Rubicon Global Azure Bootcamp 2018

LAB 3: Step 1 – Extend LUIS APP

Before we continue customizing the bot from LAB 2, we want to add more 'intents' to LUIS to broaden its understanding.

1. Go to Luis.ai
2. Click on Sign In
3. Sign in with the credentials from LAB 2 (if necessary)
4. Select the application (in our example "GAB2018BOT-<key>")
5. Select Intents
6. Click [Create new intent]
7. Type "Authenticate"
8. Click [Done]
9. After the intent is created, you can start adding sentences to train it. Before we do this, we will first create the other intents for this LAB. Repeat the following and create intents named BrokenBoiler, ErrorCode and PlanDate:
    a. Select Intents in the left menu
    b. Click [Create new intent]
    c. Type the name
    d. Click [Done]
10. Select Intents in the left menu
11. You should see 10 intents.

Before we continue, we want to add some prebuilt Entities to our App before we start adding sentences to train it.

12. Select Entities in the left menu
13. Click [Manage prebuilt entities]
14. Check the following entities:
    a. Number
    b. Email
    c. DateTimeV2

Let's continue training our app

1. Select Intents in the left menu
2. Click on "Authenticate"
3. Type the following sentence in the first input field:

   "My email is test@test.com"

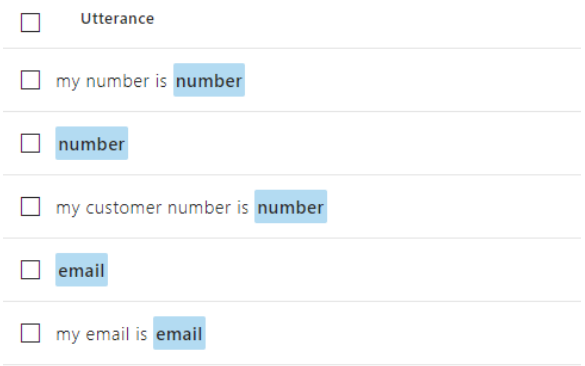
4. LUIS should have automatically detected the email, and replaced the sentence with "My email is **email**"
5. Type the following sentence in the first input field:
   "test@test.com"
6. Type the following sentence in the first input field:

   "My customer number is 1234567"

7. LUIS should have automatically detected the number, and replaced the sentence with "My customer number is **number**"
8. Type the following sentences in the first input field:

   "1234567"
   "My number is 1234567"

9. You should have 5 sentences looking like this:



10. We are ready with this intent.

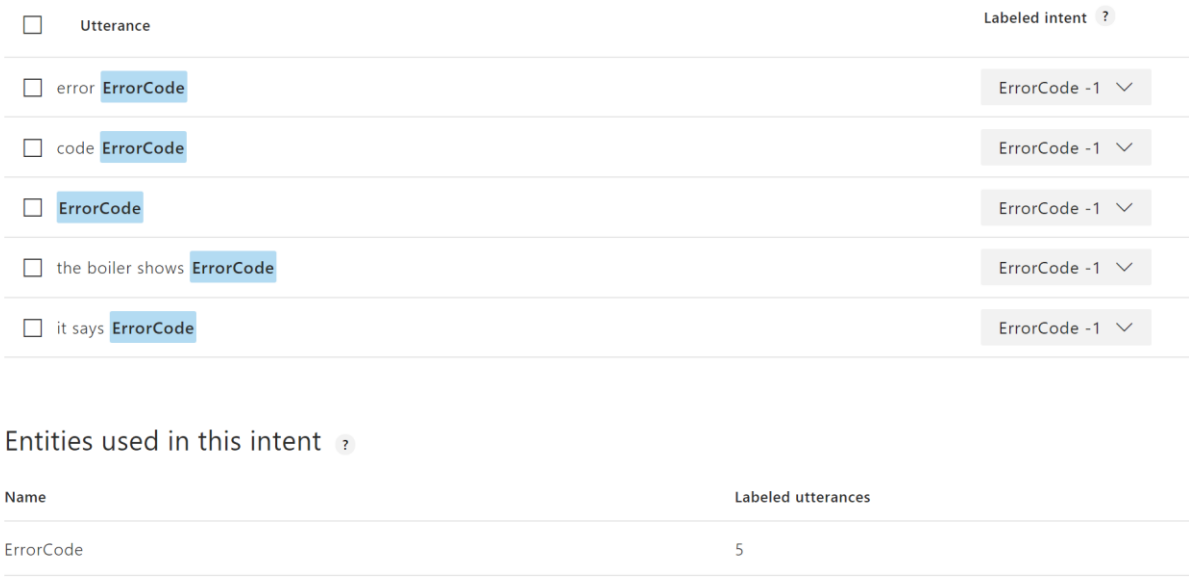
Let's add another one…

11. Select Intents in the left menu
12. Click on "BrokenBoiler"
13. We are not going to add any entities here, so just type the following 5 sentences in the first input field:
    "My boiler is broken"
    "I have no heat"
    "I have no hot water"
    "It is broken"
    "It does not work"

    (Add any sentences that you think can help identify how people would tell you their boiler is broken. The more examples LUIS has, the better it can predict when people mean this.)

14. Select Intents in the left menu
15. Click on "ErrorCode"
16. We are going to introduce only one new entity here, called "ErrorCode". Add the following sentences and replace, E01, E10, E120, E1 and E4 with the entity "ErrorCode". Note, with the first sentence you need to create the entity first. It is a **simple** entity.
    "It says E01"
    "The boiler shows E10"
    "E120"
    "code E1"
    "error E4"

17. Your intent should look like this:



Utterance	Labeled intent
error ErrorCode	ErrorCode -1
code ErrorCode	ErrorCode -1
ErrorCode	ErrorCode -1
the boiler shows ErrorCode	ErrorCode -1
it says ErrorCode	ErrorCode -1

Entities used in this intent

Name	Labeled utterances
ErrorCode	5

The last intent is PlanDate. Use the following sentences to train LUIS. We are going to use a built-in entity (just like the email).

18. Select Intents in the left menu
19. Click on “PlanDate”
20. Type the following sentence in the first input field:

    “I am home next Thursday”

21. Luis should have detected that it is a date and replaced Thursday with **datetimeV2**
22. Now add the following sentences:
    “next Wednesday”
    “next week on Monday”
    “November 13th”
    “end of the week”
23. The last 4 should all be changed to **datetimeV2**

We can now train our app.

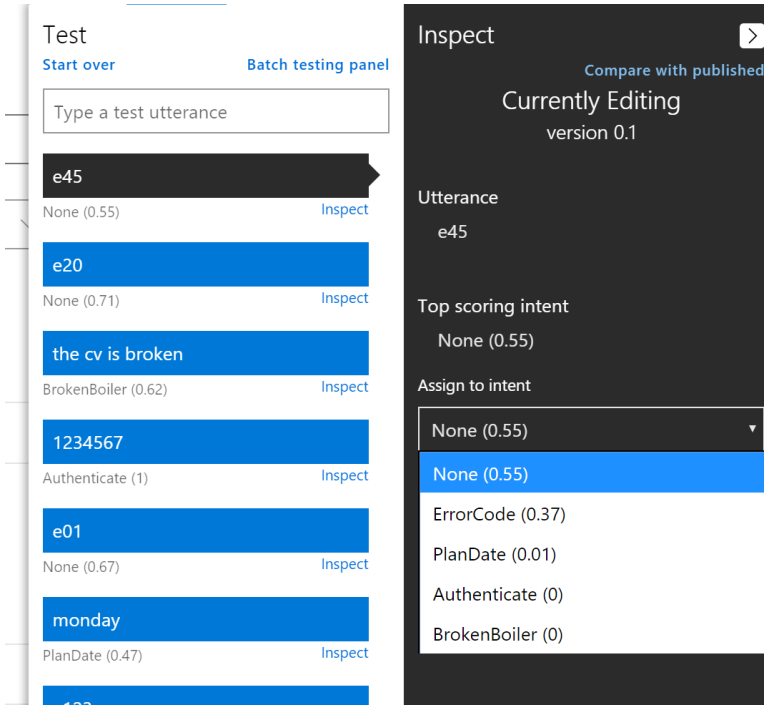
24. Click [Train] on the top menu
25. After training is done.
26. Click on [Publish] on top menu
27. Select Production in the Publish To option.
28. Click [Publish to production slot]

You can now test your app

1. Click on [Test] in the top menu.
2. Type a sentence (different from the once you used to train)

NOTE:

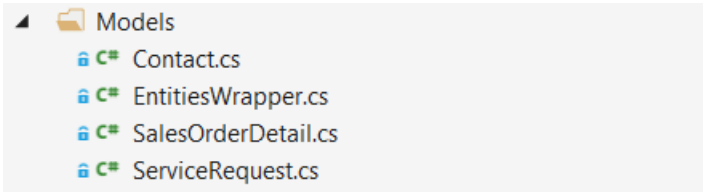
- Typing an error code like “E345” can result in a none intent. If you click on the sentence, you can inspect the answer and assign it to a new intent and retrain your app. This enables you to finetune your app. In the example below “E45” is 55% undefined and “37%” ErrorCode intent. After finetuning your app, you need to train and publish it again.



LAB 3: Step 2 - Connect to Microsoft Dynamics CRM

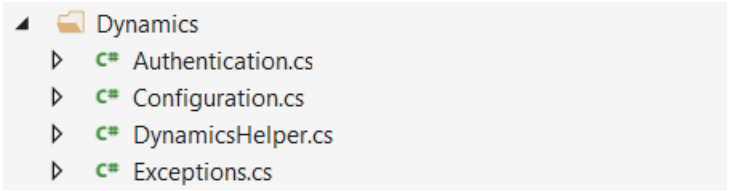
1. Open the bot solution from LAB 2 in Visual Studio
2. Go to Tools - NuGet Package Manager - Manage NuGet Packages for Solution...
3. Go to Browse and search for package Microsoft.IdentityModel.Clients.ActiveDirectory
4. Install the latest version of the package to the SimpleEchoBot project

5. Create a new folder “Models” under the SimpleEchoBot project and add the 4 classes from Lab3/Code/Models:



TIP: to download a file, open the file on Github, right click the Raw button and choose Save target as. Another option is to clone or download the entire Git repository from the project homepage.

6. Create a new folder “Dynamics” under the SimpleEchoBot project and add the 4 classes from Lab3/Code/Dynamics:



7. Log in to the Azure portal and navigate to your web bot's App Service component.
8. Go to the Application settings blade under Settings
9. Scroll down to Application settings, and use Add new setting to add the following 4 settings. The values will be provided on site by Rubicon:


ResourceUrl
WebAPIUrl
ApplicationID
SecretId
+ Add new setting

LAB 3: Step 3 - Add dialogs that interact with CRM

We are now going to add some custom dialogs to the chatbot. You may want to use the Dialogs\DialogTemplate.cs file as a basis for creating new dialogs. Also, use the code of the various dialogs in the example project to get an idea of how to create your own.

Identify Customer dialog

1. Add a new IdentifyCustomerDialog.cs class to the Dialogs folder of your bot to identify the customer.
2. Have the dialog ask for the users e-mail address:

```
await context.PostAsync($"Can you give me your e-mail address?");
```

3. Create a new CRM request uri string which queries CRM contacts by e-mail address and provide the e-mail address returned by LUIS:

```
requestContactsUri = $"contacts?$select=fullname,contactid,firstname&$top=5&$filter=(emailaddress2 eq '{emailAddress}')";
```

4. Use the Dynamics helper to execute the request to the Dynamics CRM backend:

```
var response = await DynamicsHelper<Contact>.HttpClient.GetAsync(requestContactsUri, HttpCompletionOption.ResponseHeadersRead);
JObject repsonseAccounts = JsonConvert.DeserializeObject<JObject>(await response.Content.ReadAsStringAsync());
```

   You may need to add a reference to Newtonsoft.Json and using Newtonsoft.Json.Linq

5. If the customer is found in CRM, finish the Identify Customer dialog so that the customer returns to the BasicLuisDialog:

```
context.Done(customerContext);
```

6. Now we need to add an “Authenticate” intent to the BasicLuisDialog to route the user to the Identify Customer dialog:

```
[LuisIntent("Authenticate")]
public async Task AuthenticateIntent(IDialogContext context, LuisResult result)
{
    await context.Forward(new IdentifyCustomerDialog(customerContext),
        this.ResumeAfterCustomerIdentification, context.Activity, CancellationToken.None);
}
```

7. And implement a ResumeAfterCustomerIdentification function in the BasicLuisDialog for when the user returns after completing the Identify Customer dialog:

```
private async Task ResumeAfterCustomerIdentification(IDialogContext context, IAwaitable<object> result)
{
    customerContext = await result as DynamicsContextController;
    if (customerContext.CustomerId.HasValue)
    {
        await context.PostAsync($"Welcome {customerContext.FirstName}");
    }
}
```

8. Publish your bot and test your new dialog.

   HINT: you can test your Identify Customer dialog using the following e-mail addresses:
   john.doe@example.com, jane.doe@example.com and jan.jansen@example.com

Identify Boiler dialog

1. Add another dialog that will query CRM sales orders for boilers sold to the customer:

```
var fetchXml = string.Format(ContactInfo.RetrieveOrderByContact, this.customerContext.CustomerId);
var salesOrderDetailsList = await DynamicsHelper<SalesOrderDetail>.GetFromCrmFetchXml(fetchXml, "salesorderdetails");
```

2. If only one boiler is found, automatically select it and return to the BasicLuisDialog.
3. If multiple boilers are found, ask the user to select the correct boiler. This can be done using a PromptDialog:

```
PromptDialog.Choice(
    context: context,
    resume: this.ResumeAfterBoilerChoice,
    options: salesOrderDetailsList,
    prompt: "We found multiple boilers, about which one do you have a question?",
    retry: "Sorry, I didn't understand, can you please select the right boiler.",
    promptStyle: PromptStyle.Auto);
```

4. Add a ResumeAfterBoilerChoice function, that handles selection of a boiler in the PromptDialog selection list:

```
private async Task ResumeAfterChChoise(IDialogContext context, IAwaitable<SalesOrderDetail> result)
{
    var order = await result;
    if (order != null)
    {
        customerContext.CustomerCh = order;
        context.Done(this.customerContext);
    }
}
```

5. And add an IdentityBoiler intent and a ResumeAfterIdentifyBoiler function to the BasicLuisDialog that handles the routing.
6. Publish your bot and test your new dialog.

HINT: John Doe has 0 boilers, Jane Doe has 1 boiler and Jan Jansen has 2 boilers

Make Appointment dialog

LUIS is able to translate user input such as “Next week” or “Coming monday” to date objects. We can use this feature to allow the user to make a service appointment.

1. Add yet another dialog that will ask the customer for a timeframe to make a service appointment.
2. If LUIS successfully translates the users input to an actual date/time, it will return an Entity of type datetimeV2. We can check this by:

```
var entity = result.Entities.FirstOrDefault(x => x.Type.StartsWith("builtin.datetimeV2."));
entity?.Resolution?.TryGetValue("values", out object formattedValues);
```

3. If an Entity of type datetimeV2 exists, it can be a range of different kinds of datetimeV2, such as a single date, a date range, a single time, a time range and combinations of date and time ranges. To handle all these different kinds of datetimeV2 values, we can use a parser:

```
Chronic.Parser parser = new Chronic.Parser();
EntityRecommendation date = new EntityRecommendation();
Chronic.Span resultSpan = null;
result.TryFindEntity(entity.Type, out date);
resultSpan = parser.Parse(date.Entity);
```

4. After getting the desired timeframe, create a ServiceRequest to make an appointment within CRM:

```
var request = new ServiceRequest()
{
    CustomerId = this.customerContext.CustomerId,
    Title = "Service appointment for " + this.customerContext.CustomerCh.ProductName,
    Description = $"Service appointment for broken boiler",
    ScheduledOn = serviceAppointment
}

var response = await DynamicsHelper<ServiceRequest>.WriteToCrm(JsonConvert.SerializeObject(request), "incidents");
JObject result = JsonConvert.DeserializeObject<JObject>(await response.Content.ReadAsStringAsync());
```

5. Add an MakeAppointment intent and a ResumeAfterMakeAppointment function to the BasicLuisDialog that handles the routing.
6. Don’t forget to thank the user for contacting customer support!

You're Done

Congratulations, you have successfully implemented a AI infused chatbot that uses a Dynamics CRM backend!

Still have some time left?

What about:

- Asking the user for any error codes the boiler might show
- Using the Q&A from LAB 1 to try to answer most of the user's questions
- Use Azure Cognitive Services to allow the user to upload a photo of the boiler and try to automatically identify make and model, and perhaps even the error code

Or … just call it a day, have drink and chat :)

We hope you enjoyed!