CS 4366: Senior Capstone Project
Dr. Sunho Lim
Project #4 - Implementation and Partial-Demo - Project Report
EmergenSeek

Suhas Bacchu Derek Fritz Kevon Manahan Annie Vo Simon Woldemichael

April 1, 2019

Abstract

In this report, we describe and detail the progress that we have made so far in developing the backend and mobile client for the EmergenSeek application. The report will be broken up into sections such that, each section will answer the questions posed in the "Deliverables" section of the Project 4 statement file.

1 Project Restatement

As a reminder, EmergenSeek is a mobile application which will provide users with multiuse, centralized emergency information and notification services. This application will provide friends and family members with priority connections in times of emergency or crisis. For the scope of the following two months, within this class, our main function requirements are as follows.

1. S.O.S. button emergency broadcast — The user shall be able to utilize the mobile client to press and hold an S.O.S. button for automated notification of contacts and emergency services.
2. Emergency service locator — The user shall be able to utilize the mobile client to search for emergency service (i.e. hospitals, pharmacies, police boxes)
3. Periodic notifications to contacts (location-based polling) — The user shall be able to utilize the mobile client to periodically send their location information to contacts.
4. Granular permission definitions for contacts — The user shall be given full control over what contacts receive what level of information.
5. Lock screen display of health information for emergency services — In the case of an S.O.S. situation, the user shall have their health information displayed for the convince of first responders.

   Since beginning implementation, we have found the need for two additional Lambda functions. ESGetUser and ESCreateUser. These are necessary for retrieving existing users and creating new users of the application, respectively.

2 Lambda Functions

This section should be used as a reference when there is any mention of an implemented Lambda function. The functions are prefixed with `ES` (short for EmergenSeek). This section will detail 6 things:

1. The HTTP method necessary for interacting with the Lambda function.
2. The API Gateway route associated with the function.
3. The JavaScript Object Notation (JSON) request body required by the function to produce desired output.
4. The JSON response returned by the API for a request.
5. A simple-to-understand description of what the function performs as a result of the request.
6. Services and API's that this function is dependent upon.

   (i) Name: ESSendSMSNotification
   (ii) HTTP Method: POST

(iii) API Gateway Route: /sms

(iv) JSON Request Body:

```
{
     "user_id": String
     "type": int,
     "message": String
     "last_known_location": double[2]
     }
```

(v) JSON Response Body:

```
{
     "body": String
        }
```

(vi) Dependencies: DynamoDB, Twilio

3 Project Implementation

In this section we will describe how we implemented the project and give an overview of how the backend and mobile client are communicating, as well as the role of all of the technologies involved.

As a whole, and general rule when following a client-server architecture, the backend is an abstraction of all of the operations necessary for the mobile applications functionality. An end-user or a developer on the front-end team will never need to know of the serverless architectural pattern [2] that our backend follows. Beginning from deriving out system features and requirements, we determined what Flutter components would be necessary to expose this functionality to the user and additionally determined the respective Golang AWS Lambda function to provide that functionality via an API call. That being said, the following subsections will describe precisely how this was achieved for each of the functionalities referenced in section 1.

3.1 1 - S.O.S. button emergency broadcast

This feature depends on three functionalities. On the client side, the user will press and hold an S.O.S. button. After a set amount of time, visualized via a loading bar, the user will have the option to select an emergency severity. At the moment the client only implements a “severe” and “mild” emergency option.

- If the user selects `severe`, the application will make a request to ESSendSMSNotification and ESSendEmergencyVoiceCall via an HTTP POST request.

References

[1] Cloudcraft `https://cloudcraft.co`

[2] Build Specification Reference for AWS CodeBuild `https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html`

[3] Testing Flutter apps `https://flutter.dev/docs/testing`

[4] Automating Publishing to the Play Store `https://github.com/codepath/android_guides/wiki/Automating-Publishing-to-the-Play-Store`

[5] What is the AWS Serverless Application Model (AWS SAM)? `https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html`