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 Design Documentation

In this section, we will discuss the various details of our design documents.

1.1 UML Class Diagram

1.2 UML Use-case Diagram

1.3 UML Sequence Diagrams

2 Client Analysis

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

3 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.

3.1 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:

3.2 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.

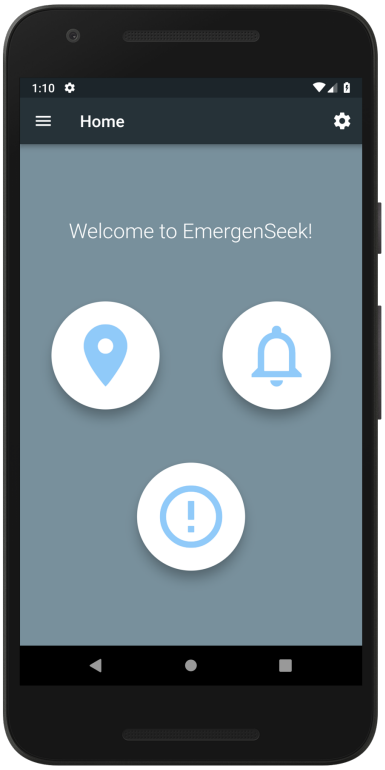

Figure 1: Screenshot of the mobile client's home page.

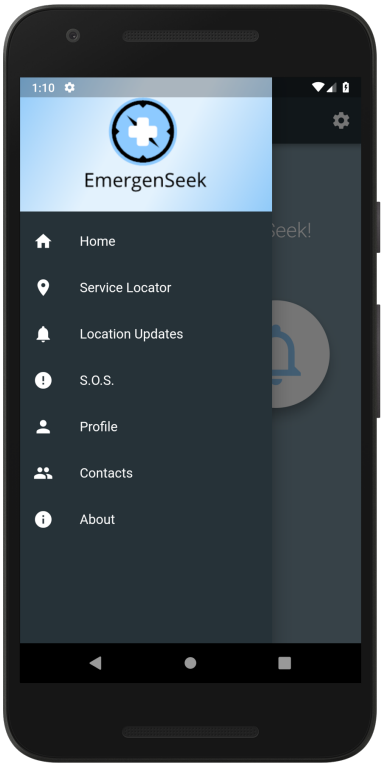

Figure 2: Screenshot of the mobile client's navigation menu.

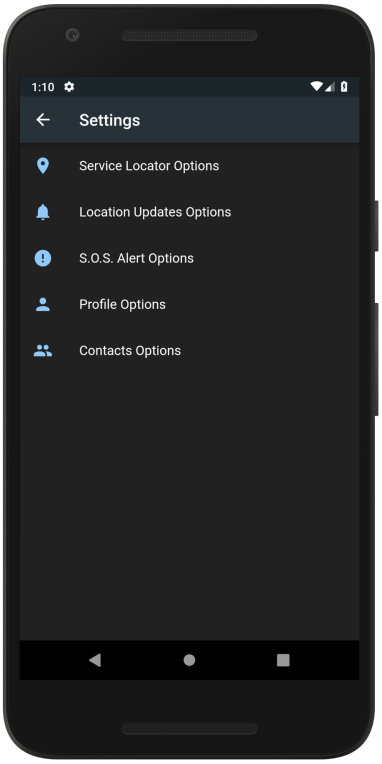

Figure 3: A screenshot of a placeholder page on the mobile client.

```
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 4: 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`
.

These dependencies include any external APIs, datastores, or databases. Functions are prefixed with `ES` and suffixed with a meaningful name which explains their role.

3.2.1 Lambda Function - ESUpdateContactPermissions

1. Function:
2. Dependencies:

3.2.2 Lambda Function - ESSendEmergencySMSMessage

3.2.3 Lambda Function - ESGetLockscreenInfo

3.2.4 Lambda Function - ESCreateUser

3.2.5 Lambda Function - ESSendEmergencyVoiceCall

3.2.6 Lambda Function - ESPollLocation

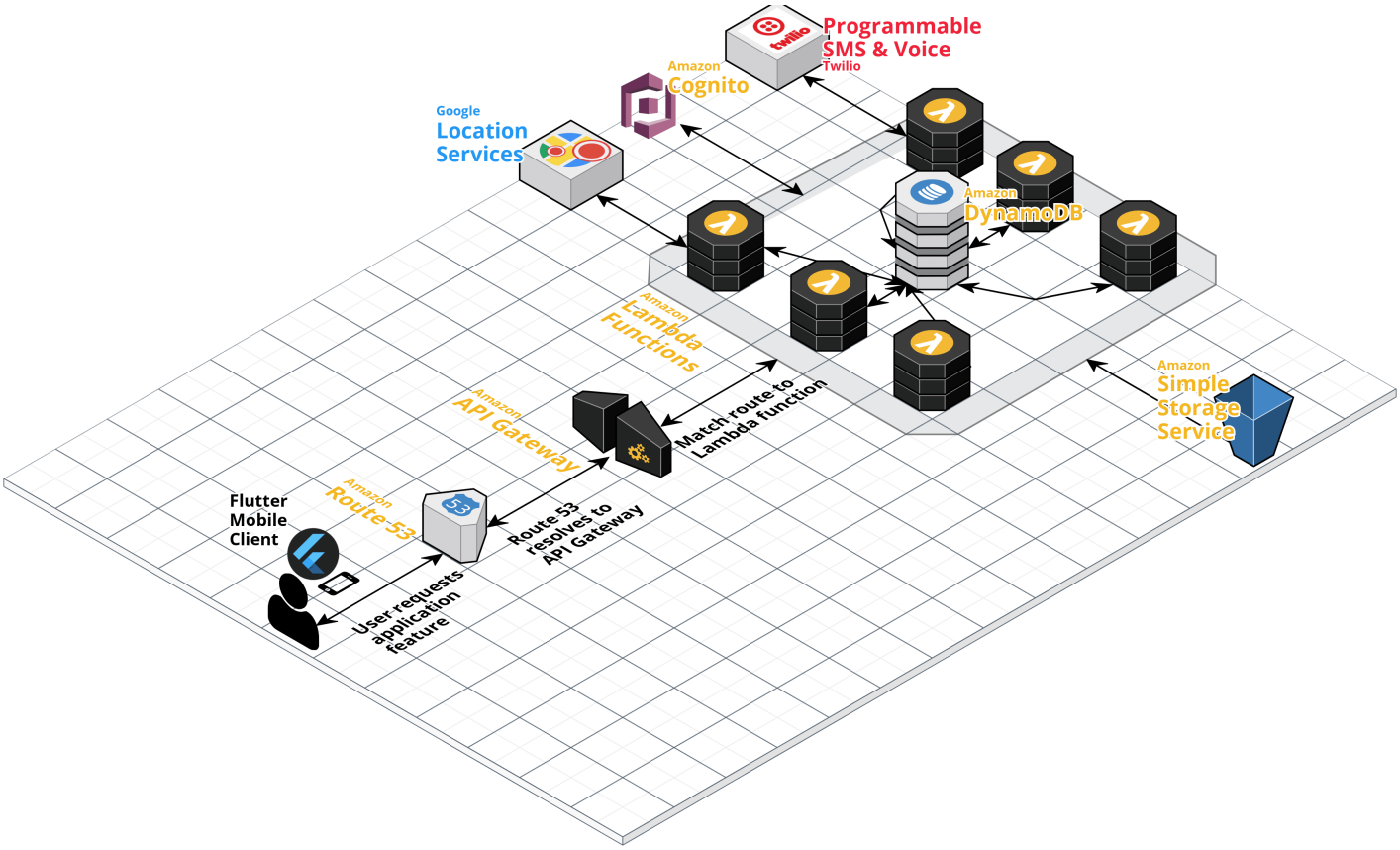

Figure 5: Cloudcraft [1] diagram of the AWS resources and external APIs necessary for the backend.

References

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