

# Membangun ChatBot dengan Blazor Hybrid dan Azure Open AI – Part II

**Junindar, ST, MCPD, MOS, MCT, MVP**

***Lisensi Dokumen:***

*Copyright © 2003 IlmuKomputer.Com*

*Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarakan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.*

*junindar@gmail.com*

<http://junindar.blogspot.com>

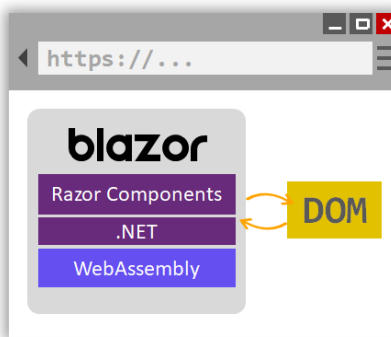
## Abstrak

Blazor Hybrid adalah pendekatan inovatif yang menggabungkan framework Blazor dengan .NET MAUI (Multi-platform App UI). Yang memungkinkan kita sebagai *developer* untuk membangun aplikasi *cross platform* menggunakan teknologi web yang sudah dikenal. Dalam aplikasi Blazor Hybrid, komponen Razor berjalan secara native di perangkat dan dirender ke kontrol Web View yang tertanam melalui local interop. Sehingga komponen ini tidak berjalan di browser dan tidak melibatkan WebAssembly.

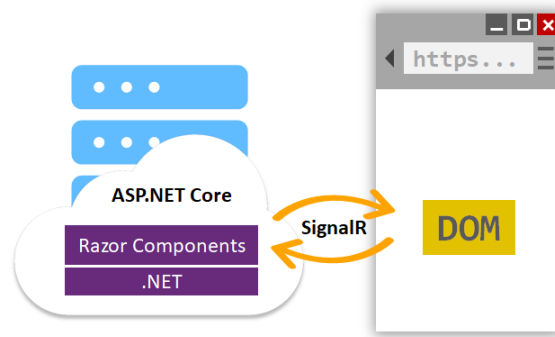
## Pendahuluan

Blazor adalah Web Framework yang bersifat Open Source dimana aplikasi Web yang bersifat client-side interactive dapat dikembangkan dengan menggunakan .Net (C#) dan HTML. Pada saat ini C# biasa digunakan untuk melakukan proses back-end dari aplikasi web. Dengan menggunakan fitur baru dari ASP.NET Core yaitu Blazor, kita dapat membangun interactive WEB dengan menggunakan C# dan .NET .

Code .Net berjalan pada Web Assembly, yang artinya kita dapat menjalankan “NET“ didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java maupun Flash.

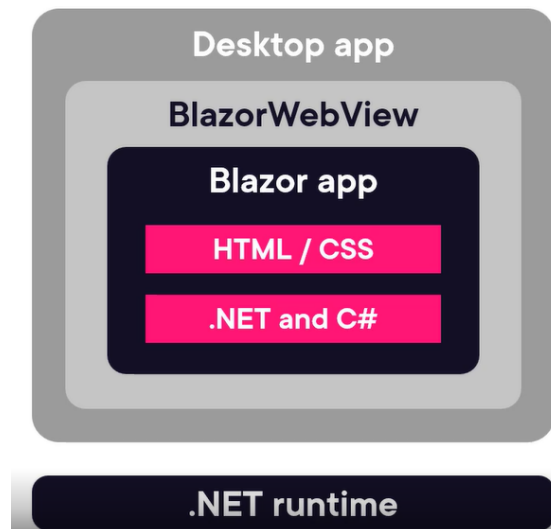


Dengan menggunakan Blazor kita dapat memilih apakah menggunakan WebAssembly (Client Side), seperti yang telah dijelaskan di atas atau Blazor yang dijalankan diatas server. Dengan menggunakan cara kedua kita memerlukan SignalR untuk menghubungkan antara client (browser) dan server app. Sebagai contoh jika user melakukan proses klik button pada browser, maka data akan dikirimkan ke Server menggunakan SignalR dan hasilnya akan dikembalikan ke client dengan mengupdate DOM pada client.



Aplikasi Blazor menggunakan runtime .NET yang didasarkan pada Web Assembly atau disingkat WASM. WASM didukung secara native oleh mayoritas browser.

Blazor Hybrid adalah pendekatan inovatif yang menggabungkan framework Blazor dengan .NET MAUI (Multi-platform App UI). Yang memungkinkan kita sebagai *developer* untuk membangun aplikasi *cross platform* menggunakan teknologi web yang sudah dikenal. Dalam aplikasi Blazor Hybrid, komponen Razor berjalan secara native di perangkat dan dirender ke kontrol Web View yang tertanam melalui local interop. Sehingga komponen ini tidak berjalan di browser dan tidak melibatkan WebAssembly.

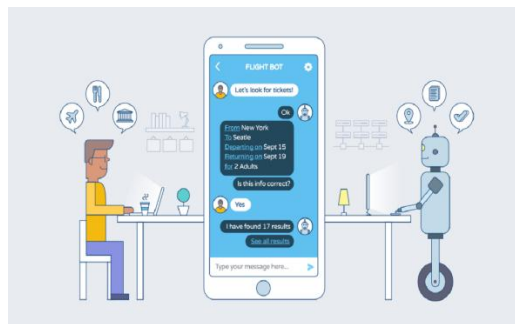


Disarankan untuk membaca dan menyelesaikan latihan pada ebook dari penulis tentang Blazor Hybrid pada tautan berikut. <https://tinyurl.com/4t3d9bw6>

## Chatbot

Chatbot atau chatterbot adalah sebuah layanan obrolan robot/tokoh virtual dengan kecerdasan buatan atau AI (Artificial Intelligent) yang menirukan percakapan manusia melalui pesan suara, obrolan teks ataupun keduanya.

Pada dasarnya bots bekerja dengan cara melihat kata kunci dalam data yang masuk dan membalasnya dengan kata kunci yang paling cocok, atau pola kata-kata yang paling mirip dari basis data tekstual. Artinya, jika pengguna mengirim suatu permintaan maka bots akan membalasnya dengan respon yang spesifik sesuai dengan kata kunci yang dikirim.



**Gambar**

Sebagai program komputer yang dirancang untuk berinteraksi dengan manusia melalui antarmuka percakapan, seperti pesan teks atau suara. Chat bot menggunakan kecerdasan buatan dan pemrosesan bahasa alami untuk memahami pertanyaan atau perintah yang diberikan oleh pengguna, lalu memberikan respons atau melakukan tindakan yang sesuai berdasarkan pemrograman atau algoritma yang telah ditetapkan.

Chat bot dapat digunakan untuk berbagai tujuan, seperti memberikan informasi, menjawab pertanyaan pengguna, melakukan reservasi, memberikan layanan pelanggan, atau bahkan hanya untuk hiburan. Mereka dapat diimplementasikan dalam berbagai platform, termasuk situs web, aplikasi seluler, layanan pesan instan, dan banyak lagi.

## Microsoft Azure

Microsoft Azure adalah platform komputasi awan yang disediakan oleh Microsoft. Ini adalah salah satu penyedia layanan komputasi awan terkemuka di dunia. Azure menyediakan berbagai layanan cloud yang mencakup infrastruktur sebagai layanan

(Infrastructure as a Service/IaaS), platform sebagai layanan (Platform as a Service/PaaS), dan perangkat lunak sebagai layanan (Software as a Service/SaaS).

Azure memungkinkan pengguna untuk menyimpan data, menjalankan aplikasi, dan mengelola sumber daya IT di lingkungan cloud, tanpa perlu membangun dan memelihara infrastruktur fisik sendiri. Platform ini menawarkan berbagai fitur dan layanan termasuk komputasi awan, penyimpanan awan, basis data awan, kecerdasan buatan, Internet of Things (IoT), analisis data, keamanan, dan banyak lagi.

### **Azure OpenAI Services**

Azure OpenAI Services adalah sejumlah layanan yang disediakan oleh Microsoft Azure yang memungkinkan pengembang untuk mengintegrasikan teknologi kecerdasan buatan yang dikembangkan oleh OpenAI ke dalam aplikasi mereka. Ini mencakup layanan seperti Azure Cognitive Services, yang menyediakan berbagai fitur AI dan Machine Learning, serta integrasi khusus dengan model-model canggih yang dikembangkan oleh OpenAI untuk tugas-tugas seperti pemrosesan bahasa alami, pengenalan gambar, dan lainnya.

Dengan menggunakan Azure OpenAI Services, pengembang dapat dengan mudah memanfaatkan kemampuan AI dan Machine Learning tanpa perlu mengembangkan model mereka sendiri dari awal. Ini mempercepat proses pengembangan aplikasi yang cerdas dan membantu dalam menciptakan solusi yang lebih canggih untuk berbagai macam masalah bisnis dan teknis.

Azure OpenAI Service menyediakan akses REST API ke model OpenAI yang powerfull termasuk seri model GPT-4, GPT-4 Turbo dengan Vision, GPT-3.5-Turbo. Selain itu, seri model GPT-4 dan GPT-3.5-Turbo baru kini telah tersedia secara umum. Model ini dapat dengan mudah disesuaikan dengan tugas spesifik namun tidak terbatas pada pembuatan konten, ringkasan, pemahaman gambar, pencarian semantik, dan terjemahan natural language ke kode.

Pada artikel ini tidak menjelaskan apa itu blazor hybrid, bagaimana bekerja dengan project maupun cara-cara untuk menambahkan item pada project, karena semuanya telah dijelaskan baik pada ebook maupun artikel-artikel sebelumnya. Pastikan anda telah

menyelesaikan latihan-latihan pada artikel maupun ebook tersebut. Artikel ini akan fokus membuat chatbot dengan menggunakan Azure Open AI pada Blazor Hybrid.

Sebelum kita mulai dengan membuat project pada Blazor Hybrid, pastikan kita telah memiliki Azure Account terlebih dahulu. Jika belum memiliki Azure Account, kita dapat membuatnya terlebih dahulu disini <https://azure.microsoft.com/en-us/free/> .

Pada latihan sebelumnya, kita telah berhasil membangun sebuah aplikasi chatbot sederhana yang memanfaatkan layanan OpenAI. Setiap kali pengguna mengajukan sebuah pertanyaan, chatbot tersebut akan meneruskannya ke model yang telah kita konfigurasi, lalu menampilkan balasan yang dihasilkan oleh model tersebut. Proses ini memberikan gambaran dasar mengenai bagaimana cara berkomunikasi dengan OpenAI melalui aplikasi kita.

Meskipun fungsionalitas dasar tersebut sudah berjalan dengan baik, aplikasi pada artikel sebelumnya masih memiliki beberapa kekurangan penting. Salah satunya adalah tidak adanya fitur untuk menyimpan riwayat percakapan. Sebagian besar chatbot modern baik yang ada di website, aplikasi mobile, maupun platform komunikasi lainnya menyediakan kemampuan untuk melihat kembali percakapan sebelumnya. Fitur ini sangat berguna karena pengguna sering kali ingin meninjau jawaban atau informasi yang telah diberikan sebelumnya tanpa perlu mengulang pertanyaan yang sama.

Sayangnya, aplikasi yang kita buat sebelumnya belum memiliki kemampuan tersebut. Setiap percakapan hanya disimpan sementara dalam memori selama aplikasi berjalan. Ketika aplikasi ditutup dan dibuka kembali, seluruh percakapan akan hilang. Hal ini tentu bisa menjadi masalah, terutama apabila percakapan tersebut berisi informasi penting yang ingin kita simpan atau rujuk kembali di kemudian hari. Bayangkan jika pengguna bertanya sesuatu yang kompleks dan mendapatkan jawaban panjang dari chatbot, lalu setelah aplikasi ditutup, jawaban tersebut tidak bisa dilihat lagi. Ini jelas mengurangi kenyamanan dan efektivitas penggunaan aplikasi.

Oleh karena itu, pada latihan dalam artikel ini kita akan meningkatkan aplikasi chatbot dengan menambahkan fitur penyimpanan percakapan. Dengan adanya fitur ini, setiap pertanyaan dan jawaban akan disimpan secara permanen (misalnya dalam database atau file), sehingga dapat ditampilkan kembali saat aplikasi dibuka. Dengan cara ini, chatbot

akan terasa lebih profesional, lebih berguna, dan lebih mendekati standar aplikasi chatbot pada umumnya.

Dalam proses pengembangan fitur ini, kita juga akan mempelajari konsep-konsep penting seperti manajemen sesi, struktur data percakapan, dan cara menyimpan serta memuat data secara efisien. Hal ini tidak hanya menambah fungsi aplikasi, tetapi juga memperdalam pemahaman kita dalam membangun sistem yang lebih realistis dan siap digunakan.

Mari kita mulai latihan ini dengan mengikuti langkah-langkah yang akan dijelaskan secara bertahap.

Note : Untuk memudahkan pemahaman terhadap kode atau sintaks yang akan kita gunakan, sangat disarankan untuk membuat proyek baru terlebih dahulu dan menyalin seluruh kode dari artikel sebelumnya. Pastikan proyek tersebut berjalan dengan baik sebelum melanjutkan, agar kita dapat fokus pada penambahan fitur baru tanpa terganggu oleh kesalahan dari konfigurasi sebelumnya.

- Buat sebuah Interface dengan nama “IChatHistoryService“ seperti dibawah.

```
public interface IChatHistoryService
{
    Task SaveHistoryAsync(List<ChatMessage> messages);
    Task<List<ChatMessage>> LoadHistoryAsync();
    void ClearHistory();
}
```

- Lalu kita lanjutkan dengan membuat class-nya, sebagai implementasi dari Interface diatas.

```
public class ChatHistoryService : IChatHistoryService
{
    private readonly string historyFilePath;

    public ChatHistoryService()
    {
        historyFilePath = Path.Combine(FileSystem.AppDataDirectory, "chat_history.json");
    }
}
```

Alasan utama kita memilih AppDataDirectory sebagai lokasi penyimpanan riwayat percakapan adalah karena direktori ini bersifat khusus untuk aplikasi (per-aplikasi) dan tidak bersifat global. Artinya, setiap aplikasi akan memiliki foldernya sendiri yang terisolasi dari aplikasi lain. Hal ini memberikan beberapa keuntungan. Pertama, data yang kita simpan tidak akan bercampur dengan data milik aplikasi lain, sehingga mengurangi risiko konflik nama file, korupsi data, atau akses tidak sengaja

dari program lain. Kedua, pendekatan ini sejalan dengan konsep sandboxing, yaitu mekanisme yang membatasi ruang gerak aplikasi agar tidak mengakses bagian sistem yang tidak seharusnya. Banyak sistem operasi modern seperti Android, iOS, maupun Windows Store App menerapkan sandboxing demi keamanan dan privasi pengguna.

Selain itu, direktori AppDataDirectory juga telah dirancang khusus oleh sistem operasi untuk menyimpan data internal aplikasi, seperti konfigurasi, cache, atau file riwayat yang tidak ditujukan untuk dilihat atau diedit langsung oleh pengguna. Karena direktori ini berada dalam area “data aplikasi”, sistem operasi biasanya memberikan izin tulis secara otomatis tanpa perlu konfigurasi tambahan. Dengan kata lain, kita dapat menyimpan dan membaca file di sini dengan aman tanpa harus meminta izin spesial dari pengguna (misalnya izin akses penyimpanan eksternal), yang sering menjadi kendala pada platform seperti Android.

Penggunaan AppDataDirectory juga memberikan kejelasan tujuan penyimpanan. File yang berada di direktori ini dianggap sebagai bagian dari ekosistem internal aplikasi bukan sebagai dokumen publik seperti gambar, PDF, atau file unduhan. Oleh karena itu, direktori ini ideal digunakan untuk menyimpan file-file yang bersifat internal, seperti:

- Pengaturan aplikasi (configuration settings),
- Cache atau temporary data,
- Riwayat percakapan pengguna,
- Metadata atau data pendukung lainnya.

Dengan memilih pendekatan ini, kita tidak hanya mengikuti praktik terbaik dalam pengembangan perangkat lunak modern, tetapi juga memastikan bahwa aplikasi kita lebih aman, lebih stabil, mudah dipelihara, dan tetap mematuhi standar platform.

```
public async Task SaveHistoryAsync(List<ChatMessage> messages)
{
    try
    {
        var json = JsonSerializer.Serialize(messages,
            new JsonSerializerOptions { WriteIndented = true });
        await File.WriteAllTextAsync(historyFilePath, json);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Gagal menyimpan history chat: {ex.Message}");
    }
}
```

Metode SaveHistoryAsync bertugas menyimpan riwayat percakapan chatbot ke dalam file. Metode ini bersifat asynchronous (async Task) agar proses penulisan file tidak mengganggu performa aplikasi. “messages” (List<ChatMessage>) terlebih dahulu diubah menjadi format JSON menggunakan JsonSerializer.Serialize dengan opsi WriteIndented = true supaya hasilnya rapi dan mudah dibaca.

Hasil JSON tersebut kemudian disimpan ke file menggunakan File.WriteAllTextAsync pada path yang sudah ditentukan sebelumnya. Seluruh proses diletakkan dalam blok try-catch untuk mencegah aplikasi crash jika terjadi error, seperti kegagalan akses file atau masalah penyimpanan. Jika terjadi kesalahan, pesan error akan ditampilkan melalui Console.WriteLine.

```
public async Task SaveHistoryAsync(List<ChatMessage> messages)
{
    try
    {
        var json = JsonSerializer.Serialize(messages,
            new JsonSerializerOptions { WriteIndented = true });
        await File.WriteAllTextAsync(historyFilePath, json);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Gagal menyimpan history chat: {ex.Message}");
    }
}
```

Sedangkan untuk method LoadHistoryAsync digunakan untuk membaca dan memuat kembali riwayat percakapan chatbot dari file JSON. Pertama, metode memeriksa apakah file riwayat (historyFilePath) benar-benar ada menggunakan File.Exists. Jika file belum pernah dibuat, metode langsung mengembalikan List<ChatMessage> kosong agar aplikasi tetap berjalan normal. Jika file ada, isinya dibaca secara asynchronous menggunakan File.ReadAllTextAsync, kemudian teks JSON tersebut diubah kembali (deserialize) menjadi objek List<ChatMessage> menggunakan

JsonSerializer.Deserialize. Operator ?? digunakan untuk memastikan bahwa jika hasil deserialization adalah null, tetap dikembalikan list kosong.

```
public void ClearHistory()  
{  
    if (File.Exists(historyFilePath))File.Delete(historyFilePath);  
}
```

Metode ClearHistory digunakan untuk menghapus seluruh riwayat percakapan yang sebelumnya disimpan dalam file. Proses ini dimulai dengan memeriksa apakah file riwayat benar-benar ada menggunakan File.Exists(historyFilePath). Pengecekan ini penting untuk mencegah error, karena jika kita mencoba menghapus file yang tidak ada, operasi File.Delete dapat menyebabkan exception. Dengan memastikan file benar-benar tersedia, metode ini menjadi lebih aman.

Jika file ditemukan, maka File.Delete(historyFilePath) akan dijalankan untuk menghapus file tersebut dari penyimpanan. Setelah file dihapus, riwayat percakapan dianggap bersih atau di-reset sepenuhnya.

Lalu register interface dan class diatas pada “MauiProgram.cs”, seperti pada gambar dibawah ini.

```
builder.Services.AddHttpClient<IChatService, ChatService>();  
builder.Services.AddSingleton<IChatHistoryService, ChatHistoryService>();  
return builder.Build();
```

Selanjutnya buka file Home.razor, pada file ini kita akan menambahkan sebuah button untuk menghapus history dari chatbot. Ketikkan seperti pada sintaks dibawah ini.

```
<div class="card-footer bg-light rounded-bottom">  
    <button class="btn btn-outline-danger btn-sm mb-3" @onclick="ClearHistory">  
        Clear History  
    </button>  
</div>
```

Dilanjutkan dengan membuat method “ClearHistory” seperti dibawah.

```
private async Task ClearHistory()  
{  
    historyService.ClearHistory();  
    messages.Clear();  
    await InvokeAsync(StateHasChanged);  
}
```

Metode “ClearHistory()” ini bertugas menghapus riwayat percakapan baik dari penyimpanan lokal maupun dari memori aplikasi. Baris “historyService.ClearHistory();” memanggil service untuk menghapus file riwayat yang tersimpan, sedangkan “messages.Clear();” mengosongkan “messages” yang

sedang ditampilkan di UI. Terakhir, “await InvokeAsync(StateHasChanged);” memastikan antarmuka pengguna diperbarui agar perubahan segera terlihat oleh pengguna.

```
protected override async Task OnInitializedAsync()
{
    messages = await historyService.LoadHistoryAsync();
}

protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        await ScrollToBottom();
    }
}
```

Dua method diatas digunakan untuk mengatur behaviour dari komponen saat pertama kali ditampilkan. Pada “OnInitializedAsync()”, komponen memuat riwayat percakapan dari penyimpanan melalui “historyService.LoadHistoryAsync()” dan menyimpannya ke dalam variabel messages, sehingga data lama langsung ditampilkan ke pengguna. Setelah proses render selesai, “OnAfterRenderAsync()” dijalankan dan jika ini adalah render pertama (firstRender == true), maka metode ScrollToBottom() dipanggil untuk men-scrolling tampilan otomatis ke bagian paling bawah, memastikan pengguna langsung melihat pesan terbaru.

```
private async Task SendMessage()
{
    //Sintaks detail pada project lampiran
}
```

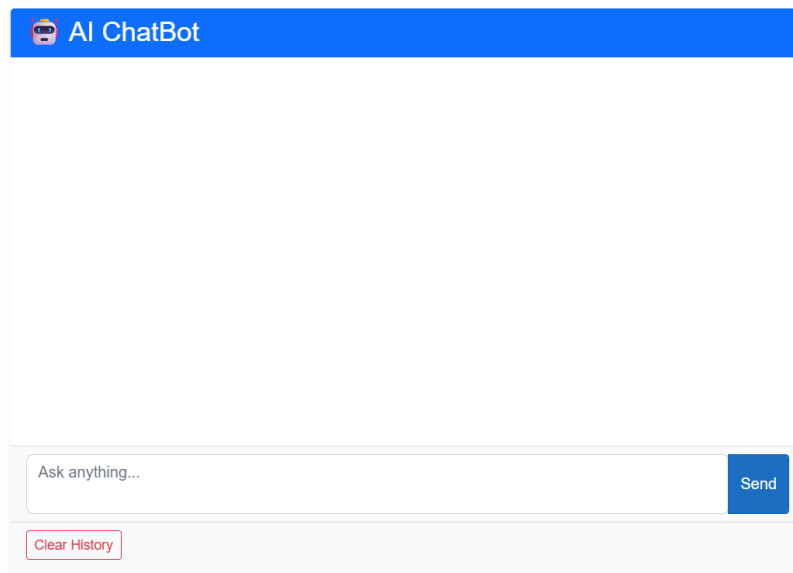
Terdapat beberapa penambahan sintaks pada method SendMessage. Berikut penjelasan detail dari sintaks yang ada pada method SendMessage.

```
await historyService.SaveHistoryAsync(messages);
await InvokeAsync(StateHasChanged);
await ScrollToBottom();
```

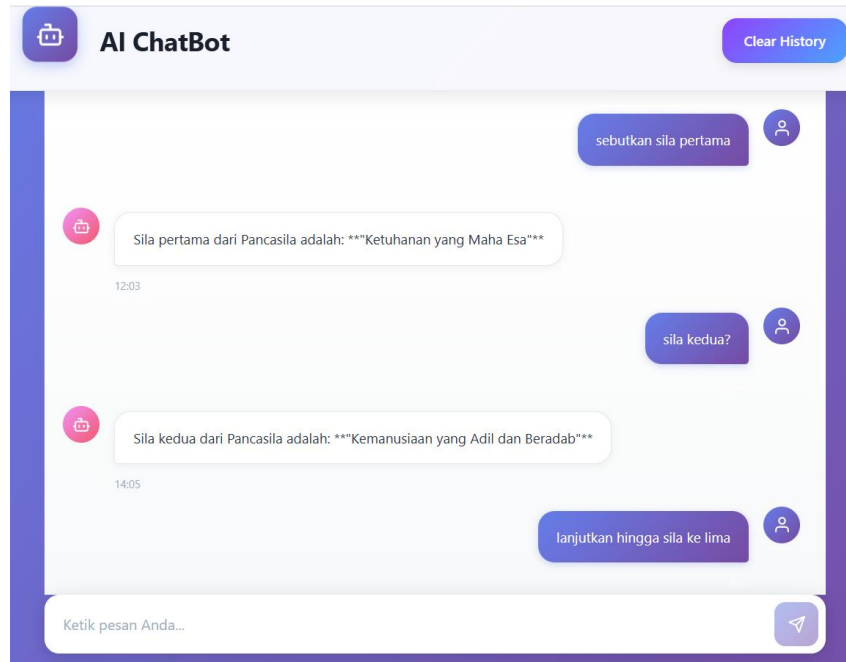
Potongan kode ini menjalankan tiga method penting, setelah pesan baru ditambahkan dalam aplikasi chat. Pertama, “await historyService.SaveHistoryAsync(messages)” digunakan untuk menyimpan seluruh “messaging” saat ini ke penyimpanan local (json file), sehingga riwayat chat tetap tersimpan meskipun aplikasi ditutup. Selanjutnya, “await InvokeAsync(StateHasChanged)” memastikan antarmuka pengguna diperbarui agar pesan terbaru langsung terlihat setelah penyimpanan selesai. Terakhir, “await ScrollToBottom()” untuk men-scrolling ke bagian bawah secara otomatis sehingga pengguna selalu melihat pesan terbaru tanpa perlu menggulir secara manual.

```
try
{
    var reply = await chatService.SendMessageAsync(messages);
    messages.Add(new ChatMessage { Role = "assistant", Content = reply,
        Timestamp = DateTime.Now });
}
catch
{
    messages.Add(new ChatMessage { Role = "assistant", Content = "Terjadi kesalahan.",
        Timestamp = DateTime.Now });
}
finally
{
    isLoading = false;
    userInput = string.Empty;
    await historyService.SaveHistoryAsync(messages);
    await InvokeAsync(StateHasChanged);
    await ScrollToBottom();
}
```

Untuk sintak diatas ini digunakan menangani proses pengiriman pesan ke Open AI Service dan melakukan pembaruan UI. Pada blok **try**, aplikasi mengirim seluruh riwayat chat ke `chatService.SendMessageAsync`, lalu menambahkan balasan AI ke "messages". Jika terjadi error, blok **catch** menambahkan pesan kesalahan dari asisten agar pengguna tetap mendapat respons. Pada blok **finally**, status loading dimatikan, input pengguna dikosongkan, riwayat chat disimpan, UI diperbarui, dan tampilan otomatis digulir ke pesan terbaru untuk menjaga pengalaman pengguna tetap lancar. Jalankan program dan pastikan mendapatkan hasil seperti dibawah. lakukan percakapan ke Chatbot, lalu matikan program dan hidupkan Kembali. Pastikan history percakapan masih ada pada saat program dihidupkan.



Jika ingin mendapatkan hasil seperti gambar dibawah, agar kelihatan lebih profesional. Sintaksnya dapat dilihat pada project lampiran. Yang banyak berubah adalah dari sisi UI-nya saja. Sedangkan back-end atau C#-nya hanya beberapa bagian saja yang berubah.

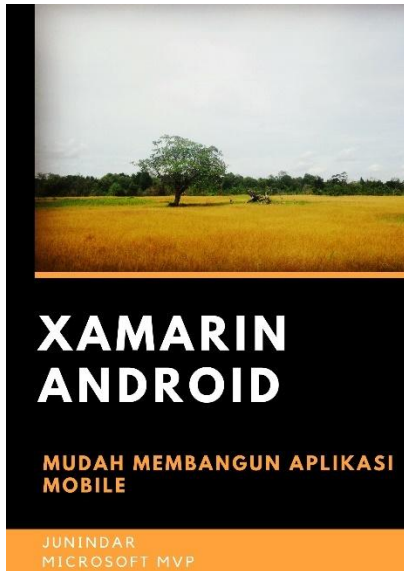


## **Penutup**

Sedangkan untuk memudahkan dalam memahami isi artikel, maka penulis juga menyertakan dengan full source code project latihan ini, dan dapat di download disini

<https://junindar.blogspot.com/2025/10/pada-latihan-sebelumnya-kita-telah.html>

## Referensi



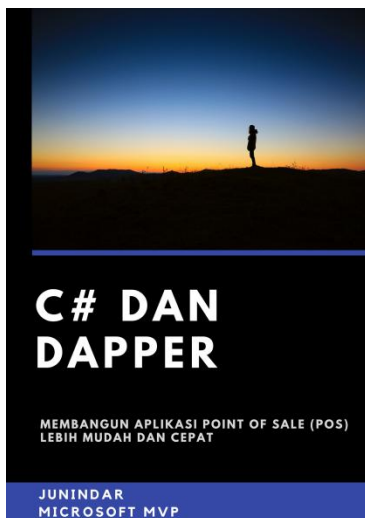
<https://play.google.com/store/books/details?id=G4tFDgAAQBAJ>



<https://play.google.com/store/books/details?id=VSLiDQAAQBAJ>



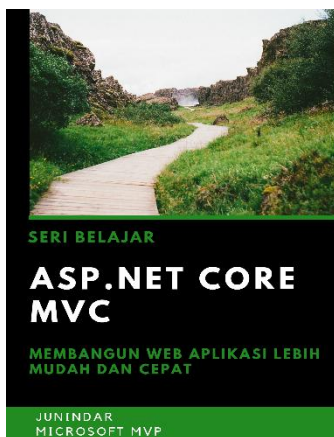
[https://play.google.com/store/books/details/Junindar\\_Xamarin\\_Forms?id=6Wg-DwAAQBAJ](https://play.google.com/store/books/details/Junindar_Xamarin_Forms?id=6Wg-DwAAQBAJ)



[https://play.google.com/store/books/details/Junindar\\_C dan Dapper Membangun Aplikasi POS Point?id=6TErDwAAQBAJ](https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ)



<https://play.google.com/store/books/details/Junindar ASP NET MVC Membangun Aplikasi Web Lebih?id=XLlyDwAAQBAJ>



<https://play.google.com/store/books/details/Junindar ASP NET CORE MVC?id=x Ee5DwAAQBAJ>

## Biografi Penulis.



Junindar Lahir di Tanjung Pinang, 21 Juni 1982. Menyelesaikan Program S1 pada jurusan Teknik Inscreenatika di Sekolah Tinggi Sains dan Teknologi Indonesia (ST-INTEN-Bandung). Junindar mendapatkan Award Microsoft MVP VB pertanggal 1 oktober 2009 hingga saat ini. Senang mengutak-atik computer yang berkaitan dengan bahasa pemrograman. Keahlian, sedikit mengerti beberapa bahasa pemrograman seperti : VB.Net, C#, SharePoint, ASP.NET, VBA. Reporting: Crystal Report dan Report Builder. Database: MS Access, MY SQL dan SQL Server. Simulation / Modeling Packages: Visio Enterprise, Rational Rose dan Power Designer. Dan senang bermain gitar, karena untuk bisa menjadi pemain gitar dan seorang programmer sama-sama membutuhkan seni. Pada saat ini bekerja di salah satu Perusahaan Consulting dan Project Management di Malaysia sebagai Senior Consultant. Memiliki beberapa sertifikasi dari Microsoft yaitu Microsoft Certified Professional Developer (MCPD – SharePoint 2010), MOS (Microsoft Office Specialist) dan MCT (Microsoft Certified Trainer) Mempunyai moto hidup: **“Jauh lebih baik menjadi Orang Bodoh yang giat belajar, dari pada orang Pintar yang tidak pernah mengimplementasikan ilmunya”**.