

Building a Skype Bot with Microsoft Bot Framework

DS '17 Hands-On Cognitive Lab

Archit Gupta

Cognitive Research Associate
Miracle Software Systems, Inc.

December , 2017

Building a Skype Bot with Microsoft Bot Framework

Introduction

This document contains a step-by-step guide to take you through the various options with the Microsoft Bot Framework and will teach you how to create a Bot that can recognize name and ID given by user. This guide was prepared by **Miracle's Innovation Labs** 😊

Prerequisites

All attendees must have their own workstation(with Internet) to participate in the lab(Both PC and Mac are compatible). The following installation pre-requisites will help to make the hands-on lab experience easier.

- Create a **Microsoft Account**(Will be needed for using the Bot Framework and LUIS)
- Install the following items and ensure that you are able to access them from your command line,
 - **Node JS** - <https://nodejs.org/en/download/>
 - **NPM** – Should come along with Node JS
 - **Git(Optional)** - <https://git-scm.com/downloads>
 - **ngrok** - <https://ngrok.com/download>
- Create your **Skype Account** to be able to add bots - <https://www.skype.com/en/new/>
- Create your **LUIS account** to be able to train your bot - <https://www.luis.ai/home>

- Get Access to **Bot Framework** where you can Register your bot and integrate with Skype - <https://dev.botframework.com/bots/new>

Technology Involved

- Server Side – Node JS
- NLP – LUIS
- Middleware - Microsoft Bot Framework

Let's Get Started!

The following steps will outline how you can create **Erin Bot** and integrate it into Skype. Users will be able to directly message your bot and perform operations such as displaying name and ID entered by user.

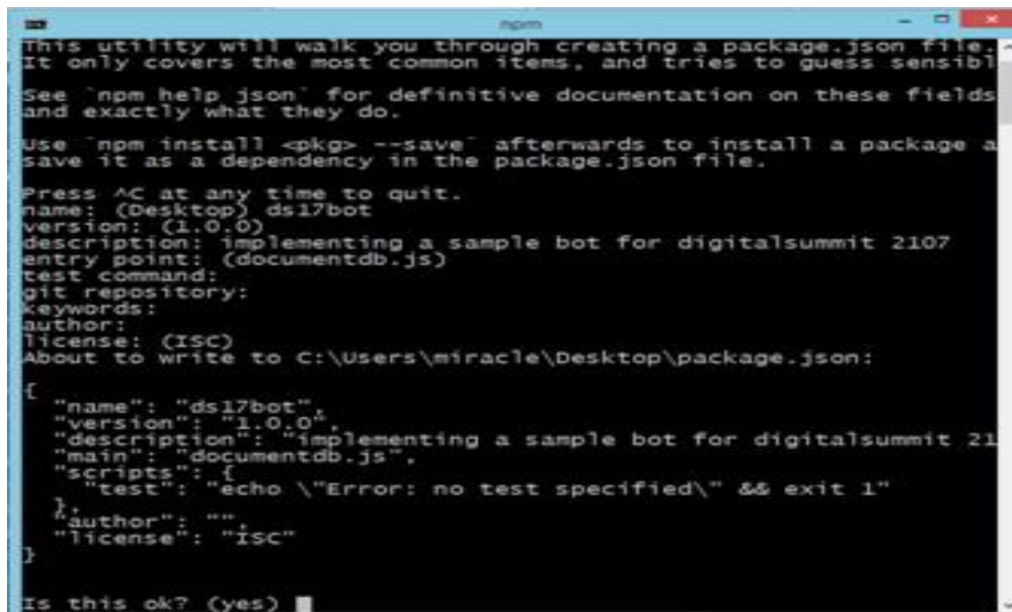
Step #1 – Creating a Static Bot with MS Bot Framework

The first step will be to use the Bot Builder SDK for Node JS and build a static backend that can respond to user messages. We will then first simulate the bot with the Bot Emulator and will then register your bot with the bot framework.

Download the **Bot Emulator** here : <https://emulator.botframework.com/>

#1 | Create Bot Backend and Test with Emulator

Create a folder for your bot backend and then navigate into that folder. Initialize your node project using the **npm init** command.



```
npm
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible
defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (Desktop) ds17bot
version: (1.0.0)
description: implementing a sample bot for digitalsummit 2107
entry point: (documentdb.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\miracle\Desktop\package.json:

{
  "name": "ds17bot",
  "version": "1.0.0",
  "description": "implementing a sample bot for digitalsummit 2107",
  "main": "documentdb.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this ok? (yes)
```

Add the required node modules using the following command,

npm install --save botbuilder restify moment

Create an **app.js** file and paste the following code to handle the user's messages and send the responses.

//Add the modules that are required

```
var restify = require('restify');
var builder = require('botbuilder');
var LUIS = require('luissdk');
```

// Setup Restify Server

```
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3000, function () {
  console.log("-----");
  console.log("Bot is running with the address : "+server.url);
  console.log("-----");
});
```

```
});
```

```
// Create chat bot
```

```
var connector = new builder.ChatConnector({  
  appId: "",  
  appPassword: ""
```

```
});
```

```
var bot = new builder.UniversalBot(connector);  
server.post('/api/messages', connector.listen());
```

```
//=====
```

```
/      Bots                                                    Dialogs
```

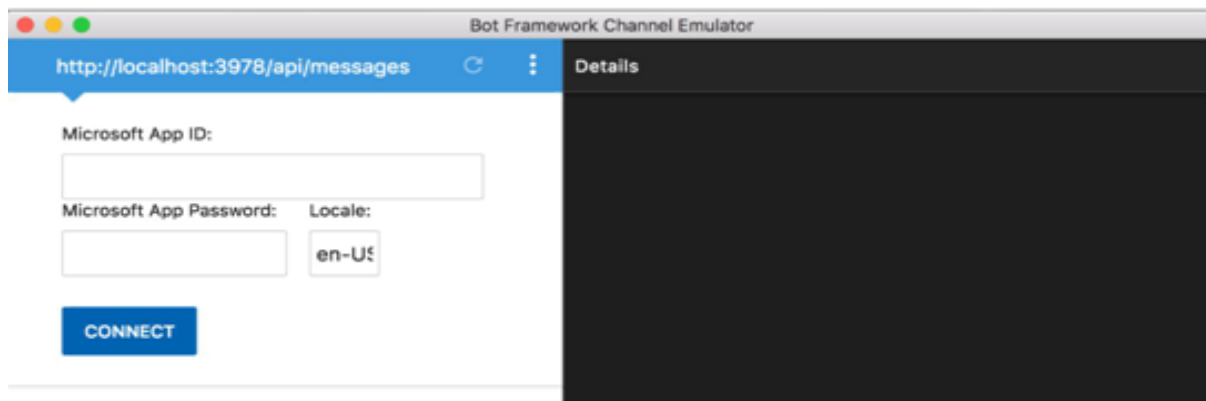
```
//=====
```

```
bot.dialog('/', function (session) {  
  session.send("Hello World");  
});
```

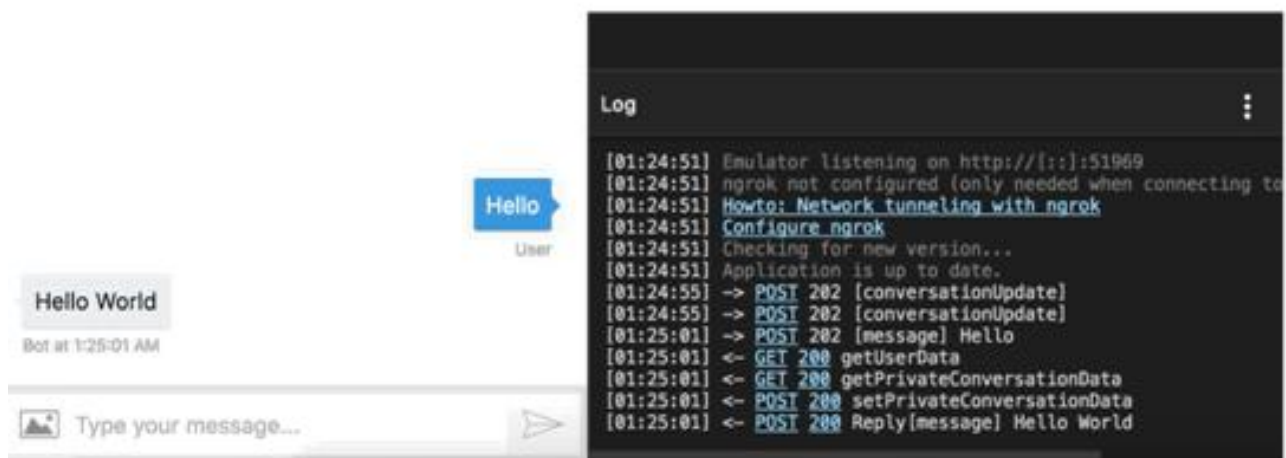
Run your backend by using the following command, **node app.js**

```
C:\Users\miracle\Desktop\DS'17 SESSION\Samplebot>node ds17  
-----  
December 5th 2017, 11:38:21 pm : Erin is running with the address : http://[::]  
:3000  
-----
```

You can then open the Bot Emulator and configure your endpoint as follows.

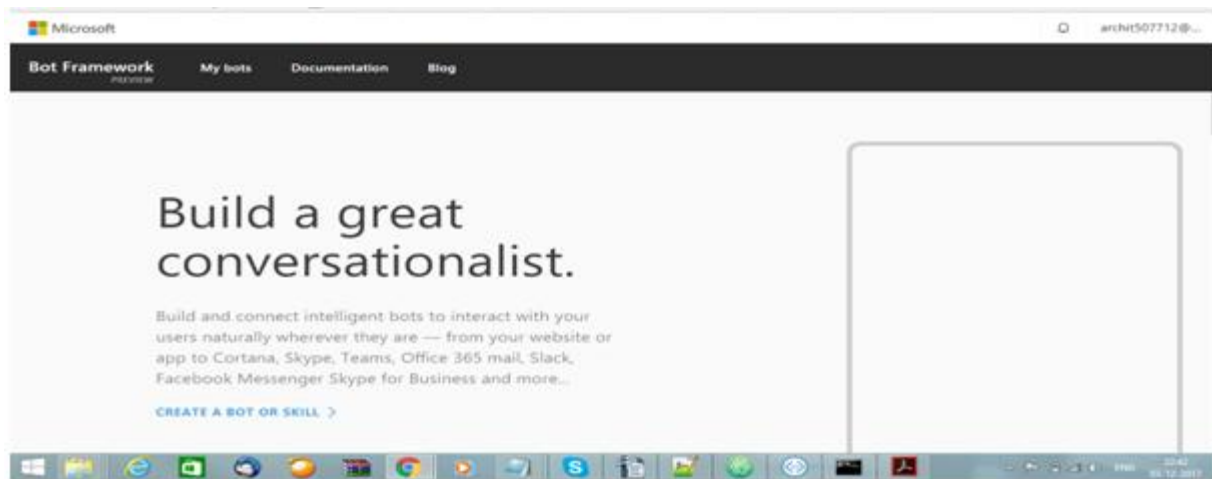


You can then send a message and receive the **“Hello World”** response.

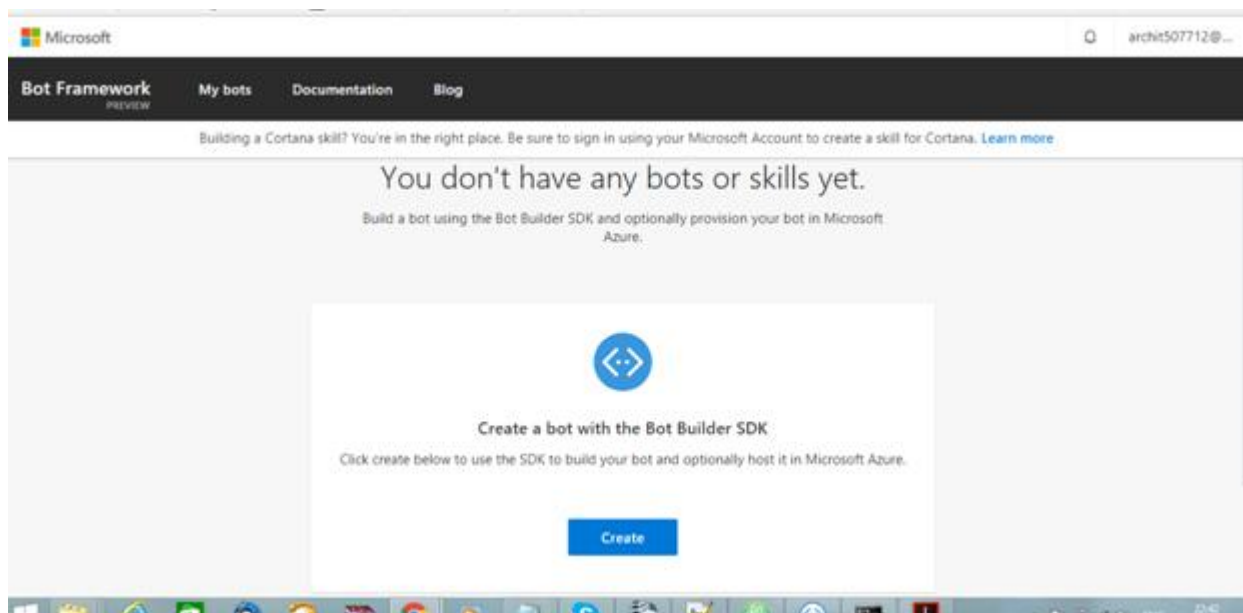


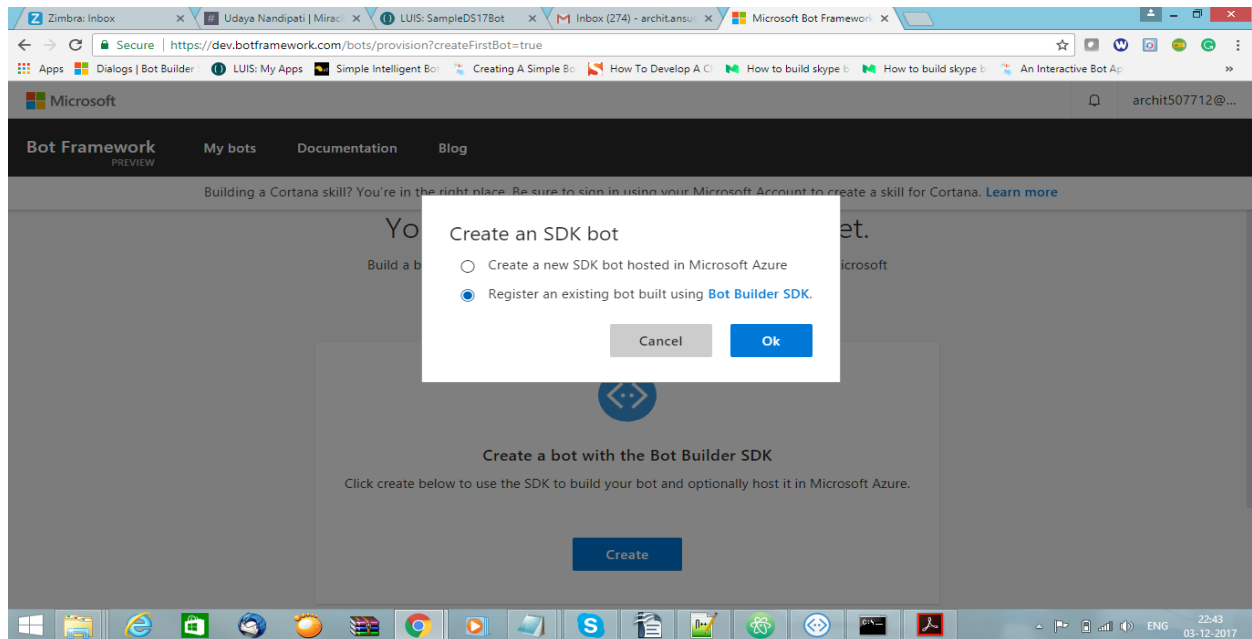
#2 | Register and Test with the Bot Framework

Login at <https://dev.botframework.com/> using your Microsoft ID.



Once you are signed in, click on **“Register a Bot”** to register a new bot.

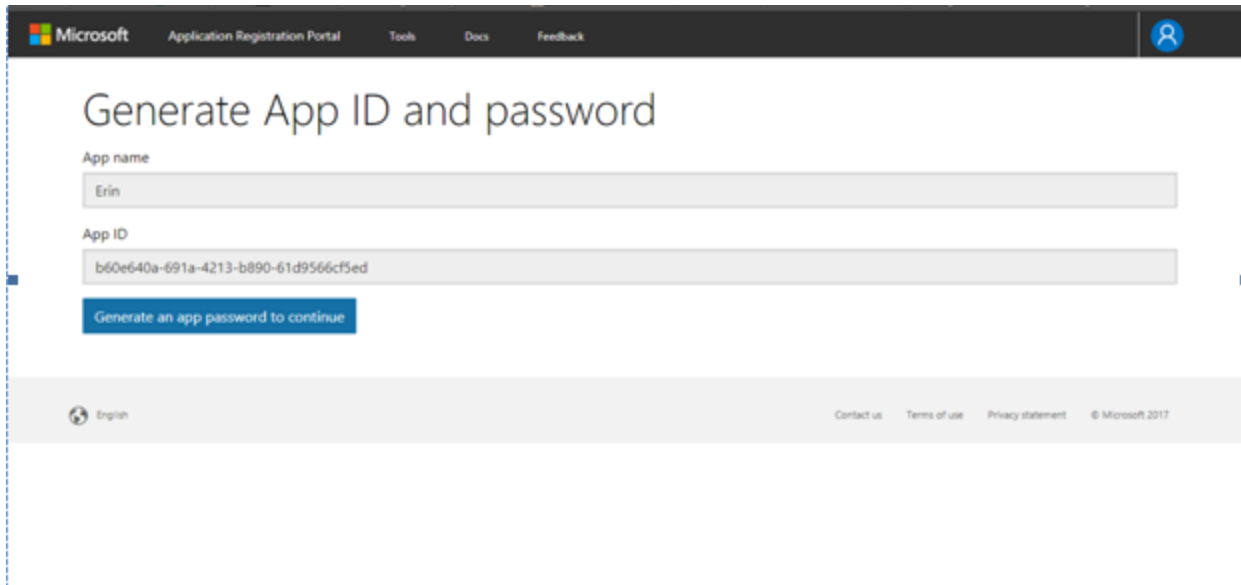




Fill in the required details as shown below.

A screenshot of the "Tell us about your bot" form in the Microsoft Bot Framework developer portal. The form is titled "Bot profile" and includes an "Icon" section with a blue circular icon containing a white code symbol and the text "Upload custom icon (30K max, png only)". Below the icon section are three required fields: "Display name" (containing "Erin"), "Bot handle" (containing "Erin"), and "Long description" (containing "Bot which can recognise name and id of the user"). Each field has a small question mark icon to its left. The form is set against a light gray background with a dark header bar.

Create your new **APP_ID** and **APP_PWD**.



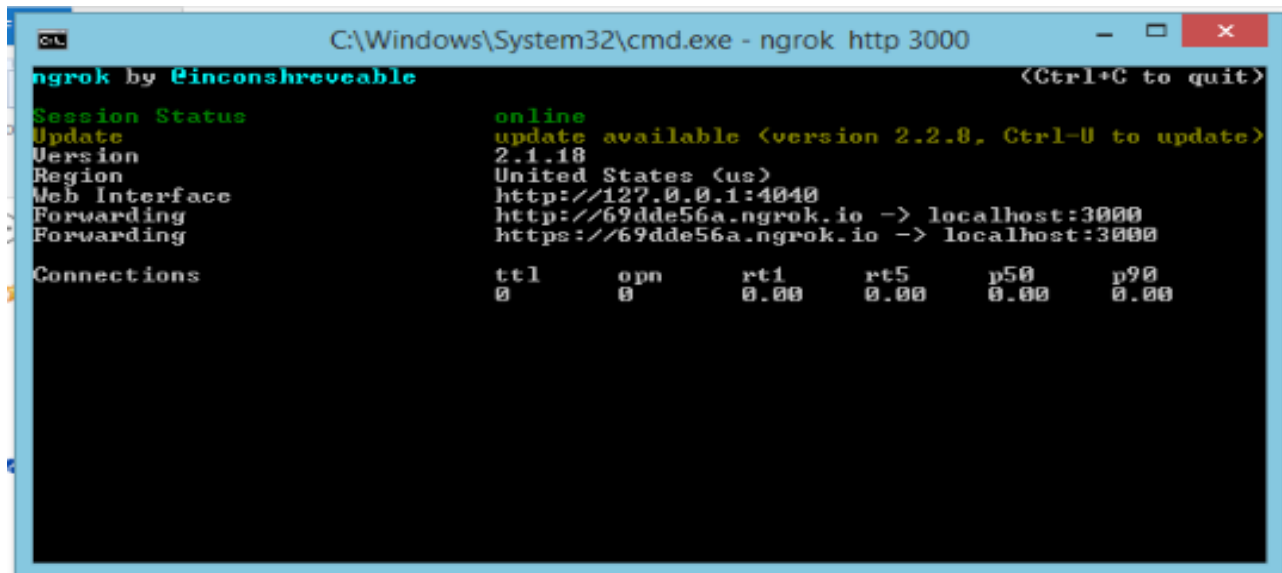
The screenshot shows the Microsoft Application Registration Portal. The header includes the Microsoft logo, 'Application Registration Portal', and links for 'Tools', 'Docs', and 'Feedback'. A user profile icon is in the top right. The main heading is 'Generate App ID and password'. Below it, the 'App name' field contains 'Erin'. The 'App ID' field contains a long alphanumeric string: 'b60e640a-691a-4213-b890-61d9566cf5ed'. A blue button labeled 'Generate an app password to continue' is positioned below the App ID field. The footer shows a language selector set to 'English' and links for 'Contact us', 'Terms of use', 'Privacy statement', and '© Microsoft 2017'.

Configure your Bot credentials in your app.js file as shown below.

```
var connector = new builder.ChatConnector({
  appId: "<app_id>",
  appPassword: "<app_pwd>"
});
```

Now go ahead and run your backend once again with the below command and then open another terminal and start a **ngrok** tunnel for port **3000**.

```
node app.js
ngrok http 3000
```



```
C:\Windows\System32\cmd.exe - ngrok http 3000
ngrok by @inconshreveable (Ctrl+C to quit)

Session Status: online
Update: update available (version 2.2.0, Ctrl-U to update)
Version: 2.1.18
Region: United States (us)
Web Interface: http://127.0.0.1:4040
Forwarding: http://69dde56a.ngrok.io -> localhost:3000
              https://69dde56a.ngrok.io -> localhost:3000

Connections:
  ttl    opn    rt1    rt5    p50    p90
   0     0     0.00   0.00   0.00   0.00
```

Copy the above HTTPs Forwarding URL and add /api/messages to the end.

Configure the messaging endpoint in the Bot Framework registration.

Configuration

Messaging endpoint

<https://69dde56a.ngrok.io/api/messages>

Register your bot with Microsoft to generate a new App ID and password

[Manage Microsoft App ID and password](#)


Go ahead and save the registration, click on Test Connection and then test it out.

Step #2 – Integrating your bot with Skype

Once we have a sample bot up and running, we can now integrate with the channel of our choice – in this lab we will see how you can integrate your bot with Skype.

#1 | Configure Skype in Bot Framework Channels

Note : Please create your Skype Account before you start this lab.



Sign in

Skype name, email or phone number

Sign in

Create new account





[Problems signing in?](#) [Sign in with Facebook](#)

To add the Bot to Skype, click on Add to Skype button on the bot configuration page by following the link <http://dev.botframework.com> for creating a Skype Application for your bot.

[Bot Framework](#) [My bots](#) [Documentation](#) [Blog](#) [Archit](#)











Erin [CHANNELS](#) [ANALYTICS](#) [SETTINGS](#) [← Test](#)

Connect to channels


Name	Health	Published	
 Skype	Running	--	Edit 
 Web Chat	Running	--	Edit 

[Get bot embed codes](#)


Add a channel

You will be directed to a **Skype** configuration page for the **Bot**.
Click the button to **Add to Contacts**.



Erin

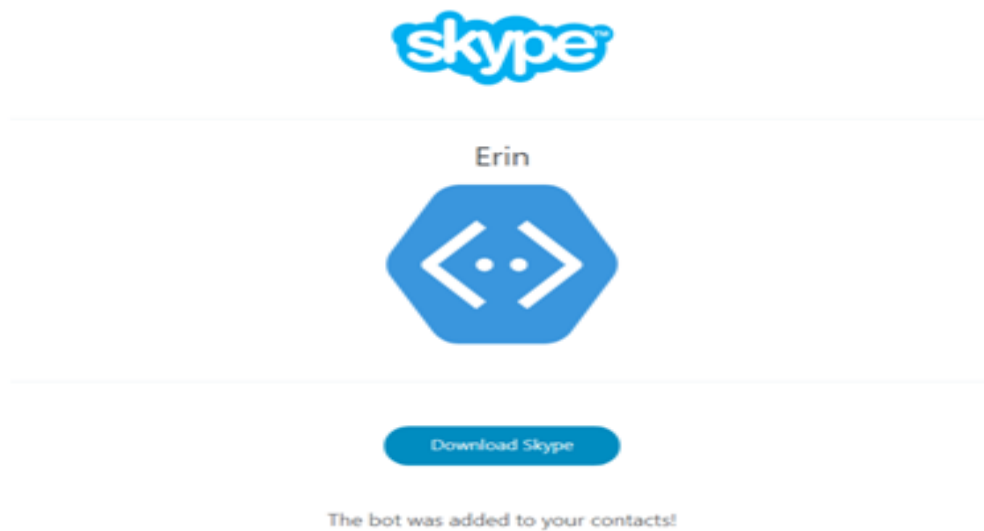


[Add to Contacts](#)

Sample Bot which can recognise Name and ID.

Capabilities

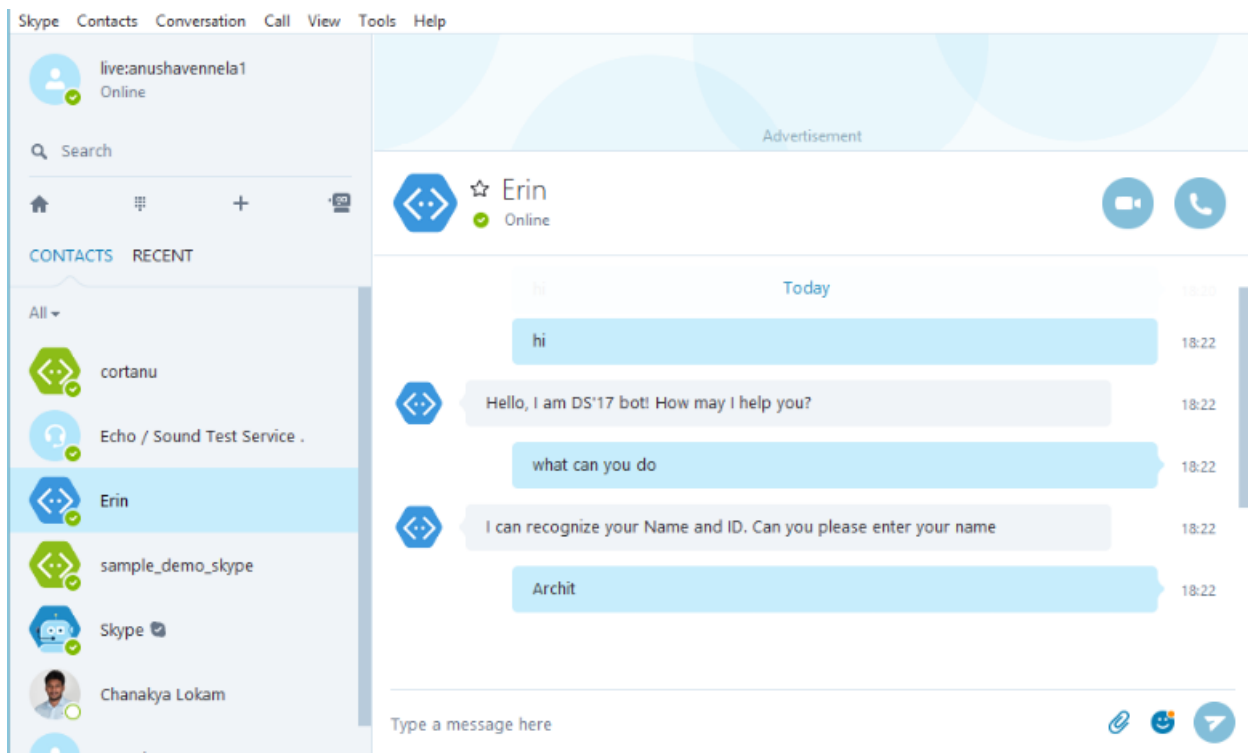
Send and receive instant messages and photos



Skype will open and you can select the **Bot** from your contacts and converse with it.

#2 | Test the bot with Skype

To verify your bot was created successfully, go back to your skype channel and under Direct Messages you should see the bot you created is now there. Open Skype, and navigate to ERIN and enter Hello ERIN, it will give response as **Hello! I'm Ds'17 Bot. How may I help you?**



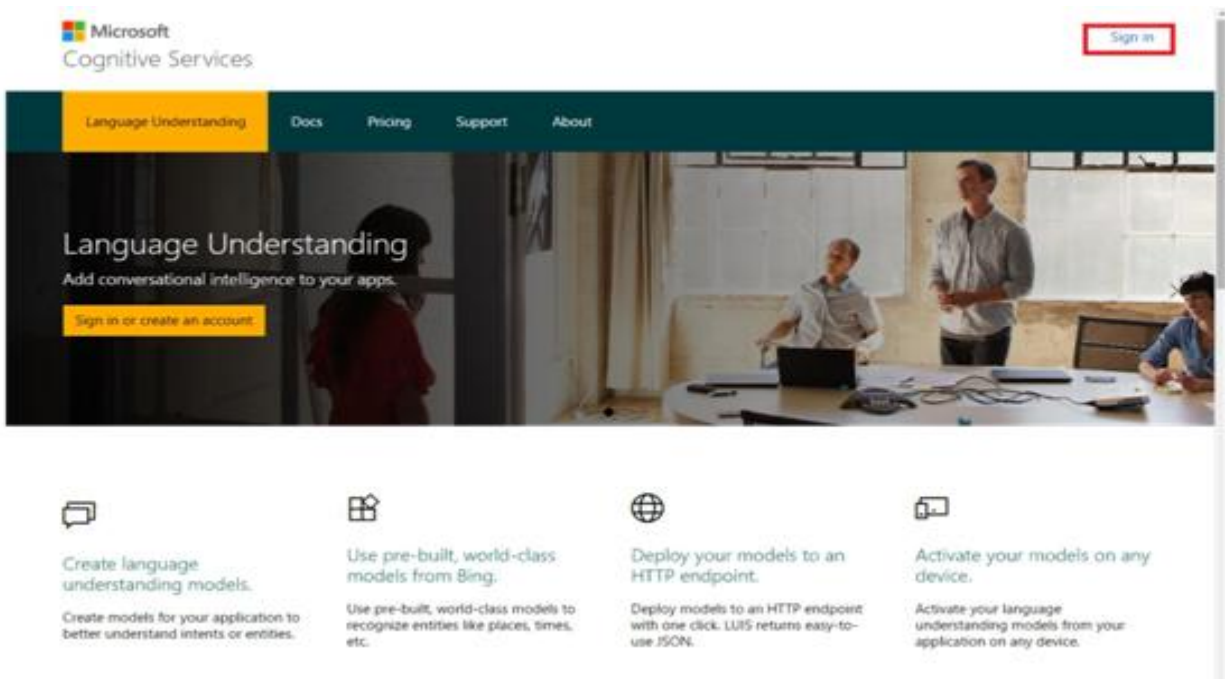
Lab #3 – Creating your LUIS Model

Our bot is pretty cool now, but it would be cooler if we can add Natural Language Understanding capabilities. In this lab you will add intents and entities to your LUIS Model and test them out.

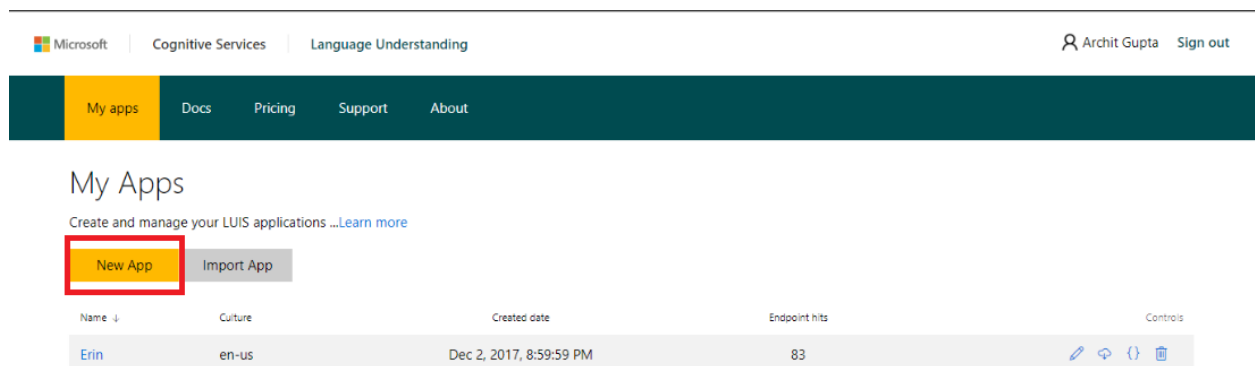
#1 | Create Your Entities and Intents

First, head over to the LUIS homepage at www.luis.ai where you will be making your dialog model.

Log in using your Microsoft account, and after a few moments your workspace will be configured. You will be prompted to enter information about yourself, and agree to the Terms of Service.



Once you have accepted, you will be forwarded to your My Apps page where you will create your Natural Language dialog model.



First click on **New App**, input a name, and then press **Create** .

You will then find yourself at the Dashboard for your newly created app, and can start adding Intents and Entities that form the framework of your bots' conversation.

Your first step will be to create and train an Intent. You can think of an intent as an action your end user wishes to carry out that is processed and executed by your bot, either by replying to the user with relevant dialog and information, or triggering your backend application to perform certain task.

To begin creating an intent, click on the Intent tab under your apps dashboard, and select **Add Intent**.

The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there's a navigation bar with 'My apps', 'Docs', 'Pricing', 'Support', and 'About'. The 'My apps' tab is active, showing a list of apps. The app 'Erin' is selected, with version '0.1'. On the left sidebar, 'Intents' is highlighted. The main content area shows the 'Intents' page for 'Erin'. It includes a search bar, a table with columns 'Intent Name' and 'Utterances', and a message: 'No custom intents yet. Add your first intent now.' The 'Add Intent' button is highlighted with a red box.

Intent Name	Utterances
None	0

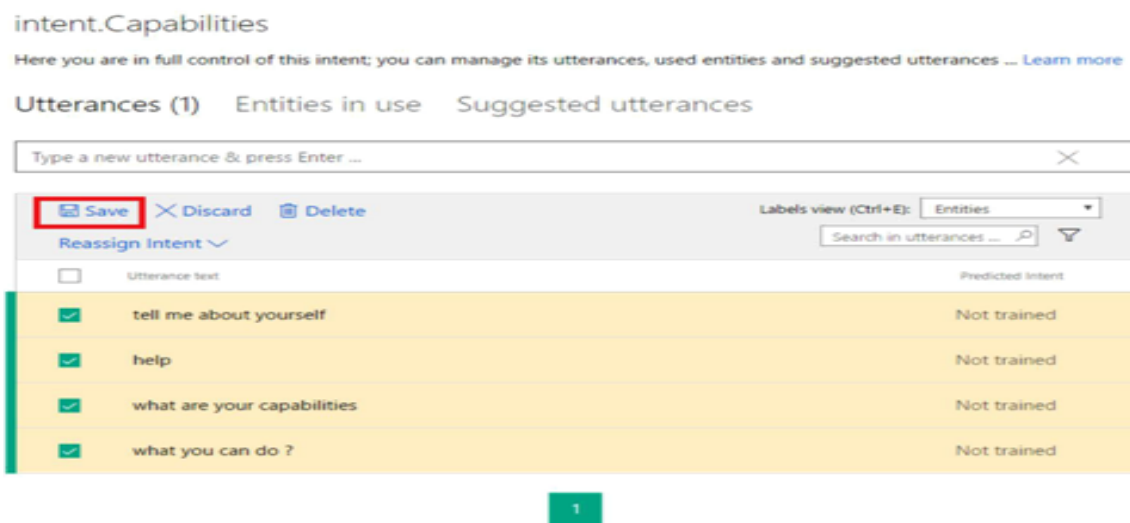
We will begin by adding a simple intent that responds with the capabilities of the bot when the end user is facing an issue or has an off topic question. Give the name **intent.Capabilities** to describe what the intent does, and click save.

The 'Add Intent' dialog box is shown. It has a title bar with a close button. The main content area has a label 'Intent name (REQUIRED)' and a text input field containing 'intent.Capabilities'. At the bottom, there are two buttons: 'Save' (highlighted in orange) and 'Cancel' (greyed out).

You will then need to train your Capabilities intent on example inputs or utterances the user might enter when wanting to know what your bot can do. Try adding a few example like

- What are your capabilities?
- What can you do
- Tell me about yourself
- Help

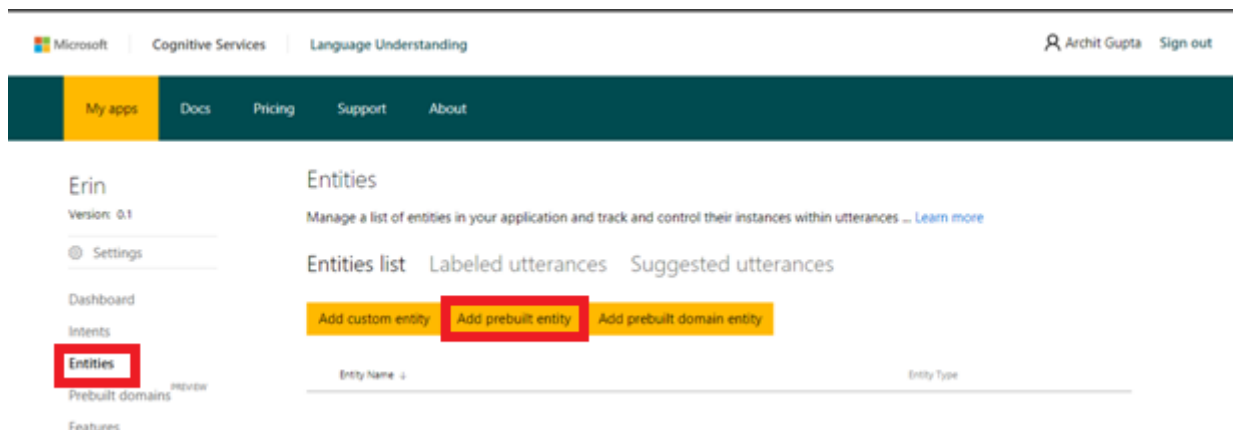
Once you have a few examples, click save to add your inputs to the training data used to make the model for this intent.



Now it is time to create your first entity. An entity can be thought of as a label for an object, topic, or key identifier for something specific an end user wants to perform an action on.

We will be using prebuilt entity that contains number which is related to single entity type.

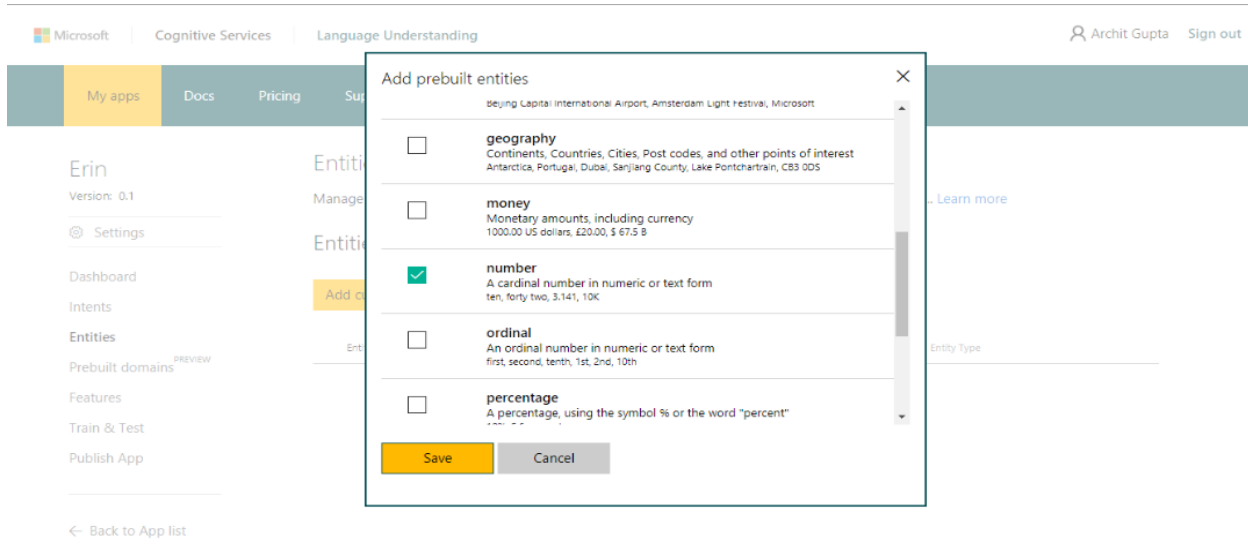
First navigate to the **Entities** tab under the Dashboard for your application.



Next, click **Add prebuilt entity**.

Now select number from the list of prebuilt entities.

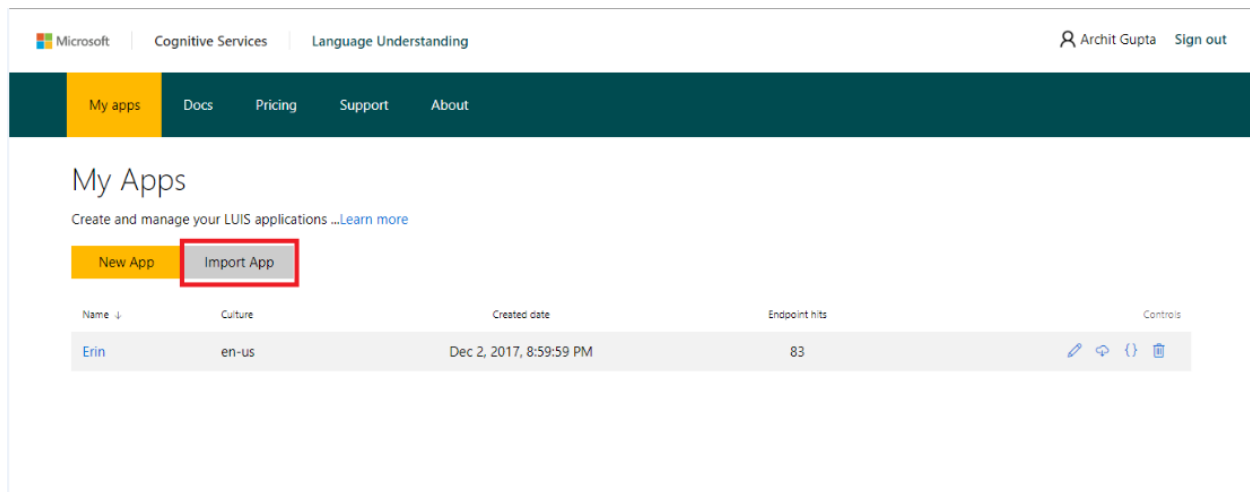
When finished your screen should look similar to this,



To finish your prebuilt entity, click Save and LUIS will add it to your application.

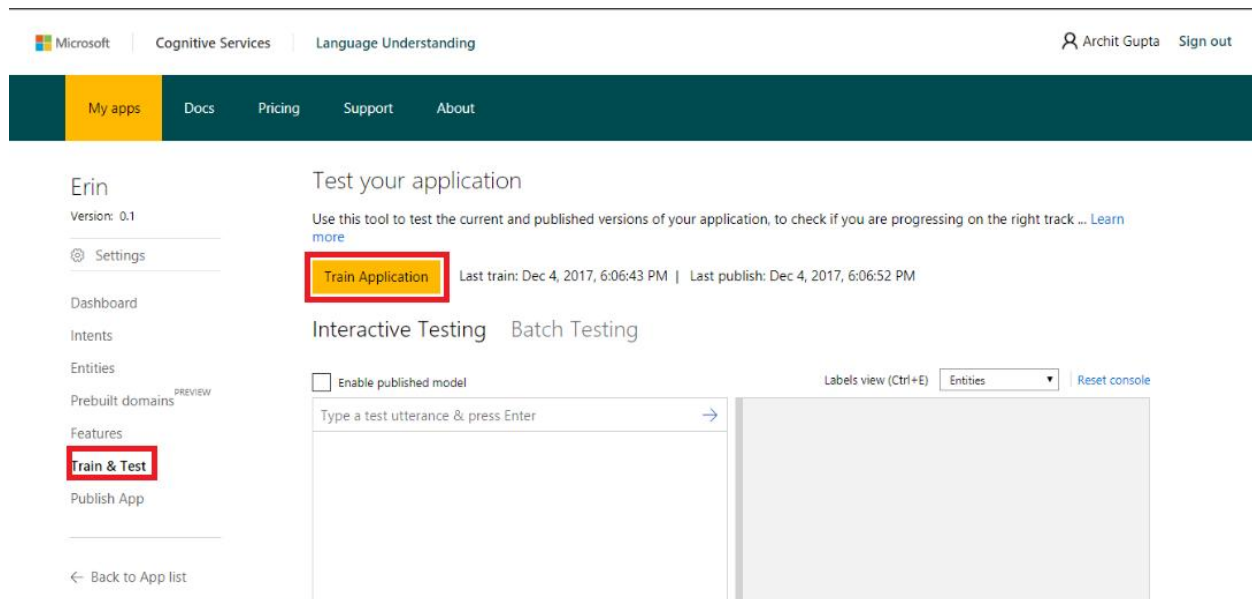
Now that you are familiar with creating an Intent and Entity, import our finished dialog model to LUIS where you will train and publish the finished app.

To add the finished model, first save the `/example/dialog.json` file from our Github repository to your desktop. Then click on My Apps on the top pane, click **Import App**, and upload the file you saved to your desktop.



#2 | Train, Test and Publish your LUIS App

Now that you have a completed LUIS model, you will need to train your application on your example utterances to create the model used for identifying your Intents. To do this, select **Train and Test** under your apps dashboard, and click **Train Application**.



LUIS will then create the model used for handling Natural Language. Make note that you will need to re-train your model when entering new intents, entities, or example utterances to have your bots dialog reflect your new changes.

Once the training is completed, try entering the below sentences in the Interactive Testing window to see how LUIS classifies intents and labels your entities,

- Hi, how are you?
- What are your capabilities?

You should see the top scoring intent for each utterance and your entity keywords labeled with the entity name they fall under.

The last step in creating your LUIS model is to publish your App and create the endpoint URL for integrating into your backend.

To do this, click on **Publish App** under the Dashboard, under publish to select Production or Staging and then click on **Publish to Production slot**.

Dashboard
Intents
Entities
Prebuilt domains
Features
Train & Test
Publish App

Publish to
Production

Timezone:
(GMT) Western Europe Time, London, Lisbon, Casablanca

☐ Enable verbose endpoint response
☐ Enable Bing spell checker

Publish to production slot

< Back to App list

Resources and Keys

Add Key

☒ US Regions ☐ Europe Regions ☐ Asia Regions

Resource Name	Region	Key String	Endpoint
Starter_Key	westus	af4dd6f3a59d4ed49f033aa3308418f9	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/70254391-c8c5-4b2a-9677-d7c551dc94f4?subscription-key=af4dd6f3a59d4ed49f033aa3308418f9&verbose=true&timezoneOffset=0&q=

After a few moments your application should be **published**, and you will be given an Endpoint URL. Make note of your URL as later labs will need it for integrating your LUIS dialog model.

Step #4 – Adding Basic Dialog Matching Capabilities

Once we have a LUIS Model we can now configure our backend to be able to handle these user intents. We will add a Recognizer to our code and create dialogs that match and respond to the user.

#1 | Add LUIS Recognizer to your Bot Backend

Navigate back to your backend project folder and run the following command,

```
npm install --save luis-sdk
```

Go back to your **app.js** file and replace the file with the following code,

```
//Add the modules that are required  
var restify = require('restify');  
var builder = require('botbuilder');
```

```
var LUIS = require('luissdk');

// Setup Restify Server

var server = restify.createServer();

server.listen(process.env.port || process.env.PORT || 3000, function () {

    console.log("-----");
    console.log( "Bot is running with the address : "+server.url);

    console.log("-----");
});

// Create chat bot

var connector = new builder.ChatConnector({
    appId:"<bot-app-id>",
    appPassword: "<bot-app-pwd>"
});

var bot = new builder.UniversalBot(connector);

var model = '<luissdk-model-publish-url>';

var recognizer = new builder.LuisRecognizer(model);

var dialog = new builder.IntentDialog({ recognizers: [recognizer] });

server.post('/api/messages', connector.listen());

//=====
```

```
/      Bots                                                    Dialogs
//=====
```

```
bot.dialog('/', dialog);
```

Note : Configure the above code with your LUIS and Bot Framework Credentials

#2 | Handle Basic Dialog Flows

We can now add a recognizer to match intents and respond back to user. **First add a default handler for any intents that are not recognized.**

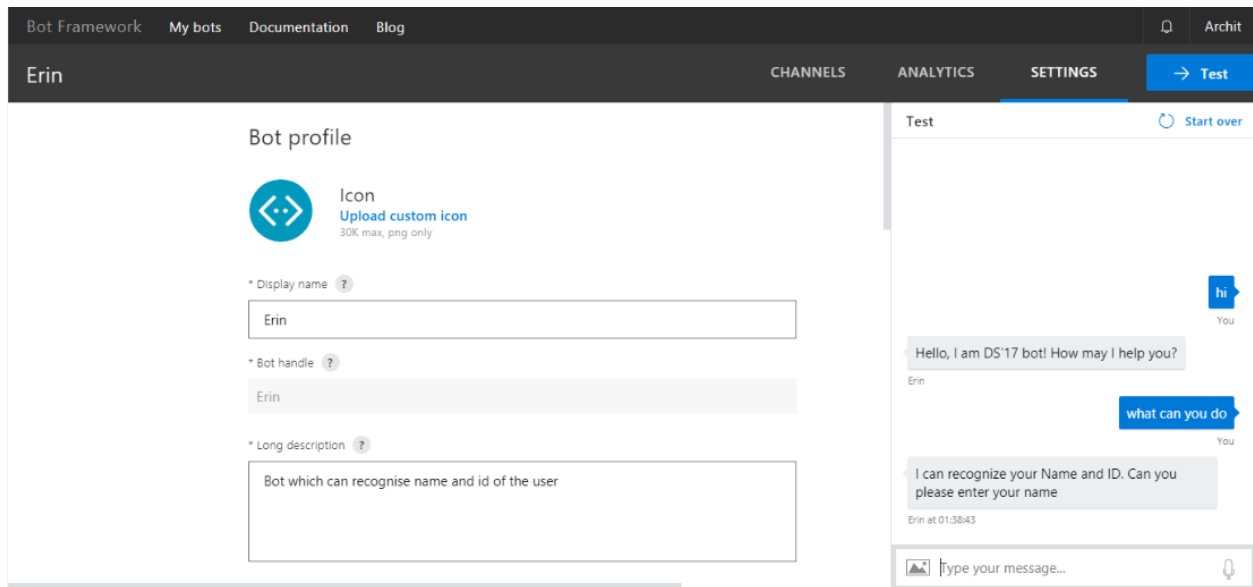
```
dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't understand. I'm the  
<BotName>Bot, how can I help you? "));
```

Then add a handler for the intent 'intent.Greetings'.

```
dialog.matches('intent.Greetings',[  
  function(session,args){  
    session.send("Hi, I'm <BotName>! How can I help you today?");  
  }  
]);
```

You can add more handlers for your other intents as well..After you have added all of the one-step dialogs your app.js file should look like this. Refer **app.js** file in **Git repository**

You can now run your code and **ngrok** once again and test the messages.



After you add all these steps your final **app.js** file should look similar to the file in the **/ds17-luis-lab/example/app.js** location of the GitHub repository. **Please make sure that you update your bot registration credentials in the below code.**

For any questions regarding the lab please feel free to reach out to innovation@miraclesoft.com. **We hope you enjoyed creating bots with us 😊**