Getting Started

Introduction to The Buzz

The Buzz is an app where users can post the movie and concert events on it in order to find people with the same interest. This project basically consist of four parts

- Admin: a command-line administrative app for management of the tables
- Backend: the server for The Buzz using the Java Spark web framework and deployed on heroku
- Web: a TypeScript web front-ends
- Android: an Android app for The Buzz developed with Android studio

Basic Requirements

- Git: we use Git to track our works
- Terminal: Mac has good default terminal. For Windows, the default terminal is fine but I recommend Git Bash
- VS code: I recommend the visual studio code for coding
- Bitbucket account: you should create an account so that we could share out project with you

How to Get Started

1. Go to the repo on Bitbucket (we will add you to the project)
2. Copy the command starts with `git clone`
3. Create a folder in the local where you want to work on your project
4. Run the `git clone` command in your terminal to get a copy of the project to your folder
5. `git checkout` to the branch you want to work on
6. Start working
7. After finishing your work, use command `git add` to add the files you want to save.
8. Then, `git commit` to save your works
9. Finally, `git push` to save your work on the Bitbucket repo

Developer Instructions

Admin

Software Requirements

- Java JDK: for code compiling
- Maven: for project constrution and please follow thehttps://maven.apache.org/install.html to install the maven

App.java

App.java is the main program of the Admin app. It contains the set up of the Database, interaction with user, and calling the functions in the database.

Command-line Interface

Here is the example of how to get user's command in the console:

```
    /**
     * Print the table menu for our program
     */
    static void tblMenu() {
        System.out.println("Choose a table");
        System.out.println("  [M] tblData");
        System.out.println("  [U] tblUser");
        System.out.println("  [C] tblComment");
        System.out.println("  [L] tblLike");
        System.out.println("  [D] tblDislike");
        System.out.println("  [F] tblFile");
        System.out.println("  [q] Quit to main menu");
        System.out.println("  [?] Help (this message)");
    }

    /**
     * Ask the user to enter a menu option; repeat until we get a valid option
     *
     * @param in A BufferedReader, for reading from the keyboard
     *
     * @return The character corresponding to the chosen menu option
     */
    static char promptTables(BufferedReader in) {
        // The valid actions:
        String actions = "MUCLDFq?";

        // We repeat until a valid single-character option is selected
        while (true) {
            System.out.print("[" + actions + "] :> ");
            String action;
            try {
                action = in.readLine();
            } catch (IOException e) {
                e.printStackTrace();
                continue;
            }
            if (action.length() != 1)
                continue;
            if (actions.contains(action)) {
                return action.charAt(0);
            }
            System.out.println("Invalid Command");
        }
    }
```

Response

Here is the example of how we react depends on user input:

```
        if (action == '1') {
            while (true) {
                action = promptTables(in);
                if (action == '?') {
                    tblMenuLite();

                } else if (action == 'M') {
                    int mid = getInt(in, "Enter the message ID");
                    if (mid == -1)
                        continue;
                    Database.RowData res = db.selectOneFromData(mid);
                    if (res != null) {
                        System.out.println("  [" + res.mId + "] " + res.mSubject);
                        System.out.println("  [" + res.uId + "] " + res.userName);
                        System.out.println("  --> " + res.mMessage);
                        System.out.println("  --> " + res.mlikes);
                        System.out.println("  --> " + res.mdislikes);
                        System.out.println("  --> " + res.mDate);
                        System.out.println("  --> " + res.mLink);
                        System.out.println("  --> " + res.fileId);
                    }

                }
```

Database.java

Database is responsible for the interaction with our heroku database. To do that, we have to write the prepared SQL statements like follow:

```
        db.uCreateTable = db.mConnection
                .prepareStatement("CREATE TABLE tblUser (uid SERIAL PRIMARY KEY, username VARCHAR(50) "
                        + "NOT NULL, email VARCHAR(500) NOT NULL "
                        + ", intro VARCHAR(500) NOT NULL, quota INTEGER)");
        db.uDropTable = db.mConnection.prepareStatement("DROP TABLE tblUser");

        // Standard CRUD operations
        db.uDeleteOne = db.mConnection.prepareStatement("DELETE FROM tblUser WHERE uid = ?");
        db.uInsertOne = db.mConnection.prepareStatement("INSERT INTO tblUser VALUES (default, ?, ?, ?)");
        db.uSelectAll = db.mConnection.prepareStatement("SELECT uid, username FROM tblUser");
        db.uSelectOne = db.mConnection.prepareStatement("SELECT * from tblUser WHERE uid = ?");
        db.uUpdateOne = db.mConnection.prepareStatement("UPDATE tblUser SET username = ?, intro = ? WHERE
```

Then, write the functions to set up the values for the question marks and execute the statements:

```
/**
 * Insert a row into the database
 *
 * @param email The subject for this new row
 *
 * @return The number of rows that were inserted
 */
int insertRowToUser(String email) {
    int count = 0;
    try {
        uInsertOne.setString(1, email.split("@")[0]);
        uInsertOne.setString(2, email);
        uInsertOne.setString(3, "This person is lazy, so nothing's here");
        count += uInsertOne.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return count;
}
```

Compiling and Running

To compile the code, run `mvn package` in the terminal to compile the program and run the tests. Then, run `mvn exec:java` with our database url:
`DATABASE_URL=postgres://ptswzskpnbqwdn:ad5fb569055109e8808cf7cda7356dee5f9e428bb64b064fdb0849e85b72b8c8@ec2-107-21-201-238.compute-1.amazonaws.com:5432/d9bk6p14rm5u9e mvn exec:java`

Backend

Software Requirements

- Java SDK: code compiling basic
- Maven: for package and deploy

Java Spark Web Framework

We use Java Spark to map the Routes to the corresponding Spark functions so that when front-ends make a Json call to a specific url, backend can give the response to that request.

Here is the example of how to respond to the "/listmessages" post request in backend:

```
    Spark.post("/listmessages", (request, response) -> {
        // ensure status 200 OK, with a MIME type of JSON
        response.status(200);
        response.type("application/json");
        SimpleRequest req = gson.fromJson(request.body(), SimpleRequest.class);
        String sk = req.sessionKey;
        String em = req.uEmail;
        String key = "";
        try {
            key = mc.get(em);
        } catch (TimeoutException te) {
            return gson.toJson(new StructuredResponse("error", "Timeout during set or get: " + te.getMessa
        } catch (InterruptedException ie) {
            return gson.toJson(new StructuredResponse("error", "Interrupt during set or get: " + ie.getMes
        } catch (MemcachedException me) {
            return gson.toJson(new StructuredResponse("error", "Memcached error during get or set: " + me.
        }
        if (sk.equals(key)){
            return gson.toJson(new StructuredResponse("ok", null, db.readAll()));
        }
        return gson.toJson(new StructuredResponse("error", "session key not correct..", null));
    });
```

Google API

We use Google API in the backend for authentication purpose. Here is how we check the auth token sent from the front-ends with Google OAuth:

```
        OAuthRequest req = gson.fromJson(request.body(), OAuthRequest.class);
        String idTokenString = req.id_token;
        response.status(200);
        response.type("application/json");

        // Obtain user's Gmail using the token provided by Google

        String email;

        GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.Builder(new NetHttpTransport(), new Jac
            .setAudience(Collections.singletonList("689219964832-6m703l22ir6jh9ra1m1lhrgg12bv7olt.apps.goo
            .build();

        GoogleIdToken idToken = verifier.verify(idTokenString);
        if (idToken != null) {
            Payload payload = idToken.getPayload();
            email = payload.getEmail();
            if (!email.contains("@lehigh.edu")) {
                return gson.toJson(new StructuredResponse("error", "invalid email", null));
            }
        } else {
            return gson.toJson(new StructuredResponse("error", "invalid id token", null));
        }
```

For more details, please check outhttps://developers.google.com/identity/sign-in/web/backend-auth

Also, we use Google API for uploading files in the event posts to the Google Drive:

```
        Drive service = getService();

        File fileMetadata = new File();
        String path = "metadata_" + filename;
        fileMetadata.setName(path);
        FileContent mediaContent = new FileContent(mimeType, file);
        File result = service.files().create(fileMetadata, mediaContent).setFields("id").execute();
```

Check out https://developers.google.com/drive/api/v3/quickstart/java for more details

Web

Software Requirements

- NPM: for package management

TypeScript

TypeScript is a superset of JavaScript, which lets us write event-based code acting in response to network traffic and keyboard/mouse events to change the information within the HTML elements. Basically, it is in charge of the functionality of the web page such as setting up the response of button and put the information on the web page.

The following is the example of how to show QR code to the user on website:

```
class ShowQR {

    private static readonly NAME = "ShowQR";

    /**
     * Track if the Singleton has been initialized
     */
    private static isInit = false;

    /**
     * init() is called from an AJAX GET, and should populate the form if and
     * only if the GET did not have an error
     */
    private static init() {
        if (!ShowQR.isInit) {
            $("body").prepend(Handlebars.templates[ShowQR.NAME + ".hb"]());
            // $("#"+Navbar.NAME+"-showQR").click(Navbar.login);
            $("#" + ShowQR.NAME + "-Close").click(ShowQR.hide);
            ShowQR.isInit = true;

        }
    }

    /**
     * Refresh() doesn't really have much meaning, but just like in sNavbar, we
     * have a refresh() method so that we don't have front-end code calling
     * init().
     */
    public static refresh() {
        ShowQR.init();
    }

    /**
```

```
     * Hide the ShowQR.  Be sure to clear its fields first
     */
    private static hide() {

        $("#" + ShowQR.NAME + "").val("");
        $("#" + ShowQR.NAME).modal("hide");
    }

    /**
     * Show the ShowQR.  Be sure to clear its fields, because there are
     * ways of making a Bootstrap modal disapper without clicking Close, and
     * we haven't set up the hooks to clear the fields on the events associated
     * with those ways of making the modal disappear.
     */

    public static show() {
        var qrdata = uid + "\n" + uemail + "\n" + ukey;

            QRCode.toCanvas(document.getElementById('canvas'), qrdata, function (error: any) {
                if (error) console.error(error)
                    $.ajax({
                        type: "POST",
                        url: "/generateQR",
                        dataType: "json",
                        data: JSON.stringify({ uId: uid, uEmail: uemail, sessionKey: ukey }),
                    });

              });
    }
} // end class ShowQR
```

Handlebars

Handlebars is in the hb folder. It is in charge of the layout of our website. It set up the location of the messages, size of button and so on. The following is an example of the QR code modal:

```
<div id="ShowQR" class="modal fade" role="dialog">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h4 class="modal-title">View Your QR Code</h4>
            </div>
            <div class="modal-body">
                <input type = "text" id = "qr-data" onchange = "generateQR()">
                <div id="qrcode"></div>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" id="ShowQR-Close">Close</button>
            </div>
        </div>
    </div>
</div>
```

CSS

CSS is in charge of the styling of the web page, such as the color of the button. We have a css folder but we have not made any CSS format yet. If you want to write any CSS format please put it in css folder.

Deploy

Since the web has to be deployed with the backend together, to deploy the web on our heroku server, first you have to run deploy.sh with `sh deploy.sh` in the terminal. Then, use command `mvn heroku:deploy` in the backend folder to deploy your web on the server.

If you have add new .hb or .ts files, please add the .hb files to the deploy.sh and add the .ts files to app.ts reference when you want to deploy.

.hb files in deploy.sh:

```
 # step 6: compile handlebars templates to the deploy folder
node_modules/handlebars/bin/handlebars hb/ElementList.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/EditEntryForm.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/NewEntryForm.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/Navbar.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/ShowDetail.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/Login.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/UserProfile.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/ShowComments.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/ShowQR.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
node_modules/handlebars/bin/handlebars hb/EditUserProfile.hb >> $TARGETFOLDER/$WEBFOLDERNAME/templates.js
```

.ts files in app.ts:

```
/// <reference path="ts/EditEntryForm.ts"/>
/// <reference path="ts/NewEntryForm.ts"/>
/// <reference path="ts/ElementList.ts"/>
/// <reference path="ts/Navbar.ts"/>
/// <reference path="ts/ShowDetail.ts"/>
/// <reference path="ts/Login.ts"/>
/// <reference path="ts/UserProfile.ts"/>
/// <reference path="ts/ShowComments.ts"/>
/// <reference path="ts/ShowQR.ts"/>
/// <reference path="ts/EditUserProfile.ts"/>
```

Android

Software Requirements

- Android Studio 3.5
- Disable Hyper-V if you are using Windows system

Getting Started

1. Open the Android Studio
2. Click "Open an existing project"
3. Choose the Android folder in cse216_teamname

Compiling

Click the green right arrow button on the top left of the Android Studio to install and run your program.

If you want to run your program on a virtual phone, you can create an emulator in the Android Studio

build.gradle

You need to put all the configurations and depedencies here.

Here is an example of our build.gradle:

```
 apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "edu.lehigh.cse216.yut222.phase0"
        minSdkVersion 15
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation 'com.google.android.gms:play-services-auth:17.0.0'
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.android.material:material:1.0.0'
    implementation 'org.apache.commons:commons-io:1.3.2'
    implementation 'com.google.android.gms:play-services-vision:19.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
    implementation 'com.android.support:multidex:1.0.3'
    implementation 'com.android.volley:volley:1.1.0'
    androidTestImplementation('com.android.support.test:runner:0.5', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    androidTestImplementation 'androidx.test:rules:1.3.0-alpha02'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
}
```

Java Files

Java Files are use to develop the functionality of the activities.

Here is an example of our WelcomeActivity.java to set up the function for the buttons on the page:

```
public class WelcomeActivity extends AppCompatActivity {
    Button bu=null;
    Button bu2=null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_welcome);

        bu=(Button)findViewById(R.id.button2);
        bu2=(Button)findViewById(R.id.button3);
    }

    public void logout(View view){
        SharedPreferences sharedpreferences = PreferenceManager.getDefaultSharedPreferences(getApplication
        SharedPreferences.Editor editor = sharedpreferences.edit();
        editor.clear();
        editor.commit();
        finish();
    }

    public void close(View view){
        Intent i = new Intent(WelcomeActivity.this, MainActivity.class);
        startActivity(i);
    }

}
```

XML Files

XML Files are just like HTML. They are used for formatting the interface. Here is an example of how we format our WelcomeActivity page:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">



    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Logout"
        android:onClick="logout"
        android:id="@+id/button2"
        android:layout_gravity="center_horizontal"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="191dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Main Page"
        android:onClick="close"
        android:id="@+id/button3"
        android:layout_below="@+id/button2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="69dp" />

</RelativeLayout>
```

Google API

We use Google API for login purpose.

Integrating with project

Add the following to the app-level build.gradle:

```
apply plugin: 'com.android.application'
    ...

    dependencies {
        implementation 'com.google.android.gms:play-services-auth:17.0.0'
    }
```

A Quick Start example for Google signin flow

```
public class SignInActivity extends AppCompatActivity implements
        View.OnClickListener {

    private static final String TAG = "SignInActivity";
    private static final int RC_SIGN_IN = 9001;

    private GoogleSignInClient mGoogleSignInClient;
    private TextView mStatusTextView;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Views
        mStatusTextView = findViewById(R.id.status);

        // Button listeners
        findViewById(R.id.sign_in_button).setOnClickListener(this);
        findViewById(R.id.sign_out_button).setOnClickListener(this);
        findViewById(R.id.disconnect_button).setOnClickListener(this);

        // [START configure_signin]
        // Configure sign-in to request the user's ID, email address, and basic
        // profile. ID and basic profile are included in DEFAULT_SIGN_IN.
        GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestEmail()
                .build();
        // [END configure_signin]

        // [START build_client]
        // Build a GoogleSignInClient with the options specified by gso.
        mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
        // [END build_client]

        // [START customize_button]
        // Set the dimensions of the sign-in button.
        SignInButton signInButton = findViewById(R.id.sign_in_button);
        signInButton.setSize(SignInButton.SIZE_STANDARD);
        signInButton.setColorScheme(SignInButton.COLOR_LIGHT);
        // [END customize_button]
    }

    @Override
    public void onStart() {
        super.onStart();

        // [START on_start_sign_in]
        // Check for existing Google Sign In account, if the user is already signed in
        // the GoogleSignInAccount will be non-null.
        GoogleSignInAccount account = GoogleSignIn.getLastSignedInAccount(this);
        updateUI(account);
        // [END on_start_sign_in]
    }

    // [START onActivityResult]
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Result returned from launching the Intent from GoogleSignInClient.getSignInIntent(...);
        if (requestCode == RC_SIGN_IN) {
            // The Task returned from this call is always completed, no need to attach
            // a listener.
            Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
            handleSignInResult(task);
        }
    }
```

```
    // [END onActivityResult]

    // [START handleSignInResult]
    private void handleSignInResult(Task<GoogleSignInAccount> completedTask) {
        try {
            GoogleSignInAccount account = completedTask.getResult(ApiException.class);

            // Signed in successfully, show authenticated UI.
            updateUI(account);
        } catch (ApiException e) {
            // The ApiException status code indicates the detailed failure reason.
            // Please refer to the GoogleSignInStatusCodes class reference for more information.
            Log.w(TAG, "signInResult:failed code=" + e.getStatusCode());
            updateUI(null);
        }
    }
    // [END handleSignInResult]

    // [START signIn]
    private void signIn() {
        Intent signInIntent = mGoogleSignInClient.getSignInIntent();
        startActivityForResult(signInIntent, RC_SIGN_IN);
    }
    // [END signIn]

    // [START signOut]
    private void signOut() {
        mGoogleSignInClient.signOut()
                .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        // [START_EXCLUDE]
                        updateUI(null);
                        // [END_EXCLUDE]
                    }
                });
    }
    // [END signOut]

    // [START revokeAccess]
    private void revokeAccess() {
        mGoogleSignInClient.revokeAccess()
                .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        // [START_EXCLUDE]
                        updateUI(null);
                        // [END_EXCLUDE]
                    }
                });
    }
    // [END revokeAccess]

    private void updateUI(@Nullable GoogleSignInAccount account) {
        if (account != null) {
            mStatusTextView.setText(getString(R.string.signed_in_fmt, account.getDisplayName()));

            findViewById(R.id.sign_in_button).setVisibility(View.GONE);
            findViewById(R.id.sign_out_and_disconnect).setVisibility(View.VISIBLE);
```

```
            findViewById(R.id.sign_out_and_disconnect).setVisibility(View.VISIBLE);
        } else {
            mStatusTextView.setText(R.string.signed_out);

            findViewById(R.id.sign_in_button).setVisibility(View.VISIBLE);
            findViewById(R.id.sign_out_and_disconnect).setVisibility(View.GONE);
        }
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.sign_in_button:
                signIn();
                break;
            case R.id.sign_out_button:
                signOut();
                break;
            case R.id.disconnect_button:
                revokeAccess();
                break;
        }
    }
}
```

For more details, here is the link of the tutorial for Android Google Signin:https://developers.google.com/identity/sign-in/android/sign-in