EmergenSeek

CS 4366: Senior Capstone Project
Dr. Sunho Lim
Texas Tech University
Project #5
Validation and Full Demo
EmergenSeek

Spring 2019 Final Report
by

Suhas Bacchu
Derek Fritz
Kevon Manahan
Annie Vo
Simon Woldemichael

Abstract

In this final report, we describe and detail the validation of EmergenSeek, a multiuse, cross-platform mobile application providing simple access to emergency services and contact connections. As with previous reports, we will structure the report by enumerating on the frontend and backend. Dissimilarly, we will only describe necessary details in the validation of our implementations, to the extent that, any other developer with some programming background will be able to successfully implement our system's components. Only necessary details will be presented to the reader and care should be taken when reading them[1]. Restatements from previous deliverable reports will be made to remind the reader of pre-defined, but important information. Also, a glossary is provided at the end of the report to define terms and jargon that may be confusing or unfamiliar for the reader.
Keywords: API Gateway, CI/CD, Cloud, Dart, Flutter, Golang, Lambda, Mobile, Serverless

[1]Concepts such as continuous integration, continuous deployment, microservices, serverless computing, NoSQL databases, APIs, and widget-based mobile development will be discussed. These new concepts may be difficult to grasp initially, but sufficient details will be provided.

Contents

1 Introduction

The development of EmergenSeek has been an effective software engineering experience for our group. In this final submission, we will discuss our project by first restating the idea, project motivations, and our successful implementations. Then, we will discuss the final state of our UML diagrams and architecture, for this Spring semester. Next, a walkthrough of the our development lifecycle, using only necessary details and supplementing those details with additional code comments, snippets, and screenshots. This walkthrough will allow the user to understand the structure of our application, both frontend and backend. Then, we will describe the current state of our serverless application repository; our collection of Lambda functions and a brief discussion on the microservices software architectural pattern that we follow. Lastly, we will display views of the mobile application and their respective functionality.

It is very important to note that our project uses programming languages which are young in maturity and may not be familiar to the reader. But, our backend, written in Go, has syntax very similar to the C and C++ programming languages. Our frontend, written in Dart, has syntax very similar to the Java programming language. As a result of these similarities, it is permissible for the reader to assume what the code does as the syntax is, for the most part, straightforward. To conclude we will detail team member contributions and potential steps moving forward with the project.

1.1 Walkthrough

The previously mentioned walkthrough will consist of the following, and be enumerated for a single Lambda function and a single Flutter application screen. This section should be referred to in the Project Development Lifecycle section:

1. Feature expectation. Answer the question of "What feature do we need to implement and how do we implement it?"
2. Lambda function: Implement the feature and its expectations around the requirements of using AWS Lambda
3. Informal testing: Get an initial working version of the API Lambda function
4. Formal testing: Write unit tests for the implementation
5. Integration and Infrastructure as Code: Define the deployment
6. Continuous Integration: Do the deployment
7. Integrate Lambda functionality with the frontend

    7a. Create feature view. The view is effectively the screen that the user will interact with.

    7b. Create feature model. Define how features will be referenced by the state of the application.

    7c. Interface with Lambda function. Make calls via an HTTP client to the Lambda function

As all of our project code is open-source, readers are free to reference code repositories for our project. They are located at https://github.com/emergenseek/backend and https://github.com/emergenseek/Flutter-Client for our backend and frontend code, respectively. Readers who would like more insight, or have questions are welcome to open an issue or pull request on either of these GitHub repositories.

2 Project Overview

2.1 Project

EmergenSeek is a mobile application which provides users with multiuse, centralized emergency information and notification services. This application gives friends and family members priority connections in times of emergency or crisis. Our sucessfully implemented, core feature are as follows:

1. S.O.S. button emergency broadcast — Users are able to utilize the mobile client to invoke an S.O.S. button for automated notification of contacts and emergency services.
2. Emergency service locator — Users are able to utilize the mobile client to search for emergency service (hospitals, and pharmacies). Currently, searching will return results up to a fixed radius of 20 miles.

3. Granular permission definitions for contacts — The user is given full control over what contacts receive what level of information. This is achieved by setting alert tiers on a per-contact basis.

4. 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 respondents.

Features which are partially implemented are as follows:

1. Periodic notifications to contacts (location-based polling) — The user shall be able to utilize the mobile client to send periodically send their location information to contacts. While the logic is in-place, we currently do not have a successful implementation of repeating the location polling.

2. Retrieval of international emergency service numbers (ambulance, fire, and police), depending on the user's current location. Connections to the front-end have not been made yet.

Additionally, successfully implemented, quality of user-experience features are as follows. While the application is still functional without them, deploying the application for public, functional use, cannot not be possible without them.

1. User registration and login — Users are able to register for an account and track their personal information, necessary in an emergency event.

2. Settings persistence – Users are able to change settings within the application to fit their preference.

Lastly, additional quality of user-experience features that have not been, but maybe implemented in the future are:

1. More extensive error handling and reporting (via something like the Catcher, a Flutter package or Sentry.io a multi-language, multi-framework error reporting and management service)

2. More in-depth user preferences, settings and options

3. Alternative sign-up and log-in methods (Google, Facebook, etc.)

3 Implementation Details

The implementation of EmergenSeek may be broken of into two core component, Amazon Web Services' (AWS) Lambda and the cross-platform mobile development framework Flutter. These two components also have their own respective, external dependcies and services. Our Lambda functions were written using the Go programming language and the Flutter application depends on the Dart programming language. This section is broken up into two subsections which describe the

3.1 Subsystems

The backend of the application will be comprised of the following. For brevity, each is labeled with a tier:

1. Application Tier - AWS Lambda, a serverless, Functions-as-a-Service compute offering to run our Go code coupled with AWS API Gateway.

2. Database Tier - DynamoDB to store user data and information. We selected a NoSQL database because we knew that there would be changes to our data and schema, and wanted to rapidly make these changes without formally defining schemas which would be otherwise harder to change in the future.

3. Notification Tier - The Twilio API provides the system with programmable SMS and voice communications. It should be noted that our application does not use trial plan of Twilio and instead uses the paid version of the service. This allows us to contact any valid phone number through Twilio's API.

4. Location Tier - Google's Places and MapQuest's Geocoding APIs are used to better define the location of users and various emergency services. For instance, translation of latitude and longitude to human readable address.

The subsystems of the application are divided up into single-responsibility, service-oriented architecture. This means that the backend consists of several Lambda functions and each function is responsible for only one thing. In total, at the time of this report, there are *six* Lambda functions. The break down of their functionality and responsibilities are enumerated using a brief description, followed by any additional dependencies that they may have. These dependencies include any external APIs, datastores, or databases. Functions are prefixed with `ES` and suffixed with a meaningful name which explains their role.

4 Glossary

- Cross-platform application - software which has a single implementation, but may be executed on different distributions. (i.e. one code base, running on many different operating systems or processor architectures.)
- Serverless application repository - single responsibility application definitions which act as independent entities. The application repository as a whole defines the API for our mobile application. (i.e. each serverless application is a Lambda function, and the lambda function is responsible for a system feature.)
- Application Programming Interface (API) - Definitions and communication protocols used for building the structure of software.
- HyperText Transfer Protocol (HTTP) - An application protocol used heavily for websites and services throughout the Internet.

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

List of Figures