CS 4366: Senior Capstone Project
Dr. Sunho Lim
Project #3 - Software Design Specification - Project Report
EmergenSeek

Suhas Bacchu Derek Fritz Kevon Manahan Annie Vo Simon Woldemichael

February 27, 2019

Abstract

In this report, we describe and detail the layout and design of EmergenSeek. The structure of the system, at a high-level, consists of two components; a mobile client, developed using a Dart-based, cross-platform mobile framework called Flutter, and a backend API, developed using Amazon Web Services and the Go programming language. Each of these components is also broken down into their own modules and components. Respective design documentation, including UML class, use case, and sequence diagrams, are included as well.

1 Project Restatement

As a reminder, EmergenSeek is a mobile application which will provide users with multiuse, centralized emergency information and notification. 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, (or 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 send 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.

2 Testbed

Before discussing the design documentation and the modular composition of the backend and frontend, we will describe how our project, both backend and frontend, will be tested and validated throughout development. At the core of our testbed is an Amazon Web Service known as CodeStar. CodeStar is comprised of four components; CodeCommit, CodeBuild, CodeDeploy, and CodePipeline. CodeCommit entails how the code and tooling of our project will be maintained. In this instance, we will be using two GitHub repositories, one for the backend and one for the frontend. Both are publicly available at `https://github.com/emergenseek`.

On every `git push` event (from a developers local machine to GitHub), CodeStar will begin a build sequence; this is where CodeBuild comes into play. CodeBuild will read a YAML specification [2], and per that specification, build our project. For our backend repository, the respective Go commands, will be invoked. This involves building all of our Lambda functions into binaries, testing them using our own custom unit tests, and preparing the static binaries for deployment by compressing them. Similarly, for the frontend, the respective Flutter commands will be invoked. This, again, involves testing building the Dart code into both an Android APK [3] for distribution on the Google Play store and an iOS IPA for distribution on the Apple iOS App Store [1].

[1] Because distribution of apps to Apple's iOS App Store requires a developer partnership, which requires the payment of a fee, deployment of the iOS binary is outside of the scope of this class

3 Design Documentation

In this section, we will discuss the various details of our design documents. In each subsection below, diagrams are displayed and followed by a brief description of each of the components within the associated diagram.

3.1 UML Class Diagram

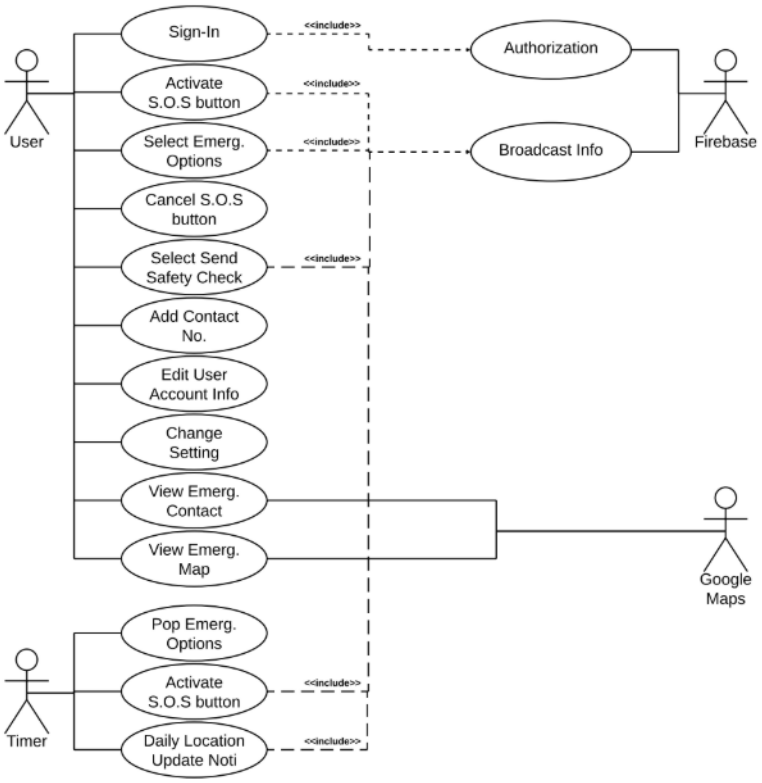

Figure 1: Use-case Diagrams for Emegenseek

3.2 UML Use-case Diagram

3.3 UML Sequence Diagrams

4 Client Analysis

In this section, we will enumerate the various modules and components responsible for keeping the frontend functional.

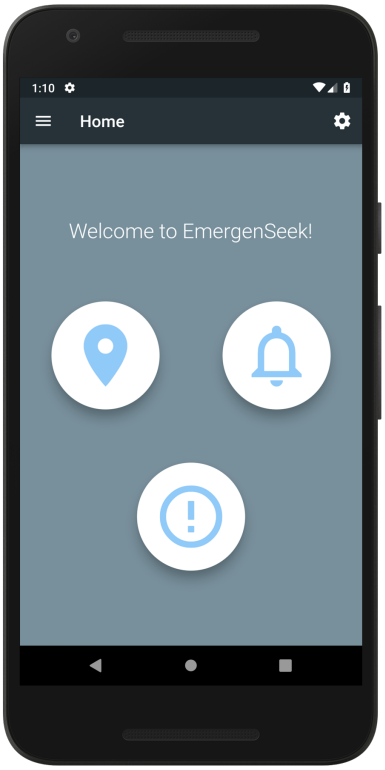

Figure 2: Screenshot of the mobile client's home page.

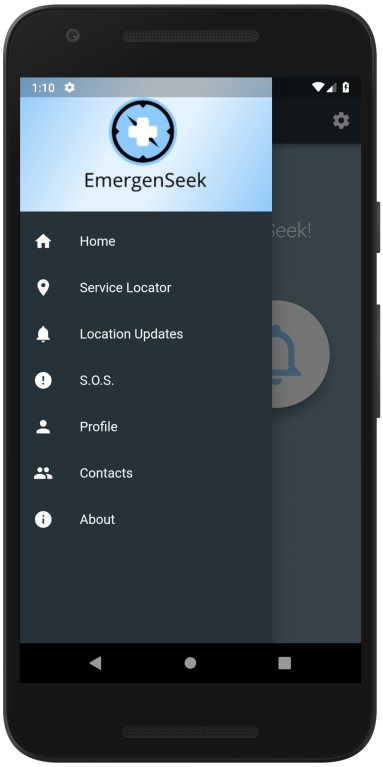

Figure 3: Screenshot of the mobile client's navigation menu.

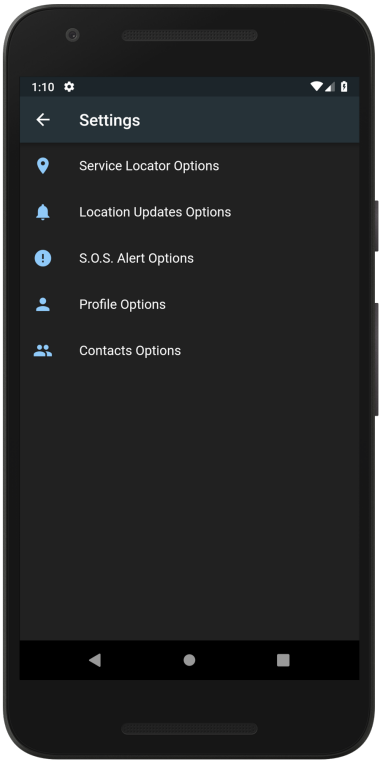

Figure 4: A screenshot of the mobile client's settings page.

5 Backend Analysis

In this section, we will enumerate the various modules, components, and objects responsible for keeping the backend functional. It should be noted that there is no literal `class` instance within the Go programming language, but objects may still be enumerated through the `struct` qualifier. Class attributes are defined as `struct` fields. Additionally, the backend will follow a decoupled, microservices architecture. Each of these microservices will be defined as a singular, AWS Lambda function which will serve a single purpose.

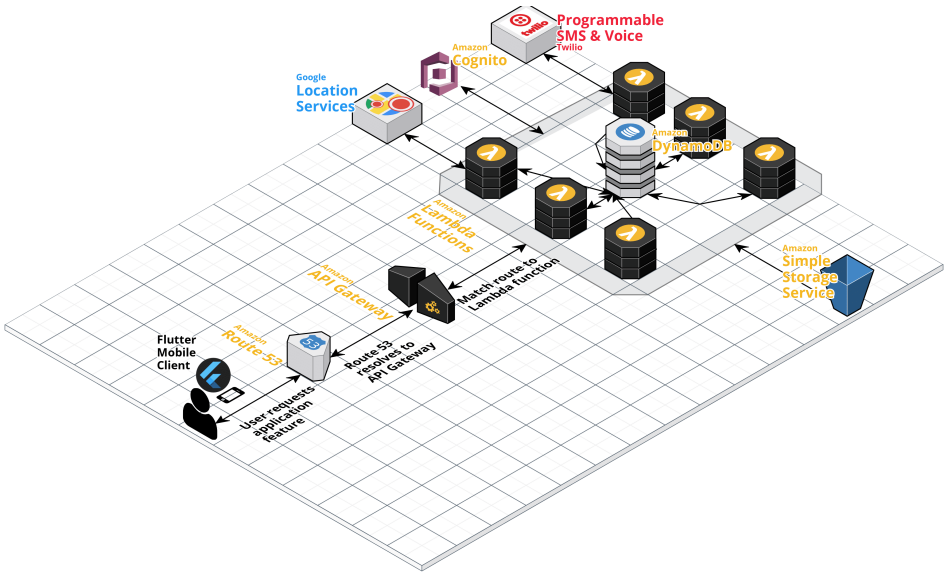

Figure 5: Cloudcraft [1] diagram of the AWS resources and external APIs necessary for the backend.

5.1 Models

The models of this application contain Go structures will replicate those defined in the UML Class Diagrams. The functions defined within those models are represented in Go as methods on the `struct`. For example:

```
type LocationPoller struct {
        // The location to be used for the current poll
        Location  []float32 `json:"location"`

        // The user to be used for the current LocationPoller instance
        User      *User     `json:"user"`

        // How often the location should be polled
        Frequency int       `json:"frequency"`
}

func (*c LocationPoller) DoSevere() error {
        // Method logic here
}
```

Figure 6: The code listing above describes how, in Go, a struct is associated with its methods. The method is a pointer receiver on the struct that it is related to. This is similar to how a `class`, in a language such as Java, has `class methods`

.

5.2 Subsystems

The backend of the application will utilize AWS Lambda, a serverless, Functions-as-a-Service offering to run our Go code, DynamoDB to store user data and information, the Twilio API to provide the system with programmable SMS and voice communications, and Google's Geocoding and Maps APIs to better define the location of users and various emergency services. 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.

5.2.1 Lambda Function - ESUpdateContactPermissions

1. Function:

2. Dependencies:

5.2.2 Lambda Function - ESSendEmergencySMSMessage

5.2.3 Lambda Function - ESGetLockscreenInfo

5.2.4 Lambda Function - ESCreateUser

5.2.5 Lambda Function - ESSendEmergencyVoiceCall

5.2.6 Lambda Function - ESPollLocation

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