# Assignment 1: Dining Concierge AI Agent

Start Assignment

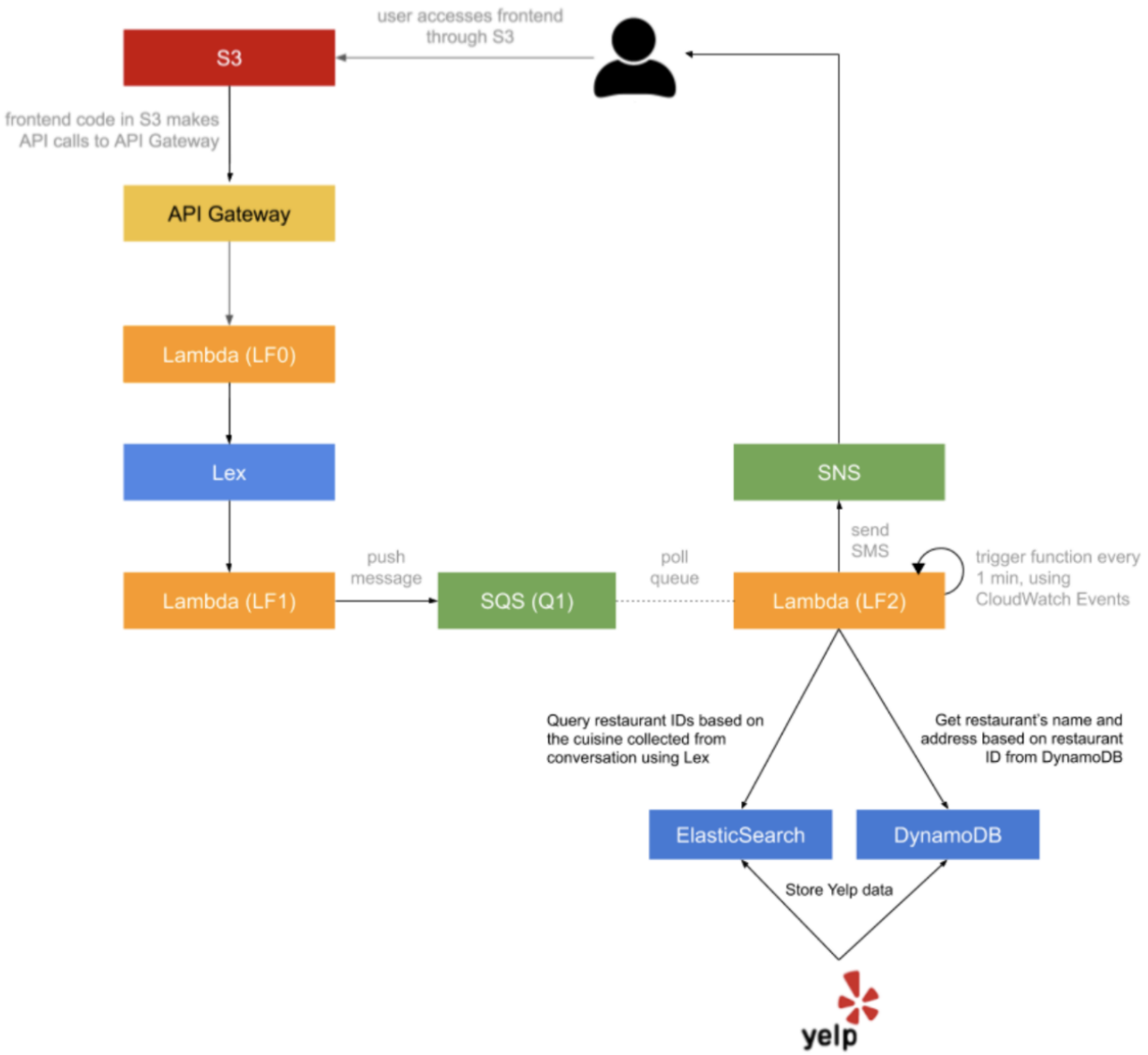
**Due** Oct 12 by 11:59pm **Points** 110 **Submitting** a text entry box, a website url, or a file upload
**Available** Sep 21 at 12am - Oct 31 at 11:59pm about 1 month

Due Date: Oct 12th 11:59pm

You may do this homework in a group of size two students.

Architecture Diagram if you unable to view at the bottom of the doc:

## Architecture Diagram




Customer Service is a core service for a lot of businesses around the world and it is getting disrupted at the moment by Natural Language Processing-powered applications. In this first assignment you will implement a serverless, microservice-driven web application. Specifically, you will build a Dining Concierge chatbot that sends you restaurant suggestions given a set of preferences that you provide the chatbot with thro

Course Chat

**Outline:**

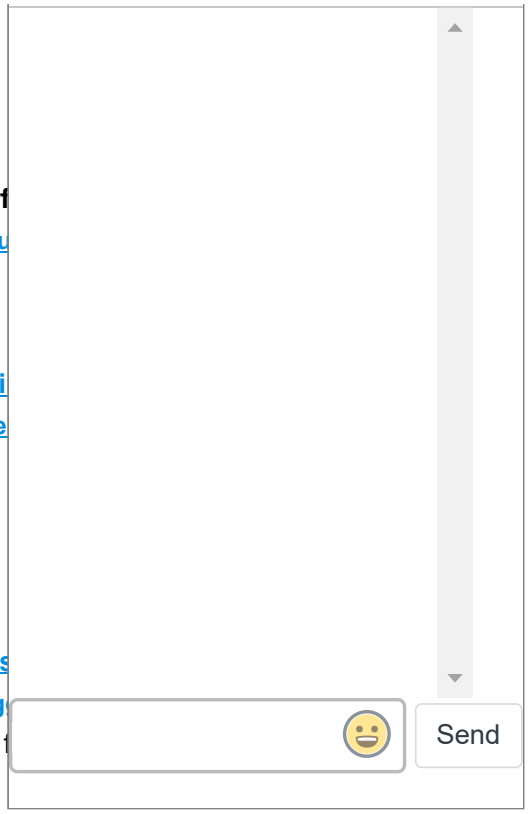
This assignment has the following requirements:

- **Build and deploy the frontend of the application**
    1. **Repurpose the following frontend starter application to interf**
        1. **https://github.com/ndrppnc/cloud-hw1-starter (https://githu**
- **Host your frontend in an AWS S3 bucket**
    1. Set the bucket up for website hosting
    2. **https://docs.aws.amazon.com/AmazonS3/latest/dev/Hosti (https://docs.aws.amazon.com/AmazonS3/latest/dev/HostingWe**
- **Build the API for the application**
    1. Use API Gateway to setup your API
        1. use the following API/Swagger specification for your API
- **https://github.com/001000001/aics-columbia-s2018/blob/master/aics (https://github.com/001000001/aics-columbia-s2018/blob/master/aics-swag**
- Use **http://editor.swagger.io/ (http://editor.swagger.io/)** to visualize this
- You can **import the Swagger file into API Gateway**
    - **https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-import-api.html (https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-import-api.html)**
- **Create a Lambda function (LF0)** that performs the chat operation
    - Use the request/response model (interfaces) specified in the API specification above
- For now, just implement a boilerplate response to all messages:
- ex. User says anything, Bot responds: "I'm still under development. Please come back later."
- Notes

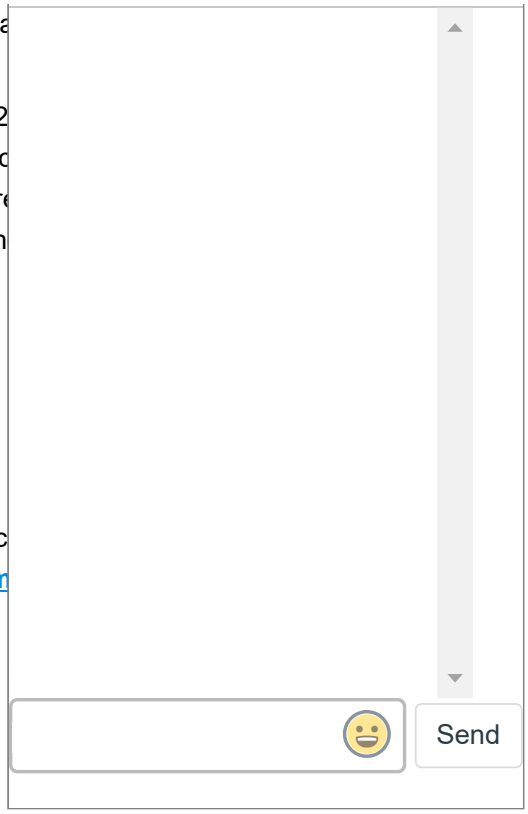
1. You will need to **enable CORS on your API methods**

- **https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors.html (https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors.html)**
- API Gateway can **generate an SDK for your API**, which you can use in your frontend. It will take care of calling your API, as well as session signing the API calls -- an important security feature
- **https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-generate-sdk-javascript.html (https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-generate-sdk-javascript.html)**
- **Build a Dining Concierge chatbot using Amazon Lex.**
    1. Create a new bot using the Amazon Lex service. Read up the documentation on all things Lex, for more information: **https://docs.aws.amazon.com/lex/latest/dg/getting-started.html (https://docs.aws.amazon.com/lex/latest/dg/getting-started.html)**
    2. Create a Lambda function (LF1) and use it as a code hook for Lex, which essentially entails the invocation of your Lambda before Lex responds to any of your requests -- this gives you the chance to manipulate and validate parameters as well as format the bot's responses. More documentation on Lambda code hooks at the following link: **https://docs.aws.amazon.com/lex/latest/dg/using-lambda.html (https://docs.aws.amazon.com/lex/latest/dg/using-lambda.html)**
    3. Bot Requirements:
        1. Implement at least the following three intents:

- GreetingIntent
- ThankYouIntent
- DiningSuggestionsIntent



- The implementation of an intent entails its setup in Amazon Lex as well a function code hook.
- Example: for the GreetingIntent you need to 1. create the intent in Lex, 2 console, 3. implement the handler for the GreetingIntent in the Lambda c request for the GreetingIntent you compose a response such as “Hi there
- For the DiningSuggestionsIntent, you need to collect at least the followin through conversation:
- Location
- Cuisine
- Dining Time
- Number of people
- Phone number
- Based on the parameters collected from the user, push the information c etc.) to an SQS queue (Q1). More on SQS queues here: **https://aws.am (https://aws.amazon.com/sqs/)**
- Also confirm to the user that you received their request and that you will list of restaurant suggestions.

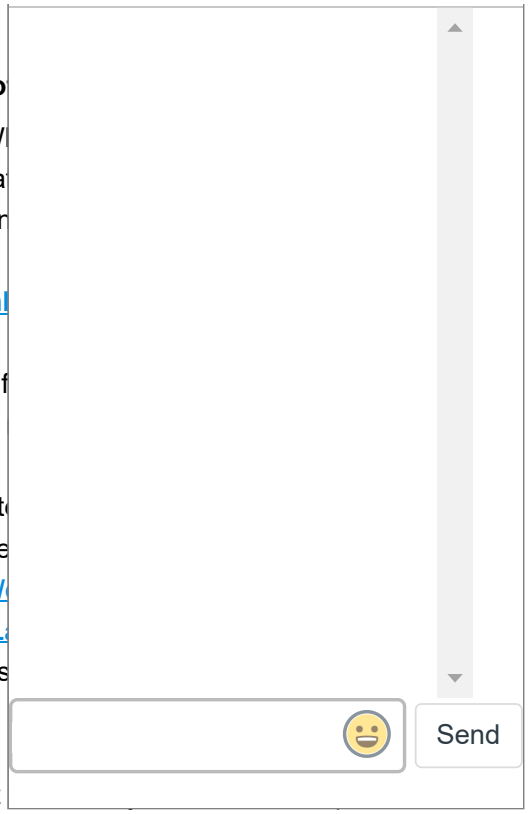



- **Integrate the Lex chatbot into your chat API**
    1. Use the AWS SDK to call your Lex chatbot from the API Lambda (LF0).
    2. When the API receives a request, you should 1. extract the text message from the API request, 2. send it to your Lex chatbot, 3. wait for the response, 4. send back the response from Lex as the API response.

- **Use the Yelp API to collect 5,000+ random restaurants from Manhattan.**
    1. Use the following tools:
        1. Yelp API
- Get restaurants by your self-defined cuisine types
    - You can do this by adding cuisine type in the search term ( ex. Term: chinese restaurants)
- Each cuisine type should have 1,000 restaurants or so.
- Make sure your restaurants don’t duplicate.
- **DynamoDB (https://aws.amazon.com/dynamodb/)** (a noSQL database)
- Create a DynamoDB table and named “yelp-restaurants”
- Store the restaurants you scrape, in DynamoDB (one thing you will notice is that some restaurants might have more or less fields than others, which makes DynamoDB ideal for storing this data)
- With each item you store, make sure to attach a key to the object named “insertedAtTimestamp” with the value of the time and date of when you inserted the particular record
- Store those that are necessary for your recommendation. (Requirements: Business ID, Name, Address, Coordinates, Number of Reviews, Rating, Zip Code)
- Note: you can perform this scraping from your computer or from your AWS account -- your pick.

- **Create an ElasticSearch instance using the AWS ElasticSearch Service.**
- Create an ElasticSearch index called “restaurants”
- Create an ElasticSearch type under the index “restaurants” called “Restaurant”
- Store partial information for each restaurant scraped in ElasticSearch under the “restaurants” index, where each entry has a “Restaurant” data type.
    - You only need to store RestaurantID and Cuisine for each restaurant

- **Build a suggestions module, that is decoupled from the Lex chatbo**
  1. Create a new Lambda function (LF2) that acts as a queue worker. W
     from the SQS queue (Q1), 2. gets a random restaurant recommenda
     conversation from ElasticSearch and DynamoDB, 3. formats them ar
     phone number included in the SQS message, using SNS
     (https://docs.aws.amazon.com/sns/latest/dg/SMSMessages.html
     (https://docs.aws.amazon.com/sns/latest/dg/SMSMessages.html) ).
     1. Use the DynamoDB table “yelp-restaurants” (which you created f
        about the restaurants (restaurant name, address, etc.), since the
        have only a small subset of fields from each restaurant.
     2. Modify the rest of the LF2 function if necessary to send the user t
  2. Set up a CloudWatch event trigger that runs every minute and invoke
     result: https://docs.aws.amazon.com/AmazonCloudWatch/latest/
     (https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/RunL
     queue worker Lambda to poll and process suggestion requests on its




**In summary,** based on a conversation with the customer, your LEX chatbot
‘cuisine’. You will search through ElasticSearch to get random suggestions of restaurant IDs with this cuisine. At this point, you would also need to query the DynamoDB table with these restaurant IDs to find more information about the restaurants you want to suggest to your customers like name and address of the restaurant. Please note that you do not need to worry about filtering restaurants based on neighbourhood in this assignment. Filter only using the cuisine.

**Extra Credit**

Implement state for your concierge application, such that it remembers your last search for both location and category. When a user returns to the chat, they should automatically receive a recommendation based on their previous search. You can use DynamoDB to store intermediary state information and a separate Lambda function to handle the recommendation based on the last search.

**Example Interaction**

**User: Hello**

Bot: Hi there, how can I help?

**User: I need some restaurant suggestions.**

Bot: Great. I can help you with that. What city or city area are you looking to dine in?

**User: Manhattan**

Bot: Got it, Manhattan. What cuisine would you like to try?

**User: Japanese**

Bot: Ok, how many people are in your party?

**User: Two**

**Course Chat**

Bot: A few more to go. What date?

**User: Today**

Bot: What time?

**User: 7 pm, please**

Bot: Great. Lastly, I need your phone number so I can send you my findings.

**User: 123-456-7890**

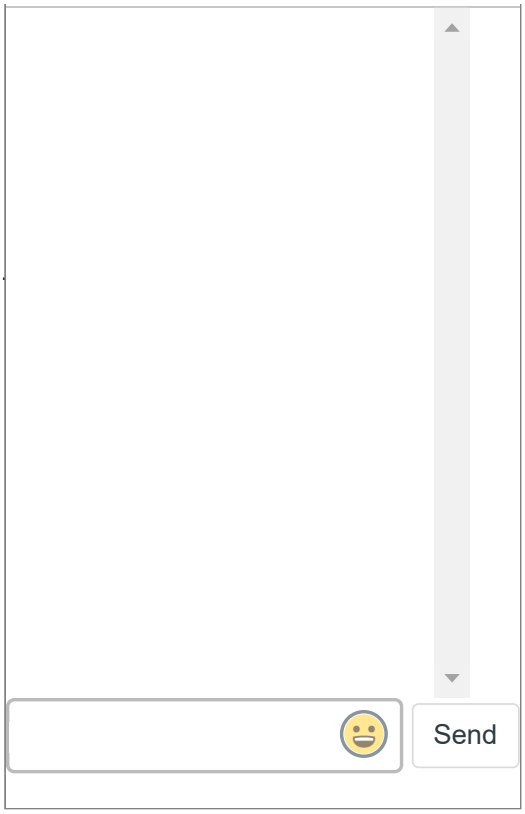
Bot: You're all set. Expect my suggestions shortly! Have a good day.

**User: Thank you!**

Bot: You're welcome.

(a few minutes later)

*User gets the following text message:*




"Hello! Here are my Japanese restaurant suggestions for 2 people, for today at 7 pm: 1. Sushi Nakazawa, located at 23 Commerce St, 2. Jin Ramen, located at 3183 Broadway, 3. Nikko, located at 1280 Amsterdam Ave. Enjoy your meal!"

**ANNEX**

Architecture Diagram


**Course Chat**