**Howl: Restaurant Search App**

By Group 4:

Cuong Thinh “Eric” Dao (300275805)

Samuel Navarro (300291991)

Scott Myron (300283742)

# Introduction

Our application is based on the popular restaurant searching application, Yelp [1]. The name of our application, Howl, was chosen because it is a synonym of the verb “yelp”. The initial goal was to produce an application that would use the Yelp API to recommend local restaurants, search for restaurants based on search terms and/or filters, and provide a map view of the locations of these restaurants. Due to issues using the Yelp API with Java it was necessary to change to Foursquare’s API [2] (see Errors section). However, Foursquare is a web application that provides similar services to Yelp. Thus, the main functionality of our application was not hindered by the change. We also created a custom app logo using the free online tool Free Logo Design [3].

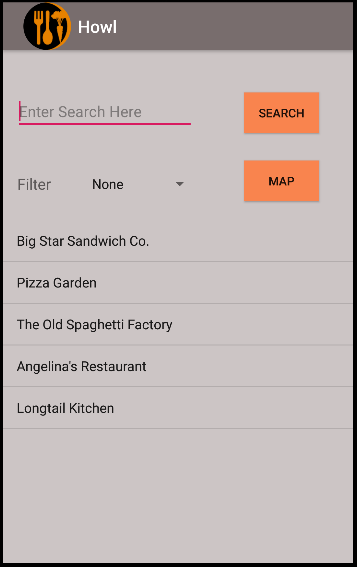
Figure 1 displays the opening screen of the Howl app when the device is geographically located in Douglas College, New Westminster. By default, the app presents a list of five local restaurant recommendations as provided by the Foursquare API. Note that the final version of Howl does not retrieve the geographical location of the device from its GPS but rather must be hardcoded in (see Errors section).

Figure . Howl app opening screen with device located at Douglas College

As mentioned, a user can search for restaurants by typing terms in the search text box (Figure 2). The resulting list will provide a scrollable selection of restaurants that fit the search parameters and are located within a 10Km radius of the device. Similarly, a user can search by using preestablished filters as seen in Figure 3. The five available filters are Coffee, Chinese, Japanese, Dessert, and Fast Food. The search feature also works if a user decides to provide both a search term and a filter. For example, a user could decide to set the filter for “Fast Food” but also provide a search term “burger” to find fast food restaurants that specialize in hamburgers.

It is important to keep in mind that categorization of restaurants in Howl is completely dependent on how Foursquare categorizes specific locals. For example, at Douglas College there is a Tim Horton’s coffee shop in the cafeteria. Although this venue does appear if we provide the search term “Tim Horton’s”, it does not show up if we simply filter for “Coffee”. Meaning that in Foursquare’s database, this Tim Horton’s shop has not been marked with the Coffee Shop ID. Hence the discrepancy with this venue and a couple of others.

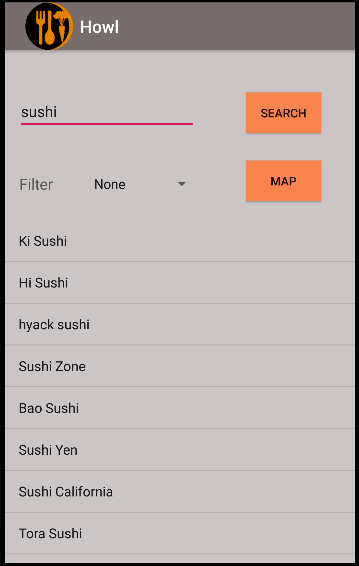
The final feature of the Howl app is the ability to display restaurant search results in a map format. Figure 4 shows an example of such a search. If a user taps on a marker they will see the name of the given local in a small pop-up window above the marker. The user can use the back button on the device to return to the main window. Note that the map markers are only updated when the application is first opened and when the search button is clicked. If the filter is changed and/or a search term is added but the Search button is not clicked, the restaurant markers will be those of the latest search.

Figure . Howl app restaurant search with user provided term "sushi" and device located at Douglas College

# Procedure

When the Howl app starts it runs the MainActivity class. In this class’s onCreate() method we initialize the UI’s View elements (e.g., btnSearch, txtSearch, etc.) and fetch the Client ID and API Key stored in string.xml. More importantly, the recomRestaurants() method is executed. recomRestaurants() is a custom method that executes the API request for the five recommendations of local restaurants. The URL request within the method is defined as follows:

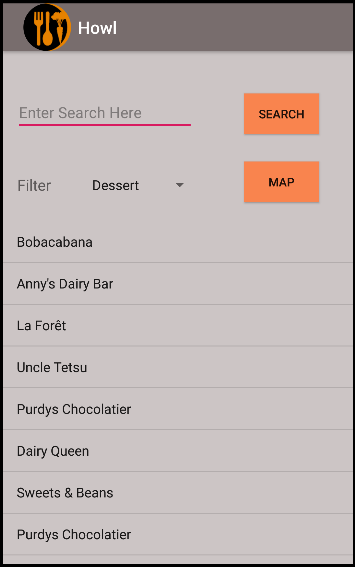


Figure . Howl app restaurant search with filter term "Dessert" and device located at Douglas College

String url = ***BASE\_URL*** + inputUrl + **"&ll="** + latitude + **","** + longitude + **"&client\_id="** + *clientID* + **"&client\_secret="** + *apiKey* + ***API\_VERSION***;

Where ***BASE\_URL*** is defined as https://api.foursquare.com/v2/venues/ and is common to all Foursquare API requests as defined by their documentation. The inputUrl is provided as an input to the method and is defined to be the string explore?section=food&limit=5, which are the parameters that will limit the API response to 5 venues of type food. Note that we need to specify section=food with Foursquare. Their web application is a service to find interesting places to visit in an area, not just restaurants. The **ll** URL parameter is the longitude and latitude of the device, which are provided as inputs to the recomRestaurants(). The **client\_id** and **client\_secret** parameters are the Client ID and API Key respectively, which are global variables. Finally, ***API\_VERSION***

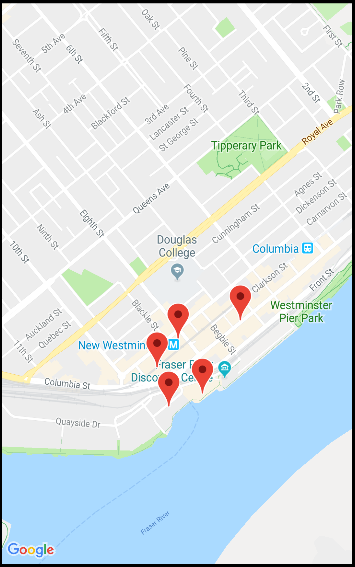
is defined as the string &v=20181115 where the URL parameter v is the API’s version and is writen in a YYYYMMDD date format. This last parameter is also common amongst all Foursquare API requests.

Figure . Howl app restaurant search in Google Map display with device located at Douglas College

recomRestaurants()consists of 49 lines of code, but the most relevant portion is the following 10 lines within the definition of onResponse():

**for**(**int** i = 0; i < restaurants.length(); i++){  
 JSONObject restaurant = restaurants.getJSONObject(i).getJSONObject(**"venue"**);  
 JSONObject location = restaurant.getJSONObject(**"location"**);  
  
 **name** = restaurant.getString(**"name"**);  
 **lat** = location.getDouble(**"lat"**);  
 **lng** = location.getDouble(**"lng"**);  
 **result**.add(**new** Restaurant(**name**, **new** LatLng(**lat**, **lng**)));  
}

Prior to the for-loop we have already filtered the JSON response to the point where two JSON objects can be declared, restaurant and location. From these objects, we can obtain the restaurant names, latitudes, and longitudes which will be used as parameters to instantiate Restaurant objects. These objects are iteratively added into a **result** list of type List<Restaurant>. Restaurant is a custom class created to hold relevant information to each local. This would be the class where attributes could be added to also hold information such as a venue’s rating or address. These could then be displayed in Howl’s search results. However, due to issues with adding sub-elements to a ListView, this was not implemented (see Errors section). Note that rendering the restaurant list into the ListView of the MainActivity is performed by updateListView(). We will describe this method later in this section.

After a user has opened Howl, they can start searching for venues by filters or search terms. We will first describe how a filter is created. Filters are relevant to explaining how searching by terms works. The following is the click-listener for the Search button:

**btnSearch**.setOnClickListener(**new** View.OnClickListener() {  
 **@Override  
 public void** onClick(View v) {  
  
 String filter = setFilter();  
  
 *//Search for restaurants by the given search terms and filters within 10Km.* String url = **"search?"** + filter +

**"&radius=10000&query="** + **txtSearch**.getText().toString();  
 searchRestaurants(**latitude**, **longitude**, url);  
  
 }  
});

A user selects a filter via the Spinner in the main window. When a user taps the Search button, the variable filter is set by setFilter(). This method is a simple switch that will look at the selection made in the Spinner and output the corresponding categoryId parameter value. Note that there is always a default categoryId for “Food” since the Foursquare API will search for none food related venues otherwise. Once setFilter() has provided a categoryId we execute searchRestaurants(), which uses the filter as part of its input. The URL request for a Foursquare venue search is very similar to that of a venue recommendation. The only differences are the addition of the radius, and query parameters. These fields set the radius of the search and the query terms respectively. In addition, we replace section with categoryId for similar results. The definition of searchRestaurants()is nearly identical to that of recomRestaurants()with a for-loop adding restaurants to a list.

As mentioned before, rendering the restaurants into our ListView is performed by updateListView(). This method is called by both recomRestaurants()and searchRestaurants()with the corresponding restaurant list as parameter. The main body of updateListView()consists of:

*itemsList* = **new** ArrayList<>();  
*locations* = **new** ArrayList<>();  
  
**for** (Restaurant restaurant:  
 restaurants) {  
 **if** ((restaurant.get\_name() != **null**) && (restaurant.get\_latlng() != **null**))  
 {  
 *itemsList*.add(restaurant.get\_name());  
 *locations*.add(restaurant.get\_latlng());  
 }  
}  
  
ArrayAdapter<String> adapter = **new** ArrayAdapter<>(**this**,  
 android.R.layout.***simple\_list\_item\_1***, *itemsList*);  
**lstResults**.setAdapter(adapter);

The purpose of this method is twofold. The main task is to use the adapter to update our ListView with the names of the venues in the input list, restaurants. The second task is to also add the restaurants locations to the static global list *locations*. This list will be called in the MapActivity to render the markers for the venue locations.

[WRITE MAP SECTOIN HERE]

# Results

# Errors

# References

1. <https://www.yelp.ca/vancouver>
2. <https://developer.foursquare.com/docs>
3. <https://www.freelogodesign.org/>