COMS E6998 Final Project Report

Cloud Travel Planner

Chong Hu (ch3467), Wenjie Chen (wc2685), Xinyue Wang (xw2647). Yuanmeng Xia (yx2548)

Problem Statement

Planning can be extremely complex when multiple people are involved in one trip. We try to build a web application dealing with travel planning, including attraction recommendations, schedule arrangement, and collecting partners' opinions. By using this application, you can have a list of attractions without worrying about not having a choice. And you will get a detailed schedule arrangement with clear instructions. Last but not least, you can share your schedule with your partners allowing them to edit the schedule with you or just let them mark their choices to help you make the decision. You can even chat with them in a small chat room. It is a scale application based on AWS.

Overall Architecture

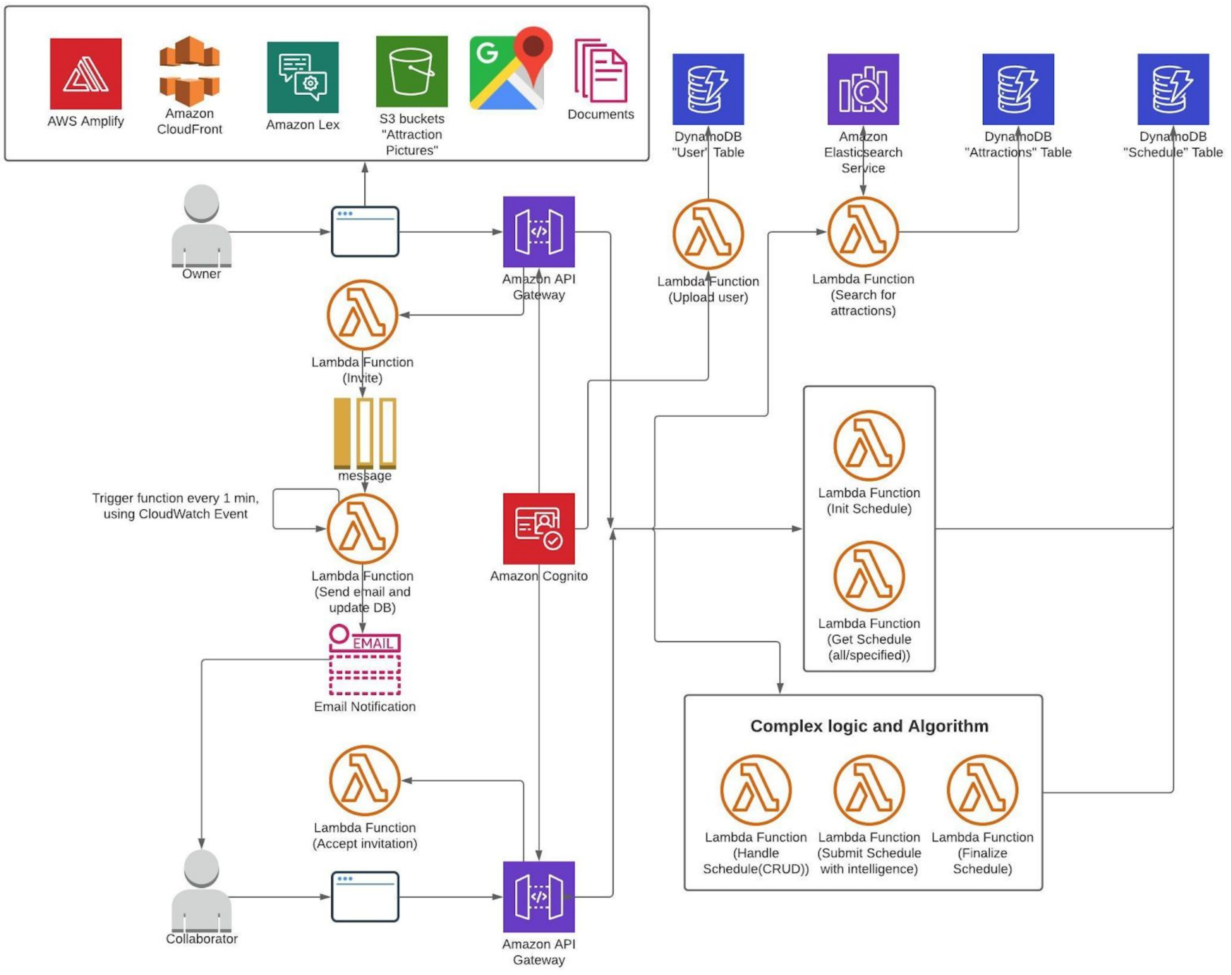

AWS Amplify
Amazon CloudFront
Amazon Lex
S3 buckets "Attraction Pictures"
Documents
DynamoDB "User" Table
Amazon Elasticsearch Service
DynamoDB "Attractions" Table
DynamoDB "Schedule" Table
Owner
Amazon API Gateway
Lambda Function (Upload user)
Lambda Function (Search for attractions)
Lambda Function (Invite)
message
Trigger function every 1 min, using CloudWatch Event
Lambda Function (Send email and update DB)
Amazon Cognito
Lambda Function (Init Schedule)
Lambda Function (Get Schedule (all/specified))
EMAIL
Email Notification
Complex logic and Algorithm
Lambda Function (Accept invitation)
Lambda Function (Handle Schedule(CRUD))
Lambda Function (Submit Schedule with intelligence)
Lambda Function (Finalize Schedule)
Collaborator
Amazon API Gateway

Figure 1. Overall Architecture part 1

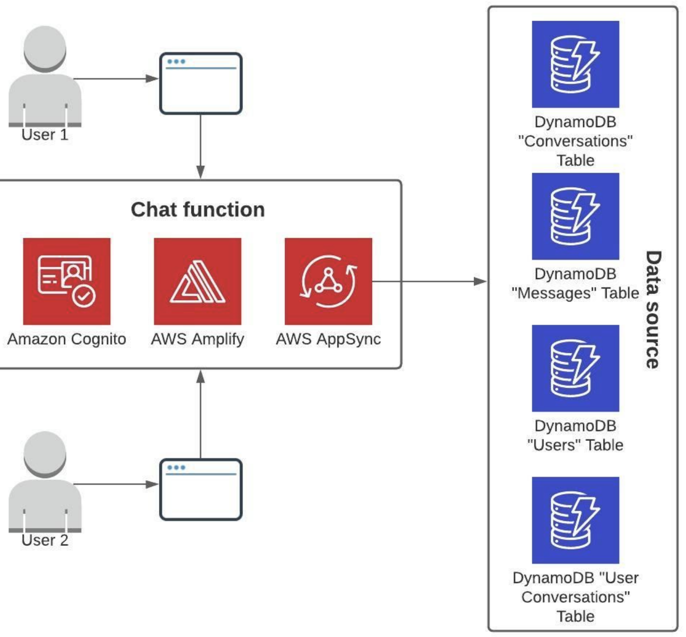


Figure 2. Overall Architecture part 2 (chat room function)

As a web application for travel planning, our functions fall into 3 main categories, namely user-related, schedule-related, and attraction-related. Our overall architecture is shown in figure 1. It primarily uses Amazon Web Service, which provides our application reliability and scalability. We use AWS Cognito to manage our user identities. A user can create a travel schedule through Lambda Function by adding various places of interest by searching in elastic search. Users can also invite his/her friends by email through AWS SQS and SES service. Other details will be introduced later.

The whole application is deployed via AWS Amplify, which is a set of tools to build scalable full-stack applications. The Amplify service will call CloudFormation to construct backend services and use CloudFront to deploy the front end web application. Amazon CloudFront is integrated into the application to ensure low latency data transfer.

The frontend directly interfaces with Amazon Lex, AppSync, S3 bucket, Google API. It interacts with API Gateway to call backend functions and get corresponding responses to show in the front. After users determine their schedule, the output pdf file representing the schedule can be downloaded through the frontend, which is denoted in figure 1 as documents, by calling the backend Lambda.

All backend logic is implemented in the AWS Lambda and other AWS cloud services. Lambda functions work with AWS services including API Gateway, Cognito, Elastic Search, DynamoDB, S3, SQS, and SNS. As mentioned above, API Gateway is how the frontend calls the backend. Amazon Cognito provides user authentication and identification for the whole application. In our backend service, ElasticSearch is responsible for searching for attractions. All data except images are stored in DynamoDB. S3 bucket is introduced to store image data. SNS is applied here to notify the user about the invitation. Before calling SNS to send an email, all requests will be pushed into an SQS. And then to allow the asynchronous processing, the trigger activated by CloudWatch Event is added to the function calling SNS.

Scenarios

For the primary editor or in our application case we call it schedule owner, the basic flow is like:

1. If the user already has an account, log in to the account; if not, sign in to an account and then log in.
2. After logging in, there are multiple functions the user can choose:
    a. To create a new schedule, a new preselect type schedule will be created with title and other related information.
    b. To continue the previous schedule, depending on the scheduling stage jumps to the corresponding step.
3. For the preselect type schedule, there are multiple functions the user can choose:
    a. To invite another editor, an email notification will be sent to that user. The invitation flow will be discussed later.
    b. To modify the schedule, the user can add/remove attractions.
    c. To discard the schedule, the user can delete the schedule.
    d. To mark the attraction as like/dislike, the user can do so in the attraction tab within the schedule.
    e. To move the schedule to the next stage, the user can submit the schedule. This will turn the flow to step 4. And during the transfer process, the application will automatically fill some attractions into the schedule according to the choice the user made when submitting.
4. For the editing type schedule, there are multiple functions the user can choose:
    a. To invite another editor, an email notification will be sent to that user. The invitation flow will be discussed later.
    b. To modify the schedule, the user can move the order of attractions.
    c. To discard the schedule, the user can delete the schedule.
    d. To move the schedule to the next stage, the user can finalize the schedule. This will turn the flow to step 5.
5. For the completed type schedule, there are multiple functions the user can choose:
    a. The user can view the webpage containing information about your schedule.
    b. The user can also download the pdf file of your schedule.
6. By clicking the chat room button on the side navigation bar, the user can join a private chat room.

For the co-editor who is invited to edit the schedule owned by another user, the basic flow is:

1. Click the invitation link. If the user already has an account, log in to the account; if not, sign in to an account and then log in. Then accept the invitation.
2. Depending on the scheduling stage jumps to the corresponding step.
3. For the preselect type schedule, there are multiple functions the co-editor can choose:
    a. To modify the schedule, the co-editor can add/remove attractions.
    b. To mark the attraction as like/dislike, the co-editor can do so in the attraction tab within the schedule.
    c. To move the schedule to the next stage, the co-editor needs to wait for the owner to submit the schedule.
4. For the editing type schedule, there are multiple functions the co-editor can choose:
    a. To modify the schedule, the co-editor can move the order of attractions.
    b. To move the schedule to the next stage, the co-editor needs to wait for the owner to finalize the schedule. This will turn the flow to step 5.
5. For the completed type schedule, there are multiple functions the co-editor can choose:
    a. Co-editor can view the webpage containing information about your schedule.
    b. Co-editor can also download the pdf file of your schedule.
6. Co-editor can join the private chat room by clicking the chat room button on the side navigation bar.

High-level details of implementation

In this section, we will first discuss the high-level implementation details of our backend structure. The most important part of our backend is how to design different Lambda functions and how to invoke other AWS cloud services. Then we give a brief introduction about other core components, our data schema in the storage, and how to organize backend APIs. After that, we will present the frontend implementation.

Frontend Implementation

Overall Page Design

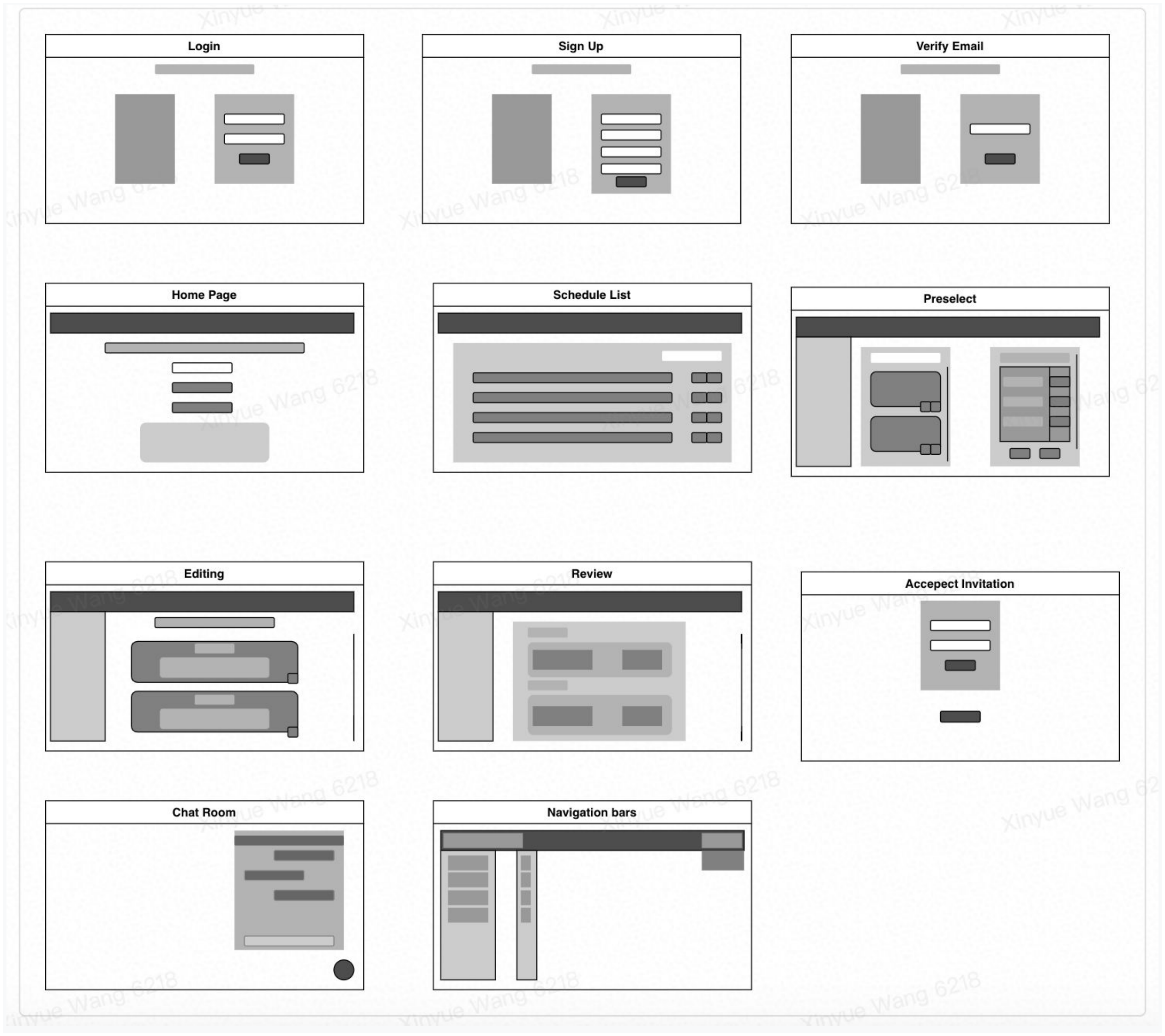


Figure 3. Overall Page Design

Details

We have 9 view pages with different components to complete this project.

- Login Page
  A registered user will be able to login into our website with his account.

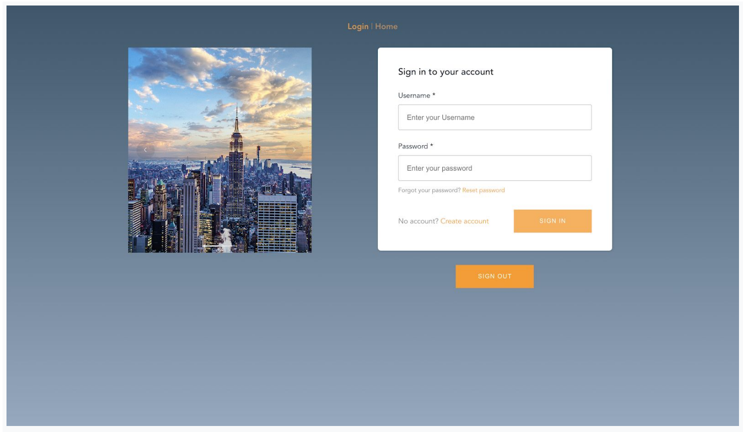


Figure 4. Login page

- Sign Up Page

  A new coming user will be able to create an account with his email address.

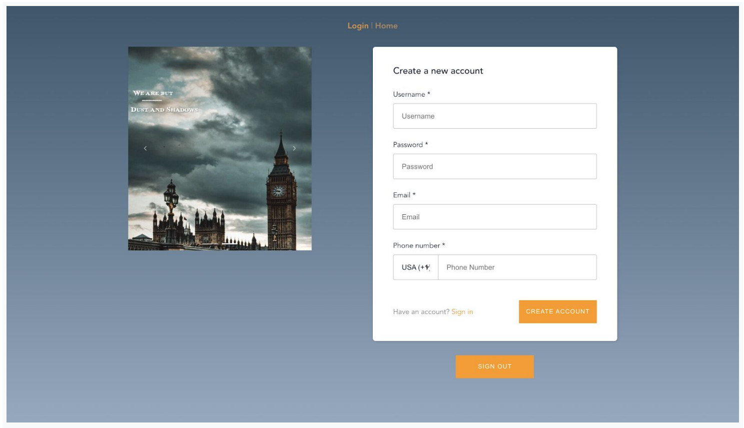
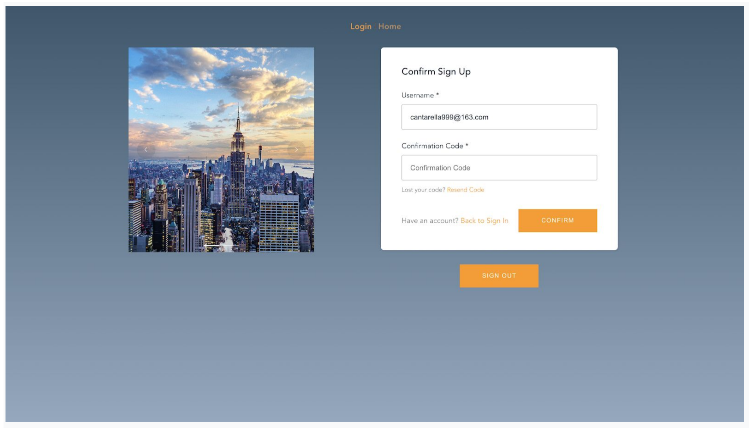


Figure 5. Sign up page

- Verify Email Page

  A new coming user will receive a verification code, and by entering the correct code, an account is successfully created. Then this user will be redirected to the login page to start his trip planning.



Username *: cantarella999@163.com

Figure 6. Verify email page

- Home Page
  On the home page, a user can either create a new trip plan by selecting the destination and clicking the button or talking with our bot. Also, by clicking on the “Continue” button, he can review all his schedules and take the next step.

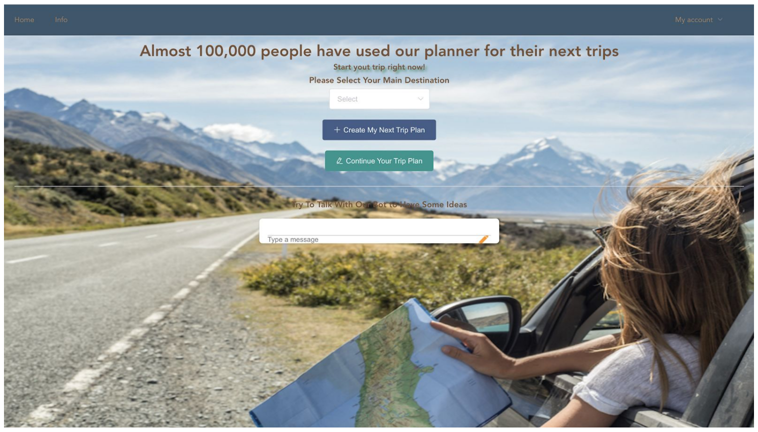
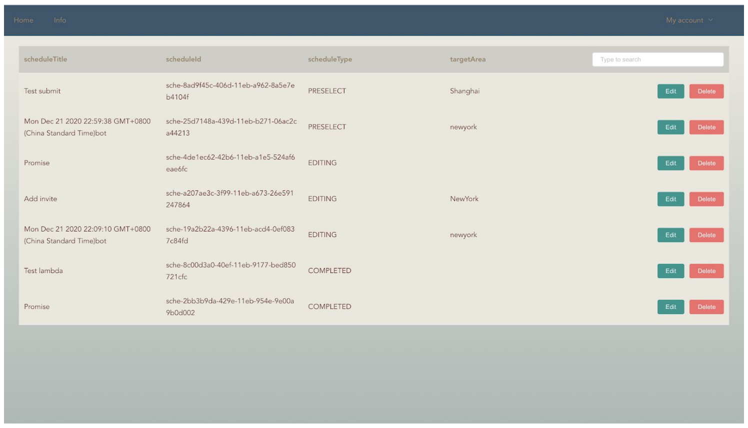

Home
Info
My account
Almost 100,000 people have used our planner for their next trips
Start yout trip right now!
Please Select Your Main Destination
Select
Create My Next Trip Plan
Continue Your Trip Plan
Try To Talk With Our Bot to Have Some Ideas
Type a message

Figure 7. Home page

- Schedule List Page
  On the schedule list page, a user can either delete one schedule or continue with one by clicking on edit.



scheduleTitle	scheduleId	scheduleType	targetArea	
Test submit	sche-8ad9f45c-406d-11eb-a962-8a5e7eb4104f	PRESELECT	Shanghai	Edit Delete
Mon Dec 21 2020 22:59:38 GMT+0800 (China Standard Time)bot	sche-25d7148a-439d-11eb-b271-06ac2ca44213	PRESELECT	newyork	Edit Delete
Promise	sche-4de1ec62-42b6-11eb-a1e5-524af6eae6fc	EDITING		Edit Delete
Add invite	sche-a207ae3c-3f99-11eb-a673-26e591247864	EDITING	NewYork	Edit Delete
Mon Dec 21 2020 22:09:10 GMT+0800 (China Standard Time)bot	sche-19a2b22a-4396-11eb-acd4-0ef0837c84fd	EDITING	newyork	Edit Delete
Test lambda	sche-8c00d3a0-40ef-11eb-9177-bed850721cfc	COMPLETED		Edit Delete
Promise	sche-2bb3b9da-429e-11eb-954e-9e00a9b0d002	COMPLETED		Edit Delete

Figure 8. Schedule list page

- Preselect Page
  In the Preselect Page, a user can add or delete attractions from his wish list. For the attractions presented in the left hand, a user can enter a keyword to search for specific attractions, like nature or iconic. To see more details, he can click the view button to see a popup chart with more information about this attraction.

  At the right-hand table, a user can add a like or cancel a like for each attraction selected. For the schedule owner, he can decide whether this attraction is good enough to visit by considering the like count and click the finish selection button. After that, our website will ask what trip mode the user likes to do a recommendation and prepare a schedule for him. For the user who has created the schedule with our chatbot, we stored his preference and he can just click the next step button to continue.

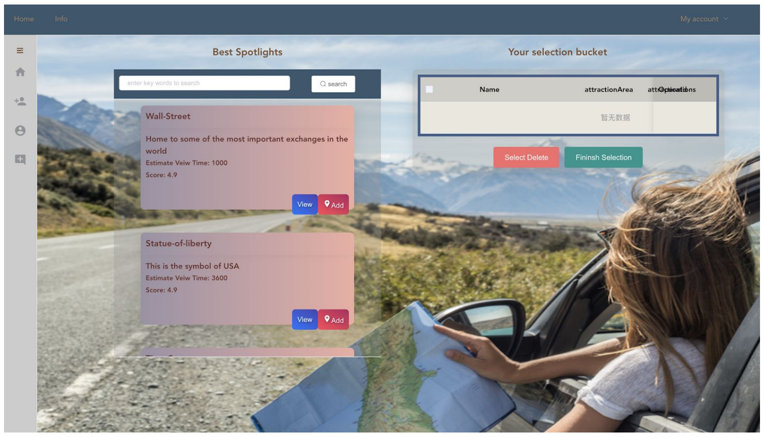


Figure 9. Preselect page

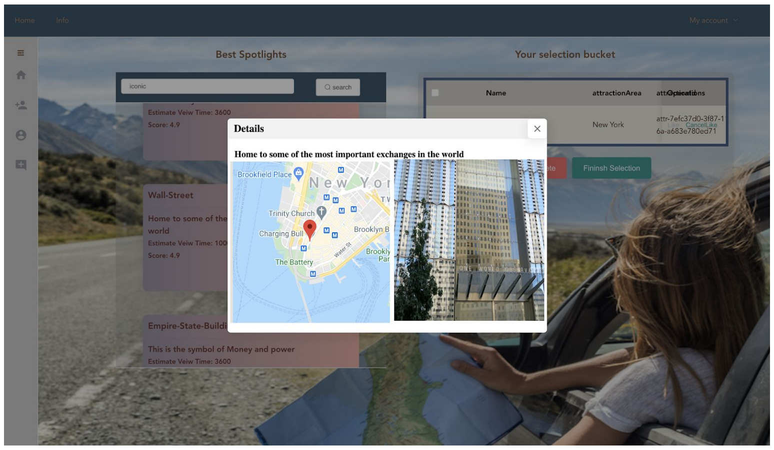
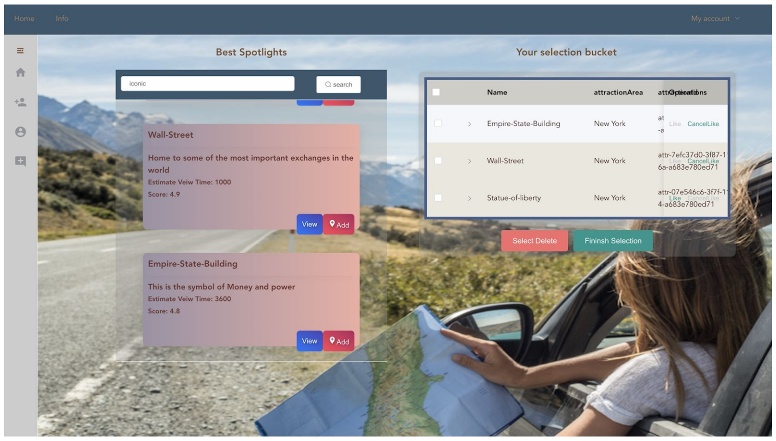
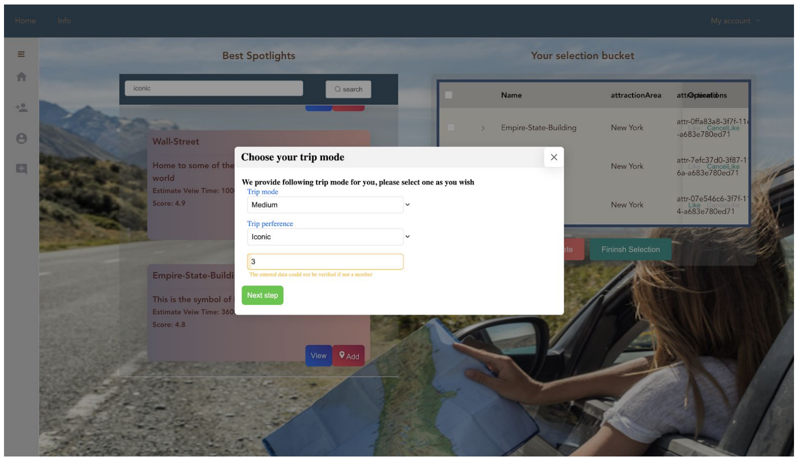


Figure 10. Preselect view detail page



Name	attractionArea	Operations
Empire-State-Building	New York	Like CancelLike
Wall-Street	New York	attr-7efc37d0-3f87-1 6a-a683e780ed71 Like CancelLike
Statue-of-liberty	New York	attr-07e546c6-3f7f-1 4-a683e780ed71 Like CancelLike

Figure 11. Preselect like count page



	Name	attractionArea	attrOperations
	Empire-State-Building	New York	attr-0ffa83a8-3f7f-11e -a683e780ed71
		New York	attr-7efc37d0-3f87-1 6a-a683e780ed71
		New York	attr-07e546c6-3f7f-1 4-a683e780ed71

Choose your trip mode

We provide following trip mode for you, please select one as you wish

Trip mode: Medium

Trip perference: Iconic

3

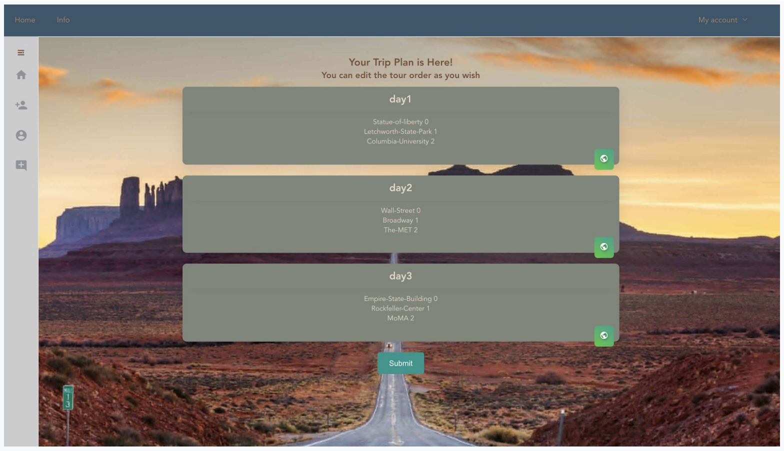
The entered data could not be verified if not a number

Next step

Figure 12. Preselect like finish selection page

- Schedule Edit Page

  On this page, a recommended trip schedule is displayed. Users can drag each item from one card into another card to modify the tour order. By clicking the green globe button, a map marked with all attractions on that day will be displayed. Click on the submit button, users will be redirected to the review page.



Your Trip Plan is Here!
You can edit the tour order as you wish

day1
Statue-of-liberty 0
Letchworth-State-Park 1
Columbia-University 2

day2
Wall-Street 0
Broadway 1
The-MET 2

day3
Empire-State-Building 0
Rockfeller-Center 1
MoMA 2

Figure 13. Schedule Edit page

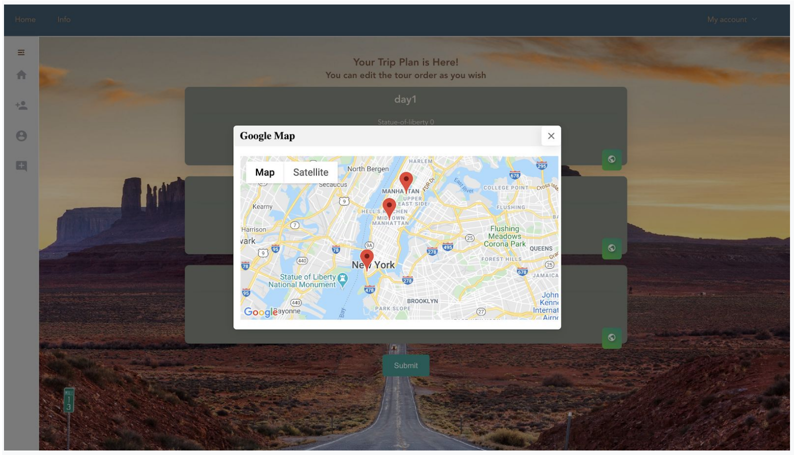


Figure 14. Schedule Edit view maps popup

- Schedule Review Page
  On this page, a final review of the trip schedule will be displayed. By clicking on the download button, a pdf generated by our server will be downloaded automatically.

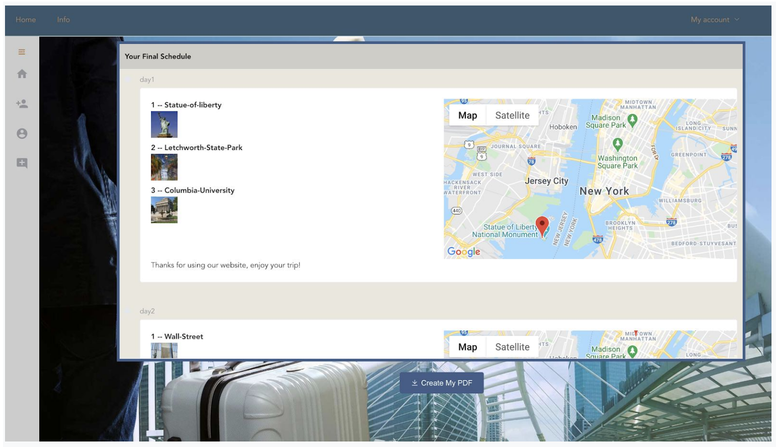


Figure 15. Schedule Review page

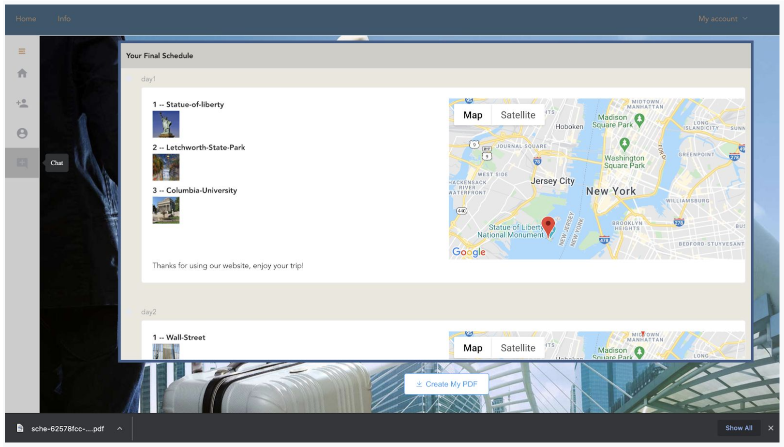
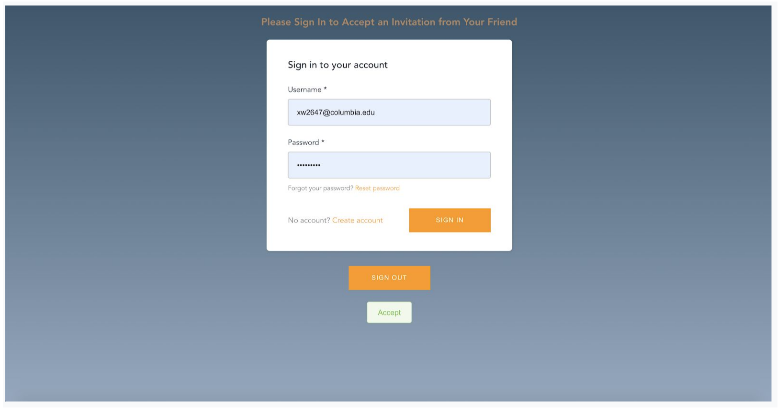


Figure 16. Schedule auto-download page

- Invitation Accept Page
  On this page, a user who received an invitation from the schedule owner can log in to his account and by clicking the accept button, a page that an owner wants his friends to see will be displayed.

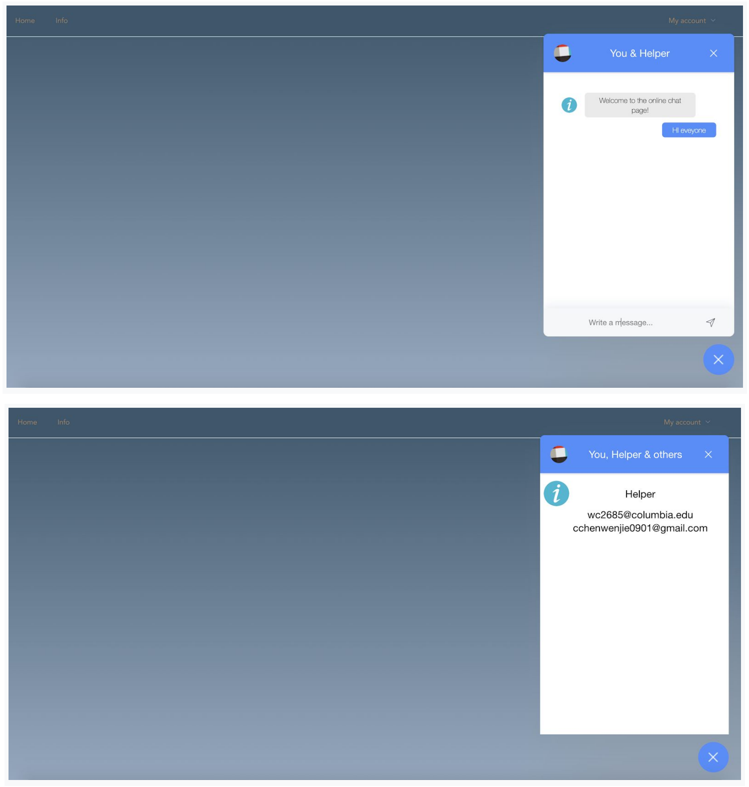

Please Sign In to Accept an Invitation from Your Friend
Sign in to your account
Username *
Password *
Forgot your password? Reset password
No account? Create account
SIGN IN
SIGN OUT
Accept

Username *: xw2647@columbia.edu

Figure 17. Invitation Accept page

- Online Chat Page

  By clicking the chat button on the side navigation bar, a new window will be opened, and users who have the access to this schedule can have an online chat here. Also, by clicking the title, a list of users who are in this schedule will be displayed.



Welcome to the online chat page!

Hi everyone

Helper
wc2685@columbia.edu
cchenwenjie0901@gmail.com

Figure 18. Private chat room page

- Navigation bars

  We designed two types of navigation bars to give our users a better experience.

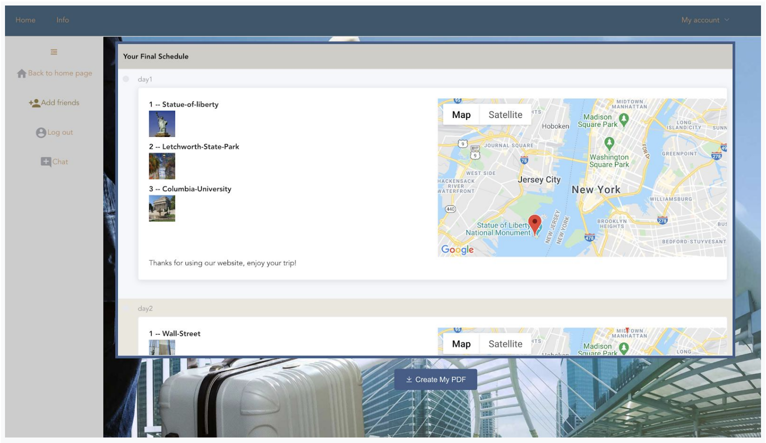


Figure 19. Overall Page Design

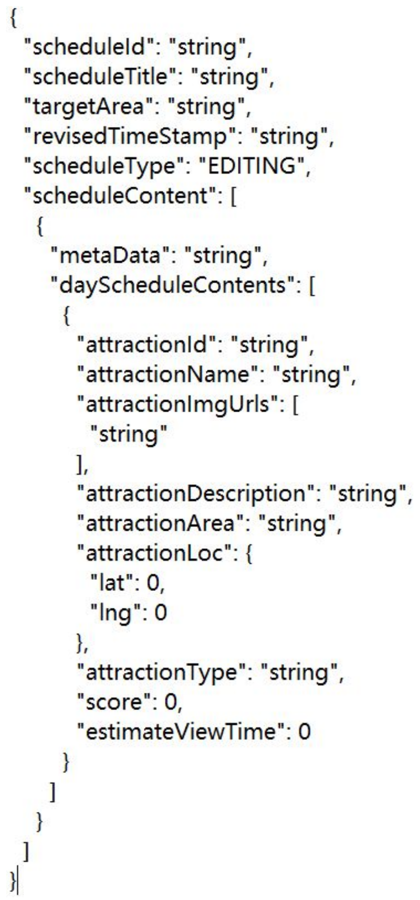
Backend Implementation

API Design

API Gateway

- GET /accept/{scheduleId}: accepts the invitation
    - scheduleId [string] (path parameter) required
    - editorId [string] (query parameter)
- GET /attraction/_search: searches the attraction according to query parameters
    - pageNo [string] (query parameter)
    - pageSize [string] (query parameter)
    - q [string] (query parameter)
- POST /chatbot: communicates between LEX and the frontend and creates PRESELECT schedule
- GET /invite/{userId}: sends invitation
    - userId [string] (path parameter) required
    - scheduleId [string] (query parameter)
- GET /schedule: retrieves schedule list by user id
    - pageNo [string] (query parameter)
    - pageSize [string] (query parameter)
- POST /schedule: creates a preselect type schedule
    - userId [string] (query)
    - targetArea [string] (query)
    - scheduleTitle [string] (query)
- GET /schedule/{scheduleId}: retrieves a schedule
    - scheduleId [string] (path) required

- POST /schedule/{scheduleId}: updates a schedule
  - scheduleId [string] (path) required
  - EditingSchedule [object] (body) required
    - Example:



```
{
  "scheduleId": "string",
  "scheduleTitle": "string",
  "targetArea": "string",
  "revisedTimeStamp": "string",
  "scheduleType": "EDITING",
  "scheduleContent": [
    {
      "metaData": "string",
      "dayScheduleContents": [
        {
          "attractionId": "string",
          "attractionName": "string",
          "attractionImgUrls": [
            "string"
          ],
          "attractionDescription": "string",
          "attractionArea": "string",
          "attractionLoc": {
            "lat": 0,
            "lng": 0
          },
          "attractionType": "string",
          "score": 0,
          "estimateViewTime": 0
        }
      ]
    }
  ]
}
```

- DELETE /schedule/{scheduleId}: deletes a schedule
  - scheduleId [string] (path) required
- PATCH /schedule/{scheduleId}: partially updates a schedule
  - scheduleId [string] (path) required
  - EditingSchedule [object] (body) required
- GET /schedule/{scheduleId}/attraction/{attractionId}: retrieves like/dislike information for a certain attraction in the schedule
  - scheduleId [string] (path) required
  - attractionId [string] (path) required
- POST /schedule/{scheduleId}/attraction/{attractionId}: updates like/dislike information for a certain attraction in the schedule
  - scheduleId [string] (path) required
  - attractionId [string] (path) required
- PUT /schedule/{scheduleId}/attraction/{attractionId}: generates initial like/dislike information for a certain attraction in the schedule
  - scheduleId [string] (path) required
  - attractionId [string] (path) required

- DELETE /schedule/{scheduleId}/attraction/{attractionId}: deletes an attraction in a schedule
    - scheduleId [string] (path) required
    - attractionId [string] (path) required
- GET /schedule/{scheduleId}/download: returns a pdf file containing schedule information
    - scheduleId [string] (path) required
- GET /schedule/{scheduleId}/finish: transfers a certain schedule from editing stage to completed stage
    - scheduleId [string] (path) required
- GET /schedule/{scheduleId}/submit: transfers a certain schedule from preselect stage to editing stage and pops some attractions to the schedule without user's preference
    - scheduleId [string] (path) required
- POST /schedule/{scheduleId}/submit: transfers a certain schedule from preselect stage to editing stage and pops some attractions to the schedule according to the user's preferences
    - scheduleId [string] (path) required
    - userId [string] (query)
    - schedule [object] (body)

Lambda functions:

- proj_attraction_searcher:
    - Endpoint path: /attraction/_search
    - Method: GET

  This Lambda function calls ElasticSearch querying with user's inputs and returns a list of attractions with their detailed information with specific page size and the page number. In ElasticSearch, we store attraction area, attraction name, attraction type, created timestamp, labels, object key. Some of the return information like the image URL requires the frontend to download and display. And the remaining part allows the frontend to show directly.

- proj_schedule_handler:
    - Endpoint path: /schedule/{scheduleId}
    - Method: DELETE, GET, PATCH, POST

  This Lambda function is responsible for deleting, retrieving, partially updating, and completely updating after creating a schedule which the function proj_schedule_initializer does. The function of this Lambda is to manipulate the DynamoDB table "scheduleTable".

- proj_schedule_attraction_selector:
    - Endpoint path: /schedule/{scheduleId}/attraction/{attractionId}
    - Method: DELETE, GET, POST, PUT

This Lambda function provides users with the ability to indicate whether they like or dislike the corresponding attractions in the preselect type schedule. The DELETE method is used to delete a certain attraction in a preselect type schedule. The GET method allows the user to access attraction vote information. The POST method is used to add/reduce like counts for a certain attraction. The PUT method is used for the first time attraction add. All users who have voted for a certain attraction will be recorded in DynamoDB.

- proj_invite_user:
  - Endpoint path: /invite/{userId}
  - Method: GET

  This Lambda function pushes messages needed to be further sent in an email into an SQS. The actual sending request will be handled in proj_send_invitation.

- proj_login_cognito2db:
  - Trigger after a user is confirmed in Cognito.

  This function will store user id, user name, and user email gotten from Cognito into DynamoDB table “userTable” and “usersTable”

- proj_send_invitation:
  - Trigger by Cloud Watch Event.

  This function will send schedule id, schedule owner name, and an invitation accept URL to the invited user.

- proj_schedule_initializer:
  - Endpoint path: /schedule
  - Method: GET, POST

  The GET method here will return a list of schedule according to the user id, page number, and page size. The POST method will create a preselect type schedule for the user and return the schedule information.

- proj_generate_pdf:
  - Endpoint path: /schedule/{scheduleId}/download
  - Method: GET

  This Lambda function will return a pdf file containing schedule information according to the schedule id given in the path parameter.

- proj_accept_invitation
  - Endpoint path: /accept/{scheduleId}
  - Method: GET

  This Lambda function will add the invited user id to the schedule editor id list.

- proj_schedule_submitter
  - Endpoint path: /schedule/{scheduleId}/submit

- Method: GET, POST

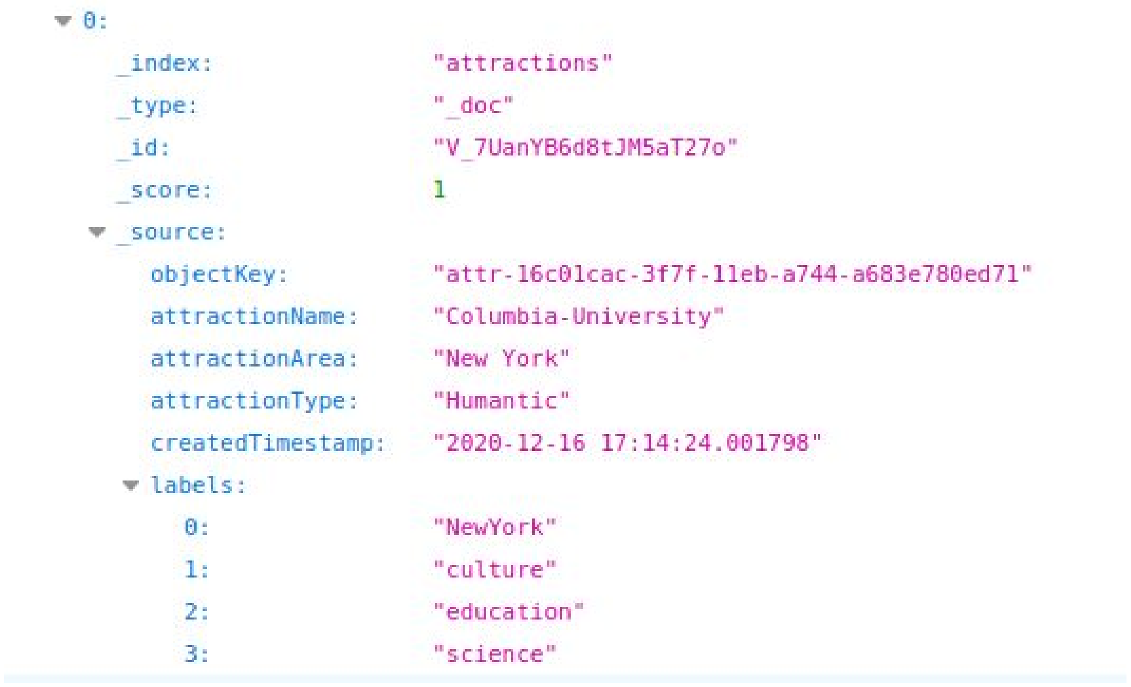
This function will transfer the schedule from the PRESELECT stage to the EDITING stage. And during the transfer process, the function will add attractions to the schedule based on the user's choice for travel mode (busy, medium, or relax), attraction preference, number of trip days, and the score of attractions. This step includes attraction/schedule recommendation, making the user plans the travel easily.

- proj_schedule_finalizer
  - Endpoint path: /schedule/{scheduleId}/finish
  - Method: GET

  This function will transfer the schedule from the EDITING stage to the COMPLETED stage.

ElasticSearch

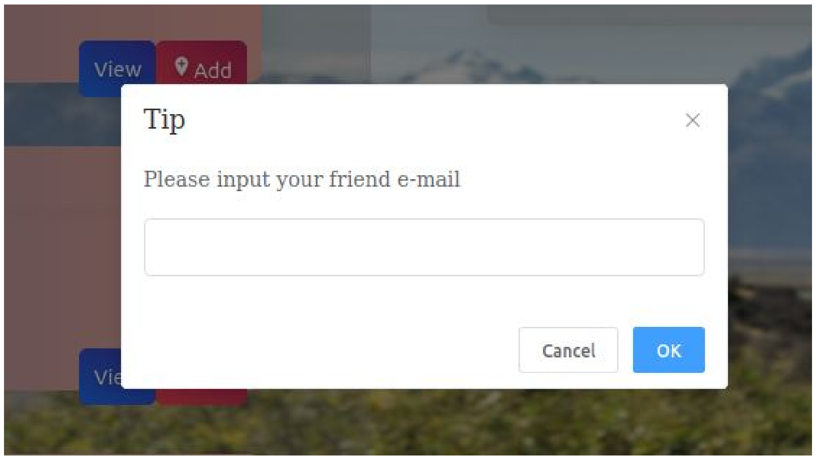
To help users better search attractions in our system, we indexed all attractions provided by our system with corresponding labels in our ElasticSearch service. Each attraction is identified by its unique id and the Lambda Function can acquire additional information from DynamoDB based on that id. Currently, we provide attraction names, attraction area, attraction type, and labels for users to search for different attractions. In ElasticSearch, the information is organized as follows.



```
▼ 0:
    _index:             "attractions"
    _type:              "_doc"
    _id:                "V_7UanYB6d8tJM5aT27o"
    _score:             1
  ▼ _source:
      objectKey:        "attr-16c01cac-3f7f-11eb-a744-a683e780ed71"
      attractionName:   "Columbia-University"
      attractionArea:   "New York"
      attractionType:   "Humantic"
      createdTimestamp: "2020-12-16 17:14:24.001798"
    ▼ labels:
        0:              "NewYork"
        1:              "culture"
        2:              "education"
        3:              "science"
```

Email Sending

SNS is applied here to notify the user about the invitation. Users can add their friends through email and all invitations will be handled by the Lambda function. Before calling SNS to send an email, all requests will be pushed into an SQS. And then to allow the asynchronous processing, the trigger activated by CloudWatch Event is added to the function calling SNS.



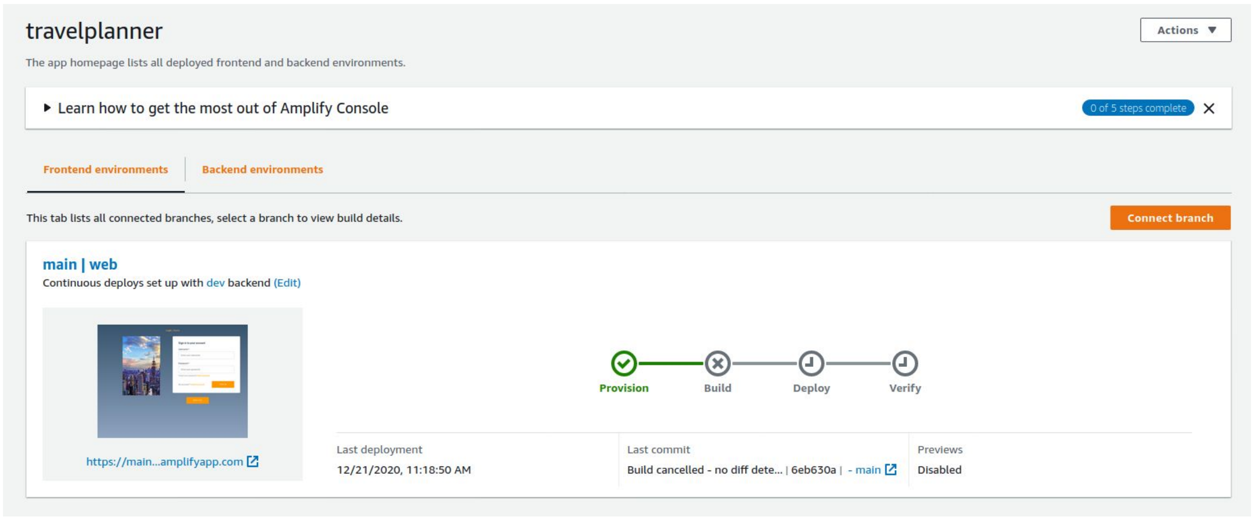
Tip

Please input your friend e-mail

Figure 20. Email Sending

Amplify

AWS Amplify is a set of tools to build scalable full-stack applications. The Amplify service will call CloudFormation to construct backend services and use CloudFront to deploy the front end web application. In our project, we use Amplify to manage our backend components, like Cognito(auth), Appsync(chat room), and Lex(chatbot). The Amplify service will also listen to our git repo. Each time we revise our code, the service will update the CloudFormation stack and re-deploy our frontend page if needed. We can manage our whole project, both frontend, and backend, in the Amplify console.



travelplanner

The app homepage lists all deployed frontend and backend environments.

Learn how to get the most out of Amplify Console

This tab lists all connected branches, select a branch to view build details.

main | web

Continuous deploys set up with dev backend (Edit)

https://main...amplifyapp.com

Last deployment	Last commit	Previews
12/21/2020, 11:18:50 AM	Build cancelled - no diff dete... | 6eb630a | - main	Disabled

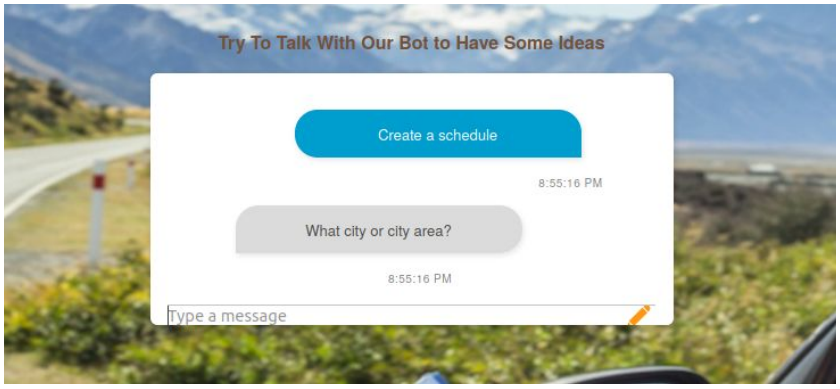
Figure 21. Amplify Console

Cognito

We use Cognito to manage our user identities. Users need to sign in / sign up to use our cloud travel planner. If and only If a user with the corresponding authentication, he/she can do the corresponding operation.

Lex Chatbot

After users sign in to our cloud travel planner, they can talk with the chatbot to create a new schedule. Our frontend page is connecting to Lex directly through Amplify. Amplify will configure the chatbot and let the front-end communicate with the backend chatbot. Our chatbot can help you to create a new schedule with only a few steps. The Lex Chatbot is integrated with frontend UI as follows.



Try To Talk With Our Bot to Have Some Ideas

Create a schedule

8:55:16 PM

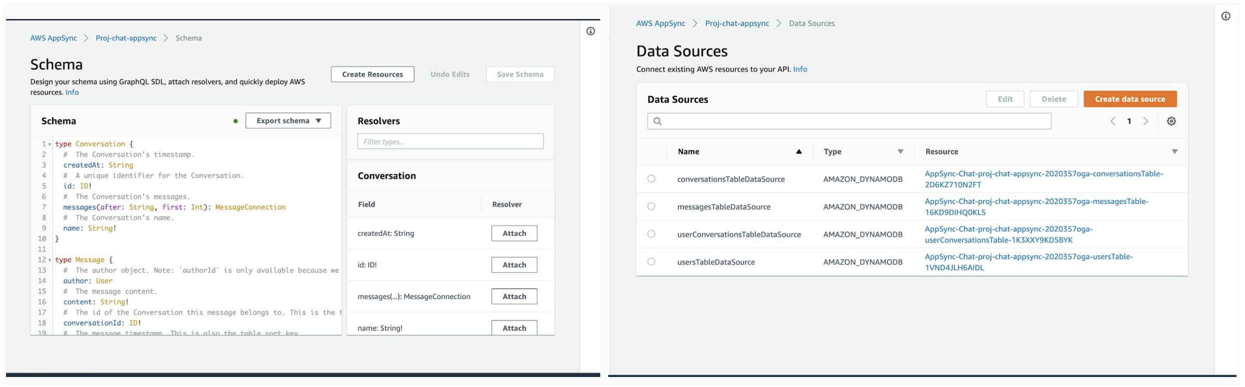
What city or city area?

8:55:16 PM

Figure 22. Chatbot

AppSync

We use AppSync to build multiple private chat rooms for our website. For each schedule, a private chat room is created, and when the friends of the owner accept the invitation, he will have access to the private chat room. With managed GraphQL subscriptions, AWS AppSync can push real-time data updates over Websockets to the users in the same chat room, hence an online chat is successfully implemented.



AWS AppSync > Proj-chat-appsync > Schema

Schema

Design your schema using GraphQL SDL, attach resolvers, and quickly deploy AWS resources. Info

Schema

```
type Conversation {
  #  The Conversation's timestamp.
  createdAt: String
  #  A unique identifier for the Conversation.
  id: ID!
  #  The Conversation's messages.
  messages(after: String, first: Int): MessageConnection
  #  The Conversation's name.
  name: String!
}

type Message {
  #  The author object. Note: `authorId` is only available because we
  author: User
  #  The message content.
  content: String!
  #  The id of the Conversation this message belongs to. This is the t
  conversationId: ID!
  #  The message timestamp. This is also the table sort key.
```

Resolvers

Conversation

Field	Resolver
createdAt: String	Attach
id: ID!	Attach
messages(...): MessageConnection	Attach
name: String!	Attach

AWS AppSync > Proj-chat-appsync > Data Sources

Data Sources

Connect existing AWS resources to your API. Info

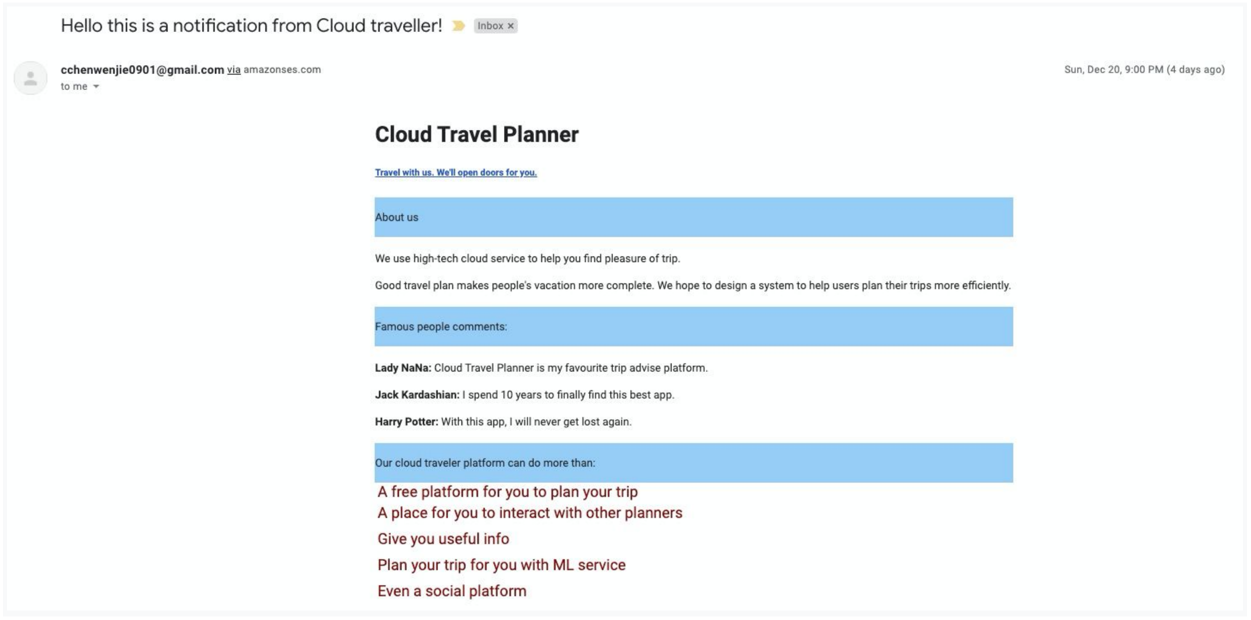
Data Sources

Name	Type	Resource
conversationsTableDataSource	AMAZON_DYNAMODB	AppSync-Chat-proj-chat-appsync-2020357oga-conversationsTable-2D6KZ710N2FT
messagesTableDataSource	AMAZON_DYNAMODB	AppSync-Chat-proj-chat-appsync-2020357oga-messagesTable-16KD9DIHQ0KLS
userConversationsTableDataSource	AMAZON_DYNAMODB	AppSync-Chat-proj-chat-appsync-2020357oga-userConversationsTable-1K3XXY9KD5BYK
usersTableDataSource	AMAZON_DYNAMODB	AppSync-Chat-proj-chat-appsync-2020357oga-usersTable-1VND4JLH6AIDL

Figure 23. AppSync Console

Pinpoint

To send advertisements and new updates to customers, we use Pinpoint for segmenting them into different groups such as user A like nature, user B like art, etc. sending emails to them weekly. Thus our platform connects with customers over channels. Amazon Pinpoint can grow with the app and scale globally to billions of messages per day across channels. This is a draft of our new update through email sending by Pinpoint.



Hello this is a notification from Cloud traveller! Inbox ×

cchenwenjie0901@gmail.com via amazonses.com
to me

Sun, Dec 20, 9:00 PM (4 days ago)

Cloud Travel Planner

Travel with us. We'll open doors for you.

About us

We use high-tech cloud service to help you find pleasure of trip.

Good travel plan makes people's vacation more complete. We hope to design a system to help users plan their trips more efficiently.

Famous people comments:

Lady NaNa: Cloud Travel Planner is my favourite trip advise platform.

Jack Kardashian: I spend 10 years to finally find this best app.

Harry Potter: With this app, I will never get lost again.

Our cloud traveler platform can do more than:

A free platform for you to plan your trip
A place for you to interact with other planners
Give you useful info
Plan your trip for you with ML service
Even a social platform

Figure 24. Overall Page Design

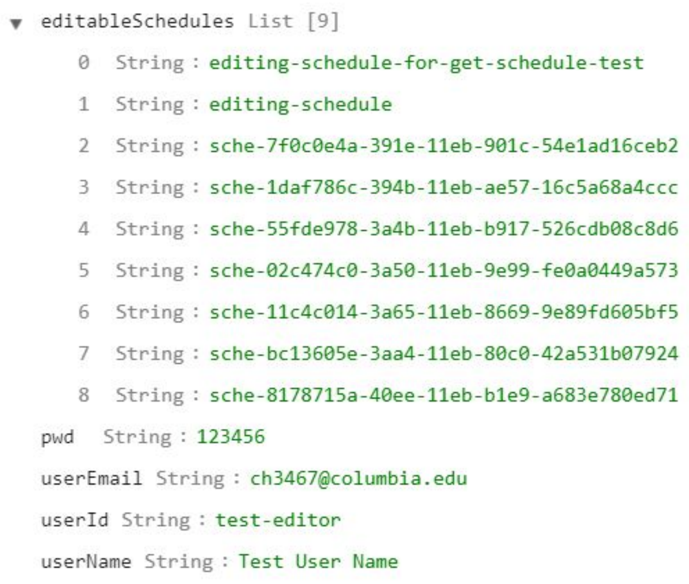
GoogleMap API Integration

To give users a better experience, GoogleMap API is utilized in this project. We first stored the latitude and longitude coordinates for each attraction in our DynamoDB, and when a user wants to see more information about one attraction, or his schedule, by clicking the button, our frontend service will call the GoogleMap API with according coordinates, and the more detail information including google map result will be displayed after getting a response from google.

Data schema

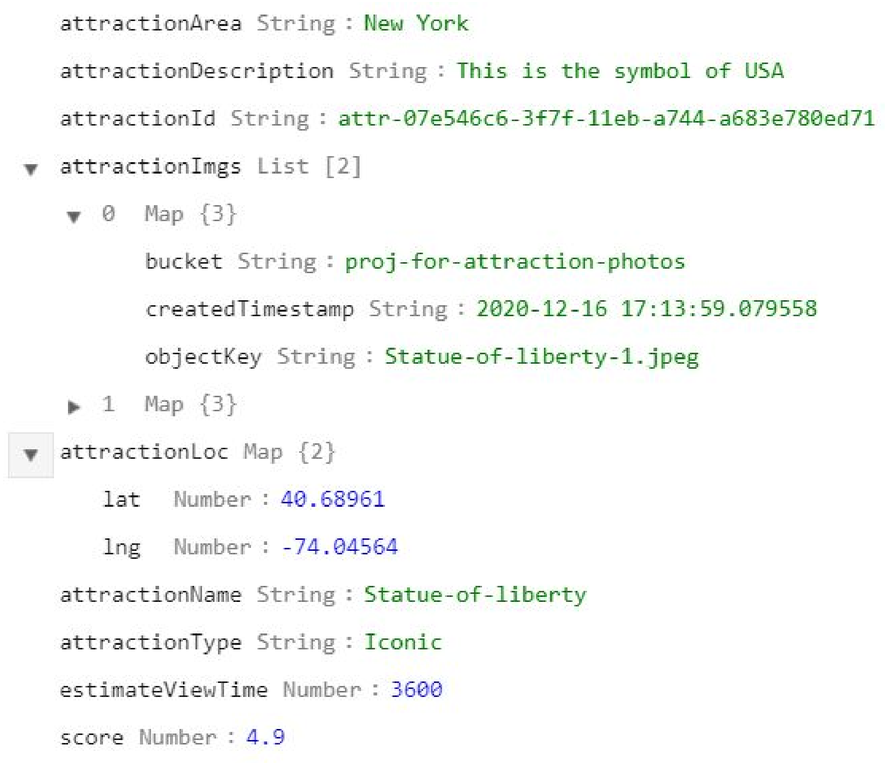
- DynamoDB: `userTable`
  - Content:
    - userId (column): [string] user id. uuid
    - userName (column): [string] user name

        - userEmail (column): [string] user email
        - pwd (column): [string] user password
        - editableSchedules (column): [list] editable schedule list
            - editableSchedule (list element): [string] schedule id
    - Example:



```
▼ editableSchedules List [9]
    0  String : editing-schedule-for-get-schedule-test
    1  String : editing-schedule
    2  String : sche-7f0c0e4a-391e-11eb-901c-54e1ad16ceb2
    3  String : sche-1daf786c-394b-11eb-ae57-16c5a68a4ccc
    4  String : sche-55fde978-3a4b-11eb-b917-526cdb08c8d6
    5  String : sche-02c474c0-3a50-11eb-9e99-fe0a0449a573
    6  String : sche-11c4c014-3a65-11eb-8669-9e89fd605bf5
    7  String : sche-bc13605e-3aa4-11eb-80c0-42a531b07924
    8  String : sche-8178715a-40ee-11eb-b1e9-a683e780ed71
  pwd  String : 123456
  userEmail String : ch3467@columbia.edu
  userId String : test-editor
  userName String : Test User Name
```

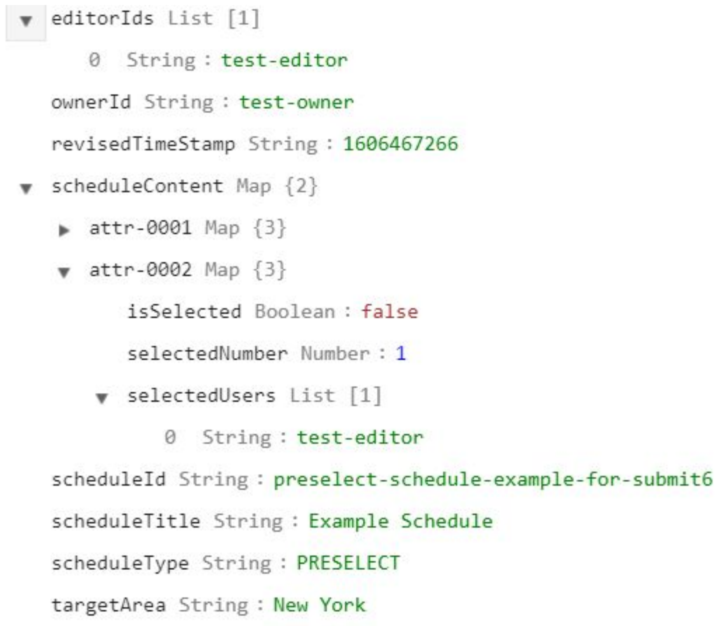
- DynamoDB: `attractionTable`
    - Content:
        - attractionId (column): [string] attraction id, uuid with prefix
        - attractionName (column): [string] attraction name
        - attractionDescription (column): [string] [not required] attraction description
        - attractionArea (column): [string] attraction location city
        - attractionLocation (column): [map] attraction in google map
            - lat (key): [number] latitude
            - lng (key): [number] longitude
        - attractionType (column): [String] attraction type
        - score (column): [number] [0-5] attraction score
        - estimateViewTime (column): [number (in second)] estimated view time
        - attractionImgs (column): [list] attraction image list
            - attractionImg (list element): [map]
                - bucket (key): [string] image bucket in S3.
                - createdTimestamp (key): [string (unix time)] created time '2020-11-12T12:40:02.001798'.
                - objectKey (key): [string] image object key in S3
    - Example:



```
  attractionArea String : New York
  attractionDescription String : This is the symbol of USA
  attractionId String : attr-07e546c6-3f7f-11eb-a744-a683e780ed71
▼ attractionImgs List [2]
  ▼ 0  Map {3}
       bucket String : proj-for-attraction-photos
       createdTimestamp String : 2020-12-16 17:13:59.079558
       objectKey String : Statue-of-liberty-1.jpeg
  ▶ 1  Map {3}
▼ attractionLoc Map {2}
     lat   Number : 40.68961
     lng   Number : -74.04564
  attractionName String : Statue-of-liberty
  attractionType String : Iconic
  estimateViewTime Number : 3600
  score Number : 4.9
```

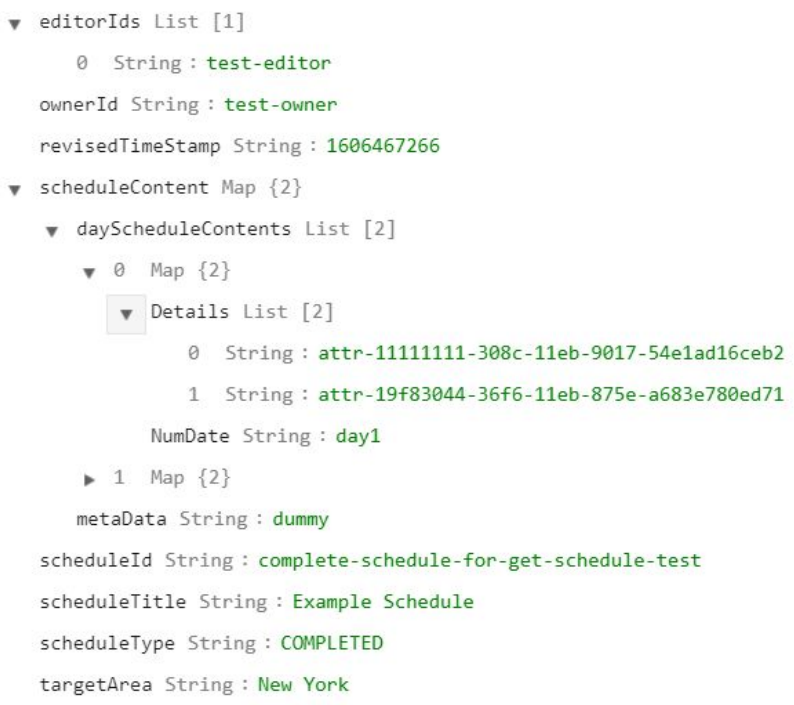
- DynamoDB `scheduleTable`
  - Content:
    - scheduleId (column): [string] attraction id, uuid with prefix
    - scheduleTitle (column): [string] [not required] schedule title shown in front-end. Use '<ownerName>-<ScheduleId>' by default for front-end.
    - revisedTimeStamp (column): [string (unix time integer)] last revised time
    - targetArea (column): [string] target location city.
    - ownerId (column): [string] owner user id
    - editorIds (column): [list] editor user ids
      - editorId (list element): [string] editor user id.
    - scheduleType (column): [string] schedule type for different stages.
      - PRESELECT is a stage that only contains user-interested attractions.
      - EDITING is a stage that the draft schedule is generated and all editors can edit the schedule.
      - COMPLETED is a stage that a schedule is generated and no one can edit the schedule.
    - scheduleContent (column): [list] schedule content. Differs from different schedule types.
      - For PRESELET schedule: [Map]
        - attractionId (Key): [Map] selected attraction id
          - isSelected (Key): [boolean] whether owner selects as must-visiting place

            - selectedNumber (Key): [number] how many user select this attraction
            - selectedUsers (Key): [list] users that select this attraction
                - selectedUser (list element): [string] user id
    - For EDITING/COMPLETED schedule: [Map]
        - daySchedulesContents(Key): [list]
            - dayScheduleContent(list element):[Map]
                - Details(Key): [list]
                    - attractionId(list element): [string] attraction id
                - NumDate(Key): [string] indicates the day number in format like “day1”
        - metaData(Key):[string] describe the data
- Example:
    - For PRESELECT schedule:



```
▼ editorIds List [1]
    0  String : test-editor
  ownerId String : test-owner
  revisedTimeStamp String : 1606467266
▼ scheduleContent Map {2}
  ▶ attr-0001 Map {3}
  ▼ attr-0002 Map {3}
      isSelected Boolean : false
      selectedNumber Number : 1
    ▼ selectedUsers List [1]
        0  String : test-editor
  scheduleId String : preselect-schedule-example-for-submit6
  scheduleTitle String : Example Schedule
  scheduleType String : PRESELECT
  targetArea String : New York
```

    - For EDITING/COMPLETED schedule:



```
▼ editorIds List [1]
     0  String : test-editor
  ownerId String : test-owner
  revisedTimeStamp String : 1606467266
▼ scheduleContent Map {2}
   ▼ dayScheduleContents List [2]
      ▼ 0  Map {2}
         ▼ Details List [2]
               0  String : attr-11111111-308c-11eb-9017-54e1ad16ceb2
               1  String : attr-19f83044-36f6-11eb-875e-a683e780ed71
            NumDate String : day1
      ▶ 1  Map {2}
     metaData String : dummy
  scheduleId String : complete-schedule-for-get-schedule-test
  scheduleTitle String : Example Schedule
  scheduleType String : COMPLETED
  targetArea String : New York
```

- Attraction Image Bucket
  - S3 bucket
    - ObjectKey: image file name
    - Object: image file

- Chat Room

  - DynamoDB “conversationsTable`
    - Content:
      - id (column): [string] scheduleId
      - name (column): [string] scheduleId
      - createdAt (column): [date] create date

  - DynamoDB “messagesTable`
    - Content:
      - conversationId (column): [string] scheduleId
      - createdAt (column): [date] create date
      - content (column): [string] message

  - DynamoDB “userConversationsTable`
    - Content:
      - userId (column): [string] userId
      - conversationId (column): [string] scheduleId

- DynamoDB “usersTable`
    - Content:
        - cognitoId (column): [string] cognitoId
        - id (column): [string] cognitoId
        - username (column): [string] username
        - registered (column): [boolean] true

Conclusion

In this project, we build a Cloud Travel Planner system based on AWS Cloud services. We use different AWS cloud services to manage, build, integrate, and deploy our project. Our project primarily uses Amazon Web Service, which provides our application reliability and scalability. We use AWS Cognito to manage our user identities. Amplify is applied in our project to manage our frontend and backend and deploy our project on the cloud. DynamoDB is used as data storage in the backend to store different types of information. Different Lambda Functions are used to provide different serverless functionalities. We use Lex to provide a chatbot for users to create a trip schedule conveniently and quickly. A chat room based on AppSync is integrated into our project so that users of one schedule can chat with each other. Beyond that, many other services are applied in our project to provide more functionalities and more features. It’s a good opportunity to practice how to utilize different functionalities on the cloud.