Movie Recommendation Bot Sample

## Overview

# This article will demonstrate how to create a bot for suggesting movies to watch.

## Prerequisites

# Visual Studio [2017](https://www.visualstudio.com/downloads)

# Bot Builder SDK v4 template for [C#](https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4)

# [Bot Framework Emulator](https://github.com/Microsoft/BotFramework-Emulator/releases)

# Knowledge of [ASP.Net Core](https://docs.microsoft.com/aspnet/core/) and asynchronous programming in [C#](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/index)

## Scenario

# You want a way to suggest movies to watch. You have access to a database that contains information about movies. The user of the bot will be asked questions such as:

# What kind of movie do you want to watch?

# Who is your favorite actor?

# How many stars should this movie have?

# After the bot has asked all of these questions, it should be able to recommend a movie.

## Getting Started

# A bot can be created in many different ways such as through the Azure Portal or by the command line. Because we are using Visual Studio 2017 for this example, we can create a bot by using the Bot Builder SDK v4 template for C#.

# You can get the template from <https://marketplace.visualstudio.com/items?itemName=BotBuilder.BotBuilderV4>

# If you don’t get all the NuGet packages you need for this sample when you install the template or the SDK, then try checking the following pages for more help.

# <https://www.nuget.org/packages/Microsoft.Bot.Builder/4.0.1-preview>

# <https://www.nuget.org/packages/Microsoft.Bot.Builder.Dialogs/4.0.1-preview>

# In Visual Studio, create a new bot project.

# Visual Studio project

# We will be testing our bot with the Bot Framework Emulator a little later but let’s install it now. You can get it from GitHub at <https://github.com/Microsoft/BotFramework-Emulator/releases>

# We should have all we need to get started now. If you need a working example, feel free to download my project at <https://github.com/stevenjacox/MovieRecommendationBot>

# Our bot will be designed as a simple waterfall discussion bot. This means that we will ask one question at a time and if there is an error it will start over.

# Starting with the bot class (implements the IBot interface) in the project that you created when you used the template, add some code that will allow us to use dialogs from the Microsoft.Bot.Builder.Dialogs namespace.

using **Microsoft.Bot.Builder.Dialogs;**

# using Microsoft.Bot.Builder.Prompts;

Now let’s design our prompts and dialogs to be able to handle our three questions. We need a DialogSet object and a couple of variables to use in our conversation.

private readonly DialogSet dialogs;

static string movieType;

static string favoriteActor;

I went ahead and mocked the database with a simple Movies class to represent data access. There are ways to persist the data in our conversation but I will leave that out for this example. For more information; see [Persist User Data](https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-tutorial-persist-user-inputs?view=azure-bot-service-4.0&tabs=cstab).

Movies movies = new Movies();

In the bot constructor, we will specify everything we need for this movie recommendation bot.

public MovieRecommendationBot()

{

dialogs = new DialogSet();

// Define our dialog

dialogs.Add("movieRecommendation", new WaterfallStep[]

{

async (dc, args, next) =>

{

await dc.Context.SendActivity("Welcome to the movie recommendation service.");

await dc.Prompt("movieTypePrompt", "What type of movie are you looking for?");

},

async(dc, args, next) =>

{

movieType = args["Text"].ToString().ToLowerInvariant();

// Ask for next info

await dc.Prompt("favoriteActorPrompt", "Who is your favorite actor?");

},

async(dc, args, next) =>

{

favoriteActor = args["Text"].ToString().ToLowerInvariant();

// Ask for next info

await dc.Prompt("ratingPrompt", "What is the least number of stars you would want in this movie?");

},

async(dc, args, next) =>

{

Movie movieMatch;

movieMatch = movies.MovieList.Where (movie => (movie.Genre.ToLowerInvariant().Equals(movieType)

&& movie.Actors.Contains(favoriteActor.ToLowerInvariant()))

&& movie.Rating > (int)args["Value"]

).FirstOrDefault();

if (movieMatch == null)

{

movieMatch = movies.MovieList.Where (movie => (movie.Genre.ToLowerInvariant().Equals(movieType)

|| movie.Actors.Contains(favoriteActor.ToLowerInvariant()))

&& movie.Rating > (int)args["Value"]

).FirstOrDefault();

}

if (movieMatch != null)

{

string msg = "I found a movie that you might want to check out. " +

$"\nMovie Name: {movieMatch.MovieName.ToString()} " +

$"\nRelease Year: {movieMatch.ReleaseYear.ToString()} " +

$"\nGenre: {movieMatch.Genre.ToString()} " +

$"\nRating: {movieMatch.Rating} " +

$"\nActors: {string.Join(", ", movieMatch.Actors.ToArray())}";

await dc.Context.SendActivity(msg);

await dc.End();

}

else

{

Movie goodMovie = movies.MovieList.Where(movie => movie.Rating >= (int)args["Value"]).First();

if (goodMovie == null)

{

Random rnd = new Random();

int index = rnd.Next(0, movies.MovieList.Count() -1);

//Random movie becomes good movie when you don't have a match

goodMovie = movies.MovieList.ElementAt(index);

}

string msg = "I couldn't find an exact match for you but may I recommend the following: " +

$"\nMovie Name: {goodMovie.MovieName.ToString()} " +

$"\nRelease Year: {goodMovie.ReleaseYear.ToString()} " +

$"\nGenre: {goodMovie.Genre.ToString()} " +

$"\nRating: {goodMovie.Rating}" +

$"\nActors: {string.Join(",", goodMovie.Actors.ToArray())}";

await dc.Context.SendActivity(msg);

await dc.End();

}

}

});

// Add a prompt for the reservation date

dialogs.Add("movieTypePrompt", new Microsoft.Bot.Builder.Dialogs.TextPrompt());

// Add a prompt for the party size

dialogs.Add("favoriteActorPrompt", new Microsoft.Bot.Builder.Dialogs.TextPrompt());

// Add a prompt for the user's name

dialogs.Add("ratingPrompt", new Microsoft.Bot.Builder.Dialogs.NumberPrompt<int>(Culture.English));

}

You can see that the names of the prompts added to the DialogSet must match the names that are used with the Context.

For example:

await dc.Prompt("favoriteActorPrompt", "Who is your favorite actor?"); needs a matching prompt such as dialogs.Add("favoriteActorPrompt", new Microsoft.Bot.Builder.Dialogs.TextPrompt());

Be careful because if they don’t match then you might get a “Value cannot be null” error.

Our class implements the IBot interface, which only requires that OnTurn method be defined. Here we only check for “recommend” or “movie” to start the conversation with the bot.

public async Task OnTurn(ITurnContext context)

{

if (context.Activity.Type == ActivityTypes.Message)

{

// The type parameter PropertyBag inherits from

// Dictionary<string,object>

var state = ConversationState<Dictionary<string, object>>.Get(context);

var dc = dialogs.CreateContext(context, state);

await dc.Continue();

// Additional logic can be added to enter each dialog depending on the message received

string activityText = context.Activity.Text.ToLowerInvariant();

if (!context.Responded)

{

if (activityText.Contains("recommend") || activityText.Contains("movie"))

{

await dc.Begin("movieRecommendation");

}

else

{

await context.SendActivity($"You said '{context.Activity.Text}'; maybe you could ask me to recommend you movie?");

}

}

}

}

You should have everything you need to run this bot now. However, you will want to start this bot without debugging (Ctrl+F5) because we will be debugging with the Bot Framework Emulator.

After you have the bot running, open the Bot Framework Emulator and go to File -> Open Bot. The dialog should help you find the .bot file that was created when you used the VS Template. It should look something like this:

{

"name": "MovieRecommendationBot",

"secretKey": "",

"services": [

{

"appId": "",

"id": "http://localhost:3978/api/messages",

"type": "endpoint",

"appPassword": "",

"endpoint": "http://localhost:3978/api/messages",

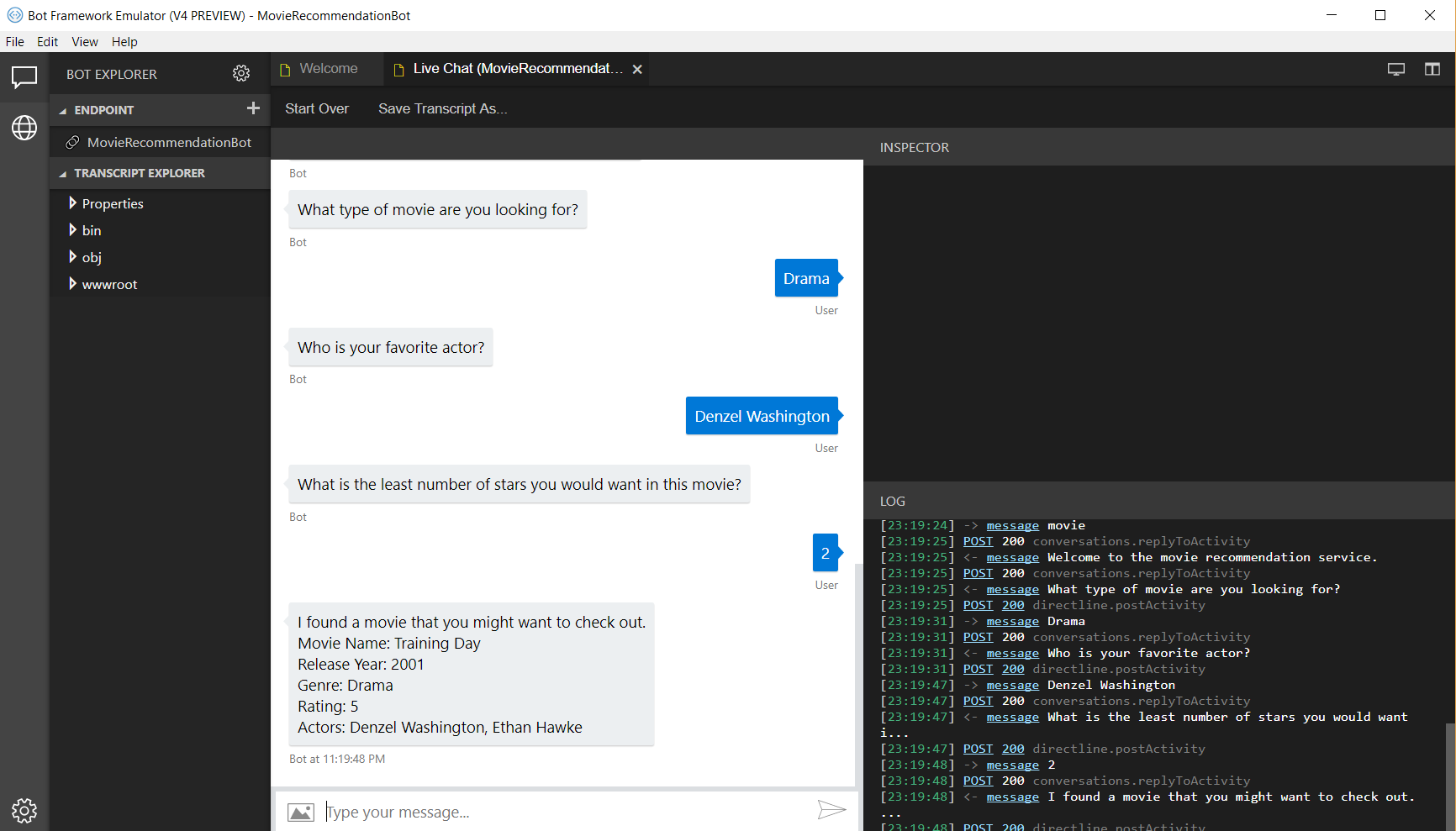
"name": "MovieRecommendationBot"

}

]

}

If you are running the bot and you have connected to the bot, you should see a chat screen like this.



Go ahead and try different questions. If you are getting 400 errors, then something is wrong and you have to sure that the bot is running in VS.

## Conclusion

This tutorial showed the creation of a movie recommendation bot and the use of the Bot Framework Emulator.