

End If

End If

End Sub

Private Sub SellButton\_Click(sender As Object, e As EventArgs) Handles SellButton.Click

Dim stock As New Stock

Dim sharestosell As String = SharesSell.Text

stock.Sell(row, symbol, Integer.Parse(sharestosell))

Me.SharesSell.Text = ""

Me.Hide()

theHoldingForm.LoadGrid()

End Sub

End Class

### Output

Migrating your VB.NET WinForms **SellForm** to a Blazor Server application involves several steps, including rethinking the user interface and how events are handled in a web environment. Here's an outline of how you can approach this, along with some sample Blazor code:

**Step 1: Define the Razor Component**

Create a new Razor component in your Blazor Server project for the sell functionality. This component will handle the user input for selling shares.

SellForm.razor

@page "/sellform"

@inject HoldingsService HoldingsService

<h3>Select the Number of Shares to Sell for Symbol @Symbol</h3>

@if (IsVisible)

{

<EditForm Model="@this" OnValidSubmit="HandleValidSubmit">

<DataAnnotationsValidator />

<div class="form-group">

<label for="SharesToSell">Shares to Sell:</label>

<InputNumber @bind-Value="SharesToSell" class="form-control" id="SharesToSell" />

<ValidationMessage For="@(() => SharesToSell)" />

</div>

<button type="submit" class="btn btn-primary">Sell</button>

<button class="btn btn-secondary" @onclick="Cancel">Cancel</button>

</EditForm>

}

@code {

[Parameter] public string Symbol { get; set; }

[Parameter] public double QuantityHeld { get; set; }

public bool IsVisible { get; set; } = false;

public int SharesToSell { get; set; }

protected override void OnParametersSet()

{

IsVisible = !string.IsNullOrEmpty(Symbol);

}

private void HandleValidSubmit()

{

if (SharesToSell > QuantityHeld)

{

// Show an error message. Consider using a modal or a toast notification.

}

else

{

// Call the service to sell the shares

HoldingsService.SellShares(Symbol, SharesToSell);

// Reset the form and hide it

ResetAndHide();

}

}

private void Cancel()

{

ResetAndHide();

}

private void ResetAndHide()

{

SharesToSell = 0;

IsVisible = false;

}

}

**Step 2: Create a Service for Operations**

Create a service to handle the sell operation. This service will interact with your Entity Framework Core context to perform database operations.

public class HoldingsService

{

private readonly YourDbContext \_context;

public HoldingsService(YourDbContext context)

{

\_context = context;

}

public async Task SellShares(string symbol, int sharesToSell)

{

// Implement the logic to sell shares

// This might involve updating the database to reflect the sale

await \_context.SaveChangesAsync();

}

}

**Step 3: Handle Validation**

In the Blazor component, **DataAnnotationsValidator** and **ValidationMessage** are used for validation. You can define validation rules directly in your model or use **FluentValidation**.

**Step 4: Showing and Hiding the Component**

The **IsVisible** property controls the visibility of the form. You can toggle this property based on user actions elsewhere in your application, such as selecting a row in a grid.

**Step 5: User Feedback**

For showing messages (like the **MessageBox.Show** in WinForms), consider using a JavaScript interop call to display browser alerts, or use a component library that provides modal dialogs or toast notifications for a more integrated user experience.

**Integrating with Other Components**

You might need to integrate this component with other parts of your Blazor application, such as a grid displaying holdings. You can use EventCallback to notify parent components of changes or actions, like a successful sell operation.

**Summary**

This example outlines a basic approach to migrate your WinForms sell form to a Blazor component. Depending on your application's complexity and requirements, you might need to adjust this example, particularly around data handling, validation, and user feedback.

### Notes

OK: Pasting code in after creating my Sell.Blazor component as new Blazor page.

About 1.5 hours of manual fixup, required: working now. Almost all fixup, fairly obvious, to match actual new C# names of classes, but also some required to use new EntityFrameworkCore 8.0.2 vs. VB.NET using EntityFramework 6.4.4.  
  
And of course, navigation, messages, and various basic UI elements. All this, UI stuff and tying exactly into generated Business Services and Data Services classes is really manual. But works, after fixup.

And learning also, on the fly, a bit, still especially Blazor concepts, and EntityFrameworkCore 8.0.2 for .NET 8. All new from MS and new to me.   
  
Now onto the LAST VB.NET FORM (AccountForm). Let’s go.

## ChatGTP Interaction 70 – Direct Code Migration UI Tier

### Input

Please migrate this VB.NET WinForm from VS 2022, to a .NET 8 Blazor Server App component:

Imports System.Data.Entity.Core

Imports System.Data.SqlClient

Imports System.Configuration

' This VB Form Is Built Using Pure ADO.NET, with No Model Classes for Database or use of the Entity Framework, or Business Logic Classes

' It illustrates a more manual/spagetti approach in a Win Forms App, to test migration via ChatGPT to Blazor/Razor and C#

Public Class AccountForm

Dim connectionString As String

Private Sub AccountForm\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

connectionString = "Data Source=GREG-SURFACE4\SQLEXPRESS;Initial Catalog=ACCOUNTDB;Integrated Security=True;TrustServerCertificate=True;"

Using connection As New SqlConnection(connectionString)

' Open the connection

connection.Open()

' Create a SqlCommand to execute the query

Using command As New SqlCommand("Select FULLNAME, ADDRESS, EMAIL, CREDITCARD from dbo.ACCOUNTPROFILE where USERID = '" + GlobalSettings.theLoginAccount.UserID + "'", connection)

' Execute the command and process the results

Using reader As SqlDataReader = command.ExecuteReader()

While reader.Read()

' Process each row. For example, print the first column value

TextBoxFullName.Text = reader(0).ToString()

TextBoxAddress.Text = reader(1).ToString()

TextBoxEmail.Text = reader(2).ToString()

TextBoxCreditCard.Text = reader(3).ToString()

End While

End Using

End Using

End Using

End Sub

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click

Try

Using connection As New SqlConnection(connectionString)

' Open the connection

connection.Open()

Dim updateSql As String = "UPDATE ACCOUNTPROFILE SET ADDRESS =@Address, FULLNAME = @FullName, EMAIL = @Email, CREDITCARD = @CreditCard WHERE USERID = @Id"

Using command As New SqlCommand(updateSql, connection)

' Add parameter to the command to prevent SQL injection

command.Parameters.AddWithValue("@Id", GlobalSettings.theLoginAccount.UserID)

command.Parameters.AddWithValue("@Address", TextBoxAddress.Text)

command.Parameters.AddWithValue("@FullName", TextBoxFullName.Text)

command.Parameters.AddWithValue("@CreditCard", TextBoxCreditCard.Text)

command.Parameters.AddWithValue("@Email", TextBoxEmail.Text)

Dim affectedRows As Integer = command.ExecuteNonQuery()

MsgBox("Your Information Has Been Updated!")

End Using

End Using

Catch ex As DataException

' Handle database-related exceptions

MessageBox.Show("Database error: " & ex.Message)

Catch ex As ApplicationException

' Handle application-specific exceptions

MessageBox.Show("Application error: " & ex.Message)

Catch ex As Exception

' Handle unexpected exceptions

MessageBox.Show("An unexpected error occurred: " & ex.Message)

End Try

Me.Hide()

End Sub

Private Sub Button2\_Click(sender As Object, e As EventArgs) Handles Button2.Click

Me.Hide()

End Sub

End Class

### Output

Migrating your VB.NET WinForms **AccountForm** to a .NET 8 Blazor Server component involves several key changes. You'll move from an event-driven model to a component-based model, replace direct ADO.NET calls with Entity Framework Core for database operations, and adapt the UI from WinForms to Razor syntax. Given the manual approach of your ADO.NET usage and the absence of model classes, we'll need to introduce some structure for the Blazor version.

**Step 1: Setup Entity Framework Core**

First, ensure Entity Framework Core is set up in your project. This involves creating a **DbContext** and defining your model. Since your original code doesn't use EF models, we'll need to define a simple model based on your SQL query.

AccountProfile Model

public class AccountProfile

{

public string UserId { get; set; } // Adjust types as needed

public string FullName { get; set; }

public string Address { get; set; }

public string Email { get; set; }

public string CreditCard { get; set; }

}

Application DbContext

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext

{

public DbSet<AccountProfile> AccountProfiles { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{ }

}

Register **ApplicationDbContext** in your **Startup.cs** or **Program.cs**, depending on your project setup:

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

**Step 2: Create the Blazor Component**

Next, create a Blazor component for the account form. This component will handle data loading and updating, similar to your WinForms code but adapted to the Blazor lifecycle.

AccountForm.razor

@page "/accountform"

@using YourNamespace.Data // Update with your actual namespace

@inject ApplicationDbContext DbContext

@if (accountProfile != null)

{

<EditForm Model="@accountProfile" OnValidSubmit="HandleValidSubmit">

<DataAnnotationsValidator />

<div class="form-group">

<label for="FullName">Full Name:</label>

<InputText id="FullName" class="form-control" @bind-Value="accountProfile.FullName" />

</div>

<div class="form-group">

<label for="Address">Address:</label>

<InputText id="Address" class="form-control" @bind-Value="accountProfile.Address" />

</div>

<div class="form-group">

<label for="Email">Email:</label>

<InputText id="Email" class="form-control" @bind-Value="accountProfile.Email" />

</div>

<div class="form-group">

<label for="CreditCard">Credit Card:</label>

<InputText id="CreditCard" class="form-control" @bind-Value="accountProfile.CreditCard" />

</div>

<button type="submit" class="btn btn-success">Update</button>

</EditForm>

}

else

{

<p><em>Loading...</em></p>

}

@code {

private AccountProfile accountProfile;

protected override async Task OnInitializedAsync()

{

accountProfile = await DbContext.AccountProfiles

.FirstOrDefaultAsync(a => a.UserId == GlobalSettings.TheLoginAccount.UserId); // Adapt for your user identification logic

}

private async Task HandleValidSubmit()

{

// Update logic

DbContext.AccountProfiles.Update(accountProfile);

await DbContext.SaveChangesAsync();

// Show a success message or navigate away

}

}

**Step 3: Handling User Identification**

Since **GlobalSettings.TheLoginAccount.UserId** is used to identify the user, ensure you have a similar mechanism in your Blazor app to retrieve the current user's ID. This could involve authentication and user management, which is typically handled via ASP.NET Core Identity in Blazor applications.

**Step 4: Database Connection Configuration**

Make sure your database connection string is configured in your Blazor app's **appsettings.json**:

{

"ConnectionStrings": {

"DefaultConnection": "Data Source=GREG-SURFACE4\\SQLEXPRESS;Initial Catalog=ACCOUNTDB;Integrated Security=True;TrustServerCertificate=True;"

}

}

**Step 5: UI Considerations and Validation**

The Blazor component uses **EditForm**, **InputText**, and other form components for data binding and validation. Adjust the UI according to your application's needs, and consider adding more sophisticated validation as required.

**Summary**

This migration outlines converting a form that directly uses ADO.NET for database operations into a Blazor component using Entity Framework Core. It introduces structure and leverages EF Core for data access, improving maintainability and scalability.

### Notes

This is an interesting last VB.NET Form: because I was using straight ADO.NET vs. EntityFramework 6.4.4. As an additional experiment for the project. Not all Forms the same, or factored for EntityFramework (this being the one Form exception to that, for illustrative purposes, test ChatGTP).  
  
**Note that ChatGTP 4 remembered from earlier context: I also want to move to EntityFrameworkCore 8.0.2**  
Cool!

Starting now: note also most of these steps have been completed, as rest of app is built now, save for this form. So should be easier. Creating and pasting code into AccountForm.Razor.

* 1. As before, must fixup my using statements and dependency injections, and rendor mode at top of page, manually.
  2. Just copying from Login.Razor, since it also exclusively deals with an AccountProfile/Account.

Adding this to my AccountCurrent Interface and Implementation as GetAccountDetailsAsync:  
  
Task<AccountProfile> GetAccountDetailsAsync(string userId, string password);

Done!

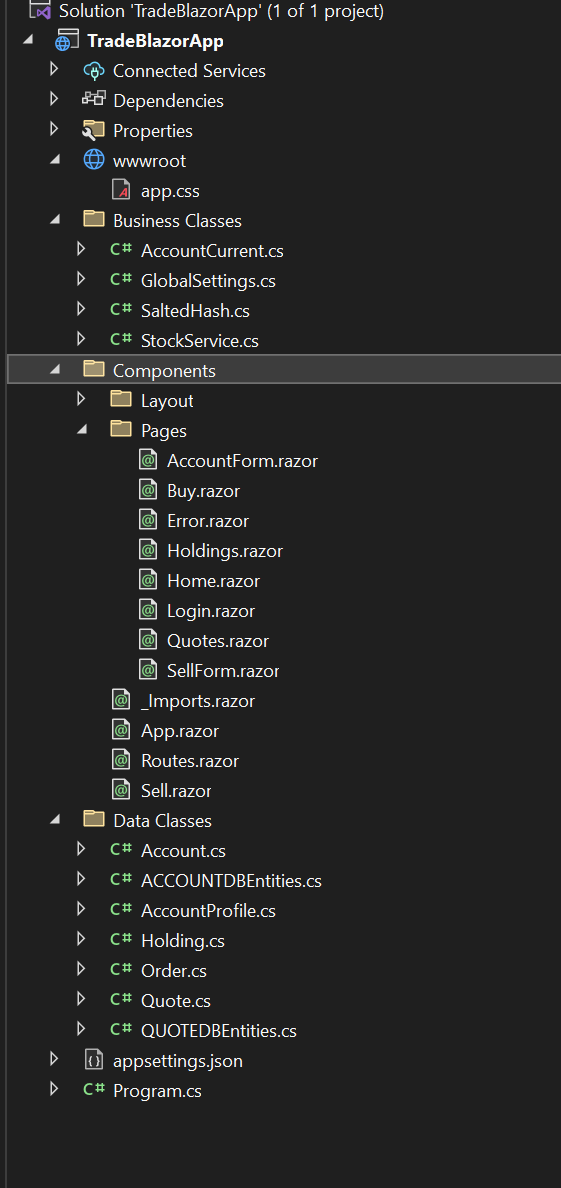
# Phase 6 – Test/Debug

## Final Testing Stage Notes

**Big Note: Somewhere along the line, ChatGTP in the Sell Function of StockService as migrated, inserted a using phrase with \_quote dbcontext contained.  
  
THIS WAS BAD. As, .NET, it turns out, then disposes this global class variable, on exit (async dispose).**

This cost me 3 hours to try to figure out why I kept getting runtime exception on sell, after trying to sell a second holding “Cannot Access a Disposed Object”  
  
I figured it out, and removed the using. I also purposefully created a new Instance of TradeService in the Razor Form itself, versus a static async call.

But I also found ChatGTP, generated a couple of places with dbContext within Using statements, and found this troublesome, as it meant .NET would dispose of these objects after existing the using statement, and since these instances were used multiple times (depending on how user interacts with a form, and the Async nature of Blazor and EntityFrameworkCore, I would get “Cannot access a Disposes Object”: especially on my sell form. Took awhile to figure out (an hour): then fixed by removing the “Using” block.  
  
ChatGTP was pretty useless here: I just figured it out after debugging sessions for awhile. All Good.  
  
All UI BASIC FORMS MIGRATED NOW!  
  
All seems test out ok.  
  
On to making the UI formatted more like the VB App.  
  
**But APP 100% FUNCTIONALLY IS MIGRATED NOW save for some navigation, formatting of UI.**

**Here is my Structure Now, All Forms Migrated to Blazor Components (aka Pages):  
  
**

Now on to UI Optimizations!

# Phase 7 Stylizing the UI Pages to Mimic Look and Feel of VB.NET Win Forms

Here I will use ChatGTP to try to make the UI more like the VB UI in terms of colors, fonts, background images, formatting. **All of this is “buried” in VB.NET property sheets, so while ChatGTP can help me, it will be fairly manual as it cannot read directly these property sheets. Also, as a Web app, I want to use StyleSheets to keep the formatting clean/maintainable.**

## ChatGTP Interaction 71 – Formatting Generation HTML/CSS UI Tier

### Input

can you build me a sample CSS style sheet that I can use to: 1) Set background color to Web color "DarkSlateGray" 2) Fronts for labels and inputs to Microsoft Sans Serif, 12pt 3) The use of a background image on some pages named "AboutBackground.jpg"

### Output

Certainly! Below is a sample CSS stylesheet that matches your specifications. You can add this CSS to your project's stylesheet file and link it to your HTML documents. This CSS includes:

1. Setting the background color to "DarkSlateGray" for the body of your pages.
2. Setting the font family to "Microsoft Sans Serif" and the font size to 12pt for all labels and input elements.
3. Adding a background image named "AboutBackground.jpg" for a specific class, which you can apply to any page or page section you want the background image to appear on.

/\* Sets the background color for the entire page \*/

body {

background-color: DarkSlateGray;

}

/\* Sets the font for labels and inputs to Microsoft Sans Serif, 12pt \*/

label, input {

font-family: 'Microsoft Sans Serif', sans-serif;

font-size: 12pt;

}

/\* Class to add a background image to specific pages or sections \*/

.about-background {

background-image: url('AboutBackground.jpg');

background-size: cover; /\* Cover the entire page or section \*/

background-position: center; /\* Center the background image \*/

}

To use the **.about-background** class, you would add it to the HTML element representing the page or section where you want the background image to appear. For example:

<div class="about-background">

<!-- Content for the section with the background image goes here -->

</div>

Remember to ensure the path to "AboutBackground.jpg" is correct based on your project's directory structure. If "Microsoft Sans Serif" is not available, browsers will fall back to a generic sans-serif font.

### Notes

Cool! But I had a devil of a time figuring out which .CSS file to place these elements into. Two choices:

1. App.css in the WWWRoot:
2. MainLayout.css in the app/components project element

**Answer, for now, is App.Css.**  
But found VS 2022/Blazor app was not picking up newly added elements UNTIL is started doing CTRL-F5 in browser, hard refresh.

## ChatGTP Interaction 72 - Formatting Generation HTML/CSS UI Tier

### Input

If I have the following HTML, how do I align the elements within a table?

<div>

<label class="white-label" for="userId">User ID:</label>

<InputText class="custom-input" id="userId" @bind-Value="loginModel.UserId" />

</div>

<div>

<label class="white-label" for="password">Password:</label>

<InputText id="password" @bind-Value="loginModel.Password" type="password" />

</div>

### Output

To align elements within a table structure, you first need to wrap your existing elements in a **<table>** layout. However, since you're using Blazor with components like **InputText**, it's important to maintain the structure that Blazor expects for data binding and event handling. Here's how you can organize your elements into a table structure while keeping your Blazor components intact:

<table>

<tr>

<td>

<label class="white-label" for="userId">User ID:</label>

</td>

<td>

<InputText class="custom-input" id="userId" @bind-Value="loginModel.UserId" />

</td>

</tr>

<tr>

<td>

<label class="white-label" for="password">Password:</label>

</td>

<td>

<InputText class="custom-input" id="password" @bind-Value="loginModel.Password" type="password" />

</td>

</tr>

</table>

In this structure:

* Each **div** is replaced by a **tr** (table row) element to denote each row of the table.
* The **label** and **InputText** components for each field are wrapped in **td** (table data) elements, which represent table cells.

To further align and style the table, you can add CSS. For instance, if you want to ensure the labels and input fields are aligned properly and have some spacing, you can add the following CSS:

table {

width: 100%; /\* Adjust based on your layout needs \*/

border-collapse: collapse; /\* Optional: removes the space between the borders of adjacent cells \*/

}

td {

padding: 8px; /\* Adds spacing inside each cell \*/

vertical-align: top; /\* Aligns content to the top of the cell \*/

}

.label {

white-space: nowrap; /\* Prevents the label from wrapping and keeps it on a single line \*/

}

.custom-input {

width: 100%; /\* Makes the input take up the full width of its cell \*/

}

You can add this CSS to your global stylesheet or within a **<style>** tag in the same component. Adjust the styles as needed to fit your design requirements.

### Notes

Impressive it does still know this is for a Blazor app, so formatted accordingly the generated HTML/advice.

## ChatGTP Interaction 73 - Formatting Generation HTML/CSS UI Tier

### Input

User

I have the following table (Blazor Component), and I want better formatting, including inn/outer borders as well as font size 12 and font color white. Can you help?

<table class="table">

<thead>

<tr>

<th>Symbol</th>

<th>Company Name</th>

<th>Price</th>

<th>Open</th>

<th>Volume</th>

</tr>

</thead>

<tbody>

@foreach (var quote in quotes)

{

<tr @onclick="@(() => BuyStock(quote))">

<td>@quote.Symbol</td>

<td>@quote.CompanyName</td>

<td>@quote.Price.ToString()</td>

<td>@quote.Open1.ToString()</td>

<td>@quote.Volume.ToString()</td>

</tr>

}

</tbody>

</table>

### Output

To enhance the formatting of your Blazor table component with inner/outer borders, a font size of 12 pixels, and a font color of white, you can create a custom CSS class or modify the existing **.table** class if it's not being used elsewhere for different styles. Here's an example of how you can define such a class in your CSS:

.custom-table {

border-collapse: collapse; /\* Collapses border to a single border for inner and outer borders \*/

width: 100%; /\* Optional: Adjusts the table width as needed \*/

font-size: 12px; /\* Sets the font size to 12 pixels \*/

color: white; /\* Sets the font color to white \*/

}

.custom-table th, .custom-table td {

border: 1px solid #ddd; /\* Sets the border color and width for table cells \*/

padding: 8px; /\* Adds padding inside table cells \*/

}

.custom-table th {

background-color: #04AA6D; /\* Optional: Sets a background color for the header \*/

color: white; /\* Ensures that the header text is white \*/

}

In your Blazor component, you should replace the **class="table"** with **class="custom-table"** to apply these styles:

<table class="custom-table">

<thead>

<tr>

<th>Symbol</th>

<th>Company Name</th>

<th>Price</th>

<th>Open</th>

<th>Volume</th>

</tr>

</thead>

<tbody>

@foreach (var quote in quotes)

{

<tr @onclick="@(() => BuyStock(quote))">

<td>@quote.Symbol</td>

<td>@quote.CompanyName</td>

<td>@quote.Price.ToString()</td>

<td>@quote.Open1.ToString()</td>

<td>@quote.Volume.ToString()</td>

</tr>

}

</tbody>

</table>

Ensure you include or link to the CSS file containing these styles in your Blazor app. You might need to adjust the colors and paddings to match your app's theme and requirements. The example assumes a dark theme (given the white font color), so you might want to adjust the header background and border colors as needed.

### Notes

Cool again, working on.

## ChatGTP Interaction 74 - Formatting Generation HTML/CSS UI Tier

### Input

In the following Blazor Code, I would like the buttons laid out vertically each in a table row, and the table centered in the page:

@page "/"

@rendermode InteractiveServer

@using TradeBlazorApp.Business\_Classes

@inject NavigationManager NavigationManager

<body style="background-image: url('AboutBackground.jpg'); background-size: cover; background-position: center;">

<!-- Page content goes here -->

<PageTitle>Trade Home</PageTitle>

<!--I had to add rendermode InteractiveServer line manually, else no events fire-->

<!--This entire page was first generated by CHatGTP by migrating the MainNav form from VB.NET WIn Forms, then manually fixed up bellow where noted-->

<label class="red-label">@labelText</label>

<div>

<button @onclick="ShowQuotes">Quotes</button> <!--I Changed the name here from Form1 to Quotes-->

<button @onclick="ToggleLogin">Toggle Login</button>

<button @onclick="ShowHoldings">Holdings</button>

<button @onclick="ShowAccountForm">Account</button>

</div>

</body>

@code {

// Your code logic remains the same here

string labelText;

public void ShowQuotes() //I changed from ShowForm1 to ShowQuotes.

{

NavigationManager.NavigateTo("/quotes");

//I Changed the name here from form1 to quotes

}

private void ToggleLogin()

{

if (GlobalSettings.LoggedIn)

{

GlobalSettings.LoggedIn = false;

// Add logic to reset login state as needed

}

else

{

NavigationManager.NavigateTo("/login");

}

}

private void ShowHoldings()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/holdings");

}

else

{

labelText = "You Must Login First!";

}

}

private void ShowAccountForm()

{

if (GlobalSettings.LoggedIn)

{

NavigationManager.NavigateTo("/accountform");

}

else

{

labelText = "You Must Login First!";

}

}

}

### Output

To lay out the buttons vertically each in a table row and center the table on the page in your Blazor component, you can modify the HTML structure to use a table and apply some CSS styles for centering. Here's how you can adjust your Blazor component:

First, wrap your buttons inside a table structure, with each button in its own table row (**<tr>**). Then, use CSS to center the table and style the buttons if needed.

**Updated Blazor Component:**

@page "/"

@rendermode InteractiveServer

@using TradeBlazorApp.Business\_Classes

@inject NavigationManager NavigationManager

<body style="background-image: url('AboutBackground.jpg'); background-size: cover; background-position: center;">

<PageTitle>Trade Home</PageTitle>

<div style="display: flex; justify-content: center; align-items: center; height: 100vh;">

<table class="button-table">

<tr>

<td><button @onclick="ShowQuotes">Quotes</button></td>

</tr>

<tr>

<td><button @onclick="ToggleLogin">Toggle Login</button></td>

</tr>

<tr>

<td><button @onclick="ShowHoldings">Holdings</button></td>

</tr>

<tr>

<td><button @onclick="ShowAccountForm">Account</button></td>

</tr>

</table>

</div>

</body>

**CSS (add this to your site's CSS file or within a <style> tag in your component):**

.button-table {

margin: 0 auto; /\* Center the table horizontally \*/

}

.button-table button {

display: block; /\* Make buttons fill the table cell \*/

width: 100%; /\* Optional: Adjust button width as needed \*/

margin: 5px 0; /\* Add some space between buttons \*/

}

**Explanation:**

* The table containing the buttons is wrapped inside a **<div>** with Flexbox styling (**display: flex;**) to center it vertically and horizontally within the viewport (**height: 100vh** ensures it takes up the full viewport height).
* The **.button-table** class centers the table horizontally with **margin: 0 auto;**.
* Each button is placed in its own table row (**<tr>**) for vertical layout.
* The buttons are styled to fill the width of the table cells, with some margin added for spacing.

This setup should give you a vertically aligned set of buttons centered on the page, with each button in its own table row. Adjust the CSS as needed to fit your design requirements.

### Notes

# Appendix: Why You Cannot “Just Start” With ChatGTP to Understand Target Technologies

Note: This was a failed experiment! Took several days: Try to just start with VB.NET code, and learn Blazor thru ChatGTP without any other knowledge. After this, revert to start of document, after which I have taken MSLearn tutorials on Blazor with .NET 8. Though do note:  
  
**Still never trained on EntityFramework (VB.NET uses 6.4.4, Blazor target latest = 8.0.2)**

## PHASE 1: Simple VB.NET Win Form Code, with Databinding to SQL Server:  Port to Blazor and Learn Blazor Basics, Through ChatGTP with No Prior Knowledge of Blazor, no “External” Tutorials.

## ChatGPT:  Interaction 1

### Input:

Please translate this VB.NET win form application to blazor:  
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> \_

Partial Class Form1

    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.

    <System.Diagnostics.DebuggerNonUserCode()> \_

    Protected Overrides Sub Dispose(ByVal disposing As Boolean)

        Try

            If disposing AndAlso components IsNot Nothing Then

                components.Dispose()

            End If

        Finally

            MyBase.Dispose(disposing)

        End Try

    End Sub

    'Required by the Windows Form Designer

    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer

    'It can be modified using the Windows Form Designer.

    'Do not modify it using the code editor.

    <System.Diagnostics.DebuggerStepThrough()> \_

    Private Sub InitializeComponent()

        Me.components = New System.ComponentModel.Container()

        Dim resources As System.ComponentModel.ComponentResourceManager = New System.ComponentModel.ComponentResourceManager(GetType(Form1))

        Me.Button1 = New System.Windows.Forms.Button()

        Me.QUOTEDBDataSet = New WindowsApp1.QUOTEDBDataSet()

        Me.QUOTEBindingSource = New System.Windows.Forms.BindingSource(Me.components)

        Me.QUOTETableAdapter = New WindowsApp1.QUOTEDBDataSetTableAdapters.QUOTETableAdapter()

        Me.TableAdapterManager = New WindowsApp1.QUOTEDBDataSetTableAdapters.TableAdapterManager()

        Me.QUOTEBindingNavigator = New System.Windows.Forms.BindingNavigator(Me.components)

        Me.BindingNavigatorMoveFirstItem = New System.Windows.Forms.ToolStripButton()

        Me.BindingNavigatorMovePreviousItem = New System.Windows.Forms.ToolStripButton()

        Me.BindingNavigatorSeparator = New System.Windows.Forms.ToolStripSeparator()

        Me.BindingNavigatorPositionItem = New System.Windows.Forms.ToolStripTextBox()

        Me.BindingNavigatorCountItem = New System.Windows.Forms.ToolStripLabel()

        Me.BindingNavigatorSeparator1 = New System.Windows.Forms.ToolStripSeparator()

        Me.BindingNavigatorMoveNextItem = New System.Windows.Forms.ToolStripButton()

        Me.BindingNavigatorMoveLastItem = New System.Windows.Forms.ToolStripButton()

        Me.BindingNavigatorSeparator2 = New System.Windows.Forms.ToolStripSeparator()

        Me.BindingNavigatorAddNewItem = New System.Windows.Forms.ToolStripButton()

        Me.BindingNavigatorDeleteItem = New System.Windows.Forms.ToolStripButton()

        Me.QUOTEBindingNavigatorSaveItem = New System.Windows.Forms.ToolStripButton()

        Me.QUOTEDataGridView = New System.Windows.Forms.DataGridView()

        Me.DataGridViewTextBoxColumn6 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        Me.DataGridViewTextBoxColumn7 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        Me.DataGridViewTextBoxColumn4 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        Me.DataGridViewTextBoxColumn3 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        Me.DataGridViewTextBoxColumn5 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        Me.DataGridViewTextBoxColumn8 = New System.Windows.Forms.DataGridViewTextBoxColumn()

        CType(Me.QUOTEDBDataSet, System.ComponentModel.ISupportInitialize).BeginInit()

        CType(Me.QUOTEBindingSource, System.ComponentModel.ISupportInitialize).BeginInit()

        CType(Me.QUOTEBindingNavigator, System.ComponentModel.ISupportInitialize).BeginInit()

        Me.QUOTEBindingNavigator.SuspendLayout()

        CType(Me.QUOTEDataGridView, System.ComponentModel.ISupportInitialize).BeginInit()

        Me.SuspendLayout()

        '

        'Button1

        '

        Me.Button1.Location = New System.Drawing.Point(689, 131)

        Me.Button1.Name = "Button1"

        Me.Button1.Size = New System.Drawing.Size(182, 83)

        Me.Button1.TabIndex = 0

        Me.Button1.Text = "Button1"

        Me.Button1.UseVisualStyleBackColor = True

        '

        'QUOTEDBDataSet

        '

        Me.QUOTEDBDataSet.DataSetName = "QUOTEDBDataSet"

        Me.QUOTEDBDataSet.SchemaSerializationMode = System.Data.SchemaSerializationMode.IncludeSchema

        '

        'QUOTEBindingSource

        '

        Me.QUOTEBindingSource.DataMember = "QUOTE"

        Me.QUOTEBindingSource.DataSource = Me.QUOTEDBDataSet

        '

        'QUOTETableAdapter

        '

        Me.QUOTETableAdapter.ClearBeforeFill = True

        '

        'TableAdapterManager

        '

        Me.TableAdapterManager.BackupDataSetBeforeUpdate = False

        Me.TableAdapterManager.QUOTETableAdapter = Me.QUOTETableAdapter

        Me.TableAdapterManager.UpdateOrder = WindowsApp1.QUOTEDBDataSetTableAdapters.TableAdapterManager.UpdateOrderOption.InsertUpdateDelete

        '

        'QUOTEBindingNavigator

        '

        Me.QUOTEBindingNavigator.AddNewItem = Me.BindingNavigatorAddNewItem

        Me.QUOTEBindingNavigator.BindingSource = Me.QUOTEBindingSource

        Me.QUOTEBindingNavigator.CountItem = Me.BindingNavigatorCountItem

        Me.QUOTEBindingNavigator.DeleteItem = Me.BindingNavigatorDeleteItem

        Me.QUOTEBindingNavigator.ImageScalingSize = New System.Drawing.Size(24, 24)

        Me.QUOTEBindingNavigator.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.BindingNavigatorMoveFirstItem, Me.BindingNavigatorMovePreviousItem, Me.BindingNavigatorSeparator, Me.BindingNavigatorPositionItem, Me.BindingNavigatorCountItem, Me.BindingNavigatorSeparator1, Me.BindingNavigatorMoveNextItem, Me.BindingNavigatorMoveLastItem, Me.BindingNavigatorSeparator2, Me.BindingNavigatorAddNewItem, Me.BindingNavigatorDeleteItem, Me.QUOTEBindingNavigatorSaveItem})

        Me.QUOTEBindingNavigator.Location = New System.Drawing.Point(0, 0)

        Me.QUOTEBindingNavigator.MoveFirstItem = Me.BindingNavigatorMoveFirstItem

        Me.QUOTEBindingNavigator.MoveLastItem = Me.BindingNavigatorMoveLastItem

        Me.QUOTEBindingNavigator.MoveNextItem = Me.BindingNavigatorMoveNextItem

        Me.QUOTEBindingNavigator.MovePreviousItem = Me.BindingNavigatorMovePreviousItem

        Me.QUOTEBindingNavigator.Name = "QUOTEBindingNavigator"

        Me.QUOTEBindingNavigator.PositionItem = Me.BindingNavigatorPositionItem

        Me.QUOTEBindingNavigator.Size = New System.Drawing.Size(1616, 33)

        Me.QUOTEBindingNavigator.TabIndex = 2

        Me.QUOTEBindingNavigator.Text = "BindingNavigator1"

        '

        'BindingNavigatorMoveFirstItem

        '

        Me.BindingNavigatorMoveFirstItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorMoveFirstItem.Image = CType(resources.GetObject("BindingNavigatorMoveFirstItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorMoveFirstItem.Name = "BindingNavigatorMoveFirstItem"

        Me.BindingNavigatorMoveFirstItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorMoveFirstItem.Size = New System.Drawing.Size(34, 20)

        Me.BindingNavigatorMoveFirstItem.Text = "Move first"

        '

        'BindingNavigatorMovePreviousItem

        '

        Me.BindingNavigatorMovePreviousItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorMovePreviousItem.Image = CType(resources.GetObject("BindingNavigatorMovePreviousItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorMovePreviousItem.Name = "BindingNavigatorMovePreviousItem"

        Me.BindingNavigatorMovePreviousItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorMovePreviousItem.Size = New System.Drawing.Size(34, 20)

        Me.BindingNavigatorMovePreviousItem.Text = "Move previous"

        '

        'BindingNavigatorSeparator

        '

        Me.BindingNavigatorSeparator.Name = "BindingNavigatorSeparator"

        Me.BindingNavigatorSeparator.Size = New System.Drawing.Size(6, 25)

        '

        'BindingNavigatorPositionItem

        '

        Me.BindingNavigatorPositionItem.AccessibleName = "Position"

        Me.BindingNavigatorPositionItem.AutoSize = False

        Me.BindingNavigatorPositionItem.Name = "BindingNavigatorPositionItem"

        Me.BindingNavigatorPositionItem.Size = New System.Drawing.Size(50, 31)

        Me.BindingNavigatorPositionItem.Text = "0"

        Me.BindingNavigatorPositionItem.ToolTipText = "Current position"

        '

        'BindingNavigatorCountItem

        '

        Me.BindingNavigatorCountItem.Name = "BindingNavigatorCountItem"

        Me.BindingNavigatorCountItem.Size = New System.Drawing.Size(54, 25)

        Me.BindingNavigatorCountItem.Text = "of {0}"

        Me.BindingNavigatorCountItem.ToolTipText = "Total number of items"

        '

        'BindingNavigatorSeparator1

        '

        Me.BindingNavigatorSeparator1.Name = "BindingNavigatorSeparator"

        Me.BindingNavigatorSeparator1.Size = New System.Drawing.Size(6, 6)

        '

        'BindingNavigatorMoveNextItem

        '

        Me.BindingNavigatorMoveNextItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorMoveNextItem.Image = CType(resources.GetObject("BindingNavigatorMoveNextItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorMoveNextItem.Name = "BindingNavigatorMoveNextItem"

        Me.BindingNavigatorMoveNextItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorMoveNextItem.Size = New System.Drawing.Size(34, 28)

        Me.BindingNavigatorMoveNextItem.Text = "Move next"

        '

        'BindingNavigatorMoveLastItem

        '

        Me.BindingNavigatorMoveLastItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorMoveLastItem.Image = CType(resources.GetObject("BindingNavigatorMoveLastItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorMoveLastItem.Name = "BindingNavigatorMoveLastItem"

        Me.BindingNavigatorMoveLastItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorMoveLastItem.Size = New System.Drawing.Size(34, 28)

        Me.BindingNavigatorMoveLastItem.Text = "Move last"

        '

        'BindingNavigatorSeparator2

        '

        Me.BindingNavigatorSeparator2.Name = "BindingNavigatorSeparator"

        Me.BindingNavigatorSeparator2.Size = New System.Drawing.Size(6, 6)

        '

        'BindingNavigatorAddNewItem

        '

        Me.BindingNavigatorAddNewItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorAddNewItem.Image = CType(resources.GetObject("BindingNavigatorAddNewItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorAddNewItem.Name = "BindingNavigatorAddNewItem"

        Me.BindingNavigatorAddNewItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorAddNewItem.Size = New System.Drawing.Size(34, 28)

        Me.BindingNavigatorAddNewItem.Text = "Add new"

        '

        'BindingNavigatorDeleteItem

        '

        Me.BindingNavigatorDeleteItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.BindingNavigatorDeleteItem.Image = CType(resources.GetObject("BindingNavigatorDeleteItem.Image"), System.Drawing.Image)

        Me.BindingNavigatorDeleteItem.Name = "BindingNavigatorDeleteItem"

        Me.BindingNavigatorDeleteItem.RightToLeftAutoMirrorImage = True

        Me.BindingNavigatorDeleteItem.Size = New System.Drawing.Size(34, 28)

        Me.BindingNavigatorDeleteItem.Text = "Delete"

        '

        'QUOTEBindingNavigatorSaveItem

        '

        Me.QUOTEBindingNavigatorSaveItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image

        Me.QUOTEBindingNavigatorSaveItem.Image = CType(resources.GetObject("QUOTEBindingNavigatorSaveItem.Image"), System.Drawing.Image)

        Me.QUOTEBindingNavigatorSaveItem.Name = "QUOTEBindingNavigatorSaveItem"

        Me.QUOTEBindingNavigatorSaveItem.Size = New System.Drawing.Size(23, 23)

        Me.QUOTEBindingNavigatorSaveItem.Text = "Save Data"

        '

        'QUOTEDataGridView

        '

        Me.QUOTEDataGridView.AutoGenerateColumns = False

        Me.QUOTEDataGridView.ColumnHeadersHeightSizeMode = System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize

        Me.QUOTEDataGridView.Columns.AddRange(New System.Windows.Forms.DataGridViewColumn() {Me.DataGridViewTextBoxColumn6, Me.DataGridViewTextBoxColumn7, Me.DataGridViewTextBoxColumn4, Me.DataGridViewTextBoxColumn3, Me.DataGridViewTextBoxColumn5, Me.DataGridViewTextBoxColumn8})

        Me.QUOTEDataGridView.DataSource = Me.QUOTEBindingSource

        Me.QUOTEDataGridView.Location = New System.Drawing.Point(285, 309)

        Me.QUOTEDataGridView.Name = "QUOTEDataGridView"

        Me.QUOTEDataGridView.RowHeadersWidth = 62

        Me.QUOTEDataGridView.RowTemplate.Height = 28

        Me.QUOTEDataGridView.Size = New System.Drawing.Size(1006, 333)

        Me.QUOTEDataGridView.TabIndex = 3

        '

        'DataGridViewTextBoxColumn6

        '

        Me.DataGridViewTextBoxColumn6.DataPropertyName = "COMPANYNAME"

        Me.DataGridViewTextBoxColumn6.HeaderText = "COMPANYNAME"

        Me.DataGridViewTextBoxColumn6.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn6.Name = "DataGridViewTextBoxColumn6"

        Me.DataGridViewTextBoxColumn6.Width = 150

        '

        'DataGridViewTextBoxColumn7

        '

        Me.DataGridViewTextBoxColumn7.DataPropertyName = "SYMBOL"

        Me.DataGridViewTextBoxColumn7.HeaderText = "SYMBOL"

        Me.DataGridViewTextBoxColumn7.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn7.Name = "DataGridViewTextBoxColumn7"

        Me.DataGridViewTextBoxColumn7.Width = 150

        '

        'DataGridViewTextBoxColumn4

        '

        Me.DataGridViewTextBoxColumn4.DataPropertyName = "PRICE"

        Me.DataGridViewTextBoxColumn4.HeaderText = "PRICE"

        Me.DataGridViewTextBoxColumn4.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn4.Name = "DataGridViewTextBoxColumn4"

        Me.DataGridViewTextBoxColumn4.Width = 150

        '

        'DataGridViewTextBoxColumn3

        '

        Me.DataGridViewTextBoxColumn3.DataPropertyName = "VOLUME"

        Me.DataGridViewTextBoxColumn3.HeaderText = "VOLUME"

        Me.DataGridViewTextBoxColumn3.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn3.Name = "DataGridViewTextBoxColumn3"

        Me.DataGridViewTextBoxColumn3.Width = 150

        '

        'DataGridViewTextBoxColumn5

        '

        Me.DataGridViewTextBoxColumn5.DataPropertyName = "HIGH"

        Me.DataGridViewTextBoxColumn5.HeaderText = "HIGH"

        Me.DataGridViewTextBoxColumn5.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn5.Name = "DataGridViewTextBoxColumn5"

        Me.DataGridViewTextBoxColumn5.Width = 150

        '

        'DataGridViewTextBoxColumn8

        '

        Me.DataGridViewTextBoxColumn8.DataPropertyName = "CHANGE1"

        Me.DataGridViewTextBoxColumn8.HeaderText = "CHANGE1"

        Me.DataGridViewTextBoxColumn8.MinimumWidth = 8

        Me.DataGridViewTextBoxColumn8.Name = "DataGridViewTextBoxColumn8"

        Me.DataGridViewTextBoxColumn8.Width = 150

        '

        'Form1

        '

        Me.AutoScaleDimensions = New System.Drawing.SizeF(9.0!, 20.0!)

        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font

        Me.ClientSize = New System.Drawing.Size(1616, 785)

        Me.Controls.Add(Me.QUOTEDataGridView)

        Me.Controls.Add(Me.QUOTEBindingNavigator)

        Me.Controls.Add(Me.Button1)

        Me.Name = "Form1"

        Me.Text = "Form1"

        CType(Me.QUOTEDBDataSet, System.ComponentModel.ISupportInitialize).EndInit()

        CType(Me.QUOTEBindingSource, System.ComponentModel.ISupportInitialize).EndInit()

        CType(Me.QUOTEBindingNavigator, System.ComponentModel.ISupportInitialize).EndInit()

        Me.QUOTEBindingNavigator.ResumeLayout(False)

        Me.QUOTEBindingNavigator.PerformLayout()

        CType(Me.QUOTEDataGridView, System.ComponentModel.ISupportInitialize).EndInit()

        Me.ResumeLayout(False)

        Me.PerformLayout()

    End Sub

    Friend WithEvents Button1 As Button

    Friend WithEvents QUOTEDBDataSet As QUOTEDBDataSet

    Friend WithEvents QUOTEBindingSource As BindingSource

    Friend WithEvents QUOTETableAdapter As QUOTEDBDataSetTableAdapters.QUOTETableAdapter

    Friend WithEvents TableAdapterManager As QUOTEDBDataSetTableAdapters.TableAdapterManager

    Friend WithEvents QUOTEBindingNavigator As BindingNavigator

    Friend WithEvents BindingNavigatorAddNewItem As ToolStripButton

    Friend WithEvents BindingNavigatorCountItem As ToolStripLabel

    Friend WithEvents BindingNavigatorDeleteItem As ToolStripButton

    Friend WithEvents BindingNavigatorMoveFirstItem As ToolStripButton

    Friend WithEvents BindingNavigatorMovePreviousItem As ToolStripButton

    Friend WithEvents BindingNavigatorSeparator As ToolStripSeparator

    Friend WithEvents BindingNavigatorPositionItem As ToolStripTextBox

    Friend WithEvents BindingNavigatorSeparator1 As ToolStripSeparator

    Friend WithEvents BindingNavigatorMoveNextItem As ToolStripButton

    Friend WithEvents BindingNavigatorMoveLastItem As ToolStripButton

    Friend WithEvents BindingNavigatorSeparator2 As ToolStripSeparator

    Friend WithEvents QUOTEBindingNavigatorSaveItem As ToolStripButton

    Friend WithEvents QUOTEDataGridView As DataGridView

    Friend WithEvents DataGridViewTextBoxColumn6 As DataGridViewTextBoxColumn

    Friend WithEvents DataGridViewTextBoxColumn7 As DataGridViewTextBoxColumn

    Friend WithEvents DataGridViewTextBoxColumn4 As DataGridViewTextBoxColumn

    Friend WithEvents DataGridViewTextBoxColumn3 As DataGridViewTextBoxColumn

    Friend WithEvents DataGridViewTextBoxColumn5 As DataGridViewTextBoxColumn

    Friend WithEvents DataGridViewTextBoxColumn8 As DataGridViewTextBoxColumn

End Class

### Output

To translate the provided VB.NET WinForms code to a Blazor (WebAssembly or Server) application in C#, you will have to rewrite the user interface logic, since Blazor uses a component-based architecture with Razor syntax, which is quite different from the WinForms event-driven model. Below is a simplified translation focusing on the main aspects: displaying data in a table and a button click event. Note that Blazor applications use services for data access, and components for UI, so the exact translation would depend on your application's architecture.

1. **Razor Component for Displaying Data (e.g., Quotes.razor):**

@page "/quotes"

@inject IQuoteService QuoteService

<h3>Quotes</h3>

@if (quotes == null)

{

    <p><em>Loading...</em></p>

}

else

{

    <table class="table">

        <thead>

            <tr>

                <th>Company Name</th>

                <th>Symbol</th>

                <th>Price</th>

                <th>Volume</th>

                <th>High</th>

                <th>Change</th>

            </tr>

        </thead>

        <tbody>

        @foreach (var quote in quotes)

        {

            <tr>

                <td>@quote.CompanyName</td>

                <td>@quote.Symbol</td>

                <td>@quote.Price</td>

                <td>@quote.Volume</td>

                <td>@quote.High</td>

                <td>@quote.Change</td>

            </tr>

        }

        </tbody>

    </table>

}

<button class="btn btn-primary" @onclick="LoadData">Load Data</button>

@code {

    private List<Quote> quotes;

    protected override async Task OnInitializedAsync()

    {

        quotes = await QuoteService.GetQuotesAsync();

    }

    private async Task LoadData()

    {

        quotes = await QuoteService.GetQuotesAsync();

    }

}

public interface IQuoteService

{

    Task<List<Quote>> GetQuotesAsync();

}

**C# Service for Data Access (e.g., IQuoteService.cs and QuoteService.cs):**

public class QuoteService : IQuoteService

{

    public async Task<List<Quote>> GetQuotesAsync()

    {

        // Implement your data fetching logic here

        // This could be from an API, database, etc.

        return new List<Quote>();

    }

}

**Quote Model (e.g., Quote.cs):**

public class Quote

{

    public string CompanyName { get; set; }

    public string Symbol { get; set; }

    public decimal Price { get; set; }

    public int Volume { get; set; }

    public decimal High { get; set; }

    public decimal Change { get; set; }

}

**Register the Service in Startup.cs or Program.cs (Blazor Server or WebAssembly respectively):**

For Blazor Server:

public class Startup

{

    public void ConfigureServices(IServiceCollection services)

    {

        services.AddScoped<IQuoteService, QuoteService>();

        // Other service registrations

    }

    // Configure method...

}

**For Blazor WebAssembly:**

public class Program

{

    public static async Task Main(string[] args)

    {

        var builder = WebAssemblyHostBuilder.CreateDefault(args);

        builder.Services.AddScoped<IQuoteService, QuoteService>();

        // Other service registrations

        await builder.Build().RunAsync();

    }

}

This example demonstrates a basic structure for a Blazor application that displays a list of quotes in a table and refreshes the data when a button is clicked. Depending on your requirements, you might need to implement additional features like data editing, sorting, or filtering.

### Notes

Now:    
  
I create a new Blazor Web Assembly project, using the sample pages in the template (home, counter, weather).

Spend 30 mins looking through the template app.  Blazor all new to me, completely unfamiliar with layout of project, how the components work together.

Time spent:  30 mins

Copy the weather page, Weather.razor, rename as Quotes.Razor

Figure out where the nav menu is, add a new nav menu to the new page Quotes, rename page from the copies weather page to quotes in Quotes.razor.

Test and run.  New nav works, to the Quotes.razor page, which is still just a copy of the weather page logic.

Time spent: 10 min

Next, replace the Quotes.razor page, 100%, with the ChatGPT 4 generated code for Quotes.Razor.

This has interface to interface IQuoteService.cs, the class QuoteService, and the model class Quote.cs (for data access).  
  
So now I need to figure out how to add these .cs files.  On to ChatGPT:

## ChatGTP Interaction 2

### Input

how and where do I add a c# class to a Blazor project in visual studio 2022

### Output

Adding a C# class to a Blazor project in Visual Studio 2022 involves a few straightforward steps. Here's how you can do it:

1. **Open Your Project:** Start by opening your Blazor project in Visual Studio 2022. You can do this by selecting "Open a project or solution" from the start window or "File" > "Open" > "Project/Solution" from the menu bar if Visual Studio is already open.
2. **Select the Project:** In the Solution Explorer, which is usually on the right side of the Visual Studio interface, locate your Blazor project. This is important because a solution can contain multiple projects, and you want to make sure you're adding the class to the correct one.
3. **Add New Class:** Right-click on the project name or a specific folder within the project where you want to add the class. Choose "Add" from the context menu, and then select "Class...". This will open the "Add New Item" dialog.
4. **Configure the Class:** In the "Add New Item" dialog, you'll be able to choose from different types of files. Make sure "Class" is selected (it's usually selected by default). You will then need to give your class a name. It's important to follow C# naming conventions, so start your class name with a capital letter and use CamelCase for multi-word names. For example, **MyNewClass.cs**.
5. **Add Class Content:** After you click "Add", Visual Studio will create the class file and open it for editing. You'll see a new C# file with a basic class structure. You can now add properties, methods, and other class members as needed.
6. **Save Your Work:** Remember to save your work by pressing **Ctrl + S** or by selecting "File" > "Save" from the menu bar.
7. **Use Your Class:** Once the class is added, you can use it in your Blazor components or other parts of your application by creating instances of the class or by accessing its static members, depending on how you've defined it.

Here's an example of what the initial content of your new class might look like:

csharpCopy code

using System; namespace YourProjectNamespace { public class MyNewClass { // Add properties, methods, etc. here } }

Make sure to replace **YourProjectNamespace** with the actual namespace of your project, which Visual Studio will usually fill in automatically based on where you add the class.

If your Blazor project is structured into different layers or areas (e.g., Data, Services, Components), it's a good practice to add your class to the appropriate project or folder to keep your project organized.

### Notes

I go through the steps, creating:

-One Interface (IQuoteService.cs); pasted code from ChatGPT

-One class named IQuoteService, pasted code from ChatGPT

-One class named Quote.cs, pasted code from ChatGPT  
  
**BUILD SUCCEEDS!**Time Spent:  15 mins after first reading through, and looking at some online docs outside of ChatGPT around classes in Blazor.

**Now Run in debug:    
  
I GET NO NEW PAGE, and “UNHANDLED ERROR HAS OCCURRED” at bottom of the web browser UI.**But I forgot the last steps from CHAT GPT:    
  
“**Register the Service in Startup.cs or Program.cs (Blazor Server or WebAssembly respectively):”**

But there is no Startup.cs class, or Program.cs files in the template project.    
  
I discover in the template app, the weather page demonstrates a server component, and the code to “register” seems to be embedded in page itself, Weather.razor.  
  
I realize, I could do this, likely, instead of creating a Startup.cs class, but not sure where to put it.

Time spent:  10 mins  
  
Back to ChatGTP  (interaction 3)

## ChatGTP Interaction 3

### INPUT

How and where and why would I add a startup.cs class to my Blazor project

### OUTPUT

Adding a **Startup.cs** class to a Blazor project is common in Blazor Server applications, as it plays a critical role in configuring the app's services and the request pipeline. For Blazor WebAssembly projects, the startup configuration is typically done in **Program.cs** due to its client-side nature. However, if you're working with a Blazor Server project or have a specific architectural preference or requirement for your Blazor WebAssembly app, you might still want to add a **Startup.cs** class.

**How to Add Startup.cs to a Blazor Project:**

1. **Open Your Project**: Launch Visual Studio 2022 and open your Blazor project.
2. **Add New Item**: In the Solution Explorer, right-click on your project and choose "Add" > "New Item...".
3. **Choose Class**: In the "Add New Item" dialog, select "Class" and name it **Startup.cs**.
4. **Add Configuration**: After creating **Startup.cs**, you need to add two essential methods: **ConfigureServices** for configuring services and **Configure** for configuring the HTTP request pipeline.

Your **Startup.cs** might look something like this for a basic setup:

**using Microsoft.AspNetCore.Builder;**

**using Microsoft.AspNetCore.Hosting;**

**using Microsoft.Extensions.DependencyInjection;**

public class Startup

{

    public void ConfigureServices(IServiceCollection services)

    {

        services.AddRazorPages();

        services.AddServerSideBlazor();

        // Add other services here

    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

    {

        if (env.IsDevelopment())

        {

            app.UseDeveloperExceptionPage();

        }

        else

        {

            app.UseExceptionHandler("/Error");

            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

            app.UseHsts();

        }

        app.UseHttpsRedirection();

        app.UseStaticFiles();

        app.UseRouting();

        app.UseEndpoints(endpoints =>

        {

            endpoints.MapBlazorHub();

            endpoints.MapFallbackToPage("/\_Host");

        });

    }

}

**Where to Configure Startup.cs in Your Project:**

In a Blazor Server app, the **Startup.cs** class is typically located at the root of the project for easy access and convention. For Blazor WebAssembly, if you choose to use a similar pattern for structuring your app, you would place it at the root as well, although configuring services and components might differ due to the client-side nature of Blazor WebAssembly.

**Why Use Startup.cs:**

* **Organization**: Separating configuration into a **Startup.cs** class helps keep your project organized and adheres to the separation of concerns principle. It centralizes service registration and middleware configuration.
* **Conventions and Readability**: It follows ASP.NET Core conventions, making it easier for developers familiar with ASP.NET Core to understand the structure of your app.
* **Flexibility**: It provides a clear location to configure services and middleware, making your application more modular and easier to maintain or extend.

**Configuring the Application to Use Startup.cs:**

If you're adding **Startup.cs** to a Blazor WebAssembly app or if it wasn't included by default in your Blazor Server app, you need to configure the application to use it. This is typically done in the **Program.cs** file by configuring the Web Host Builder:

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>();

            });

}

### Notes

This code tells the application to use the **Startup** class for configuration. It's a crucial step to ensure that your **Startup.cs** class is actually invoked and used by the application.  
  
  
I create a new Startup.cs class in the root of the project, and paste ChatGTP code.  Myriad of issues:  
  
The three using statements have “Assembly reference not found”.  Use VS CTRL-ENTER to fix:  “Remove unnecessary using statements”.  
  
After that:  I still have a solution with errors that looks like this:  
  
A screenshot of a computer program

Description automatically generated

Time Spent:  30 minutes.

I now realize, that the original project I created—on ChatGTP advice, was to create a Web Assembly Standalone Project.  Not a Blazor Server App.  However, the sample pages VS creates include a page, Weather.CS, that shows how to “fetch data from a server”.  So I am massively confused:  
  
Should I have started with a Blazor server App, instead?  
I see now the distinction between Razor code that runs in browser, vs.Blazor server code.  I think its cool, but not sure now how to add all the right server components to my Solution.  Should this be a separate project in the Solution?  Or embed in the existing project?  I need server logic, to do the databinding to my client-side UI elements.  So, I will not start to unpack this yet, going through code and a bit more education on how Blazor works, clientside and serverside.  This will take a few hours of learning.  I could go through some MSDN tutorials, but I will start with ChatGPT:

## ChatGTP Interaction 4

### INPUT

Explain the difference between blazor client code and blazor server code

### OUTPUT

Blazor is a web framework from Microsoft that allows developers to build interactive web applications using C# instead of JavaScript. Blazor can run in two distinct modes: Blazor Server and Blazor WebAssembly (Client). Each mode has a different architecture and operates differently. Here's a breakdown of the key differences between Blazor Server code and Blazor Client (WebAssembly) code:

**Blazor Server:**

1. **Server-side Execution:** In Blazor Server, the application's components are executed on the server. All the logic, including event handling and data binding, runs in the server-side .NET runtime.
2. **SignalR Communication:** The browser and the server communicate via a SignalR connection, a real-time web socket communication framework. When an event occurs in the browser, a message is sent to the server over SignalR, the server processes the event, and then sends back the UI diff to update the UI in the browser.
3. **Latency and Scalability:** Because of the constant communication between the client and the server for every UI event, applications might experience latency, especially with poor network conditions. Scalability can also be a concern since each client requires an active server connection, consuming server resources.
4. **No WebAssembly:** Blazor Server doesn't use WebAssembly. The code runs entirely on the server, and only UI updates are exchanged with the client.
5. **Thin Clients:** Since the application logic is executed on the server, clients don't need to be powerful, making Blazor Server a good choice for scenarios where clients have limited resources.

**Blazor WebAssembly (Client):**

1. **Client-side Execution:** In Blazor WebAssembly, the application's components and logic run directly in the browser using WebAssembly. The .NET runtime is downloaded to the browser, allowing the C# code to run on the client-side.
2. **No Active Server Connection Required:** Once the application is loaded in the browser, it can operate without a continuous connection to the server. This can reduce latency seen in server-based interactions and lessen server load, as the server is only contacted when necessary (e.g., API calls for data).
3. **Uses WebAssembly:** The Blazor WebAssembly apps run on a WebAssembly-based .NET runtime that's executed in the browser. This allows for near-native performance for client-side operations.
4. **Requires Capable Clients:** Since the application runs in the client's browser, it requires more resources on the client-side compared to Blazor Server. Clients need to be capable of downloading the app and running the WebAssembly runtime.
5. **Offline Support:** Blazor WebAssembly can work offline or with limited connectivity once the application is fully downloaded, making it suitable for progressive web applications (PWA).

In summary, the choice between Blazor Server and Blazor WebAssembly depends on the specific requirements of the application, such as whether low latency and offline capabilities are more important (favoring Blazor WebAssembly) or whether minimizing client resources and leveraging server-side capabilities are the priority (favoring Blazor Server).

---------------------  
  
Oh boy, I have more learning to do.  Basically, I understand, but of course what I want is really a hybrid of these two.  Key elements operating on client, but databinding on server, where can communicate with my SQL Database.  Can I do a hybrid model, in a single project?  The sample page (Weather.cs) suggests yes, I can, as it promotes itself of an example of “Fetching Data from a Server” in the comments.  
  
<PageTitle>Weather</PageTitle>

<h1>Weather</h1>

<p>**This component demonstrates fetching data from the server**.</p>

So, more investigation, via ChatGTP:

## ChatGTP INTERACTION 5

### INPUT

can a standalone Blazor Web Assembly project also contain server logic to get data from a database?

### OUTPUT

A standalone Blazor WebAssembly project is primarily client-side and runs in the browser using WebAssembly. It is designed to execute the UI and application logic on the client side, without a direct connection to a server for processing each interaction. However, it can still interact with servers or backend services, such as databases, but this is typically done through API calls over HTTP or similar protocols.

To access data from a database in a standalone Blazor WebAssembly application, you would typically implement a separate backend service, such as a REST API or a gRPC service, which the Blazor application can call to retrieve or store data. This backend service would contain the server logic needed to interact with the database, handle business logic, perform authentication and authorization, etc.

Here's a high-level overview of how this might work:

1. **Backend Service:** Create a backend service using ASP.NET Core, Node.js, or any other technology capable of serving HTTP requests. This service will contain the server logic to interact with your database. It will expose endpoints (APIs) for the operations you need to perform, such as querying, inserting, updating, and deleting data.
2. **Database Access:** The backend service will handle all direct interactions with the database, ensuring that data access is secure and efficient. This might involve using Entity Framework for .NET, Mongoose for Node.js with MongoDB, or other ORM/data access technologies depending on your stack.
3. **API Calls from Blazor WebAssembly:** Your Blazor WebAssembly application will make HTTP requests to the backend service's endpoints to perform data operations. You can use HttpClient in Blazor WebAssembly to send these requests and process the responses.
4. **CORS Configuration:** Since your Blazor WebAssembly application will likely be served from a different origin than your backend service, you'll need to configure Cross-Origin Resource Sharing (CORS) in your backend to allow requests from your Blazor application.
5. **Authentication and Authorization:** If your application requires user authentication, you will implement this in your backend service. The Blazor WebAssembly app can then integrate with the authentication mechanism (e.g., using OAuth, OpenID Connect) to secure API calls.

### Notes

While a standalone Blazor WebAssembly project doesn't contain server logic within the same project, it can certainly communicate with a separate backend service that contains the necessary server logic for database interactions. This architecture allows you to leverage the scalability and security of server-side data access while still providing a rich, interactive client-side UI with Blazor WebAssembly.

Time spent, thinking, and ChatGTP interactions: 15 minutes

OK:  AHA.  This is still confusing…it basically tells  me:  just use std Web Services for all logic you need running on the server.  Classically, this should be the business logic layer and the data access layer.  I could do that.  No BLAZOR code on server.  But ChatGTP earlier interactions, seem to indicate I can do something on the client.  My hunch is, this last interaction/response from ChatGTP is RIGHT!    
  
AHA MOMENT:  USE BLAZOR FOR RICH CLIENT UI BUILDING.  USE RESTFUL APIS for interactions/databinding with server.  But, this seems chatty to me.  VERY DIFFERENT FROM ASP.NET, where all logic runs on the server to generate pages/UI sent to client as thin-client UI (though of course those pages my have client side javascript in them as well).  
  
SO:  THINKING, WHAT PATH TO EXPLORE?  I will first explore just trying to make existing project work, and sort out factoring between client and server logic later.  I still don’t understand blazor server: but think it must be a replacement for ASP.NET?  all server-side generated pages?  BUT THEN how achieve what I think its all about:  connecting rich web client components to backend server logic, all done in Blazor (combo of blazer client and blazor server).  This is what I must figure out.  I think I need to start by examining that VS template page, the Weather page that “fetches data from a server”.

## ChatGTP Interaction 6

### Input

Can you tell me what the Weather.razor page illustrates in the Visual Studio 2022 template for Razor Standalone Web Assemblies?

### Output

The **Weather.razor** page in the Visual Studio 2022 template for a standalone Blazor WebAssembly application typically serves as a demonstration of how to implement data fetching and display within a Blazor component. This page often includes a sample implementation showcasing various Blazor features such as component markup, code blocks, dependency injection, and HTTP client usage.

Here is a general overview of what the **Weather.razor** page might illustrate:

1. **Component Structure:** The page demonstrates the structure of a Blazor component, which includes HTML markup for rendering the UI and a **@code** block for the C# logic.
2. **Dependency Injection:** It often uses dependency injection to bring in required services, such as **HttpClient**, which is used to make HTTP requests from the client-side Blazor application.
3. **HTTP Client Usage:** The component typically makes an HTTP GET request to a predefined API endpoint (often a sample endpoint provided within the template) to fetch weather forecast data. This demonstrates how to use **HttpClient** to call APIs and handle responses.
4. **Asynchronous Data Fetching:** The example shows how to perform asynchronous operations in Blazor components, using **async** and **await** keywords to fetch data without blocking the UI thread.
5. **Data Display:** The fetched data is usually stored in a local variable, and the component's markup section displays this data in a table or another suitable format. This illustrates how to bind C# variables to HTML elements and dynamically update the UI based on the fetched data.
6. **Error Handling:** The example might include basic error handling to demonstrate how to deal with exceptions or errors that occur during the data fetching process.
7. **Use of Models:** The template might define a C# class (model) to represent the weather forecast data, showing how to use models to work with strongly typed data in Blazor applications.

This **Weather.razor** page serves as a practical example for developers to understand and learn how to build Blazor components that interact with backend services, fetch and display data, and handle user interactions. The exact implementation details can vary depending on the template version and any updates to the Visual Studio templates, but the core concepts illustrated by the page generally remain consistent.

Time spent reading, thinking:  25 mins

Decision point questions in my head:  
  
Should I abandon ChatGTP for awhile:  and first do one or more of:  
  
1. Take Microsoft Tutorials on creating a complete server side Blazor project?  And maybe their tutorial on creating a whizzier client side Blazor (Web Assembly standalone) project?  Do some focused learning (a few hours at least, outside of ChatGTP, on these step by step tutorials, using ChatGTP for specific questions on that process?  To better understanding factoring between client and server apps in Blazor?  
  
2.  Should I just use ChatGTP to build me such a tutorial?  (YES:  I will try this, as a next step, given project goals).  HOWEVER, the VS WEB ASSEMBLY template sample pages already seem to have much of this.    
  
3.  Again:  I want a hybrid app—some stuff in Blazor on client, some Blazor server.  But I am not sure that is the right approach.  I am beginning to think:  Use Blazor on client for a rich UI, that mimics control experience in VB.NET Win Forms.  Use strict Restful APIs for databinding calls to server.  
  
4.  Much learning to do.  How far can ChatGTP Take me on this learning?

## ChatGTP INTERACTION 7

### Input

can you build me a step by step tutorial on creating a Blazor client app that demonstrates server side databinding to a Blazor server app?

### Output

Creating a Blazor client app that demonstrates server-side data binding to a Blazor server app involves several steps. Here's a step-by-step tutorial to guide you through the process:

**Step 1: Setup Your Environment**

1. **Install .NET SDK**: Ensure you have the latest .NET SDK installed on your machine. You can download it from the .NET website.
2. **Install Visual Studio**: For a comprehensive development environment, install the latest version of Visual Studio with the ASP.NET and web development workload. You can download it from the Visual Studio website.

**Step 2: Create the Server App**

1. **Open Visual Studio** and select "Create a new project".
2. Choose "Blazor App" from the project template list and click "Next".
3. Name your project (e.g., **BlazorServerApp**), choose a location, and click "Create".
4. Select "Blazor Server App" and click "Create".

**Step 3: Define a Model**

1. Right-click on your server project (**BlazorServerApp**) in the Solution Explorer, select "Add" > "Class", name it **Product.cs** (or any model name you prefer).
2. Define properties in your model class. For example:

public class Product

{

    public int Id { get; set; }

    public string Name { get; set; }

    public decimal Price { get; set; }

}

**Step 4: Create a Service**

1. Right-click on the server project, select "Add" > "New Folder" and name it **Services**.
2. Inside the **Services** folder, right-click and select "Add" > "Class", name it **ProductService.cs**.
3. Implement a service to manage product data. For example:

using System.Collections.Generic;

using System.Linq;

public class ProductService

{

    private static readonly List<Product> Products = new List<Product>

    {

        new Product { Id = 1, Name = "Product 1", Price = 100 },

        // Add more products as needed

    };

    public IEnumerable<Product> GetAllProducts() => Products;

    public Product GetProductById(int id) => Products.FirstOrDefault(p => p.Id == id);

}

**Step 5: Register the Service**

1. Open **Startup.cs** in your server project.
2. In the **ConfigureServices** method, register your service for dependency injection:

services.AddSingleton<ProductService>();

**Step 6: Create the Client App**

1. Right-click on the solution in Solution Explorer, select "Add" > "New Project".
2. Choose "Blazor WebAssembly App", click "Next".
3. Name your project (e.g., **BlazorClientApp**), and click "Create".
4. Uncheck "Configure for HTTPS" if not needed and ensure "ASP.NET Core hosted" is checked to create a client app hosted by an ASP.NET Core server.

**Step 7: Setup HTTP Client in Client App**

1. In the **Program.cs** of your client project, add the following to configure the HTTP client:

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });

**Step 8: Create Client-Side UI**

1. In the client project, navigate to the **Pages** folder, right-click and add a new "Razor Component".
2. Name it **FetchData.razor** or similar.
3. Implement the UI and logic to fetch data from the server. Use **@inject HttpClient Http** to inject the HTTP client.
4. Use the **OnInitializedAsync** method to make an HTTP GET request to the server app's endpoint that returns product data (you may need to create this endpoint in the server app if it doesn't exist).

**Step 9: Run Your Application**

1. Set both projects (**BlazorServerApp** and **BlazorClientApp**) as startup projects.
2. Run the solution. Your client app should now fetch data from the server app and display it.

**Additional Notes**

* Ensure CORS is properly configured on the server to allow requests from your client app, especially if they are hosted separately.
* You might need to adjust the URLs in your HTTP requests based on where your server app is hosted in relation to your client app.

This tutorial outlines the basic steps to create a Blazor client app with server-side data binding. Depending on your specific requirements, you may need to adjust or extend these steps.

### Notes

Greg Notes:  Step 5:  There is no Startup.cs in my project:  There is a program.cs.  When adding code to program.cs, get error:  “The name services does not exist in the current context.”

Did I create the right type of project (I chose Blazor WebApp)

(later down, discover, Startup.cs is for .NET versions below 6.0.)  Bootstrap process in Program.cs now, consolidated and supposedly simplified.  But this is real time (took me awhile, in CHATGPT , to target answers for .NET 8.0)

## ChatGTP INTERACTION 8

### INPUT:

In a Blazor Web App what's the difference between startup.cs and program.cs?

### OUTPUT

In a Blazor Web App, **Startup.cs** and **Program.cs** play different roles in the application's configuration and startup process. Their usage can vary slightly depending on whether you're working with a Blazor Server App or a Blazor WebAssembly (WASM) App. Here's a breakdown of their differences and roles:

**Program.cs**

* **Primary Role**: Acts as the entry point of the application. It's where the application starts running.
* **Contents**: In **Program.cs**, you typically find code that creates a host for the web application. For a Blazor Server App, this involves setting up and configuring an ASP.NET Core Web Host (**IWebHostBuilder**). For a Blazor WebAssembly App, it involves configuring the WebAssembly host.
* **Configuration**: It's responsible for configuring app-wide settings, such as logging, dependency injection services, and the application's server (in the case of a Blazor Server App). In Blazor WebAssembly apps, it's also where you configure services available in the client-side application, like HTTP Client services for making web requests.
* **Updates**: In more recent versions of .NET (starting from .NET 6), the startup process has been simplified, and for many applications, the configuration and startup process traditionally found in **Startup.cs** has been moved to **Program.cs**, making **Startup.cs** optional or even obsolete in some cases.

**Startup.cs**

* **Primary Role**: Used specifically in Blazor Server Apps and earlier versions of ASP.NET Core applications to configure services and the app's request pipeline.
* **Contents**: Contains two primary methods: **ConfigureServices** and **Configure**.
  + **ConfigureServices**: Used to add services to the app's dependency injection container. These services can then be used throughout the application.
  + **Configure**: Used to define how the app responds to HTTP requests, effectively setting up the middleware pipeline.
* **Configuration**: In a Blazor Server App, **Startup.cs** is where you configure server-side aspects like Entity Framework Core, Identity services, MVC services for controllers and views (if used alongside Blazor), and SignalR (which Blazor Server uses under the hood for its real-time communication).
* **Updates**: As mentioned, in the latest .NET versions, the configuration and services setup traditionally done in **Startup.cs** can be moved to **Program.cs**, especially for simpler applications or to follow the newer minimal hosting model introduced in .NET 6. This change aims to reduce complexity and boilerplate code.

**Summary**

* In **Blazor Server Apps**, **Startup.cs** used to be more commonly used for detailed configuration of services and middleware, whereas **Program.cs** was the entry point setting up the host. However, with the advent of .NET 6 and later, the trend is to consolidate this setup within **Program.cs**, potentially eliminating the need for **Startup.cs**.
* In **Blazor WebAssembly Apps**, **Program.cs** is typically where you configure the client-side application, including services available in the Blazor app. **Startup.cs** is not commonly used in this model, as the client-side app's configuration is more streamlined and often doesn't require the extensive setup seen in server-side applications.

This evolution reflects the .NET team's efforts to simplify the application startup process and reduce the amount of boilerplate code developers need to manage.

## ChatGTP INTERACTION 9

### INPUT

I am building a Blazor Web App in .NET 8, and in program.cs, I get the following error when adding this code: services.AddSingleton<ProductService>(); "The name services does not exist in the current context" Why?

### OUTPUT

The error message "The name 'services' does not exist in the current context" typically occurs when the code snippet is placed outside of a method or a scope where the **services** variable is defined and available. In the context of a Blazor Web App, especially with .NET 8, the setup for dependency injection and service configuration is usually done inside the **Program.cs** file, but it needs to be within the proper method or lambda expression where the **services** collection is available.

For .NET 6 onwards, the template for web applications, including Blazor, was updated to use a more simplified top-level statement approach, which may differ from older .NET versions. Here's how you typically configure services in a Blazor Web App with .NET 6, 7, or 8:

1. **Ensure you are within the right scope**: The **AddSingleton**, **AddScoped**, or **AddTransient** methods should be called on the **IServiceCollection** instance within the scope where it's available. In a Blazor Web App, this is typically done within the **builder.Services** configuration section.
2. **Use the builder instance**: In .NET 6 and later, the **Program.cs** file uses a **WebApplicationBuilder** instance, commonly referred to as **builder**, to set up the application. You should add services to the **IServiceCollection** via the **builder.Services** property.

Here's an example of how you might configure your **ProductService** in **Program.cs**:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddSingleton<ProductService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

    app.UseExceptionHandler("/Error");

    app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

Make sure that your **services.AddSingleton<ProductService>();** line is within the scope where **builder.Services** is being configured. If your **Program.cs** does not follow this structure, please adjust it according to the .NET version you are using and ensure that the service registration is done within the appropriate configuration block.

### Notes

I used this to successfully fix the issue!

Back to tutorial ChatGPT Created for Me:

Created client app.  Now get error and its very unclear how to use:

“OnInitializedAsync();” to “to make an HTTP GET request to the server app's endpoint that returns product data (you may need to create this endpoint in the server app if it doesn't exist).” Per tutorial

## ChatGTP INTERACTION 10

### INPUT

In a .NET 8 Blazor web assembly app, how to I use the OnInitializedAsync method to make an HTTP GET request to the server app's endpoint ?

### OUTPUT

To make an HTTP GET request from a Blazor WebAssembly (WASM) app to a server app's endpoint, you typically use the **HttpClient** class provided by .NET. In a Blazor WASM app, **HttpClient** is usually injected as a dependency into the component where you need to make the HTTP call.

Here's a step-by-step guide on how to use the **OnInitializedAsync** lifecycle method to make an HTTP GET request in your Blazor WebAssembly app:

1. **Inject HttpClient**: First, ensure that **HttpClient** is available for dependency injection in your Blazor app. In a typical Blazor WASM app, **HttpClient** is added to the service collection in the **Program.cs** file and is configured to be available for dependency injection.
2. **Inject into Component**: In the component where you want to make the HTTP GET request, use the **@inject** directive to inject an instance of **HttpClient**.
3. **Override OnInitializedAsync**: Use the **OnInitializedAsync** method to make the HTTP GET request. This method is asynchronous, allowing you to await the HTTP request without blocking the UI thread.
4. **Make the HTTP GET Request**: Use the **HttpClient** instance to send a GET request to your server app's endpoint and process the response.

Here is an example of how this might look in a Blazor component:

csharpCopy code

@page "/fetch-data" @inject HttpClient Http @code { private YourDataType[] data; protected override async Task OnInitializedAsync() { // Replace "your-endpoint-url" with the actual endpoint URL data = await Http.GetFromJsonAsync<YourDataType[]>("your-endpoint-url"); } }

In this example, **YourDataType** should be replaced with the type that matches the data structure you expect to receive from the server. The **GetFromJsonAsync<T>** method is used to send the HTTP GET request and deserialize the JSON response directly into an array of **YourDataType**.

**Note:**

* Ensure that the endpoint URL ("your-endpoint-url") is correct and that the server is configured to accept requests from your Blazor WASM app's origin to avoid CORS (Cross-Origin Resource Sharing) issues.
* If your server requires authentication, make sure to include the necessary authentication headers or tokens with your HTTP requests.
* You might need to add the **System.Net.Http.Json** package to your project to use the **GetFromJsonAsync<T>** extension method, which provides a convenient way to deserialize JSON content from an HTTP response.

Time now spent: 90 more minutes

### NOTES

I used the above to figure out basics of making a client set GET request.  I needed to of course create a new model class, Product.cs, that mimicks the same data class on the server.  That code now looks like this in the FetchData.razor component:  
  
@page "/fetch-data"

@inject HttpClient Http

@code {

    private Product[] data;

    protected override async Task OnInitializedAsync()

    {

        // Replace "your-endpoint-url" with the actual endpoint URL

        data = await Http.GetFromJsonAsync<Product[]>("your-endpoint-url");

    }

}

The solution builds:  NOW I NEED TO change:  
  
"your-endpoint-url"  to an actual endpoint.  I don’t think I have a server-side endpoint defined.  So looking at.

First, lets see what happens when run the app.  OK I get:

In builder.Services.AddSingleton<ProductService>():    
  
A black screen with white text

Description automatically generated

In the server project Program.cs when running.

## ChatGTP INTERACTION 11

### INPUT

When running a Blazor server app in .NET 8.0, I get the following error the Program.cs: System.InvalidOperationException: 'The service collection cannot be modified because it is read-only.' Why?

### OUTPUT

The error message "System.InvalidOperationException: 'The service collection cannot be modified because it is read-only.'" typically occurs when there is an attempt to add services to the **IServiceCollection** after the application's service provider has been built and the services have been locked for modification. This is a common issue when trying to register services outside of the designated startup configuration phase in the **Program.cs** file or in a similar startup configuration file.

In a Blazor Server app with .NET 8.0, services are usually registered and configured within the **Program.cs** file using the **WebApplicationBuilder** pattern. It's important to add services to the **IServiceCollection** before the application's service provider is finalized and the app starts running. Once the **Build** method of the **WebApplicationBuilder** is called, and the app starts running, the service collection becomes read-only, and attempting to add more services will result in the error you encountered.

Here are a few things to check and ensure in your **Program.cs** file:

1. **Service Registration Order**: Make sure all service registrations, including calls to **AddSingleton**, **AddScoped**, **AddTransient**, etc., are placed before the **Build** method of the **WebApplicationBuilder** is called.
2. **Avoid Modifications After Build**: Ensure that no code is attempting to modify the service collection after the application has been built using **builder.Build()** and before the application starts running with **app.Run()**.

Here's an example structure for the **Program.cs** in a Blazor Server app:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container here

builder.Services.AddSingleton<YourService>();

var app = builder.Build();

// Configure the HTTP request pipeline here if needed

app.Run();

In this structure, ensure all service registrations are done before **var app = builder.Build();**. If you have any logic that runs after **app.Run();** and tries to modify the service collection, it will cause the error you're seeing.

If you're still encountering the issue, carefully review your **Program.cs** file and any other configuration files for places where the service collection might be modified after the application has started. It's also helpful to look for any custom startup tasks or middleware that might inadvertently attempt to modify the service collection.

### NOTES

This looks like good advice!  
I just moved the line of code, to before the builder.Build() call.  Makes sense.  Compile/Run:  
  
Now I am here:

Offending line of code in Program.cs, again, the bootstrap process:  app.MapFallbackToPage("/\_Host");

A screenshot of a computer program

Description automatically generated

## CHATGPT INTERACTION 12:

### INPUT

In my Blazor app, I get the following error, how do I fix it? "System.InvalidOperationException: 'Unable to find the required services. Please add all the required services by calling 'IServiceCollection.AddRazorPages' inside the call to 'ConfigureServices(...)' in the application startup code.' "

### OUTPUT

The error message you're encountering indicates that your Blazor application is missing the necessary service registrations for Razor Pages in the **Startup.cs** or **Program.cs** file, depending on your app's structure (especially relevant for .NET 5 and onwards). This can occur in Blazor Server apps that also utilize Razor Pages or MVC features.

To resolve this issue, you should ensure that the **AddRazorPages** method is called on the **IServiceCollection** within your application's startup configuration. Here's how you can do this:

**For .NET Core 3.1 or earlier:**

If your application is using the **Startup.cs** class, make sure to update the **ConfigureServices** method like this:

public void ConfigureServices(IServiceCollection services)

{

    services.AddRazorPages(); // Add this line

    // Include other services like services.AddServerSideBlazor() if it's a Blazor Server app

}

**For .NET 5 or later:**

For applications structured with a **Program.cs** (following the minimal hosting model introduced in .NET 6), you should add the service in the **Program.cs** file:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages(); // Add this line

// Include other services like builder.Services.AddServerSideBlazor() for Blazor Server apps

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

    app.UseExceptionHandler("/Error");

    app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

After adding **AddRazorPages()**, ensure that your application is correctly setting up endpoints for Razor Pages in the **Configure** method (for apps using **Startup.cs**) or within the **app** configuration in **Program.cs**. This typically involves calling **app.UseEndpoints(endpoints => { endpoints.MapRazorPages(); ... });** in .NET Core 3.1 or **app.MapRazorPages();** in .NET 5 or later.

Make sure to rebuild and rerun your application after making these changes. If the issue persists or you encounter other errors, it might be helpful to review the overall configuration and integration of services and components in your Blazor app.

GREG NOTES:  Add the line suggested into Program.cs:    
builder.Services.AddRazorPages(); // Add this line

Time Spent:  35 mins  
  
Now I get the following runtime error (unhandled by server):  
  
**InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }.**

## CHATGPT INTERACTION 13:

### INPUT

In my Blazor .NET 8 application, when running it, I get the following error, how do I fix it? InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }.

### OUTPUT

The error message you're encountering, **InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }**, typically indicates a problem with the routing in your Blazor .NET 8 application. This error often occurs when the application is unable to locate the **\_Host.cshtml** file, which is crucial for a Blazor Server application, as it serves as the main entry point for the app.

Here are some steps you can take to troubleshoot and potentially fix this issue:

1. **Check the \_Host.cshtml File Location:**
   * Ensure that the **\_Host.cshtml** file is located in the **Pages** directory of your Blazor Server project. This is the default location, and moving it elsewhere without proper routing adjustments can lead to the mentioned error.
2. **Examine the Startup.cs or Program.cs Configuration:**
   * In .NET 8, the startup configuration might be in **Program.cs**. Check your app's configuration to ensure it's set up to serve the Blazor app correctly.
   * Ensure you have the endpoint routing for Blazor Server set up in your **Startup.cs** or **Program.cs**. You should have something similar to this:

app.UseEndpoints(endpoints =>

{

    endpoints.MapBlazorHub();

    endpoints.MapFallbackToPage("/\_Host");

});

* + This code snippet is crucial as it sets up the necessary routing for Blazor Server, including the fallback to **\_Host.cshtml**.

1. **Ensure Correct SDK and Runtime:**
   * Make sure you are using the correct SDK and runtime for .NET 8. You can verify this by running **dotnet --version** in your terminal or command prompt.
2. **Check for Typos and Naming Conventions:**
   * Double-check the file name and route values for any typos. The file should be named **\_Host.cshtml** and not have any variations in casing or spelling.
3. **Clean and Rebuild the Solution:**
   * Sometimes, a clean and rebuild can fix issues related to file locations and references. Use the **Clean Solution** and **Rebuild Solution** options in your IDE, or run **dotnet clean** followed by **dotnet build** from the command line.
4. **Check for Routing Conflicts:**
   * Ensure there are no routing conflicts in your application that might be causing the framework to miss the **\_Host.cshtml** file.
5. **Review Changes:**
   * If the application was working before and suddenly stopped, try to recall the most recent changes. Reverting recent changes or examining them can provide clues.

If after trying these steps the issue persists, consider providing more context or code snippets related to how you've configured your Blazor application, as there might be a more specific issue within your app's setup.

## ChatGTP Interaction 14:

### INPUT

In Blazor, .NET 8, what is the proper use of builder.Services.AddRazorPages();

### OUTPUT

In a Blazor application using .NET 8, the **builder.Services.AddRazorPages();** call is used to register Razor Pages services with the application's dependency injection (DI) container. This is essential when your Blazor app incorporates Razor Pages alongside Blazor components, or when you use Razor syntax within Blazor components.

In a typical Blazor Server app, Razor Pages might be used for handling parts of the app that are more document-centric or when you need to mix server-rendered UI logic with client-side Blazor components. For example, the **/\_Host.cshtml** file in a Blazor Server application is itself a Razor Page that hosts the Blazor components.

**Proper Use in Program.cs**

In the minimal hosting model introduced in .NET 6 and continued in .NET 8, you set up your services in the **Program.cs** file. Here's how you might use **AddRazorPages** properly in a Blazor Server app:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages(); // Registers Razor Pages services

builder.Services.AddServerSideBlazor(); // Registers Blazor Server-specific services

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

    app.UseExceptionHandler("/Error");

    app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host"); // Maps the fallback endpoint to \_Host.cshtml

app.Run();

**When to Use AddRazorPages()**

* **Mixing Razor Pages and Blazor**: If your app combines Blazor components with traditional Razor Pages, you'll need to register Razor Pages services to handle the Razor Page requests.
* **Using Razor Syntax**: Even if you're not using standalone Razor Pages, if your Blazor app utilizes Razor syntax within components (**.razor** files) or layout files, you might still need to include **AddRazorPages()** for certain features to work correctly.
* **Custom Routing**: When implementing custom routing or endpoints that rely on Razor Pages, including **AddRazorPages()** ensures that these pages are properly handled by the ASP.NET Core pipeline.

**Considerations**

* **Performance**: Adding services that your application doesn't use can slightly increase memory usage and startup time. Only add services that your application requires.
* **Compatibility**: Ensure that other middleware and configurations in your app are compatible with Razor Pages if you're integrating them into a Blazor app. This includes things like authentication, authorization, and custom middleware that might interact with page routing.

In summary, **AddRazorPages()** is used in Blazor apps primarily when integrating or leveraging Razor Pages functionality alongside Blazor components. It's an essential part of setting up the application's services for scenarios that require Razor Page handling.

## ChatGTP Interaction 15

### INPUT

I get the following error when running my Blazor app, on startup, what is wrong?

### OUTPUT

The error message you're encountering indicates that the Blazor Server application is unable to locate the fallback endpoint, typically defined by the **/\_Host.cshtml** file. This issue can arise due to several reasons, but it often points to problems with the configuration of the application's routing or the physical location of the **/\_Host.cshtml** file within your project.

Here's a checklist to troubleshoot and potentially fix the issue:

**1. Verify the Location of /\_Host.cshtml**

Ensure that the **/\_Host.cshtml** file is placed in the correct location within your project structure. For a standard Blazor Server app, it should be located in the **Pages** directory.

**2. Check Program.cs or Startup.cs Configuration**

Ensure that your application's startup configuration is correctly set up to use Razor Pages and to define the fallback route:

If using **Program.cs** (typical in .NET 6 and later, including .NET 8):

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages(); // Adds Razor Pages services

builder.Services.AddServerSideBlazor(); // Adds Blazor Server services

var app = builder.Build();

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host"); // Maps the fallback route to \_Host.cshtml

app.Run();

If using **Startup.cs** (typical in .NET Core 3.1 and .NET 5):

public void ConfigureServices(IServiceCollection services)

{

    services.AddRazorPages(); // Adds Razor Pages services

    services.AddServerSideBlazor(); // Adds Blazor Server services

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

    // Other middleware configurations...

    app.UseRouting();

    app.UseEndpoints(endpoints =>

    {

        endpoints.MapBlazorHub();

        endpoints.MapFallbackToPage("/\_Host"); // Maps the fallback route to \_Host.cshtml

    });

}

**3. Ensure /\_Host.cshtml is Correctly Configured**

The **/\_Host.cshtml** file should include the correct Blazor Server directives and setup. A typical **/\_Host.cshtml** file looks like this:

@page "/"

@namespace YourAppName.Pages

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>

<html>

...

<body>

    <app>

        <component type="typeof(App)" render-mode="ServerPrerendered" />

    </app>

    <script src="\_framework/blazor.server.js"></script>

</body>

</html>

**4. File Properties**

Check the properties of the **/\_Host.cshtml** file in your solution explorer. Ensure that the "Build Action" is set to "Content" and "Copy to Output Directory" is set to "Do not copy" or "Copy if newer". This ensures the file is correctly included when the application is built and published.

**5. Routing Conflicts**

If your application defines custom routes or uses areas, ensure there's no conflict that might prevent the routing system from resolving the **/\_Host.cshtml** page correctly.

If you've gone through these steps and the issue persists, consider providing more details about your application's structure, any custom configurations, and the exact environment (IDE, .NET version, etc.) you're using for further assistance.

### Notes

My sample project, created from the ChatGTP tutorial and VS 2022 server app creation (without sample pages), has no

/\_Host.cshtml file?  
  
Where should I put it?  
  
Time Spent: Exploring/thinking:  30 mins

## ChatGTP INTERACTION 16:

### INPUT

In what folder for a Razor .NET 8 application, should I place a /\_Host.cshtml file?

### OUTPUT

For a Blazor Server application using Razor in .NET 8, the **/\_Host.cshtml** file should be placed in the **Pages** folder of your project. This is the conventional location for Razor Pages in an ASP.NET Core application, and it allows the framework to correctly locate and serve the **/\_Host.cshtml** file when the application runs.

The **/\_Host.cshtml** file acts as the main entry point for your Blazor Server application, hosting the Blazor components on the server side. It's where the initial HTML page is generated and sent to the client, including the necessary references to Blazor's JavaScript files and any other static assets or content you want to include on the page.

Here's a basic structure of what your project directory might look like:

YourBlazorApp/

│

├── Pages/

│   ├── \_Host.cshtml  // This is where your \_Host.cshtml file should be

│   └── Other Razor Pages (if any)

│

├── wwwroot/

│   └── ... // Static files like CSS, JS, and images

│

├── \_Imports.razor

├── App.razor

└── Program.cs

When setting up your application, ensure that the **/\_Host.cshtml** file is correctly referenced in your application's routing configuration, typically done in the **Program.cs** file for .NET 6 and later versions, including .NET 8. You would usually have a line in your **Program.cs** file that looks something like this:

app.MapFallbackToPage("/\_Host");

This line tells the application to use **/\_Host.cshtml** as the fallback endpoint, meaning that if no other routes match a request, the application will serve this page.

### Notes

So the page as directed, did not build, missing a using statement.  I had to piece together a new \_Host.cshtl page, that looks like this:  
  
@page "/"

@namespace BlazorServerApp

@using BlazorServerApp.Components

@model BlazorServerApp.Components.Pages.\_HostModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>

<html>

...

<body>

    <app>

        <**component** **type**="typeof(App)" **render-mode**="ServerPrerendered" />

    </app>

    <script src="\_framework/blazor.server.js"></script>

</body>

</html>

}

I then run the app and again, get the same error: **An unhandled exception occurred while processing the request.**

InvalidOperationException: Cannot find the fallback endpoint specified by route values: { page: /\_Host, area: }.

SO NOW LETS TRY STACKOVERFLOW:

StackOverflow:

I found one similar question, but predates .NET 8.0 (2019 post).  It did not help.

OK FAIL SO FAR:  REVERTING TO MICROSOFT LEARNING MATERIALS HERE, AFTER this ChatGTP:

## ChatGTP INTERACTION 17:

### INPUT

Can you show me a good .NET 8 tutorial on building my first Blazor app, with server and client logic?

### OUTPUT

For learning how to build your first Blazor application with both server and client logic in .NET 8, I recommend starting with the official Microsoft documentation and tutorials. They offer comprehensive and up-to-date resources that cover various aspects of Blazor development, including setting up your development environment, creating a project, building components, and handling server-client communication.

Here's a step-by-step guide to get you started:

**1. Prerequisites**

Ensure you have the following installed:

* .NET 8 SDK
* A code editor like Visual Studio 2022, Visual Studio Code, or JetBrains Rider

**2. Official Blazor Documentation**

Start with the official Blazor documentation on Microsoft's website:

* **Blazor Overview**: Visit Microsoft's Blazor Overview to understand the basics and architecture of Blazor.
* **Get Started with Blazor**: Follow the Get Started guide for a step-by-step tutorial on setting up your environment and creating a new Blazor project. This guide provides instructions for using Visual Studio, Visual Studio Code, or the .NET CLI.

**3. Building Your First Blazor App**

* **Build a Blazor todo list app**: One of the best ways to learn is by doing. Microsoft provides a tutorial for building a todo list application with Blazor here. This tutorial covers components, event handling, data binding, and more.

**4. Learning Server-Client Logic in Blazor**

* **Blazor Server Documentation**: Dive deeper into Blazor Server-specific features and how to handle server-client communication by reading the Blazor Server documentation.
* **Call a web API from ASP.NET Core Blazor**: Understanding how to interact with a web API is crucial for handling server-client logic. Microsoft provides a guide on this topic here.

**5. Additional Resources**

* **Blazor Workshops**: Microsoft also offers a Blazor workshop, which is a more in-depth and hands-on way to learn Blazor. Check out the Blazor Workshop on GitHub.
* **.NET Learn Path**: For a more structured learning path, explore the Blazor modules on Microsoft's Learn platform: .NET Learn Path.

**6. Community Resources**

* **Awesome Blazor**: Explore the Awesome Blazor GitHub repository, which contains a collection of blogs, courses, samples, and tools for Blazor.

Remember, building a project and experimenting with code while you learn is one of the best ways to understand new concepts. Don't hesitate to modify the examples you work through in tutorials to explore how things work and to make the projects your own.

Time Spent:  30 mins

# PHASE 2:  Need External Learning as Phase 1 Not Successful In Teaching me the “Basics” of Blazor.

[lesson, don’t start with a code migration as way to learn the new target platform].  Must have basic understanding of new platform concepts first.

Expectation:  Once I have this basic understanding, especially of semantics/Blazor architecture, structure, etc.:  Will be productive in migrations.  Will see.

### Notes:

OK, I am Now Doing the Microsoft Tutorial Here:  
  
[Blazor Tutorial | Build your first app (microsoft.com)](https://dotnet.microsoft.com/en-us/learn/aspnet/blazor-tutorial/intro)

Timespent Completed this tutorial in 15  mins

Next is not a Tutorial, but an “Overview” by MS: on the topics I really want to know about.

SO FAR (just using ChatGTP to try to learn, and migrate my very first simple VB APP to Blazor (client/server hybrid app):  
  
VERY FRUSTRATED.  But LEARNING.  
  
I had to abandon the auto-generated “Blazor Tutorial” I asked “it: to build for me:  Now, to get to MS Docs, Learning (still not great, but better), to understand the overall concepts, vision, structure, between x-platform browrser “Razor Components” re-usable (can be c#, hey cool, not just JavaScript):  and Data Models and general interaction with server-side URIs/lets call them REST endpints.  
  
I am, frustrated, with ChatGTP:  but feel once I hit the “breakthrough” in my understanding:    
  
Translating code from VB Win Forms (.NET), to Blazor, will become very productive.  BUT I DO NOT KNOW YET!

SPENT 1 Day Going Through Blazor Tutorial, up to Pizza App which is only available for .NET 6.  So, trying to piece together from MS Blazor Sample for .NET 8.0  
  
Opened the .NET 8.0 BlazorWebApp Sample in VS2022.  
  
This builds and runs successfully, and has a ton of useful samples.  Is this enough to now go back to my VB Win Forms app, and start migrating?  Let’s see.

Time spent:  1 Day in tutorials on Microsoft, to learn basic Blazor concepts.  8 hours

OK NOW: Refer back to main document, as my new starting point. Got what I needed, rest of way will be main flow, via ChatGTP migration.

I never used the new BlazorSamples (for .NET 8.0). FYI. Though would likely provide useful if extending the project and learning more advanced Blazor 8 concepts.