**CMPE436 Term Project Report**

**Discount Helper**

**Elif Çalışkan 2016400183**

**1.Abstract:**

Because of the increasing inflation rates, it is very hard to find the desired product for a reasonable amount of money. In order to solve that problem, I though about creating an app that allows users to see nearby stores and their discounts. To add these discounts, an API might be used but in order to implement an addition mechanism, I divided the users into two parts: retailers and guests. For communication between the server and client, I used TCP sockets.

**2.Introduction:**

Since I had experience about writing Android apps, first I started to implement client part. First, there should be a selection between guest and retailer.

If the user has selected guest button, the user should see a map with some stores and his/her location on it. If the user selects one store and clicks on the eye button, then the discounts of that store can be seen.

If the user has selected retailer button, the user should see the list of his/her stores. Each store item has a location icon at the leftmost part. If this icon is clicked, the location of that map is shown on the map. If the user wants to delete that store, server looks to the database to see if this store has any other owner. If it was owned by another retailer, the store is deleted from the current retailer’s list. Else, the store is deleted permanently.

If the user wants to add a new store, user can click on the floating button with “+” sign on it. Then, a map opens with the user’s current location selected as default. If the user wants to change the store location, he/she can select a different location by tapping on the map. Then if the user clicks on “Add your shore here” button, a dialog opens so that the user can type in the desired name. After approving the change, newly created store is added to guest’s map and retailer’s store list.

There is also an option to partially own a store. By clicking on the floating button with search icon, the map opens with some different colored icons on it. The blue location icons represent the stores that are owned by the user, purples are owned by other retailers. If the user clicks on a store in the store list, the discounts of that store is listed. The retailer can add a discount by clicking on floating button with + icon. In the opening dialog, the name and discount amount are selected. Added discount can now be seen by guest users and other owners of that store.

**3.Approach/Methodology:**

In order to communicate with the server, I decided to use some codes for different functionalities. These are listed below:

**100** Add Retailer: With the typed username, email and password a retailer is added to database and assigned a new id to him/ her. If there is no file called “users.json” it is created and the newly created retailer is added. If it is accomplished, the server returns “100”.

**101** Add Store: With the typed name, selected latitude and longitude, the corresponding address and owner id’s list, a new store is created and with an assigned id, it is added to database. In order not to create any problem, the new id is computed by adding 1 to the maximum store id in the database. If there is no file called “stores.json” it is created and the newly created retailer is added. If it is accomplished, the server returns “101”.

**102** Add Discount: With the typed product name and selected percentage discount value, a new discount is created and an id is assigned to it. Then it is added to database. In order not to create any problem, the new id is computed by adding 1 to the maximum discount id in the database. If there is no file called “discounts.json” it is created and the newly created retailer is added. If it is accomplished, the server returns “102”.

**103** Login: By getting the typed name and password from the user and comparing with all the credentials in the database, it finds the corresponding username and if password is correct it returns “103”. Else it returns error code.

**104** Get All Stores: If the code is 104, the database in Stores.json is sent to client.

**105** Get Retailer’s Stores: With the given retailer id, the store that belong to the user with that id is returned.

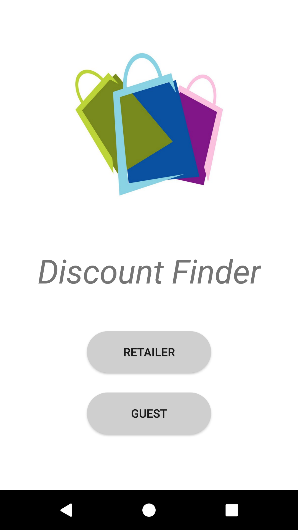
**106** Get Store’s Discounts: With the given store id, all the discounts that belong to the store with that id is returned.

**107** Delete Discount: With the given discount information, the discount in database is deleted. If it is accomplished, the server returns “107”.

**108** Delete Store: With the given store information, the database is checked. If there is another owner of this store, the store is deleted from that user’s list by deleting user’s id from owner list of that store. Else, the store is deleted permanently from the database with every discount in it. If it is accomplished, the server returns “108”.

**109** Add Ownership: With given store information and retailer id, the owner list is updated by adding that id. If it is accomplished, the server returns “109”.

**404** Error Code: If any functionality is not satisfied, it returns “404”.

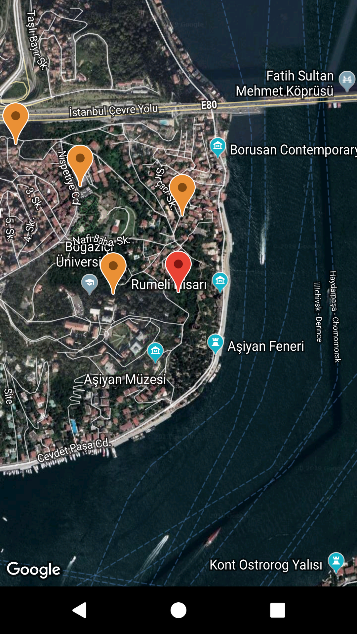
**3.1. Client:**

**3.1.1 MainActivity:**

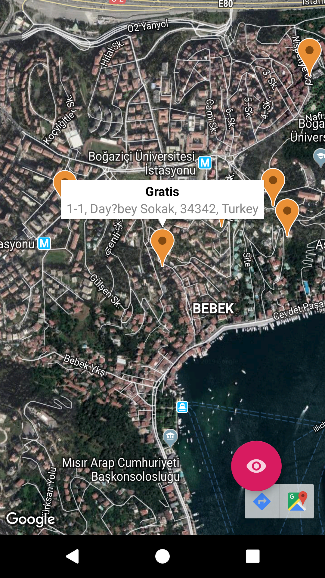
In this activity, the user selects between guest or retailer. For that, I used two buttons for each option. For design purposes, I added a shopping bag image.

**3.1.2: Maps Activity**

In order to use a map, I used Google Maps API. MapsActivity is used for many purposes. For understanding the desired functionality, I used intents. I added String fields to intents such as “type” with the values of “guest” and “retailer”.

**3.1.2.1 Guest functionality:**

If the user is a guest, the map is filled with location icons of stores. If a store is clicked and eye floating button is pressed, the discounts of that store appears. In order to create that functionality first, I checked the intent for type.

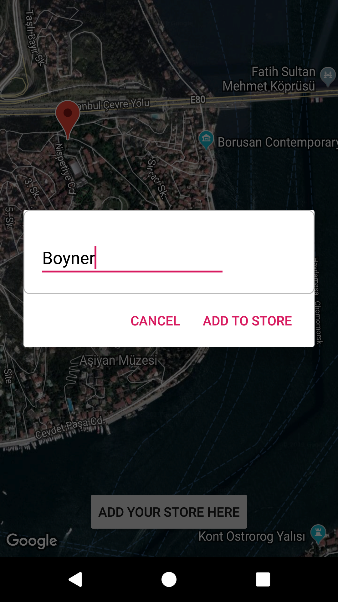
If the type equals to “guest”, then I call getAllStores() function that sends “104” message to server to get all the stores. In order that to be responsive, the a new request is created every 3 seconds using a Handler and its postDelayed() function. This request is only called if the activity is visible.

Also, I added onClickListener to markers on the map. If any marker is clicked, the eye button becomes visible. And their address can be seen in the description of that icon besides the store name. If any store is clicked, the camera moves so that the location icon is in the middle of the map using moveCamera() method.

If the eye floating action button is clicked, DiscountActivity is started with an explicit intent.

**3.1.2.2. Retailer Show Store Functionality:**

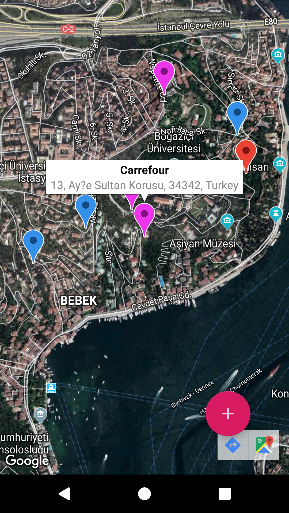
If the type is equal to “retailer” and there is store information in intent, a marker with the desired longitude and latitude is created. Also, the address is added in a snippet. Since this is a basic functionality, I will not add a descriptive image.

**3.1.2.3 Retailer Add Store Functionality:**

If the user type is retailer and there is no store information in the intent, the map opens containing only the current location marker. This marker shows user’s location and changes its position if the user moves (using LocationListener). If the user clicks the map, the map gets erased and a new marker is created at the clicked location. Since the map has an onClickListener, the latitude and longitude can be achieved from parameter LatLng arg0.

After creating the marker, user can click on “Add your store here” button at the bottom. A new dialog fragment is created with an editText. I created a custom layout for that dialog. The reason for using a dialog fragment rather than a regular one is that it is easier to create, and the desired functionality is very shallow for a regular fragment. After the user clicks add button in dialog, the MapsActivity is finished and RetailerActivity with newly added store is opened.

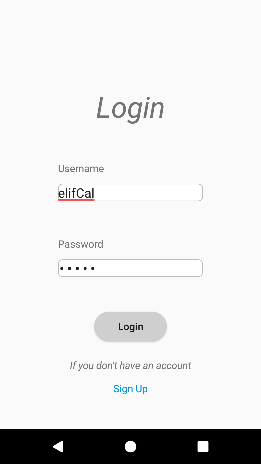
**3.1.2.4 Retailer Add Ownership Functionality:**

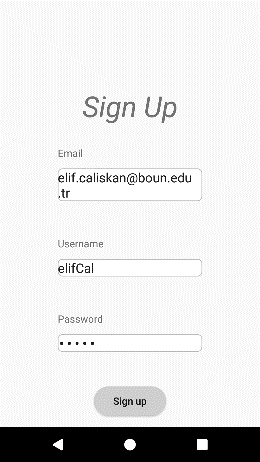
If the user type is retailer and there is a true search boolean in the intent, map opens containing every store. But in addition to “104” request to retrieve all stores, “105” message is sent to server in order to obtain only the stores that belong to that user. The stores that belong to the user are represented with blue markers, the other stores are represented with purple and the current user location is represented with a red marker.

If the user clicks on a purple marker (using onClickListener), a floating action button with a + sign appears at the right bottom of the map. If the user clicks on that button, MapsActivity is finished and RetailerActivity is created again with the added store.

At each functionality except 3.1.2.2, a new request is created to get all stores with code “104” at each 3 seconds.

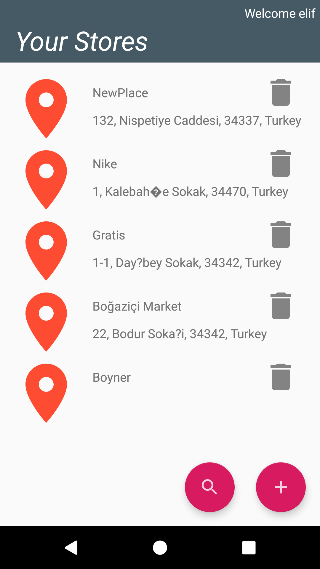
**3.1.3 LoginActivity:**

In this activity, there are two editTexts: one for username and one for password. For password, I used password inputType. For login button background, I used a custom drawable. If the user clicks on login button, I first check if every editText is nonempty. If there is an empty ****editText, I create a warning in editTexts. If each value is filled, I send a “103” request to server. If the login is accomplished and the message from server is “103”, I create an intent to RetailActivity. To go to SignupActivity, the textView at the bottom is used.

**3.1.4 SignupActivity:**

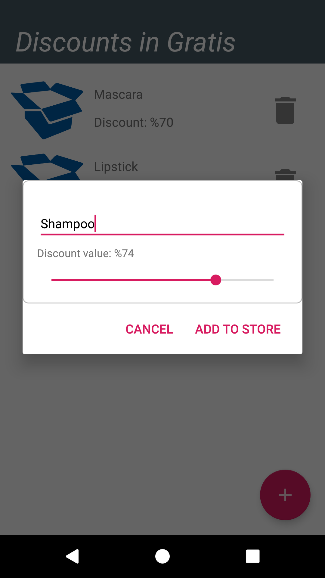
This activity has three editTexts. If every editText is filled and sign up button is clicked, the request of “100” is sent to server. If the message from server is “100” and it is added to database, I create an intent to LoginActivity.

**3.1.5 RetailerActivity:**

This activity has a custom adapter for showing store items. StoreAdapter has interfaces for clicking location icon, clicking store or delete button. Since retailer activity implements these interfaces, the functions are also implemented. If the user clicks on location, the MapActivity is opened(3.1.2.2). If the user clicks on a store, DiscountActivity is opened. If the user tries to delete a store, a request with “108” code is sent to server. After deleting, the store list is received from the server in order to visualize the changes.

For receiving stores of a certain retailer, “105” coded request is sent to server. After getting that list, adapter’s dataset is updated and notifyDatasetChanged() function is called. There are two floating buttons at the bottom of the layout. If the user clicks on search icon, the MapsActivity opens with add ownership functionality (3.1.2.4). If the user clicks on add button, MapsActivity opens with add store functionality (3.1.2.3).

**3.1.6 DiscountActivity:**

 This activity has a custom adapter for showing discount items. DiscountAdapter has an interface for deleting discount. If the user tries to delete a discount, a request with “107” code is sent to server. After deleting, the discount list is received from the server in order to visualize the changes.

For receiving discounts of a certain store, “106” coded request is sent to server. After getting that list, adapter’s dataset is updated and notifyDatasetChanged() function is called. “106” coded request is sent each 3 seconds in order to be responsive to changes. If the user is a retailer there is an add floating button. If the user clicks on it, a dialog fragment opens with an editText and a seekBar.

**3.1.7 MessageSender:**

For communicating with the server, I used AsyncTask. In each operation, it creates a new socket with the given ip from AWS and port 6000. The string in the parameter is sent to server by print writer. Then it waits for a message using input stream reader. According to the message, it decides what to return as a String.

**3.2 Server**

**3.2.1 Database:** For databases, I used three json files: users.json, stores.json, discounts.json

**3.2.2 Communication:** I used TCP sockets with port 6000. I created one serverSocket and using that server socket I created a new socket by waiting for a message. After creating a socket, I create a ServerThread for finding and applying the appropriate functionality.

**3.2.3 Models**

**3.2.3.1 User:** Each retailer has username, email, id and password. After sign up operation, a new Retailer is created.

**3.2.3.2 Store:** Each store has store name, id, ownerIds list, latitude, longitude and store address.

**3.2.3.3 Discount:** Each discount has a product name, discount id, store id.

**3.2.4 Server:**  After creating a socket for every server thread, each thread starts to work. After receiving the message with input stream reader and reading it with buffered reader, the message is checked according to the codes (explained above). Each functionality is done inside the model class in a static function. Each function returns a result string. This string is sent by PrintWriter.

**3.2.5 BinarySemaphore:** In every write operation, I lock the appropriate mutex. In Store class, I used storeMutex etc. But read operations are not locked.

**4.Demonstration/Experiments:**

After writing the code, it was crucial to test it on different devices simultaneously in order to check if the code works correctly. But I had a difficulty while running on a real device since the Wifi was a shared one. After switching to hotspot, I solved that problem and had the opportunity to test the responsiveness of the app.

Before the demonstration in class, I tested the app with emulator and my own device. I opened the guest page in emulator and add store functionality on my device. After adding a store in real device, I saw a new marker appear in emulator too. Then I clicked on a store in emulator and opened their discount list. In real device when I added a new discount, it also appeared on discount page of guest user too. In class, with my device and emulator from a friend’s computer, we tested each functionality.

**5. References:**

1) <https://piazza.com/boun.edu.tr/fall2019/cmpe436/resources>

2) <https://www.geeksforgeeks.org/socket-programming-in-java/>

3) <https://developer.android.com/>

4) <https://www.youtube.com/watch?v=9Rwopuah2Q0&t=1s>

**6. Appendix:**

Since there are many lines of code, I am writing the crucial files. Projects are added to piazza as a zip format.

Server side: DiscountServer

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.LinkedList;

import java.net.ServerSocket;

import java.net.Socket;

public class DiscountServer {

private static LinkedList<Server> threads = new LinkedList<>();

public static void main(String[] args) {

DiscountServer server = new DiscountServer();

server.start(6000);

}

private void start(int port) {

try {

ServerSocket serverSocket = new ServerSocket(port);

while (true) {

Socket socket = serverSocket.accept();

Server thread = new Server(socket);

threads.add(thread);

thread.start();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

Server thread:

import com.google.gson.\*;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

public class Server extends Thread{

private static Socket s;

private static BufferedReader br;

private static InputStreamReader isr;

private static String message;

private static PrintWriter pw;

static JSONParser parser = new JSONParser();

Server(Socket socket) {

this.s = socket;

try {

isr = new InputStreamReader(s.getInputStream());

br = new BufferedReader(isr);

pw = new PrintWriter(s.getOutputStream());

} catch (Exception e) {

e.printStackTrace();

}

}

@Override

public void run() {

try {

System.out.println("message is :");

message = br.readLine();

System.out.println(message);

String outMessage = "404";

GsonBuilder builder = new GsonBuilder();

builder.setPrettyPrinting();

Gson gson = builder.create();

String first = message.substring(0, 3);

message = message.substring(3);

if(first.equals("100")) {

outMessage = Retailer.createRetailer(message, parser, gson);

}

else if(first.contentEquals("101")) {

outMessage = Store.createStore(message, parser, gson);

}

else if(first.equals("102")) {

outMessage = Discount.createDiscount(message, parser, gson);

}

else if(first.equals("103")) {

outMessage = Retailer.login(message, parser, gson);

}

else if(first.equals("104")) {

outMessage = Store.getAllStores(parser, gson);

}

else if(first.equals("105")) {

outMessage = Store.getStoresOfRetailer(message, parser, gson);

}

else if(first.equals("106")) {

outMessage = Discount.getDiscountOfStore(message, parser, gson);

}

else if(first.equals("107")) {

outMessage = Discount.deleteDiscount(message, parser, gson);

}

else if(first.equals("108")) {

outMessage = Store.deleteStore(message, parser, gson);

}

else if(first.equals("109")) {

outMessage = Store.addOwnership(message, parser, gson);

}

//Write JSON file

System.out.println(outMessage);

//pw = new PrintWriter(s.getOutputStream());

sendMessage(outMessage);

br.close();

s.close();

}

catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void sendMessage(String msg) {

pw.write(msg+"\n");

pw.flush();

pw.close();

}

}