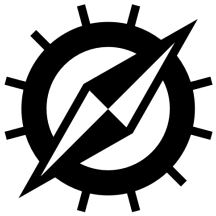


HUMSPOTTERS

CS 458 - SOFTWARE ENGINEERING - CAL POLY HUMBOLDT - FALL 2023

HUMSPOT APPLICATION - PROGRAMMER'S MANUAL

Document No: ST1-0005 Rev B

Date: 15 Feb 2024

Prepared by:
David Yaranon
Nathan Peralta
Sean Ross
Danny Nguyen

Revision History

Revision	Description	Author	Verifier	Date
A	Initial Release	D. Yaranon, S. Ross	Team	7 Dec 2023
B	Updating links	D. Yaranon	D. Yaranon	15 Feb 2024

Table of Contents

1. Scope

1.1. Purpose

This Software Requirements Specification (SRS) applies to the Humspot Application made for Cal Poly Humboldt's CS 458 Software Engineering class, term Fall 2023. The SRS defines relevant program code and functionalities along with detailed instructions to aid a future developer in understanding, improving upon, and/or implementing new features to Humspot.

1.2. Application Overview

Humspot was created to answer the question "What is there to do in Humboldt?" The app is aimed mostly at new Cal Poly Humboldt students, but applies to anyone who lives in the area. The application aims to deliver the latest events and attractions in Humboldt county in an easy to understand interface. Users of the application should find it useful for discovering new places to go and things to do.

1.3. Document Overview

- **Section 1 -** Defines the scope of the document.
- **Section 2 -** Explains how developers can get started running and making code changes to the application.
- **Section 3 -** Defines how the application was built on both the frontend and backend. Relevant files and functions for each section are given and their purpose explained.

2. Getting Started

2.1. Downloading the Source Code

The source code for Humspot is available at https://github.com/Humspot/Humspot. Clone the repository into your desired workspace using the code below.

```
git clone https://github.com/Humspot/Humspot.git
git checkout -b <new-branch-name> # more info in section 2.4
cd Humspot/Humspot
```

2.2. Running the Application in a Development Environment

Before running the application, make sure to have Node.js installed on your computer. (https://nodejs.org/en) (https://nodejs.org/en/learn/getting-started/how-to-install-nodejs).

Next, reach out to our developer team at dev@humspotapp.com for two required files. Note that these files are not included in the git repo as they contain sensitive information and private keys.

File 1: aws-exports.js; place this file in the src folder

File 2: .env; place this file in the secondary Humspot folder (Humspot/Humspot)

Lastly, run the following commands in your terminal.

```
npm install
npm run dev
```

Your application should be running at https://localhost:5173/

2.3. Building for Production

In this context, building for production means compiling and loading the app onto a native iOS/Android phone. See Ionic's docs on (Deploying to iOS and Android)

2.3.1. iOS

In order to build for iOS, a Mac computer is required along with the XCode application (https://developer.apple.com/xcode/). To build, use the following commands.

```
npm install
npm build
npx cap sync ios
npx cap open ios
```

This should open the XCode application, where you can select your device and press the play button to run the application.

NOTE: Visit (https://capacitorjs.com/docs/ios) for more information.

2.3.2. Android

In order to build for Android, the Android Studio application is required (https://developer.android.com/studio). To build, use the following commands.

```
npm install
npm build
npx cap sync android
npx cap open android
```

This should open the Android Studio application, where you can select your device and press the play button to run the application.

NOTE: Visit (https://capacitorjs.com/docs/android) for more information.

2.4. Version Control Procedures

It is recommended to work on your own branch whenever making new changes to Humspot. The following command will create a branch separate from the "main" branch, allowing you to make changes and not alter the main production code of Humspot.

```
git checkout -b <new-branch-name> # give your branch a new name
```

After updating, adding, or deleting code, you can push your changes to the GitHub servers using the following commands.

```
git add .
git commit -m "Commit message specifying what was changed"
git push -u origin new-branch-name # use the name you chose above
```

Do not, under any circumstances, push to the *main* branch without prior approval from the developer team!

3. Application Architecture

3.1. Frontend

Humspot was built with an Ionic/React frontend using Typescript. The app is ported to native iOS/Android apps using CapacitorJS.

Ionic - https://ionicframework.com/docs + https://ionicframework.com/docs/react

React - https://react.dev/

Capacitor - https://capacitorjs.com/

Typescript - https://www.typescriptlang.org/

The frontend is currently only built for mobile, so when viewing in the browser it is best to use the dev tools to view it in a simulated mobile environment. Read the docs linked above for more information on how Humspot was built.

3.1.1 Frontend Files

The frontend files are located in the **src** directory (with the exception of **App.tsx** and **App.css**. Find below the main application pages along with the components used for each page.

App.tsx

Main entry point for the application. Contains page routes and application components. When creating a new page for the application, make sure to add the route to this file.

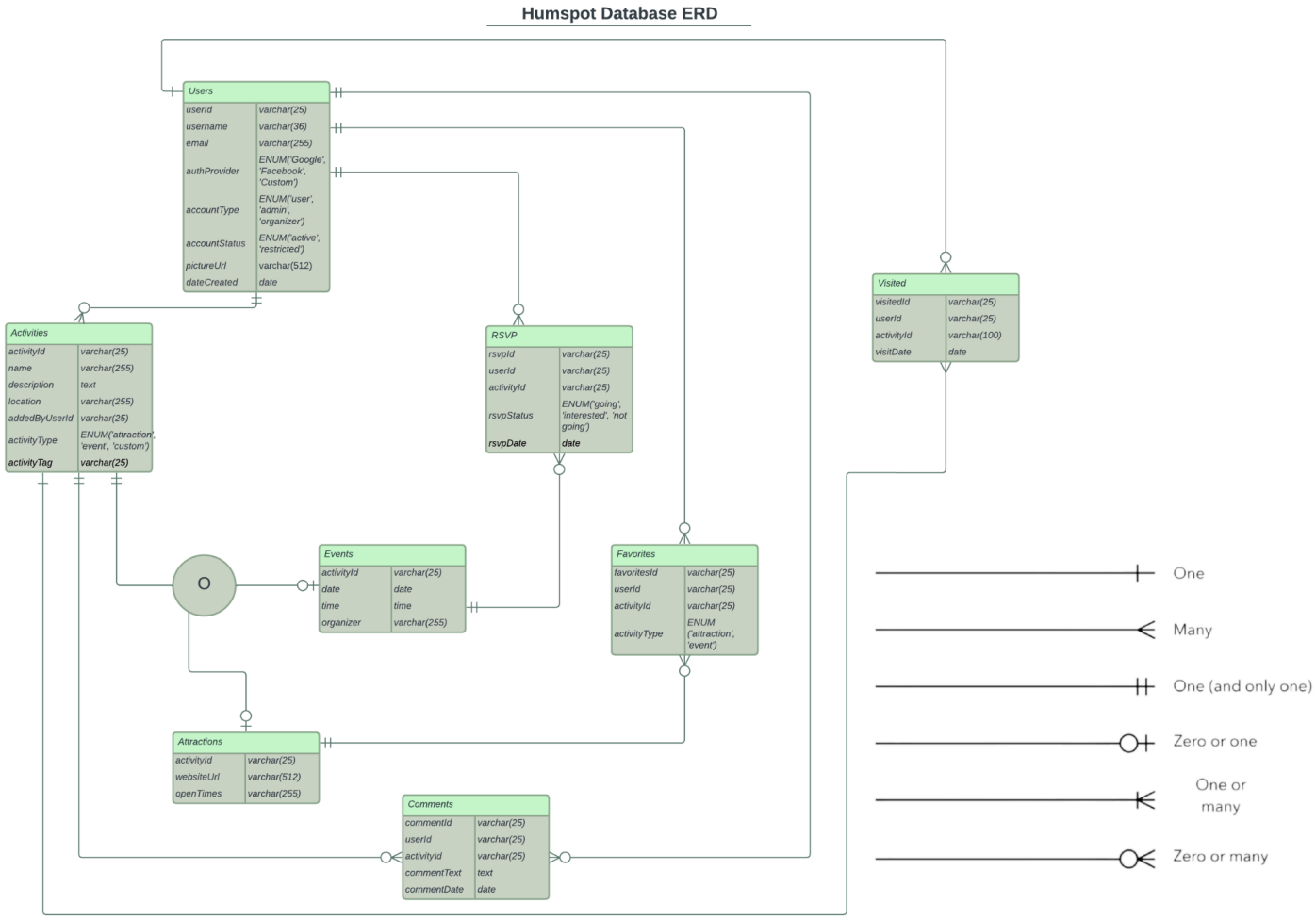
To be filled in later with updated components…

3.2. Backend

Humspot was built with an Amazon Web Services (AWS) backend (https://aws.amazon.com/).

3.2.1. Database

Humspot utilizes a mySQL database through AWS (https://aws.amazon.com/rds/). To see the contents of the database, reach out to our developer team at dev@humspotapp.com for the credentials. Find below the Database Entity Relationship Diagram (ERD).



DESTRUCTION NOTICE – Destroy by any method that will prevent disclosure of contents or reconstruction of the documents.

3.2.2. Application Programming Interface (API)

The APIs used by the Humspot application were written in Typescript; these files can be found in the "Humspot/lambdas" folder. Each child folder here contains an *index.ts* file containing the source code.

The API utilizes the AWS lambda service (https://aws.amazon.com/lambda/), which allows serverless code execution. This code then interacts with other AWS services, such as the aforementioned mySQL AWS RDS database, AWS API Gateway, and AWS Amplify authentication service. These lambda functions are called through functions defined in *server.ts*

In order to add your own lambda function (API) to Humspot, reach out to our developer team at dev@humspotapp.com for access to the AWS console. Your account will then be added to the development team on AWS, allowing you to publish code.

Once you have an AWS account / user added, run the following command.

```
npm install -g aws-cli
```

To create a new lambda function, visit (https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html#getting-started-create-function) for reference; then run the following commands

```
# assuming you are using Typescript
cd lambdas
mkdir <new-lambda-function> # pick a name for your function
cd <new-lambda-function>
npm init # keep pressing enter to accept all defaults
touch index.ts
touch tsconfig.json
npm install
```

Copy the contents of tsconfig.json from another lambda folder (or from the code below) and paste it into your newly created tsconfig.json

```
{
  "compilerOptions": {
    "module": "CommonJS",
    "moduleResolution": "Node",
    "target": "ES2017",
    "noImplicitAny": true,
    "preserveConstEnums": true,
    "outDir": "./dist",
    "sourceMap": true
  }
}
```

Replace the contents of package.json with the code below, replacing the code in the angle brackets <> with the name of your function and your name respectively.

```
{
  "name": "<new-lambda-function>",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "prebuild": "rm -rf dist",
    "build": "esbuild index.ts --bundle --minify --sourcemap
--platform=node --target=es2020 --outfile=dist/index.js",
    "postbuild": "cd dist && zip -r index.zip index.js*"
  },
  "author": "<YOUR-NAME>",
  "license": "ISC",
  "devDependencies": {
    "@types/aws-lambda": "^8.10.124",
    "@types/mysql2": "github:types/mysql2",
    "@types/node": "^20.8.3",
    "esbuild": "^0.19.4"
  },
  "dependencies": {
    "mysql2": "^3.6.1"
  }
}
```

After coding your lambda function, run the following to push to the AWS server.

```
npm install
npm run build
aws lambda create-function --function-name <new-lambda-function>
--runtime "nodejs18.x" --zip-file "fileb://dist/index.zip" --handler
index.handler --region us-west-1 # change <new-lambda-function> with
your created name
```

The newly created lambda function should appear in the AWS console (https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1#/functions)

To attach an endpoint to the lambda function, visit (https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html#apigateway-add) and follow the instructions there. This will allow you to invoke the lambda function using a URL.

It is not recommended to edit the existing lambda functions. If you do, please creare a pull request on GitHub so that it may be discussed with the development team.

3.2.3. Backend Files

Find below the relevant backend lambda function files and their descriptions. Also see the API Gateway URL associated with the lambda function, along with the function in *server.ts* that uses the URL.

Humspot/lambda/add-activity-to-favorites/index.ts - adds an activity to a User's favorites list. (This means adding a new row to the Favorites table). It returns the favoriteID associated with the newly created favorite row (assuming successful creation).

- URL: https://0l7ll7i47l.execute-api.us-west-1.amazonaws.com/default/add-activity-to-favorites
- Method type: POST
- Required parameters: *userID : string, activityID: string*

```
/**

* @function handleAddToFavorites

* @description Adds the activity (event or attraction) to the User's
favorites list.

*

* @param {string} userID the ID of the currently logged in user

* @param {string} activityID the ID of the activity (primary key of
Activities table)

*

* @returns {Promise<AddToFavoritesResponse>} a status message along with the
newly created favoriteID.

*/

export const handleAddToFavorites = async (userID: string, activityID:
string): Promise<AddToFavoritesResponse> => { ... }
```

Humspot/lambda/add-activity-to-rsvp/index.ts - adds an activity to a User's RSVP list. Returns the RSVPID associated with the newly created RSVP row (assuming successful creation)

- URL: https://n3slxwct06.execute-api.us-west-1.amazonaws.com/add-activity-to-rsvp
- Method type: POST
- Required parameters: *userID: string, activityID: string, rsvpDate: string*

```
/**
* @function handleAddToRSVP
* @description Adds an activity to a User's RSVP list.
*
* NOTE: List in this context refers to a row entry in the RSVP table.
*
* @param {string} userID the ID of the logged in user.
* @param {string} activityID the If of the activity (primary key of the
Activities table).
* @param {string} rsvpDate the date the user visited the Activity (Event /
Attraction)
*/
export const handleAddToRSVP = async (userID: string, activityID: string,
rsvpDate: string): Promise<AddToRSVPResponse> => { ... }
```

Humspot/lambda/add-activity-to-visited/index.ts - Adds an activity to a User's visited list (adds a new row to the User table). Returns the visitedID associated with the newly created visit row (assuming successful creation)

- URL: https://rdqlt3q1ye.execute-api.us-west-1.amazonaws.com/default/add-activity-to-visited
- Method type: POST
- Required parameters: *userID: string, activityID: string visitDate: string*

```
/**
* @function handleAddToVisited
* @description Adds an activity to a User's visited list.
*
* NOTE: List in this context refers to a row entry in the Visited table.
*
* @param {string} userID the ID of the logged in user.
* @param {string} activityID the If of the activity (primary key of the Activities table).
* @param {string} visitDate the date the user visited the Activity (Event / Attraction)
*/
export const handleAddToVisited = async (userID: string, activityID: string, visitDate: string): Promise<AddToVisitedResponse> => { ... }
```

Humspot/lambda/add-attraction/index.ts - adds an attraction to the database (adds a new row to the Attractions table). Returns the attractionID associated with the newly created event (assuming successful creation). This is meant to be invoked within the **approve-activity-submission** lambda, as the attraction must first be approved by an admin.

- URL: https://fuqzv9ftlh.execute-api.us-west-1.amazonaws.com/add-attraction
- Method type: POST
- Required parameters: *newAttraction: HumspotAttraction*

```
/**
* @function handleAddAttraction
* @description Calls the AWS API gateway /add-attraction. This will add a new attraction to the database
*
* @param {HumspotAttraction} newAttraction the attraction to be added.
*
* @returns {Promise<AddAttractionResponse>} response containing a message of success or error. If successful, the newly added attractionID is returned.
*/
export const handleAddAttraction = async (newAttraction: HumspotAttraction): Promise<AddAttractionResponse> => {
```

Humspot/lambda/add-comment/index.ts - add a comment to an activity. (Adds a row entry to the Comments table). Returns the commentID associated with the newly created comment (assuming successful creation)

- URL: https://2z5r6b16d2.execute-api.us-west-1.amazonaws.com/default/add-comment
- Method type: POST
- Required parameters: comment: *HumspotCommentSubmit*

```
/**
* @function handleAddComment
* @description calls the AWS API gateway /add-comment. This will add a row to
the Comments table.
*
* @param {HumspotCommentSubmit} comment the user comment data.
* @param {Blob | null} blob the blob data for a comment image.
* @param {string} activityName the name of the activity
*/
export const handleAddComment = async (comment: HumspotCommentSubmit, blob:
Blob | null, activityName: string) => { ... }
```

Humspot/lambda/add-event/index.ts - adds an event to the database (adds a new row to the Events table). Returns the attractionID associated with the newly created event (assuming successful creation). This is meant to be invoked within the **approve-activity-submission** lambda, as the event must first be approved by an admin.

- URL: https://38tmnse7u5.execute-api.us-west-1.amazonaws.com/add-event
- Method type: POST
- Required parameters: comment: *event: HumspotEvent*

```
/**
* @function handleAddEvent
* @description Calls the AWS API gateway /add-event. This will add a new
event to the database.
*
* @param {HumspotEvent} newEvent the event to be added.
*
* @returns {Promise<AddEventResponse>} response containing a message of
success or error.
* If successful, the newly added eventID is returned.
*/
export const handleAddEvent = async (newEvent: HumspotEvent):
Promise<AddEventResponse> => {
```

Humspot/lambda/add-rating/index.ts - adds a rating to an attraction (1 to 5 stars). A new row is added to the Ratings table.

- URL: https://alxkusswyl.execute-api.us-west-1.amazonaws.com/add-rating
- Method type: POST
- Required parameters: comment: *userID: string, activityID: string, rating: number*

```
/**
* @function handleAddRating
* @description Adds a rating to a specified activity (of type attraction).
*
* @param {string} userID
* @param {string} activityID
* @param {number} rating
*/
export const handleAddRating = async (userID: string, activityID: string,
rating: number): Promise<{ message: string; success: boolean; }> => {
```

Humspot/lambda/approve-activity-submission/index.ts - Approves an activity submitted by a user. If an event, a new entry is added to the Events table, if an attraction, a new entry is added to the Attractions table. Tags are added to the ActivityTags table (if any), and photo URLs are added to the ActivityPhotos table (if any).

- URL: https://ekm437chy2.execute-api.us-west-1.amazonaws.com/approve-activity-submission
- Method type: POST

```
/**
* @function handleApproveActivitySubmission
* @description allows admins to approve an activity submitted by the user.
*
* @param {string} adminUserID
* @param {SubmissionInfo} info
* @param {string} reason the message to be sent to the user who submitted the activity
*/
export const handleApproveActivitySubmission = async (adminUserID: string, info: SubmissionInfo, reason: string) => { ... }
```

Humspot/lambda/approve-organizer-submission/index.ts - Approves a user to become an organizer. The accountType column value in the Users table is updated to 'organizer' for the specified user. Sends an email to the user with a message from the approving admin.

- URL: https://ke5m3wbgdb.execute-api.us-west-1.amazonaws.com/approve-organizer-submission
- Method type: POST

```
/**
* @function handleApproveOrganizer
* @description updates a user's account type to 'organizer'
*
* @param {string} approverID
* @param {string} submitterID
* @param {string} submitterEmail
* @param {string} submissionID
* @param {string} reason
*/
export const handleApproveOrganizer = async (approverID: string, submitterID: string, submitterEmail: string, submissionID: string, reason: string) => {
```

Humspot/lambda/deny-organizer-submission/index.ts - Sends an email to the user who submitted a request with a reason why their submission to become organizer was denied.

- URL: https://ojes65o5w5.execute-api.us-west-1.amazonaws.com/deny-organizer-submission
- Method type: POST

```
/**
* @function handleDenyOrganizer
*
* @param {string} approverID
* @param {string} submitterID
* @param {string} submitterEmail
* @param {string} submissionID
* @param {string} reason
*/
export const handleDenyOrganizer = async (approverID: string, submitterID:
string, submitterEmail: string, submissionID: string, reason: string) => {
...}
```

Humspot/lambda/get-activities-given-tag/index.ts - Retrieves the activities from the database given the page number and tag. Each page pulls 10 activities from the database. NOTE: The name of this function in AWS is get-events-given-tag.

- URL: https://0xegux6hd0.execute-api.us-west-1.amazonaws.com/get-events
- Method type: GET

```
/**
* @function handleGetActivitiesGivenTag
* @description Gets an array of events that have a certain tag associated
with it.
* It returns 10 events at a time, and more can be loaded by incrementing the
pageNum param.
*
* @param {number} pageNum the page number which corresponds to the offset
when selecting rows in the table
* @param {string} tag the event tag
*
* @returns {Promise<GetActivitiesGivenTagResponse>} a status message along
with an array of events that have a certain tag associated with it.
*/
export const handleGetActivitiesGivenTag = async (pageNum: number, tag:
string): Promise<{ message: string; activities: any[]; success: boolean; }>
=> {
```

Humspot/lambda/get-comments-given-userid/index.ts - Retrieves a user's comments and RSVPs from the database given the page number and the userID.Each page pulls 20 comments and/or RSVP'd events from the database.

- URL: https://t9skeznpb6.execute-api.us-west-1.amazonaws.com/get-comments-given-userid
- Method type: GET

```
/**

* @function handleGetInteractionsGivenUserID

* @description gets an array of comments and/or RSVP'd events from a specified user.

* It returns 20  at a time, and more can be loaded by incrementing the pageNum param.

*

* @param {number} pageNum

* @param {string} userID

*

* @returns {Promise<GetCommentsResponse>} a status message, and, if successful, an array of 20 comments and/or RSVP'd events of type GetCommentsResponse

*/

export const handleGetInteractionsGivenUserID = async (pageNum: number, userID: string): Promise<GetInteractionsResponse> => {...}
```

Humspot/lambda/get-favorites-and-visited-and-rsvp-status/index.ts - Retrieves whether a user has visited or favorited or RSVP'd for an activity. NOTE: The name of the function in AWS is get-visited-and-favorites-status.

- URL: https://99yxfoi1wb.execute-api.us-west-1.amazonaws.com/get-visited-and-favorites-status
- Method type: GET

```
/**

* @function handleGetFavoritesAndVisitedAndRSVPStatus

* @description gets whether the user has favorited, visited, and/or RSVP'd
for the activity.

*

* @param {string} userID

* @param {string} activityID

* @returns {GetFavoritesAndVisitedAndRSVPStatusResponse} an object containing
the statuses along with a message

*/

export const handleGetFavoritesAndVisitedAndRSVPStatus = async (userID:
string, activityID: string):
Promise<GetFavoritesAndVisitedAndRSVPStatusResponse> => {
```

Humspot/lambda/handle-user-login/index.ts - called after a user logs in. If it is their first time, this function will create a new row in the mySQL Users table. If the user had logged in before, this function will return an object containing the user's information from the mySQL Users table.NOTE: The name of this function in AWS is create-user.

- URL: https://5w69nrkqcj.execute-api.us-west-1.amazonaws.com/create-user
- Method type: POST

```
/**
* @function handleUserLogin
* @description Calls the AWS API gateway /create-user. This will create a new
user in the database if first time logging in.
*
* @param {string | null} email
* @param {string | null} username
*
* @returns {Promise<LoginResponse>} response containing a message of success
or error.
* If successful, the user object is returned of type HumspotUser.
*/
export const handleUserLogin = async (email: string | null, username: string
| null, isGoogleAccount: boolean): Promise<LoginResponse> => {
```

Humspot/lambda/submit-attraction-for-approval/index.ts - adds a pending attraction submission to the Submissions table. Emails the admins that an activity was submitted and should be looked at for approval.

- URL: https://jlc16t5zgj.execute-api.us-west-1.amazonaws.com/submit-attraction-for-approval
- Method type: POST

```
/**
* @function handleSubmitAttractionForApproval
* @description adds a row to the Submissions table
*
* @param {HumspotAttraction} attraction the attraction submission info
*/
export const handleSubmitAttractionForApproval = async (attraction:
HumspotAttraction) => { ... }
```

Humspot/lambda/submit-event-for-approval/index.ts - adds a pending event submission to the Submissions table. Emails the admins that an event was submitted and should be looked at for approval.

- URL: https://b2k3lpio1c.execute-api.us-west-1.amazonaws.com/submit-event-for-approval
- Method type: POST

```
/**
* @function handleSubmitEventForApproval
*
* @param {HumspotEvent} event
*/
export const handleSubmitEventForApproval = async (event: HumspotEvent) => {
```

Humspot/lambda/submit-request-for-coordinator/index.ts - submits a request to the admin for a user to become an organizer (called coordinator here, interchangeable)

- URL: https://rgncdxgadk.execute-api.us-west-1.amazonaws.com/submit-request-for-coordinator
- Method type: POST

```
export const handleSubmitRequestToBecomeOrganizer = async (data:
OrganizerRequestSubmission) => { ... }
```

Humspot/lambda/get-school-events-and-add-to-database/index.ts - Pulls the data from the HSU XML Event Data and stores events in the database.

- URL: NONE
- Method type: POST; Not meant to be invoked by anyone as it is automated!

Humspot/lambda/scrape-google-events/index.ts - runs every 3 days. It uses SerpApi to retrieve the top 50 local Humboldt events from Google. This information is then parsed and put into a format that the mySQL Events table expects so that it can be entered into it.

- URL: NONE
- Method type: POST; Not meant to be invoked by anyone as it is automated!