SECTION C

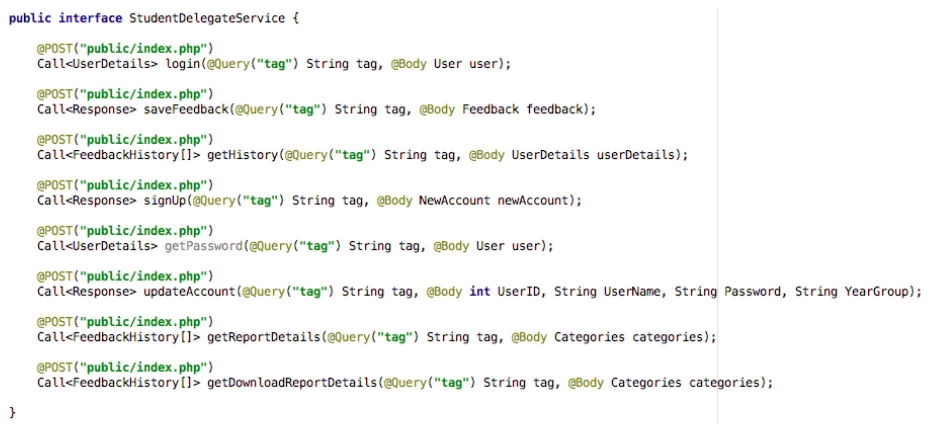
USE OF LIBRARIES AND OTHER TOOLS

RETROFIT (24/07/18)

My advisor advised me to use the "Retrofit"[1] library to connect to the database via my APIs in my application (Appendix C). Retrofit made it easier for me to retrieve and upload data using JSON[2].

STUDENTDELEGATESERVICE INTERFACE (24/07/18)

I created this interface to help facilitate communication with my API. This interface worked in conjunction with the "Retrofit" library.



```
public interface StudentDelegateService {

    @POST("public/index.php")
    Call<UserDetails> login(@Query("tag") String tag, @Body User user);

    @POST("public/index.php")
    Call<Response> saveFeedback(@Query("tag") String tag, @Body Feedback feedback);

    @POST("public/index.php")
    Call<FeedbackHistory[]> getHistory(@Query("tag") String tag, @Body UserDetails userDetails);

    @POST("public/index.php")
    Call<Response> signUp(@Query("tag") String tag, @Body NewAccount newAccount);

    @POST("public/index.php")
    Call<UserDetails> getPassword(@Query("tag") String tag, @Body User user);

    @POST("public/index.php")
    Call<Response> updateAccount(@Query("tag") String tag, @Body int UserID, String UserName, String Password, String YearGroup);

    @POST("public/index.php")
    Call<FeedbackHistory[]> getReportDetails(@Query("tag") String tag, @Body Categories categories);

    @POST("public/index.php")
    Call<FeedbackHistory[]> getDownloadReportDetails(@Query("tag") String tag, @Body Categories categories);

}
```

The screenshot above shows the interface. These functions were called from within my control classes in Java.

SENDING EMAIL LIBRARIES

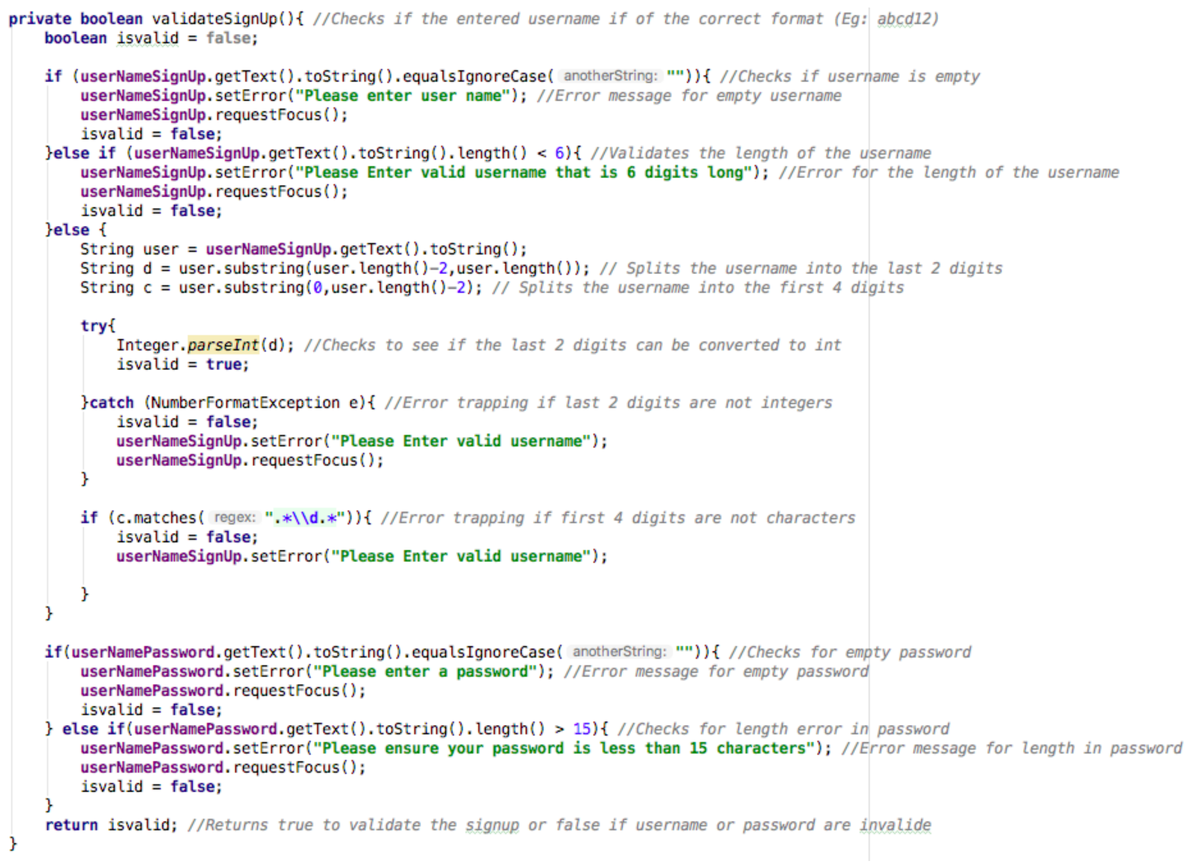
For sending emails, I used a combination of 3 libraries: "activation.jar", "additionnal.jar", and "mail.jar". These libraries can be found in the bibliography[3]. I also made 3 helper classes called "ByteArrayDataSource.java", "GMailSender.java", and "JSSEProvider.java" (Appendix E).

[1] "Retrofit." Square, square.github.io/retrofit/.
[2] tutorialspoint.com. "JSON Tutorial." Www.tutorialspoint.com, Tutorials Point, www.tutorialspoint.com/json/.
[3] "Google Code Archive - Long-Term Storage for Google Code Project Hosting." Google, Google, code.google.com/archive/p/javamail-android/downloads.

NON STANDARD ALGORITHMS EXPLAINED

VALIDATION OF SIGN UP (03/08/18)

COMPLEXITY OF SOLUTION



```java
private boolean validateSignUp(){ //Checks if the entered username if of the correct format (Eg: abcd12)
    boolean isvalid = false;

    if (userNameSignUp.getText().toString().equalsIgnoreCase( anotherString: "")){ //Checks if username is empty
        userNameSignUp.setError("Please enter user name"); //Error message for empty username
        userNameSignUp.requestFocus();
        isvalid = false;
    }else if (userNameSignUp.getText().toString().length() < 6){ //Validates the length of the username
        userNameSignUp.setError("Please Enter valid username that is 6 digits long"); //Error for the length of the username
        userNameSignUp.requestFocus();
        isvalid = false;
    }else {
        String user = userNameSignUp.getText().toString();
        String d = user.substring(user.length()-2,user.length()); // Splits the username into the last 2 digits
        String c = user.substring(0,user.length()-2); // Splits the username into the first 4 digits

        try{
            Integer.parseInt(d); //Checks to see if the last 2 digits can be converted to int
            isvalid = true;

        }catch (NumberFormatException e){ //Error trapping if last 2 digits are not integers
            isvalid = false;
            userNameSignUp.setError("Please Enter valid username");
            userNameSignUp.requestFocus();
        }

        if (c.matches( regex: ".*\\d.*")){ //Error trapping if first 4 digits are not characters
            isvalid = false;
            userNameSignUp.setError("Please Enter valid username");

        }
    }

    if(userNamePassword.getText().toString().equalsIgnoreCase( anotherString: "")){ //Checks for empty password
        userNamePassword.setError("Please enter a password"); //Error message for empty password
        userNamePassword.requestFocus();
        isvalid = false;
    } else if(userNamePassword.getText().toString().length() > 15){ //Checks for length error in password
        userNamePassword.setError("Please ensure your password is less than 15 characters"); //Error message for length in password
        userNamePassword.requestFocus();
        isvalid = false;
    }
    return isvalid; //Returns true to validate the signup or false if username or password are invalide
}
```

When the function above is called, the username and password are read and validated. I had to use **nested IF** statements for conditional responses and **parsed** the data between integer types.

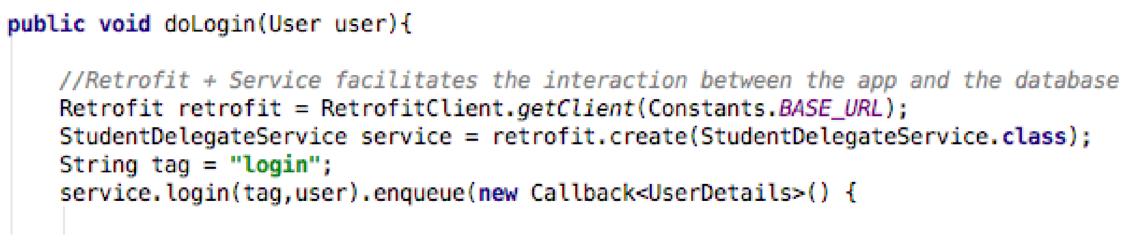
INGENUITY OF SOLUTION

I had to **think ahead** to catch possible errors when validating user inputs using **try catch statements**. As this function is generic (showing **abstract thinking**) to all username and password requirements, I could call the same function in other areas of my app- saving code.

LOG IN FUNCTION (06/08/18 – 10/08/18)

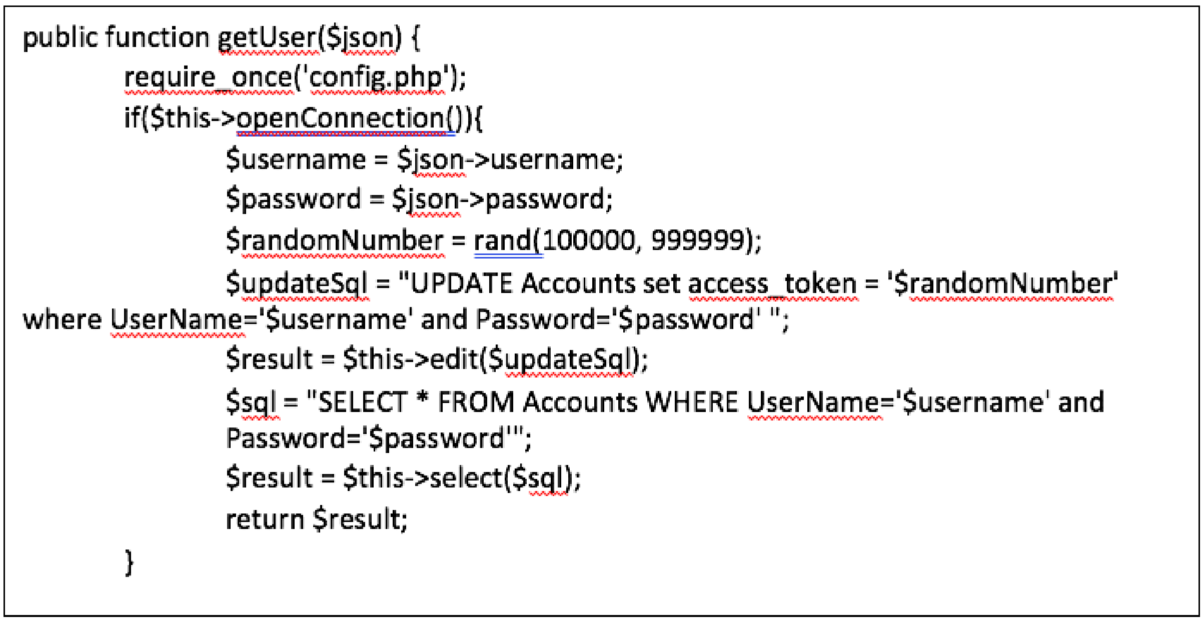
COMPLEXITY OF SOLUTION

The log in function required logical thinking as I had to manage reading databases, validation, and opening relevant android activities.



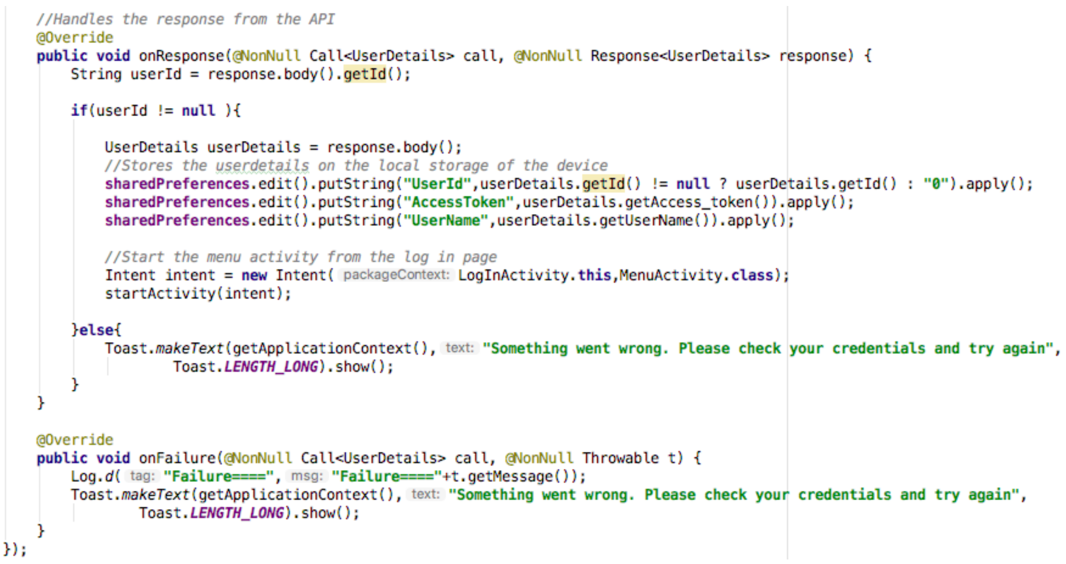
```
public void doLogin(User user){

    //Retrofit + Service facilitates the interaction between the app and the database
    Retrofit retrofit = RetrofitClient.getClient(Constants.BASE_URL);
    StudentDelegateService service = retrofit.create(StudentDelegateService.class);
    String tag = "login";
    service.login(tag,user).enqueue(new Callback<UserDetails>() {
```

The code above shows the **objects** of the Retrofit class and StudentDelegatesService **interface.** The .login() function works with the .enqueue() function to call the **PHP API** and pass in the relevant **arguments** as seen below:



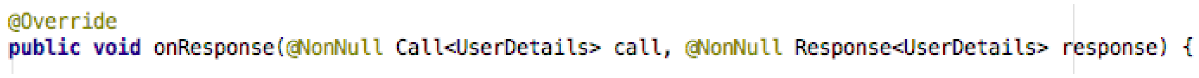
```
public function getUser($json) {
        require_once('config.php');
        if($this->openConnection()){
                $username = $json->username;
                $password = $json->password;
                $randomNumber = rand(100000, 999999);
                $updateSql = "UPDATE Accounts set access_token = '$randomNumber'
where UserName='$username' and Password='$password' ";
                $result = $this->edit($updateSql);
                $sql = "SELECT * FROM Accounts WHERE UserName='$username' and
                Password='$password'";
                $result = $this->select($sql);
                return $result;
        }
```

Using **decomposition**, the code first opens a connection to my database and then **searches** for the given username. It returns an object of type UserDetails, which is handled on the next page.



```
    //Handles the response from the API
    @Override
    public void onResponse(@NonNull Call<UserDetails> call, @NonNull Response<UserDetails> response) {
        String userId = response.body().getId();

        if(userId != null ){

            UserDetails userDetails = response.body();
            //Stores the userdetails on the local storage of the device
            sharedPreferences.edit().putString("UserId",userDetails.getId() != null ? userDetails.getId() : "0").apply();
            sharedPreferences.edit().putString("AccessToken",userDetails.getAccess_token()).apply();
            sharedPreferences.edit().putString("UserName",userDetails.getUserName()).apply();

            //Start the menu activity from the log in page
            Intent intent = new Intent( packageContext: LogInActivity.this,MenuActivity.class);
            startActivity(intent);

        }else{
            Toast.makeText(getApplicationContext(), text: "Something went wrong. Please check your credentials and try again",
                    Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onFailure(@NonNull Call<UserDetails> call, @NonNull Throwable t) {
        Log.d( tag: "Failure====", msg: "Failure===="+t.getMessage());
        Toast.makeText(getApplicationContext(), text: "Something went wrong. Please check your credentials and try again",
                Toast.LENGTH_LONG).show();
    }
});
```

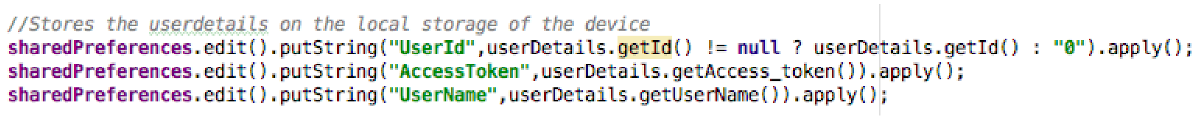
The onResponse() method handles the API response. I also overwrote the onFailure() method which handles the response in case any errors occur, showing **thinking ahead**.



```
@Override
public void onResponse(@NonNull Call<UserDetails> call, @NonNull Response<UserDetails> response) {
```

This can be considered as **polymorphism** as I am **overwriting** from the Retrofit class.

INGENUITY OF THE SOLUTION

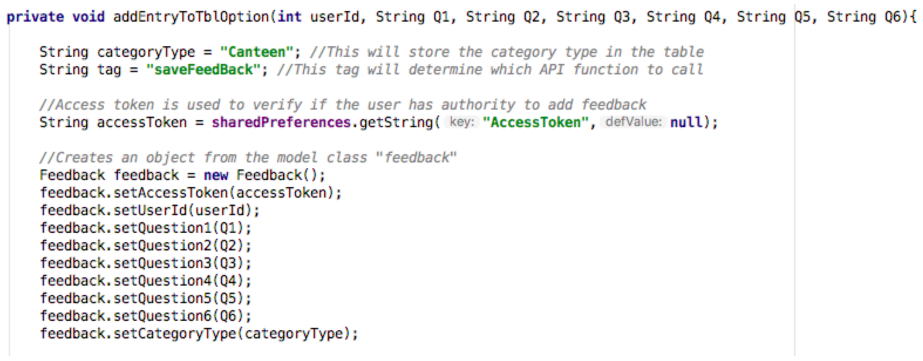


```
//Stores the userdetails on the local storage of the device
sharedPreferences.edit().putString("UserId",userDetails.getId() != null ? userDetails.getId() : "0").apply();
sharedPreferences.edit().putString("AccessToken",userDetails.getAccess_token()).apply();
sharedPreferences.edit().putString("UserName",userDetails.getUserName()).apply();
```

Storing the results of my **PHP API call** in the “sharedpreferences” of the user allowed me to save the user details directly on the device. Hence, later on, I could simply access the stored details from the phone, making this app a **semi distributed system**.

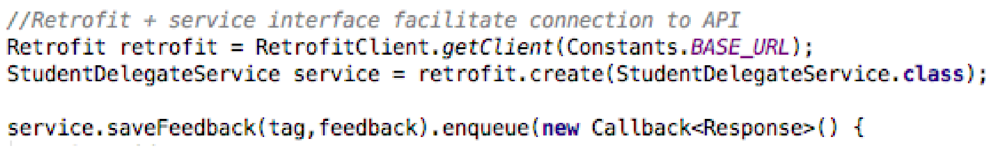
ADDING FEEDBACK TO DATABASE (11/08/18 – 16/08/18)

COMPLEXITY OF SOLUTION



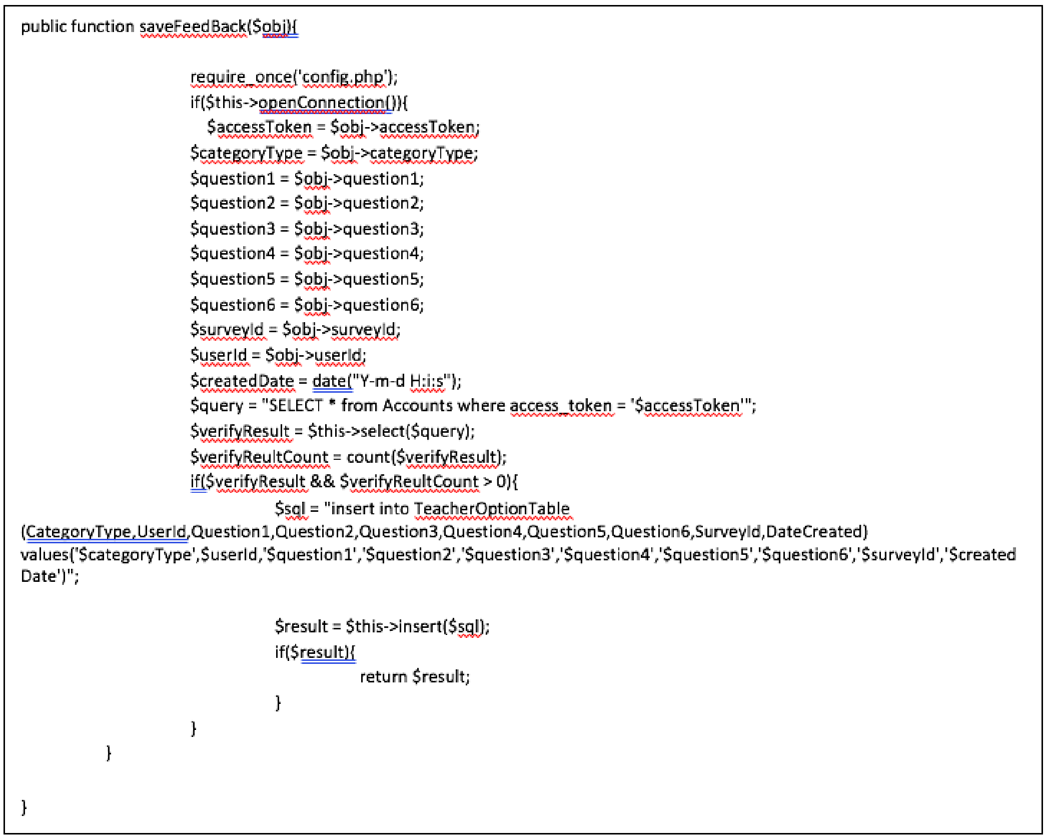
```java
private void addEntryToTblOption(int userId, String Q1, String Q2, String Q3, String Q4, String Q5, String Q6){

    String categoryType = "Canteen"; //This will store the category type in the table
    String tag = "saveFeedBack"; //This tag will determine which API function to call

    //Access token is used to verify if the user has authority to add feedback
    String accessToken = sharedPreferences.getString( key: "AccessToken", defValue: null);

    //Creates an object from the model class "feedback"
    Feedback feedback = new Feedback();
    feedback.setAccessToken(accessToken);
    feedback.setUserId(userId);
    feedback.setQuestion1(Q1);
    feedback.setQuestion2(Q2);
    feedback.setQuestion3(Q3);
    feedback.setQuestion4(Q4);
    feedback.setQuestion5(Q5);
    feedback.setQuestion6(Q6);
    feedback.setCategoryType(categoryType);
```

Thinking procedurally, I first stored my data on the database.



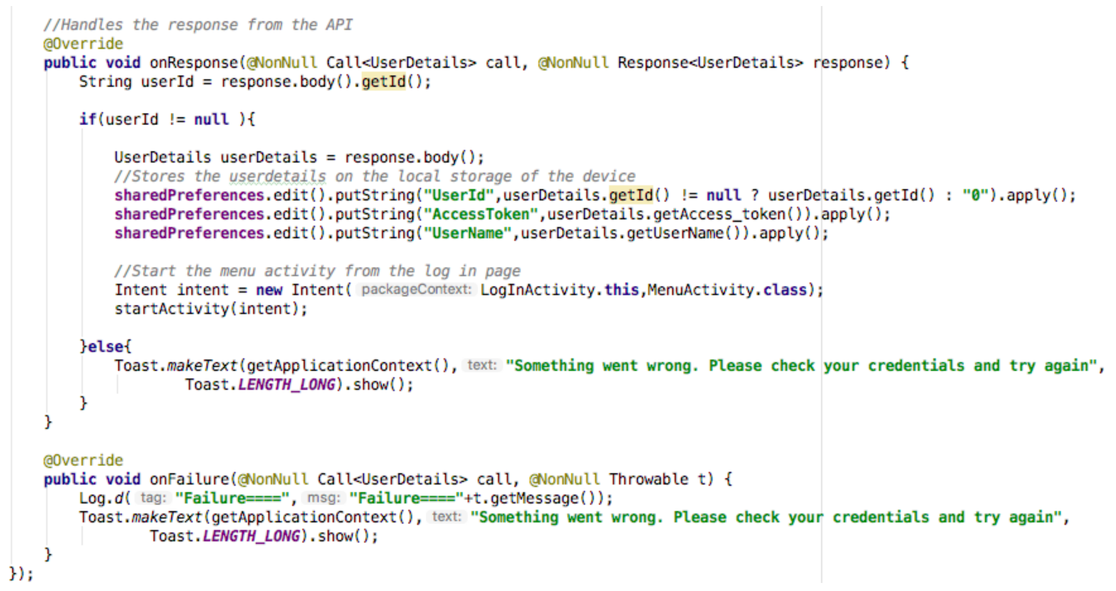
```java
//Retrofit + service interface facilitate connection to API
Retrofit retrofit = RetrofitClient.getClient(Constants.BASE_URL);
StudentDelegateService service = retrofit.create(StudentDelegateService.class);

service.saveFeedback(tag,feedback).enqueue(new Callback<Response>() {
```

The code above calls the **server side** PHP APIs.



```php
public function saveFeedBack($obj){

                        require_once('config.php');
                        if($this->openConnection()){
                          $accessToken = $obj->accessToken;
                        $categoryType = $obj->categoryType;
                        $question1 = $obj->question1;
                        $question2 = $obj->question2;
                        $question3 = $obj->question3;
                        $question4 = $obj->question4;
                        $question5 = $obj->question5;
                        $question6 = $obj->question6;
                        $surveyId = $obj->surveyId;
                        $userId = $obj->userId;
                        $createdDate = date("Y-m-d H:i:s");
                        $query = "SELECT * from Accounts where access_token = '$accessToken'";
                        $verifyResult = $this->select($query);
                        $verifyReultCount = count($verifyResult);
                        if($verifyResult && $verifyReultCount > 0){
                                    $sql = "insert into TeacherOptionTable
(CategoryType,UserId,Question1,Question2,Question3,Question4,Question5,Question6,SurveyId,DateCreated)
values('$categoryType',$userId,'$question1','$question2','$question3','$question4','$question5','$question6','$surveyId','$created
Date')";

                                    $result = $this->insert($sql);
                                    if($result){
                                                return $result;
                                    }
                        }
            }

}
```

The SQL commands **insert** the data and a response code is returned to the app.



```
    //Handles the response from the API
    @Override
    public void onResponse(@NonNull Call<UserDetails> call, @NonNull Response<UserDetails> response) {
        String userId = response.body().getId();

        if(userId != null ){

            UserDetails userDetails = response.body();
            //Stores the userdetails on the local storage of the device
            sharedPreferences.edit().putString("UserId",userDetails.getId() != null ? userDetails.getId() : "0").apply();
            sharedPreferences.edit().putString("AccessToken",userDetails.getAccess_token()).apply();
            sharedPreferences.edit().putString("UserName",userDetails.getUserName()).apply();

            //Start the menu activity from the log in page
            Intent intent = new Intent( packageContext: LogInActivity.this,MenuActivity.class);
            startActivity(intent);

        }else{
            Toast.makeText(getApplicationContext(), text: "Something went wrong. Please check your credentials and try again",
                    Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onFailure(@NonNull Call<UserDetails> call, @NonNull Throwable t) {
        Log.d( tag: "Failure====", msg: "Failure===="+t.getMessage());
        Toast.makeText(getApplicationContext(), text: "Something went wrong. Please check your credentials and try again",
                Toast.LENGTH_LONG).show();
    }
});
```

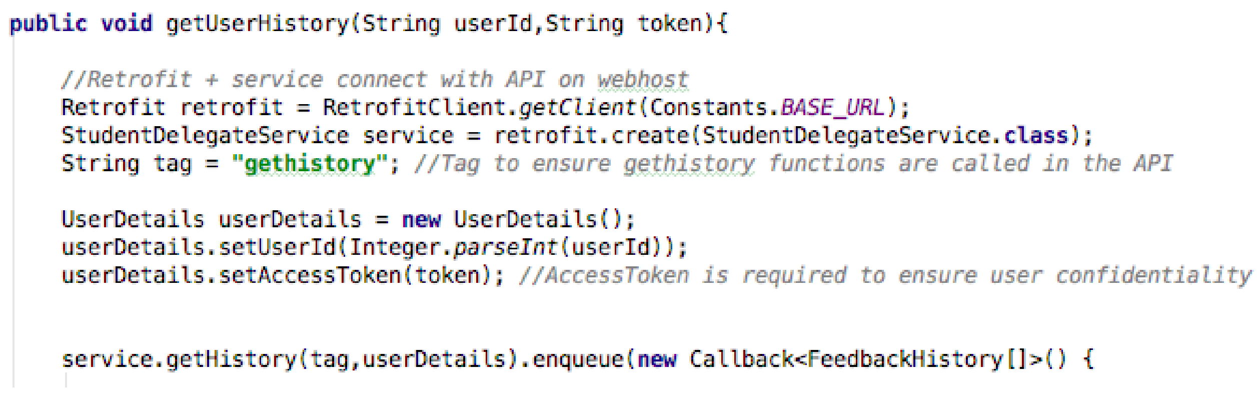
If the response “code” is 200, then a success message is printed. In the case of failure, an error message is given.

INGENUITY OF SOLUTION

Thinking logically, the use of the “feedback” class allowed me to use the same code for all survey pages as this class is generic to all solutions: evidence of **thinking abstractly**. I also had to **think procedurally** by using **decomposition** because the processes could be broken down into 3 steps: collect data, call the API, then handle the response.

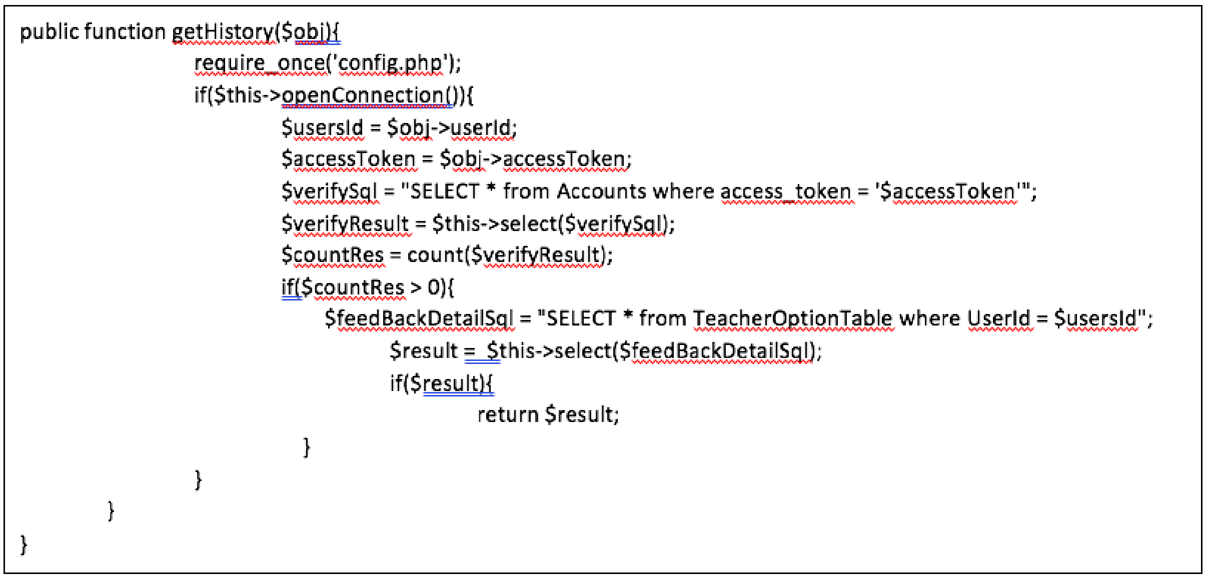
VIEW USER HISTORY (17/08/18 – 20/08/18)

COMPLEXITY OF SOLUTION



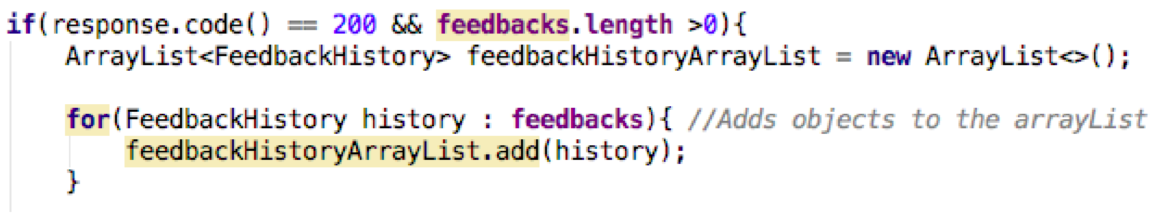
```java
public void getUserHistory(String userId,String token){

    //Retrofit + service connect with API on webhost
    Retrofit retrofit = RetrofitClient.getClient(Constants.BASE_URL);
    StudentDelegateService service = retrofit.create(StudentDelegateService.class);
    String tag = "gethistory"; //Tag to ensure gethistory functions are called in the API

    UserDetails userDetails = new UserDetails();
    userDetails.setUserId(Integer.parseInt(userId));
    userDetails.setAccessToken(token); //AccessToken is required to ensure user confidentiality


    service.getHistory(tag,userDetails).enqueue(new Callback<FeedbackHistory[]>() {
```

With the use of the **Retrofit library** and my interface, the code above connects to my PHP API and sends an object of type UserDetails.



```php
public function getHistory($obj){
                        require_once('config.php');
                        if($this->openConnection()){
                                    $usersId = $obj->userId;
                                    $accessToken = $obj->accessToken;
                                    $verifySql = "SELECT * from Accounts where access_token = '$accessToken'";
                                    $verifyResult = $this->select($verifySql);
                                    $countRes = count($verifyResult);
                                    if($countRes > 0){
                                          $feedBackDetailSql = "SELECT * from TeacherOptionTable where UserId = $usersId";
                                                $result = $this->select($feedBackDetailSql);
                                                if($result){
                                                            return $result;
                                      }
                        }
            }
}
```

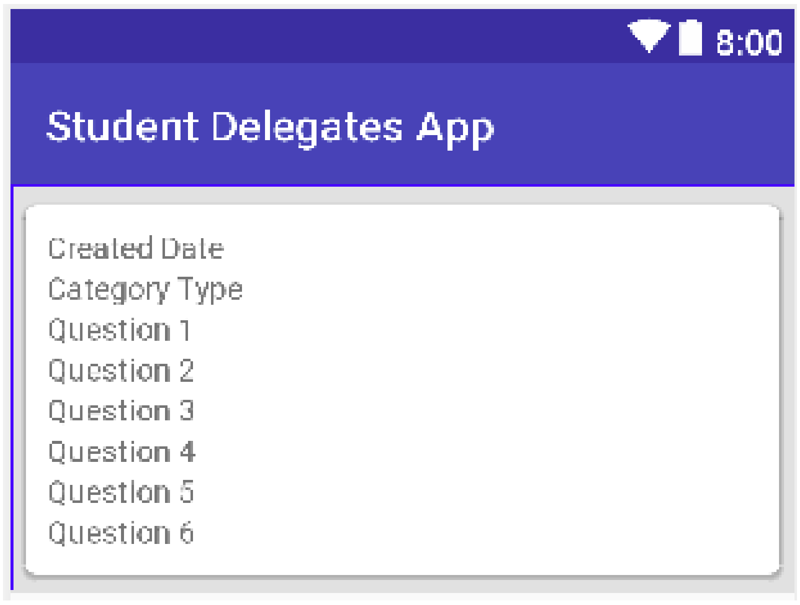
The PHP code above **searches** the database using the **primary key** UserID. A response is sent as an object which contains all the relevant data.



```java
if(response.code() == 200 && feedbacks.length >0){
    ArrayList<FeedbackHistory> feedbackHistoryArrayList = new ArrayList<>();

    for(FeedbackHistory history : feedbacks){ //Adds objects to the arrayList
        feedbackHistoryArrayList.add(history);
    }
```

After using the API call (Appendix C) for reading the database, I decided to store the response in an array list. Arraylists are crucial here as they are dynamic in size; unlike normal arrays which are of a finite size. As I do not know how much data is stored in my database, a **dynamic data structure** was needed. Array list have a time efficiency of O(N). Using an **iterative**

approach, whilst I originally designed the view history page as a table, **involving my client** (Appendix C, entry 33) revealed that he wanted a scrollable list instead. Hence, I had to design my own card layout to best match my client's needs, seen below.



Created Date
Category Type
Question 1
Question 2
Question 3
Question 4
Question 5
Question 6

I had to create my own History Adapter class as well which was implemented to view all my history data. This adapter class is shown below, highlighting **abstract thinking** because the code was reused for each dataset.

```
public class HistoryAdapter extends RecyclerView.Adapter<HistoryAdapter.MyViewHolder> {
```

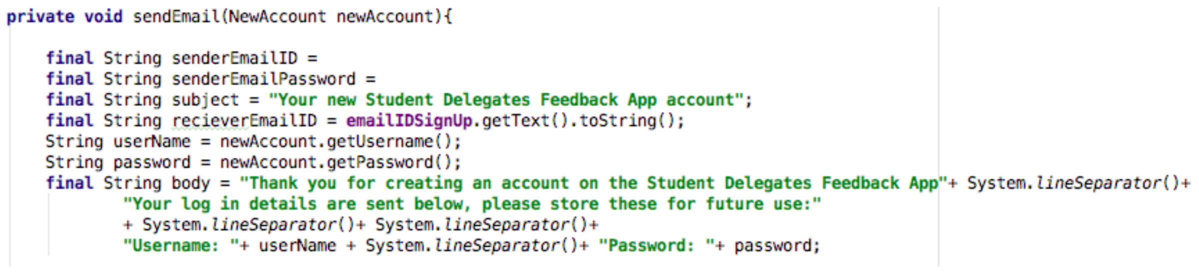
I had to make use of **inheritance** when **extending** the "ViewHolder" **super class**. This gave me additional features in the class such as access to the "OnBindViewHolder()" function.

INGENUITY OF SOLUTION

Thinking ahead was crucial here as I had to consider how I wanted to display my data before I read the data. I also had to **think abstractly** to ensure that I was writing efficient code. I solved this problem by creating a history card view using **XML** (which all my data will adapt) and by creating an **adapter class** which inflates each of my dataset. Doing this allowed me to use the same class for each record I read from the database instead of manually displaying each record.

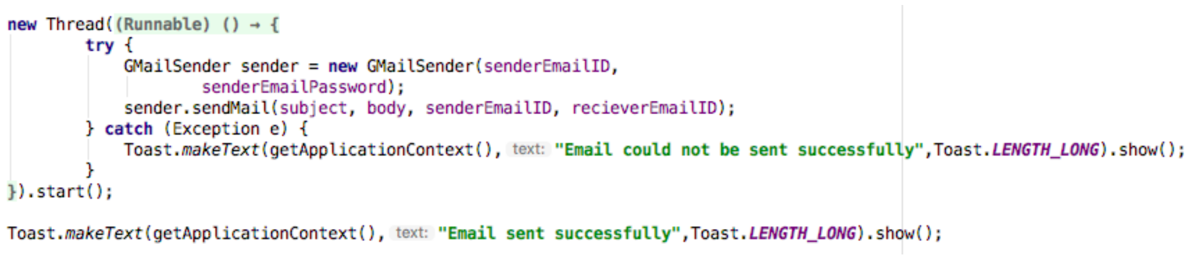
SEND EMAIL (21/08/18)

COMPLEXITY OF SOLUTION



```
private void sendEmail(NewAccount newAccount){

    final String senderEmailID =
    final String senderEmailPassword =
    final String subject = "Your new Student Delegates Feedback App account";
    final String recieverEmailID = emailIDSignUp.getText().toString();
    String userName = newAccount.getUsername();
    String password = newAccount.getPassword();
    final String body = "Thank you for creating an account on the Student Delegates Feedback App"+ System.lineSeparator()+
            "Your log in details are sent below, please store these for future use:"
            + System.lineSeparator()+ System.lineSeparator()+
            "Username: "+ userName + System.lineSeparator()+ "Password: "+ password;
```

Thinking procedurally, I first had to set up the contents of the email by receiving the user email ID. I then had to create the body of my email and then send the email (seen below). Note: contact details have been removed for privacy concerns.

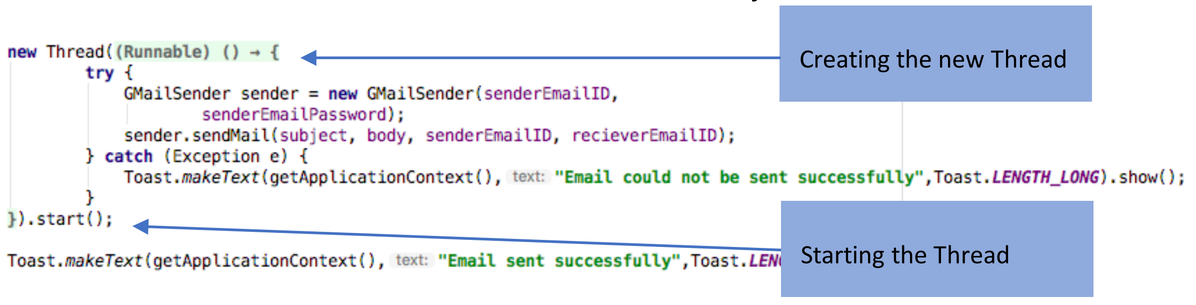


```
new Thread((Runnable) () → {
        try {
            GMailSender sender = new GMailSender(senderEmailID,
                    senderEmailPassword);
            sender.sendMail(subject, body, senderEmailID, recieverEmailID);
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), text: "Email could not be sent successfully",Toast.LENGTH_LONG).show();
        }
}).start();

Toast.makeText(getApplicationContext(), text: "Email sent successfully",Toast.LENGTH_LONG).show();
```

I had to use a **GMailSender helper class**, along with 2 other helper classes and 3 libraries (explained before) to send the email. **Thinking ahead**, I used **try catch statements** to deal with possible errors.

INGENUITY OF SOLUTION

I decided to send the email using a separate **thread** (showing **concurrent thinking**) to ensure that any errors in sending the email do no interrupt the flow of the application. I decided to use the **Gmail API** to reduce code and have better functionality.

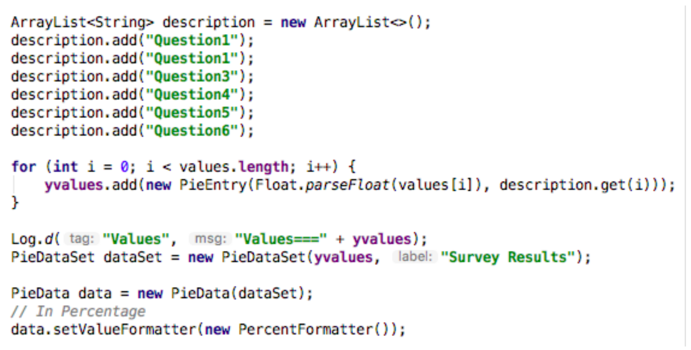


```
new Thread((Runnable) () → {
        try {
            GMailSender sender = new GMailSender(senderEmailID,
                    senderEmailPassword);
            sender.sendMail(subject, body, senderEmailID, recieverEmailID);
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), text: "Email could not be sent successfully",Toast.LENGTH_LONG).show();
        }
}).start();

Toast.makeText(getApplicationContext(), text: "Email sent successfully",Toast.LEN
```

GENERATE REPORTS (24/08/18 – 29/08/18)

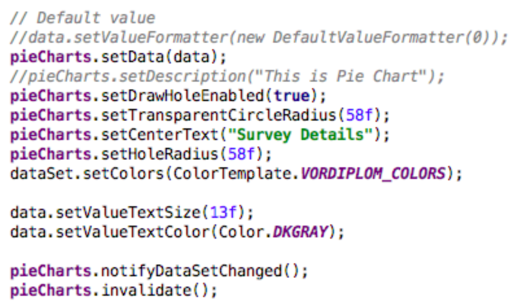
COMPLEXITY OF SOLUTION

VIEW PIE CHARTS



```
ArrayList<String> description = new ArrayList<>();
description.add("Question1");
description.add("Question1");
description.add("Question3");
description.add("Question4");
description.add("Question5");
description.add("Question6");

for (int i = 0; i < values.length; i++) {
    yvalues.add(new PieEntry(Float.parseFloat(values[i]), description.get(i)));
}

Log.d( tag: "Values",  msg: "Values===" + yvalues);
PieDataSet dataSet = new PieDataSet(yvalues,  label: "Survey Results");

PieData data = new PieData(dataSet);
// In Percentage
data.setValueFormatter(new PercentFormatter());
```

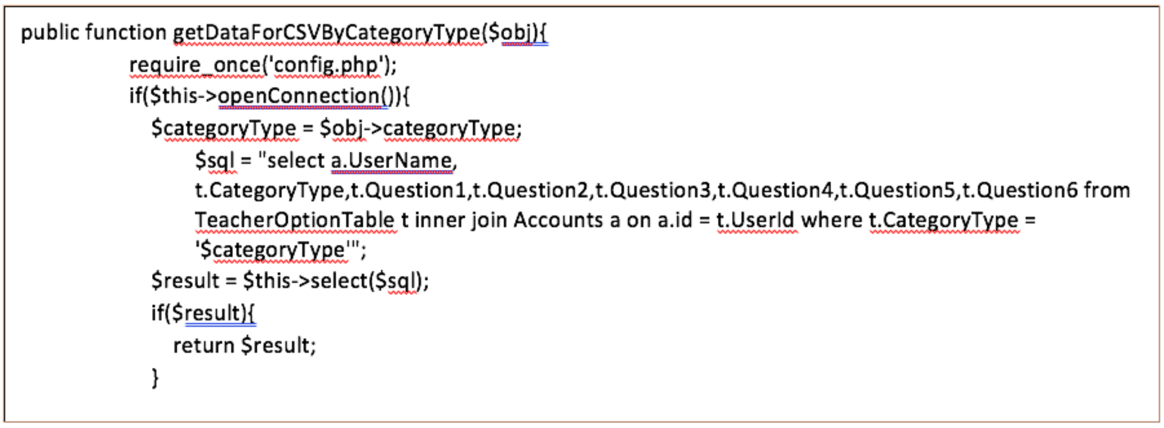
Creating an **object** of PieDataSet, I was able to pass in the description **ArrayList**.



```
// Default value
//data.setValueFormatter(new DefaultValueFormatter(0));
pieCharts.setData(data);
//pieCharts.setDescription("This is Pie Chart");
pieCharts.setDrawHoleEnabled(true);
pieCharts.setTransparentCircleRadius(58f);
pieCharts.setCenterText("Survey Details");
pieCharts.setHoleRadius(58f);
dataSet.setColors(ColorTemplate.VORDIPLOM_COLORS);

data.setValueTextSize(13f);
data.setValueTextColor(Color.DKGRAY);

pieCharts.notifyDataSetChanged();
pieCharts.invalidate();
```

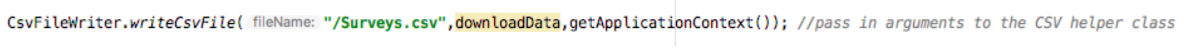
Using **predefined functions**, I was able to create and display the pie charts.

COMPILE REPORTS



```
public function getDataForCSVByCategoryType($obj){
            require_once('config.php');
            if($this->openConnection()){
              $categoryType = $obj->categoryType;
                  $sql = "select a.UserName,
                  t.CategoryType,t.Question1,t.Question2,t.Question3,t.Question4,t.Question5,t.Question6 from
                  TeacherOptionTable t inner join Accounts a on a.id = t.UserId where t.CategoryType =
                  '$categoryType'";
              $result = $this->select($sql);
              if($result){
                return $result;
              }
```

Using **SQL calls** and a PHP API, I was able to get the data for the **CSV files**.



```
CsvFileWriter.writeCsvFile( fileName: "/Surveys.csv",downloadData,getApplicationContext()); //pass in arguments to the CSV helper class
```

Using the **constructor** of the CSVFileWriter **helper class**, I was able to create the CSV file above.

BIBLIOGRAPHY

1. Vogel, Lars, et al. “Quick Links.” Vogella.com, 12 Sept. 2017, www.vogella.com/tutorials/Retrofit/article.html#exercise-using-retrofit-to-query-gerrit-in-java.

2. tutorialspoint.com. “JSON Tutorial.” Www.tutorialspoint.com, Tutorials Point, www.tutorialspoint.com/json/.

3. “Retrofit.” Square, square.github.io/retrofit/.

4. “Google Code Archive - Long-Term Storage for Google Code Project Hosting.” Google, Google, code.google.com/archive/p/javamail-android/downloads.

Words: 1003