The Architecture of a Ride Sharing Application

Mao Zheng, Yifan Gu, and Chaohui Xu
Department of Computer Science, University of Wisconsin – La Crosse, La Crosse, WI, USA

Abstract - In the development of a lite version of a ride sharing application, we encounter some real time requirements. For example, a rider wants to track the driver’s location in real time, once a driver accepts the trip request. Compared to the traditional method in which the driver sends the location every minute to the app server and the rider receives the information every minutes from the server, we used Google Cloud Messaging (GCM) to send messages to client devices in this project. That means the driver sends GCM notification containing the location information to the GCM connection server. Then the information is sent to the rider. As a result, the rider will be able to view the driver’s location in real time. The benefit of using GCM is the scalability and performance advantages over the app server. There is also battery life savings for the clients’ mobile devices. This paper presents the architectural design of using GCM connection server in a ride sharing application.

Keywords: Mobile App, Google Cloud Messaging, Connection Server

1 Introduction

In this paper, we present a lite version of a ride sharing application. There are two main types of users: riders and drivers. A rider can request a car by providing his/her preferred start location and the destination of the trip. The rider can also view the cost of the trip, track the driver, pay and rate the driver after the trip. A driver can view all the requests from different riders, select a request, and pick up the rider. It requires some real-time communications between the rider and the driver. The information about the trip and all the requests will be recorded into a database. The project is hosted in Microsoft Azure cloud platform. NodeJS is used to write server-side code to provide RESTful APIs. MongoDB is a backbone database, and Redis is used as the cache to allow a fast system response.

In this ride sharing application, we use Google Cloud Messaging (GCM) to send messages to client devices. That means the driver sends a GCM notification containing his/her location information to the GCM connection server, then the information is sent to the rider. As a result, the rider will be able to view the driver’s locations in real time. The benefit of using GCM is the scalability and performance advantages over the app server. There is also the battery life savings for the clients’ mobile devices.

The paper is organized as follows: In section 2, we briefly introduce GCM and explain how we incorporate it in our project. In section 3, we present the architectural design of our ride sharing architecture. In section 4, we explain our testing scenario for real time functionalities. Section 5 concludes the paper and outlines the directions of our future work.

2 Google Cloud Messaging (GCM)

Google Cloud Messaging (GCM) is a free service that enables developers to send messages between servers and client apps. This includes downstream messages from servers to client apps, and upstream messages from client apps to servers [1]. A GCM implementation includes a Google connection server, an app server that interacts with the connection server via HTTP or XMPP protocol, and a client app. The client app is a GCM-enabled client app.

When clients need to communicate with each other, traditionally every communication goes through the app server. For example, the driver sends his/her location to the app server every minute or every second. The rider will need to send a HTTP request to get the driver’s location from the server every minute or second. The smaller of the interval between the requests, the better the accuracy. The rider can get the driver’s location immediately whenever the driver sends his/her location to the server. However, this causes a lot of traffic to and from the server. In addition, all the polling requests from the rider’s device would consume the battery significantly and slow down the app or device itself.

In our project, GCM is used to send messages to the client’s devices. That means the driver sends GCM notification containing the location information to the GCM connection server. Then the information is sent to the rider. As a result, the rider will be able to view the driver’s locations in real time. Hence the clients’ real time communication is through the GCM connection server. Once the trip is completed or the rider cancels the request, the information will be saved to the app server accordingly. The benefit of using GCM is the scalability and performance advantages over the app server. There is also the battery life savings for the clients’ mobile devices.

3 The Application Architecture

Figure 1 is the architecture diagram for our ride sharing application. The system is a client-server model. There are two client side applications, one is for the rider, and the other
is for the driver. The rider can request a car by providing his/her preferred start location and the destination of the trip. The rider can also select vehicle type, view the cost of the trip, track the driver’s current location, pay and rate the driver after the trip. The driver can view all the trip requests from different riders within a predefined distance, select a request, and pick up the rider.

5) Email module: this module is responsible for sending four types of the email. a) sending an email to a rider or a driver once the registration is approved; b) sending a warning email to the driver if his/her rating is below a threshold; c) sending an email to notify a driver to close his/her account since the rating did not improve within a time period after the warning was issued; d) sending the trip receipt to the rider.

6) Data model module: this module defines the schema of all the data needed to be stored in MongoDB. It includes the user’s registration and trip details.

In the server side, there are total of six modules:

1) Admin module: this module is for the web server part of our project. It deals with search rider or driver, approves the driver’s registration, review all the trip data.

2) Rider module: this module is responsible for all the functionalities of rider.

3) Driver module: this module implements all the responsibilities of the driver.

4) Authentication module: this module consists of three parts, each part is to authenticating each type of user: driver, rider and admin.

Figure 2 Rider’s App

Once the driver accepts the trip request, the rider can view the driver’s current location.

Figure 4 and Figure 5 are screen shots on the rider’s app. The rider can see where the driver’s current location is on the map.
Testing

The developers have conducted white-box testing during the development of the app. Black-box testing has been used in integration testing and system testing. The scenario below has been repeatedly used to focus on the real-time communications between riders and drivers.

Scenario Testing:

- Rider1 makes trip1 request
- Rider2 makes trip2 request
- Driver1 is within 10 miles from the trip1’s start locations
- Driver2 is more than 10 miles far away from the trip2’s start locations
- Driver1 receives the trip1 request, but Driver2 does not
- Driver1 picks up the trip1’s order
- Driver2 moves closer to the trip2’s start location
- Now Driver2 will receive trip2’s request

The scenario described above is illustrated in Figure 6. We use the circle to represent the 10 miles range from the start location of the rider’s request.
The application demonstrated the correct results and all the trip information was recorded in the database correctly.

Figure 7 is the code segments in the driver’s app, sending the driver’s location to the rider’s app. Figure 8 is the code segments in the rider’s app, receiving the driver’s location from the driver’s app.

sendMyGeo(rider_gcm_token, driverGeo, heading){
  $f.gcm(
    key: FIREBASE_API_KEY,
    token: rider_gcm_token.token,
    data: {
      driverGeo: driverGeo,
      heading: heading
    },
  );
}

PushNotification.configure({
  onRegister: (gcm_token) => {
    this.setState({gcm_token});
  },
  onNotification: (notification) => {
    if(notification.driverGeo){
      var driverGeo = JSON.parse(notification.driverGeo);
      this.setState({driverGeo}, this.updateRegion);
      var driverHeading = parseInt(notification.heading);
      this.setState({driverHeading});
    },
  },
  senderID: "728367311402",
  popInitialNotification: true,
  requestPermissions: true,
});

5 Conclusions

This ride sharing application has been used to practice a number of the latest technologies. In this paper, we mainly introduced GCM to implement the real time communications between the rider and the driver. We have been conducted a number of testing to ensure a small set of functionalities were implemented correctly and efficiently. For our next version, we are interested in adding the reservation functionality so that the rider can request a trip for the next day. In the current version, all the trip requests are for the current time. A response from a driver must be within 15 minutes or the rider needs to make another trip request.

6 References


