GROWTH ACCELERATION PARTNERS

We Migrated a Legacy App with ChatGPT: Here's What Happened

The good, the bad and the ugly of migrating a VB.NET app to C# Blazor using generative AI

by DeeDee Walsh, GAP's VP of Developer Marketing & Business Development
Posted on Mobilize.Net on April 21, 2024

We've all asked ourselves the same question: can generative AI migrate an old legacy application? The answer is "yes," but there are a lot of caveats.

In this post, we'll share our experiences and learnings from using ChatGPT 4.0 to migrate a legacy VB.NET Winforms application to a modern C#/Blazor web application. The project aimed to use the power of generative AI's natural language processing and code generation capabilities to streamline the migration process.

GROWTH ACCELERATION PARTNERS

Tech Specs

The source application was a .NET 4.8/Entity Framework 6.4.4 VB.NET WinForms app developed in Visual Studio 2022. The target application was .NET 8/Entity Framework Core 8.0.2 C#/Blazor Server web app, also in VS2022.

What We Learned

Form Mapping and Understanding

Translating VB.NET WinForms to a Blazor web app required more than just code conversion. It involved understanding the structure, features and complexities of the source application, defining the scope of the migration and planning the architecture of the target Blazor application.

Migration Approach

We approached the migration in these phases:

- Analyzing and planning
- Setting up the Blazor project
- Migrating business logic
- Developing the UI with Blazor components
- Handling data access and state management
- Implementing authentication and authorization
- Testing and deployment
- Training and documentation

Factoring the Application

The VB.NET application lacked proper separation of concerns, with business logic and data access mixed in the UI layer. For an effective migration, it was crucial to refactor the application into distinct layers - UI, business logic and data access.

Iterative Migration and Learning Curve

The migration was an iterative process, requiring frequent adjustments and learning. As we gained more understanding of Blazor and .NET 8, the code generated by ChatGPT needed to be fine-tuned. The learning curve involved adapting to new concepts and best practices in C#, Blazor and Entity Framework Core.

Areas Where ChatGPT Exceeded Expectations

Code Translation:

ChatGPT (and generative AI in general) demonstrated impressive capabilities in translating VB.NET code to C#, handling complex constructs and providing syntactically correct code. Here are a few examples:

Business Logic Migration

Before (VB.NET):

```
Public Function Login(userid As String, password As String) As AccountCurrent
    Dim theAccount As New AccountCurrent
    theAccount.UserID = userid
    Using accountcontext As New ACCOUNTDBEntities()
        Dim parameter = New SqlParameter("@userId", userid)
        Dim result = accountcontext.ACCOUNTPROFILEs.SqlQuery("SELECT * FROM
dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter)
        For Each profile In result
            theAccount.UserID = profile.USERID
            theAccount.Password = profile.PASSWORD
            theAccount.AccountID = profile.ACCOUNTID
            theAccount.Salt = profile.SALT
            theAccount.Name = profile.FULLNAME
        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
```

After (C#)

```
public async Task<AccountCurrent> LoginAsync(string userId, string password)
{
    var theAccount = new AccountCurrent();
    using (var accountContext = new ACCOUNTDBEntities())
    {
        var parameter = new SqlParameter("@userId", userId);
        var accountProfiles = await accountContext.ACCOUNTPROFILEs
            .FromSqlRaw("SELECT * FROM dbo.ACCOUNTPROFILE WHERE USERID =
@userId", parameter)
            .ToListAsync();
        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;
        }
        if (theAccount.Password == null)
        {
            GlobalSettings.LoggedIn = false;
        }
        else
        {
            var verifier = SaltedHash.Create(theAccount.Salt,
theAccount.Password);
            GlobalSettings.LoggedIn = verifier.Verify(password);
        }
    }
    return theAccount;
}
```

UI Migration

Before (VB.NET)

```
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
```

After (Blazor):

```
@page "/"
@inject NavigationManager NavigationManager
@inject IAuthenticationService AuthenticationService

<h3>Welcome</h3>

@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()
    {
        NavigationManager.NavigateTo("/form1");
    }

    private void ToggleLogin()
    {
        if (AuthenticationService.IsLoggedIn)
        {
            AuthenticationService.Logout();
        }
        else
        {
            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
        }
    }
}
```

These examples show how ChatGPT translated the code from VB.NET to C#, handling differences in syntax, control flow and UI paradigms.

Best Practices:

- **Identifying best practices:** ChatGPT often suggested best practices and modern approaches, such as using async/await, dependency injection and separating concerns, even when the source code didn't follow these patterns.
- **Providing explanations:** Along with generating code, ChatGPT offered clear explanations and reasoning behind its suggestions, helping us understand the rationale and learn new concepts.
- **Handling different target platforms:** ChatGPT was able to generate code targeting different platforms including Node.js/Angular, showcasing its versatility.
- **Database class generation:** Effective creation of database models from DDL.
- **Debugging support:** Played a significant role in resolving code issues.

Limitations of Generative AI

While ChatGPT was helpful in generating code snippets and providing guidance, it had limitations. The generated code often needed manual adjustments to align with the latest .NET versions and Blazor best practices.

Here are a few examples we encountered:

Entity Framework Core Usage:

ChatGPT generated the following code for querying the database using Entity Framework Core:

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

However, this code uses the `SqlQuery` method, which is not available in Entity Framework Core. The code needed to be manually adjusted to use the `FromSqlRaw` method instead:

```
var accountProfiles = _accountContext.ACCOUNTPROFILEs
    .FromSqlRaw("SELECT * FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId",
parameter)
    .ToList();
```

Asynchronous Code:

ChatGPT sometimes generated synchronous code, even when asynchronous operations were required. For example:

```
var result = _accountContext.ACCOUNTPROFILEs.FromSqlRaw("SELECT * FROM
dbo.ACCOUNTPROFILE WHERE USERID = @userId", parameter).ToList();
```

This code needed to be manually adjusted to use asynchronous methods and the `await` keyword:

```
var accountProfiles = await _accountContext.ACCOUNTPROFILEs
    .FromSqlRaw("SELECT * FROM dbo.ACCOUNTPROFILE WHERE USERID = @userId",
parameter)
    .ToListAsync();
```

Blazor Component Lifecycle Methods:

ChatGPT generated code that used incorrect lifecycle methods in Blazor components. For example:

```
protected override void OnInitialized()
{
    // Code to fetch data
}
```

This code needed to be manually adjusted to use the correct asynchronous lifecycle method, `OnInitializedAsync`:

```
protected override async Task OnInitializedAsync()
{
    // Code to fetch data asynchronously
}
```

Incorrect Namespace References:

ChatGPT sometimes generated code with incorrect namespace references. For example:

```
using System.Data.Entity;
```

This namespace is used in Entity Framework 6 but not in Entity Framework Core. The code needed to be manually adjusted to use the correct namespace:

```
using Microsoft.EntityFrameworkCore;
```

These examples illustrate some of the situations where the code generated by ChatGPT required manual intervention to align with the specific requirements, best practices and APIs of the target platform (.NET 8 and Blazor in this case.)

In addition to the having to make manual adjustments to the code, we encountered some other issues:

Can't Get Holistic View of the Project:

Some other limitations we encountered with the lack of context awareness, is that there's no way to look holistically at your app in current generative AI services. It's a very piecemeal process. We had to paste code snippets into ChatGPT, then copy the code to Visual Studio. Over and over again. It's a piecemeal process.

Lack of Consistency in its Approach:

Each ChatGPT response provided solutions that differed from the previous attempt. And sometimes the recommended steps to follow would vary from session to session so it would take some experimenting to find the right approach.

Must Consistently Reset ChatGPT Context:

As a developer, it's on you to review, understand and adapt the generated code to ensure it fits into your project and adheres to your team's standards and practices. For example the most tedious part of migrating using ChatGPT was that we had to set context every single time we migrated code. Target queries have to be set to precise versions of .NET, Blazor and Entity Framework targets. Sometimes in the SAME chat session.

Can't Generate New VS Solutions or Projects:

None of the services can create Visual Studio solutions or projects so it requires a manual cut and paste process. Cut and paste over and over.

Wrong Direction:

Some advice was wrong on migration approach. For example, ChatGPT recommended we start with business logic layer instead of DB tier/model classes. It's also important to point out that while ChatGPT can generate code that is syntactically correct and follows general programming patterns, it may not always be aware of the specific nuances, versions or best practices of the target platform.

Importance of Understanding the Target Platform

Attempting to migrate the application without a solid understanding of Blazor and .NET 8 proved challenging. Taking the time to learn the basics of the target platform through tutorials and documentation (generative AI was not a great tutor in this scenario) was crucial for making informed decisions and effectively using ChatGPT.

Recommendations

So, we figured we'd roll up our sleeves and let ChatGPT handle this VB.NET WinForms to Blazor migration. After all, AI's the future... amirite? Well, we quickly learned that understanding the goal is just as important as knowing where you're starting from. Turns out, blindly diving into Blazor and .NET 8 without some groundwork was a recipe for confusion.

Generative AI is awesome, but for this, those tutorials and docs were our best teachers. Once we grasped the basics of Blazor, making decisions with ChatGPT made way more sense.

Lessons from GAP's ChatGPT Code Migration Adventure

ChatGPT 4.0 was a trip! We definitely pushed its limits (and ours) during this migration. Here's what we'd tell anyone else thinking of giving it a go:

1. Don't wing it... analyze first!

Can generative AI even handle your project? Figure out if it's a good fit before jumping in. Think about how complex your app is, what you want it to become, and the time and people you have on hand.

2. Plans are your friend.

Don't just code-by-chaos! Outline everything: analyzing your old codebase, setup, the migration itself, the whole nine yards. This keeps you (and ChatGPT) on track.

3. The target tech matters.

Before you let the AI loose, take a crash course in the tech you're aiming for (like Blazor and .NET 8 for us). This makes ChatGPT's suggestions way more useful.

4. First drafts are just that.

AI-generated code isn't gospel. You'll still need to review, tweak, and make sure it fits your project perfectly.

5. Humans + AI = Dream Team

ChatGPT is a powerful tool, but developers are still the stars of the show. Use AI to speed things up, but lean on your team's expertise for the big-picture decisions.

6. Notes are your memory.

Record everything — challenges, fixes, what worked, and what didn't. Oh, and those prompts you fed ChatGPT? Save those too! Trust us, this knowledge is gold for later.

So, did ChatGPT turn us into Blazor migration masters overnight? Nope. But did it make the process a heck of a lot more interesting? Absolutely! We learned a ton, and honestly, we're excited to see where generative AI takes us next. The key takeaway? It's still the human behind the keyboard that makes all the difference. AI might generate the code, but we bring it to life.

Dee Dee Walsh, GAP's VP of Developer Marketing & Business Development, is a software and tech veteran who has spent decades building developer communities in the Microsoft, open source and Snowflake ecosystems.

ABOUT GROWTH ACCELERATION PARTNERS:

At GAP, we consult, design, build and modernize revenue-generating software and data engineering solutions for clients. With modernization services and AI tools, we help businesses achieve a competitive advantage through technology.

To find out more, please visit WeAreGAP.com

company/growth-acceleration-partners/ @GrowthAccelerationPartners @GAPapps