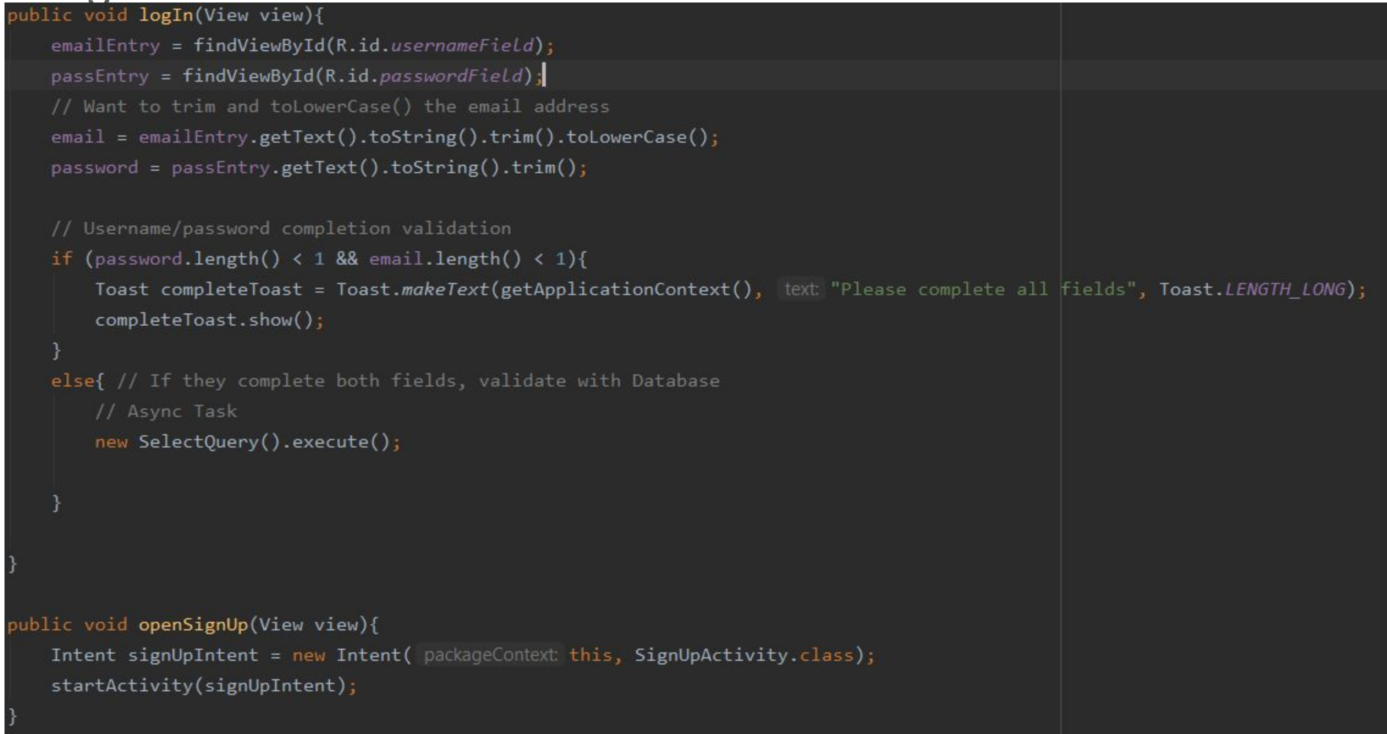
Code Release v2

Buddi’s

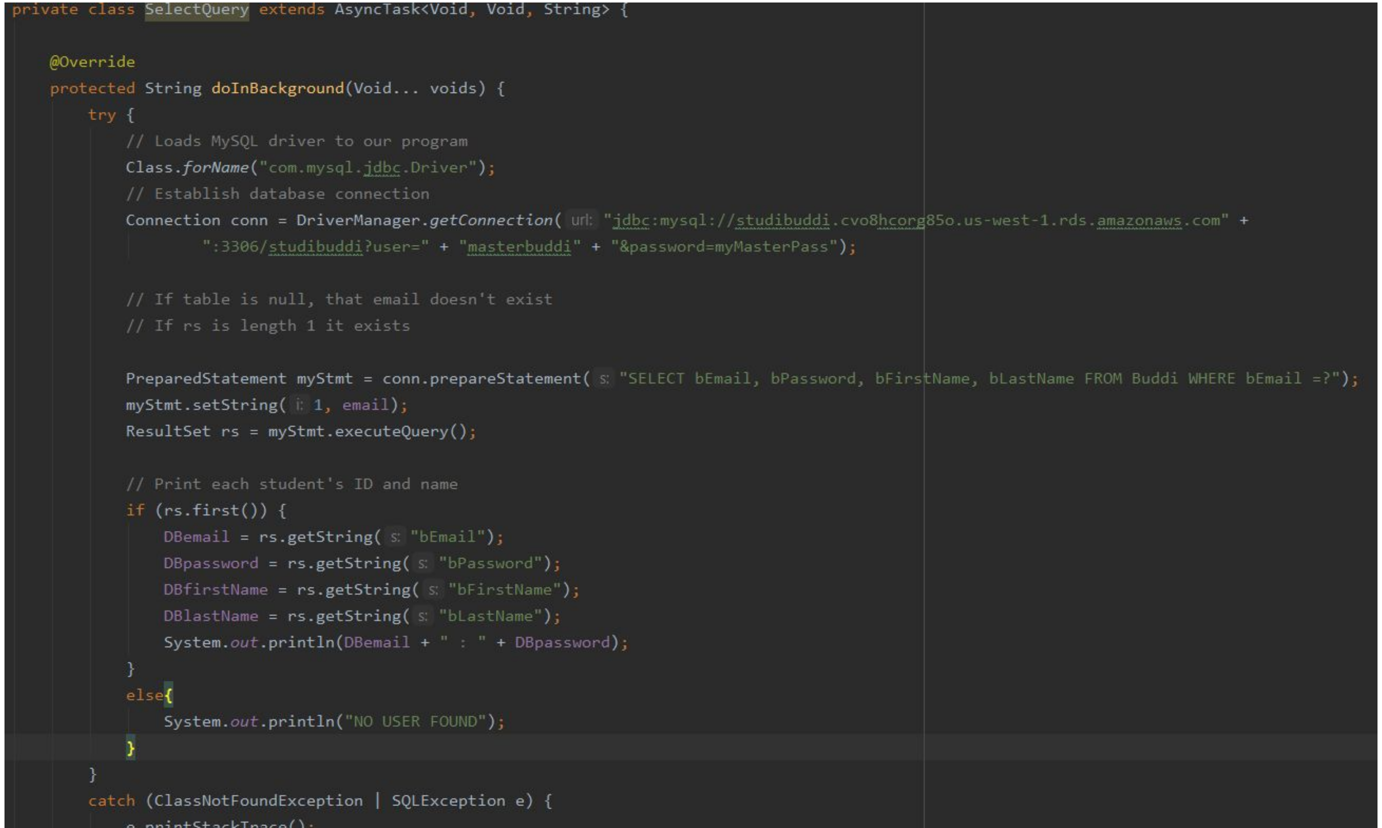
LoginActivity

This code initializes the UI, and attempts a login with the user input in the text field, with an error message if unsuccessful



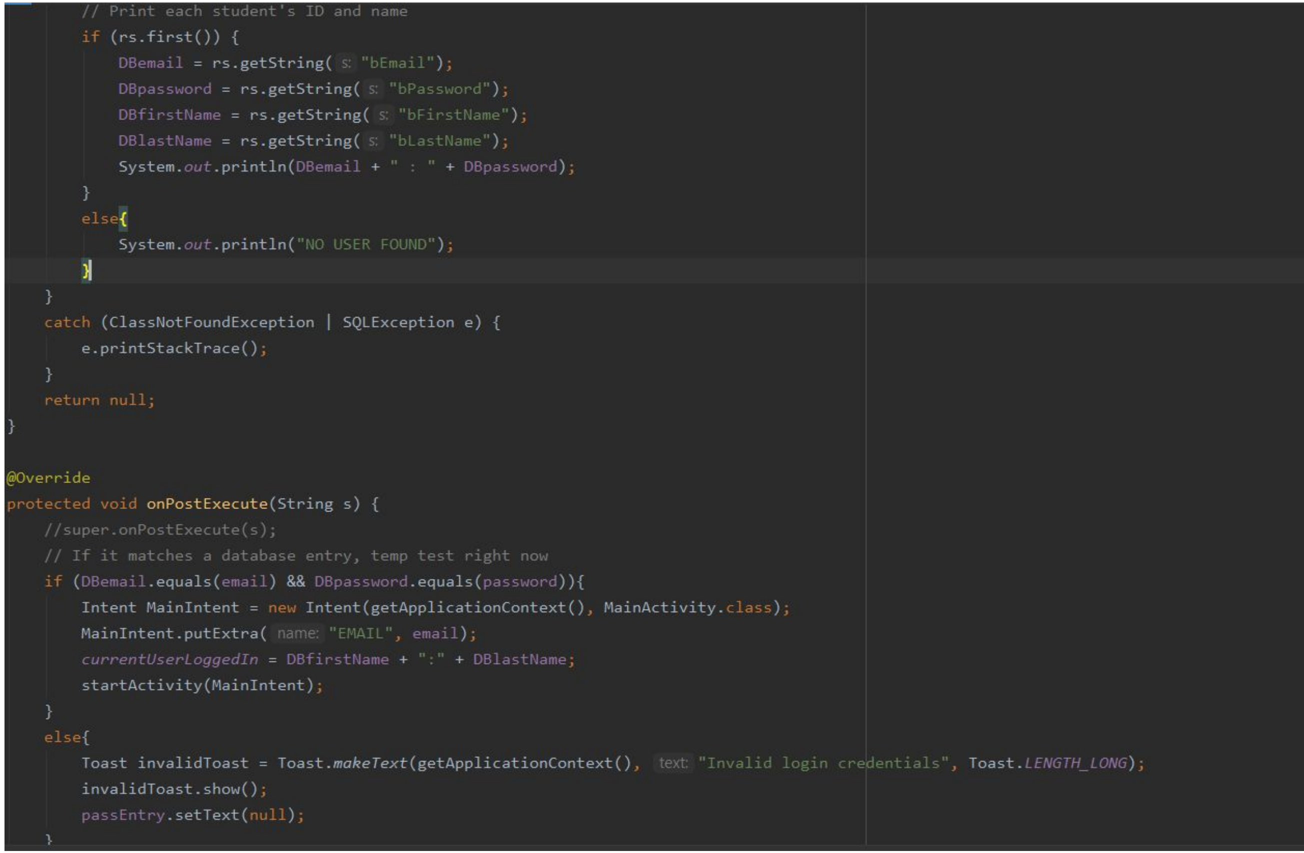
```java
public void logIn(View view){
    emailEntry = findViewById(R.id.usernameField);
    passEntry = findViewById(R.id.passwordField);
    // Want to trim and toLowerCase() the email address
    email = emailEntry.getText().toString().trim().toLowerCase();
    password = passEntry.getText().toString().trim();

    // Username/password completion validation
    if (password.length() < 1 && email.length() < 1){
        Toast completeToast = Toast.makeText(getApplicationContext(),  text: "Please complete all fields", Toast.LENGTH_LONG);
        completeToast.show();
    }
    else{ // If they complete both fields, validate with Database
        // Async Task
        new SelectQuery().execute();

    }

}

public void openSignUp(View view){
    Intent signUpIntent = new Intent( packageContext: this, SignUpActivity.class);
    startActivity(signUpIntent);
}
```

This code performs the multithreaded task of sending and retrieving the login data to the database



```
private class SelectQuery extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... voids) {
        try {
            // Loads MySQL driver to our program
            Class.forName("com.mysql.jdbc.Driver");
            // Establish database connection
            Connection conn = DriverManager.getConnection( url: "jdbc:mysql://studibuddi.cvo8hcorg85o.us-west-1.rds.amazonaws.com" +
                    ":3306/studibuddi?user=" + "masterbuddi" + "&password=myMasterPass");

            // If table is null, that email doesn't exist
            // If rs is length 1 it exists

            PreparedStatement myStmt = conn.prepareStatement( s: "SELECT bEmail, bPassword, bFirstName, bLastName FROM Buddi WHERE bEmail =?");
            myStmt.setString( i: 1, email);
            ResultSet rs = myStmt.executeQuery();

            // Print each student's ID and name
            if (rs.first()) {
                DBemail = rs.getString( s: "bEmail");
                DBpassword = rs.getString( s: "bPassword");
                DBfirstName = rs.getString( s: "bFirstName");
                DBlastName = rs.getString( s: "bLastName");
                System.out.println(DBemail + " : " + DBpassword);
            }
            else{
                System.out.println("NO USER FOUND");
            }
        }
        catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
```

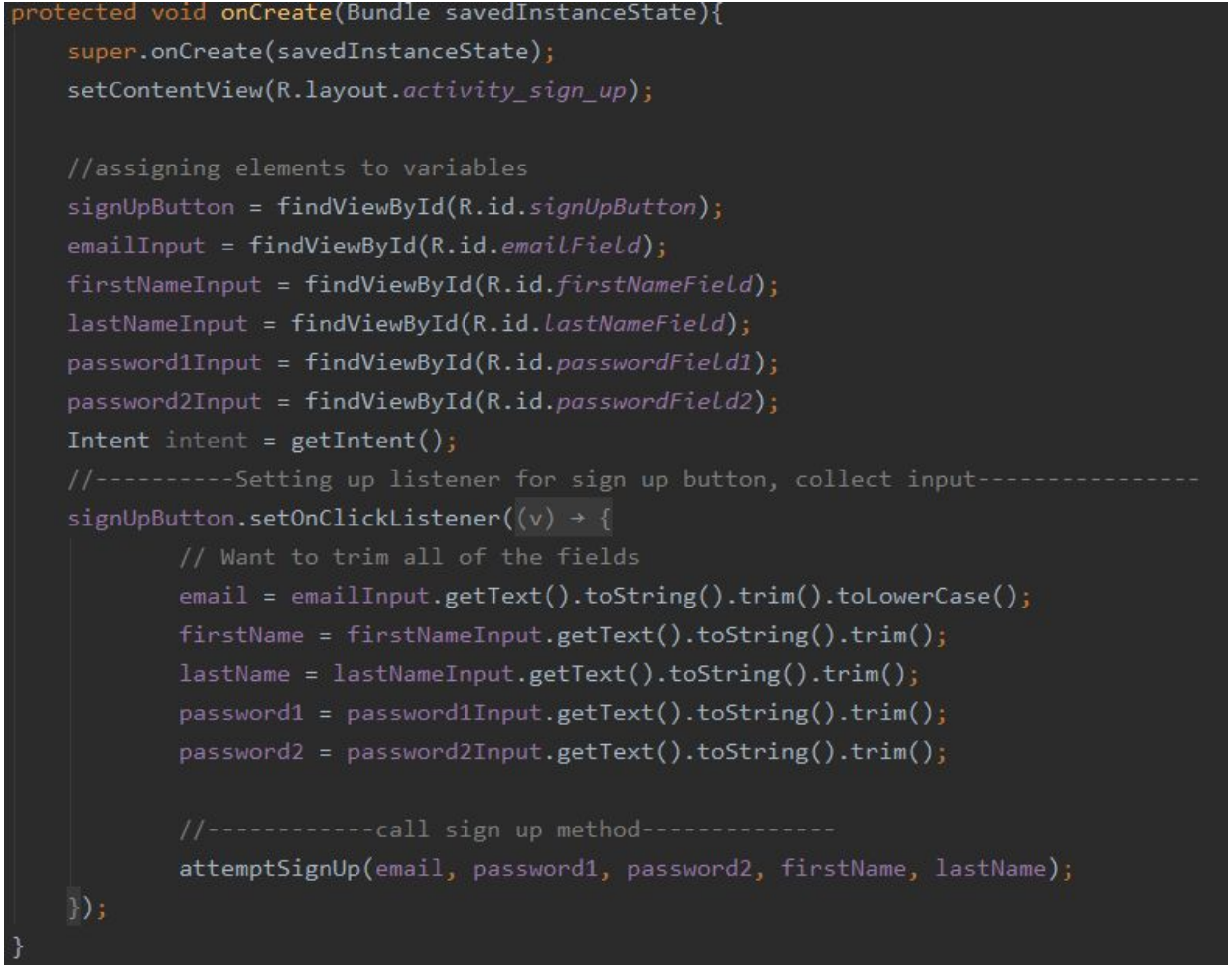
This code handles the data from the database with error checking from the login query



```
        // Print each student's ID and name
        if (rs.first()) {
            DBemail = rs.getString( s: "bEmail");
            DBpassword = rs.getString( s: "bPassword");
            DBfirstName = rs.getString( s: "bFirstName");
            DBlastName = rs.getString( s: "bLastName");
            System.out.println(DBemail + " : " + DBpassword);
        }
        else{
            System.out.println("NO USER FOUND");
        }
    }
    catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    }
    return null;
}

@Override
protected void onPostExecute(String s) {
    //super.onPostExecute(s);
    // If it matches a database entry, temp test right now
    if (DBemail.equals(email) && DBpassword.equals(password)){
        Intent MainIntent = new Intent(getApplicationContext(), MainActivity.class);
        MainIntent.putExtra( name: "EMAIL", email);
        currentUserLoggedIn = DBfirstName + ":" + DBlastName;
        startActivity(MainIntent);
    }
    else{
        Toast invalidToast = Toast.makeText(getApplicationContext(),  text: "Invalid login credentials", Toast.LENGTH_LONG);
        invalidToast.show();
        passEntry.setText(null);
    }
```

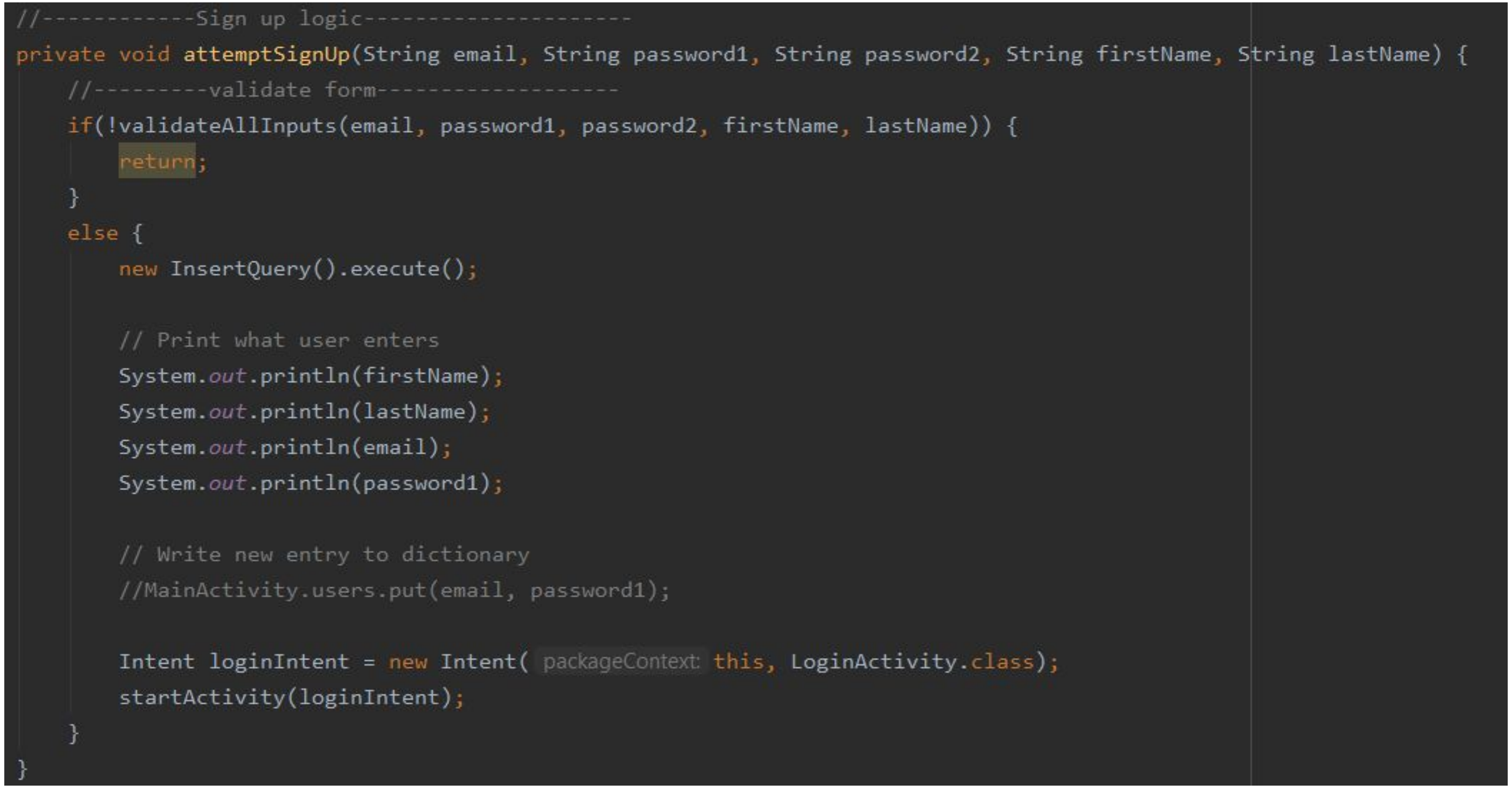
SignUpActivity

This code initializes the UI elements and gets the items via a listener on the button a user presses



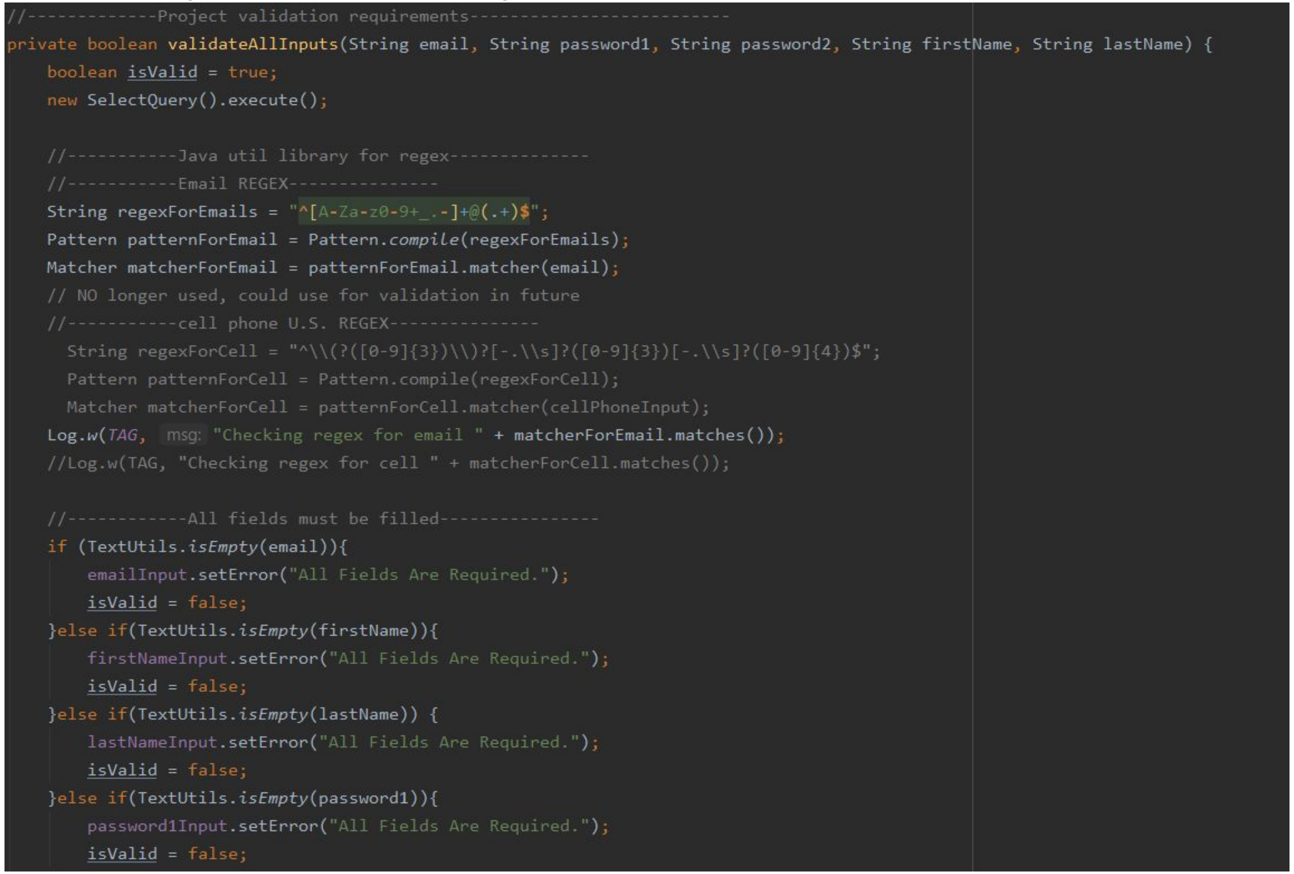
```java
protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sign_up);

    //assigning elements to variables
    signUpButton = findViewById(R.id.signUpButton);
    emailInput = findViewById(R.id.emailField);
    firstNameInput = findViewById(R.id.firstNameField);
    lastNameInput = findViewById(R.id.lastNameField);
    password1Input = findViewById(R.id.passwordField1);
    password2Input = findViewById(R.id.passwordField2);
    Intent intent = getIntent();
    //---------Setting up listener for sign up button, collect input---------------
    signUpButton.setOnClickListener((v) → {
            // Want to trim all of the fields
            email = emailInput.getText().toString().trim().toLowerCase();
            firstName = firstNameInput.getText().toString().trim();
            lastName = lastNameInput.getText().toString().trim();
            password1 = password1Input.getText().toString().trim();
            password2 = password2Input.getText().toString().trim();

            //-----------call sign up method-------------
            attemptSignUp(email, password1, password2, firstName, lastName);
    });
}
```

This method calls the input validation method for all of the user input, if successful interfaces with the DB and goes to login screen, else gives an error message.



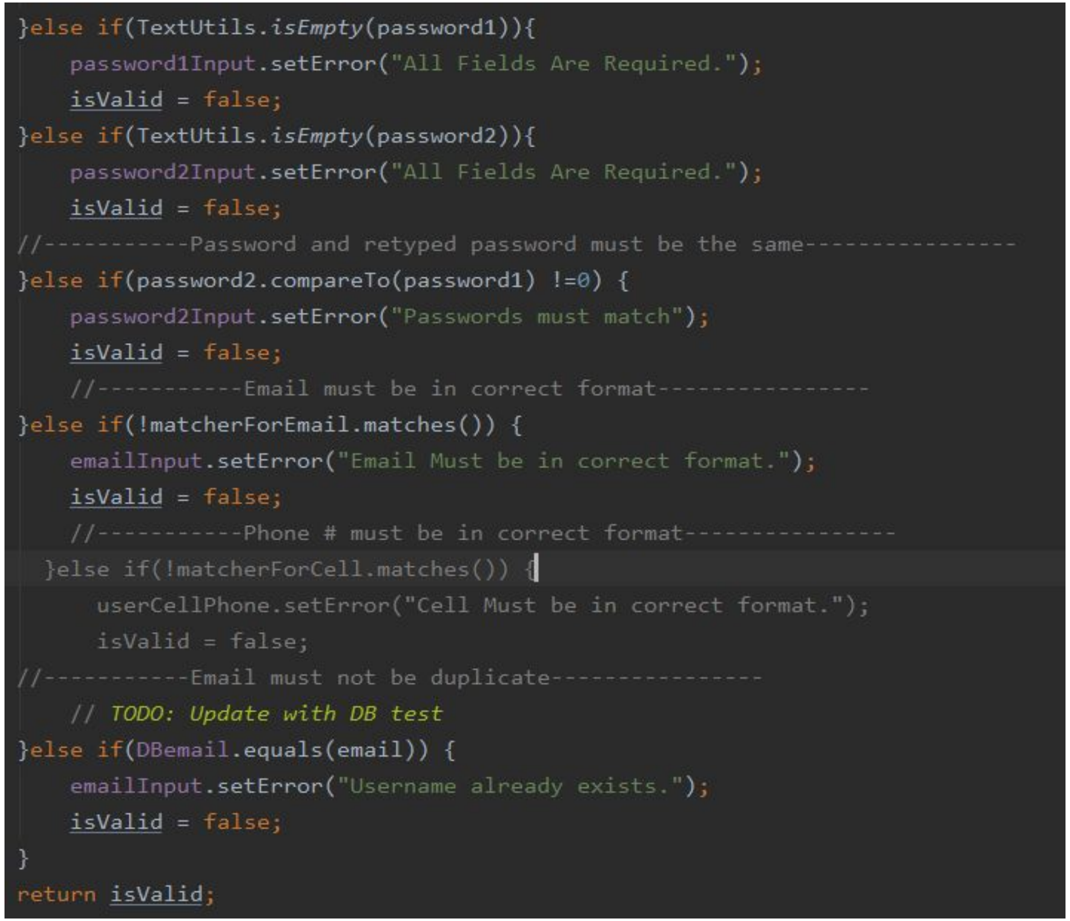
```java
//-----------Sign up logic--------------------
private void attemptSignUp(String email, String password1, String password2, String firstName, String lastName) {
    //--------validate form------------------
    if(!validateAllInputs(email, password1, password2, firstName, lastName)) {
        return;
    }
    else {
        new InsertQuery().execute();

        // Print what user enters
        System.out.println(firstName);
        System.out.println(lastName);
        System.out.println(email);
        System.out.println(password1);

        // Write new entry to dictionary
        //MainActivity.users.put(email, password1);

        Intent loginIntent = new Intent( packageContext: this, LoginActivity.class);
        startActivity(loginIntent);
    }
}
```

This function checks every input field from the user, validates that they are correct inputs. (cont next slide)



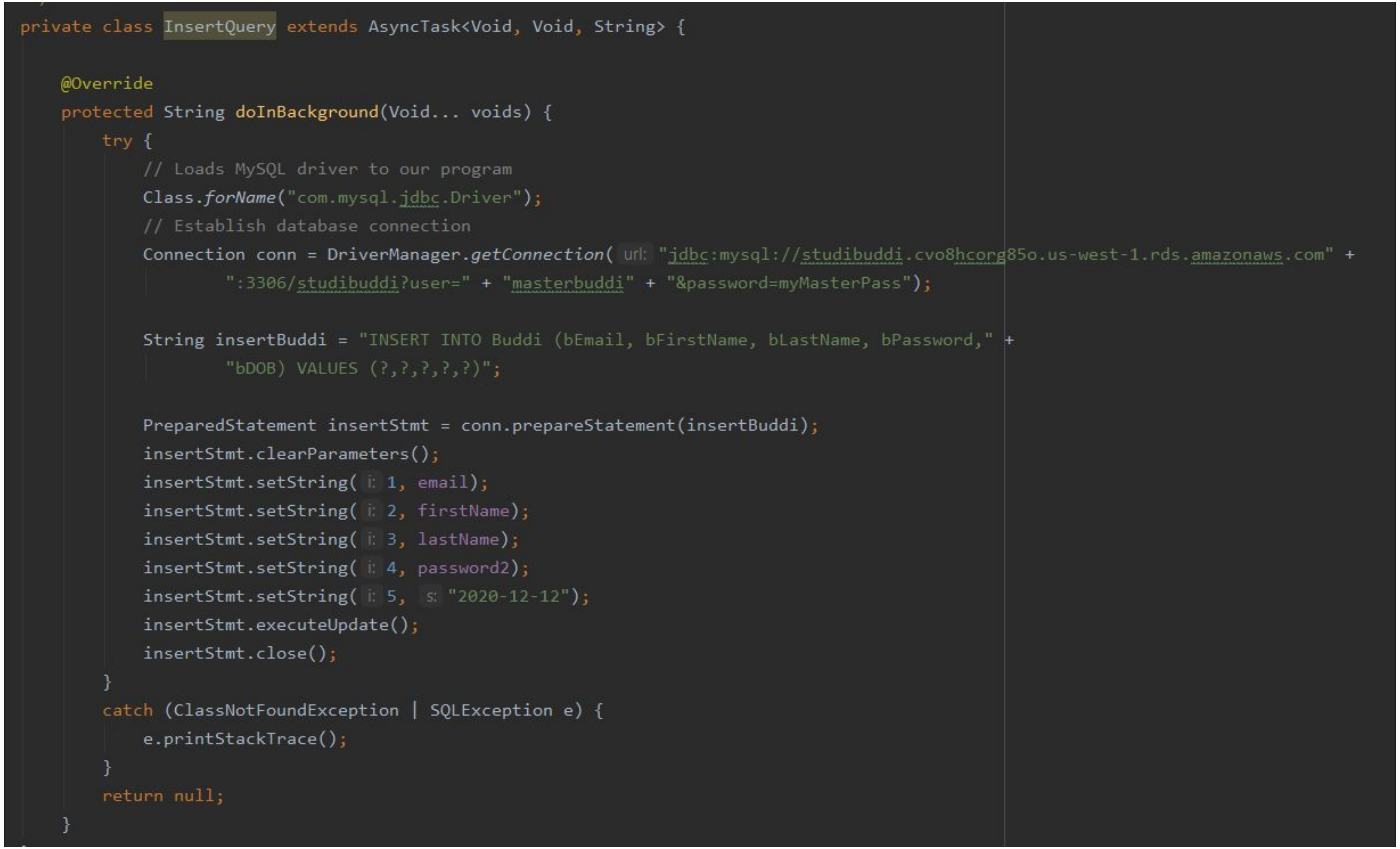
```java
//------------Project validation requirements-------------------------
private boolean validateAllInputs(String email, String password1, String password2, String firstName, String lastName) {
    boolean isValid = true;
    new SelectQuery().execute();

    //----------Java util library for regex-------------
    //----------Email REGEX--------------
    String regexForEmails = "^[A-Za-z0-9+_.-]+@(.+)$";
    Pattern patternForEmail = Pattern.compile(regexForEmails);
    Matcher matcherForEmail = patternForEmail.matcher(email);
    // NO longer used, could use for validation in future
    //----------cell phone U.S. REGEX--------------
      String regexForCell = "^\\(?([0-9]{3})\\)?[-.\\s]?([0-9]{3})[-.\\s]?([0-9]{4})$";
      Pattern patternForCell = Pattern.compile(regexForCell);
      Matcher matcherForCell = patternForCell.matcher(cellPhoneInput);
    Log.w(TAG, msg: "Checking regex for email " + matcherForEmail.matches());
    //Log.w(TAG, "Checking regex for cell " + matcherForCell.matches());

    //-----------All fields must be filled---------------
    if (TextUtils.isEmpty(email)){
        emailInput.setError("All Fields Are Required.");
        isValid = false;
    }else if(TextUtils.isEmpty(firstName)){
        firstNameInput.setError("All Fields Are Required.");
        isValid = false;
    }else if(TextUtils.isEmpty(lastName)) {
        lastNameInput.setError("All Fields Are Required.");
        isValid = false;
    }else if(TextUtils.isEmpty(password1)){
        password1Input.setError("All Fields Are Required.");
        isValid = false;
```

Gives specific error messages on the fields that are incomplete or incorrectly formatted (i.e. email or phone #)



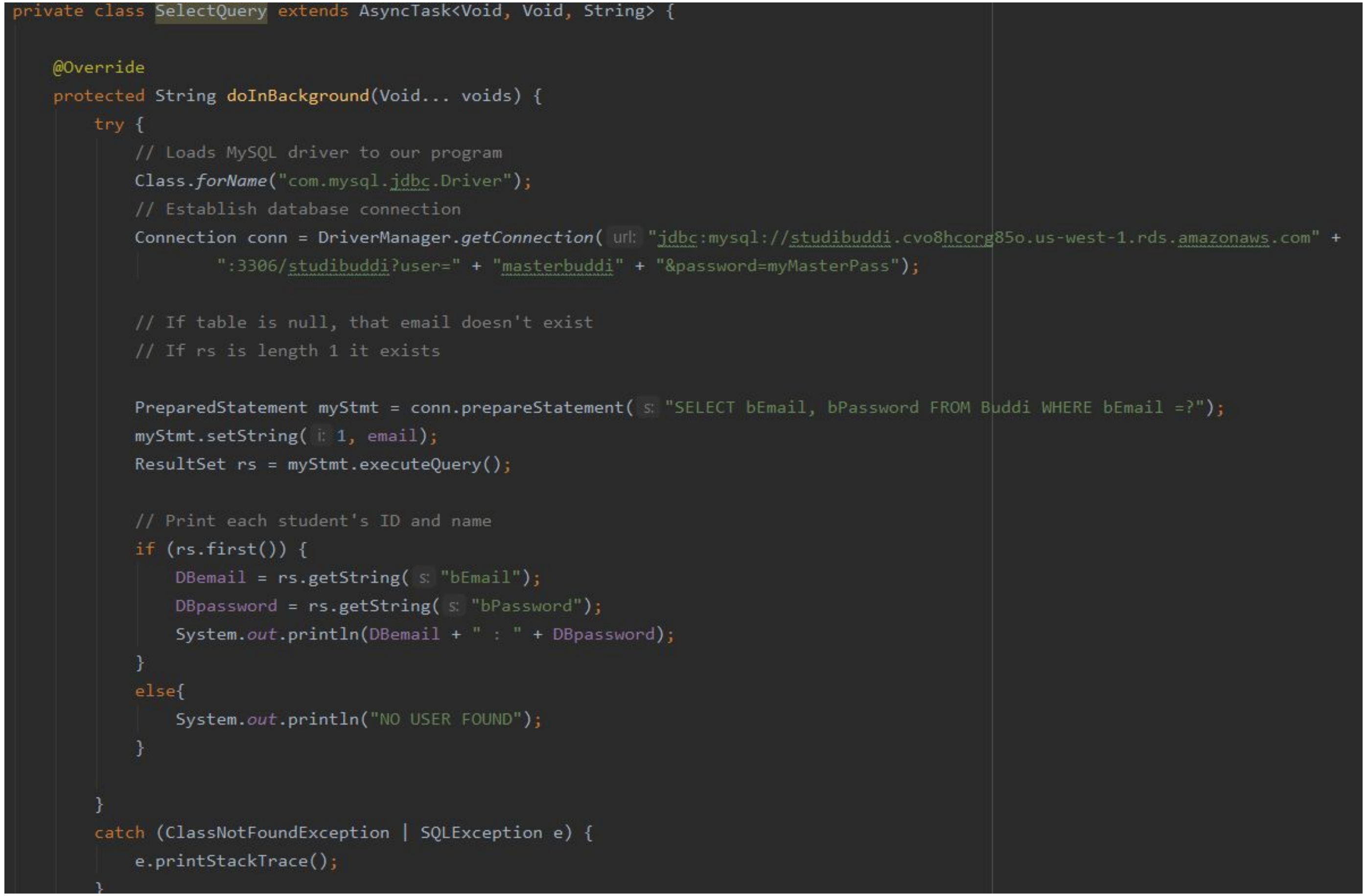
```java
}else if(TextUtils.isEmpty(password1)){
    password1Input.setError("All Fields Are Required.");
    isValid = false;
}else if(TextUtils.isEmpty(password2)){
    password2Input.setError("All Fields Are Required.");
    isValid = false;
//----------Password and retyped password must be the same---------------
}else if(password2.compareTo(password1) !=0) {
    password2Input.setError("Passwords must match");
    isValid = false;
    //----------Email must be in correct format---------------
}else if(!matcherForEmail.matches()) {
    emailInput.setError("Email Must be in correct format.");
    isValid = false;
    //----------Phone # must be in correct format---------------
 }else if(!matcherForCell.matches()) {
     userCellPhone.setError("Cell Must be in correct format.");
     isValid = false;
//----------Email must not be duplicate---------------
    // TODO: Update with DB test
}else if(DBemail.equals(email)) {
    emailInput.setError("Username already exists.");
    isValid = false;
}
return isValid;
```

Sign up Database interface and query to insert a new user into the DB



```java
private class InsertQuery extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... voids) {
        try {
            // Loads MySQL driver to our program
            Class.forName("com.mysql.jdbc.Driver");
            // Establish database connection
            Connection conn = DriverManager.getConnection( url: "jdbc:mysql://studibuddi.cvo8hcorg85o.us-west-1.rds.amazonaws.com" +
                    ":3306/studibuddi?user=" + "masterbuddi" + "&password=myMasterPass");

            String insertBuddi = "INSERT INTO Buddi (bEmail, bFirstName, bLastName, bPassword," +
                    "bDOB) VALUES (?,?,?,?,?)";

            PreparedStatement insertStmt = conn.prepareStatement(insertBuddi);
            insertStmt.clearParameters();
            insertStmt.setString( i: 1, email);
            insertStmt.setString( i: 2, firstName);
            insertStmt.setString( i: 3, lastName);
            insertStmt.setString( i: 4, password2);
            insertStmt.setString( i: 5,  s: "2020-12-12");
            insertStmt.executeUpdate();
            insertStmt.close();
        }
        catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
```

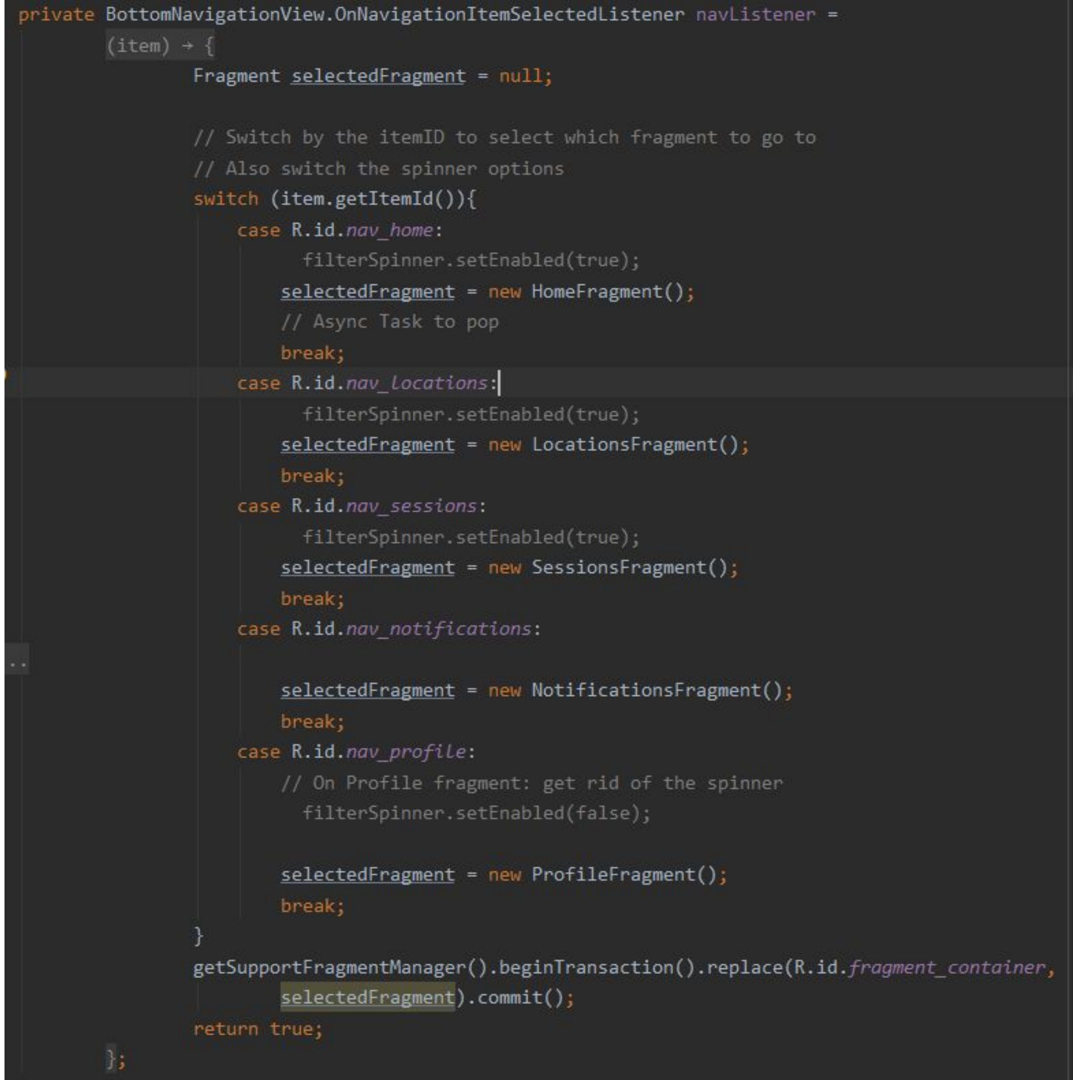
Database Query to check if the user already exists in the DB during sign up



```
private class SelectQuery extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... voids) {
        try {
            // Loads MySQL driver to our program
            Class.forName("com.mysql.jdbc.Driver");
            // Establish database connection
            Connection conn = DriverManager.getConnection( url: "jdbc:mysql://studibuddi.cvo8hcorg85o.us-west-1.rds.amazonaws.com" +
                    ":3306/studibuddi?user=" + "masterbuddi" + "&password=myMasterPass");

            // If table is null, that email doesn't exist
            // If rs is length 1 it exists

            PreparedStatement myStmt = conn.prepareStatement( s: "SELECT bEmail, bPassword FROM Buddi WHERE bEmail =?");
            myStmt.setString( i: 1, email);
            ResultSet rs = myStmt.executeQuery();

            // Print each student's ID and name
            if (rs.first()) {
                DBemail = rs.getString( s: "bEmail");
                DBpassword = rs.getString( s: "bPassword");
                System.out.println(DBemail + " : " + DBpassword);
            }
            else{
                System.out.println("NO USER FOUND");
            }

        }
        catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
```

MainActivity

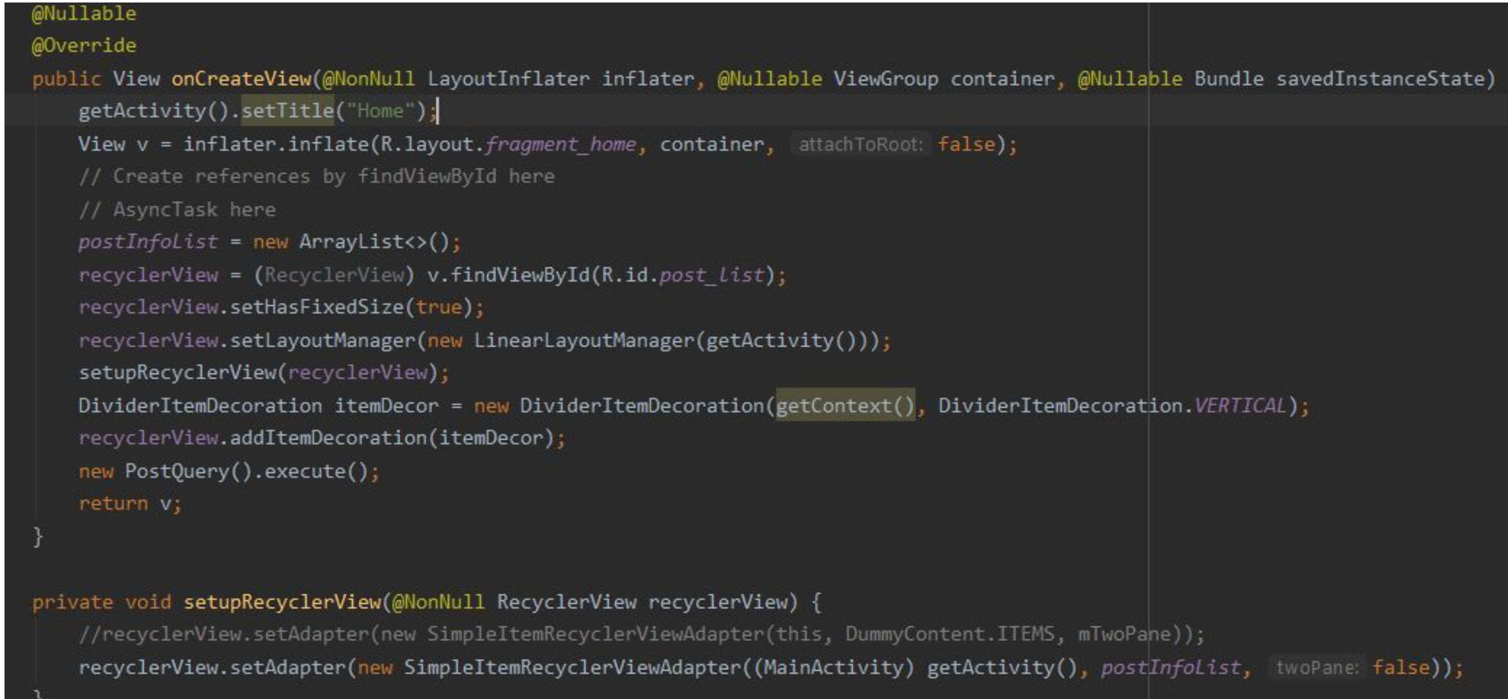
This function is the listener for our BottomNavigationBar, the bar that lets us select which aspect of the app we want to go to (e.g. home screen, profile, notifications, locations, or studi sessions). It switches between UI fragments for quick and efficient transfer of data and saves on memory.



```
private BottomNavigationView.OnNavigationItemSelectedListener navListener =
        (item) → {
            Fragment selectedFragment = null;

            // Switch by the itemID to select which fragment to go to
            // Also switch the spinner options
            switch (item.getItemId()){
                case R.id.nav_home:
                      filterSpinner.setEnabled(true);
                    selectedFragment = new HomeFragment();
                    // Async Task to pop
                    break;
                case R.id.nav_locations:
                      filterSpinner.setEnabled(true);
                    selectedFragment = new LocationsFragment();
                    break;
                case R.id.nav_sessions:
                      filterSpinner.setEnabled(true);
                    selectedFragment = new SessionsFragment();
                    break;
                case R.id.nav_notifications:

                    selectedFragment = new NotificationsFragment();
                    break;
                case R.id.nav_profile:
                    // On Profile fragment: get rid of the spinner
                      filterSpinner.setEnabled(false);

                    selectedFragment = new ProfileFragment();
                    break;
            }
            getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
                    selectedFragment).commit();
            return true;
        };
```

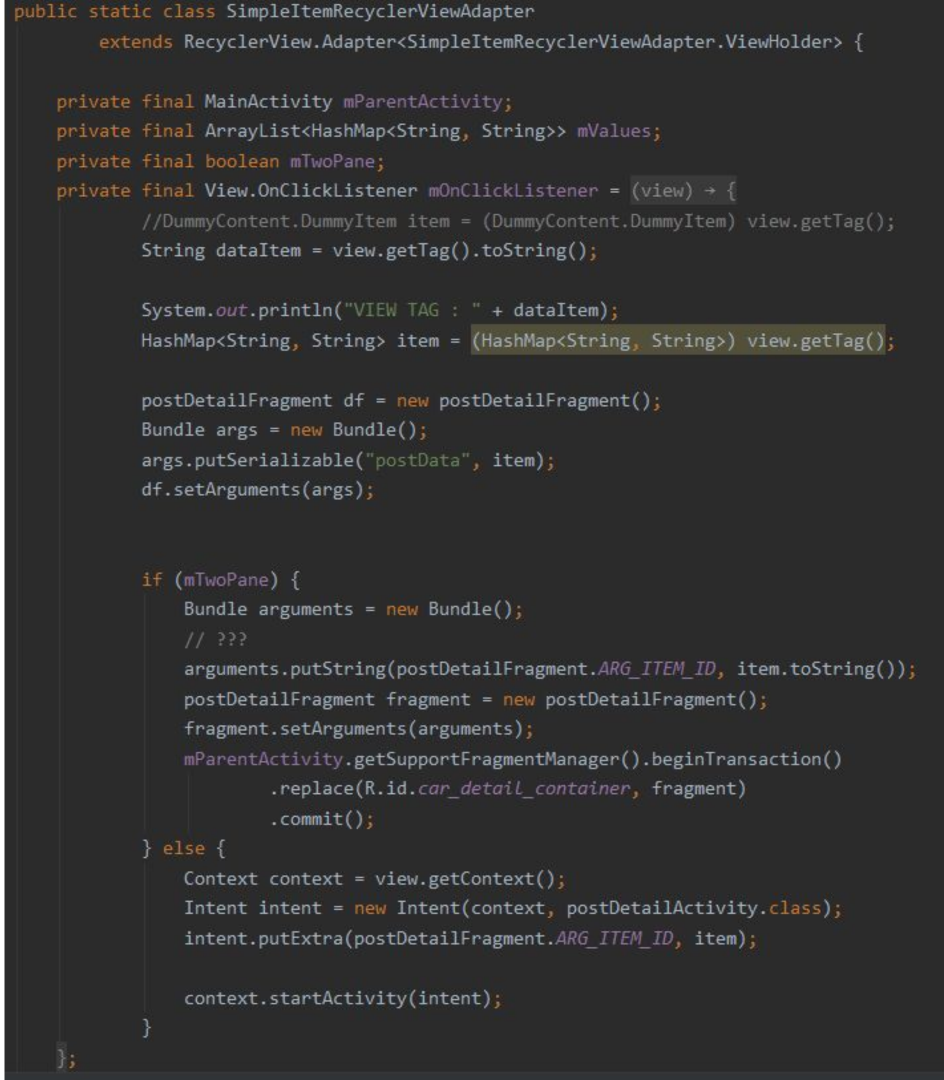
HomeFragment

This function is the UI creator for the home screen, it inflates the layout and includes a call to a constructor for the RecyclerView which we use to create and display post data efficiently.



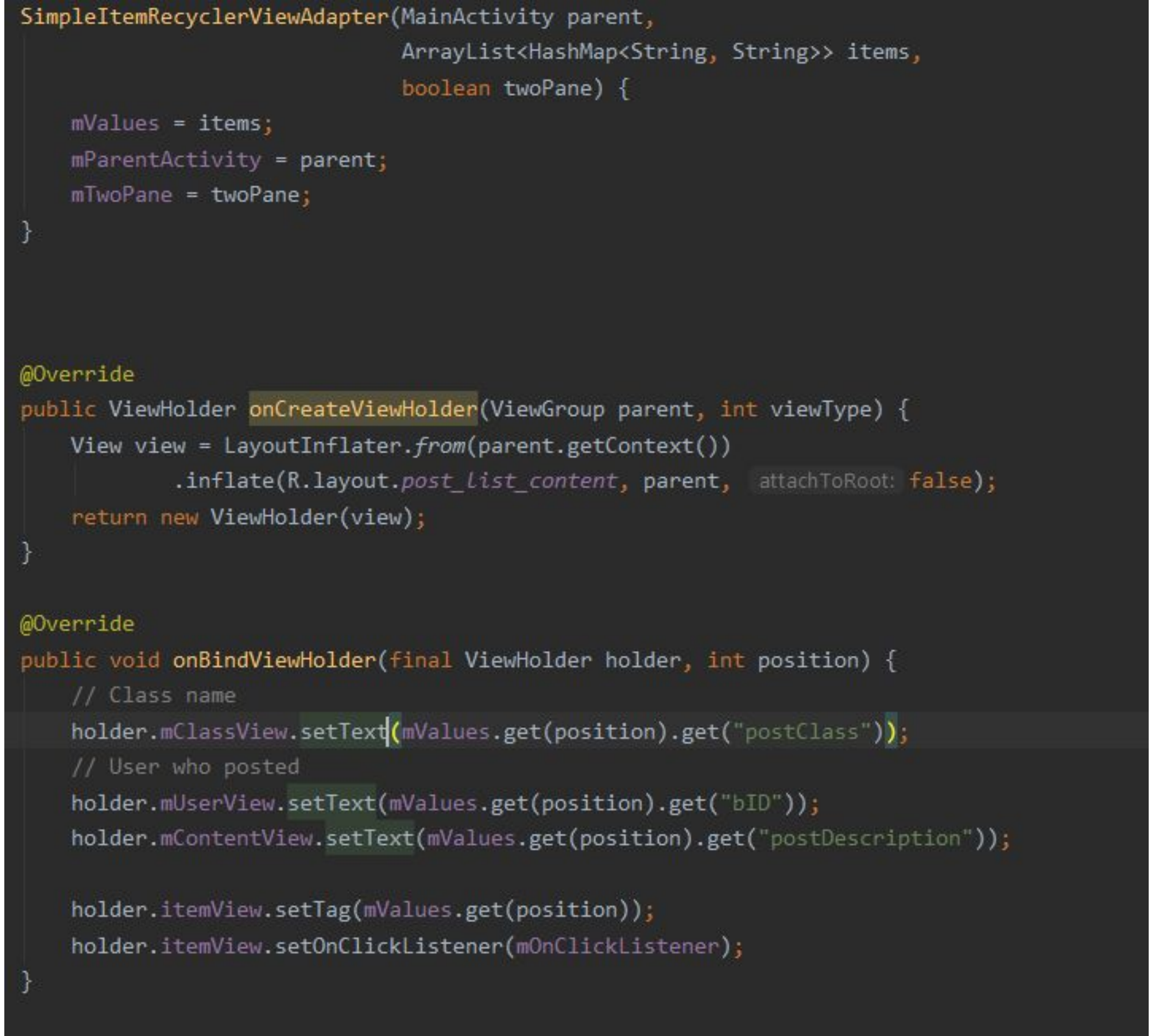
```java
@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState)
    getActivity().setTitle("Home");
    View v = inflater.inflate(R.layout.fragment_home, container,  attachToRoot: false);
    // Create references by findViewById here
    // AsyncTask here
    postInfoList = new ArrayList<>();
    recyclerView = (RecyclerView) v.findViewById(R.id.post_list);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
    setupRecyclerView(recyclerView);
    DividerItemDecoration itemDecor = new DividerItemDecoration(getContext(), DividerItemDecoration.VERTICAL);
    recyclerView.addItemDecoration(itemDecor);
    new PostQuery().execute();
    return v;
}

private void setupRecyclerView(@NonNull RecyclerView recyclerView) {
    //recyclerView.setAdapter(new SimpleItemRecyclerViewAdapter(this, DummyContent.ITEMS, mTwoPane));
    recyclerView.setAdapter(new SimpleItemRecyclerViewAdapter((MainActivity) getActivity(), postInfoList,  twoPane: false));
```

This class contains data on our RecyclerViewAdapter that allows us to dictate what happens when we click on a post. This is different for tablet or wide views than on a vertical smartphone. This is dictated by a relative width taken from earlier in the program, defined in our MainActivity. There are also dynamic layouts depending on the device used.



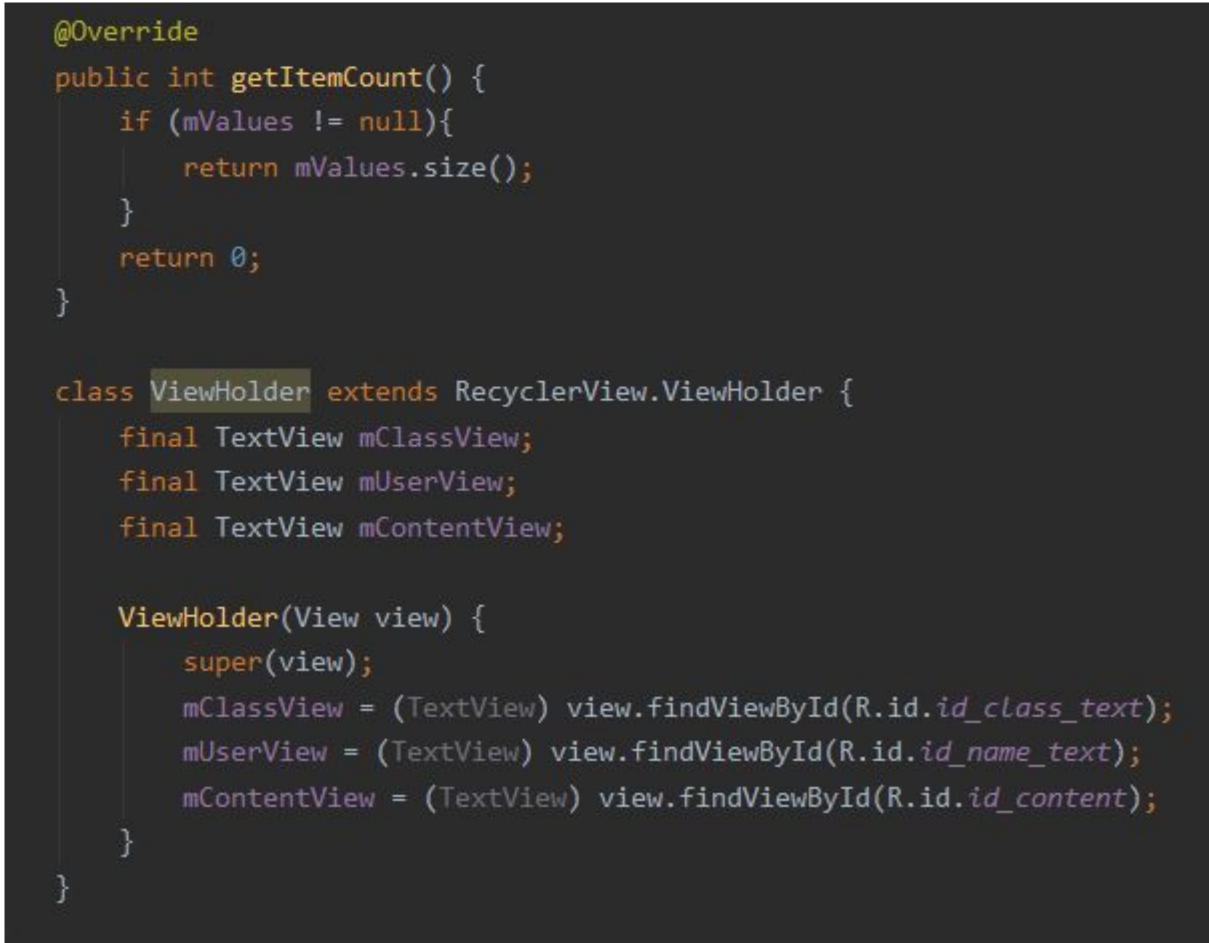
```java
public static class SimpleItemRecyclerViewAdapter
        extends RecyclerView.Adapter<SimpleItemRecyclerViewAdapter.ViewHolder> {

    private final MainActivity mParentActivity;
    private final ArrayList<HashMap<String, String>> mValues;
    private final boolean mTwoPane;
    private final View.OnClickListener mOnClickListener = (view) → {
            //DummyContent.DummyItem item = (DummyContent.DummyItem) view.getTag();
            String dataItem = view.getTag().toString();

            System.out.println("VIEW TAG : " + dataItem);
            HashMap<String, String> item = (HashMap<String, String>) view.getTag();

            postDetailFragment df = new postDetailFragment();
            Bundle args = new Bundle();
            args.putSerializable("postData", item);
            df.setArguments(args);


            if (mTwoPane) {
                Bundle arguments = new Bundle();
                // ???
                arguments.putString(postDetailFragment.ARG_ITEM_ID, item.toString());
                postDetailFragment fragment = new postDetailFragment();
                fragment.setArguments(arguments);
                mParentActivity.getSupportFragmentManager().beginTransaction()
                        .replace(R.id.car_detail_container, fragment)
                        .commit();
            } else {
                Context context = view.getContext();
                Intent intent = new Intent(context, postDetailActivity.class);
                intent.putExtra(postDetailFragment.ARG_ITEM_ID, item);

                context.startActivity(intent);
            }
    };
```

This contains the constructor for the recyclerview interactable object, and some helper methods for the UI



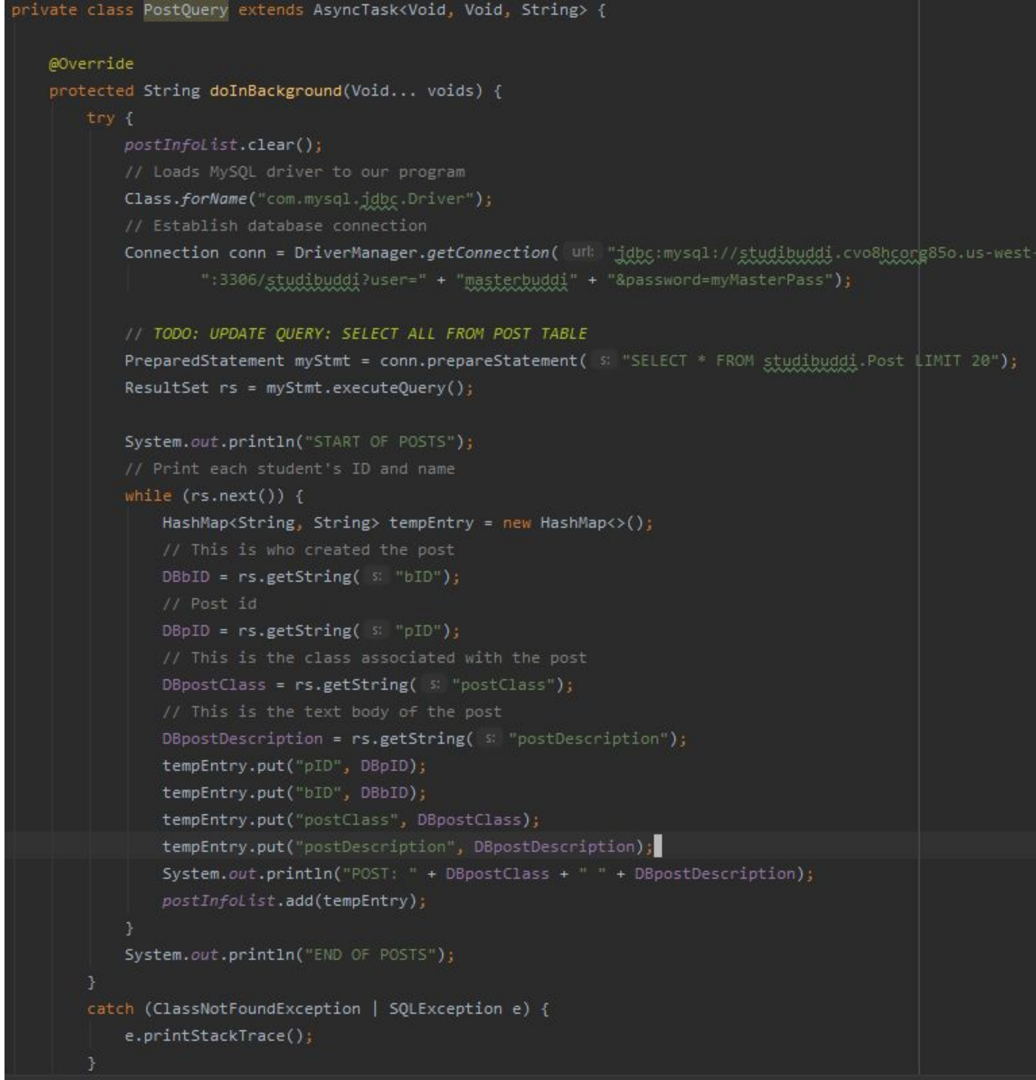
```
SimpleItemRecyclerViewAdapter(MainActivity parent,
                              ArrayList<HashMap<String, String>> items,
                              boolean twoPane) {
    mValues = items;
    mParentActivity = parent;
    mTwoPane = twoPane;
}



@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.post_list_content, parent,  attachToRoot: false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(final ViewHolder holder, int position) {
    // Class name
    holder.mClassView.setText(mValues.get(position).get("postClass"));
    // User who posted
    holder.mUserView.setText(mValues.get(position).get("bID"));
    holder.mContentView.setText(mValues.get(position).get("postDescription"));

    holder.itemView.setTag(mValues.get(position));
    holder.itemView.setOnClickListener(mOnClickListener);
}
```

Additional helper methods for the RecyclerView to get the number of items in the View, and to anchor the text elements onto the RecyclerView object



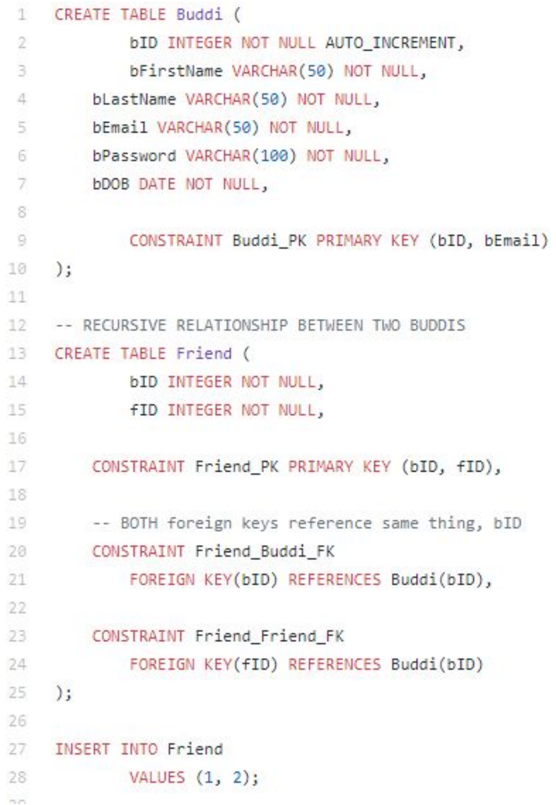
```java
@Override
public int getItemCount() {
    if (mValues != null){
        return mValues.size();
    }
    return 0;
}

class ViewHolder extends RecyclerView.ViewHolder {
    final TextView mClassView;
    final TextView mUserView;
    final TextView mContentView;

    ViewHolder(View view) {
        super(view);
        mClassView = (TextView) view.findViewById(R.id.id_class_text);
        mUserView = (TextView) view.findViewById(R.id.id_name_text);
        mContentView = (TextView) view.findViewById(R.id.id_content);
    }
}
```

Query from the database to get all of the posts for the user to see on their homescreen.



```
private class PostQuery extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... voids) {
        try {
            postInfoList.clear();
            // Loads MySQL driver to our program
            Class.forName("com.mysql.jdbc.Driver");
            // Establish database connection
            Connection conn = DriverManager.getConnection( url: "jdbc:mysql://studibuddi.cvo8hcorg85o.us-west-
                    ":3306/studibuddi?user=" + "masterbuddi" + "&password=myMasterPass");

            // TODO: UPDATE QUERY: SELECT ALL FROM POST TABLE
            PreparedStatement myStmt = conn.prepareStatement( s: "SELECT * FROM studibuddi.Post LIMIT 20");
            ResultSet rs = myStmt.executeQuery();

            System.out.println("START OF POSTS");
            // Print each student's ID and name
            while (rs.next()) {
                HashMap<String, String> tempEntry = new HashMap<>();
                // This is who created the post
                DBbID = rs.getString( s: "bID");
                // Post id
                DBpID = rs.getString( s: "pID");
                // This is the class associated with the post
                DBpostClass = rs.getString( s: "postClass");
                // This is the text body of the post
                DBpostDescription = rs.getString( s: "postDescription");
                tempEntry.put("pID", DBpID);
                tempEntry.put("bID", DBbID);
                tempEntry.put("postClass", DBpostClass);
                tempEntry.put("postDescription", DBpostDescription);
                System.out.println("POST: " + DBpostClass + " " + DBpostDescription);
                postInfoList.add(tempEntry);
            }
            System.out.println("END OF POSTS");
        }
        catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
```

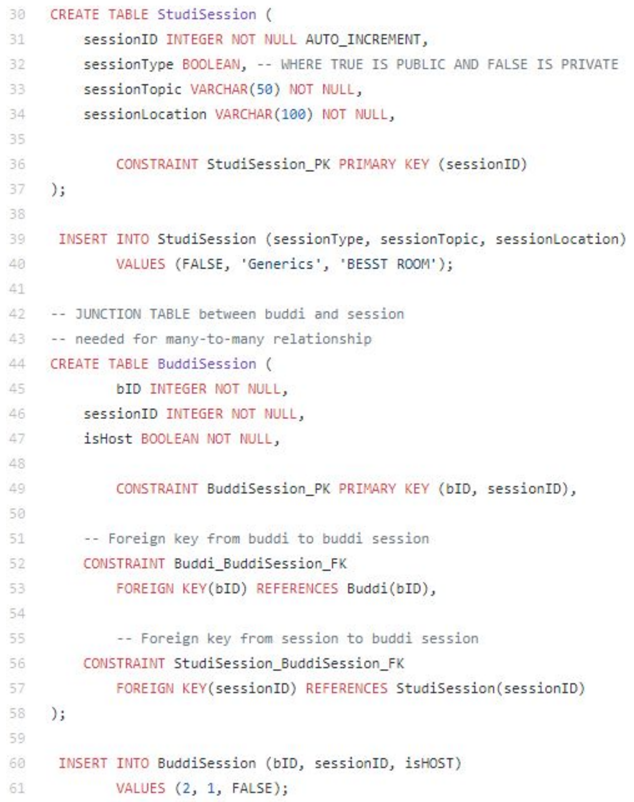
DDL (1 of 4)



```
CREATE TABLE Buddi (
        bID INTEGER NOT NULL AUTO_INCREMENT,
        bFirstName VARCHAR(50) NOT NULL,
    bLastName VARCHAR(50) NOT NULL,
    bEmail VARCHAR(50) NOT NULL,
    bPassword VARCHAR(100) NOT NULL,
    bDOB DATE NOT NULL,

        CONSTRAINT Buddi_PK PRIMARY KEY (bID, bEmail)
);

-- RECURSIVE RELATIONSHIP BETWEEN TWO BUDDIS
CREATE TABLE Friend (
        bID INTEGER NOT NULL,
        fID INTEGER NOT NULL,

    CONSTRAINT Friend_PK PRIMARY KEY (bID, fID),

    -- BOTH foreign keys reference same thing, bID
    CONSTRAINT Friend_Buddi_FK
        FOREIGN KEY(bID) REFERENCES Buddi(bID),

    CONSTRAINT Friend_Friend_FK
        FOREIGN KEY(fID) REFERENCES Buddi(bID)
);

INSERT INTO Friend
        VALUES (1, 2);
```

- DDL is the code for the creation of the database’s tables.
- This snippet shows the tables for the Buddi (our users) and Friend

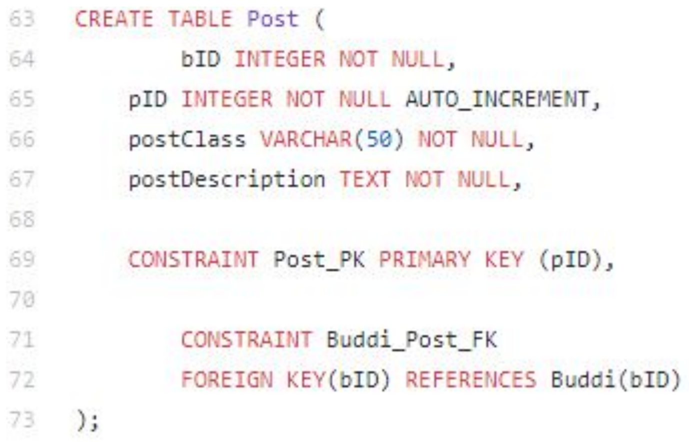
DDL (2 of 4)



```sql
CREATE TABLE StudiSession (
    sessionID INTEGER NOT NULL AUTO_INCREMENT,
    sessionType BOOLEAN, -- WHERE TRUE IS PUBLIC AND FALSE IS PRIVATE
    sessionTopic VARCHAR(50) NOT NULL,
    sessionLocation VARCHAR(100) NOT NULL,

        CONSTRAINT StudiSession_PK PRIMARY KEY (sessionID)
);

 INSERT INTO StudiSession (sessionType, sessionTopic, sessionLocation)
        VALUES (FALSE, 'Generics', 'BESST ROOM');

-- JUNCTION TABLE between buddi and session
-- needed for many-to-many relationship
CREATE TABLE BuddiSession (
        bID INTEGER NOT NULL,
    sessionID INTEGER NOT NULL,
    isHost BOOLEAN NOT NULL,

        CONSTRAINT BuddiSession_PK PRIMARY KEY (bID, sessionID),

    -- Foreign key from buddi to buddi session
    CONSTRAINT Buddi_BuddiSession_FK
        FOREIGN KEY(bID) REFERENCES Buddi(bID),

        -- Foreign key from session to buddi session
    CONSTRAINT StudiSession_BuddiSession_FK
        FOREIGN KEY(sessionID) REFERENCES StudiSession(sessionID)
);

 INSERT INTO BuddiSession (bID, sessionID, isHOST)
        VALUES (2, 1, FALSE);
```

- This snippet creates the table for our StudiSession (our meetings between users) and our BuddiSession (the child class between Buddi and StudiSession).

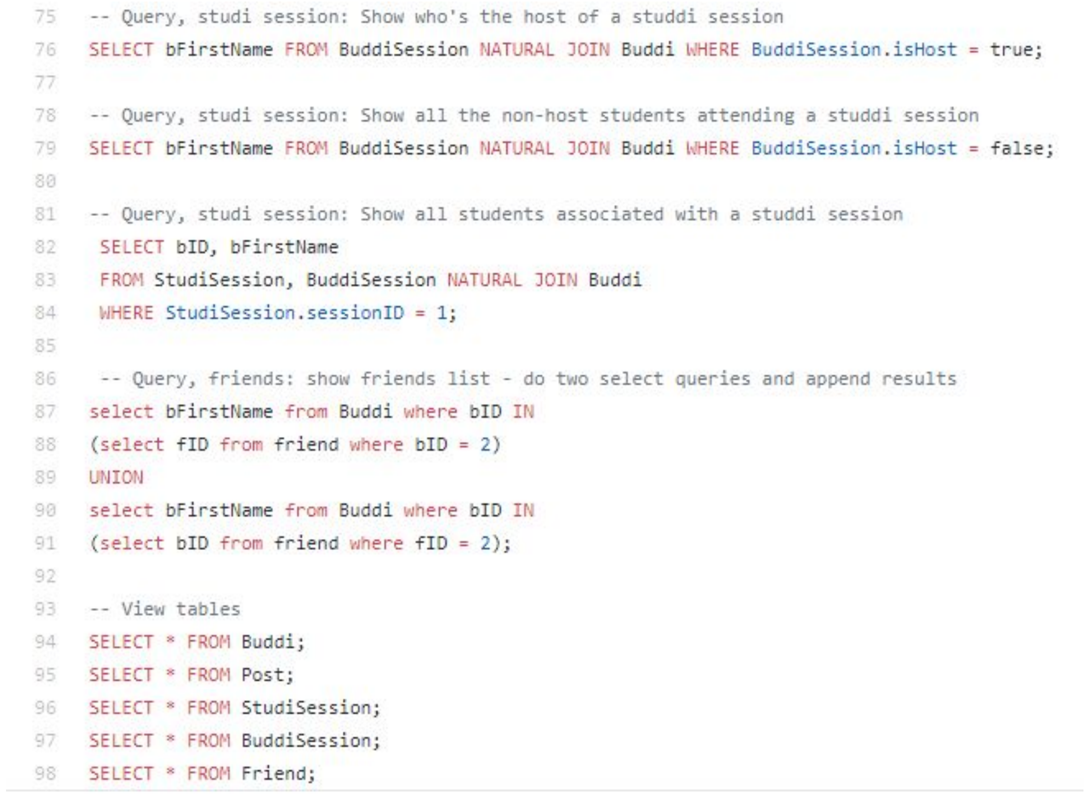
DDL (3 of 4)



```
CREATE TABLE Post (
        bID INTEGER NOT NULL,
    pID INTEGER NOT NULL AUTO_INCREMENT,
    postClass VARCHAR(50) NOT NULL,
    postDescription TEXT NOT NULL,

    CONSTRAINT Post_PK PRIMARY KEY (pID),

        CONSTRAINT Buddi_Post_FK
        FOREIGN KEY(bID) REFERENCES Buddi(bID)
);
```

- This snippet creates the table for our app’s Posts.

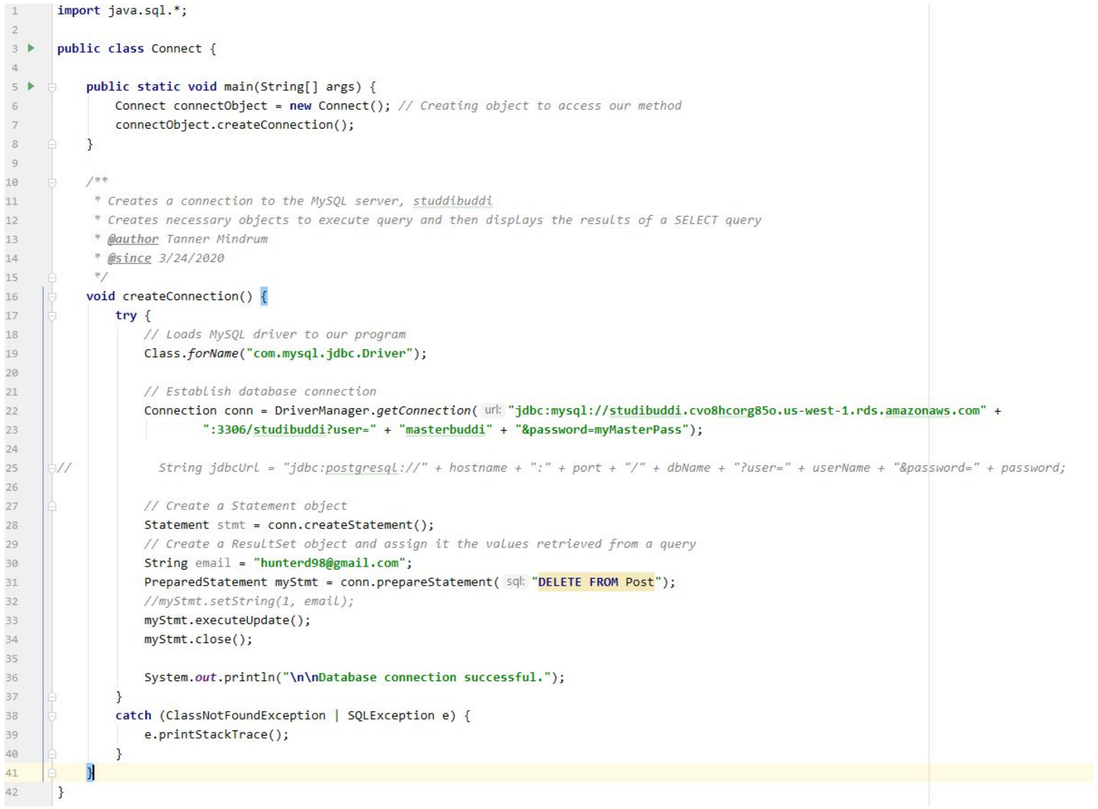
DDL (4 of 4)



```sql
-- Query, studi session: Show who's the host of a studdi session
SELECT bFirstName FROM BuddiSession NATURAL JOIN Buddi WHERE BuddiSession.isHost = true;

-- Query, studi session: Show all the non-host students attending a studdi session
SELECT bFirstName FROM BuddiSession NATURAL JOIN Buddi WHERE BuddiSession.isHost = false;

-- Query, studi session: Show all students associated with a studdi session
 SELECT bID, bFirstName
 FROM StudiSession, BuddiSession NATURAL JOIN Buddi
 WHERE StudiSession.sessionID = 1;

 -- Query, friends: show friends list - do two select queries and append results
select bFirstName from Buddi where bID IN
(select fID from friend where bID = 2)
UNION
select bFirstName from Buddi where bID IN
(select bID from friend where fID = 2);

-- View tables
SELECT * FROM Buddi;
SELECT * FROM Post;
SELECT * FROM StudiSession;
SELECT * FROM BuddiSession;
SELECT * FROM Friend;
```

- This snippet shows some of the queries to see if the database worked and to test connectivity

Connect.Java



```java
import java.sql.*;

public class Connect {

    public static void main(String[] args) {
        Connect connectObject = new Connect(); // Creating object to access our method
        connectObject.createConnection();
    }

    /**
     * Creates a connection to the MySQL server, studdibuddi
     * Creates necessary objects to execute query and then displays the results of a SELECT query
     * @author Tanner Mindrum
     * @since 3/24/2020
     */
    void createConnection() {
        try {
            // Loads MySQL driver to our program
            Class.forName("com.mysql.jdbc.Driver");

            // Establish database connection
            Connection conn = DriverManager.getConnection( url: "jdbc:mysql://studibuddi.cvo8hcorg85o.us-west-1.rds.amazonaws.com" +
                    ":3306/studibuddi?user=" + "masterbuddi" + "&password=myMasterPass");

//            String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?user=" + userName + "&password=" + password;

            // Create a Statement object
            Statement stmt = conn.createStatement();
            // Create a ResultSet object and assign it the values retrieved from a query
            String email = "hunterd98@gmail.com";
            PreparedStatement myStmt = conn.prepareStatement( sql: "DELETE FROM Post");
            //myStmt.setString(1, email);
            myStmt.executeUpdate();
            myStmt.close();

            System.out.println("\n\nDatabase connection successful.");
        }
        catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- This class is dedicated to setting up connectivity between app and database. Shows example of communication between front-end and back-end.