AWS Chatbot Challenge

Amazon Lex and AWS Lambda

Benjamin Towner

Cloud Solutions Architect / Engineer

[AWS Chatbot Challenge 2](#_Toc488084029)

[ChatBot Design Concept 2](#_Toc488084030)

[Overview 2](#_Toc488084031)

[Uniqueness 3](#_Toc488084032)

[Design Components 3](#_Toc488084033)

[Intents 3](#_Toc488084034)

[Slots 3](#_Toc488084035)

[Lambda 4](#_Toc488084036)

[Webhooks 4](#_Toc488084037)

[Entry Requirements 4](#_Toc488084038)

[Demo Video 5](#_Toc488084039)

[Github 5](#_Toc488084040)

[Access to Working Bot 5](#_Toc488084041)

[Testing Instructions 5](#_Toc488084042)

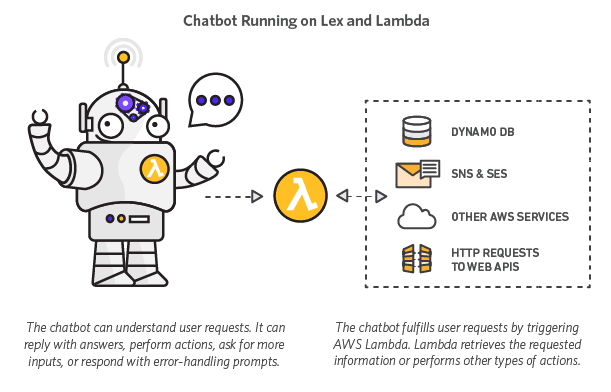
[Text Description 5](#_Toc488084043)

[Devpost Submission 5](#_Toc488084044)

# AWS Chatbot Challenge

Chatbots are changing how companies interface with their customers. With chatbots, you can easily fulfill the needs of your customers in an automated way using natural, human-like chat interfaces. Chatbots serve a variety of use cases, such as customer support, transaction fulfillment, data retrieval, or even DevOps functions (ChatOps).

However, building and running chatbots is a difficult task. First, most developers lack the deep learning expertise necessary to create bots that can intelligently interpret and respond to text. In addition, developers must also provision, manage, and scale the compute resources necessary to run the bot’s code.



What if you could build chatbots with sophisticated natural language processing and almost no operational overhead? [Amazon Lex](https://aws.amazon.com/lex/) is a fully managed service for building conversational interfaces into any application using voice and text. Lex is powered by the same deep learning technologies that power Amazon Alexa and lets you build natural language chatbots. Lex is integrated with [AWS Lambda](https://aws.amazon.com/lambda/), a service that lets you run code without provisioning or managing servers. Lambda enables you to write and run logic for your chatbot using serverless compute. Getting started with Amazon Lex and AWS Lambda is quick and easy.

# ChatBot Design Concept

## Overview

**MotherBot** helps by Sharing Calendars, Approved Contacts and provides a self-service administration platform for the *C.E.O. of the Home* to effectively schedule and organize. The AWS Chat Bot allows various processes to be made available to these household with the primary purpose of verifying the identity of the individual connections, formalizing an approval process and managing the ‘feasibility’ on the calendar.

## Uniqueness

Households are like mini-organizations, except their members or ‘***little users***’ often find themselves in a place of constant challenge and response. Seeking approval and then finding out how to facilitate getting it done is the world that ‘***Tweens’*** live in. The coordination required among Parents often requires effective communication practices regardless of your Work-life or Marital status. Technology can enable Parents or Parental guardians to manage the household workloads while providing an appropriate level of privacy and respect.

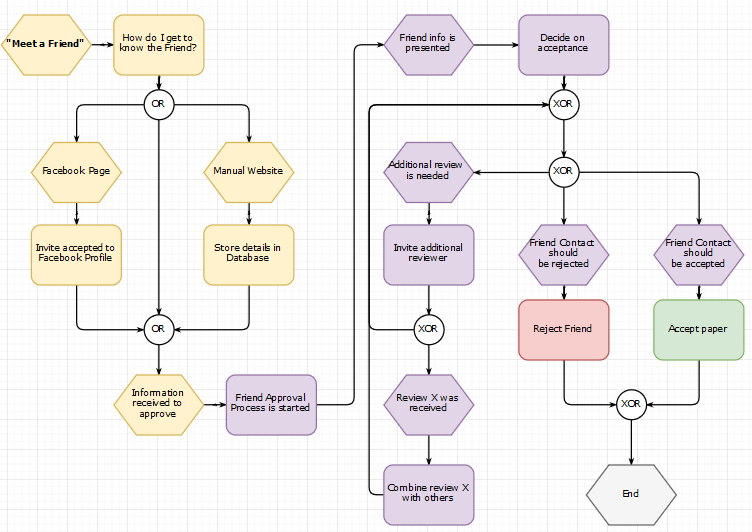
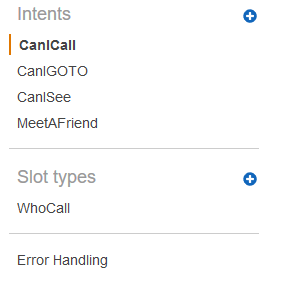
# Design Components

There are 3 main functional aspects to **MotherBot**.

1. Identity and Access Management (Contact Management)
2. Activity Approval
3. Calendar Feasibility

Models

## Intents



## Slots

* MeetAFriend
  + FriendInfo
* CanICall
  + WhoCall
* CanIGoTo:
  + ApprovedFriendsHome - Available approved Friend's House
  + ApprovedPublicPlaces – Pool Club, Library, Bowling, Mall
* CanISee:
  + Events –
  + Approved Movie
  + Approved Concert

## Lambda

Re-using some examples from other blueprints, the attached ’lex-motherbot-python.py’ can be used to create a Lambda function. This function uses the lambda\_handler to route functions based on the intent.

* Blueprints
  + lex-make-appointment
* Response Cards
* Lambda Functions as Code Hooks
* Lex- motherbot -python

def lambda\_handler(event, context):

"""

Route the incoming request based on intent.

The JSON body of the request is provided in the event slot.

"""

# By default, treat the user request as coming from the America/New\_York time zone.

os.environ['TZ'] = 'America/New\_York'

time.tzset()

logger.debug('event.bot.name={}'.format(event['bot']['name']))

return dispatch(event)

|  |  |  |
| --- | --- | --- |
| * + **elicit\_slot**   + **confirm\_intent**   + **close**   + **delegate**   + **build\_response\_card**   + **parse\_int**   + **try\_ex** | * + **increment\_time\_by\_thirty\_mins**   + **get\_random\_int**   + **get\_availabilities**   + **isvalid\_date**   + **is\_available**   + **get\_duration**   + **get\_availabilities\_for\_duration** | * + **build\_validation\_result**   + **validate\_book\_appointment**   + **build\_time\_output\_string**   + **build\_available\_time\_string**   + **build\_options**   + **make\_appointment**   + **dispatch**   + **lambda\_handler** |

# Entry Requirements

1. A **demo video** that clearly shows your bot functioning on its intended platform.
2. **Share your code** repository publicly or privately through GitHub or BitBucket with testing@devpost.com.
3. **Access to a working bot**.
4. **Testing instructions** with anything we need to know to test your bot.
5. **Text description** - include a brief explanation of what the bot Application does and what makes it unique.
6. A **completed submission form** on Devpost.

## Demo Video

Created using Skype for business, and produce the publish location.

## Github

https://github.com/bentowner/MotherBot

## Access to Working Bot

## Testing Instructions

Follow the workflow from this Document

## Text Description

This document will serve as the Text Description of MotherBot, the Challenge and the Submission.

## Devpost Submission