<u>Arlington County Code Challenge</u>

By Teju Peri
E-mail: teju.fullstackdeveloper@gmail.com

Key Aspects: Builder Mindset, Data Management, Business Intent

I have used GitHub, Visual Studio Code, libraries and tried to use several components interactively.

What is Microsoft BOT Framework

The Bot Framework is built to be intelligent and learn from user interaction. User interaction can be possible via Chatting or Text/SMS to Skype, Mail, Telegram, SMS, etc.\

Our example is just a starting point for building bots and conversational helpers using the Microsoft Bot Framework. Of course, it doesn't have to be limited to the Microsoft Teams channel, it can be targeting Slack, Skype or any other available distribution channel.

Bot Framework Service

The Bot Framework Service, which is a component of the Azure Bot Service, sends information between the user's bot-connected app and the bot. Each channel can include additional information in the activities they send. Before creating bots, it is important to understand how a bot uses activity objects to communicate with its users.

As seen below there are two activity types, *conversation update* and *message*, which might be exchanged when a user communicates with a simple echo bot.

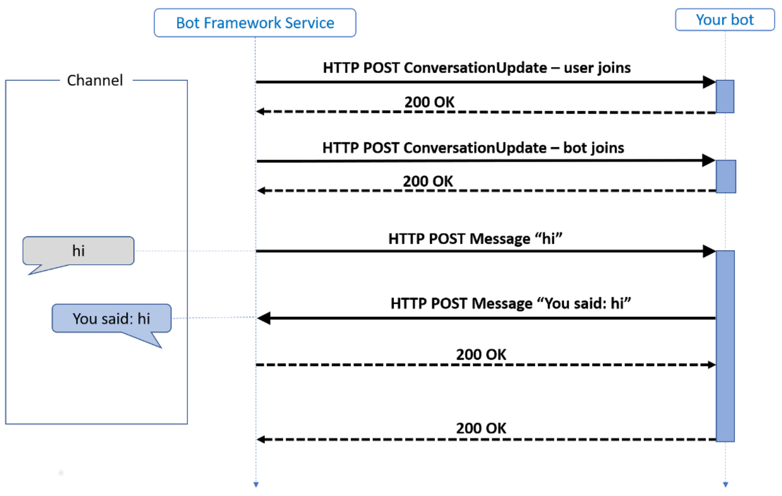
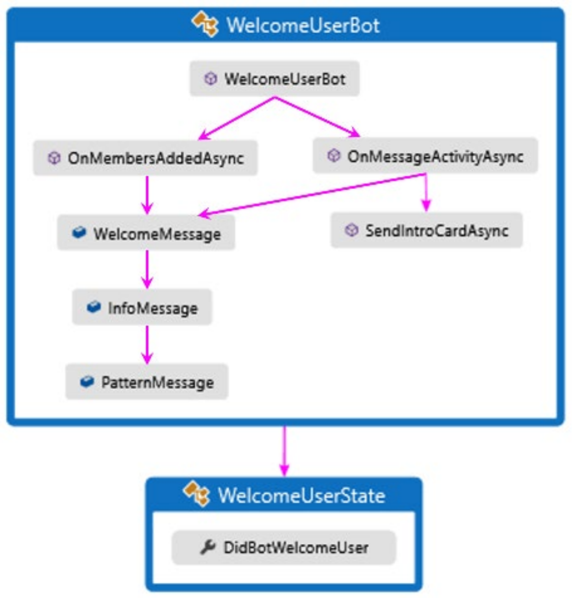

Bot Framework Service
Your bot
Channel
HTTP POST ConversationUpdate – user joins
200 OK
HTTP POST ConversationUpdate – bot joins
200 OK
hi
HTTP POST Message "hi"
You said: hi
HTTP POST Message "You said: hi"
200 OK
200 OK

Image Credit: Microsoft

The Bot Framework SDK

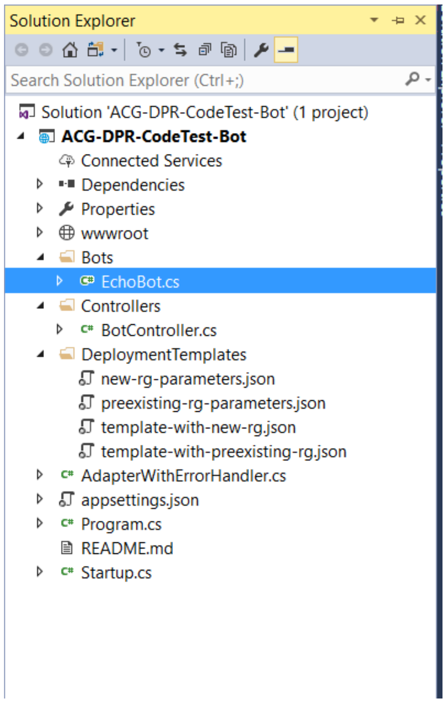
The Bot Framework SDK allows you to build bots that can be hosted on the Azure Bot Service. The service defines a REST API and an activity protocol for how your bot and channels or users can interact. The SDK builds upon this REST API and provides an abstraction of the service so that you can focus on the conversational logic. While you don't need to understand the REST service to use the SDK, understanding some of its features can be helpful.

The two main events encountered by the bot are:



OnMembersAddedAsync which is called whenever a new user is connected to your bot
OnMessageActivityAsync which is called whenever a new user input is received.

Whenever a new user is connected, they are provided with a WelcomeMessage, InfoMessage, and PatternMessage by the bot. When a new user input is received, WelcomeUserState is checked to see if DidBotWelcomeUser is set to *true*. If not, an initial welcome user message is returned to the user.



```
Solution 'ACG-DPR-CodeTest-Bot' (1 project)
  ACG-DPR-CodeTest-Bot
    Connected Services
    Dependencies
    Properties
    wwwroot
    Bots
      EchoBot.cs
    Controllers
      BotController.cs
    DeploymentTemplates
      new-rg-parameters.json
      preexisting-rg-parameters.json
      template-with-new-rg.json
      template-with-preexisting-rg.json
    AdapterWithErrorHandler.cs
    appsettings.json
    Program.cs
    README.md
    Startup.cs
```

Project Structure

My Code solution (EchoBot.cs)

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.
//
// Generated with Bot Builder V4 SDK Template for Visual Studio EchoBot v4.14.0

// Modified by Teju Peri - *** 10-6-21 ***  Arlington County - MS Solution Architect Role

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

namespace ACG_DPR_CodeTest_Bot.Bots
{
    public class EchoBot : ActivityHandler
    {
        private const string welcomeText = "This is a modified Welcome Bot sample by 
Teju. You will be introduced by the bot now ";
        private const string localeText = "welcome the user using their Locale ";
        private const string infoText = "Seems like you joined the conversation!";
        /*For simplicity sake I am just changing per the requirement.
         Please see my solution/readme for a more extensive solution/extension for this*/

        protected override async Task
OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken
cancellationToken)
        {
            //Convert to Lowercase
            var inputText = turnContext.Activity.Text.ToLowerInvariant();
            /* The following logic can be developed in a more dynamic way using
             language understanding like QnA, LUIS etc. */
            switch (inputText)
            {

                case "how are you":
                    {
                        string responseToHowru = "I'm great!";
                        await
turnContext.SendActivityAsync(MessageFactory.Text(responseToHowru, responseToHowru),
cancellationToken: cancellationToken);
                    }
                    break;

                case "hello":
                    {
                        string responseToHello = "Hello there!";
                        await
turnContext.SendActivityAsync(MessageFactory.Text(responseToHello, responseToHello),
cancellationToken);
                    }
                    break;
                default:
                    {
                        string responseEcho = $"Echo: {turnContext.Activity.Text}";
                        await
turnContext.SendActivityAsync(MessageFactory.Text(responseEcho, responseEcho),
cancellationToken);

                    }
                    break;

            }


        }

        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($"Hello Bot User - {member.Name}.
{welcomeText}", cancellationToken: cancellationToken);
                    await turnContext.SendActivityAsync($"{localeText} Dear Bot User your
locale is '{turnContext.Activity.GetLocale()}'.", cancellationToken: cancellationToken);
                    await turnContext.SendActivityAsync(infoText, cancellationToken:
cancellationToken);

                }
            }
        }
    }
}
```

Code Explanation

I have simply set messages through string variables which are sent to the user.
We are greeting the users when added to the conversation

OnMembersAddedAsync which is called whenever a new user is connected to our bot

- Hello Bot User
- Give the user their Locale
- Information Text

OnMessageActivityAsync which is called whenever a new user input is received

• How are you	• I'm great
• Hello	• Hello there!
• Default	• Echo the text

Using *ToLowerVariant ()* to convert to lowercase, and a switch case statement.
The MessageFactory.Text returns the text of the message to send.

Conclusion, further analysis, and alternative approaches

It is clear that any strategy or solution should concentrate on efficient code, modularity

Our approach is simple for the lack of time. But I will discuss certain alternative strategies and further work we can do to dig deeper for generating actionable insights.

1. We have to account for the fact that all channels do not send the conversation update activity. If we find that this bot works in the emulator, but does not in another channel the reason is most likely that the channel does not send this activity.

2. We can get the username when channel sends the user name in “From”

```
var userName = turnContext.Activity.From.Name;
```

3. Save state Changes using `SaveChangesAsync()`

4. Use Images like introduction cards to greet users

5. Set user state using a read only Bot State like `userState`

6. Initialize a new instance of the "Bot" class.

```
public MyBot(UserState _userState)
{
    userState = _userState;
}
```

7. Our example hardcodes messages. You should use LUIS or QnA for more advance language understanding.

LUIS

Language Understanding (LUIS) is a cloud-based API service that enables us to do just that so that our bot can recognize the intent of user messages, allow for more natural language from our user, and better direct the conversation flow. The ability to understand what our user means conversationally and contextually can be a difficult task, but can provide our bot a more natural conversation feel.

QnA Maker

QnA Maker is a cloud-based API service that lets us create a conversational question-and-answer layer over our existing data. We can use it to build a knowledge base by extracting questions and answers from our semi-structured content, including FAQ, manuals, and documents. We can answer users’ questions with the best answers from the QnAs in our knowledge base—automatically.Our knowledge base gets smarter, too, as it continually learns from user behavior.

8. We can further enhance our app with buttons to provide input, multi-turn dialog, cards, thumbnail, audio, media etc.