Tripal: Web Application For Trip Planning

Chen Huang
ch3371@columbia.edu
MSEE

Chenyu Xi
cx2219@columbia.edu
MSEE

Jiliang Ma
jm4750@columbia.edu
MSEE

Qinyuan Wei
qw2264@columbia.edu
MSEE

***Abstract*—We developed a novel travel assistant application called Tripal to integrate different travel websites and applications so as to reduce the searching cost for our users. Our application is built on top of AWS portfolio, along with several APIs to provide necessary travel information about flight, hotel, and attraction of the destination. Designed with a user-friendly interface and considerate functionality, our application achieves a balance between enhanced functionality and easy-to-use UI design.**

***Index Terms*—Travel, Assistant, AWS**

I. Introduction

Tripal is an application developed for customers who want to plan a trip but find it hard to gather information about their destination from different websites and applications. Considering this scenario: Ken is planning for a trip to New York, a normal process for him to make a travel plan is searching for the attractions in this area, and then use the attraction information to searchnearby hotel information. When he decides to go, he also needs to book a flight. However, the information needed for a travel plan always locates at many different websites and applications [1]. Therefore, making a trip plan would be a time-consuming work if people have to search and analyze all information separately. In order to tackle this problem, we design a travel application which collects all the necessary information to make it easier for our users to make travel plans.

The majo r function of our application is a trip advisor chatbot, where user can interact with our AI travel assistant [2] through simple conversation. Thus, it's easy for the user to start with our app. They only need to tell out chatbot their travel destinations. Once get the city name, our AI assistant will reply user with a list of attractions as well as detailed information. From the attraction list, user can pick up attractions they plan to go and get suggestions of nearby hotels. At the same time, the user can get the flight information by specifying where and when they plan to depart and return. To help the user who wants to save their searching result, we design a wishlist function in our application where people can create and check their travel plans. To sum up, the novel part of our application are:

- Simplifying search procedure for travel information by collecting and integrating results from different websites and applications.
- An AI-driven application which provides user-app interaction through implementing Natural Language Processing.
- Considerate functionality for the user to write down their travel idea and track travel history

II. Architecture

How third-party services, such as AWS Service and Google APIs, are integrated into our application is illustrated in Fig. 1. For the front-end part, we host the website in

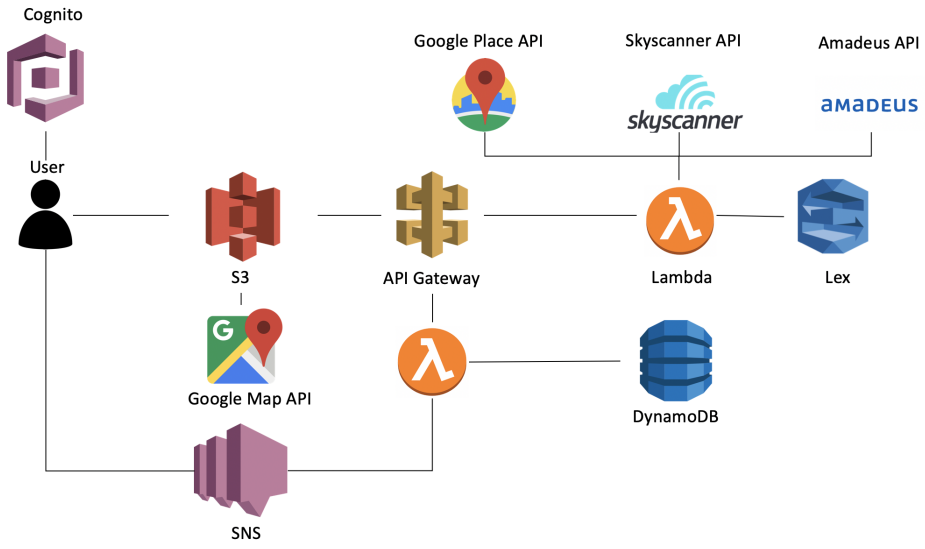


Fig. 1: Tripal Structure Overview

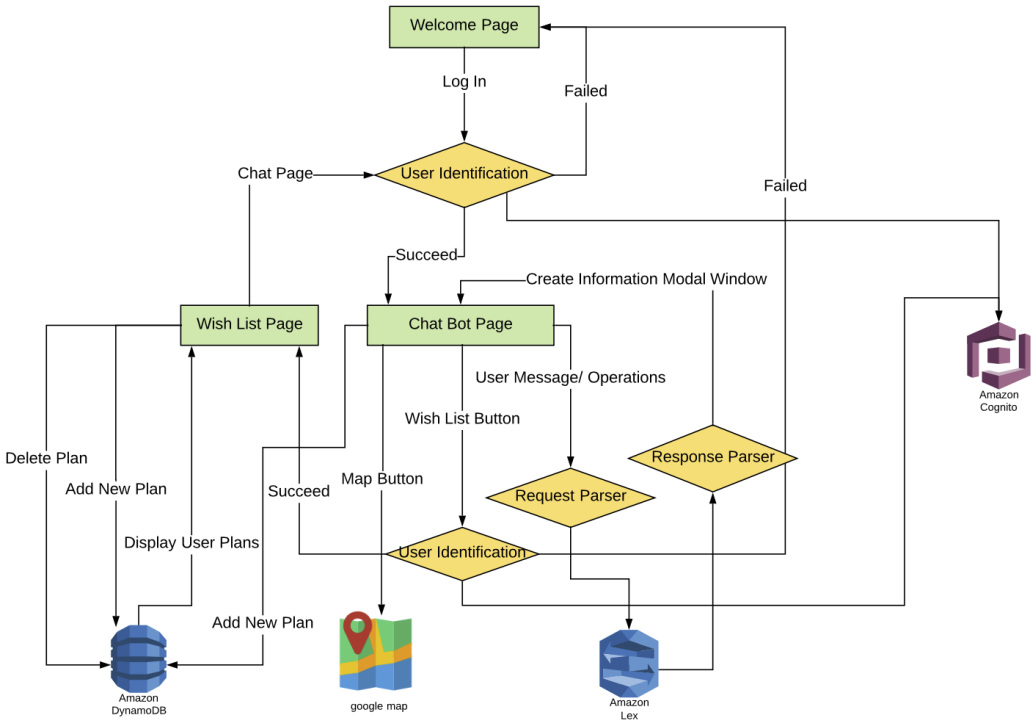


Fig. 2: Front-end Structure

S3 bucket and use Cognito for user authentication [3]. The API gateway plays the role as a single entry point to our system, excluding unauthorized access to our back-end. We implement several lambda functions to handle the request from API Gateway. We have four lambda functions to make calls to third-party APIs. Three of these lambda functions are responsible for parsing different API responses and convert them to specified formats. Then we adopt one extra lambda function called proxy lambda to integrate the API-parser lambda functions and forward information between API-parser and AWS Lex service [4].

We use DynamoDB to store travel plans. To implement the wishlist function, we have three lambda functions connecting front-end and DynamoDB for creating records, querying records and deleting records. For better user experience, we use SNS service to offer the user an option to get their wishlist content through SMS [5]. More detail of our application will be provided in the next section.

III. Algorithm and Design Detail

A. Front-end

For the front end part, our web application contains three pages in total. They are welcome page, chatbot page, and wishlist page. The front-end structure is shown in Fig. 2. Front-end work is mainly based on Bootstrap and jQuery, and we also add animation effects with CSS to improve user experience [6].

1) Welcome page and user identification: Our web application implements a user system with AWS Cognito service. The major function of the welcome page is to redirect the user to Cognito built-in Sign-in and Sign-up web pages. Once user logged in, we are able to retrieve current user's id token and access token from the URL of call back page.

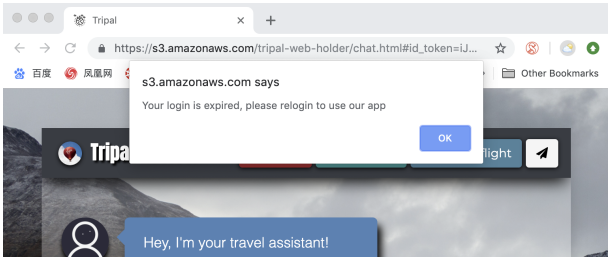


Fig. 3: Unauthorized User

We implement a user identification in both the chat page and wishlist page. The user identifier firstly obtains user's id token from URL and then use JavaScript AWS SDK to create AWS credentials with Cognito identity pool, which offers authenticated user AWS services accessibility [7]. At the same time, we use the access token to query user's detailed information such as user id and user name. User identifier identifies the current user every time they enter chat or wishlist page. For any user fails to pass the user identifier, we redirect them back to the welcome page, like Fig. 3. The logic for user identification function is shown in Fig. 4.

2) Wishlist page: The wishlist page has two functions. Firstly, it serves as a to-do application which allows the user

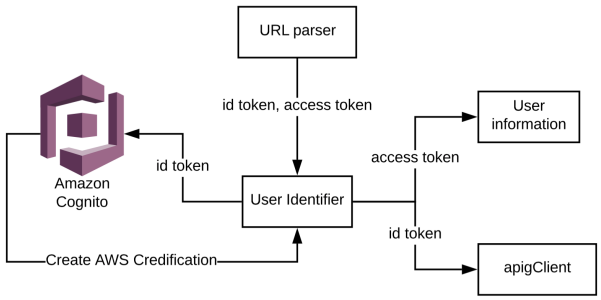


Fig. 4: Logic for User Identifier

to add and delete their travel plans. Secondly, it also allows user to send the wishlist to their phone via AWS SNS service [8]. The logic for the wishlist page is shown in Fig. 5.

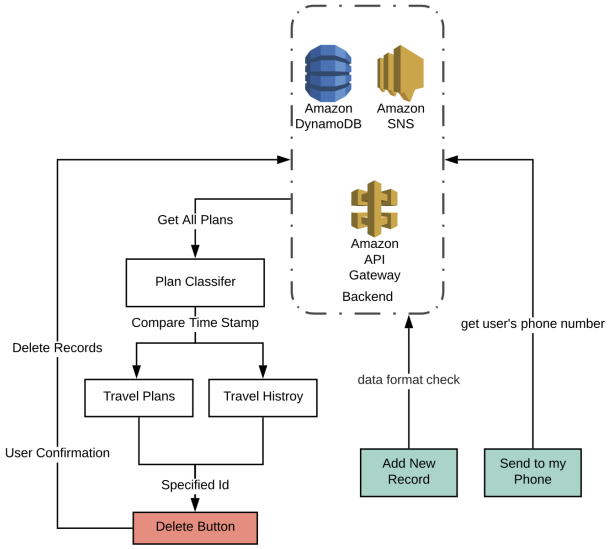


Fig. 5: Logic for Wishlist Page

After the user enters the wishlist page and passes the user identification, the front-end queries API Gateway to get all plans stored in DynamoDB that belong to the current user[9]. Then we classify the plans into two groups, future plan, and history, by comparing the plan date with the current time. For each recorded plan of the current user, we create a new li label with a bootstrap card to display the detail information. We use the record id, generated by combining user id and creation timestamp, to generate label id for each plan entry. When user hits the delete button, we use the label id to delete the same record from DynamoDB [10].

The add new plan button invokes a pop up plan creation window. User can enter plan date, place and idea and we will send this information to DynamoDB through API gateway, along with user id and auto-generated record id. When the user hits the send to my phone button, the click action will call the send wishlist function, which gets user's phone number through the prompt window and sends it along with current user's user id to API gateway.

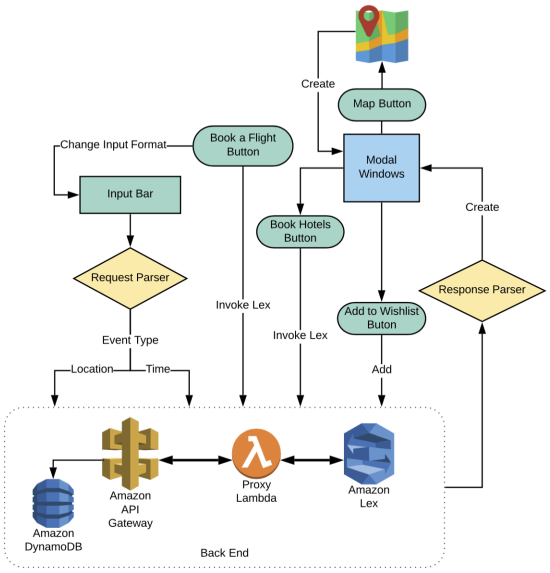


Fig. 6: Logic for Chatbot Page

3) Chatbot page: The chatbot page is the core of tripal app. The chat page has two main functions: user can book a flight through chatting with tripal advisor, and they can get recommended attractions with detailed information by simply offering tripal advisor their target city. The logic for the wishlist page is shown in Fig. 6.

As shown in Fig. 7, when the user hits the "book a flight" button in the navigation bar. We automatically send a special message to lambda proxy in order to invoke the Lex. The Lex then ask the user to fill up four fields needed for booking a flight, which are current location, destination, depart time and return time [11].

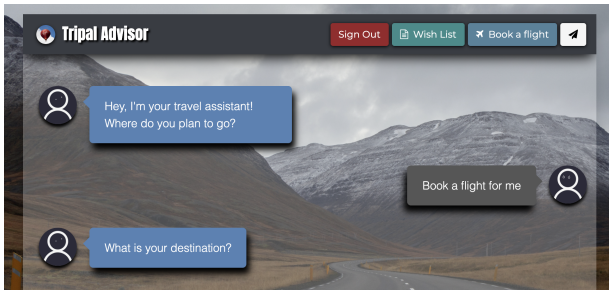


Fig. 7: Book a Flight

The input widget will automatically adapt to the scenario. That is, when the user needs to input a date, we set the type of input label in chat page from "text" to "date", as shown in Fig. 8. This modification could both improve the user experience as well as simplify the back-end work in input legality test.

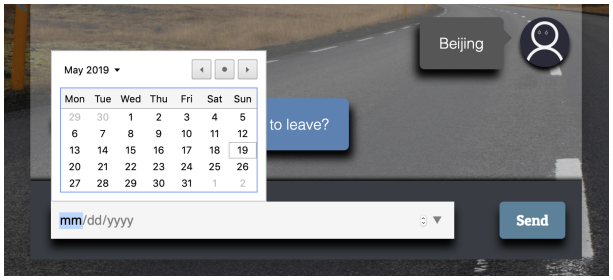


Fig. 8: Input Format: Date

User can input any city name to get relevant attractions. For each attraction, we offer four options: add this attraction to wishlist, find a nearby hotel, map, and go to attraction's website. We store the location information of each attraction when creating attractions recommendation modal window. The location information is necessary for booking hotel and display map functions.

The onclick function of map button takes longitude, latitude as well as attraction name as arguments. This function creates a template modal window to implement Google Map API, and also add an EventListener to the modal close event, which destroys the modal to avoid redundant codes. The onclick function of "book a hotel" button automatically sends the current location to the back-end to retrieve hotel information and hotel information is also shown as a clickable message.

The add to wishlist button has the same logic with the add a new plan button in wishlist page, the only difference is that we will fill the place field for the user.

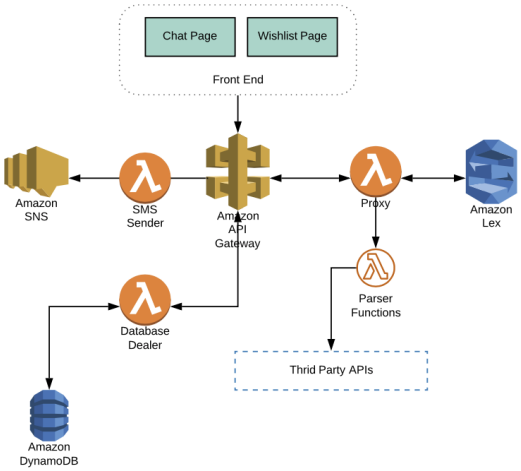


Fig. 9: Logic for Backend

To make the user interface simple and beautiful, instead of displaying all information retrieved from the back-end directly in the chat box, we use bootstrap modal with list-group or accordion collapse board to display any suggestion received from back-end. In order to achieve this, we create functions to parse the response JSON file from API gateway and use jQuery to create modal windows in front-end. The tricky part is that we bootstrap is not compatible with a nested modal window which is necessary for us to allow the user to open the map window within the attractions window. We use a hidden button in front-end to open the map window when user clicks the map button in attractions window to solve this problem.

B. Back-end

For the back-end part, as the structure is shown in Fig. 9, our web application implements lambda functions to retrieve information from third-party APIs and to parse the response to a frontend friendly format, and also use AWS back-end services, such as DynamoDB and SNS to enhance our functionality.

algorithm 1 Back-end lambda function logic

Input: message received from API Gateway

```
if "find hotel" in Input then
    Invoke Tripal-hotel lambda with lng and lat
    return hotel information
else Invoke Lex with input
    if IndentName == "Greeting" or "Thankyou" then
        return Lex response
    end if
    if slots are fulfilled then
        if IntentName == Attraction then
            Trigger Tripal-attraction lambda
            Trigger Tripal-attraction-detail lamdba
            return Attraction information
        end if
        if IntentName == Flight then
            Trigger Tripal-flight lambda
            return Flight information
        end if
    else
        return Lex response
    end if
end if
```

1) Lambda, API: Lambda functions process a variety types of requests, such as searching for attraction details, flight information, and hotel information as well as the basic interactive utterances to gather information, so our lambda function framework has an important feature that we adopt modular programming approach. Specifically, we determine different modules of our web logic into multiple lambda functions, say "Worker lambda" and write a "Driver lambda" to call the specific "Worker lambda" according to the type of request. In that way, we greatly improve our web application in the ease of use, maintenance, and reusability.

We implement the "Driver lambda" by checking if there are keywords matching according to front-end input and call the corresponding "Worker lambda". In specific, according to users' instructions, we add modifications or special characters so that Lex can trigger the right intent and gather corresponding information, such as greeting intent, thankyou intent, hotel intent, flight intent, and attraction intent. After

Lex gathers all the required slots and triggers the fulfillment, "Driver lambda" then calls the corresponding "Worker lambda" to search APIs. To illustrate our implementation, we name the "Driver lambda" as "Tripal-proxy", name the "Worker lambdas" as "Tripal-hotel", "Tripal-attraction", "Tripal-flight". Detailed back-end lambda function logic is shown in algorithm 1.

For Tripal-hotel, since hotel market is a highly competitive while profitable market, we find it hard to apply an API service from a big-name company, hence we select a comparatively smaller company called Amadeus which provides a quite powerful hotel searching function that enables us to find the best hotel offers that match our search from a wide choice of providers [12]. For the information we collect from front-end, say the latitude and longitude of an attraction, we query the Amadeus hotel API and then after clearing up the content and format, we send name, address, price, rating and contact information back to front-end. The format of Tripal-hotel response is shown in Listing 1.

Listing 1 Tripal-hotel response

```
{
    name:"hotel-name"
    rating: "hotel-rating"
    address: "hotel-address",
    contact: "hotel-contact",
    price: "hotel-price"
}
```

For Tripal-attraction, we use Google places API, which is a service that returns information about places using HTTP requests provided by Google to developers in Google cloud. Google places API supports a wide range of place requests, such as place search, place details, place photos, and place autocomplete [13]. In our application scenario, we choose to use place search combined with place details to provide useful and detailed information to Tripal users. We first query a list of popular attractions at the place from user's input with "place search" function and query the detailed information for each attraction with "place details" function to provide address, rating, top review, and website information to users. The format of Tripal-attraction response is shown in Listing 2.

Listing 2 Tripal-attraction response

```
{
    name: "attraction-name"
    address: "attraction-address",
    rating: "attraction-rating",
    review: "attraction-review",
    pic-url: "attraction-pic-url",
    location: "lng and lat"
}
```

For Tripal-flight, we use API provided by Skyscanner, which can provide price feed and real-time streaming of prices with a RESTful API format [14]. With Lex's fulfillment of flight intent, we make requests for flight suggestions on the basis of "Destination", "Departure", "Leave_time", "Back_time", then we clear up the results from API and send recommended flight with airline, leave_time, back_time, and price information to front-end. Furthermore, to provide a one-click link for users to book the ticket, we implement a Kayak link generator with string manipulation. The core point is that we search the three letter code of the airport in a JSON file with the city name from users' input and thus concatenate a valid Kayak booking link with correct travel information. The format of Tripal-flight response is shown in Listing 3.

Listing 3 Tripal-flight response

```
{
    depart-date: "departure date"
    back-date: "date of back",
    price: "flight's price",
    airline: "airline's name"
}
```

2) DynamoDB: DynamoDB is a NoSQL database service that supports key-value and document data structure. We use the DynamoDB to store and query the users' wishlist data.

Data Schema: Due to the number of records we have and the property of the records, we decided to use a key-value database. For each item, we have several keys: textID, idea, peopleID, place, planDate, and timeStamp. All corresponding values are strings. The primary key is textID, which is unique for each item. These keys can fully describe the information in wishlist and from which we can reconstruct the wishlist. The data schema is shown in TABLE I

Wishlist Document Schema		
textID	String	Primary Key
idea	*String*	Text about plan details
peopleID	*String*	Specified plan owner
planDate	*Date*	When is the plan date
timeStamp	*Date*	When the plan created
place	*String*	The destination place

TABLE I: Data Schema of Tripal Wishlist

Database Accessing: In Tripal, we use three kinds of operations to the database, which are create, query and delete. For each one of these methods, we create a lambda function to implement it. Then we link these lambda functions to the API gateway for accessing from the frontend. For create, we have a lambda function called *tripal_dynamoDB_createitem*; for query, we

have a lambda function called *tripal_dynamoDB_query*; for delete, we have a lambda function called *tripal_dynamoDB_deleteitem*. In this function, we use the function *put_item()* from the client in the package boto3 to create a new item in DynamoDB. In *tripal_dynamoDB_query*, we use the function *scan()* from the Dynamodb table to query the items by the key of items. And since the only use case here is querying by peopleID, so the key of the input can be restricted to peopleID, which can simplify the lambda function. In *tripal_dynamoDB_deleteitem*, we use the function *delete_item()* from the client in the package boto3 to delete items by its primary key. Then we use API gateway to encapsulate the lambda functions above so that the front-end can access the DynamoDB from the API gateway. We use the GET method to implement the query and delete, and use the POST method to implement the create. The format of the input to create, delete and query is shown in Listing 4, 5, and 6.

Listing 4 Input to Create

```
{
    peopleID: "peopleID of the item"
    idea: "description of the
     ↪ wishlist",
    place: "place to go",
    planDate: "the date of the plan",
    textId: "the unique id of this
     ↪ item",
    timeStamp: "the timestamp of this
     ↪ item"
}
```

Listing 5 Input to Delete

```
{
    textId: "the unique id of the
     ↪ item that want to delete"
}
```

Listing 6 Input to Query

```
{
  keyname: "peopleID",
  content: "the peopleID of the
   ↪ target items"
}
```

3) SNS: Simple Notification Service (SNS) is the messaging service provided by AWS, which is highly available, durable, secure, and fully managed. SNS can be used to fan out notifications to end users using mobile push, SMS, and email. We use SNS to send wishlist information to users by SMS.

To implement this feature, we write a lambda function called *tripal_sns*. This lambda function has two parts: First, we gather the wishlist information from DynamoDB using the query method which described in the DynamoDB part. Then we use the SNS to send the wishlist information to the target phone. The message sent to users includes information about destination, date, and detail.

After finishing the lambda function, we use API gateway to encapsulate it so that the frontend can use the SNS service from the API gateway. The method we use is POST.

4) Lex: Accepting both text and voice input, Amazon Lex is a service for building conversational interface in applications. Using deep learning as service back end, Lex is able to achieve functionalities of Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU). In our application, we use Lex to handle the user text input so that we can extract the keywords for our information searching. We build four intents in our bot. One for attraction query, one for flight query, one for greeting message and one for thank you message. Fig. 10 shows our Lex structure.

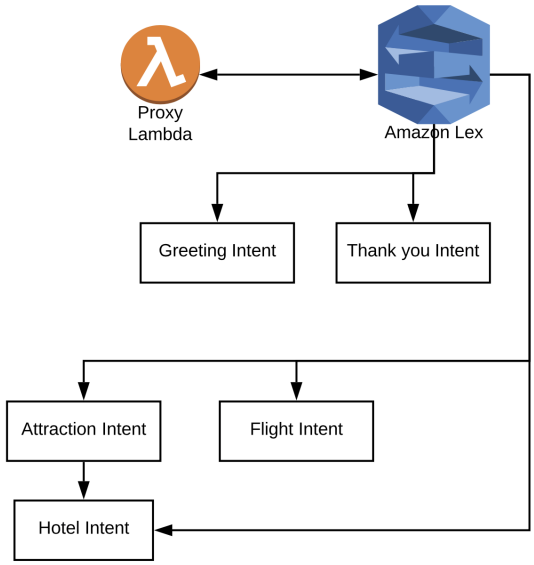


Fig. 10: Lex Structure

IV. Result

In this section, we illustrate the user interface and functions of our web application.

A. Home Page

Home page contains information about our team members and is designed to redirect the unauthorized user to the sign up or sign in page. Fig. 11 shows the home page of our application. It contains two buttons for the user to log in and visit the project's GitHub repository respectively.

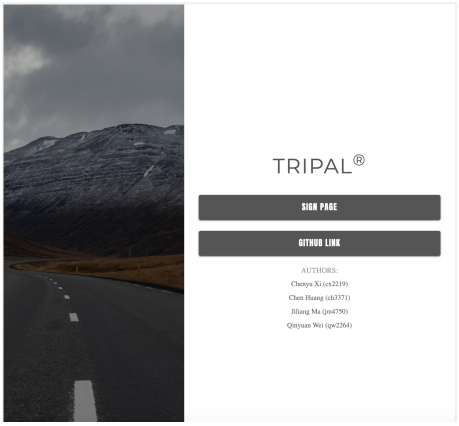
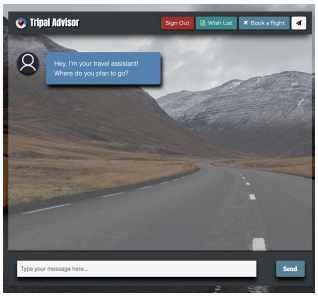


Fig. 11: Home Page

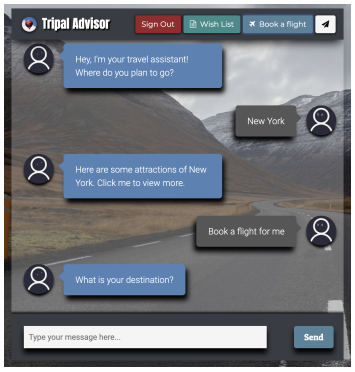
B. Chat Page

Fig. 12 shows the chat box page of our application. The AI assistant asks the user to type where they want to go and then gives the result of a list of attractions at that place based on the user's answer. The return message is displayed as a clickable button, and frontend will create a modal for attractions when the user clicks the button. Meanwhile, by clicking the Book a flight button, user can start a conversation with the chatbot to search flight tickets through our application.



(a) Chatbot: welcome

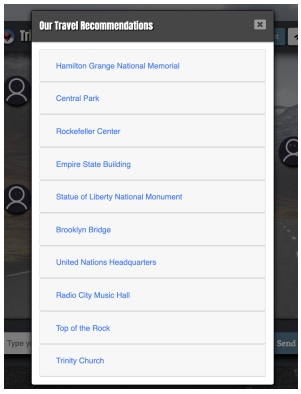
(b) Chatbot: attraction



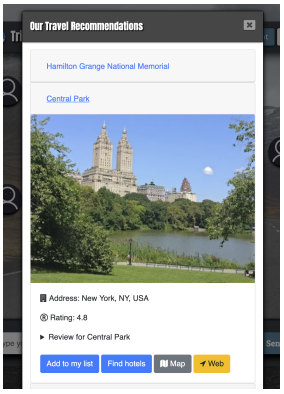
(c) Chatbot: book a flight

Fig. 12: Sample Conversation

1) Attractions Display: Once user clicks the attraction message, our application will open a modal window within which is the list of our attraction recommendations. As shown in Fig. 13, there is a list of attractions in New York within the pop-up recommendation window. We apply the accordion effect for this list. When the user clicks the name of the attraction, the attraction detail is displayed. The website button will redirect the user to the current attraction's official website.



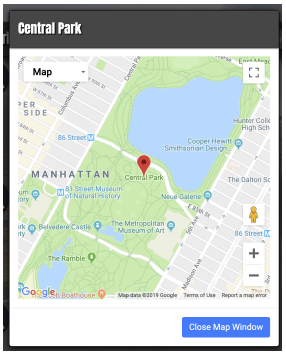
(a) Modal Window: attraction list



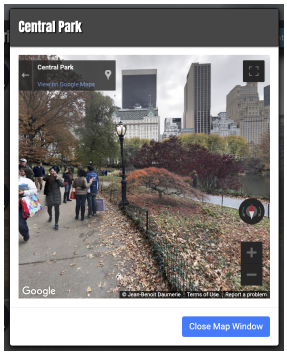
(b) Modal Window: attraction detail

Fig. 13: Sample Attraction Recommendation

2) Map View: If a user wants to find the location information about one attraction. They can easily view the nearby information through a Map window by clicking the map button on the attraction detail. The result for locating attraction on the map is shown as Fig. 14. This map window is interactable as we use Google Map API to create the map view.



(a) Google Map



(b) Google Street View

Fig. 14: Map View

3) Hotel Search: User can also query for nearby hotel information through clicking on the "Find hotels" button. The response message of our application contains a list of hotel names as well as their detail information, as shown in Fig. 15

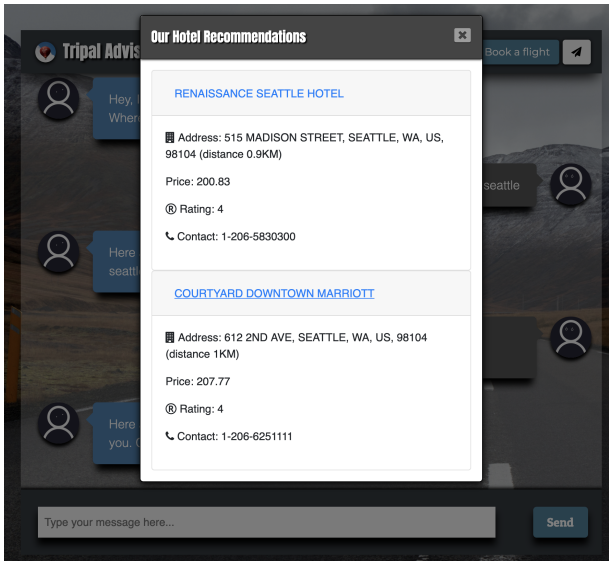


Fig. 15: Hotel Information

4) Create New Plan: User can add the attraction they are interested in the wishlist for later use from the attraction list. User can simply click the "Add to my list" button on the attraction detail page and then save their idea about the place (Fig. 16). User can access their wishlist page by clicking the "wishlist" button in the navigation bar of the chat page.

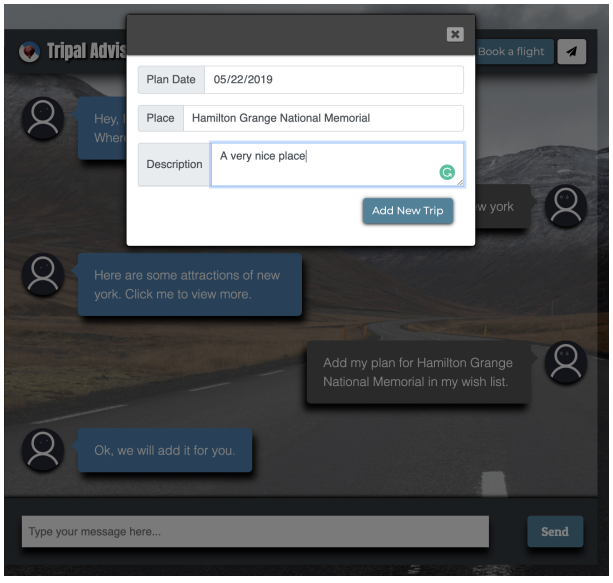


Fig. 16: Add Plan to Wishlist

5) Flight Search: After getting attractions and hotel information, the user can chat with our assistant to get flight information. The user specifies the destination, departure city, depart time and back time to get the flight information as Fig. 17. User can also go to kayak's website to book fights online.

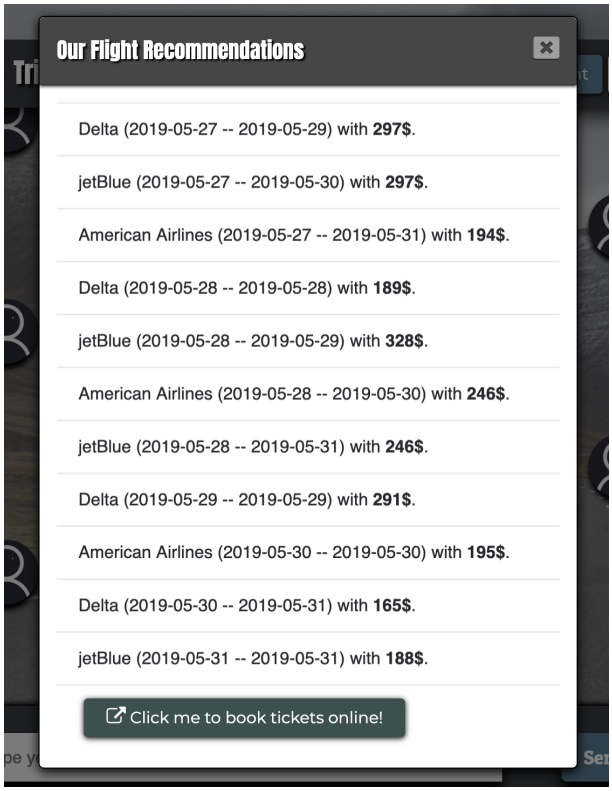


Fig. 17: Flight Information

C. Wishlist Page

The wishlist page is shown in Fig. 18. When the date of one plan in the wishlist is before today, the item in wishlist will be moved to "Trip History" automatically.

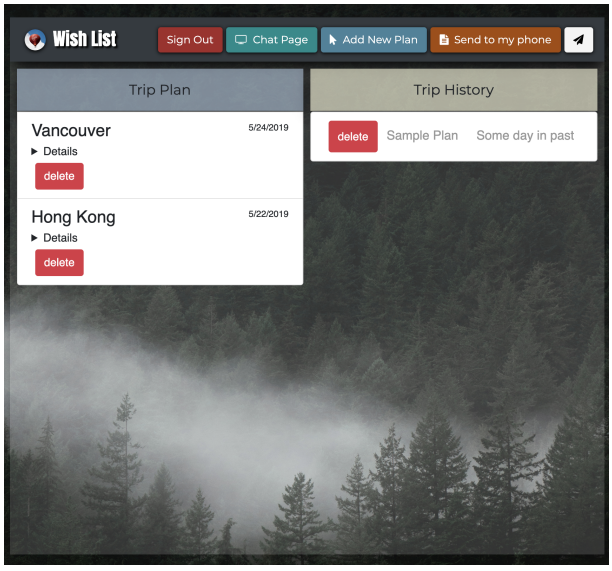


Fig. 18: Wishlist View

1) Add New Plan to Wishlist: User can add a new plan directly from wishlist by clicking "add new plan" button. The addition logic is the same as the "add to my list button" in the chat page.

2) Delete New Plan to Wishlist: We add a delete button for each plan in wishlist page to allow users to delete useless records from their wishlist. To prevent misoperation, the website will ask the user to double-check before executing the deletion. (Fig. 19)

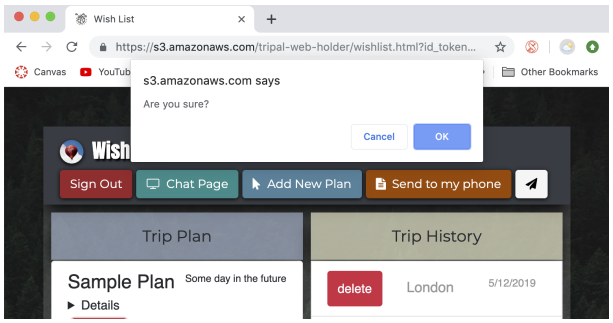


Fig. 19: Double Check Deletion

3) Send Wishlist SMS: By clicking "Send to my phone" button in the wishlist page, the user will receive an SMS on his cell phone containing all the information in the wishlist like Fig. 20

Fig. 20: wishlist SMS

V. Conclusion

In our work, we build an application which reduces the searching cost for our user when they are planning for a trip. Our application uses three APIs which are provided by Google, Skyscanner, and Amadeus to offer travel information (attractions, hotels and flights) to the users. Designing our application as a chatbox, we provide a friendly user interface and help our user to interact with it easily. We thoroughly consider the user needs for saving their idea about the trip and accessing their wishlist off-line. We use DynamoDB to store the wishlist information and SNS for a user to store their wishlist on their cell phone via SMS message.

We built our application on top of AWS, which offers cloud-based products including computing, storage, database and so on. With the help of AWS service like Cognito, Lambda, Lex, API Gateway, S3 and DynamoDB, our application is able to control user access as well as store and query information. We take advantage of using AWS to make our web application serverless, eliminating the complexity of server managements and increasing the scalability and availability of our application.

There are further functionalities we can add to our project. Firstly, we can train our Lex bot to process more dynamic user requests such as "Give me suggestions, I would like to visit beaches on the west coast". Secondly, it is more user-friendly if we can add days left notification function through SNS and SQS, for instance, we can remind users through SMS or SNS for their future travel plans next week. Last but not least, add adaption for mobile platforms such as iOS and Android would make our website more accessible for users on trips.

Acknowledgment

We would like to express our gratitude here to Prof. Sahu in the Computer Science Department of Columbia University, who provided essential suggestions and guidance for us.

References

[1] R.L. Adams, "Top Travel Websites For Planning Your Next Adventure," Forbes, Mar 29, 2016. [Online]. Available: https://www.forbes.com/sites/robertadams/2016/03/29/the-worlds-best-travel-websites/#50c6d6970a7f. [Accessed: May. 18, 2019].

[2] Anadea, "What is a Chatbot and How to Use It for Your Business," Medium, Jan 5, 2018. [Online]. Available:w https://medium.com/swlh/what-is-a-chatbot-and-how-to-use-it-for-your-business-976ec2e0a99f. [Accessed: May. 18, 2019].

[3] "Build a Serverless Web Application with AWS Lambda, Amazon API Gateway, Amazon S3, Amazon DynamoDB, and Amazon Cognito," Amazon aws, [Online]. Available: https://aws.amazon.com/getting-started/projects/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/?nc1=h_ls. [Accessed: May. 18, 2019].

[4] "Amazon Lex and AWS Lambda Blueprints," Amazon aws, [Online]. Available: https://docs.aws.amazon.com/Lex/latest/dg/Lex-lambda-blueprints.html. [Accessed: May. 18, 2019].

[5] "Building NoSQL Database Triggers with Amazon DynamoDB and AWS Lambda," Amazon aws, [Online]. Available: https://aws.amazon.com/blogs/compute/619/. [Accessed: May. 18, 2019].

[6] "Bootstrap," Bootstrap, [Online]. Available: https://getbootstrap.com/. [Accessed: May. 18, 2019].

[7] "AWS SDK for JavaScript in the Browser," Amazon aws, [Online]. Available: https://aws.amazon.com/sdk-for-browser/. [Accessed: May. 18, 2019].

[8] "Amazon Simple Notification Service," Amazon aws, [Online]. Available: https://aws.amazon.com/sns/. [Accessed: May. 18, 2019].

[9] "Amazon API Gateway," Amazon aws, [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: May. 18, 2019].

[10] "Amazon DynamoDB," Amazon aws, [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: May. 18, 2019].

[11] "Amazon Lex," Amazon aws, [Online]. Available: https://aws.amazon.com/Lex/. [Accessed: May. 18, 2019].

[12] "Amendeus hotel search API," Amendeus, [Online]. Available: https://developers.amadeus.com/self-service/category/hotel/api-doc/hotel-search. [Accessed: May. 18, 2019].

[13] "Google places API," Google, [Online]. Available: https://developers.google.com/places/web-service/intro. [Accessed: May. 18, 2019].

[14] "Skyscanner developers API Documentation," Skyscanner, [Online]. Available: https://skyscanner.github.io/slate/#api-documentation. [Accessed: May. 18, 2019].

APPENDIX

A. Links

GitHub: https://github.com/XiplusChenyu/Tripal
YouTube: https://www.youtube.com/watch?v=0OhH2FSh_ec

B. Backend

Tripal Proxy

```python
from __future__ import print_function  # Python 2/3
import json
import boto3
import os


def lambda_handler(event, context):
    # TODO implement
    text = event["messages"][0]['unstructured']['text']
    responseMessages = dict()
    responseMessages["Intent"] = "Null"

    # handle hotel request
    if "%&%" in text:
        # handle the book hotel request
        lat, lng = text.split("%&%lat:")[1].split(",", lng:")
        hotel_event = {
            "longitude": lng,
            "latitude": lat
        }
        lambda_client = boto3.client('lambda')
        lambda_response = lambda_client.invoke(
            FunctionName = "tripal_hotel",
            Payload = json.dumps(hotel_event)
        )
        lambda_response = lambda_response["Payload"].read()
        lambda_response = lambda_response.decode('utf-8')
        lambda_response = json.loads(lambda_response)

        # make response for hotel
        responseMessages['Intent'] = "Hotel"
        responseMessages['Hotel'] = lambda_response
        responseMessages['message'] = "Here are some hotel message for you. Click me to view more."

    # if not hotel query
    else:
        client = boto3.client("lex-runtime")
        response = client.post_text(
            botName=os.environ['BOT_NAME'],
            botAlias=os.environ['BOT_ALIAS'],
            userId=event["messages"][0]["unstructured"]['id'],
            inputText=text
        )

        fufillment = False

        # check if fullfilled by the value of slots
        if "slots" in response:
            fufillment = True
            for key, value in response['slots'].items():
                if value == None:
                    fufillment = False
                    break

        # handle greeting and thankyou intent
        if "intentName" in response:
            if response["intentName"] == "Greeting":
                responseMessages["message"] = response["message"]
            if response['intentName'] == "thankyou":
                responseMessages["message"] = response["message"]

        if fufillment:
            # handle the attraction request
            if response['intentName'] == "Attraction":
                attraction_event = {
                    "city": response['slots']["Location"]
                }
                lambda_client = boto3.client('lambda')
                lambda_response = lambda_client.invoke(
                    FunctionName = "tripal_attraction",
                    Payload = json.dumps(attraction_event)
                )
                lambda_response = json.loads(lambda_response["Payload"].read().decode('utf-8'))

                # get top ten attractions and search for attraction detail
                for attraction in lambda_response["attractions"][0:10]:
                    attraction_detail = {
                        "placeid": attraction["place_id"]
                    }

                    lambda_response2 = lambda_client.invoke(
                    FunctionName = "tripal_attraction_detail",
                    Payload = json.dumps(attraction_detail)
                    )
                    lambda_response2 = json.loads(lambda_response2["Payload"].read().decode('utf-8'))
                    print(lambda_response2)
                    lambda_response2 = lambda_response2['detail']
                    if "website" in lambda_response2:
                        attraction["website"] = lambda_response2["website"]
                    else:
                        attraction["website"] = "no"
                    if "typical_review" in lambda_response2:
                        attraction["typical_review"] = lambda_response2["typical_review"]
                    else:
                        attraction["typical_review"] = "no"

                # make response for attractions
                responseMessages["Intent"] = "Attraction"
                responseMessages['Attraction'] = lambda_response["attractions"][0:10]
                responseMessages['message'] = "Here are some attractions of " + response['slots']['Location'] + ". Click me to view more."


            # handle the flight request
            if response["intentName"] == "Flight":
                flight_event = {
                    "depart": response['slots']['Departure'],
                    "arrival": response['slots']['Destination'],
                    "depart_date": response['slots']['goTime'],
                    "back_date": response['slots']['backTime']
                }
                lambda_client = boto3.client('lambda')
                lambda_response = lambda_client.invoke(
                    FunctionName = "tripal_flight",
                    Payload = json.dumps(flight_event)
                )

                lambda_response = lambda_response["Payload"].read()
                lambda_response = lambda_response.decode('utf-8')
                lambda_response = json.loads(lambda_response)

                # make response for flight
                responseMessages['Intent'] = "Flight"
                responseMessages['Flight'] = lambda_response["flights"]
                responseMessages["message"] = response["message"] + "Click me to view more."

        # if not fullfill, simply pass the message
        else:

            responseMessages["Intent"] = "Null"
            responseMessages["message"] = response["message"]

    # return response
    return {
        'statusCode': 200,
        'body': responseMessages
    }
```

C. Frontend

User Identifier

```javascript
/**
 * This js is used for Cognito user auth and get user information
 */

const identity_pool_id = "us-east-1:4adb0b09-ad94-4423-ba16-a1a3750d8e45";
const aws_region = 'us-east-1';

current_user = {
    user_id: 'unk',
    user_name: '',
};

// these help user identifier to know which is the current page
wishPage = false;
chatPage = false;
userLogin = false;

try {
    id_token = location.toString().split('id_token=')[1].split('&access_token=')[0];
    access_token = location.toString().split('&access_token=')[1].split('&')[0];
} catch (e) {
    id_token = 'UNK';
    access_token= 'UNK';
}

AWS.config.region = aws_region; // Region

if (AWS.config.credentials) {
    AWS.config.credentials.clearCachedId();
}
// clear old one and create new credential

AWS.config.credentials = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: identity_pool_id,
    Logins: {
        'cognito-idp.us-east-1.amazonaws.com/us-east-1_l3NXswyeG': id_token,
    }
});

AWS.config.credentials.refresh((error) => {
    if (error) {
        console.error(error);
        alert("Your login is expired, please relogin to use our app");
        window.location.replace('https://s3.amazonaws.com/tripal-web-holder/index.html');
        console.log('Fail to log!');
    } else {
        console.log('Successfully logged!');
        let cognitoidentityserviceprovider = new AWS.CognitoIdentityServiceProvider();
        let params = {
            AccessToken: access_token /* required */
        };
        cognitoidentityserviceprovider.getUser(params, function(err, data) {
            if (err) {
                console.log(err, err.stack);
                alert("Your login is expired, please relogin to use our app");
                window.location.replace('https://s3.amazonaws.com/tripal-web-holder/index.html');
            }
```

```
            // an error occurred
            else {
                current_user.user_id = data['UserAttributes'][0]['Value'];
                current_user.user_name = `${data['UserAttributes'][2]['Value']} ${data['
                    UserAttributes'][3]['Value']}`;
                console.log(current_user);
                // Make the call to obtain credentials
                AWS.config.credentials.get(function() {

                    // Credentials will be available when this function is
                        called.
                    try {
                        apigClient = apigClientFactory.newClient({
                            accessKey: AWS.config.credentials.accessKeyId,
                            secretKey: AWS.config.credentials.secretAccessKey,
                            sessionToken: AWS.config.credentials.sessionToken
                        });
                        console.log('create apiClient');
                    }
                    catch(e) {
                        console.log('token exchange failed');
                    }
                    if (wishPage&&!chatPage){
                        getAllPlans(current_user.user_id);
                    }
                    userLogin = true;

                });
            }
            // successful response
        });


    }
});

function wishListRouter() {
    // use this function to redirect from chat page to wish page
    let redirectUrl = `https://s3.amazonaws.com/tripal-web-holder/wishlist.html?
        id_token=${id_token}&access_token=${access_token}&end`
    return window.location.replace(redirectUrl)
}

function chatRouter() {
    // use this function to redirect from chat page to wish page
    let redirectUrl = `https://s3.amazonaws.com/tripal-web-holder/chat.html?
        id_token=${id_token}&access_token=${access_token}&end`
    return window.location.replace(redirectUrl)
}

$(function () {
    // this is necessary for modal window pop-up
    $('[data-toggle="popover"]').popover()
});
```

API Message Adapter

```
/**
 * This js is used for create clickable message
 */
$empty = $('<h5>Ops, there is a desert ....</h5>'); // display this message
    when no value is returned from API

// this function convert user message to JSON format
function toBotRequest(message, userID){
    let date = new Date();
    return {
        messages: [
            {
                type: "string",
                unstructured: {
                    id: userID,
                    text: message.toString(),
                    timestamp: date.getTime().toString()
                }
            }
        ]
    };
}

// this function create a modal window to display hotel information
function attractHotel(response) {
    let type = response.body.Intent;
    let result_list = response.body[type];
    let return_message = response.body.message;
    let date = new Date();
    let model_id = 'hotel' + date.getTime();
    // create the modal window container
    let $fathermodel = $(`<div class="modal fade" id=${model_id} tabindex="-1" role="dialog
        " aria-hidden="true">
                        <div class="modal-dialog" role="document">
                            <div class="modal-content">
                                <div class="modal-header">
                                    <h5 class="modal-title">Our Hotel
                                        Recommendations</h5>
                                    <button type="button" class="close" data-dismiss=
                                        "modal" aria-label="Close" id="${
                                        model_id}Close">
                                        <span style="color: whitesmoke"><i class="
                                            fa fa-window-close" aria-hidden="
                                            true"></i></span>
                                    </button>
                                </div>
                                <div class="modal-body">
                                    <div class="accordion" id='${model_id}addOn'
                                        >
                                    </div>
                                </div>
                            </div>
                        </div>`);
    $('body').append($fathermodel);

    // if the result is empty, display a message
    if (0===result_list.length){
        $(`#${model_id}addOn`).append($empty);
    }

    // create accordion list items
    for (let i = 0; i < result_list.length; i++) {
        let currentHotel = result_list[i];
        let name = currentHotel.name;
        let price = currentHotel.price;
        let rating = currentHotel.rating;
        let hotel_distance = currentHotel.hotelDistance.distance + currentHotel.hotelDistance.
            distanceUnit;
        let hotel_address = `${currentHotel.address.lines[0]}, ${currentHotel.address.cityName}, ${
            currentHotel.address.stateCode}, ${currentHotel.address.countryCode}, ${
            currentHotel.address.postalCode}`;
        let contact = currentHotel.contact.phone;
        let kidAddOn = $(`
                            <div class="card">
                                <div class="card-header" id="heading${model_id + i}">
                                    <h2 class="mb-0">
                                        <button class="btn btn-link" type="button" data-
                                            toggle="collapse" data-target="#collapse${
                                            model_id + i}" aria-expanded="true" aria-
                                            controls="collapse${model_id + i}">
                                            ${name}
                                        </button>
                                    </h2>
                                </div>

                                <div id="collapse${model_id + i}" class="collapse" aria
                                    -labelledby="heading${model_id + i}" data-parent=
                                    "${model_id}addOn">
                                    <div class="card-body">
                                    <p><i class='fa fa-building'></i> Address: ${
                                        hotel_address} (distance ${hotel_distance})</p>
                                    <p><i class='fa fa-dollar-sign'></i>Price: ${price
                                        }</p>
                                    <p><i class='fa fa-registered'></i> Rating: ${
                                        rating}</p>
                                    <p><i class='fa fa-phone'></i> Contact: ${contact
                                        }</p>
                                </div>
                            </div>`);

        $(`#${model_id}addOn`).append(kidAddOn);
    }
    return {
        return_message: return_message,
        model_id: model_id,
        message_type: 1,
    }
}


// this function create a modal window to display Fight information, with
    same logic as above
function attractFlight(response){
    let type = response.body.Intent;
    let result_list = response.body[type];
    let return_message = response.body.message;
    let date = new Date();
    let model_id = 'flight' + date.getTime();
    let $fathermodel = $(`<div class="modal fade" id=${model_id} tabindex="-1" role="dialog
        " aria-hidden="true">
                        <div class="modal-dialog" role="document">
                            <div class="modal-content">
                                <div class="modal-header">
                                    <h5 class="modal-title">Our Flight
                                        Recommendations</h5>
                                    <button type="button" class="close" data-dismiss=
                                        "modal" aria-label="Close" id="${
                                        model_id}Close">
                                        <span style="color: whitesmoke"><i class="
                                            fa fa-window-close" aria-hidden="
                                            true"></i></span>
                                    </button>
                                </div>
                                <div class="modal-body">
                                    <ul class="list-group list-group-flush" id="
                                        ${model_id}List" style="max-height:
                                        600px; overflow: scroll">
                                    </ul>
                                </div>
                            </div>
                        </div>`);
    $('body').append($fathermodel);
    if (0===result_list.length){
        $(`#${model_id}addOn`).append($empty);
    }
    for (let i=0; i< result_list.length; i++) {
        let currentFlight = result_list[i];
        // create online store button
        if (currentFlight.kayak_link){
            let kidAddOn = $(`<li class="list-group-item">
                                <a href="${currentFlight.kayak_link}" target="_Blank"
                                    class="btn btn_top external_btn">
                                <i class="fa fa-external-link" aria-hidden="true" style="
                                    font-size: large"></i>
                                Click me to book tickets online!
                                </a></li>`);
            $(`#${model_id}List`).append(kidAddOn);
        }
        else
        {
            let carrier = currentFlight.carrier;
            let price = currentFlight.price;
            let departure_date = currentFlight.departure_date;
```

```
            let back_date = currentFlight.back_date;
            let kidAddOn = $(`<li class="list-group-item">${carrier} (${departure_date}
                -- ${back_date}) with <b>${price}$</b>.</li>`);
            $(`#${model_id}List`).append(kidAddOn);
        }
    }

    return {
        return_message: return_message,
        model_id: model_id,
        message_type: 1,
    }
}

// this is used for generate attraction recommendation window
function attractionDealer(response){
    let type = response.body.Intent;
    let result_list = response.body[type];
    let return_message = response.body.message;

    let date = new Date();
    let model_id = 'attaction' + date.getTime();
    let $fathermodel = $(`
                        <div class="modal fade" id=${model_id} tabindex="-1" role="
                            dialog" aria-hidden="true">
                        <div class="modal-dialog" role="document">
                            <div class="modal-content">
                                <div class="modal-header">
                                    <h5 class="modal-title">Our Travel
                                        Recommendations</h5>
                                    <button type="button" class="close" data-dismiss=
                                        "modal" aria-label="Close" id="${
                                        model_id}Close">
                                        <span style="color: whitesmoke"><i class="
                                            fa fa-window-close" aria-hidden="
                                            true"></i></span>
                                    </button>
                                </div>
                                <div class="modal-body">
                                    <div class="accordion" id='${model_id}addOn'
                                        >

                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>`);
    $('body').append($fathermodel);
    if (0===result_list.length){
        $(`#${model_id}addOn`).append($empty);
    }

    for (let i=0; i<result_list.length; i++){
        // create window for
        let currentAttraction = result_list[i];
        let photo = currentAttraction.photo_url;
        let website = currentAttraction.website;
        let review = currentAttraction.typical_review[0];
        let name = currentAttraction.name;
        let address = currentAttraction.formatted_address;
        let rating = currentAttraction.rating;
        let location_lat = currentAttraction.geometry.location.lat;
        let location_lng = currentAttraction.geometry.location.lng;
        let kidAddOn = $(`
                        <div class="card">
                            <div class="card-header" id="heading${model_id + i}">
                                <h2 class="mb-0">
                                    <button class="btn btn-link" type="button" data-
                                        toggle="collapse" data-target="#collapse${
                                        model_id + i}" aria-expanded="true" aria-
                                        controls="collapse${model_id + i}">
                                        ${name}
                                    </button>
                                </h2>
                            </div>

                            <div id="collapse${model_id + i}" class="collapse" aria
                                -labelledby="heading${model_id + i}" data-parent=
                                "${model_id}addOn">
                                <img class="card-img-top" src=${photo} alt="Card
                                    image cap">
                                <div class="card-body" id="body${model_id + i}">
                                <p><i class='fa fa-building'></i> Address: ${
                                    address}</p>
                                <p><i class='fa fa-registered'></i> Rating: ${
                                    rating}</p>
                                <details>
                                <summary>Review for ${name}</summary>
                                ${review}
                                </details>
                                                                    <br>
                                <button type="button" class="btn btn-primary"
                                 onclick="addNewTripinChatPage('${model_id}Close
                                    ','${name}')">Add to my list
                                 </button>

                                  <button type="button" class="btn btn-primary"
                                 onclick="findHotels('${name}', ${location_lat},
                                    ${location_lng},'${model_id}Close')">
                                    Find hotels
                                 </button>


                                 <button type="button" class="btn btn-secondary"
                                    onclick="nice(${location_lat},${
                                    location_lng},'${name}','${model_id}')"
                                    ><i class="fa fa-map"></i> Map</button>
                                </div>
                            </div>
                        </div>`);

        $(`#${model_id}addOn`).append(kidAddOn);
        if (website!='no'){
            let weblink = $(`<a href="${website}" target="_Blank" class="btn btn-warning
                ">><i class="fa fa-location-arrow"></i> Web</a>`);
            $(`#body${model_id + i}`).append(weblink);
        }
    }
    return {
        return_message: return_message,
        model_id: model_id,
        message_type: 1,
    }
}



// this function determines which dealer we use for current API response
function getBotResponse(response) {
    if(response.data.body.Intent==='Attraction')
    {
        return attractionDealer(response.data)
    }
    else if(response.data.body.Intent==='Flight'){
        return attractFlight(response.data)
    }
    else if(response.data.body.Intent==='Hotel'){
        return attractHotel(response.data)
    }
    else{
        // automatically adjust the model
        let message = response.data.body.message;
        if(message.toLowerCase().search("date") !== -1){
            $('.message_input').attr('type','date');
        }
        return{
            return_message: message,
            model_id: null,
            message_type: 0
        }
    }
}

// use this function to send request in backstage with location
    information
findHotels = function(name, location_lat, location_lng, model_id) {
    document.getElementById(model_id).click();
    $('.message_input').val(`Find Hotels around ${name}. %&%lat:${location_lat}, lng:${
        location_lng}`);
    console.log(`Find Hotels around ${name}. %&%lat:${location_lat}, lng:${location_lng}`);
    document.getElementById("send_message").click();
};


// this function is used for creating Google Map
function nice(lat, lng, title, model_id) {
    document.getElementById(model_id).click(); // close th modal window

    googleMap = {
        map: null,
        initialize: function (lat, lng) {
            let myOptions = {
                zoom: 15,
                center: new google.maps.LatLng(lat, lng),
                mapTypeControl: true,
                mapTypeControlOptions: {style: google.maps.MapTypeControlStyle.
                    DROPDOWN_MENU},
                navigationControl: true,
                mapTypeId: google.maps.MapTypeId.ROADMAP
            };
            this.map = new google.maps.Map(document.getElementById("map_canvas"),myOptions);
            let marker = new google.maps.Marker({
                position: new google.maps.LatLng(lat, lng),
                map: this.map,
                title: `${title}`
            });
        },
};

    let $body = $('body');
    let hidden_button = $(`<button class="btn btn-primary hide" id='${model_id}
        MapShown' data-toggle="modal" data-target="#google_maps_api${model_id}
        ">hidden</button>`)
    let mapWindow = $(`
                            <!--Use this model to show map-->

                            <div class="modal fade" id="google_maps_api${model_id}"
                                tabindex="-1" role="dialog"
                                aria-labelledby="mapModalLabel${model_id}" aria-hidden=
                                    "true">
                                <div class="modal-dialog">
                                    <div class="modal-content">
                                        <div class="modal-header">
                                            <h4 class="modal-title" id="mapModalLabel${
                                                model_id}">${title}</h4>
                                        </div>
                                        <div class="modal-body">
                                            <div id="map_canvas" style="width: 100%;
                                                height: 450px"></div>
                                        </div>
                                        <div class="modal-footer">
                                            <button type="button" class="btn btn-primary"
                                                data-dismiss="modal" aria-hidden="true"
                                                >Close Map Window</button>
                                        </div>
                                    </div>
                                </div>
                            </div>
                            `);

    $body.append(mapWindow); //create map on body
    $body.append(hidden_button);

    document.getElementById(`${model_id}MapShown`).click();
```

```
    $(`#google_maps_api${model_id}`).on("shown.bs.modal", function () {
        googleMap.initialize(lat, lng);
    }).on('hide.bs.modal', function () { //when close the model
        $(`#google_maps_api${model_id}`).remove();
        $(`#${model_id}MapShown`).remove();
    });
}
```

Implement Chat Interface

```
/**
 * This file is used for chat message display and API query
 */

const customer = 'right';
const robot = 'left';
chatPage = true;

function EnterPress(e){ // detect key down
    let keyDown = e || window.event;
    if(keyDown.keyCode === 13){
        document.getElementById("send_message").click();
    }
}

function bookFlight() { // invoke book a flight conversation
    $(".message_input").val('Book a flight for me');
    document.getElementById("send_message").click();
}


(function () {
    let Message;
    Message = function (arg) {
        this.text = arg.text;
        this.message_side = arg.message_side;
        this.message_type = arg.message_type;
        this.model_id = arg.model_id;
        this.draw = function (_this) {
            return function () {
                let $message;
                $message = $($('.message_template').clone().html());
                $message.addClass(_this.message_side).find('.text').html(_this.text.split('%&%')
                        [0]);
                $('.messages').append($message);
                // for destination info

                if(1 === _this.message_type){
                    $message.find('.text_btn').attr("data-toggle", "modal");
                    $message.find('.text_btn').attr("data-target", `#${_this.model_id}`);
                }
                return setTimeout(function () {
                    return $message.addClass('appeared');
                }, 0);
            };
        }(this);
        return this;
    };

    $(function () {
        let getMessageText, message_side, sendMessage, getReply;
        getMessageText = function () {
            let $message_input;
            $message_input = $('.message_input');
            return $message_input.val();
        };

        sendMessage = function (text, sender, model_id=null, m_type=null) {
            let $messages, message;
            if (text.trim() === '') {
                return;
            }
            $('.message_input').val('');
            $messages = $('.messages');
            message_side = sender;
            message = new Message({
                text: text,
                message_side: message_side,
                message_type: m_type,
                model_id: model_id,
            });
            message.draw();
            return $messages.animate(
                {
                    scrollTop: $messages.prop('scrollHeight')
                }, 300);
        };

        getReply = function(msg){
            if(current_user.user_id === 'unk'){
                sendMessage('You are illegal user! Please try to sign in first :)
                        ', robot);
                return
            }
            else if(msg.search('%&%wish') !== -1){
                sendMessage('Ok, we will add it for you.', robot);
                return
            }
            $('#waiter').removeClass('hide');
            let body = toBotRequest(msg, current_user.user_id);
            apigClient.chatbotPost({}, body, {}).then((res)=>{
                $('#waiter').addClass('hide');
                console.log(res);
                let data = getBotResponse(res);
                sendMessage(data.return_message, robot, data.model_id, data.message_type);
            }).catch((e)=>{
                // sendMessage(`Fail to get server response. Error ${e} `,
                        robot);
                sendMessage('Ops, we are unable to deal with that... Please retry
                        ', robot);
            });
        };

        $('.send_message').click(function () {
            let customer_message = getMessageText();
            $('.message_input').removeAttr('type');
            sendMessage(customer_message, customer);
            getReply(customer_message);
        });
        sendMessage(`Hey, I'm your travel assistant! Where do you plan to go?`,
                robot);
    });
}.call(this));
```

Implement Wishlist addition/deletion

```
/**
 * This js is used for plan addition, plan deletion, and send plan
 */
// current_user is defined in userDealer
function addNewTrip() {
    let place = $('#place').val();
    let idea = $('#place_idea').val();
    let date = new Date($('#date').val());

    let currentDate = new Date();
    let textid = current_user.user_id + currentDate.getTime();

    if (currentDate.getTime() > date.getTime()) {
        alert("The plan date is before today!");
        $("#closeAdd").trigger("click");
    }
    else if (date.toLocaleDateString() === "Invalid Date" || place === "") {
        alert("Miss required field");
        $("#closeAdd").trigger("click");

    }
    else {
        let doc = {
            "textId": textid,
            "peopleID": current_user.user_id,
            "place": place,
            "idea": idea,
            "timeStamp": currentDate.toLocaleDateString(),
            "planDate": date.toLocaleDateString()
        };
        console.log(doc);
        $("#closeAdd").trigger("click");
        $('#waiter').removeClass('hide');
        apigClient.postwishlistPost({}, doc, {}).then((res)=>{
            console.log(res);
            console.log('post a new plan');
            appendTrip(doc);
            $('#waiter').addClass('hide');
        }).catch((e)=>{
            console.log('fail to add a new plan');
            console.log(e);
        })

    }
}

// similar function while used in attraction card
function addNewTripinChatPage(id, name) {
    $('#exampleModalLabel').innerText = `Add new plan for ${name}`;
    $('#place').val(name);
    document.getElementById(id).click();
    document.getElementById('addPlan').click();
    $('.message_input').val(`Add my plan for ${name} in my wish list. %&%wish`);
    document.getElementById("send_message").click();
}

// this is used for plan deletion
function deletePlan(textId) {
    let r=confirm("Are you sure?");
    if(!r){
        return
    }
    console.log(textId);
    console.log(`delete id ${textId}`);
    // Todo: delete text from DB
    let para={
        textID: `${textId}`,
        peopleID: 'fake',
    };
    $('#waiter').removeClass('hide');
    apigClient.wishlistTextIDTextIDGet(para, {}, {}).then((res)=>{
        console.log(res);
        console.log('delete plan');
        $(`#${textId}`).addClass('hide');
        $('#waiter').addClass('hide');
    }).catch((e)=>{
        console.log('fail to delete new plan');
        console.log(e);
    })
}

// this function create list item,
function appendTrip(jsonResponse){
    let currentDate = new Date();
    let planDate = new Date(jsonResponse['planDate']);
    if (currentDate.getTime() > planDate.getTime()){
        // When there is not item
        $("#sampleHistory").addClass('hide');
        let $template = $(`
                <li class="list-group-item" id=${jsonResponse['textId']}>
                    <div class="d-flex w-100 justify-content-between">
```

```
                        <button onclick="deletePlan('${jsonResponse['textId']}')"
                                class="btn btn-danger">delete</button>
                        <p class="mb-1 historyIdeaTitle btn">${jsonResponse['place'
                            ]}</p>
                        <span class="historyIdeaTime btn"> <sup>${jsonResponse['
                            planDate']}</sup> </span>
                    </div>

                </li>`);
            $("#history").append($template);
        }
        else {
            $('#samplePlan').addClass('hide');
            let $template = $(`
                        <li class="list-group-item" id=${jsonResponse['textId']}>
                            <div class="d-flex w-100 justify-content-between">
                                <h5 class="mb-1 cardTitle ideaTitle ">${jsonResponse['
                                    place']}</h5>
                                <small class="ideaTime">${jsonResponse['planDate']}</small>
                            </div>
                            <details class="mb-1 ideaDisplay">
                            ${jsonResponse['idea']}
                            </details>
                            <button onclick="deletePlan('${jsonResponse['textId']}')"
                                class="btn btn-danger deletePlan">delete</button>
                        </li>`);
            $("#wishList").append($template);

        }
    }

    function getAllPlans(peopleID){
        // Once user login, display all records for current user
        console.log(`get plans for people ${peopleID}`);
        // todo: query appendTrip here to add all items in the page
        let para={
            peopleID: peopleID,
            textID: 'fake',
        };
        apigClient.wishlistPeopleIDPeopleIDGet(para, {}, {}).then((res)=>{
            console.log(res);
            let items = res['data'];
            for (let j = 0; j < items.length; j++){
                let current_res = items[j];
                appendTrip(current_res);
            }
            $('#waiter').addClass('hide');
            $('#waiter').removeAttr('style');


        }).catch((e)=>{
            console.log('failed to obtain people info');
            console.log(e);
        })
    }

    function sendMessage(){
        //This function ask user to type in their phone number and call SNS
            service
        let phone = prompt(`please enter you phone number, ${current_user.user_name}`);
        console.log(phone);
        if(phone){
            let para = {
                phone: phone,
                peopleID: current_user.user_id,
            };
            apigClient.snsPost({}, para, {}).then((res)=>{
                console.log('sent to phone');
            }).catch((e)=>{
                console.log(e);
            })
        }
    }

    wishPage = true;
```