+Creating a bot using the Microsoft Bot Framework In Node.js

Hands-on Lab Manual

Table of Contents

Lab Introduction

Objectives

After completing these self-paced labs, you will be able to:

- Have an understanding of the basics of the Bot Framework

Prerequisites

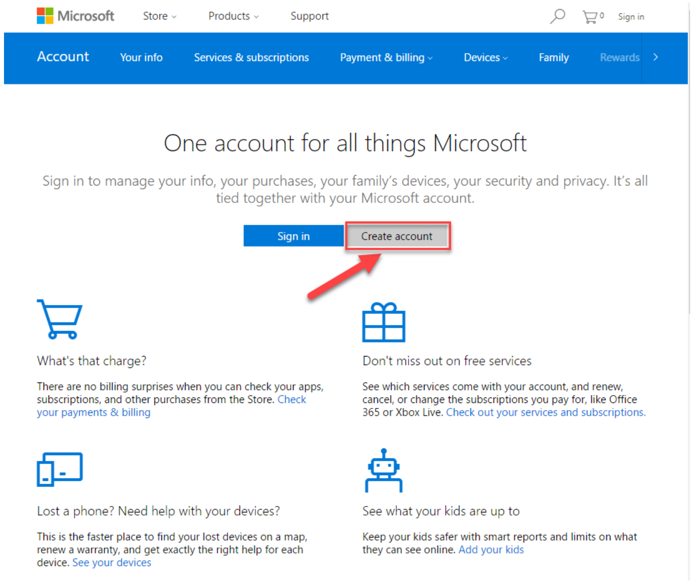
- Node.js
- A command line utility
- Basic understanding of Node.js & JavaScript

Lab Scenarios

This series of exercises is designed to show you how to get started using the Microsoft Bot Framework.

Configuration and Setup

1. Install prerequisite software
   - **Node.js** : https://nodejs.org/en/ (this will include NPM)
   - **NGrok** : https://ngrok.com/
     Skype : http://skype.com (if you want to test a Skype Bot)
   - **Visual Studio Code**. VS Code is cross platform, you can use others if you like (like sublime, Atom, Notepad++, Vim, etc..) but you will be on your own with debugging and usage.
     1. https://code.visualstudio.com/
2. **Create a Microsoft ID** (if you don't already have one)
   - Go to the Microsoft account sign-up page https://account.microsoft.com/ and **click Create account**.



One account for all things Microsoft

Sign in to manage your info, your purchases, your family's devices, your security and privacy. It's all tied together with your Microsoft account.

What's that charge?

There are no billing surprises when you can check your apps, subscriptions, and other purchases from the Store. Check your payments & billing

Don't miss out on free services

See which services come with your account, and renew, cancel, or change the subscriptions you pay for, like Office 365 or Xbox Live. Check out your services and subscriptions.

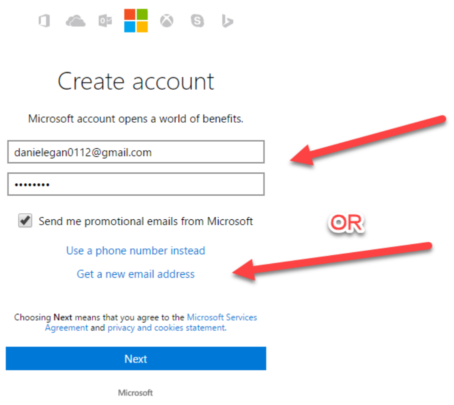
Lost a phone? Need help with your devices?

This is the faster place to find your lost devices on a map, renew a warranty, and get exactly the right help for each device. See your devices

See what your kids are up to

Keep your kids safer with smart reports and limits on what they can see online. Add your kids

   - In the User name box enter your existing email address, or click Get a new email address to create an Outlook or Hotmail address.



Create account

Microsoft account opens a world of benefits.

danielegan0112@gmail.com

••••••••

[✓] Send me promotional emails from Microsoft

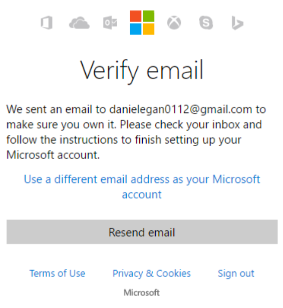
Use a phone number instead

Get a new email address

Choosing **Next** means that you agree to the Microsoft Services Agreement and privacy and cookies statement.

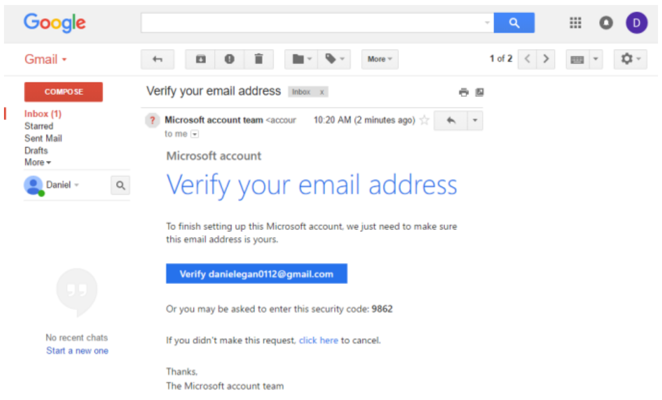
NOTE: **If you use an existing email address you will need to verify it**

before moving on.


Verify email

We sent an email to danielegan0112@gmail.com to make sure you own it. Please check your inbox and follow the instructions to finish setting up your Microsoft account.

Use a different email address as your Microsoft account



Verify your email address

Microsoft account team <accour 10:20 AM (2 minutes ago)
to me

Microsoft account

Verify your email address

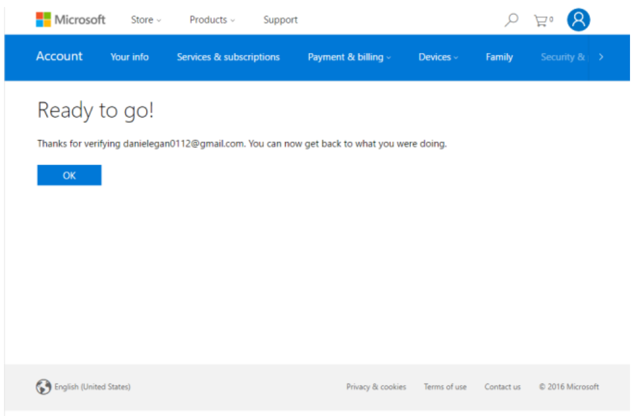
To finish setting up this Microsoft account, we just need to make sure this email address is yours.

Verify danielegan0112@gmail.com

Or you may be asked to enter this security code: 9862

If you didn't make this request, click here to cancel.

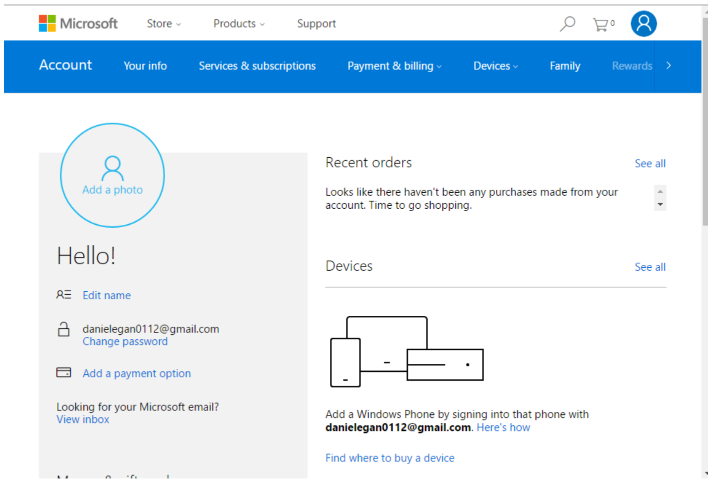
Thanks,
The Microsoft account team



Ready to go!

Thanks for verifying danielegan0112@gmail.com. You can now get back to what you were doing.

- Either path will take you to this screen



Hello!

danielegan0112@gmail.com

Looking for your Microsoft email?

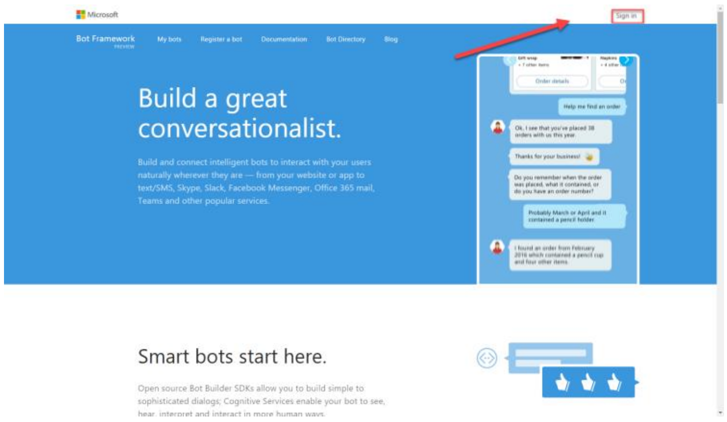
Recent orders

Looks like there haven't been any purchases made from your account. Time to go shopping.

Devices

Add a Windows Phone by signing into that phone with **danielegan0112@gmail.com**.

3. Create a BotFramework account
   - Navigate to http://BotFramework.com
   - Click on sign in



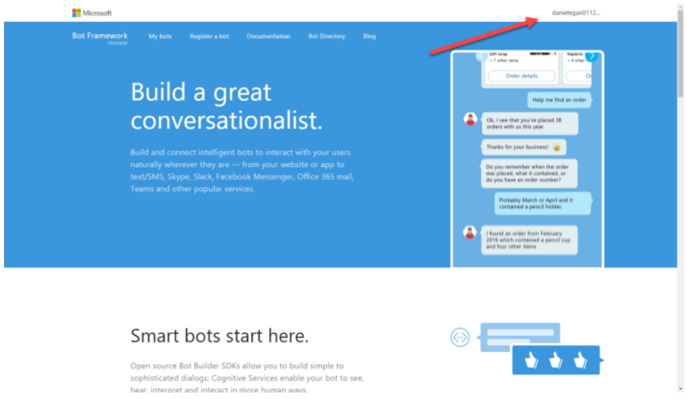
Build a great conversationalist.

Build and connect intelligent bots to interact with your users naturally wherever they are — from your website or app to text/SMS, Skype, Slack, Facebook Messenger, Office 365 mail, Teams and other popular services.

Smart bots start here.

Open source Bot Builder SDKs allow you to build simple to sophisticated dialogs; Cognitive Services enable your bot to see, hear, interpret and interact in more human ways.

   - If you are using the same browser that you used to create your Microsoft ID then you will be signed in automatically, otherwise you will need to use the ID you just created to sign in.



Build a great conversationalist.

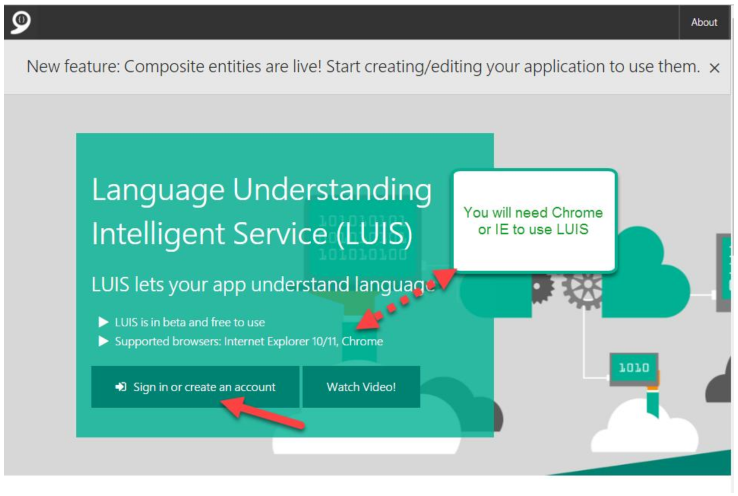
Build and connect intelligent bots to interact with your users naturally wherever they are — from your website or app to text/SMS, Skype, Slack, Facebook Messenger, Office 365 mail, Teams and other popular services.

Smart bots start here.

Open source Bot Builder SDKs allow you to build simple to sophisticated dialogs; Cognitive Services enable your bot to see, hear, interpret and interact in more human ways.

   - You can leave this window open, we will be using it later.

4. Sign-up for LUIS. Language Understanding Intelligent Services
    - https://www.luis.ai/
    - Click on: Sign in or Create Account button



About

New feature: Composite entities are live! Start creating/editing your application to use them. ×

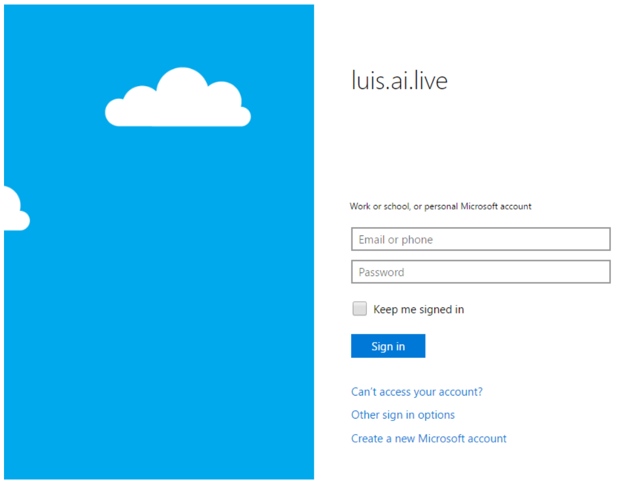
Language Understanding Intelligent Service (LUIS)

LUIS lets your app understand language

- LUIS is in beta and free to use
- Supported browsers: Internet Explorer 10/11, Chrome

Sign in or create an account
Watch Video!
You will need Chrome or IE to use LUIS
1010

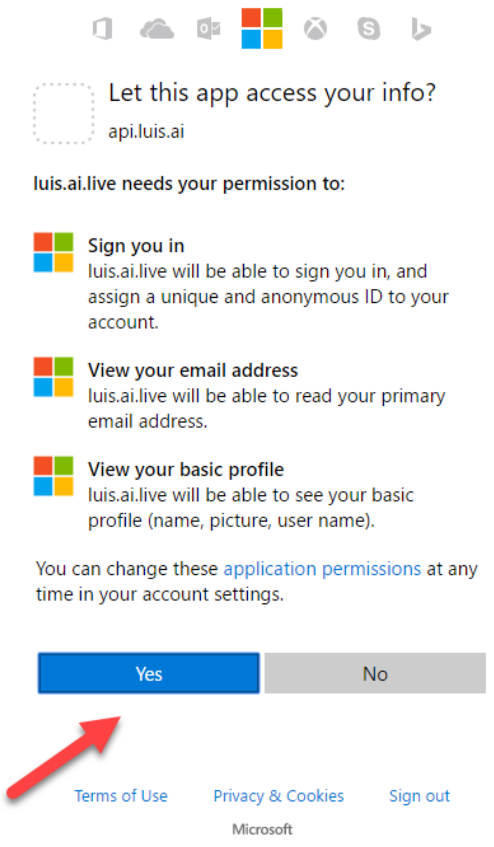
    - Sign in with your Microsoft account



luis.ai.live

Work or school, or personal Microsoft account

    - If you are still signed in it will ask you to say Yes to accept permissions. Otherwise you will need to sign in with the Microsoft ID you created earlier.



Let this app access your info?
api.luis.ai

luis.ai.live needs your permission to:

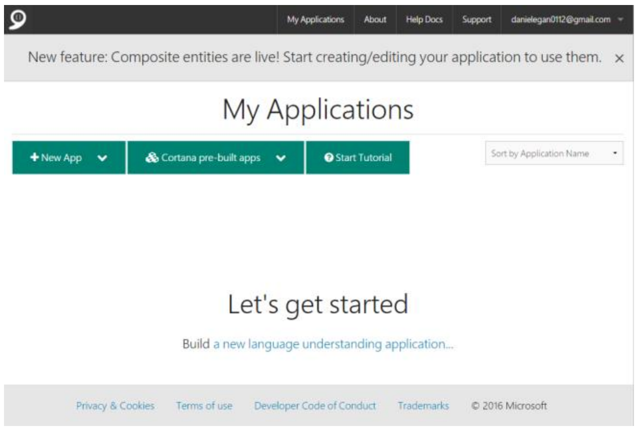
Sign you in
luis.ai.live will be able to sign you in, and assign a unique and anonymous ID to your account.

View your email address
luis.ai.live will be able to read your primary email address.

View your basic profile
luis.ai.live will be able to see your basic profile (name, picture, user name).

You can change these application permissions at any time in your account settings.

- You can walk through the quick walkthrough if you would like or click the x to close it. When finished, your screen should look like below.



New feature: Composite entities are live! Start creating/editing your application to use them.

My Applications

Let's get started

Build a new language understanding application...

- We will explain and use this later for our bot.

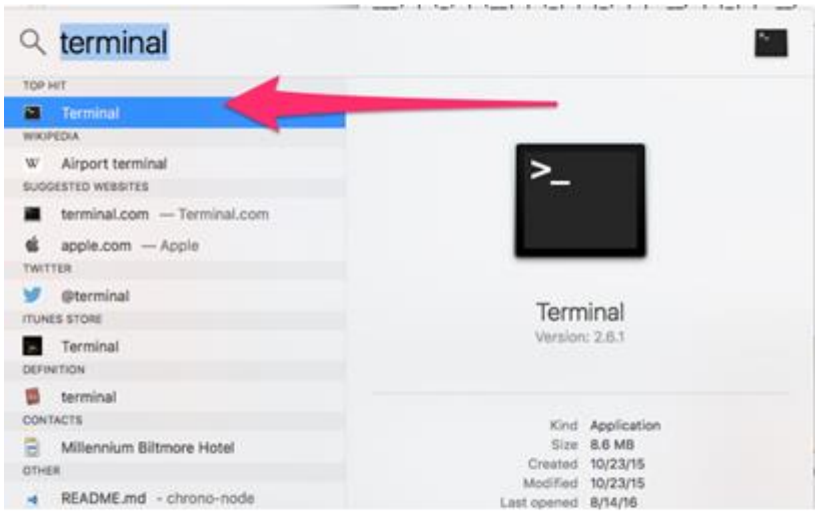
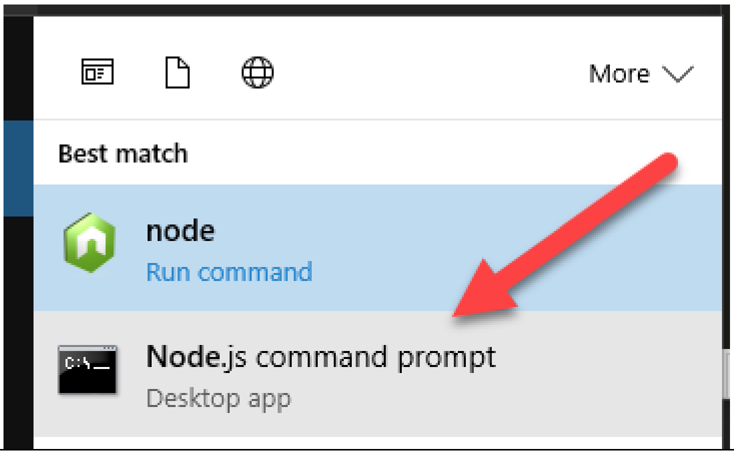
Copy/Paste of Code

You will have the option to copy/paste code snippets from this document to complete this lab. You will learn much more by typing it in yourself but sometimes in a lab format speed is needed to get through all the exercises in time.

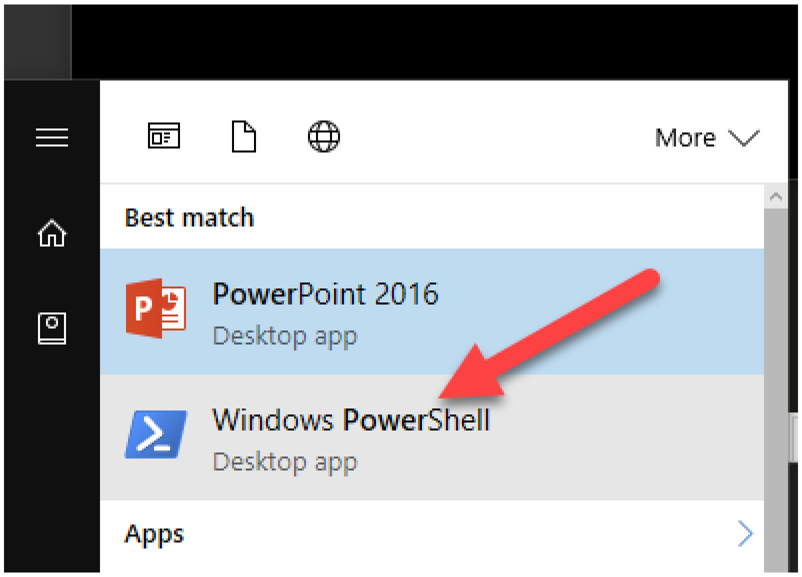
NOTE: If you are on a mac, you will be using the PDF file. Do not copy and paste from the PDF file. There is a separate file called SNIPS.txt that contain the snips you need.

Exercise 1: Basic Bot using BotBuilder

In this exercise, you will create a simple bot using the BotBuilder and

Detailed Steps
1. Open up your Terminal. (This could be Terminal on Mac, PowerShell on windows, or a terminal of your choice (cmndr, iTerm, etc…) On Mac hit **Command → Spacebar** and type terminal  terminal TOP HIT Terminal WIKIPEDIA Airport terminal SUGGESTED WEBSITES terminal.com — Terminal.com apple.com — Apple TWITTER @terminal ITUNES STORE Terminal DEFINITION terminal CONTACTS Millennium Biltmore Hotel OTHER README.md - chrono-node Terminal Version: 2.6.1 Kind Application Size 8.6 MB Created 10/23/15 Modified 10/23/15 Last opened 8/14/16 From windows hit the windows key and type node or PowerShell  More Best match node Run command Node.js command prompt Desktop app

Detailed Steps



2. Next we want to create a folder to hold our project and add some initial files. (personally I hold all project folders in a folder called projects (**C:\Projects on PC or my 'userfolder'/projects** on mac).

To create the folder, type the following from the command line to make a directory (mkdir)

```
~$ mkdir botworkshop
```

next we want to change to that directory (cd)

```
~$ cd botworkshop
```

next we will initialize it with a **package.json** file. We will use the -**y** so that it gives us the default values.

```
~$ npm init -y
```

the next step is to install the botbuilder npm module

```
~$ npm install botbuilder --save
```

finally, we want to open all of this up in **VSCode** type the following (the word code with a space then a period)

```
~$ code .
```

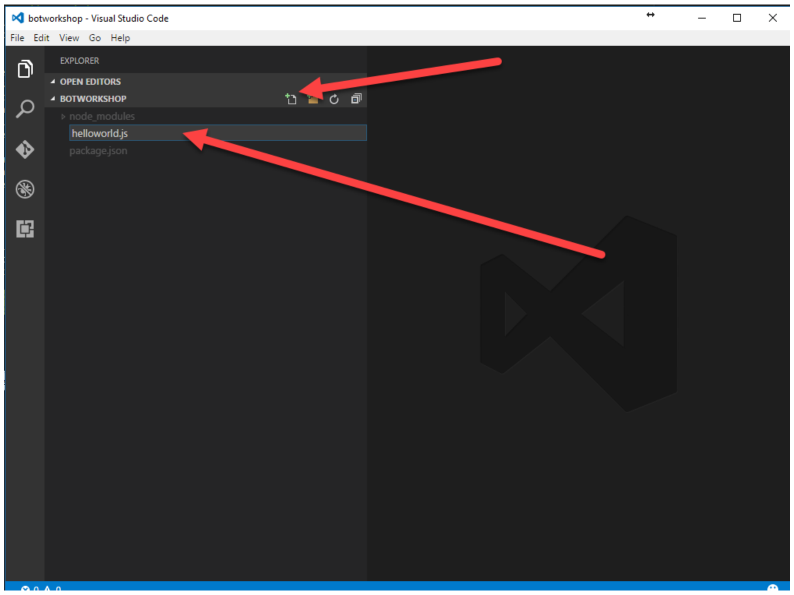
NOTE: You can also open up VSCode and select **File → Open (MAC) File →Open folder (PC)** and select the botworkshop folder we created)

Detailed Steps

This will open up our project in VSCode (again, you are free to choose your own editor if you like)

NOTE: Once you are in VSCode you can press CTRL + ` to open up a terminal window within the program

3. Now we want to create our first simple hello world bot. Click on the **Add File** icon, and name the file **helloworld.js**



4. In the file you just created, either type in or paste the following code.
NOTE: if you are using the PDF file, you can find the snippets in a separate file called SNIPS.txt

------SNIP1---

```
//Talking with the user.. as simple as possible
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

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

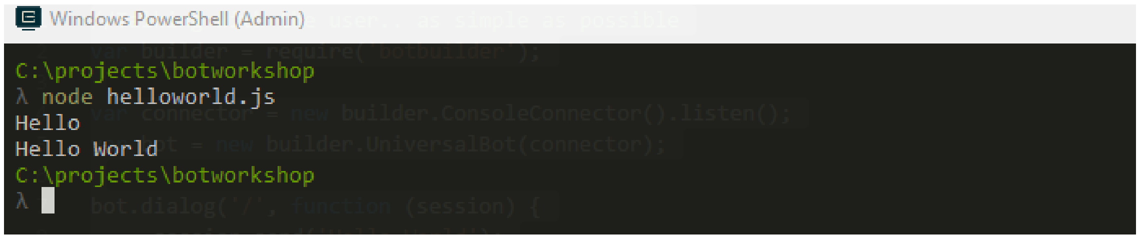
5. Now go back to your command prompt (terminal, powershell, etc) and type the following (make sure you are still in the projects/botworkshop folder)

```
~$ node helloworld.js
```

Nothing should happen, Well, you should not get any errors anyway. If all went well, it should be “listening” for you.

Detailed Steps

Type the word Hello (or anything really) and you should get back a “Hello World” from your bot.

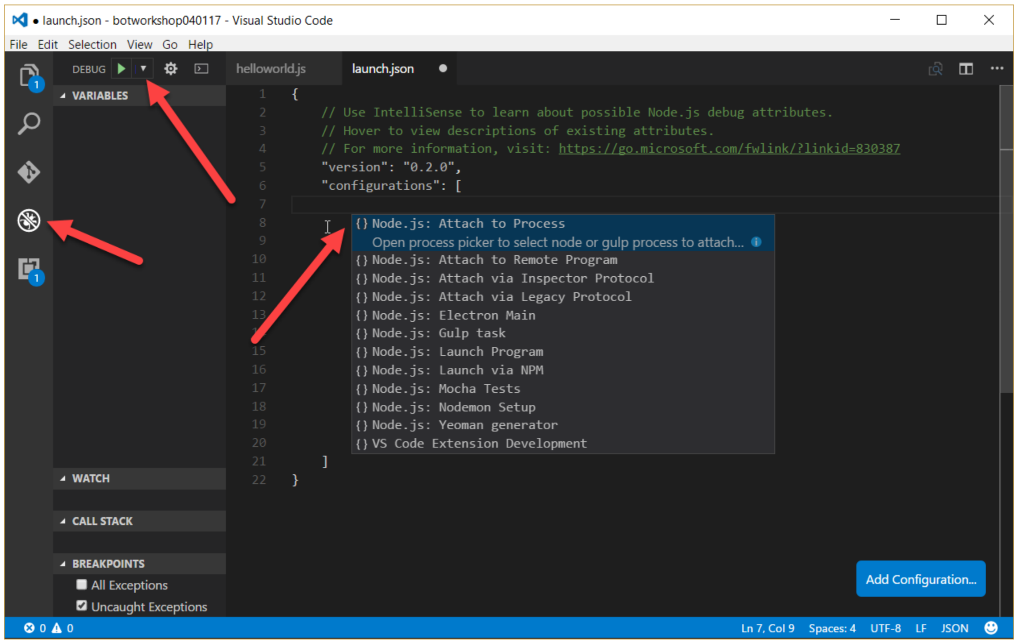


```
C:\projects\botworkshop
λ node helloworld.js
Hello
Hello World
C:\projects\botworkshop
λ
```

6. Hit **Ctl +C** to exit out of the bot and back to command line.

7. Now we want to setup a debug session. It is best to do this on the simple bot so you can get used to using the debugger.

   > In VSCode, click on the bug icon (Left side, second from bottom)
   > On the top you can see that there is no config file.
   > Click on the down arrow to the right of the green play button and select “Add Configuration”
   > This will produce a drop down, select Node.js

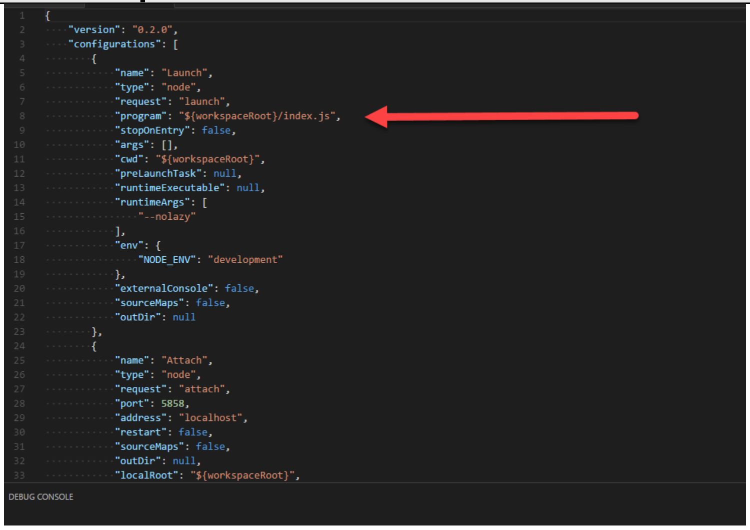

● launch.json - botworkshop040117 - Visual Studio Code
File Edit Selection View Go Help
DEBUG
helloworld.js
launch.json
VARIABLES
WATCH
CALL STACK
BREAKPOINTS
All Exceptions
Uncaught Exceptions
Add Configuration...
Ln 7, Col 9 Spaces: 4 UTF-8 LF JSON

```
{
    // Use IntelliSense to learn about possible Node.js debug attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [

    ]
}
```

```
{} Node.js: Attach to Process
   Open process picker to select node or gulp process to attach...
{} Node.js: Attach to Remote Program
{} Node.js: Attach via Inspector Protocol
{} Node.js: Attach via Legacy Protocol
{} Node.js: Electron Main
{} Node.js: Gulp task
{} Node.js: Launch Program
{} Node.js: Launch via NPM
{} Node.js: Mocha Tests
{} Node.js: Nodemon Setup
{} Node.js: Yeoman generator
{} VS Code Extension Development
```

8. This will create a **launch.json** file. Open this file and change the **“program”** attribute from **index.js** to **helloworld.js**

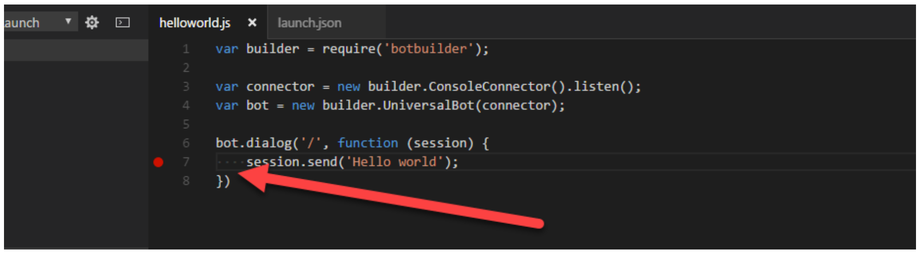
Detailed Steps



```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch",
            "type": "node",
            "request": "launch",
            "program": "${workspaceRoot}/index.js",
            "stopOnEntry": false,
            "args": [],
            "cwd": "${workspaceRoot}",
            "preLaunchTask": null,
            "runtimeExecutable": null,
            "runtimeArgs": [
                "--nolazy"
            ],
            "env": {
                "NODE_ENV": "development"
            },
            "externalConsole": false,
            "sourceMaps": false,
            "outDir": null
        },
        {
            "name": "Attach",
            "type": "node",
            "request": "attach",
            "port": 5858,
            "address": "localhost",
            "restart": false,
            "sourceMaps": false,
            "outDir": null,
            "localRoot": "${workspaceRoot}",
```

NOTE: The last step is not used in this HOL because we will be using Attach as opposed to LAUNCH but it is good to be aware of that field for the future.

9. Next open the **helloworld.js** file and put a breakpoint next to the **session.send('Hello World');** line by clicking next to the line number.



```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

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

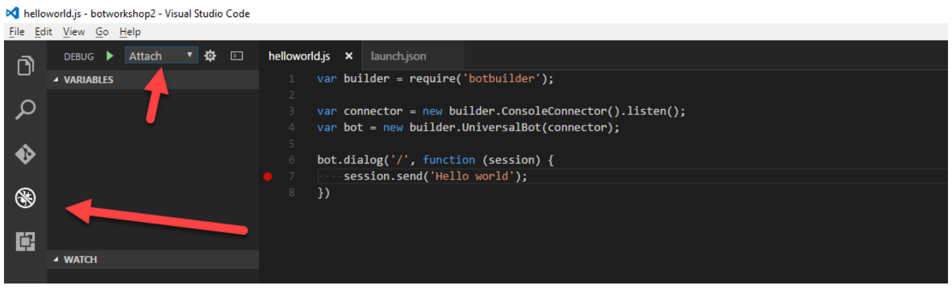
10. Now we want to run our program in debug mode. Open up your console (Terminal, Powershell, etc) and type the following:

```
~$ node --debug-brk helloworld.js
```

11. That should run the node process in debug mode listening on port 5858

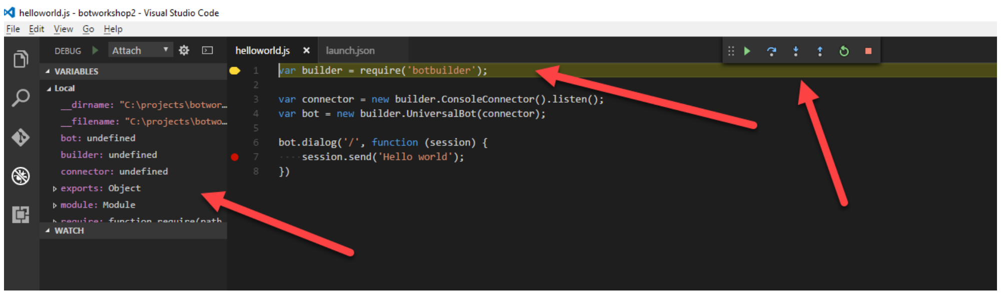
12. In VS Code, make sure you are still in the debug panel, in the debug dropdown make sure you select **attach** (to attach to the process we just started that is running on port 5858) and click on the green debug arrow to run it.

Detailed Steps



```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

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

13. This will attach to the process and the debug will be stopped on the first line of the program (that's what using **--debug-brk** does as opposed to just **--debug**) you will be able to view all the local variables on the left, step through the code (Function keys or buttons on top) and inspect variables by hovering over them. You can hit F5 to run the program (or the green arrow on the top bar.



```
__dirname: "C:\projects\botwor…
__filename: "C:\projects\botwo…
bot: undefined
builder: undefined
connector: undefined
exports: Object
module: Module
require: function require(path
```

```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

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

Spend time debugging and looking around in this simple example so you can debug more complex ones later.

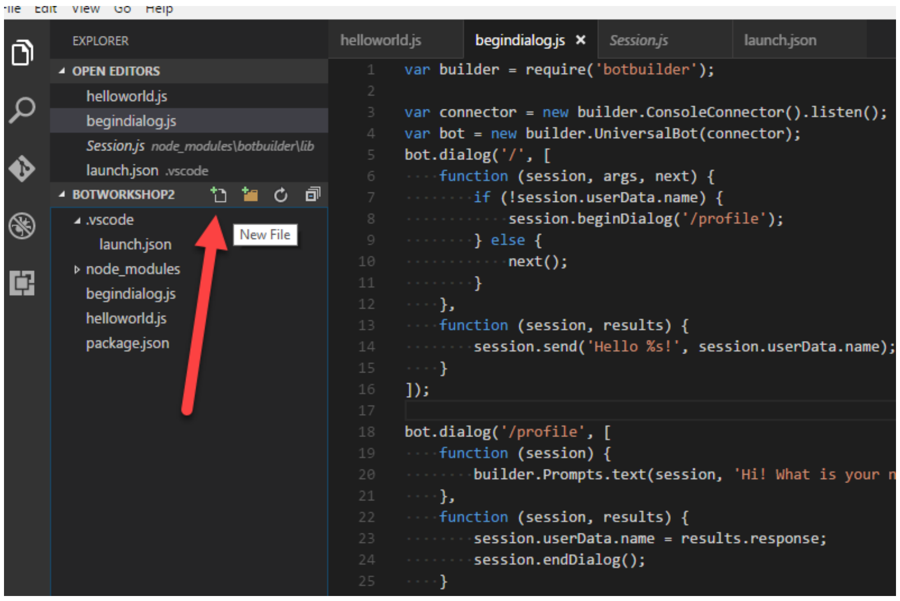
That is the end of exercise one. Now that we have everything set up and a simple bot running we will talk about how to handle greater complexity when working with your bot.

Exercise 2: Using prompts in a bot

In this exercise we will create a very simple bot so that you can get used to using different prompts in a waterfall, and collecting data.

Detailed Steps

1. Open up Visual Studio Code in the same folder (project) we started earlier. Add a new file by clicking on the new file icon and name it **promptbot.js**



```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);
bot.dialog('/', [
    function (session, args, next) {
        if (!session.userData.name) {
            session.beginDialog('/profile');
        } else {
            next();
        }
    },
    function (session, results) {
        session.send('Hello %s!', session.userData.name);
    }
]);

bot.dialog('/profile', [
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your n
    },
    function (session, results) {
        session.userData.name = results.response;
        session.endDialog();
    }
```

2. First, we need to add the require for botbuilder, and to new up our connector and bot just like we have done before.
Add the following code to the top of the **promptbot.js** file.

----- SNIP2----------------------------------

```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);
```

Next we need to add our dialog. It is one dialog, so let's look at the whole thing at once. (It makes it easier to copy/paste as well) Paste the following code directly below our bot variable **(var bot = new …)** in **promptbot.js**

------SNIP3--------------------------------------

```
bot.dialog('/', [
   function (session) {
        builder.Prompts.text(session, "Hello... What's your name?");
    },
    function (session, results) {
        session.userData.name = results.response;
```

Detailed Steps

```
        builder.Prompts.number(session, "Hi " + results.response + ", 
How many years have you been coding?");
    },
    function (session, results) {
        session.userData.coding = results.response;
        builder.Prompts.choice(session, "What language do you code Node 
using?", ["JavaScript", "CoffeeScript", "TypeScript"]);
    },
    function (session, results) {
        session.userData.language = results.response.entity;
        session.send("Got it... " + session.userData.name + 
                    " you've been programming for " + 
session.userData.coding + 
                    " years and use " + session.userData.language + 
".");
    }
]);
```

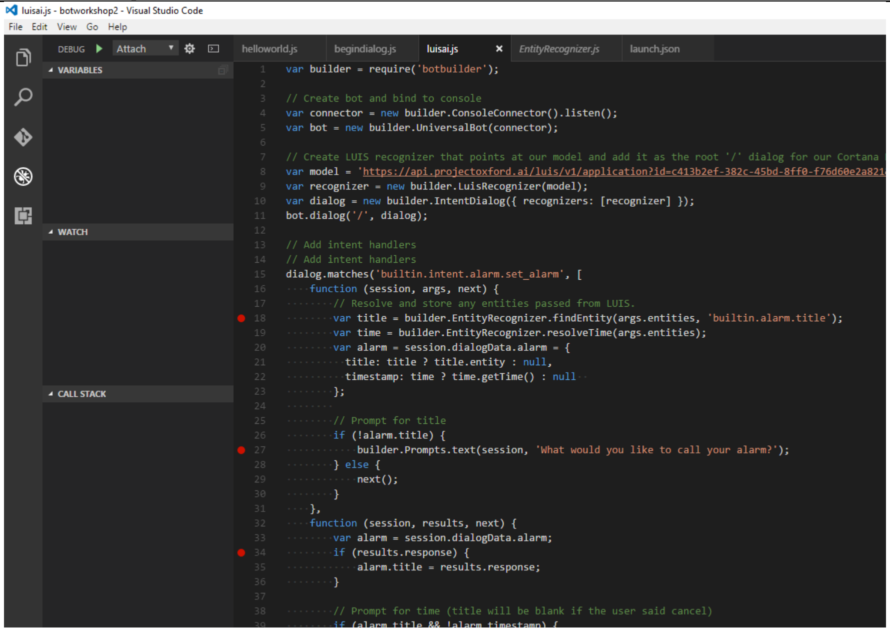
In this code we are creating a function array (between the []) so it will cascade from one question to the next. Within the waterfall we are using 3 different prompts (text, number, choice) and storing data for the user in **userData**. We then pull the data out for the final **session.send** message.

4. Now we can run the code. Go to your command prompt and type in the following.

```
~$ node --debug promptbot.js
```

5. Next go to Visual Studio Code and place some breakpoints in your code so you can step through and inspect the objects and flow.

Detailed Steps



```
var builder = require('botbuilder');

// Create bot and bind to console
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

// Create LUIS recognizer that points at our model and add it as the root '/' dialog for our Cortana
var model = 'https://api.projectoxford.ai/luis/v1/application?id=c413b2ef-382c-45bd-8ff0-f76d60e2a821
var recognizer = new builder.LuisRecognizer(model);
var dialog = new builder.IntentDialog({ recognizers: [recognizer] });
bot.dialog('/', dialog);

// Add intent handlers
// Add intent handlers
dialog.matches('builtin.intent.alarm.set_alarm', [
    function (session, args, next) {
        // Resolve and store any entities passed from LUIS.
        var title = builder.EntityRecognizer.findEntity(args.entities, 'builtin.alarm.title');
        var time = builder.EntityRecognizer.resolveTime(args.entities);
        var alarm = session.dialogData.alarm = {
          title: title ? title.entity : null,
          timestamp: time ? time.getTime() : null
        };

        // Prompt for title
        if (!alarm.title) {
            builder.Prompts.text(session, 'What would you like to call your alarm?');
        } else {
            next();
        }
    },
    function (session, results, next) {
        var alarm = session.dialogData.alarm;
        if (results.response) {
            alarm.title = results.response;
        }

        // Prompt for time (title will be blank if the user said cancel)
        if (alarm.title && !alarm.timestamp) {
```

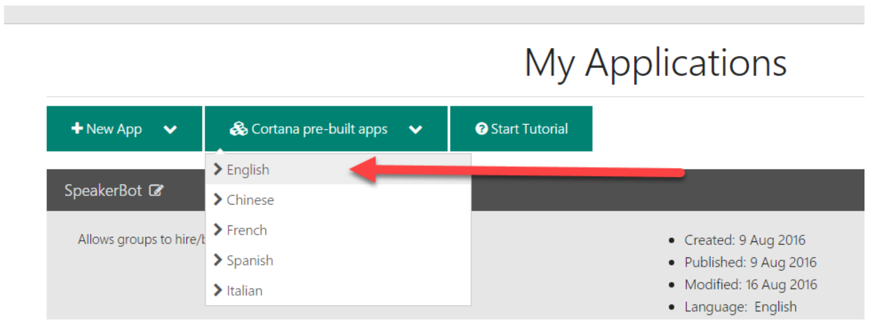
6. Step through the code as we had before to watch as things are running.

Exercise 3: Using Intent Dialogs (LUIS)

In this exercise we will create a LUIS Model. As discussed in the talk, we can create our own domain specific models, but for this HOL we will use Cortana Prebuilt App. The advantage is that we can quickly see the power of Natural Language in our bot without the training. This will create a personal assistant. To see all that this prebuilt model can do see the documentation here: https://www.luis.ai/Help/Index#PreBuiltApp

Detailed Steps

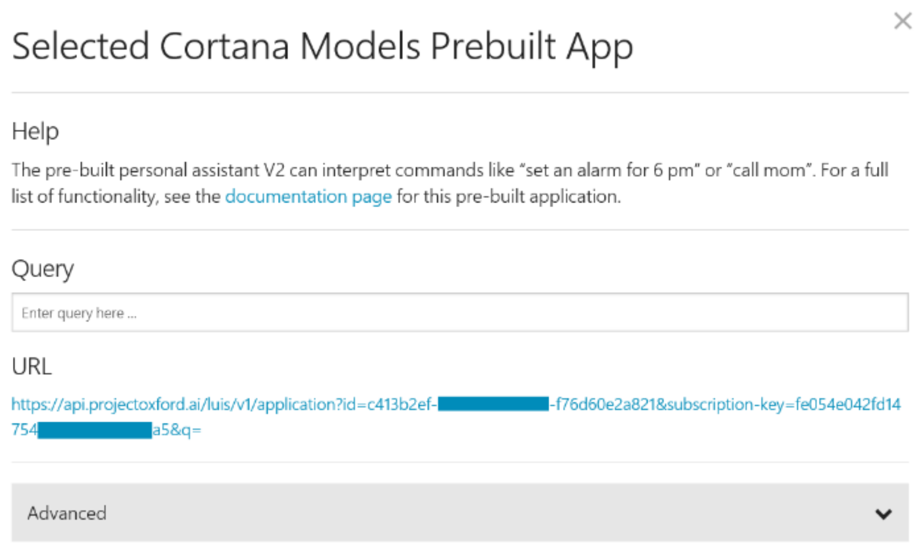
1. Sign on to http://www.LUIS.ai. You should have set this up in the first exercise, if not go back to the first section.
2. From your dashboard Select the **Cortana pre-built apps → English**



Allows groups to hire/t

- Created: 9 Aug 2016
- Published: 9 Aug 2016
- Modified: 16 Aug 2016
- Language: English

3. Once you click on your language, a Model dialog will pop up. **Copy the URL** from this screen, we will need it for our application.



Selected Cortana Models Prebuilt App

Help

The pre-built personal assistant V2 can interpret commands like “set an alarm for 6 pm” or “call mom”. For a full list of functionality, see the documentation page for this pre-built application.

Query

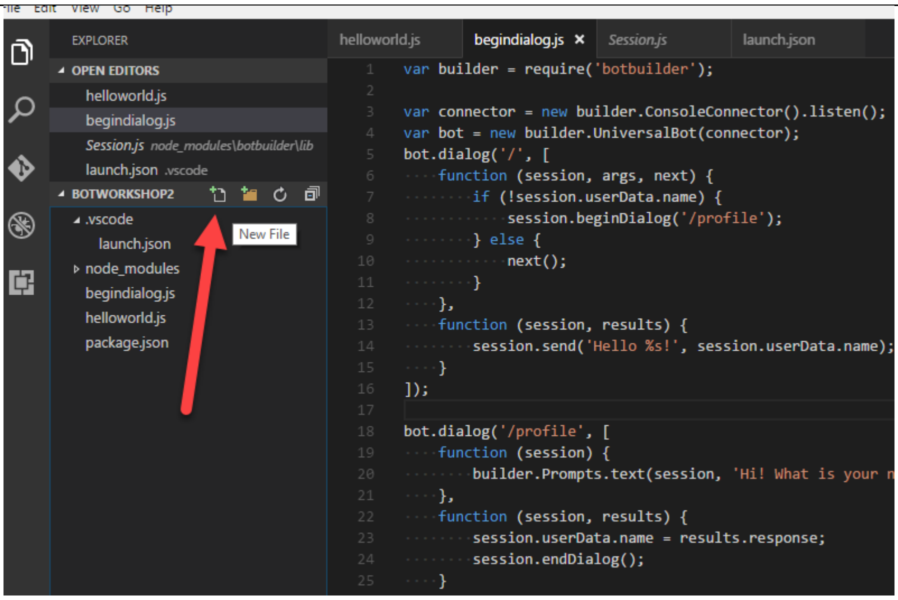
Enter query here ...

URL

https://api.projectoxford.ai/luis/v1/application?id=c413b2ef-[redacted]-f76d60e2a821&subscription-key=fe054e042fd14754[redacted]a5&q=

Advanced

4. Open up Visual Studio Code and in your project, create a file called luisai.js



```
var builder = require('botbuilder');

var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);
bot.dialog('/', [
    function (session, args, next) {
        if (!session.userData.name) {
            session.beginDialog('/profile');
        } else {
            next();
        }
    },
    function (session, results) {
        session.send('Hello %s!', session.userData.name);
    }
]);

bot.dialog('/profile', [
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your n
    },
    function (session, results) {
        session.userData.name = results.response;
        session.endDialog();
    }
```

5. Add the following code to the **luisai.js** file.

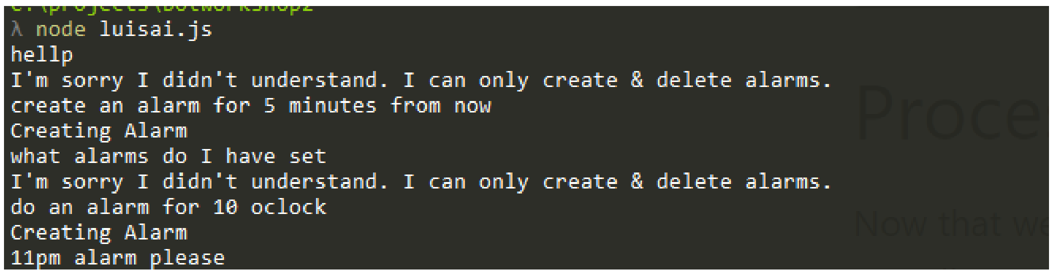
------SNIP4--

```
var builder = require('botbuilder');

// Create bot and bind to console
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

// Create LUIS recognizer that points at our model and add it as the
root '/' dialog for our Cortana Bot.
var model = '<your models url>';
var recognizer = new builder.LuisRecognizer(model);
var dialog = new builder.IntentDialog({ recognizers: [recognizer] });
bot.dialog('/', dialog);

// Add intent handlers
dialog.matches('builtin.intent.alarm.set_alarm',
builder.DialogAction.send('Creating Alarm'));
dialog.matches('builtin.intent.alarm.delete_alarm',
builder.DialogAction.send('Deleting Alarm'));
dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't
understand. I can only create & delete alarms."));
```

6. Replace **'<your models url>'** with the **url** that we copied in step 3
7. Move over to your console and run the following

```
~$ node luisai.js
```

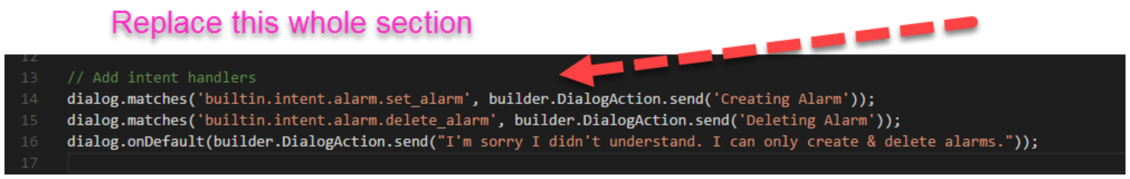
8. You can ask it to create or delete alarms for you. Try different ways of saying it and notice that you can ask it the same thing in many different ways and it understands (NLP).



```
λ node luisai.js
hellp
I'm sorry I didn't understand. I can only create & delete alarms.
create an alarm for 5 minutes from now
Creating Alarm
what alarms do I have set
I'm sorry I didn't understand. I can only create & delete alarms.
do an alarm for 10 oclock
Creating Alarm
11pm alarm please
```

Right now we are just gathering intent, we are not actually performing any actions except for printing to the screen things like **'Creating and Alarm'** or **'Deleting an Alarm'**. Let's add some functionality. This will allow us to inspect the intents coming back from LUIS and respond to them. We will start with the **.set_alarm intent**

So replace the following code:



```
// Add intent handlers
dialog.matches('builtin.intent.alarm.set_alarm', builder.DialogAction.send('Creating Alarm'));
dialog.matches('builtin.intent.alarm.delete_alarm', builder.DialogAction.send('Deleting Alarm'));
dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't understand. I can only create & delete alarms."));
```

With this code:

------SNIP5--

```
// Add intent handlers
dialog.matches('builtin.intent.alarm.set_alarm', [
    function (session, args, next) {
        // Resolve and store any entities passed from LUIS.
        var title = builder.EntityRecognizer.findEntity(args.entities, 'builtin.alarm.title');
        var time = builder.EntityRecognizer.resolveTime(args.entities);
        var alarm = session.dialogData.alarm = {
          title: title ? title.entity : null,
          timestamp: time ? time.getTime() : null
        };

        // Prompt for title
        if (!alarm.title) {
            builder.Prompts.text(session, 'What would you like to call your alarm?');
        } else {
            next();
        }
    },
    function (session, results, next) {
        var alarm = session.dialogData.alarm;
        if (results.response) {
            alarm.title = results.response;
        }

        // Prompt for time (title will be blank if the user said cancel)
        if (alarm.title && !alarm.timestamp) {
            builder.Prompts.time(session, 'What time would you like to set the alarm 
for?');
        } else {
            next();
        }
    },
    function (session, results) {
        var alarm = session.dialogData.alarm;
        if (results.response) {
            var time = builder.EntityRecognizer.resolveTime([results.response]);
            alarm.timestamp = time ? time.getTime() : null;
        }

        // Set the alarm (if title or timestamp is blank the user said cancel)
        if (alarm.title && alarm.timestamp) {
            // Save address of who to notify and write to scheduler.
            alarm.address = session.message.address;
            alarms[alarm.title] = alarm;

            // Send confirmation to user
            var date = new Date(alarm.timestamp);
            var isAM = date.getHours() < 12;
            session.send('Creating alarm named "%s" for %d/%d/%d %d:%02d%s',
                alarm.title,
                date.getMonth() + 1, date.getDate(), date.getFullYear(),
                isAM ? date.getHours() : date.getHours() - 12, date.getMinutes(), isAM ? 
'am' : 'pm');
        } else {
            session.send('Ok... no problem.');
        }
    }]);
```

In the previous section of code, we are using a number of the techniques we have discussed. We are using **intent matching, text prompts, time prompts, and a waterfall**. Next we need to add an intent dialog for deleting an alarm.

Paste the following code **below** the last section in **luisai.js**.

------SNIP6--

```
dialog.matches('builtin.intent.alarm.delete_alarm', [
    function (session, args, next) {
        // Resolve entities passed from LUIS.
        var title;
        var entity = builder.EntityRecognizer.findEntity(args.entities, 
'builtin.alarm.title');
        if (entity) {
            // Verify its in our set of alarms.
            title = builder.EntityRecognizer.findBestMatch(alarms, 
entity.entity);
        }

        // Prompt for alarm name
        if (!title) {
            builder.Prompts.choice(session, 'Which alarm would you like 
to delete?', alarms);
        } else {
            next({ response: title });
        }
    },
    function (session, results) {
        // If response is null the user canceled the task
        if (results.response) {
            delete alarms[results.response.entity];
            session.send("Deleted the '%s' alarm.", 
results.response.entity);
        } else {
            session.send('Ok... no problem.');
        }
    }
]);
```

As you can see, it is very similar to the add alarm section with the addition of using the **choice prompt**. Now we need to add two more pieces to make it complete. We need to add back our default match section (one line of code) and a very simple implementation of an alarm. Paste the following code at the **bottom** of the **luisai.js** file.

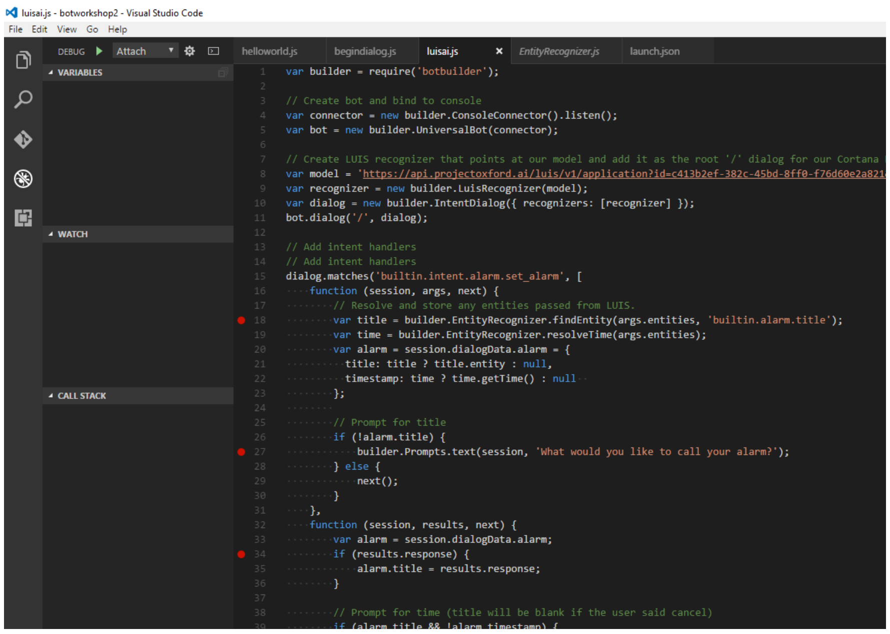
------SNIP7---

```
dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't
understand. I can only create & delete alarms."));

// Very simple alarm scheduler
var alarms = {};
setInterval(function () {
    var now = new Date().getTime();
    for (var key in alarms) {
        var alarm = alarms[key];
        if (now >= alarm.timestamp) {
            var msg = new builder.Message()
                .address(alarm.address)
                .text("Here's your '%s' alarm.", alarm.title);
            bot.send(msg);
            delete alarms[key];
        }
    }
}, 15000);
```

9. Now we can run the code. Go to your command prompt and type in the following.

```
~$ node --debug luisai.js
```

10. Next go to Visual Studio Code and place some breakpoints in your code so you can step through and inspect the entities, as they are being set.



```
var builder = require('botbuilder');

// Create bot and bind to console
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

// Create LUIS recognizer that points at our model and add it as the root '/' dialog for our Cortana
var model = 'https://api.projectoxford.ai/luis/v1/application?id=c413b2ef-382c-45bd-8ff0-f76d60e2a821
var recognizer = new builder.LuisRecognizer(model);
var dialog = new builder.IntentDialog({ recognizers: [recognizer] });
bot.dialog('/', dialog);

// Add intent handlers
// Add intent handlers
dialog.matches('builtin.intent.alarm.set_alarm', [
    function (session, args, next) {
        // Resolve and store any entities passed from LUIS.
        var title = builder.EntityRecognizer.findEntity(args.entities, 'builtin.alarm.title');
        var time = builder.EntityRecognizer.resolveTime(args.entities);
        var alarm = session.dialogData.alarm = {
          title: title ? title.entity : null,
          timestamp: time ? time.getTime() : null
        };

        // Prompt for title
        if (!alarm.title) {
            builder.Prompts.text(session, 'What would you like to call your alarm?');
        } else {
            next();
        }
    },
    function (session, results, next) {
        var alarm = session.dialogData.alarm;
        if (results.response) {
            alarm.title = results.response;
        }

        // Prompt for time (title will be blank if the user said cancel)
        if (alarm.title && !alarm.timestamp) {
```

11. Step through the code as we had before to watch as things are running.

Exercise 4: Connecting to Skype and Webchat

In this exercise we are going to connect your bot to a Skype and a Webchat channel. In doing so, we ARE NOT going to be hosting our bot in the cloud but running them from our laptops. We are doing this for 3 reasons:

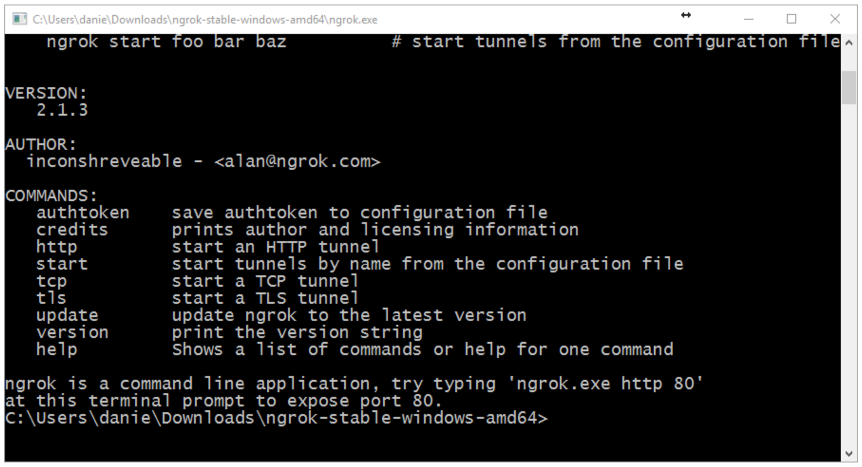
1) It takes out some of the complexity of the workshop by not having to troubleshoot uploading and running a node app in your cloud of choice.
2) We don't have to walk 30 + people through setting up a cloud account.
3) It is valuable to use this technique to troubleshoot your bot prior to production so it is a valuable skill.

SPECIAL NOTE: An alternative for testing is to use the open source / cross platform emulator to test. You can find that here : https://docs.botframework.com/en-us/tools/bot-framework-emulator/

The first thing we need to do is to run NGrok. You should have downloaded NGrok from https://ngrok.com/ as it states on the website, NGrok allows you to **create "Secure tunnels to localhost"**. This means we can make the bot framework think that our bot is hosted in the cloud by having an internet accessible URL for the bot hosted on our laptop.

Detailed Steps

1. Download and unzip Ngrok on your computer
2. Once unzipped double-click on the **ngrok.exe** file. This will open its own command window.



```
C:\Users\danie\Downloads\ngrok-stable-windows-amd64\ngrok.exe
    ngrok start foo bar baz          # start tunnels from the configuration file


VERSION:
   2.1.3

AUTHOR:
  inconshreveable - <alan@ngrok.com>

COMMANDS:
   authtoken    save authtoken to configuration file
   credits      prints author and licensing information
   http         start an HTTP tunnel
   start        start tunnels by name from the configuration file
   tcp          start a TCP tunnel
   tls          start a TLS tunnel
   update       update ngrok to the latest version
   version      print the version string
   help         Shows a list of commands or help for one command

ngrok is a command line application, try typing 'ngrok.exe http 80'
at this terminal prompt to expose port 80.
C:\Users\danie\Downloads\ngrok-stable-windows-amd64>
```

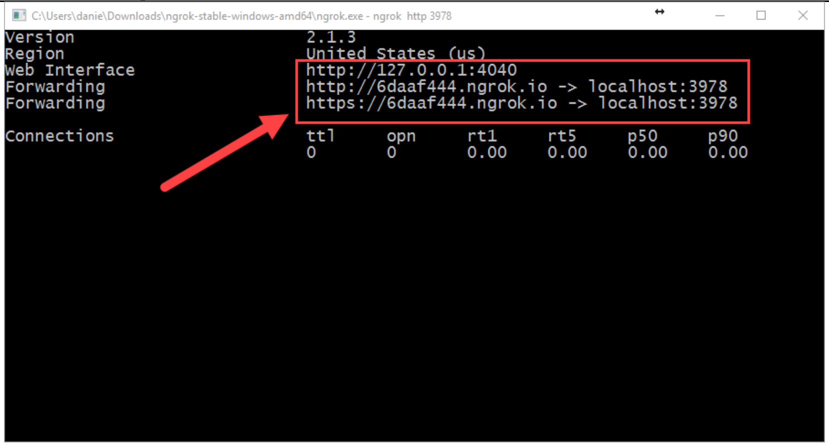
3. At the prompt in this window type the following.

```
~$ ngrok http 3978
```

NOTE: On a MAC you will need to type ./ngrok http 3978

You should see the following in your command window.

Detailed Steps



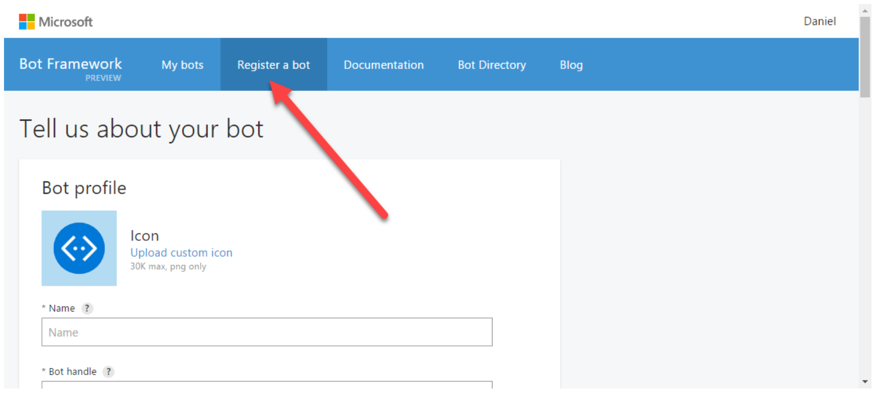
```
C:\Users\danie\Downloads\ngrok-stable-windows-amd64\ngrok.exe - ngrok  http 3978
Version                       2.1.3
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://6daaf444.ngrok.io -> localhost:3978
Forwarding                    https://6daaf444.ngrok.io -> localhost:3978

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

We care about two things in this window.
1. The forwarding URLs (both http and https) that will be our external address for our bot to reach our local machine.
2. The Web interface address. We will use this to track traffic coming into this port for our bot.

Leave this running. We will need this for the rest of this walkthrough.

4. Next we need to set up our bot on the BotFramework page. Open a browser and go to http://BotFramework.com . You should have already set up an account at the beginning of this HOL.

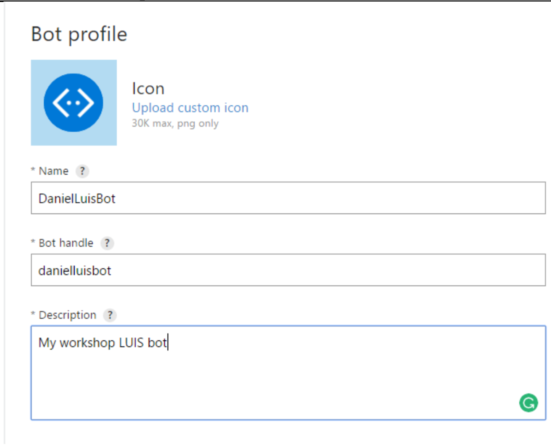
5. If you are not already signed in, sign in with the MS account you created earlier. When you are signed in click on the Register a bot in the menu.



6. Fill out the bot registration form. We will go through it section by section and touch on the important fields.

Name: <Name of your bot> Self-explanatory
Bot Handle: <your handle here> this will be used in the C# SDK when referencing your bot (not in Node.js SDK)
Description: Self-explanatory

Detailed Steps



Bot profile

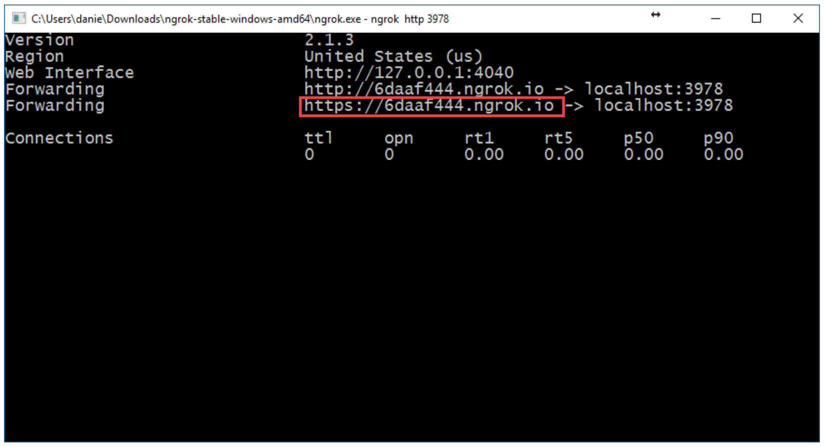
Icon

* Name: DanielLuisBot

* Bot handle: danielluisbot

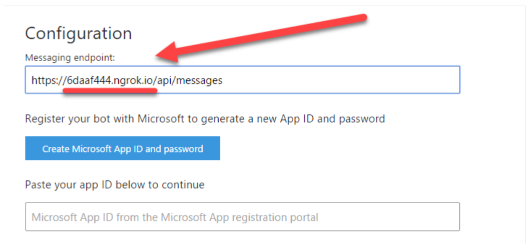
* Description: My workshop LUIS bot

7. This next section is only two fields but many steps. The first box is the Messaging endpoint. If you were hosting you bot in the cloud, then this would be the address of the site that is hosting it. Something like http://DanielSpeakerBot.com/api.messages but since we are hosting it locally we need to use the address that Ngrok gave us when we used it.



```
C:\Users\danie\Downloads\ngrok-stable-windows-amd64\ngrok.exe - ngrok  http 3978
Version                       2.1.3
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://6daaf444.ngrok.io -> localhost:3978
Forwarding                    https://6daaf444.ngrok.io -> localhost:3978

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

append the address from Ngrok with /api/messages



Configuration

Messaging endpoint: https://6daaf444.ngrok.io/api/messages

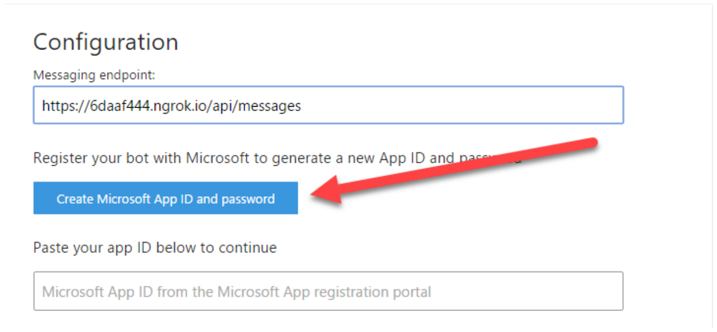
Register your bot with Microsoft to generate a new App ID and password

Paste your app ID below to continue

Microsoft App ID from the Microsoft App registration portal

Next you need to create an AppID and password for your bot. click on the "**Create Microsoft App ID and password**" button.

Detailed Steps



Configuration

Messaging endpoint:

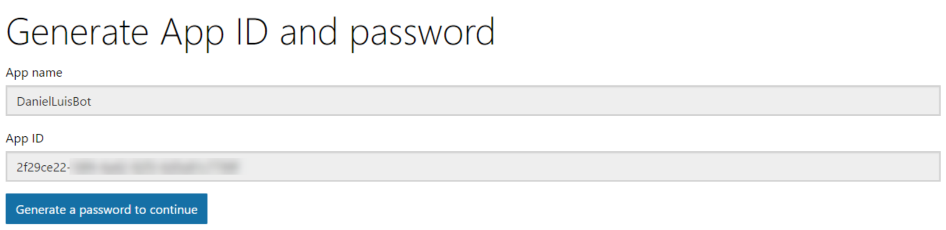
https://6daaf444.ngrok.io/api/messages

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

Paste your app ID below to continue

Microsoft App ID from the Microsoft App registration portal

When you do this a new page will pop up and give you an **App ID**.



Generate App ID and password

App name

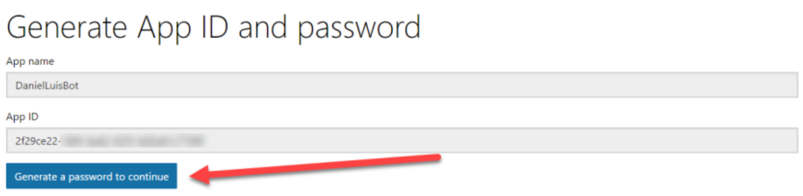
DanielLuisBot

App ID

2f29ce22-[redacted]

SAVE THIS APP ID SOMEWHERE. WE WILL NEED IT LATER.

Next click on the **Generate a password to continue** button.



Generate App ID and password

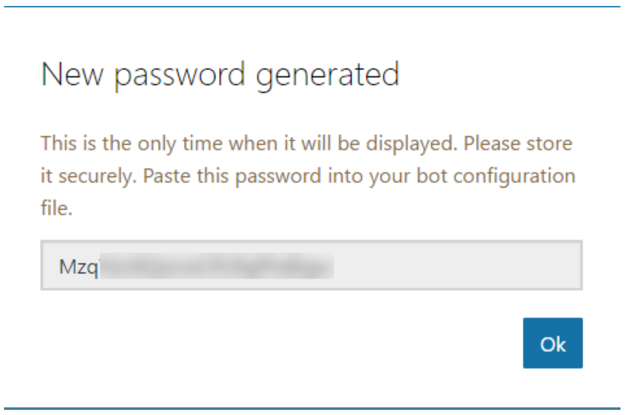
App name

DanielLuisBot

App ID

2f29ce22-[redacted]

This will pop up a modal dialog with your password.



New password generated

This is the only time when it will be displayed. Please store it securely. Paste this password into your bot configuration file.

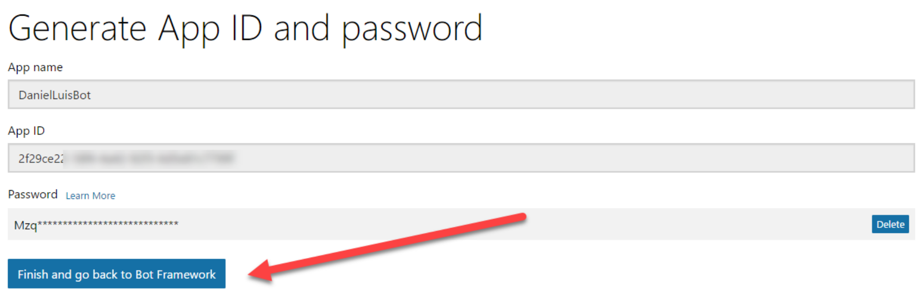
Mzq[redacted]

This is the only time it will be shown.
SAVE THIS PASSWORD ID SOMEWHERE. WE WILL NEED IT LATER

Detailed Steps

Click **ok** to continue.

Next click on the **Finish and go back to Bot Framework** button to continue.


Generate App ID and password

App name

DanielLuisBot

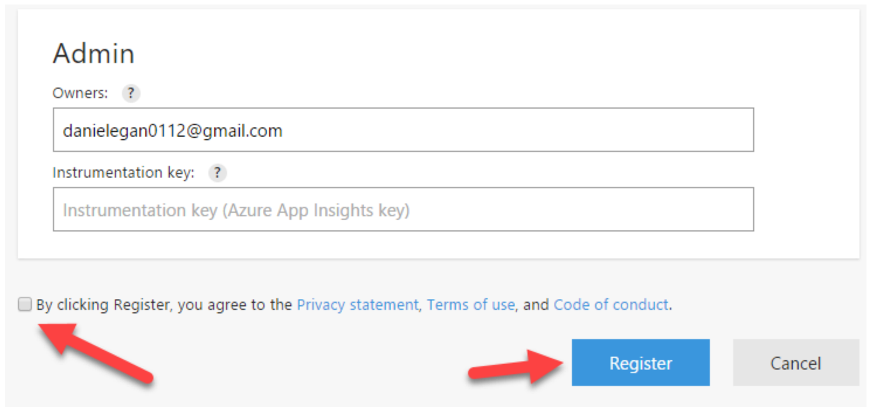
App ID

2f29ce2[redacted]

Password Learn More

Mzq****************************

8. In the final section, we do not need to add anything. (although in the future, setting up and using an App insights key will give you a bunch of great reporting)


Admin

Owners: ?

danielegan0112@gmail.com

Instrumentation key: ?

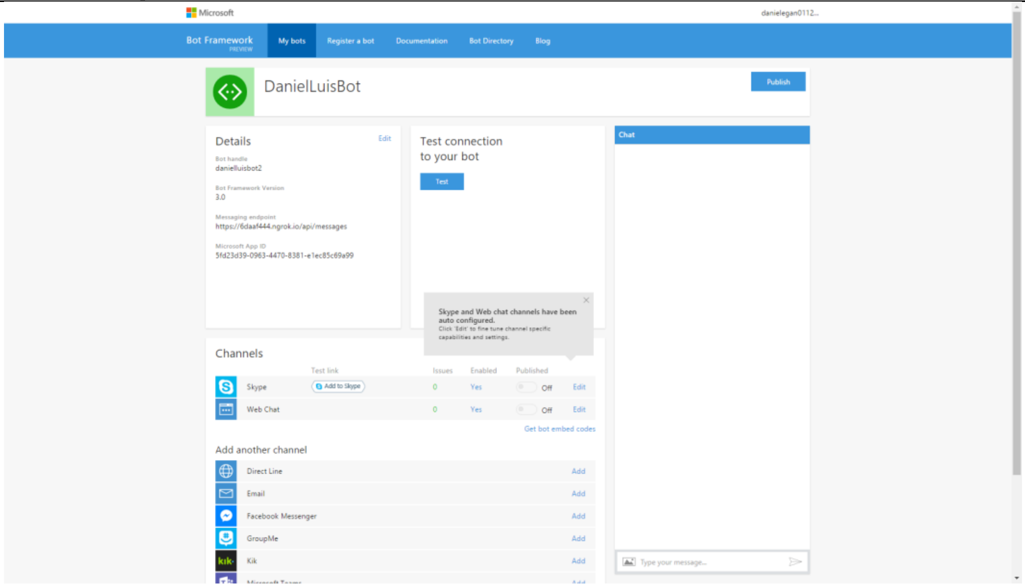
Instrumentation key (Azure App Insights key)

By clicking Register, you agree to the Privacy statement, Terms of use, and Code of conduct.

9. **Agree** to the terms and click **Register** to create your bot.

10. Leave this page up and running. We will need to come back here after we modify our bot to link it to the bot framework.

Detailed Steps



DanielLuisBot

Details

Bot handle
danielluisbot2

Bot Framework Version
3.0

Messaging endpoint
https://6daaf444.ngrok.io/api/messages

Microsoft App ID
5fd23d39-0963-4470-8381-e1ec85c69a99

Skype and Web chat channels have been auto configured.
Click 'Edit' to fine tune channel specific capabilities and settings.

Channels

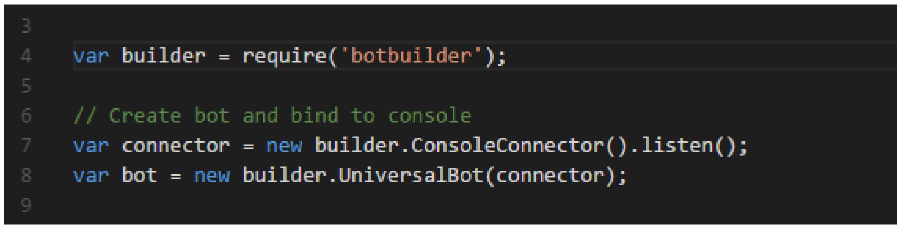
	Test link	Issues	Enabled	Published	
Skype	Add to Skype	0	Yes	Off	Edit
Web Chat		0	Yes	Off	Edit

Get bot embed codes

Add another channel

Direct Line	Add
Email	Add
Facebook Messenger	Add
GroupMe	Add
Kik	Add

11. Bring up Visual Studio Code and open up the **luisai.js** file. We will be modifying the following section of that file.



```

var builder = require('botbuilder');

// Create bot and bind to console
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);

```

Up until now, we have been using the ConsoleConnector. Now we are going to be using the ChatConnecter. But first we need to add a node module that will help us with serving this app. It is called Restify.

12. Open up your console (Terminal, Powershell, etc..). Make sure you are in the botworkshop folder (or whatever you called it) and type the following.

```
~$ npm install restify --save
```

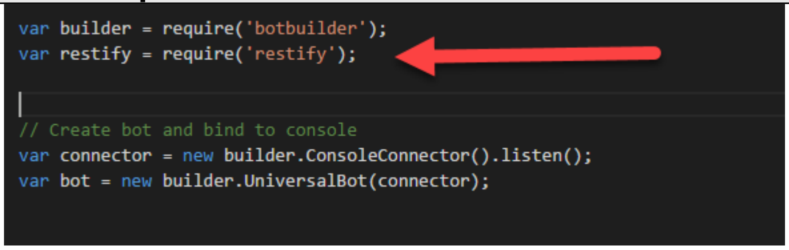
Restify is a node module that helps make rest calls easier.

13. Once that is done, open up the luisai.js file and add the following code underneath the **var builder = require('botbuilder');** code.

```
var restify = require('restify');
```

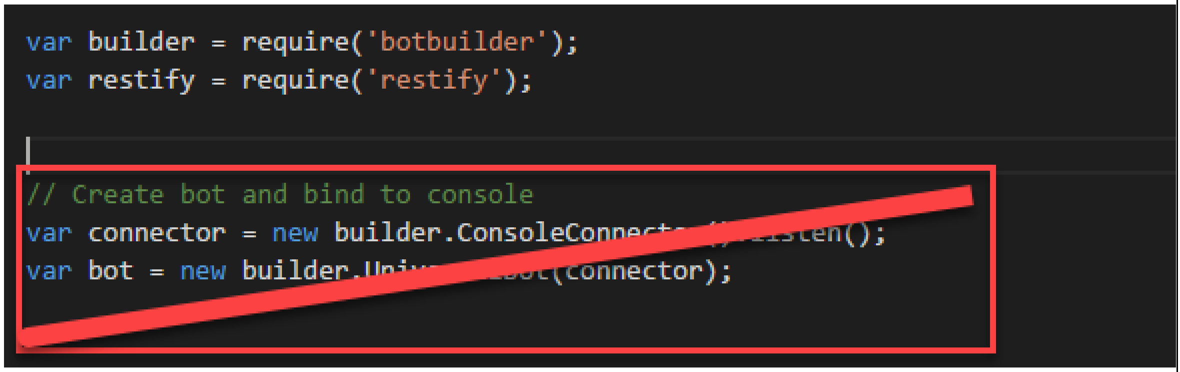
The code should look similar to this

Detailed Steps



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


// Create bot and bind to console
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector);
```

14. Since we will be working with the new connector. Delete the connector and the bot lines so we can recreate them.



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


~~// Create bot and bind to console~~
~~var connector = new builder.ConsoleConnector().listen();~~
~~var bot = new builder.UniversalBot(connector);~~
```

With that gone, we need to set up the restify server. To do so, add the following lines under the restify require statement.

------SNIP8---

```
// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function ()
{
  console.log('%s listening to %s', server.name, server.url);
});
```

We are setting up a server that will look for an environment variable called PORT, if it does not find one, it will start on port **3978** (which is why we used that port for NGrok).

Next we want to create the new ChatConnector (instead of the ConsoleConector).
Add the following code directly under the last code you pasted.

Detailed Steps

------SNIP9--

```
// Create chat bot
var connector = new builder.ChatConnector({
    appId: process.env.MICROSOFT_APP_ID,
    appPassword: process.env.MICROSOFT_APP_PASSWORD
});
```

Notice that the connector requires and **appId** and **appPassword**. These are what we saved when we create our bots on BotFramework.com.

Now we need to new up our bot and pass in our connector like we did before and set up where the post route is (/api/messages).

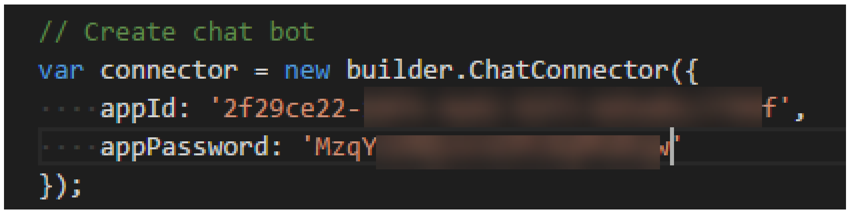
Add this code below the last code you pasted.

------SNIP10--

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

There is one last step. Adding our **appId** and **appPassword**. To protect them, you should **ALWAYS** put them in **Environment Variables** (or other safe place) . If you plan to put this in production **OR** plan to save this code in github or another repository, **DO NOT** do what we are going to do right now.

Modify your connector to add your **appId** and **appPassword**.

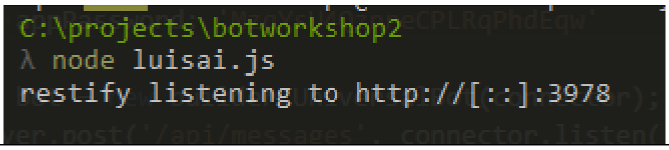


```
// Create chat bot
var connector = new builder.ChatConnector({
    appId: '2f29ce22-[redacted]f',
    appPassword: 'MzqY[redacted]w'
});
```

15. Now lets run out bot. Go to your console and run the following command.

```
~$ node luisai.js
```

You should see the restify server running on the port we specified.

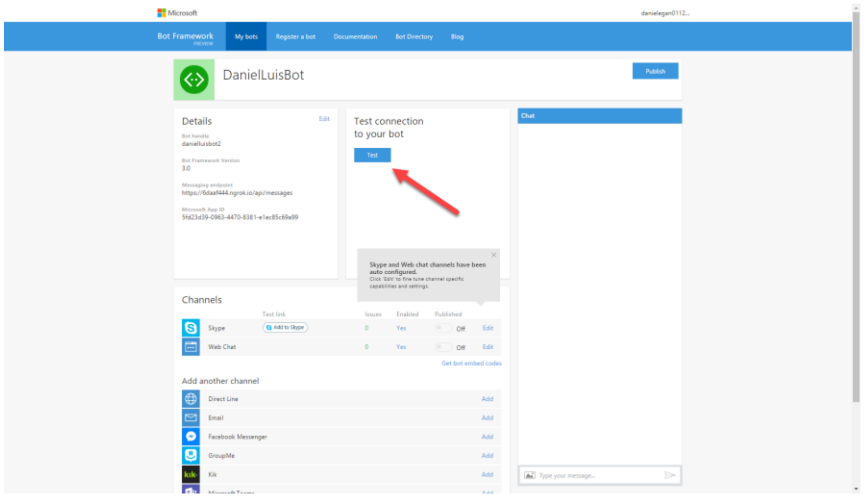


```
C:\projects\botworkshop2
λ node luisai.js
restify listening to http://[::]:3978
```

Detailed Steps

16. We can test the connection by going to your bot on **BotFramework.com**

If everything works out fine when clicking the test button, you will get back an accepted message.



DanielLuisBot

Details

Bot handle
danielluisbot2

Bot Framework Version
3.0

Messaging endpoint
https://6daaf444.ngrok.io/api/messages

Microsoft App ID
5fd23d39-0963-4470-8381-e1ec85c69a99

Test connection to your bot

Skype and Web chat channels have been auto configured.
Click 'Edit' to fine tune channel specific capabilities and settings.

Channels

	Test link	Issues	Enabled	Published	
Skype	Add to Skype	0	Yes	Off	Edit
Web Chat		0	Yes	Off	Edit

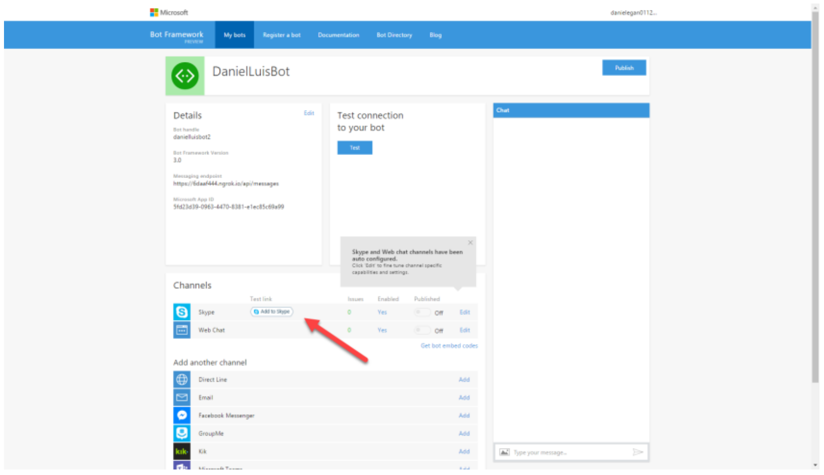
Get bot embed codes

Add another channel

Direct Line	Add
Email	Add
Facebook Messenger	Add
GroupMe	Add
Kik	Add

17. Now we can test Skype (you must have Skype installed).

In your portal, click on the Add to Skype Button



DanielLuisBot

Details

Bot handle
danielluisbot2

Bot Framework Version
3.0

Messaging endpoint
https://6daaf444.ngrok.io/api/messages

Microsoft App ID
5fd23d39-0963-4470-8381-e1ec85c69a99

Test connection to your bot

Skype and Web chat channels have been auto configured.
Click 'Edit' to fine tune channel specific capabilities and settings.

Channels

	Test link	Issues	Enabled	Published	
Skype	Add to Skype	0	Yes	Off	Edit
Web Chat		0	Yes	Off	Edit

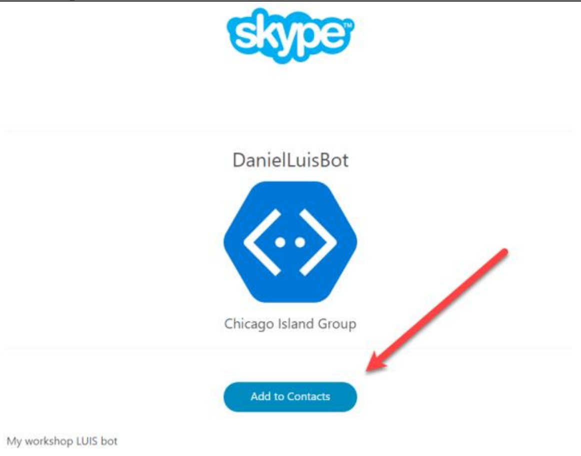
Get bot embed codes

Add another channel

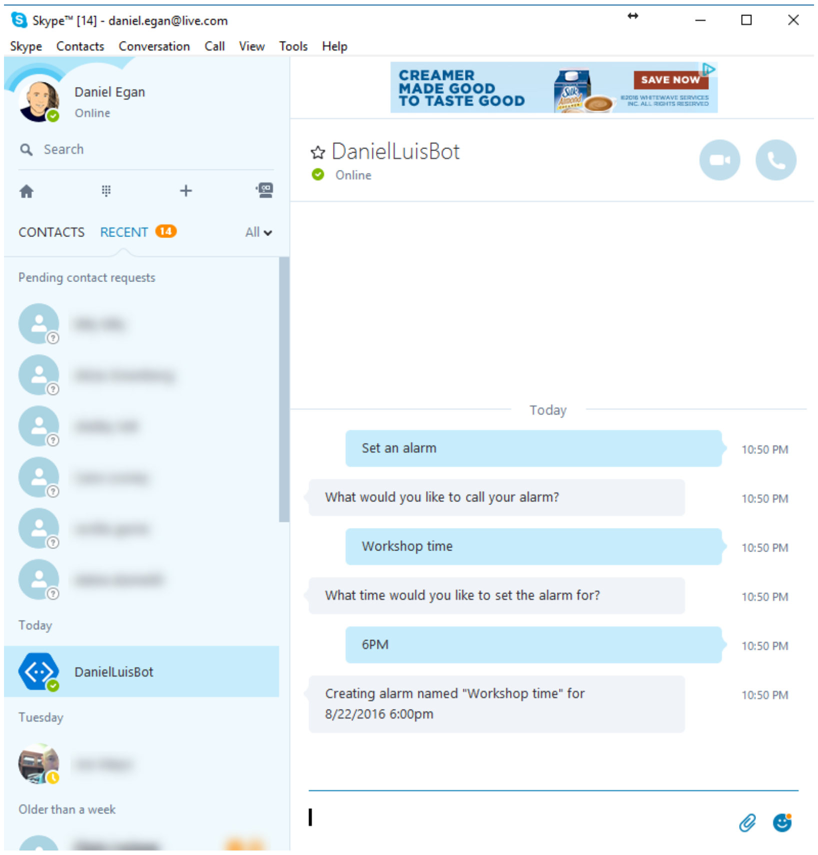
Direct Line	Add
Email	Add
Facebook Messenger	Add
GroupMe	Add
Kik	Add

When the windows pops up, click on add you Contacts.

Detailed Steps



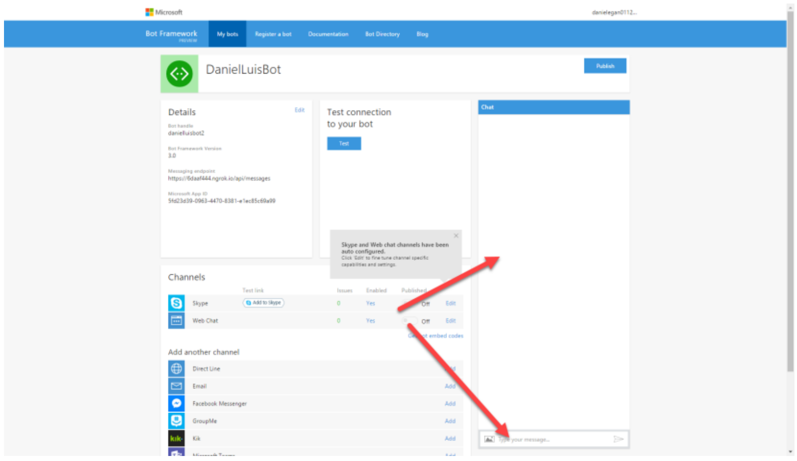
Once it is added to your contacts, you can chat away.



Set an alarm 10:50 PM
What would you like to call your alarm? 10:50 PM
Workshop time 10:50 PM
What time would you like to set the alarm for? 10:50 PM
6PM 10:50 PM
Creating alarm named "Workshop time" for 8/22/2016 6:00pm 10:50 PM

If you don't have Skype installed you can test it out by using a webchat window. This is embedded in your bot registration page.

Detailed Steps



If you want a local web control to test with you can also add one to your project. The first thing we need to do is to create a page to host the webchat control.

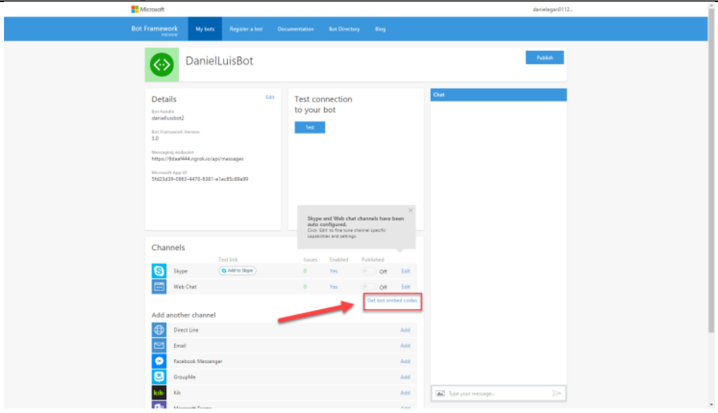
18. Open up Visual Studio Code and add a file called index.html

------SNIP11---

```
<!doctype html>
<html>
   <head>
      <title>MyAppID</title>
   </head>
   <body>



   </body>

</html>
```

19. Next, go to your bot on BotFramework.com and click on the Get bot embedded codes

Detailed Steps



Bot handle
daniellusbot2

Bot Framework Version
3.0

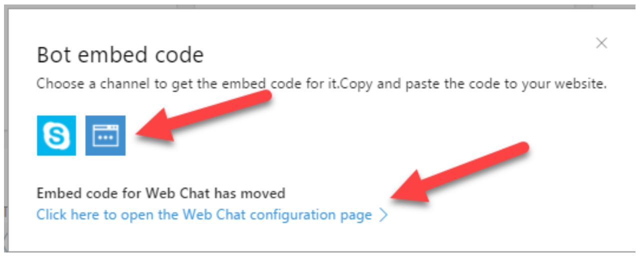
Messaging endpoint
https://8daaf444.ngrok.io/api/messages

Microsoft App ID
5fd23d39-0963-4470-8381-e1ec85c69a99

	Test link	Issues	Enabled	Published	
Skype	Add to Skype	0	Yes	Off	Edit
Web Chat		0	Yes	Off	Edit

Direct Line	Add
Email	Add
Facebook Messenger	Add
GroupMe	Add
Kik	Add

Click on the Web Chat Icon and then follow the link.



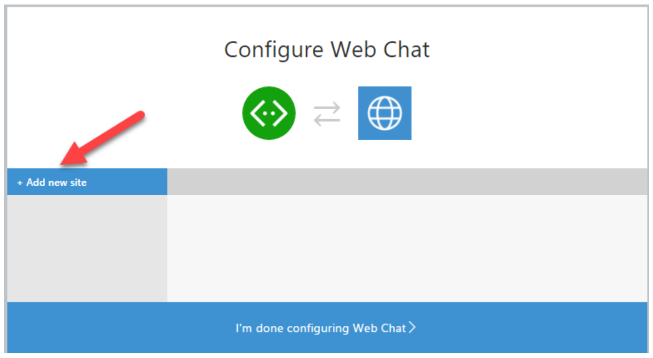
Bot embed code

Choose a channel to get the embed code for it.Copy and paste the code to your website.

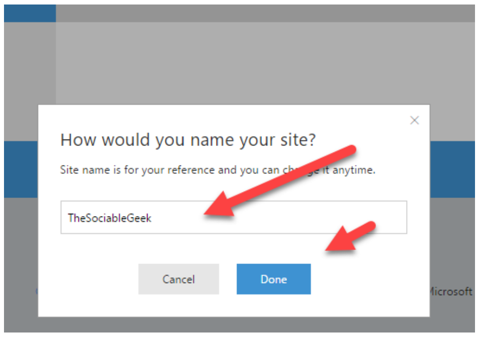
Embed code for Web Chat has moved

Click here to open the Web Chat configuration page >

Click on the Add new site link



Enter the name of the site it is going on. This is just for your purposes so that you can customize per site.



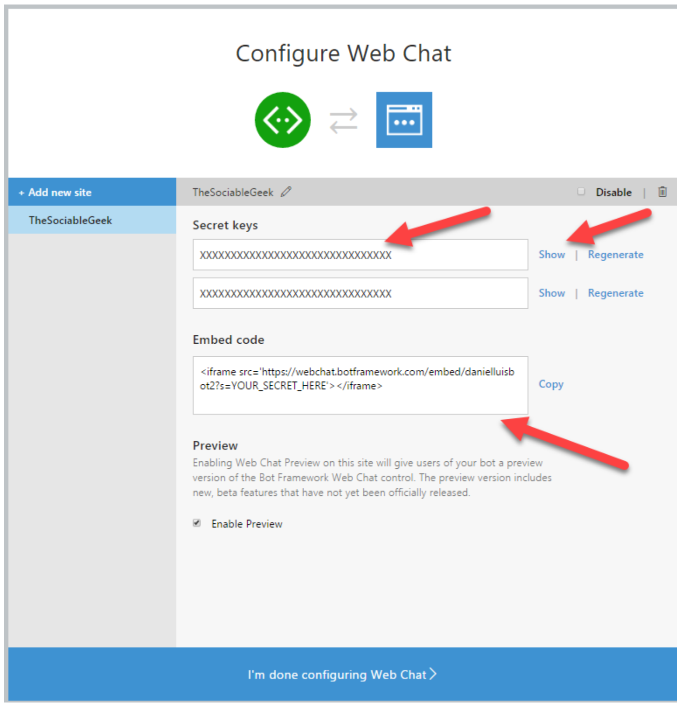
How would you name your site?

Site name is for your reference and you can change it anytime.

TheSociableGeek

Detailed Steps

This will generate both your embed code (an IFrame) and your secret keys. Click on Show on one of them so you can copy it and save it for use in the next step.



Configure Web Chat

+ Add new site
TheSociableGeek

TheSociableGeek — Disable

Secret keys
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Show | Regenerate
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Show | Regenerate

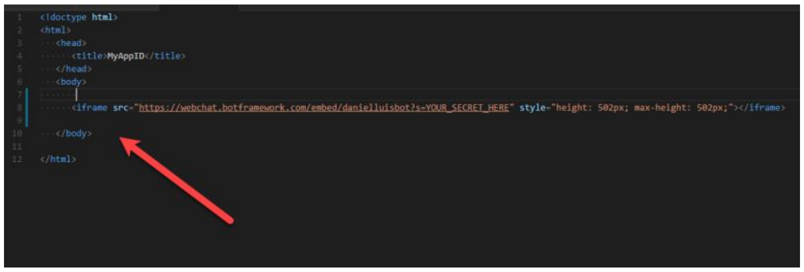
Embed code
<iframe src='https://webchat.botframework.com/embed/danielluisbot2?s=YOUR_SECRET_HERE'></iframe> Copy

Preview
Enabling Web Chat Preview on this site will give users of your bot a preview version of the Bot Framework Web Chat control. The preview version includes new, beta features that have not yet been officially released.

☑ Enable Preview

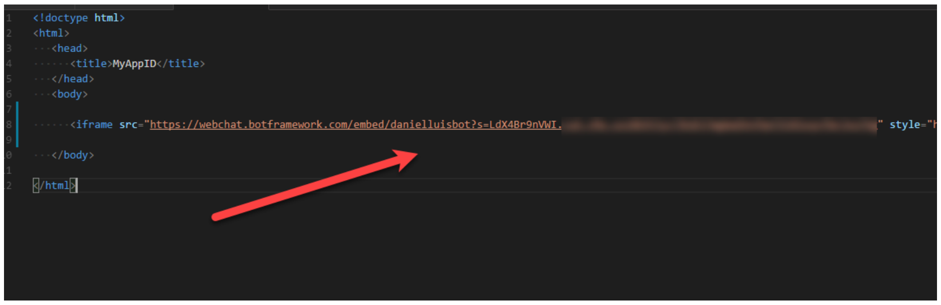
I'm done configuring Web Chat >

Copy the embed code into the body section on index.html you created.



```
<!doctype html>
<html>
   <head>
      <title>MyAppID</title>
   </head>
   <body>
      
      <iframe src="https://webchat.botframework.com/embed/danielluisbot?s=YOUR_SECRET_HERE" style="height: 502px; max-height: 502px;"></iframe>

   </body>

</html>
```

Replace the YOUR_SECRET_KEY section with the key you saved.



```
<!doctype html>
<html>
   <head>
      <title>MyAppID</title>
   </head>
   <body>

      <iframe src="https://webchat.botframework.com/embed/danielluisbot?s=LdX4Br9nVWI.[redacted]" style="

   </body>

</html>
```

Detailed Steps

Finally, to be able to access this page, you need to add a **server.get** to your **luisai.js** file.

Underneath the **server.post('/api/messages', connector.listen());** line, add the following code.

------SNIP12--

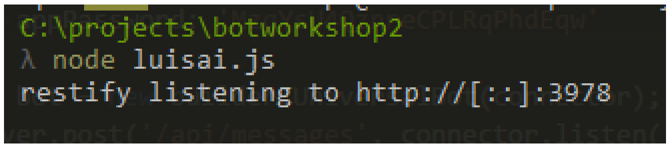
```
server.get('/', restify.serveStatic({
 directory: __dirname,
 default: '/index.html'
}));
```

This will route incoming get requests (in the browser) to our **index.html** page.

To test it out, go to your console and run the following command.

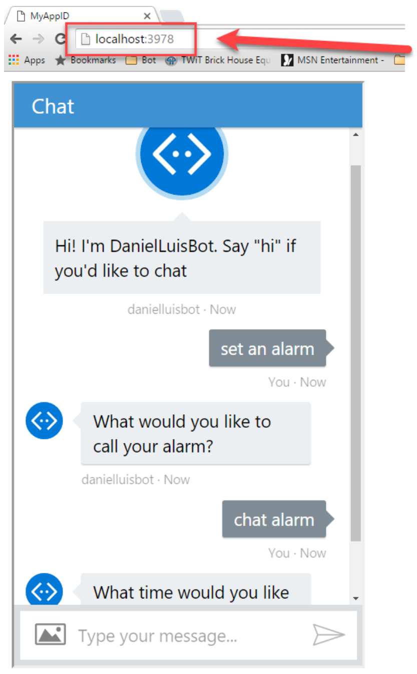
```
~$ node luisai.js
```

You should see the restify server running on the port we specified.



```
C:\projects\botworkshop2
λ node luisai.js
restify listening to http://[::]:3978
```

Next browse to http://localhost:3978 in a browser



Hi! I'm DanielLuisBot. Say "hi" if you'd like to chat
danielluisbot · Now
set an alarm
You · Now
What would you like to call your alarm?
danielluisbot · Now
chat alarm
You · Now
What time would you like
Type your message...

Now you can chat away.

Detailed Steps

Additional Resources

Copyright