# How to Create AI Bot with LUIS

## Introduction:

Every rated institutes or organization expect to serve instant customer greetings and basic interaction. As a virtual help assistant, a Chatbot might help on the spot of instant service. This article demonstrate how to create a simple Bot using Microsoft Bot Framework and LUIS which is stands for machine learning-based service to build natural language into apps, bots, and IoT devices.

## Creating Application with LUIS from scratch:

Currently Microsoft release a several packages that helps to build bots with various features. If you need more details please visit their [Bot Framework SDK](https://github.com/Microsoft/botbuilder-dotnet/) release notes and documentation on GitHub.

### In this article, you learn how to:

* Create a basic Chatbot
* Integrate Bot with LUIS
* Run and interact with it

### Required Tools and Knowledge:

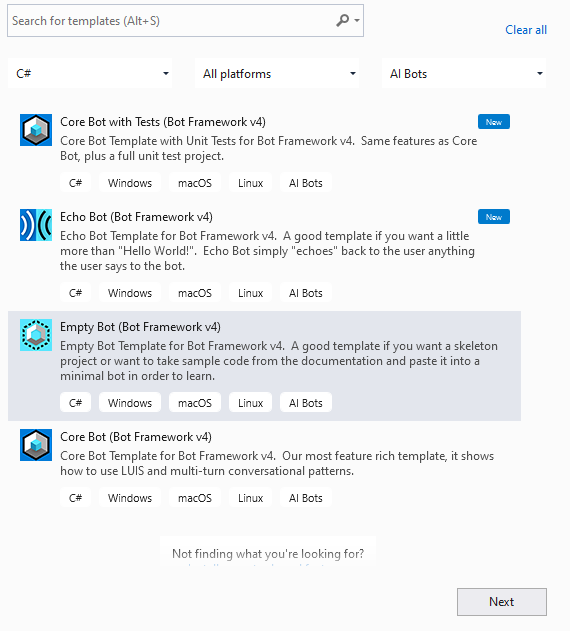
* Visual Studio 2017 or later
* Bot Framework Emulator
* LUISGen in .NET Cognitive Model Generator for LUIS

### Step1:

First go to Visual Studio marketplace and download: [Bot Framework v4 SDK Templates for Visual Studio](https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4) and install the templates on your visual Studio.

### Step2:

Open your Visual Studio and choose create a new project, select C# as your language, all platforms and AI Bot as your project type. [Or search for Bot and then] select Empty Bot and hit next.



Give your project name and complete the dialog box. It initiate a project of ASP .Net Core application with minimal codes and dependency with Bot Builder Integration.

### Step3:

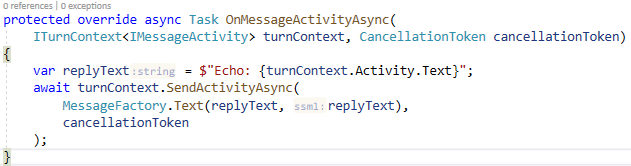
Build the project and run it for verifying this all works fine. Generally all bot application runs on port 3978 on the local machine. So, go to your Bot Framework Emulator and open bot from: <http://localhost:3978/api/messages>

If everything is ready, then bot will response with Hello World!

Reply with something, now bot have no longer to response. But you still get post 200 status on activity logger. That means the request was accepted.

### Step4:

To get the response, back to Visual Studio and open your C sharp bot file. On empty project, generally it stands on project folder and named with your [ProjectName].cs, open it and find your class named EmptyBot who inherits bot activity Handler, ActivityHandler. Here you find the overridden method of OnMembersAddedAsync who responsible for your Welcome message on first request. On this class override another method from ActivityHandler named OnMessageActivityAsync with samples goes here:



This message activity turns your Emptybot into Echobot. Rebuild it and run. Hit something from emulator and Enjoy.

### Step5:

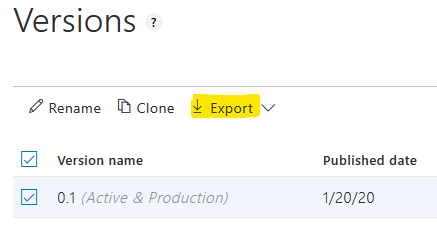
Now we have to initiate LUIS. To Start with LUIS, please go to [LUIS.ai](https://www.luis.ai), Sign Up with your Microsoft account. If you have any Azure subscription, use it, otherwise start for 3 month free trials. Go to [LUIS Application menu](https://www.luis.ai/applications) and create new app or import your app in JSON Configuration format. Create your entities, normalize with their synonyms. Then model your intents and configure them with Utterances. Within your expected training data set, train your App and publish it.

Get your Application Id from Application **Manage** Tab **->** **Settings ->** **Application Information**

Get your API Key from Application **Manage** Tab **->** **Settings** **-> Azure Resources -> Primary key**

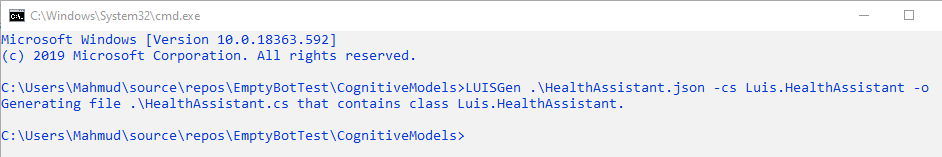
Get your Application Host Name from Application **Manage** Tab **->** **Settings** **-> Azure Resources -> Endpoint Url {**only between **https://** and **/LUIS**, *i.e.: westus.api.cognitive.microsoft.com***}**

To retrieve your application configuration again go Application **Manage** Tab **->** **Settings** **-> Versions** and check your required version of your app, then hit export.



Copy the response data and Store it in a new JSON Configuration File of your application. Hence the main application stands for ASP .NET Core, I recommend about storing this file in sub directory [named CognitiveModels] of main project. Run DOT NET CLI on this directory and hit

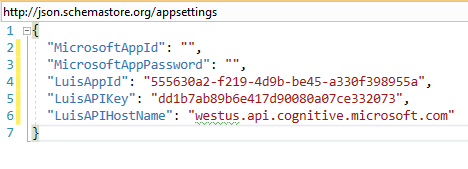
LUISGen .\ [nameofapp].json -cs LUIS.[namewanted\_as\_cs] –o



For further documentation, please review [LUISGen Documentation](https://github.com/microsoft/botbuilder-tools/blob/master/packages/LUISGen/src/npm/readme.md). Now the cognitive model is ready for deploy.

### Step6:

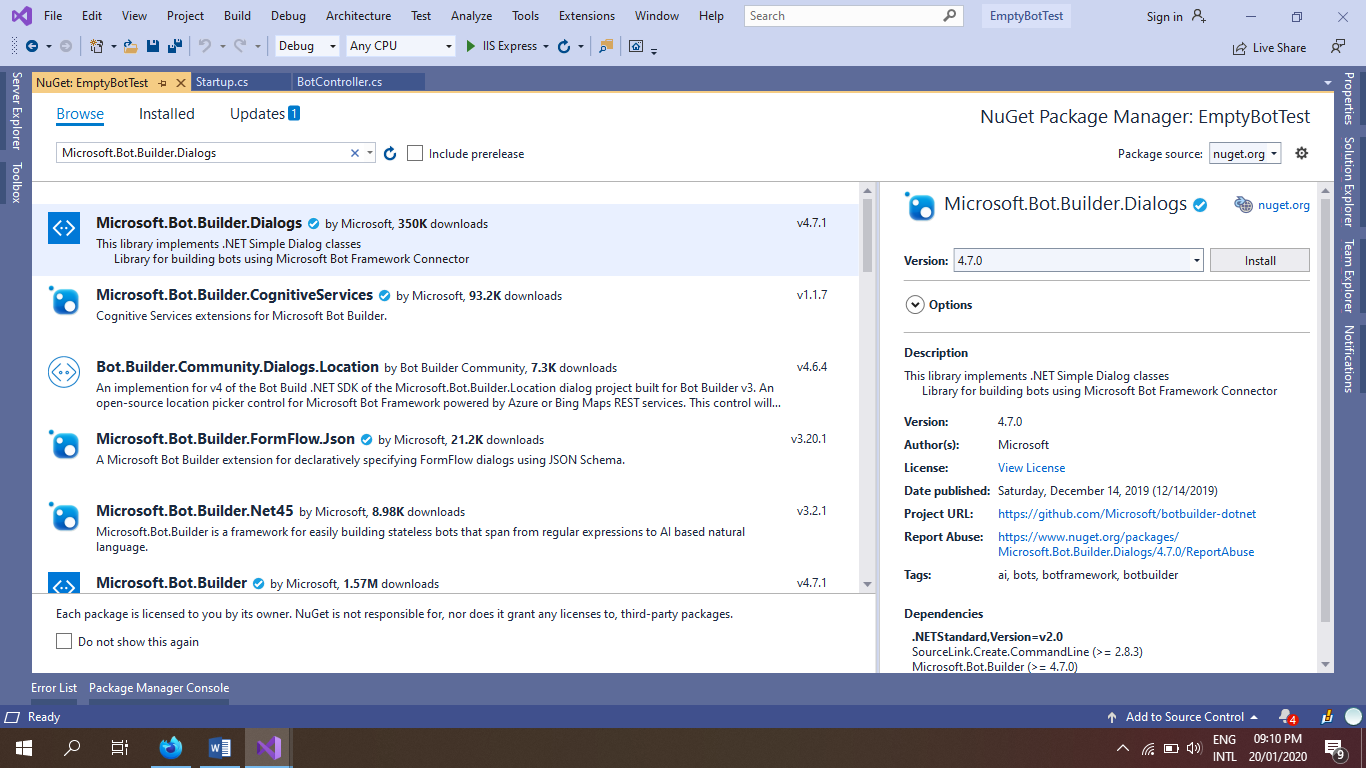
Go to your appsettings.json file and include configuration for your LUIS app: your Application Id, API Key, and Application Host Name that you retrieved in previous step.



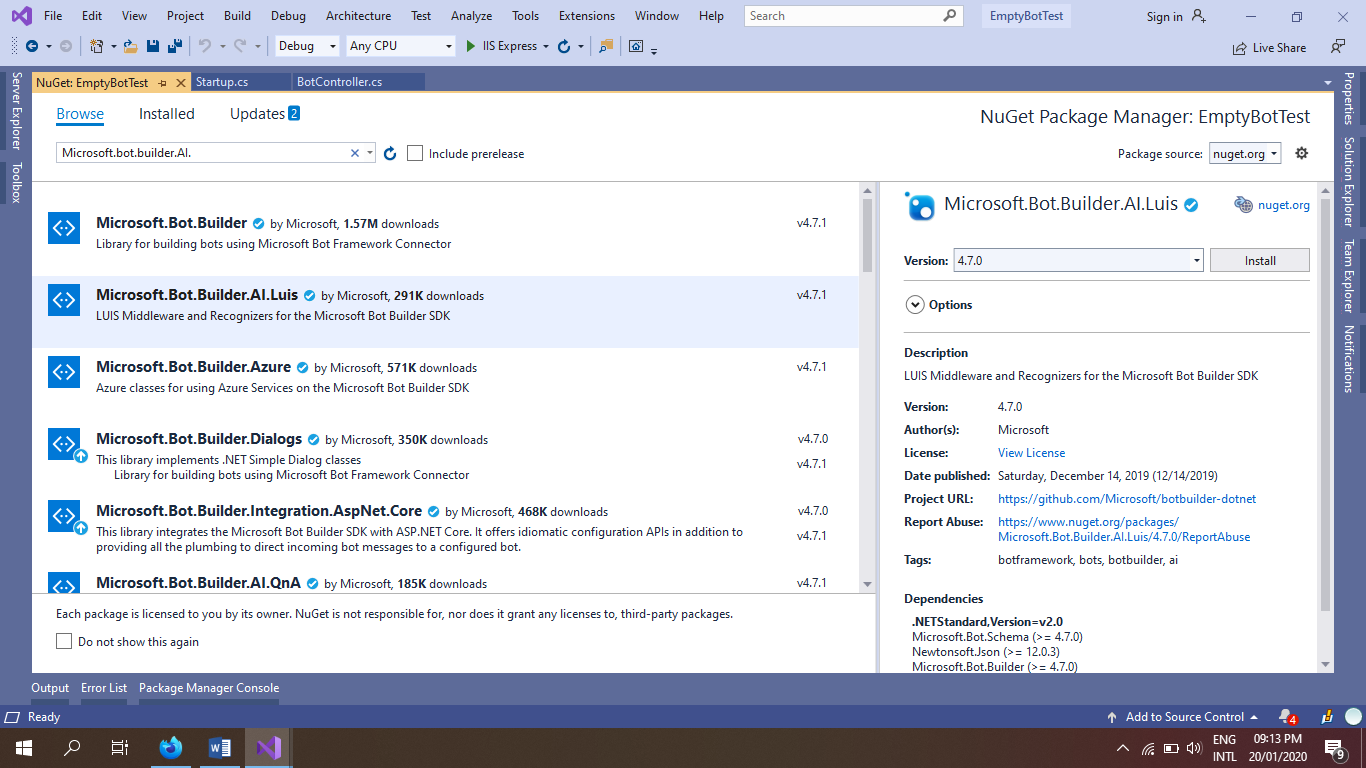
### Step7:

From the NuGet Package Manager of your project, install these packages:

* **Microsoft.Bot.Builder.Dialogs** for using the application as a dialog bot

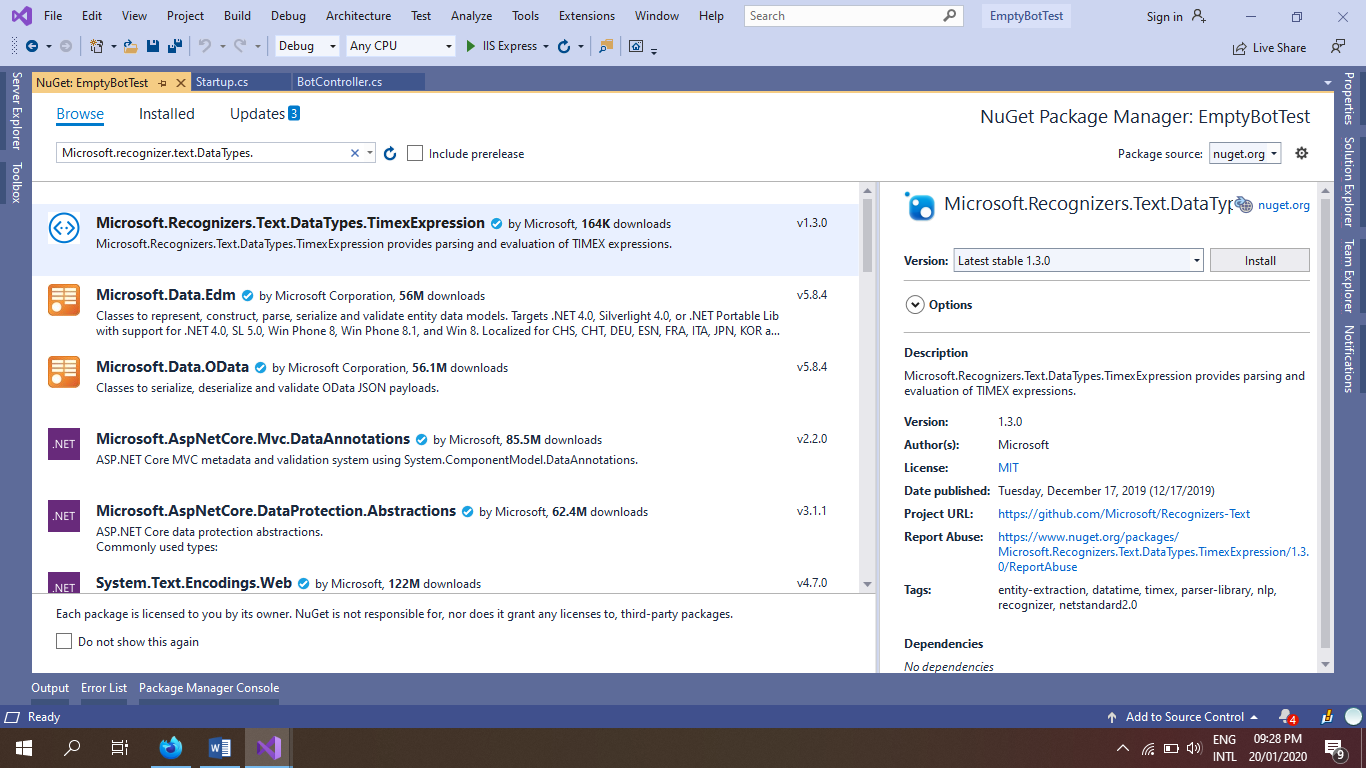


* **Microsoft.Bot.Builder.AI.LUIS** which have dependency on LUIS Runtime.



* Hence we create project from Bot Framework Sdk template, so it already installed the Integration for ASP .Net Core.
* If you need any of Data Type Recognizer within cognitive model, Please install them.

For example: I have installed **Microsoft.Recognizers.Text.DataTypes.TimexExpression**



Now verify your package references for confirming all required package are installed successfully.

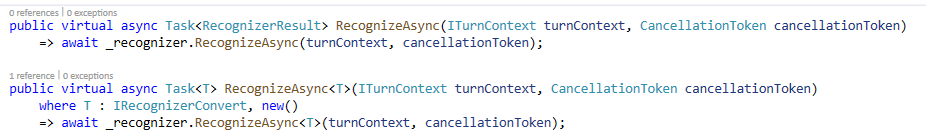
### Step8:

Design a recognizer for your LUIS model that inherits **Microsoft.Bot.Builder.IRecognizer**, Here you have to implements two missing members

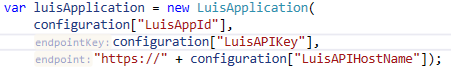
Task<T> RecognizeAsync<T>() and

Task<RecognizerResult> RecognizeAsync()

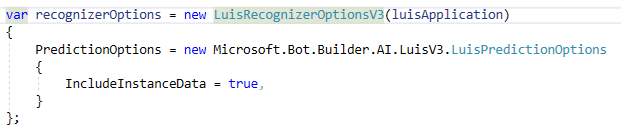
Implement these RecognizeAsync<T>() and RecognizeAsync() from a LUISRecognizer object asynchronously.



Before using the LUISRecognizer object, create a constructor for recognizer with parameter of IConfiguration. Where you have to check LUIS configurations what you configure in appsettings.json. Load the configurations and initiate a LUIS application from it.



By this LUIS application, create a LUIS recognizer option and set prediction options with instance. Currently V3 prefer to V4.

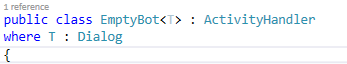


Initiate the LUISRecognizer object with this recognizer option.

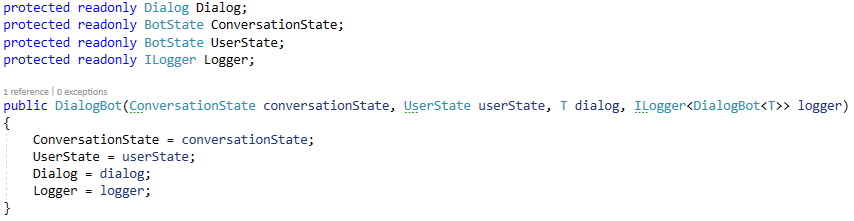
To confirm about successful initialization of LUISRecognizer object, refer a public Boolean value or a method for check, which can be used in another instances.

### Step9:

Go to your C sharp bot file, rewrite your class declaration to add feature of Dialog bot.



Now you can use Dialog component to your bot. Declare following objects: a Dialog, two bot state, one for conversation state, and another for user state, and an ILogger. Initiate them with your bot constructor.

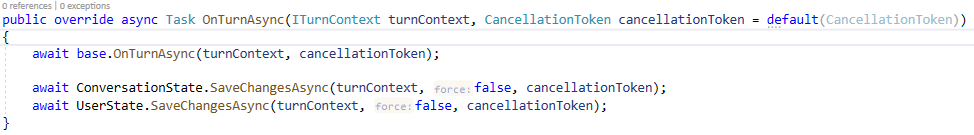


From your Dialog object, call an Asynchronous run method on your OnMembersAddedAsync method after sending the activity of your greetings, use accessor by creating dialog state from your conversation state.



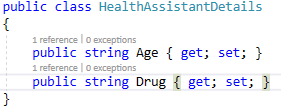
Do same for your OnMessageActivityAsync method. It will run the Dialog with the new message Activity.

Override another method, OnTurnAsync form your activity handler, await with saving your conversation and user states asynchronously, ignore forcing to save.

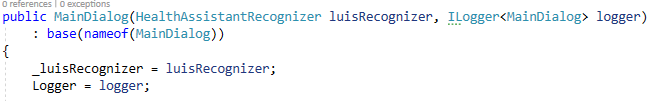


### Step10:

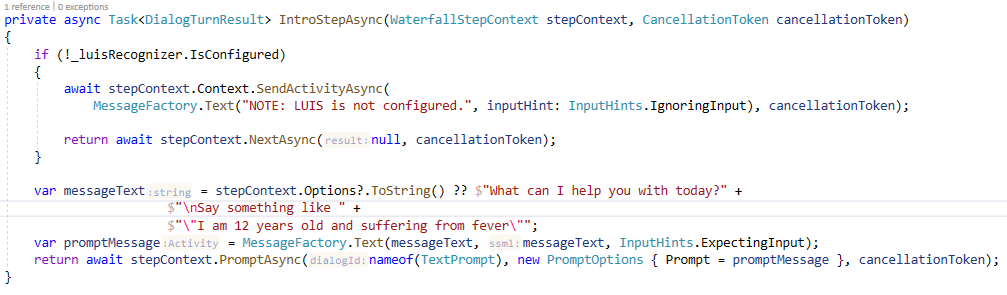
Before designing your Dialog, let’s design your model for entities.



Now time to design dialog component. Create a class for your dialog and inherit it from Microsoft.Bot.Builder.Dialogs.ComponentDialog. Declare following objects: one of your recognizer, and another ILogger. Initiate them with your dialog constructor.



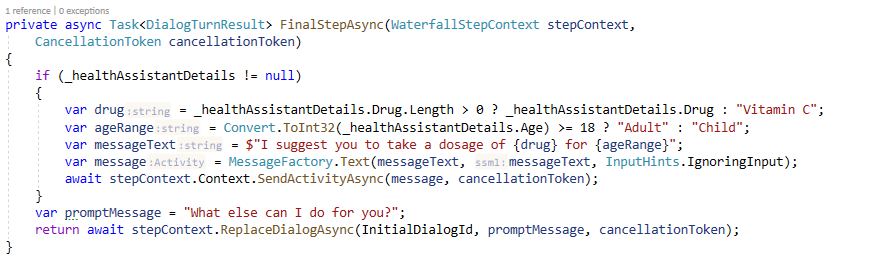
To initialize your dialog, declare a method asynchronously. Use waterfall context because we combine all required methods in a waterfall dialog, and obviously cancellation token.



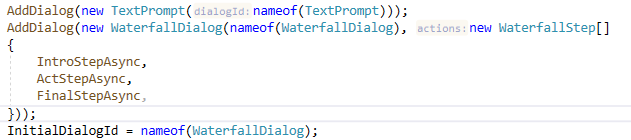
Do same works for other methods like ActStepAsync and



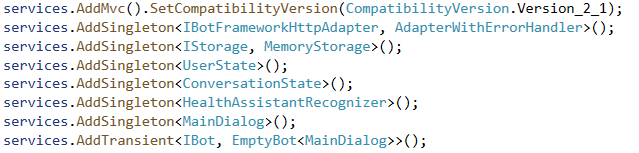
And also for FinalStepAsync



Add new text prompt and these methods in waterfall dialog



Go to your startup file, and resolve the dependency injection in services. Rewrite the Transient injection of your bot with generic aggregation of your Dialog. Add other dependency like Recognizer, User State, and Conversation State etc. with your Singleton service.



### Step11:

From your OnMessageActivityAsync remove reply component and Use logger to acknowledgement.

Now rebuild your project and run.

Test with your Bot Framework Emulator, enjoy.

