

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | WEATHER FORECAST WITH NODE.JS | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | MARZANA RAHMAN&RAIHANA RAHMAN |  | | | |
|  | | | | Date21-03-2021Course titleCse-317Teacher’s name Abu Kaisar Mohammad Masum |  | | | |
|  | | |  | | |  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**1 Abstract**

Creating chatbots is fun. With a variety of tools like AWS Lex, we can build conversational bots for business or for entertainment and use them in any text-powered applications.  
  
Implementations of conversational bots may include hailing a taxi, setting medical appointments, organizing calendars, shopping or money transfers.  
  
In this article I will guide you through the process of creating an AWS Lex-based weather bot . First, I wanted to test the possibilities of this Amazon service for building conversational interfaces

Nowadays there are many web services that provide the latest weather updates. **Open Weather Map** is a widely used [API](https://programmerblog.net/nodejs-mongodb-rest-api/) by developers. In this project, a user enters a city name and clicks the **Fetch Weather** button, weather data returned from Open Weather API is displayed to the user.

2. **INTRODUCTION:**

**Node.js:**

In this application we are using a Node.js application to get particular data from websites we selected. Node.js is an open source server environment, which can run on various platforms like Windows, Unix, Linux etc. Node.js uses JavaScript on the environment and uses asynchronous programming. Asynchronous programming basically is parallel programming, which means blocks of work run separately from each other. This eliminates the problem of waiting, which in some cases could dramatically affect the performance of the application. So, concluding one could say that the main benefit of asynchronous programming is the improved application performance and the responsiveness.

**Visual Studio Code:**

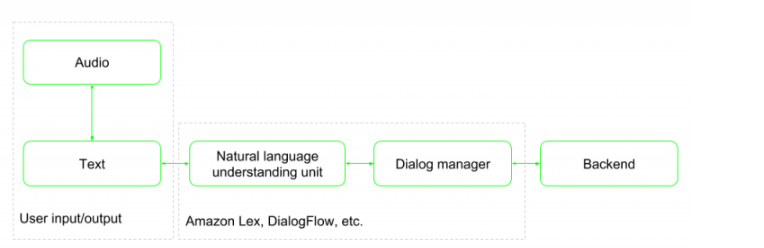
For this application I used the Visual Studio Code code editor. VSC was developed and designed for Windows, Linux and MacOS by Microsoft. It has various features such as debugging support, code competition suggestions, syntax highlighting etc. These features make the it easier to code JavaScript.

## How chatbots work:

At first glance, chatbots look just like other apps. They consist of an application layer, a database and APIs of external services. But in order to understand the user’s intent, chatbots need to be trained with the data.Based on how bots work, we can name three systems of classifying chatbots:

* **Rule-based systems** (aka **pattern matching**) – these are bots that use patterns to match queries like “Call <somebody>”, “Book a hotel in <city>” with the correct answers.</city></somebody>
* **Artificial Intelligence** – these bots use machine learning to select a category (intent) for the input phrases.
* **Hybrid** – a program that uses pattern matching, but is also backed by the human customer support.

A chatbot workflow looks as follows:

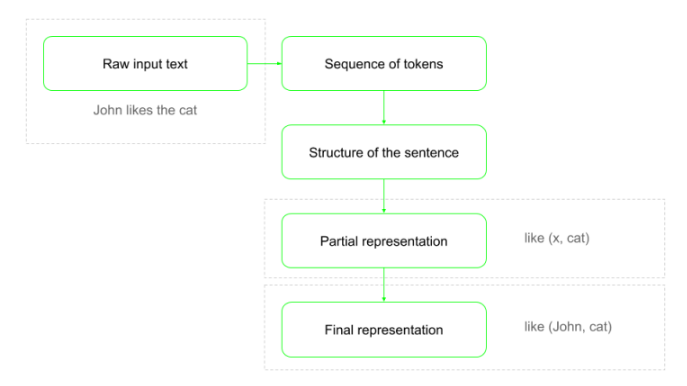


The workflow consists of two components:

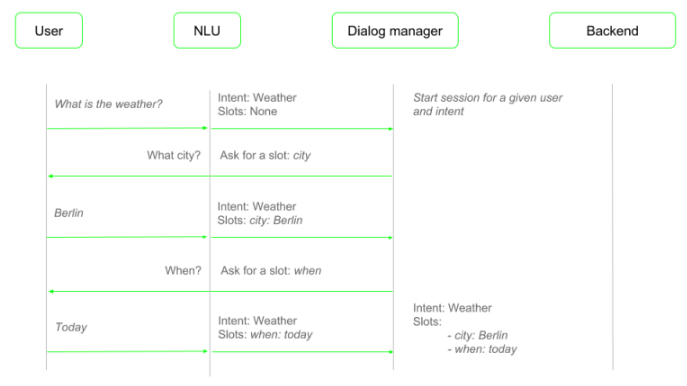
* The natural language understanding unit is responsible for natural language processing to decide which intent has to be selected (based on pattern matching or using ML techniques).
* The dialog manager holds the state of the conversation, keeps track of the selected intents as well as the slots, and communicates with the NLU unit to ask for the missing values.

The workflow consists of two components:

* The natural language understanding unit is responsible for natural language processing to decide which intent has to be selected (based on pattern matching or using ML techniques).
* The dialog manager holds the state of the conversation, keeps track of the selected intents as well as the slots, and communicates with the NLU unit to ask for the missing values.



As in any natural language algorithms, the first step is to tokenize and clean up the raw input.  
  
Tokenization splits the raw input into a sequence of “words”, removing the articles, prepositions and other parts of sentences. Next, using this tuple of words, the algorithm builds a structure tree out of the sentence. Tokenization is basically breaking up sentences into words.  
  
By using the tree, we are able to get a partial representation of where the main verb in the sentence is (eg. like which is the function) and then, in the final representation, we can assign the main noun (e.g. John) and the subject of the action (like) to be cat.The interaction between different components may look like this:



This particular workflow corresponds to the bot that I’ll describe later on in my weather bot example.

**DialogFlow vs. AWS Lex: Examples of bot platforms:**

To develop a chatbot, create one from scratch, or use a platform like Amazon’s AWS Lex to do it in an easier way.

To give a better overview of possibilities of such a service, I’ve compared AWS Lex with Google’s Dialog Flow (another popular chatbot platform available).

**DialogFlow:**

* Supports multiple languages (also multilanguage agents)
* Cooperation with backend via webhooks (not limited to Google services) or custom Cloud Functions
* Integrations with: Slack, Facebook, Twitter, Skype, Kik, Telegram, Amazon Alexa, Microsoft Cortana, etc.
* Enriched response cards for different integrations (eg. Slack, Facebook)
* Voice response via Google Assistance (using SSML format)
* Prebuilt agents that can be added to the project
* Small talks: simple agents that enhance personal experience (responding to “Hi”, “How are you”, etc.)
* Easy access to session attributes (so-called context)
* Intents can be triggered by button click, etc., not only by text recognition

**AWS Lex:**

Amazon Lex is a service for building conversational interfaces into any application using voice and text. Amazon Lex provides the advanced deep learning functionalities of automatic speech recognition (ASR) for converting speech to text, and natural language understanding (NLU) to recognize the intent of the text, to build applications with highly engaging user experiences and lifelike conversational interactions. With Amazon Lex, the same deep learning technologies that power Amazon Alexa are now available to any developer, enabling you to quickly and easily build sophisticated, natural language, conversational bots [(“chatbots”)](https://aws.amazon.com/what-is-a-chatbot/).

* Only US English
* Cooperation with backend only via AWS Lambda
* Integrations with: Slack, Facebook, Kik, Twillio SMS, (it’s possible to export chatbot to Amazon Alexa platform)
* Enriched response cards for different integrations (eg. Slack, Facebook)
* Voice response by using AWS Polly service (using SSML format)
* Session attributes only available via API

DialogFlow is a more mature product at this stage, offering multiple integrations with external services and backend resources. However, if you work inside an Amazon ecosystem, then Lex may seem like a more natural choice.

## Creating a bot: Weather information

I’ve divided this examples into 4 steps, so we can discuss them one by one at every stage of developing a bot. These steps are:

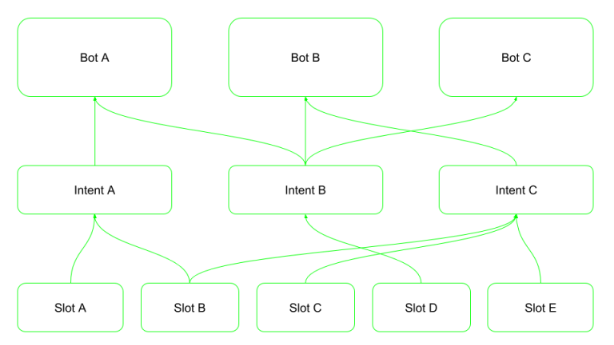
* Creating a bot
* Adding the engine
* Testing the bot

**Step 1: Create a bot:**

**AWS Lex projects are composed of three objects:**

* **Bot**
* **Intents**
* **Slots**

**The figure below depicts the relationship between these elements. A bot has to consist of at least one intent.**

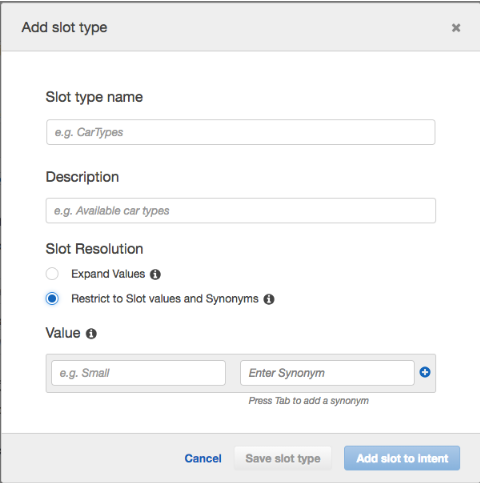


**The intent is what the user asks for, and will be executed whenever the intent is selected by the NLU module and all required slots are filled up. Intents are defined by a set of sample sentences, which are used to train the model, and slots. The sentences should be defined in such a way that they contain slots.**

**For example, if we have a slot type City, then one of our sentences could be Show me the weather in {city}. Now, the underlying ML uses this combination of sentences and slot types to train the model.**

**Slots are used to fetch the parameters required by the intent to fulfill the user request. There are two types of slots: the predefined and the custom ones. Amazon Lex supports built-in slot types from the Alexa Skills Kit.**

**Apart from the predefined slots, we can define our own slot type using a dialog shown below.**



Here, specify Slot type name and its Description. Then the Slot Resolution defines whether allowed values are restricted to the defined list. Alternatively, Slot Resolution to expand slot values, so they will be used as training data.  
  
With the enumerated restricted list of values, the synonyms that will be resolved to the declared Value.

**Set up project**

Workshop Roadmap

We use following AWS services into our project:

These Services are used in our weather bot project :

* Aws-amplify.
* Aws-lex.
* Amazon Cognito.
* Aws Cloud-formation.
* IAM Role.
* AWS Lambda.

**Fronted work**

**Follow those command** :

|  |
| --- |
| 1. mkdir weatherapplication |
|  |
|  |
| 1. cd myAngularBot |
|  |
| 1. npm install -g @angular/cli |
| 1. ng new weatherbot |
|  |
| 1. cd weatherbot |
| 1. ng g c auth |
| 1. ng g c home |
| 1. sudo npm install |
| 1. sudo npm start |
|  |
| 1. sudo npm install -g @aws-amplify/cli |
| 1. amplify configure |
|  |
| T**eam alpha weatherapp:** |
| npm install --save aws-amplify    npm install --save aws-amplify-angular |
|  |
|  |

|  |
| --- |
| sudo npm i --save @aws-amplify/ui-angular |
|  |
| amplify init |
|  |
| amplify add auth |
|  |
|  |
|  |
| code . |
|  |
| 12. Going to the polyfills.ts file  (window as any).global = window;  (window as any).process = {  env: { DEBUG: undefined },  }; |
| 13. Then go to src/app/module/app.module.ts and add following code as like as image  import {AmplifyUIAngularModule} from '@aws-amplify/ui-angular';  import Amplify from 'aws-amplify';  import amplify from '../aws-exports';  import {AmplifyChatbot} from "@aws-amplify/ui-angular";  Amplify.configure(amplify);  Amplify.configure({  Interactions: {  bots: {  "WeatherBot": {  "name": "WeatherBot",  "alias": "$LATEST",  "region": "us-east-1",  }  }  }  }); |
| 14. Then add "AmplifyUIAngularModule" into declarations[AmplifyUIAngularModule] |
| 15. Then go to src/app/app-routing.module and add following code  {  path: "login",  component: AuthComponent  },  {  path: "home",  component: HomeComponent  },  {  path: '',  redirectTo: '/login',  pathMatch: 'full'  } |
| 16. Then go to src/app/auth/auth.component.html and add following code  <amplify-authenticator></amplify-authenticator> |
| 17. Then go to src/app/home/home.component.html and add following code  <amplify-chatbot bot-name="WeatherBot"></amplify-chatbot> |
|  |
| 18. For publishing this app we use following command:  amplify publish |

**Backend work**

Create Lambda Function For Using Serverless Framework

1. **Install Aws CLI into local machine by following these steps from this link:**

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html>

* Then configure aws CLI by providing access key and secret access key

|  |
| --- |
| aws configure –profile bluebird |

1. **Then install serverless framework and configure it.**

|  |
| --- |
| npm install -g serverless  serverless login |

1. **Then check serverless through following command:**

|  |
| --- |
| serverless |

1. **Then open terminal into the directory where you want to create your lambda function**.
2. **Then write following command one after another**

|  |
| --- |
| mkdir chatbot |
| cd chatbot |
| serverless create --template aws-nodejs --path my-bot --profile bluebird |
| cd my-bot  Then open this project with visual studio with comamnd promt:  code . |

|  |
| --- |
|  |

1. It will open into visual studio code like this

|  |
| --- |
|  |

1. After creating my-bot in visual studio code it will create this files.
2. serverless.yml is the configuration file. We will cut all the comment section into this file. And change the default function in getWeather instead of hello.

|  |
| --- |
|  |

1. Then go to the handler.js and create the function name getWeather instead of hello.

|  |
| --- |
|  |

1. now open a new terminal

|  |
| --- |
|  |

1. **Then run follwong command**

|  |
| --- |
| npm init |
| npm i axios |
| sls deploy --profile bluebird |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

1. **After deploying this, it will create a stack into cloudformation**

|  |
| --- |
|  |

1. created lambda function named my-bot-dev-getWeather in the Amazon Lex

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | |  | |

**‍**

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
|  | |  |  |  |  | |
|  |  | | | | |  |