ParkWizard: Street Parking Android App

Siddharth Shah, Kunal Baweja, Dhruv Shekhawat, Anand Naik

Abstract

In this project, an android app is built to guide users to the nearest available street parking location. We use web services such as Google Cloud Messaging, Elasticsearch, SQS and Cognito to develop a point-based, crowd-sourced approach that strives to take the hassle out of street parking.

Motivation

Finding open street parking spots in metropolitan cities such as New York has always been a distressing task. Various apps have been developed in the past but with an intention of helping users book garage parking instead of finding available street parking. To mitigate this issue, we have created ParkWizard which employs various cloud services to update users with real-time street parking information.

Introduction

ParkWizard is an Android application that helps users find available street parking locations. It works on a crowd-sourced, point-based system where users earn points by reporting street parking locations and are able to use those points in order to obtain available parking spots. This point-based system works well towards maintaining a smooth information flow between users(finders) who want to find available parking spots and those(informers) who have information about them. At a point, any user can act as a finder or an informer. Users are authenticated using Amazon Cognito which assigns temporary security credentials for accessing our backend AWS resources. The app displays all the available parking spots which are within 500 meters radius from the location where the user wants to find parking. Users are initially granted 100 free points on creating an account with our app. Each time, users lose 5 points on searching a parking and gain 10 points on reporting available parking spots.

Architecture

We make use of Google OAuth to facilitate user login through Google account. At the time of login we also utilize AWS Cognito to provide users with temporary security credentials to access our app's backend resources. We have built our backend with Django webserver that communicates with Elasticsearch(ES) to store and retrieve real-time parking information. When users report parking, the information is first enqueued to SQS queue from where it is dequeued by our Django server. This ensures reliable data delivery and efficient handling of load on the server side.

On receiving the reported parking information, Django server streams it to ES which stores the data in the (lat,long, #available spots, #total spots) format. The user information is also stored on ES in the format (userid, points). When searching for parking, Django server retrieves the data from ES based on (lat,long) provided by user and sends it back to the app. We deploy our Django server on a load balanced AWS ElasticBeanstalk. Django server sends notifications about operation's status to Google Cloud Messaging which redirects it to the user's registered device.

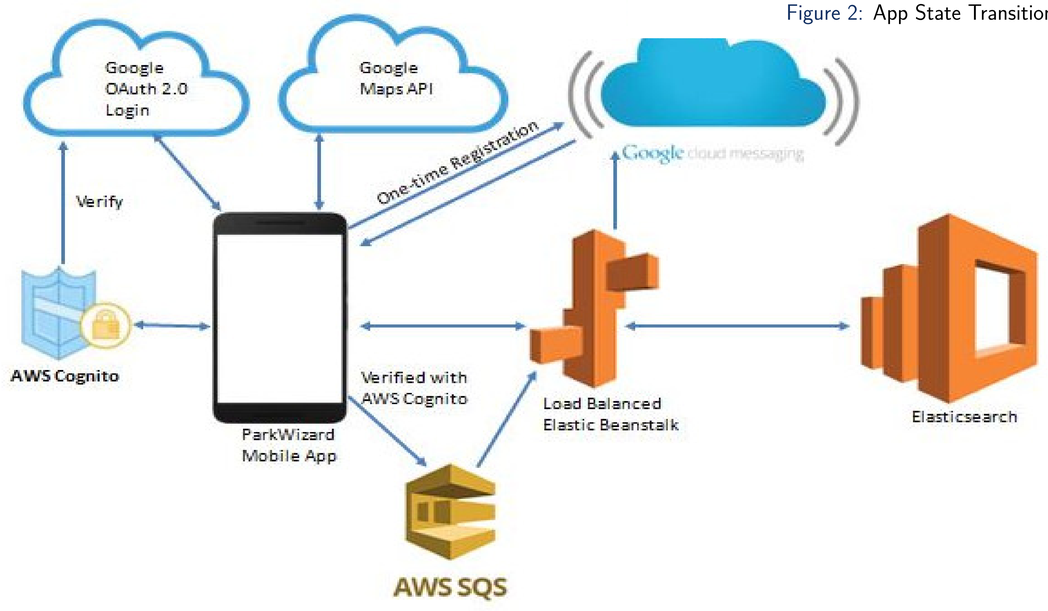

Figure 1: Architecture

Algorithm

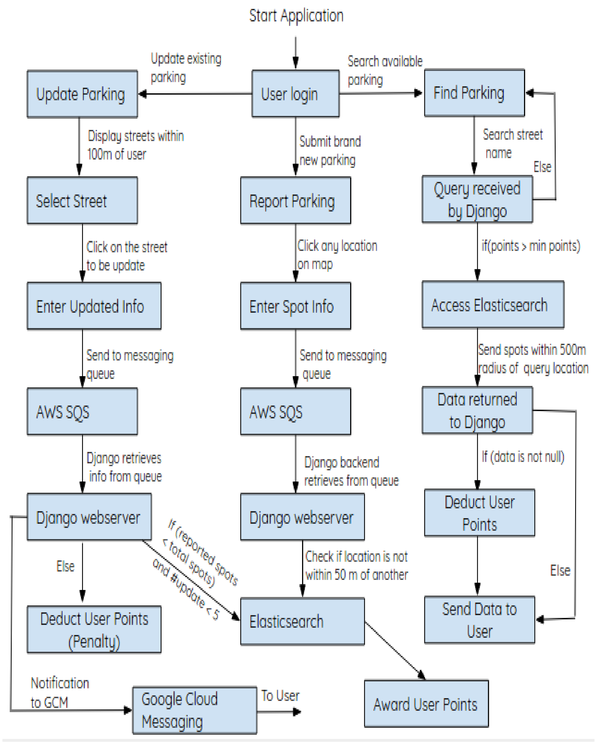

Figure 2: App State Transitions

Data Reliability

As our application is heavily dependent on the data supplied by users, we need to ensure that the information we store in ES is reliable. To accomplish this we enforce that the user only updates parking information for streets which are within 100 meter distance from the user's location. Additionally, a user can only report brand new parking spots which are not within 50 meter distance from an existing parking. This ensures that data is not duplicated for the same parking spots. We also deduct points of users who proclaim a higher number of available spots than the total number of parking spots itself. Therefore, its crucial to maintain the total possible parking spots of streets in our ES. Another feature is to only allow users to make a maximum of five parking spot updates per day. This protects against bots who may try to score unrestrained points by continuously updating parking locations.

Incentive-based Ideology

Our application does not contain any parking data of its own. It relies on its users to report and update parking spots in real-time. Therefore, we need to create some kind of an incentive for users to feed data to the app. A point-based approach works well for us here as users require a certain minimum score to be able to find parking. If they fall short of this they must feed available parking information to the app which guarantees an everlasting cycle of information supply and retrieval between users.

Conclusion

Parkwizard is an android app with a mission to mitigate the inconvenience suffered while finding street parking. It employs a crowd-sourcing, score-based approach where users earn points on reporting free parking spots and use those points to find parking when required. ParkWizard utilizes various cloud services to function as a scalable, reliable and secure application.