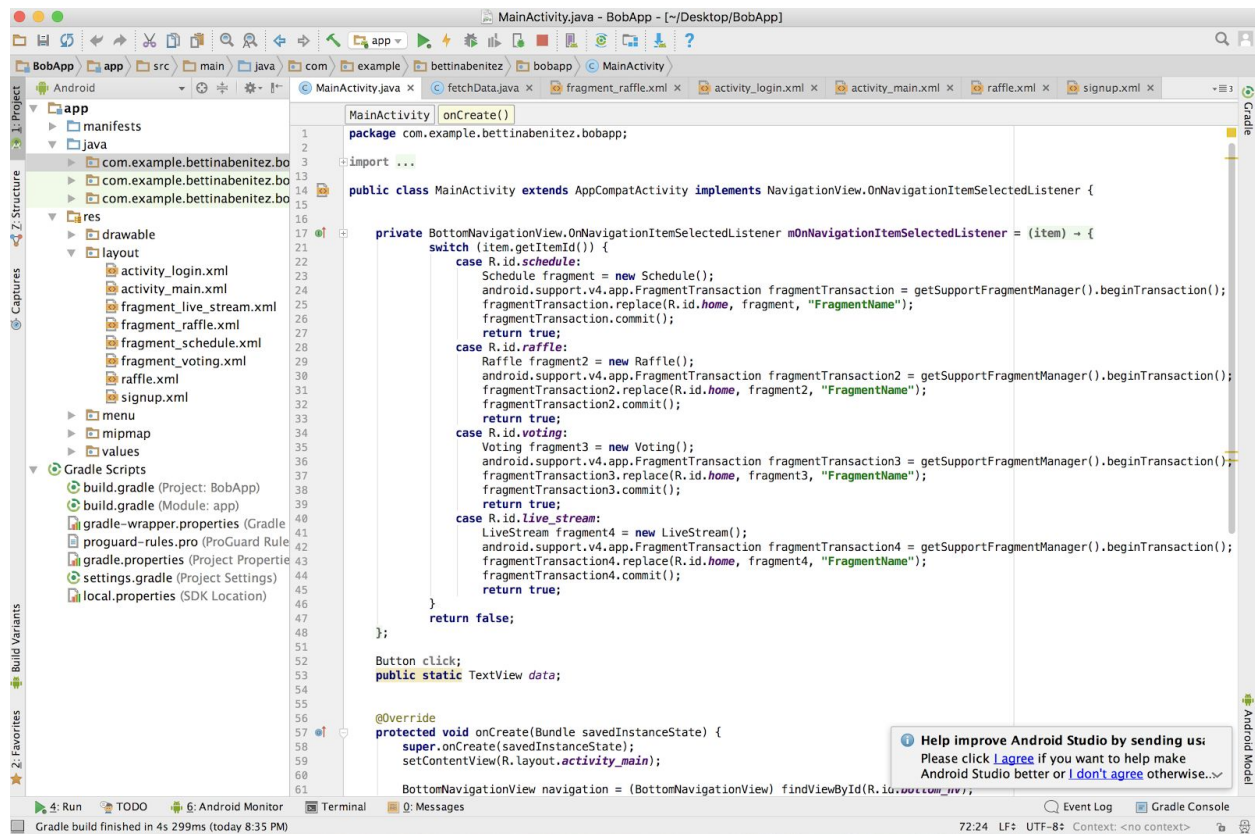


## Criterion C: Development

### THIRD PARTY TOOLS

**Android Studio:** Android Studio IDE is a software that allowed me to create my mobile application. The app allowed me to This software also allowed me to test the app's user interface with its built-in emulator. Android studio can be easily learnt by all programmers, which made it the perfect choice for app development.



## Imported libraries:

I imported Android and Java libraries in order to aid my development. These libraries gave me access to more classes and functions on Android Studio, thus making my app more complicated.

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.content.Intent;

import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.HttpURLConnection;

import android.support.v4.app.FragmentTransaction;
import android.support.design.widget.NavigationView;
import android.support.design.widget.BottomNavigationView;
import android.view.MenuItem;
import android.support.annotation.NonNull;
```

## Intent

I used the android.content.Intent collection to switch between different activities. By switching activities, my app could use different Java classes for different functions.

In order for my login activity to start, I had to change the Intent launch from the android manifest from Main activity to log in activity.

```
<activity android:name=".MainActivity" />
<activity android:name=".SignUpActivity" />
<activity android:name=".LoginActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

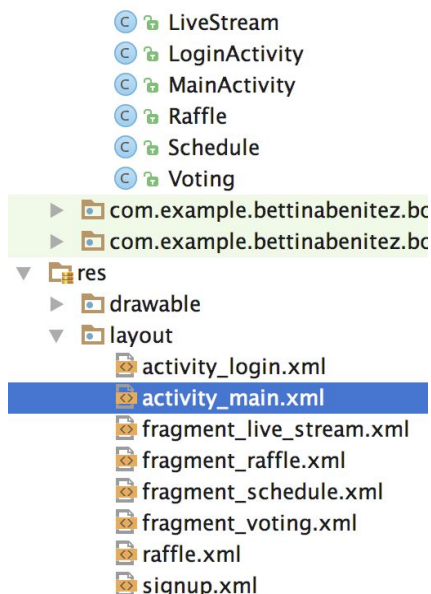
I then used the intent class to switch between activities.

```
signup.setOnClickListener((view) → {  
    Intent intent = new Intent(LoginActivity.this, SignUpActivity.class);  
    startActivity(intent);  
});
```

## Bottom Navigation Bar

Besides switching activities, I also used fragments. Fragments are segments of the user interface that could be used in an activity. Fragments allowed my app to look coherent by having the same header and bottom bar for each function of the app.

```
private BottomNavigationView.OnNavigationItemSelectedListener mOnNavigationItemSelectedListener = (item) → {  
    switch (item.getItemId()) {  
        case R.id.schedule:  
            Schedule fragment = new Schedule();  
            android.support.v4.app.FragmentTransaction fragmentTransaction = getSupportFragmentManager().beginTransaction();  
            fragmentTransaction.replace(R.id.home, fragment, "FragmentName");  
            fragmentTransaction.commit();  
            return true;  
        case R.id.raffle:  
            Raffle fragment2 = new Raffle();  
            android.support.v4.app.FragmentTransaction fragmentTransaction2 = getSupportFragmentManager().beginTransaction();  
            fragmentTransaction2.replace(R.id.home, fragment2, "FragmentName");  
            fragmentTransaction2.commit();  
            return true;  
        case R.id.voting:  
            Voting fragment3 = new Voting();  
            android.support.v4.app.FragmentTransaction fragmentTransaction3 = getSupportFragmentManager().beginTransaction();  
            fragmentTransaction3.replace(R.id.home, fragment3, "FragmentName");  
            fragmentTransaction3.commit();  
            return true;  
        case R.id.live_stream:  
            LiveStream fragment4 = new LiveStream();  
            android.support.v4.app.FragmentTransaction fragmentTransaction4 = getSupportFragmentManager().beginTransaction();  
            fragmentTransaction4.replace(R.id.home, fragment4, "FragmentName");  
            fragmentTransaction4.commit();  
            return true;  
    }  
    return false;  
};
```



For the main section of my app, I used the BottomNavigationView widget. The bottom navigation bar allows users to switch back and forth between functions of the app. I chose to use this as it makes it easier for the user to navigate through my app.

The background and header of my app was made using photoshop

To create the bottom navigation and the fragments on android

**import** android.support.design.widget.BottomNavigationView;

**import** android.support.v4.app.FragmentTransaction;

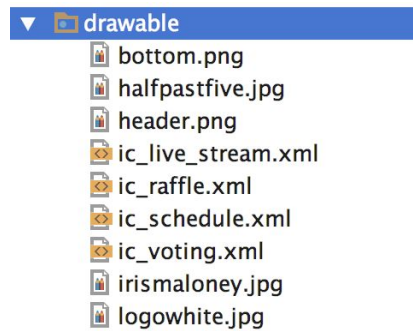
To ensure I'm switching between navigations, I used fragmentTransaction to navigate and replace the one fragment to another. I also used a switch statement, which had the id of each button that referred to a fragment layout.



## **Resource**

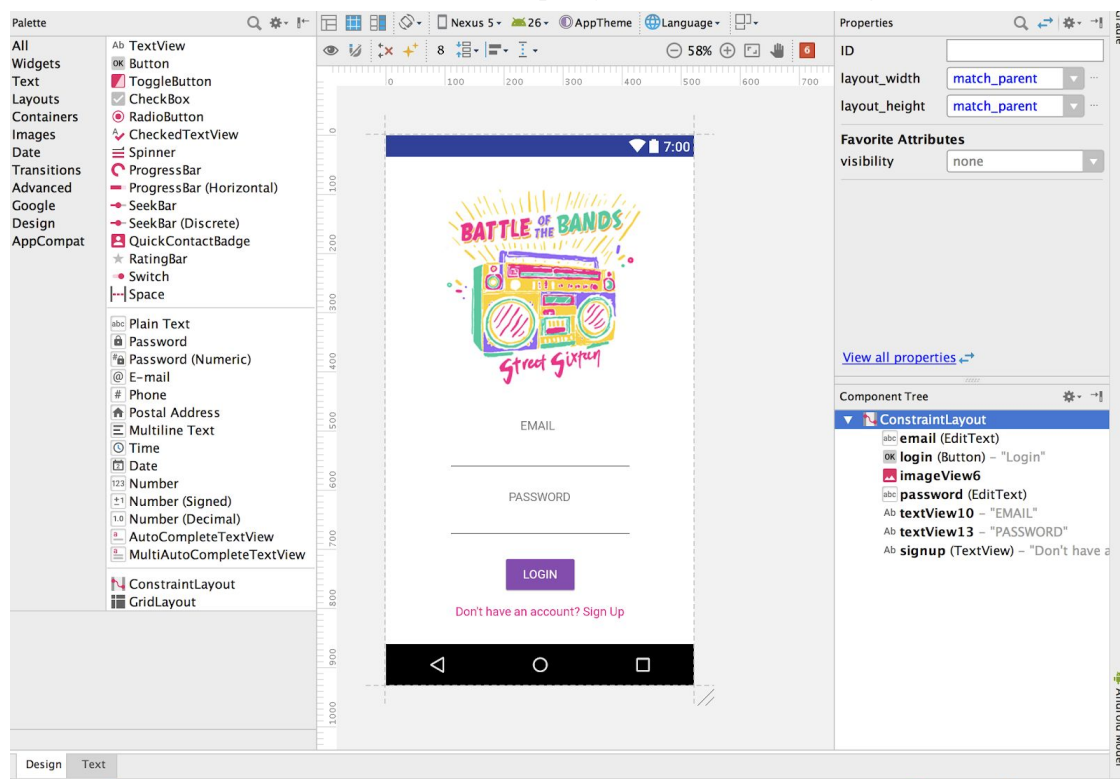
### **Layout**

For all the images added to Android studio, I had to import them to the layout folder, in png format.



## XMLs

Android Studio IDE provided a GUI toolkit which helped me design my user interface. This allowed me to develop my skills in GUI design.



## Values

For the user interface of my app, I had to add my preferred colours to the colors.xml file. This is to ensure that all colours stayed uniformed within different activities or fragments by calling the id of the colours.

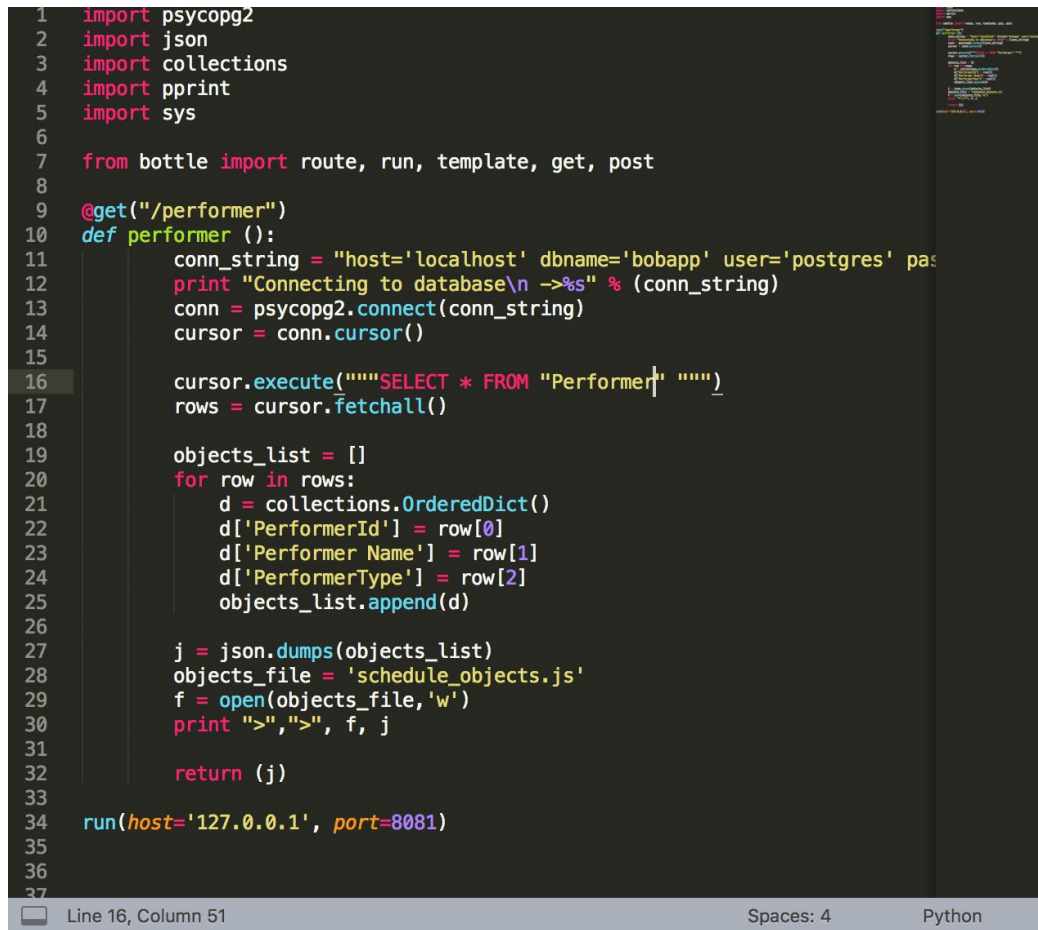


```

<color name="purpleText">#7D17B7</color>
<color name="purpleButton">#8B47B2</color>
<color name="pinkText">#FE0089</color>
<color name="pinkButton">#FA3AB2</color>
<color name="cyanText">#04E4AD</color>
<color name="white">#FFFFFF</color>
<color name="pinkDD">#F873bb</color>
<color name="yellowText">#FFD200</color>
<color name="black">#000000</color>

```

**Sublime Text:** This text editor allowed me to write my python code. Along with **Bottle.py**, Web Server Gateway Interface (WSGI) micro-web framework, and I was able to use these softwares to create the connection between my database and android by sending JSON data files.



```

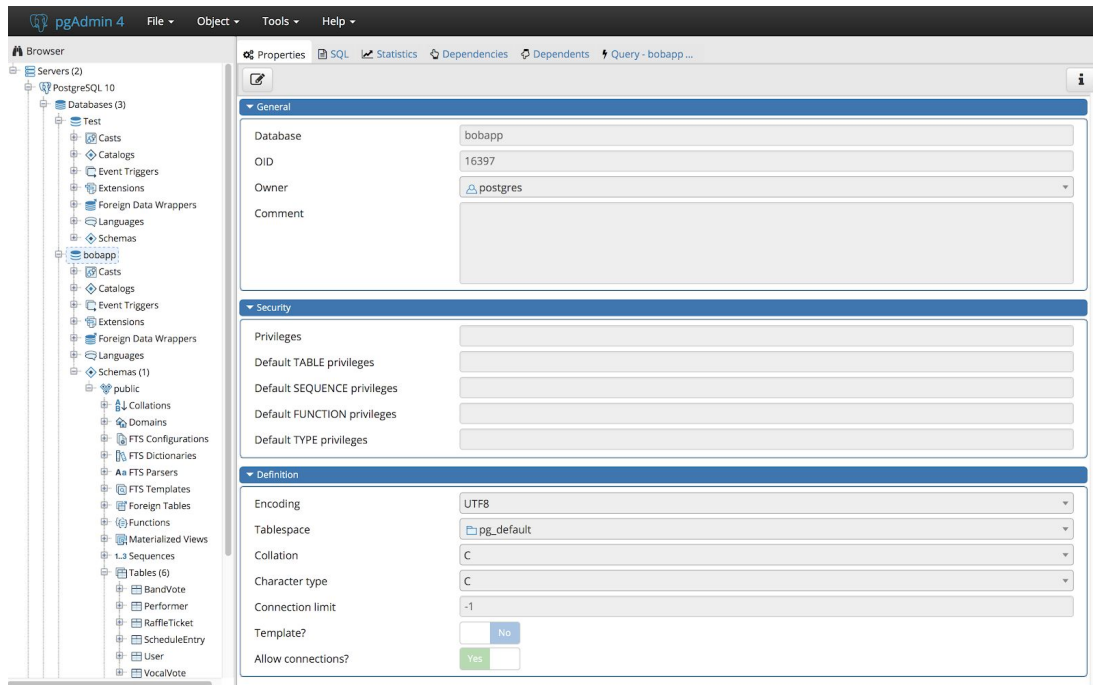
1  import psycpg2
2  import json
3  import collections
4  import pprint
5  import sys
6
7  from bottle import route, run, template, get, post
8
9  @get("/performer")
10 def performer():
11     conn_string = "host='localhost' dbname='bobapp' user='postgres' pas
12     print "Connecting to database\n ->%s" % (conn_string)
13     conn = psycpg2.connect(conn_string)
14     cursor = conn.cursor()
15
16     cursor.execute("""SELECT * FROM "Performer" """)
17     rows = cursor.fetchall()
18
19     objects_list = []
20     for row in rows:
21         d = collections.OrderedDict()
22         d['PerformerId'] = row[0]
23         d['Performer Name'] = row[1]
24         d['PerformerType'] = row[2]
25         objects_list.append(d)
26
27     j = json.dumps(objects_list)
28     objects_file = 'schedule_objects.js'
29     f = open(objects_file, 'w')
30     print ">",">", f, j
31
32     return (j)
33
34 run(host='127.0.0.1', port=8081)
35
36
37

```

Line 16, Column 51      Spaces: 4      Python

**Psycpg2:** Psycpg2 is a **PostgreSQL** adapter that let me connect my python file with my database. To get this, I downloaded this using the **pip** function.

**PostgreSQL:** This is the database that let me store all the data for my app. I used the pgAdmin4, Open Source administration and development platform. Due to this, I was able to use postgresQL and others are able to implement the database tables I've created.



Along with bottle.py, I was able to retrieve JSON data from the output of the SQL queries made in python. Below shows data from my User table and convert the data into JSON format.

The image shows a web browser window with the address bar displaying '127.0.0.1:8081/user'. The browser's address bar and tabs are visible at the top. The main content area displays a JSON array of user data: [{"UserId": 1, "GmailAddress": "benitezb@ismanila.org", "FirstName": "Bettina ", "LastName": "Benitez ", "Pasword": "hello ", "UserType": "Admin "}, {"UserId": 2, "GmailAddress": "gok@ismanila.org", "FirstName": "Katie ", "LastName": "Go ", "Pasword": "sup ", "UserType": "Admin "}, {"UserId": 3, "GmailAddress": "GmailAddress", "FirstName": "FirstName ", "LastName": "LastName ", "Pasword": "Password ", "UserType": "UserType "}].

```
[{"UserId": 1, "GmailAddress": "benitezb@ismanila.org", "FirstName": "Bettina ", "LastName": "Benitez ", "Pasword": "hello ", "UserType": "Admin "}, {"UserId": 2, "GmailAddress": "gok@ismanila.org", "FirstName": "Katie ", "LastName": "Go ", "Pasword": "sup ", "UserType": "Admin "}, {"UserId": 3, "GmailAddress": "GmailAddress", "FirstName": "FirstName ", "LastName": "LastName ", "Pasword": "Password ", "UserType": "UserType "}]]
```

## Arrays

### 2D arrays

2D arrays were used to create JSON object lists

```
objects_list = []
for row in rows:
    d = collections.OrderedDict()
    d['UserId'] = row[0]
    d['GmailAddress'] = row[1]
    d['FirstName'] = row[2]
    d['LastName'] = row[3]
    d['Pasword'] = row[4]
    d['UserType'] = row[5]
    objects_list.append(d)

j = json.dumps(objects_list)
objects_file = 'schedule_objects.js'
f = open(objects_file, 'w')
print ">", ">", f, j

return (j)
```

## Visualisation

'UserId'	'GmailAddress'	'FirstName'	'Last Name'	'Password'	'UserType'
1	john@gmail.com	John	Doe	p@ssw0rd	Admin

## Encapsulation

```
public class User {

    private static String emailAddress;
    private String bandPerformer;
    private String vocalPerformer;

    public static String getEmailAddress() { return emailAddress; }

    public String getBandPerformer() { return bandPerformer; }

    public String getVocalPerformer() { return vocalPerformer; }

    public void setEmailAddress(String emailAddress) { this.emailAddress = emailAddress; }

    public void setBandPerformer(String bandPerformer) { this.bandPerformer = bandPerformer; }

    public void setVocalPerformer(String vocalPerformer) { this.vocalPerformer = vocalPerformer; }

}
```

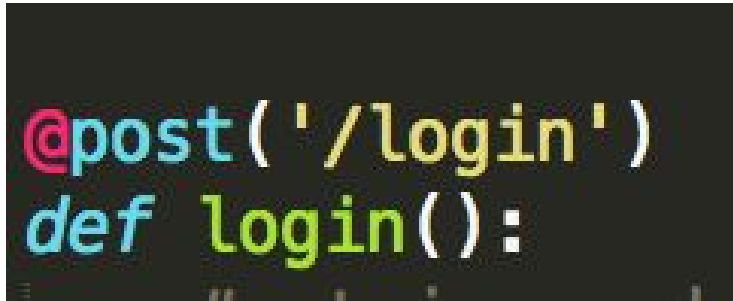


## Integrating Technologies

This app uses SQL queries, Python programming, and Java programming

Integrating technologies is important to create the three-tier architecture. This was used to connect client to server and server to database. These technologies allow my app to potentially be distributed, and different clients to use the same server that connects to the database.

PostgreSQL, Bottle, Android connection.

PostgreSQL to Bottle	<pre>try:     # read database configuration     conn_string = "host='localhost' dbname='bobapp' user='postgres' password='pkolij'"     print "Connecting to database\n -&gt;%s" % (conn_string)     conn = psycopg2.connect(conn_string)     cursor = conn.cursor()     # [execute] the INSERT statement     cursor.execute(sql, (GmailAddress ,FirstName, LastName, Password, UserType) )     # commit the changes to the database     conn.commit()     # close communication with the database     cursor.close() except (Exception, psycopg2.DatabaseError) as error:     print(error) finally:     if conn is not None:         conn.close()</pre>
Bottle posting to HTTP  Example: <a href="http://10.0.2.2:8080/login">http://10.0.2.2:8080/login</a>  The return statement parses JSON data to HTTP	
HTTP URL connection in Android Studio	To parse data from the PostgreSQL database to android, the line of code below was added to the Android Manifest: <code>&lt;uses-permission android:name="android.permission.INTERNET" /&gt;</code>

	<pre>//CONNECTING TO URL --&gt; send/retrieve JSON data URL url = new URL("http://10.0.2.2:8080/login"); //10.0.2.2 =&gt; localhost URLConnection conn = (URLConnection) url.openConnection(); //POST request method, from python code conn.setRequestMethod("POST"); conn.setRequestProperty("Content-Type", "application/json;charset=UTF-8"); conn.setRequestProperty("Accept", "application/json"); conn.setDoOutput(true); conn.setDoInput(true);</pre>
--	--

## JSON File Conversions

<u>ANDROID</u>	
<p><u>Output Stream (Serialization)</u></p> <p>The output stream is used here to write the created JSON data as a string to the output stream as a sequence of bytes.</p>	<pre>//Converting Strings into JSON format and adding them to the jsonParam Object JSONObject jsonParam = new JSONObject(); jsonParam.put("GmailAddress", GmailAddress); jsonParam.put("Password", Password);  //Sending JSON Objects to HTTP DataOutputStream outputStream = new DataOutputStream(conn.getOutputStream()); outputStream.writeBytes(jsonParam.toString());  outputStream.flush(); outputStream.close();</pre>
<p><u>Input Stream (Deserialization)</u></p> <p>The input stream class is used here to returns the next byte of the input.</p> <p>The bufferedreader class is also used to create buffering character-input stream that uses a default-sized input buffer</p> <p>From the bufferedreader, the JSON data is appended to a string</p>	<pre>//Retrieves JSON Objects from HTTP InputStream inputStream = conn.getInputStream(); BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream));  //Creates a string builder used to convert JSON data to string StringBuilder sb = new StringBuilder(); String line; //Reads the JSON data line = bufferedReader.readLine();  //checks if no JSON data was send, access is denied if (line == null) {     access = 0; // cannot move on to the next activity     Log.e("ACCESS", "denied");     signup.setText("Incorrect Email and/or Password"); }  //while line is not = null, add the JSON data to the string builder while (line != null) {     sb.append(line+"\n");     Log.e("ACCESS:", line); //checks the access through logcat     line = bufferedReader.readLine();     access = 1; //moves to the next activity }  bufferedReader.close();</pre>

builder.	
<b><u>PYTHON</u></b>	
<u>Encoding JSON</u> Json.dump <u>Decoding JSON</u> Json.loads  Encoding and decoding allows data to be converted in the logic tier to be transferred from the database tier to the client tier.	<pre>maxTicketsEncode= json.dumps(maxTickets[0]) maxTicketsDecode = json.loads(maxTicketsEncode)</pre>

### **SQL Queries**

To insert, select, and update my data from my database, I had to learn some SQL queries.

Basic SQL queries: <pre>SELECT * FROM "User"; INSERT INTO "Performer"("Performer_Id", "Performer_Name", "Performer_Type") VALUES("1", "Bettina", "Vocal"); UPDATE "ScheduleEntry" SET "Status" = 1;</pre>
--

SQL data output of inner join is from other tables

bobapp on postgres@localhost

1
SELECT "Performer\_Name" FROM "Performer" inner join "ScheduleEntry" ON "Performer"."PerformerId" = "ScheduleEntry"."PerformerId"

Data Output

Performer\_Name

character (255)

1

Black Tears

2

Cookie Problems

3

Twisted Adobo

4

Parental Advisory

5

The Dashboard D...

6

United States of H...

7

University Parkway

8

Half Past Five

9

Silvana De Dios

10

Meg Barraca & Ka...

11

Margaret Netherc...

12

Miguela Puyat

13

Yojana Narang

14

Iris Maloney

15

Ben Reedy

bobapp on postgres@localhost

1
SELECT \* FROM "ScheduleEntry"

Data Output

ScheduleId

integer

PerformerId

integer

Status

integer

Performer\_id

character (255)

1

4

2

2

2

2

5

11

2

11

3

1

9

2

9

4

2

1

2

1

5

3

10

2

10

6

6

3

0

3

7

8

4

1

4

8

9

13

1

13

9

14

7

1

7

10

15

8

1

8

11

10

5

1

5

12

7

12

1

12

13

11

14

1

14

14

12

6

1

6

15

13

15

1

15

## Debugging

I encountered a problems when creating my app, thus I used debuggers for both android and python to assist my code development.

## Android

### Log

Using Log, I was able to debug my app by checking the logcat from the Android Monitor Window

```
E/ACCESS: denied
E/ACCESS:: {"access": "accepted"}
```

```
while (line != null) {
    sb.append(line+"\n");
    Log.e("ACCESS:", line);
    line = bufferedReader.readLine();
    access = 1;
}

bufferedReader.close();

Log.i("STATUS", String.valueOf(conn.getResponseCode()));
Log.i("MSG", conn.getResponseMessage());
```

### Python

#### Exception

Used during database connection, prints the error on terminal

```
except Exception as e:
    print("Error: " + str(e))
```

Word Count: 846