SWPSD – 2023/2024

Stworzenie systemu dialogowego w C# można zrealizować za pomocą różnych podejść i bibliotek. Jednym z popularnych sposobów jest użycie Microsoft Bot Framework, który umożliwia tworzenie inteligentnych chatbotów. Poniżej przedstawiam kroki do stworzenia prostego systemu dialogowego w C#.

Krok 1: Instalacja narzędzi

1. **Visual Studio 2019/2022**: Upewnij się, że masz zainstalowane Visual Studio z odpowiednimi dodatkami do pracy z ASP.NET Core.
2. **Bot Framework SDK**: Możesz zainstalować za pomocą NuGet.

Krok 2: Tworzenie nowego projektu botowego

1. **Stwórz nowy projekt w Visual Studio**:

Otwórz Visual Studio.

1. Wybierz "Create a new project".
2. Wybierz "ASP.NET Core Web Application".
3. Nazwij projekt i kliknij "Create".
4. Wybierz "Bot Framework v4" jako template.

Krok 3: Konfiguracja projektu

1. **Zainstaluj pakiety NuGet**:
2. Kliknij prawym przyciskiem myszy na projekt i wybierz "Manage NuGet Packages".
3. Zainstaluj pakiety: Microsoft.Bot.Builder, Microsoft.Bot.Builder.Dialogs, Microsoft.Bot.Connector.

Krok 4: Tworzenie podstawowych klas

1. **EchoBot.cs**: Klasa bota, który będzie odpowiadał na wiadomości użytkownika.

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Schema;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;


public class EchoBot : ActivityHandler

{

   protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)

   {

      var replyText = $"Echo: {turnContext.Activity.Text}";

      await turnContext.SendActivityAsync(MessageFactory.Text(replyText,replyText), cancellationToken);

   }
```

```
    protected override async Task OnMembersAddedAsync(IList<ChannelAccount>
membersAdded, ITurnContext<IConversationUpdateActivity> turnContext,
CancellationToken cancellationToken)
    {
        foreach (var member in membersAdded)
        {
            if (member.Id != turnContext.Activity.Recipient.Id)
            {
                await turnContext.SendActivityAsync(MessageFactory.Text("Hello and welcome!"),
cancellationToken);
            }
        }
    }
}
```

2. **Startup.cs**: Konfiguracja aplikacji.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers().AddNewtonsoftJson();
        services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();
```

```
        services.AddTransient<IBot, EchoBot>();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseDefaultFiles();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}
```

3. **AdapterWithErrorHandler.cs**: Obsługa błędów.

```
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Extensions.Logging;
public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
{
    public AdapterWithErrorHandler(ICredentialProvider credentialProvider,
ILogger<BotFrameworkHttpAdapter> logger)
        : base(credentialProvider)
    {
        OnTurnError = async (turnContext, exception) =>
        {
            logger.LogError($"Exception caught : {exception}");
```

```
            await turnContext.SendActivityAsync("Sorry, it looks like something went wrong.");
        };
    }
}
```

Testowanie bota

1. **Bot Framework Emulator**:

Pobierz i zainstaluj Bot Framework Emulator z Bot Framework Emulator.

Uruchom emulator i skonfiguruj go do komunikacji z lokalnym botem.

2. **Uruchom bota**:

W Visual Studio uruchom projekt (F5).

Skonfiguruj emulator, używając adresu URL http://localhost:3978/api/messages.

Krok 6: Rozszerzenie funkcjonalności

Możesz rozszerzyć funkcjonalność swojego bota, dodając dialogi, obsługując różne intencje, integrując z zewnętrznymi API itp. Bot Framework SDK dostarcza wiele narzędzi i komponentów, które pomagają w budowie zaawansowanych botów.

Przykład z Dialogiem

1. **MainDialog.cs**:

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using System.Threading;
using System.Threading.Tasks;

public class MainDialog : ComponentDialog
{
    private readonly IStatePropertyAccessor<UserProfile> _userProfileAccessor;

    public MainDialog(UserState userState) : base(nameof(MainDialog))
    {
        _userProfileAccessor = userState.CreateProperty<UserProfile>("UserProfile");

        var waterfallSteps = new WaterfallStep[]
        {
            InitialStepAsync,
```

```
            FinalStepAsync,
        };

        AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
        AddDialog(new TextPrompt(nameof(TextPrompt)));
    }

    private async Task<DialogTurnResult> InitialStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
    {
        return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions {
Prompt = MessageFactory.Text("What is your name?") }, cancellationToken);
    }

    private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
    {
        var userProfile = await _userProfileAccessor.GetAsync(stepContext.Context, () => new
UserProfile(), cancellationToken);
        userProfile.Name = (string)stepContext.Result;

        await stepContext.Context.SendActivityAsync(MessageFactory.Text($"Hello
{userProfile.Name}!"), cancellationToken);

        return await stepContext.EndDialogAsync(cancellationToken: cancellationToken);
    }
}

public class UserProfile
{
    public string Name { get; set; }
}
```

BotWithDialogs.cs:

```
using Microsoft.Bot.Builder;
```

```
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;


public class BotWithDialogs : ActivityHandler
{
    private readonly Dialog _dialog;
    private readonly UserState _userState;


    public BotWithDialogs(Dialog dialog, UserState userState)
    {
        _dialog = dialog;
        _userState = userState;
    }


    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
    {
        await _dialog.RunAsync(turnContext, _userState.CreateProperty<DialogState>("DialogState"), cancellationToken);
    }


    public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default)
    {
        await base.OnTurnAsync(turnContext, cancellationToken);
        await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
    }
}
```

Aktualizacja Startup.cs:

Dodaj odpowiednie usługi do ConfigureServices:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();
    services.AddSingleton<MainDialog>();
    services.AddSingleton<UserState>();
    services.AddTransient<IBot, BotWithDialogs>();
}
```

1. **Zadanie 1**: **Dodanie funkcji logowania do bota**:

Implementacja funkcji logowania, która zapisuje każdą wiadomość użytkownika i odpowiedź bota do pliku dziennika.

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
{
    var userMessage = turnContext.Activity.Text;
    var botResponse = $"Echo: {userMessage}";

    // Logowanie wiadomości
    System.IO.File.AppendAllText("log.txt", $"User: {userMessage}\nBot: {botResponse}\n");

    await turnContext.SendActivityAsync(MessageFactory.Text(botResponse, botResponse), cancellationToken);
}
```

2. **Zadanie 1**: **Dodanie obsługi niestandardowych akcji**:

Implementacja niestandardowych akcji w klasie MainDialog.cs lub BotWithDialogs.cs, np. wyświetlanie aktualnej daty i czasu na żądanie użytkownika.

3. **Zadanie 3**: **Dodanie wsparcia dla wielu języków**:

Dodanie możliwości obsługi różnych języków poprzez wykrywanie języka użytkownika i odpowiednie tłumaczenie wiadomości.

Możesz użyć Microsoft Translator API.

4. **Zadanie 4**: **Dodanie integracji z bazą danych**:

Integracja bota z bazą danych, np. SQL Server, aby przechowywać i pobierać dane użytkowników.

5. **Zadanie 5**: **Dodanie wsparcia dla kart adaptacyjnych**:

Implementacja kart adaptacyjnych w odpowiedziach bota dla bardziej interaktywnej komunikacji.

6. **Zadanie 6**: **Dodanie funkcji uwierzytelniania użytkowników**:

Implementacja uwierzytelniania użytkowników za pomocą OAuth 2.0, aby umożliwić użytkownikom logowanie się za pomocą konta Microsoft lub Google.

7. **Zadanie 7**: **Tworzenie dynamicznych dialogów**:

Implementacja dynamicznych dialogów, które dostosowują się na podstawie wcześniejszych interakcji użytkownika.

8. **Zadanie 8**: **Implementacja funkcji powiadomień push**:

Dodanie funkcji wysyłania powiadomień push do użytkowników, np. przypomnienia o wydarzeniach lub aktualizacjach.

9. **Zadanie 9**: **Integracja z zewnętrznymi API**:

Integracja bota z zewnętrznymi API, np. pogodowymi, finansowymi, aby dostarczać użytkownikom aktualne informacje na żądanie.

10. **Zadanie 10**: **Dodanie obsługi dialogów sekwencyjnych**:

Implementacja dialogów sekwencyjnych, które prowadzą użytkownika przez serię kroków, np. składanie zamówienia, rezerwacja spotkania.

11. **Zadanie 11**: **Testowanie bota**:

Implementacja jednostkowych i integracyjnych testów dla bota, aby zapewnić poprawność działania.

12. **Zadanie 12**: **Optymalizacja wydajności**:

Optymalizacja kodu bota, aby poprawić jego wydajność i szybkość reakcji.

13. **Zadanie 13**: **Zarządzanie stanem użytkownika**:

Implementacja zarządzania stanem użytkownika, aby bot mógł pamiętać kontekst wcześniejszych rozmów i dostosowywać się do nich.

14. **Zadanie 14**: **Obsługa błędów i wyjątków**:

Dodanie bardziej zaawansowanej obsługi błędów i wyjątków, aby bot mógł odpowiednio reagować na nieprzewidziane sytuacje.

15. **Zadanie 15**: **Rozbudowa funkcji pomocy i dokumentacji**:

Implementacja funkcji pomocy, która dostarcza użytkownikom szczegółowe informacje o tym, jak korzystać z bota i jakie funkcje są dostępne.

1. Dodanie logowania do pliku:

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)

{
```

```
    var userMessage = turnContext.Activity.Text;
    var botResponse = $"Echo: {userMessage}";

    // Logowanie wiadomości
    System.IO.File.AppendAllText("log.txt", $"User: {userMessage}\nBot: {botResponse}\n");

    await turnContext.SendActivityAsync(MessageFactory.Text(botResponse, botResponse),
cancellationToken);
}
```

2. Integracja z bazą danych:

```
public class UserProfile
{
    public string UserId { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
}

// Inicjalizacja połączenia z bazą danych
var connectionString = "your_connection_string";
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Wykonywanie operacji na bazie danych
}
```