Fête: find event-venues together everywhere

Allyson Mackay, Michael Nguyen, Anubhav Saxena

Updated pitch:

What is Fête?

-finding events for parties and other occasions (gatherings)
-not needing to have your own large space because you can access them readily with this new app
-shared venues locations (such as a backyard with pool, bbq, deck or a private pond)
-inviting new friends and people you know

Who are the user groups?

-Individuals, Organizations, people looking for a particular space who do not own one or have access because it is cost prohibitive to rent a formal venue.
-Urban dwellers
-Empty nesters
-People who are invested in the sharing economy movement (co-lab work spaces)
-People looking to contribute to their local neighborhood
-People looking to monetize their owned real-estate
-People who want to get a pool but won't use it a lot and want to get their money's worth

Why is it important?

-Space is expensive and limited resource
-Promotes sharing and community connectivity
-Environmentally friendly. Promotes resource reuse as opposed to individual consumption

What are the competitors?

http://www.appvita.com/2012/06/21/eventup-find-a-venue-for-your-event/
All vacation rental home apps (air bnb, vrbo, craigslist)
Formal Party Venues without Apps (banquet halls, restaurants, bars, etc)

Why is your idea better?

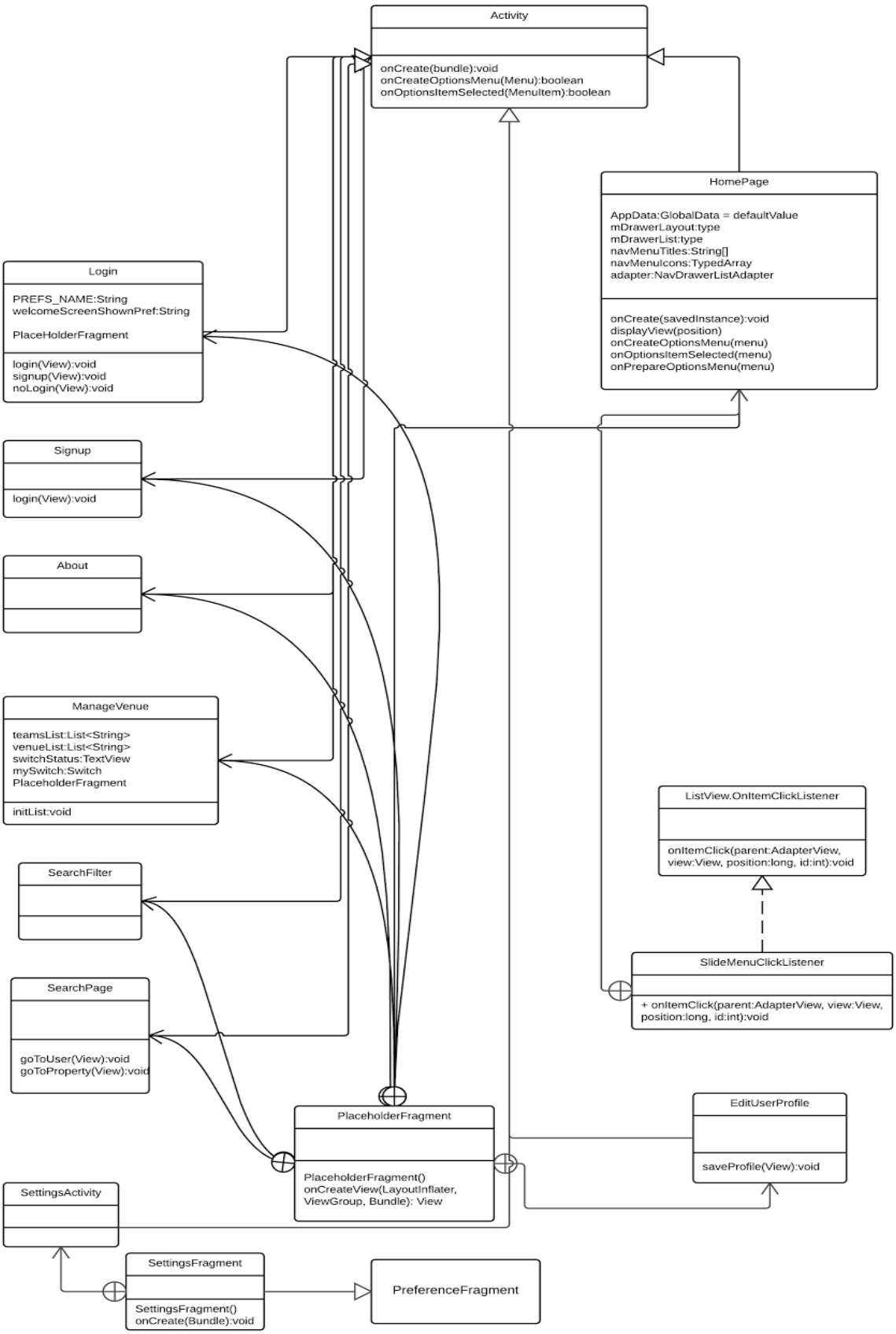
Our app is neighborhood-based. It encourages sharing of local resources with local people and benefits both those with and without physical space in their home. Fete contributes to community building by building trust between users and establishing relationships across different user groups. We will start to grow our member-base by offering free rentals or catering credits to new users (similar to uber/lyft giving away free rides).

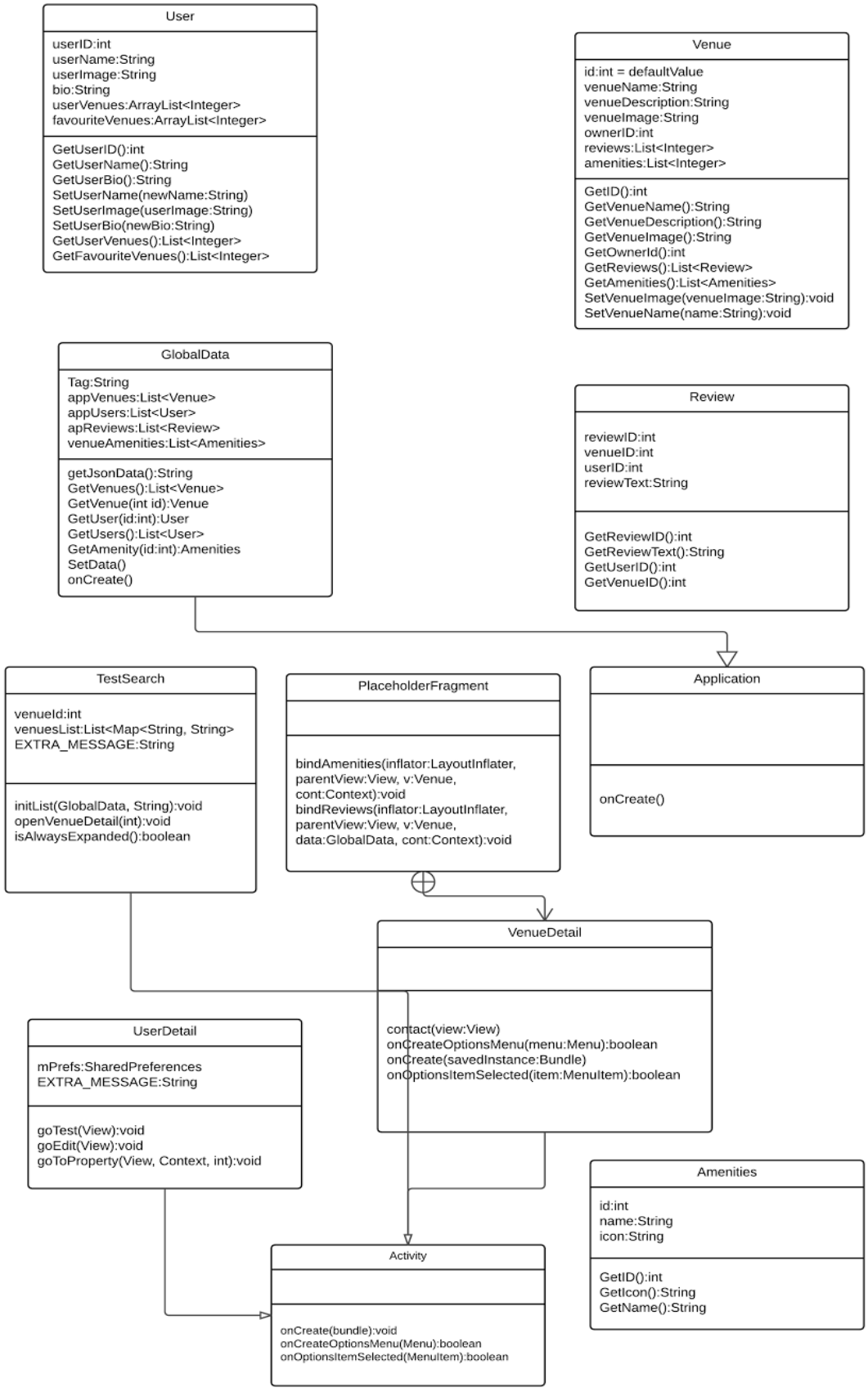
Major updates of initial app goals

Overall, our team was able to implement all features we listed in our list of initial goals.
We were able to develop a robust search feature that involved use of an adapter and data from a json file we created, which we decided to use instead of shared preferences. We were able to

use programmatically altering views, especially in our main/home activity. Furthermore, our slide-out drawer glues together our app and assists in navigation and overall user experience.

UML Diagrams


Activity
onCreate(bundle):void
onCreateOptionsMenu(Menu):boolean
onOptionsItemSelected(MenuItem):boolean
HomePage
AppData:GlobalData = defaultValue
mDrawerLayout:type
mDrawerList:type
navMenuTitles:String[]
navMenuIcons:TypedArray
adapter:NavDrawerListAdapter
onCreate(savedInstance):void
displayView(position)
onCreateOptionsMenu(menu)
onOptionsItemSelected(menu)
onPrepareOptionsMenu(menu)
Login
PREFS_NAME:String
welcomeScreenShownPref:String
PlaceHolderFragment
login(View):void
signup(View):void
noLogin(View):void
Signup
login(View):void
About
ManageVenue
teamsList:List<String>
venueList:List<String>
switchStatus:TextView
mySwitch:Switch
PlaceholderFragment
initList:void
SearchFilter
SearchPage
goToUser(View):void
goToProperty(View):void
ListView.OnItemClickListener
onItemClick(parent:AdapterView,
view:View, position:long, id:int):void
SlideMenuClickListener
+ onItemClick(parent:AdapterView, view:View,
position:long, id:int):void
PlaceholderFragment
PlaceholderFragment()
onCreateView(LayoutInflater,
ViewGroup, Bundle): View
EditUserProfile
saveProfile(View):void
SettingsActivity
SettingsFragment
SettingsFragment()
onCreate(Bundle):void
PreferenceFragment


User
userID:int
userName:String
userImage:String
bio:String
userVenues:ArrayList<Integer>
favouriteVenues:ArrayList<Integer>
GetUserID():int
GetUserName():String
GetUserBio():String
SetUserName(newName:String)
SetUserImage(userImage:String)
SetUserBio(newBio:String)
GetUserVenues():List<Integer>
GetFavouriteVenues():List<Integer>
Venue
id:int = defaultValue
venueName:String
venueDescription:String
venueImage:String
ownerID:int
reviews:List<Integer>
amenities:List<Integer>
GetID():int
GetVenueName():String
GetVenueDescription():String
GetVenueImage():String
GetOwnerId():int
GetReviews():List<Review>
GetAmenities():List<Amenities>
SetVenueImage(venueImage:String):void
SetVenueName(name:String):void
GlobalData
Tag:String
appVenues:List<Venue>
appUsers:List<User>
apReviews:List<Review>
venueAmenities:List<Amenities>
getJsonData():String
GetVenues():List<Venue>
GetVenue(int id):Venue
GetUser(id:int):User
GetUsers():List<User>
GetAmenity(id:int):Amenities
SetData()
onCreate()
Review
reviewID:int
venueID:int
userID:int
reviewText:String
GetReviewID():int
GetReviewText():String
GetUserID():int
GetVenueID():int
TestSearch
venueId:int
venuesList:List<Map<String, String>
EXTRA_MESSAGE:String
initList(GlobalData, String):void
openVenueDetail(int):void
isAlwaysExpanded():boolean
PlaceholderFragment
bindAmenities(inflator:LayoutInflater,
parentView:View, v:Venue,
cont:Context):void
bindReviews(inflator:LayoutInflater,
parentView:View, v:Venue,
data:GlobalData, cont:Context):void
Application
onCreate()
VenueDetail
contact(view:View)
onCreateOptionsMenu(menu:Menu):boolean
onCreate(savedInstance:Bundle)
onOptionsItemSelected(item:MenuItem):boolean
UserDetail
mPrefs:SharedPreferences
EXTRA_MESSAGE:String
goTest(View):void
goEdit(View):void
goToProperty(View, Context, int):void
Amenities
id:int
name:String
icon:String
GetID():int
GetIcon():String
GetName():String
Activity
onCreate(bundle):void
onCreateOptionsMenu(Menu):boolean
onOptionsItemSelected(MenuItem):boolean

<start individual write-ups>

Michael

Here is a typed-out description of the major components of the code for the Venue Detail page.

1. **GlobalData** is our data container containing all data we use in the app.

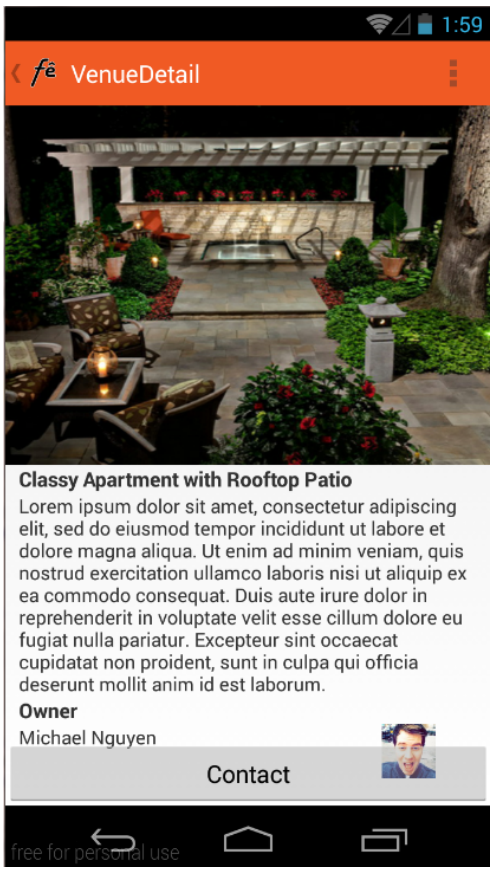
2. **getArguments()** gets data from the previous page, which in this case is venueId

3. We then use the venueId to find the venue we need to show by searching the **GlobalData**
   Venue venue = data.GetVenue(venueId);

4. We then pick up elements of our UI that we want to fill by using
   TextView venueDetail = (TextView)rootView.findViewbyID(R.id.venueDetail);

5. Once we have the UI element as object(venueDetail), we type
   venueDetail.setText(venue.GetVenueDetail());

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    final Context cont = getActivity();
1   GlobalData data = (GlobalData)(this.getActivity().getApplication());
    Bundle inputExtra = getArguments();  2
    int venueId = inputExtra.getInt("venueId");
    View rootView = inflater.inflate(R.layout.fragment_venue_detail, container, false);
3   Venue venue = data.GetVenue(venueId);
    TextView venueNameView = (TextView)rootView.findViewById(R.id.venueName);
    venueNameView.setText(venue.GetVenueName());
4   TextView venueDetailView = (TextView)rootView.findViewById(R.id.venueDetail);
5   venueDetailView.setText(venue.GetVenueDescription());
    TextView ownerNameView = (TextView)rootView.findViewById(R.id.userNameVD);
    int ownerId = venue.GetOwnerID();
    User user = data.GetUser(ownerId);
    ownerNameView.setText(user.GetUserName());
    ImageView userImageView = (ImageView)rootView.findViewById(R.id.userImageVD);
    int userImageId = getResources().getIdentifier(user.GetUserImage(), "drawable", cont.getPackageName());
    userImageView.setImageResource(userImageId);
    ImageView venueImageView = (ImageView)rootView.findViewById(R.id.imageView);
    int resID = getResources().getIdentifier(venue.GetVenueImage(), "drawable", cont.getPackageName());
    venueImageView.setImageResource(resID);
    return rootView;
}
```



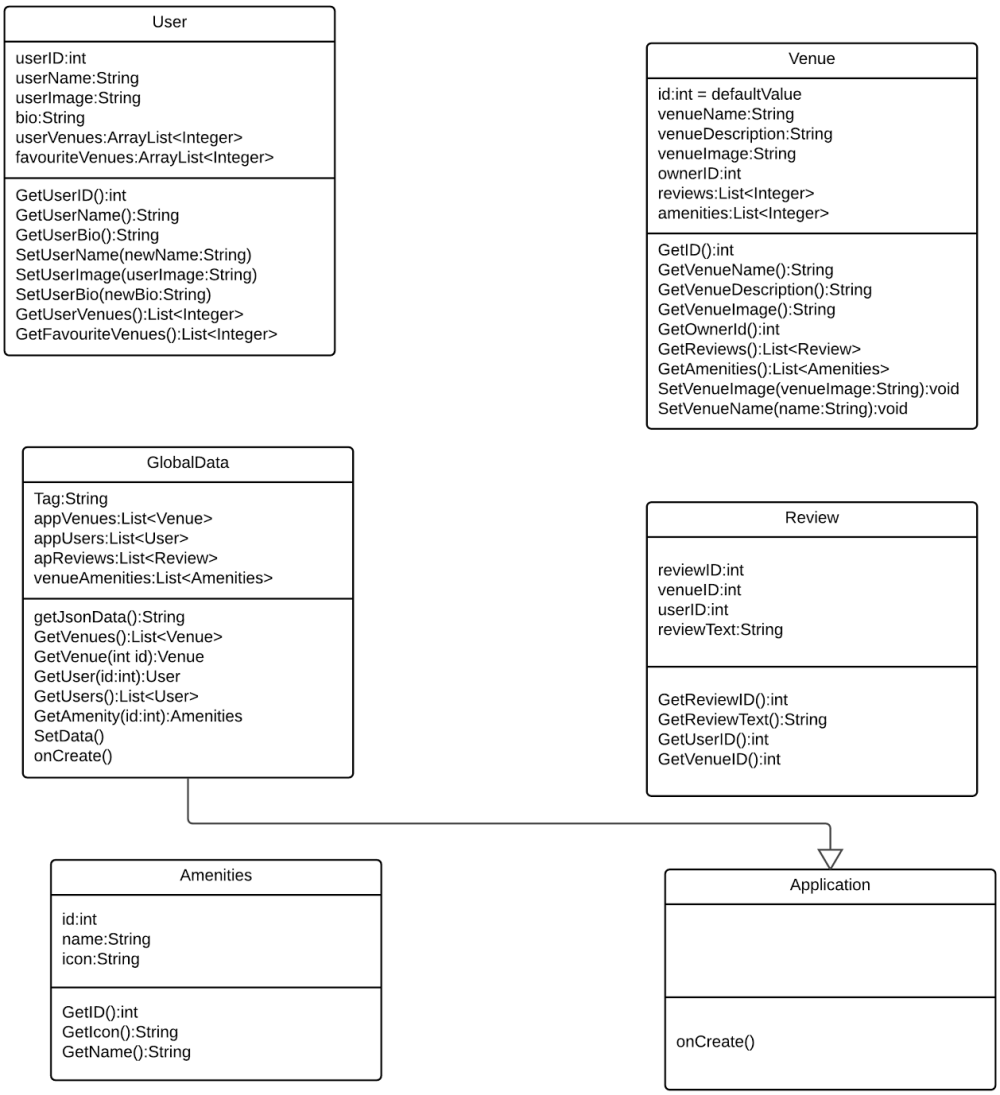
Anubhav

GlobalData class is a class inheriting from the Android application class. The android application class is inherently singleton and maintains its state throughout the lifecycle of the application. The class is used to provide functionality that the database layer of the application would have provided. It interacts with the datastore which in our case is a JSON file called fete.json stored in raw folder in resources.

The Global data class is initialized with the data only once when the application begins.

```
    @Override
    public void onCreate(){
        super.onCreate();
        //Application will initialize global data only once during application start
        SetData();
    }
```

First the onCreate of the base class is called, and then the SetData method is called which reads the data from the fete.json file and fills the corresponding objects of the code.




```
"venues": [
  {
    "venueID": 1,
    "venueName": "Welcoming house on Fuller Road",
    "venueImage": "poolimage0",
    "venueDescription": "Lorem ipsum dolor sit amet, consectetu
    "ownerID": 1,
    "venueAmenities":[
    {"amenityID":1},
    {"amenityID":2}
    ]
  },
```

```
 */
public class Venue {
    /*all the attributes defined about the ven
    so that they can be mapped easily. so each
    Also notice that all the variables are pri
    private int id;

    private String venueName;

    private String venueDescription;

    private String venueImage;

    private int ownerID;

    private ArrayList<Review> reviews;

    private ArrayList<Amenities> amenities;
```

The Venue class is similar instructure to the Venuw objects in the json file so that they can be mapped easily.
The SetData function reads the json file at the start of the application. In the code below, we first pick up all the arrays corresponding to users, reviews, venues, amenities and then iterate upon them to fill up their corresponding java classes and adding them to list.

```
public void SetData(){
    String jsonString = getJsonData();

    try {
    if (appVenues == null || appUsers == null){
        appVenues = new ArrayList<Venue>();
        appUsers = new ArrayList<User>();
        appReviews = new ArrayList<Review>();
        JSONObject jsonObject = new JSONObject(jsonString);
        JSONArray venueArray = jsonObject.getJSONArray("venues");
        JSONArray userArray = jsonObject.getJSONArray("users");
        JSONArray reviewsArray = jsonObject.getJSONArray("reviews");
        JSONArray amenitiesArray = jsonObject.getJSONArray("amenities");
        SetAmenities(amenitiesArray);

        for (int i=0; i<reviewsArray.length(); i++){
            JSONObject review = reviewsArray.getJSONObject(i);
            int reviewID = review.getInt("reviewID");
            int userID = review.getInt("userID");
            int venueID = review.getInt("venueID");
            String message = review.getString("reviewText");
            Review r = new Review(reviewID, userID, venueID, message);
            appReviews.add(r);
        }
```

I will explain the the insertion of Venue item, rest is similar to it.

```
JSONArray venueArray = jsonObject.getJSONArray("venues");
```

We first get the json array into a java JSONArray object called venueArray.

```
for(int i=0; i<venueArray.length(); i++){
    JSONObject venue = venueArray.getJSONObject(i);
    int ownerID = venue.getInt("ownerID");
    int venueID = venue.getInt("venueID");
    String venueImage = venue.getString("venueImage");
    String venueName = venue.getString("venueName");
    String venueDescription = venue.getString("venueDescription");
    JSONArray amenityList = venue.getJSONArray("venueAmenities");
    ArrayList<Amenities> amenities = new ArrayList<~>();
```

We iterate through each member of the venueArray. Each member is stored in venue JSONObject. We then extract the values from the object and fill them in seperate java objects. The venueArray contains another array called venueAmenities which includes a list of amenities that a venue can have.

```
            String venueDescription = venue.getString("venueDescription");
            JSONArray amenityList = venue.getJSONArray("venueAmenities");
            ArrayList<Amenities> amenities = new ArrayList<~>();
            for(int j=0;j<amenityList.length();j++){
                JSONObject amenity = amenityList.getJSONObject(j);
                int amenityId = amenity.getInt("amenityID");
                Amenities tempAmenity = venueAmenities.get(amenityId);
                amenities.add(tempAmenity);
            }
            ArrayList<Review> venueReviews = new ArrayList<~>();
            for(int j=0;j<appReviews.size();j++){
                Review tempReview = appReviews.get(j);
                if (tempReview.GetVenueID()==venueID){
                    venueReviews.add(tempReview);
                    break;
                }
            }
            Venue v = new Venue(venueID, venueName, venueDescription, venueImage, ownerID, venueR
            appVenues.add(v);
        }
```

We perform the same JSONArray operations on amenities and reviews. We match the amenityId present in the amenityList to get the corresponding Amenity object and add it to the amenities list. Similarly we make a venueReviews list which is essentialy a List<Review>. Once we have all the required objects that needs to be passed, we pass them to a new Venue object using parameterized constructor. This forms association between the Venues, Reviews, and Amenities which form the data base of the app. Any changes in the json file triggers the corresponding change in the code.

Allyson Mackay

Search and ListView Adapter for Search Results

I implemented search in the app using the SearchView widget. In order to search, 3 things are required: a searchable configuration, a searchable activity, and a search interface. The searchable configuration defines aspects of the search UI and is called searchable.xml and is required to be in the res/xml folder with the required

```
<?xml version="1.0" encoding="utf-8"?>

<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
    />
```

Figure 1 Searchable.xml

android:label=”@string/app_name”. The label is then added to list of searchable items in the system settings. The search activity that initiates the searchable activity in this case is the HomePage.activity which will send the query intent to the search results activity. The HomePage activity is a good choice for initiating the searchable activity since the app makes use of a navdrawer on the home page and all fragments reached via the navdrawer will also support search. This is because the navdrawer changes the fragment and not the activity. In order to let android know that HomePage activity is sending queries searchable activity, it is necessary to declare it as such in Android Manifest using the following metadata for the HomePage activity.

```
<activity
    android:name=".HomePage"
    android:label="Fete" >
    <meta-data
        android:name="android.app.default_searchable"
        android:value=".TestSearch" />
</activity>
```

Figure 2 HomePage.activity Android Manifest

The next step is to add the search interface to the app using the SearchView widget. The SearchView widget is a class provided by Android that puts an editTextView widget in the action bar with the magnifying glass icon. To do this, it’s required to edit the home_page.xml menu file by adding a SearchView item as in the following image. When a user types something in the SearchView widget and hits return, our searchable activity will begin. In order for this activity to

```
    <item android:id="@+id/action_search"
        android:title="search"
        android:icon="@android:drawable/ic_menu_search"
        android:showAsAction="collapseActionView|ifRoom"
        android:actionViewClass="android.widget.SearchView" />
```

Figure 3 Home_Page.xml

begin without failing, the search query needs to be passed to the searchable activity. This is done by the HomePage activity within the onCreateOptionsMenu callback displayed below. Please refer the comments for line by line details.

```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.home_page, menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);

    //get searchview widget menu item
    MenuItem swi= menu.findItem(R.id.action_search);
    SearchView sw= (SearchView) swi.getActionView();
    //set searchview widget hint
    sw.setQueryHint("Try Pool");

    //calling setSearchableInfo() and passing it the SearchableInfo object that represents  searchable configuration
    sw.setSearchableInfo(searchManager.getSearchableInfo(getComponentName()));

    // Do not iconify the widget; expand it by default
    sw.setIconifiedByDefault(false);

    //set onQueryText Listener for when user types search query in action bar
    sw.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        //when user returns query send the query intent to the searchable activity of TestSearch
        @Override
        public boolean onQueryTextSubmit(String query) {
            Intent intent = new Intent(HomePage.this, TestSearch.class);
            Bundle bundle = new Bundle();
            bundle.putString("query", query);
            intent.putExtras(bundle);
            startActivity(intent);
            return true;
        }

        //of usage if searchable activity and activity initiating search are the same.
        @Override
        public boolean onQueryTextChange(final String s) { return false; }
    });

    return super.onCreateOptionsMenu(menu);

}
```

Figure 4 HomePage.activity onCreateOptionsMenu Callback

This is where the searchView widget is enabled, the query is collected via the public SetOnQueryTextListener method and then sent to the searchable activity via the onQueryTextSubmit method which puts the query text inside a bundle as a string and sends it to the TestSearch activity by startActivity(intent).
In TestSearch activity, the bundle containing the query will be received and displayed using our defined search method which is a simple adapter + listview. To do this, a venue object is created ‘v’, for each item in the venues data provided by fete.json. Calling the v.GetVenueName() method retrieves the venuename for each venue object and then we look to see if the venue names from the venues class in global data that contain the query. Also employed is the java provided toLowerCase() method on both the venuename and the query so void case sensitivity. If the query is contained within the venuename, the textview will not be displayed using setVisibility and we invoke several methods define in the venue class to get the venuename, id, image, and description per each venue object and then map them to the Hashmap venue object which will ultimately be added to the venuesList. A Boolean variable flag is set to true and will make sure that the text for no query found is not displayed. That’s it in a nutshell. Not going to explain too much of the listview + adapter because that’s a whole

separate thing even if it is related and ya'll said to keep it short.

```
for (int i=0;i<featuredVenues.size();i++ ) {
    Venue v = featuredVenues.get(i);

        ///now case irrelevant
    if(v.GetVenueName().toLowerCase().contains(query.toLowerCase()) == true){
        //make sure this text is gone
        TextView noResults = (TextView) findViewById(R.id.noResults);
        noResults.setVisibility(View.GONE);
        //get venue name, venueID, venuedescription, and venue image
        String vName = v.GetVenueName();
        int vId = v.GetID();
        String mDrawableName = v.GetVenueImage();
        int resID = getResources().getIdentifier(mDrawableName , "drawable", getPackageName());
        String vDescription = v.GetVenueDescription();


        //map data to xml layout
        HashMap<String, String> venue = new HashMap<~>();
        venue.put("img", Integer.toString(resID));
        venue.put("name", vName);
        venue.put("id", Integer.toString(vId));
        venue.put("description", vDescription);
        venuesList.add(venue);

        flag = true;

    }
```

Figure 5 TestSearch.java initList() that does the 'search'

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    GlobalData data = ((GlobalData)getApplicationContext());

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_test_search);


    registerForContextMenu((ListView) findViewById(R.id.listView));


    String query = new String();
    // Get the intent, verify the action and get the query
    Intent intent = getIntent();
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
       query = intent.getStringExtra(SearchManager.QUERY);
        //set title bar text to the text query
        getActionBar().setTitle("Search Results for: " + query);
    }

    //populate list by calling initList method
    if(query != "" && venuesList.isEmpty() == true){
        initList(data, query);
    }
    else {
        Log.e("TestSearch.java", "No query variable");
    }
```

Figure 6 TestSearch.java onCreate method will receive the intent query