Josh Westenheffer

CS 499

Module 5

Enhancement 3

Enhancement List:

- Functional Changes
  - User login gives an Id
  - This Id is linked to their specific calendar data
  - Data now stays saved after closing the app and will remain in the database for each user. This allows multiple users to be logged in at once.
- Code Changes
  - Added to MyDataBaseHelper.java
  - Small additions to DayPage.java | LoginActivity.java | DataDisplayActivity.java
  - Added logic to handle new event database

QUESTION 1:

The artifact is an Android Studio calendar that initially features a basic login screen, data display screen, and an SMS screen. It was created last August during my Mobile programming class that centered around Android Studio. I wanted to choose this artifact because in my mind it has everything, I need to show how I have improved as a coder. It has a login screen that I can improve with a database to show how I can improve security and user experience. It has a data display where I can use a database in tandem with hash maps to show how I can create efficient and unique solutions. Finally, the login and data from the calendar itself is a great way to showcase how I can use a database and create a solid user experience through screens.

QUESTION 2:

This artifact when I first chose it had a lot of flexibility and possibilities that aligned with the enhancement ideas and outcomes so I chose it. For this enhancement focusing on databases, the important part was making sure user information remained after closing the app and giving them all a specific userid. This way multiple people can login and have their data stored. Before doing this enhancement, I figured this would show how I can create things like security and an app for multiple people. After completing this milestone though I think it is a great way to show how I can handle multiple files of code and create solid interactivity between them. This enhancement I focused on creating that database and userId. The main ideas of it were the userId needing to be linked among multiple pages and the database. This caused a lot of issues with the continuity and making sure this Id is used for not just the login, but populating the apps main displays.

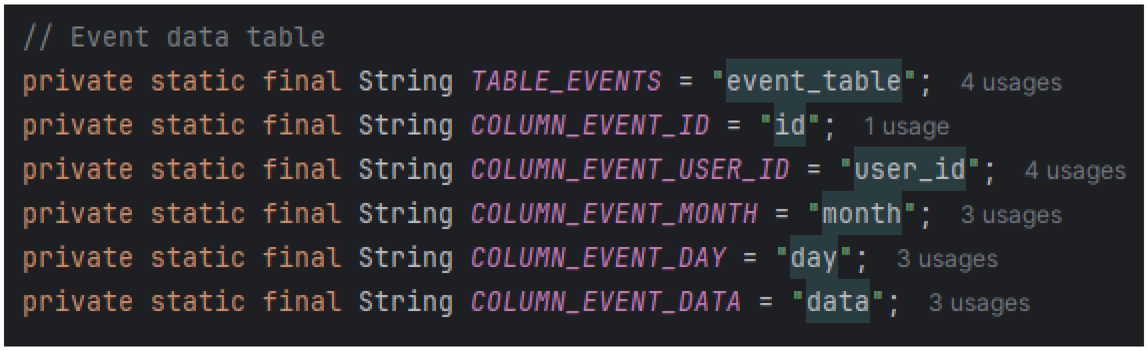
Question 3:

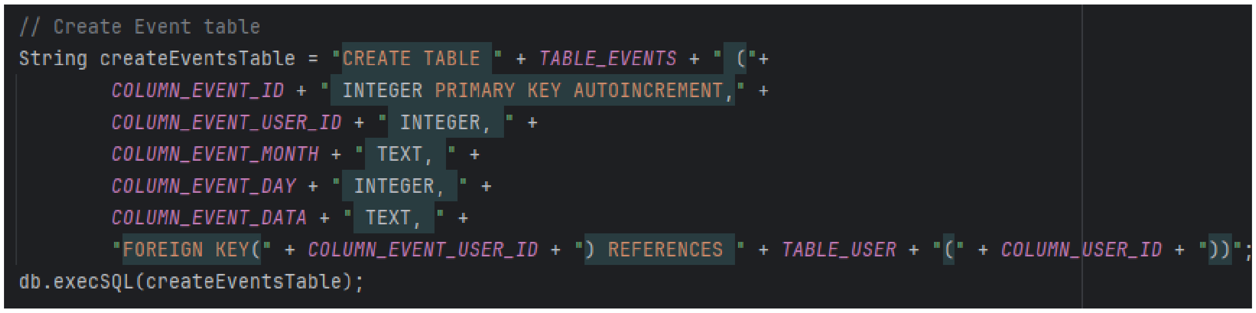
For this module I am focusing on outcomes 2 and 5. For this code review I am adding a video to help further explain the functionality and tried hard to create well-done comments. This is most important, because this code has become difficult to follow towards the end. For the 5th outcome I am focusing on the users Id to help security and how this will interact with the HashMap I have created to protect their data across multiple user logins or logging out and back in.

So, for this enhancement I don't have anything show before because there was no implementation. I'm not certain what I should put here, but since everything is new, I'm going to show the two main database retrieval/storage functions and the initialization of the new database table. Then I will talk a little bit about both my overall process. To start with, let's look at the events setup table in the database:

Below is a video as requested demonstrating the code.

Enhancement 3.mkv


```
// Event data table
private static final String TABLE_EVENTS = "event_table";  4 usages
private static final String COLUMN_EVENT_ID = "id";  1 usage
private static final String COLUMN_EVENT_USER_ID = "user_id";  4 usages
private static final String COLUMN_EVENT_MONTH = "month";  3 usages
private static final String COLUMN_EVENT_DAY = "day";  3 usages
private static final String COLUMN_EVENT_DATA = "data";  3 usages
```


```
// Create Event table
String createEventsTable = "CREATE TABLE " + TABLE_EVENTS + " ("+
        COLUMN_EVENT_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        COLUMN_EVENT_USER_ID + " INTEGER, " +
        COLUMN_EVENT_MONTH + " TEXT, " +
        COLUMN_EVENT_DAY + " INTEGER, " +
        COLUMN_EVENT_DATA + " TEXT, " +
        "FOREIGN KEY(" + COLUMN_EVENT_USER_ID + ") REFERENCES " + TABLE_USER + "(" + COLUMN_USER_ID + "))";
db.execSQL(createEventsTable);
```

Above is the code for setting the table up in the database. The key here is the variables each being represented by a column. Since I am using a nested HashMap setup to contain the users calendar information these variables are set for the keys and values. In specific here is the format and variable locations.

HashMap<String, HashMap<Integer, String>> ,

Outer key / 1st String – month

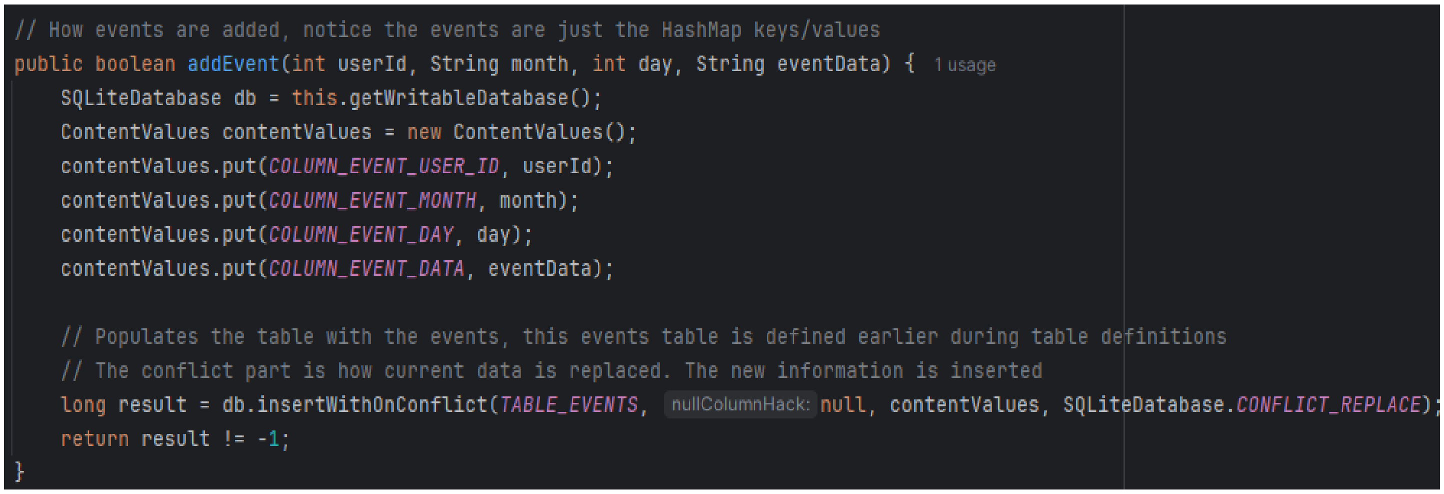
Inner Key / 1st Integer – day

Inner value / 2nd String – data (Event details)

The user_id is the link to the specific user and table. This differs from just id because just id is for each entry into the table.

Next, we have the addEvent function which is stored in the database java file. But gets called in the Day page java file. Specifically, this function gets called when the user would have selected a month, then a day, then entered their data in the textView. Once those have happened, if a user hits the save button on the screen, the addEvent function is called. Here is what that function looks like:

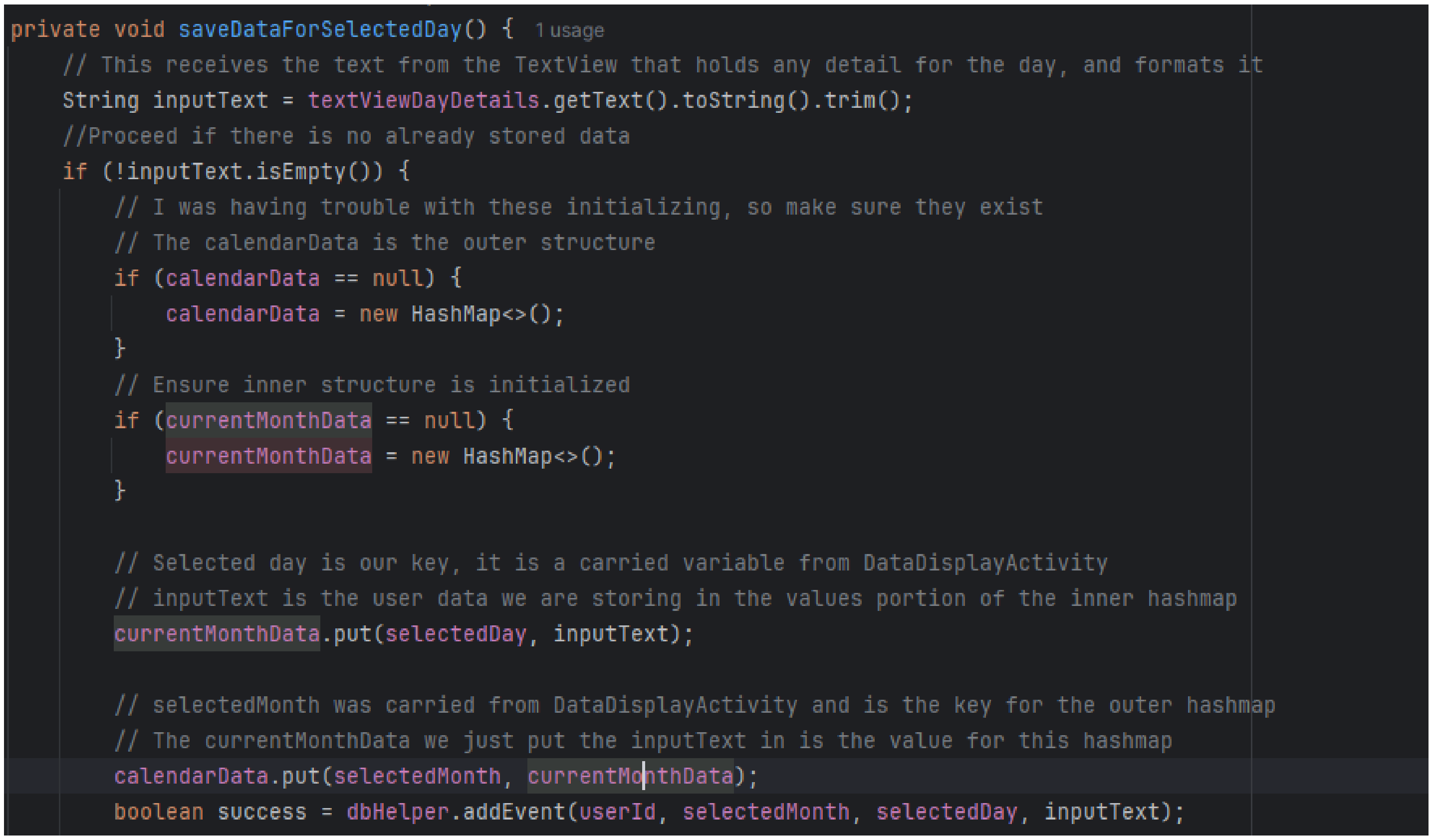
addEvent function



```java
// How events are added, notice the events are just the HashMap keys/values
public boolean addEvent(int userId, String month, int day, String eventData) {  1 usage
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(COLUMN_EVENT_USER_ID, userId);
    contentValues.put(COLUMN_EVENT_MONTH, month);
    contentValues.put(COLUMN_EVENT_DAY, day);
    contentValues.put(COLUMN_EVENT_DATA, eventData);

    // Populates the table with the events, this events table is defined earlier during table definitions
    // The conflict part is how current data is replaced. The new information is inserted
    long result = db.insertWithOnConflict(TABLE_EVENTS,  nullColumnHack: null, contentValues, SQLiteDatabase.CONFLICT_REPLACE);
    return result != -1;
}
```

addEvent call



```java
private void saveDataForSelectedDay() {  1 usage
    // This receives the text from the TextView that holds any detail for the day, and formats it
    String inputText = textViewDayDetails.getText().toString().trim();
    //Proceed if there is no already stored data
    if (!inputText.isEmpty()) {
        // I was having trouble with these initializing, so make sure they exist
        // The calendarData is the outer structure
        if (calendarData == null) {
            calendarData = new HashMap<>();
        }
        // Ensure inner structure is initialized
        if (currentMonthData == null) {
            currentMonthData = new HashMap<>();
        }

        // Selected day is our key, it is a carried variable from DataDisplayActivity
        // inputText is the user data we are storing in the values portion of the inner hashmap
        currentMonthData.put(selectedDay, inputText);

        // selectedMonth was carried from DataDisplayActivity and is the key for the outer hashmap
        // The currentMonthData we just put the inputText in is the value for this hashmap
        calendarData.put(selectedMonth, currentMonthData);
        boolean success = dbHelper.addEvent(userId, selectedMonth, selectedDay, inputText);
```

Okay, starting with the first picture, the function itself. The function needs the defined variables for the function, which the user has selected when they entered the (month, day, data). The userid is constant after login so all variables are present, then these are placed in their respective columns of the table. The insert line will cause any new data to replace any old data the table has. This way repeated days will be replaced.

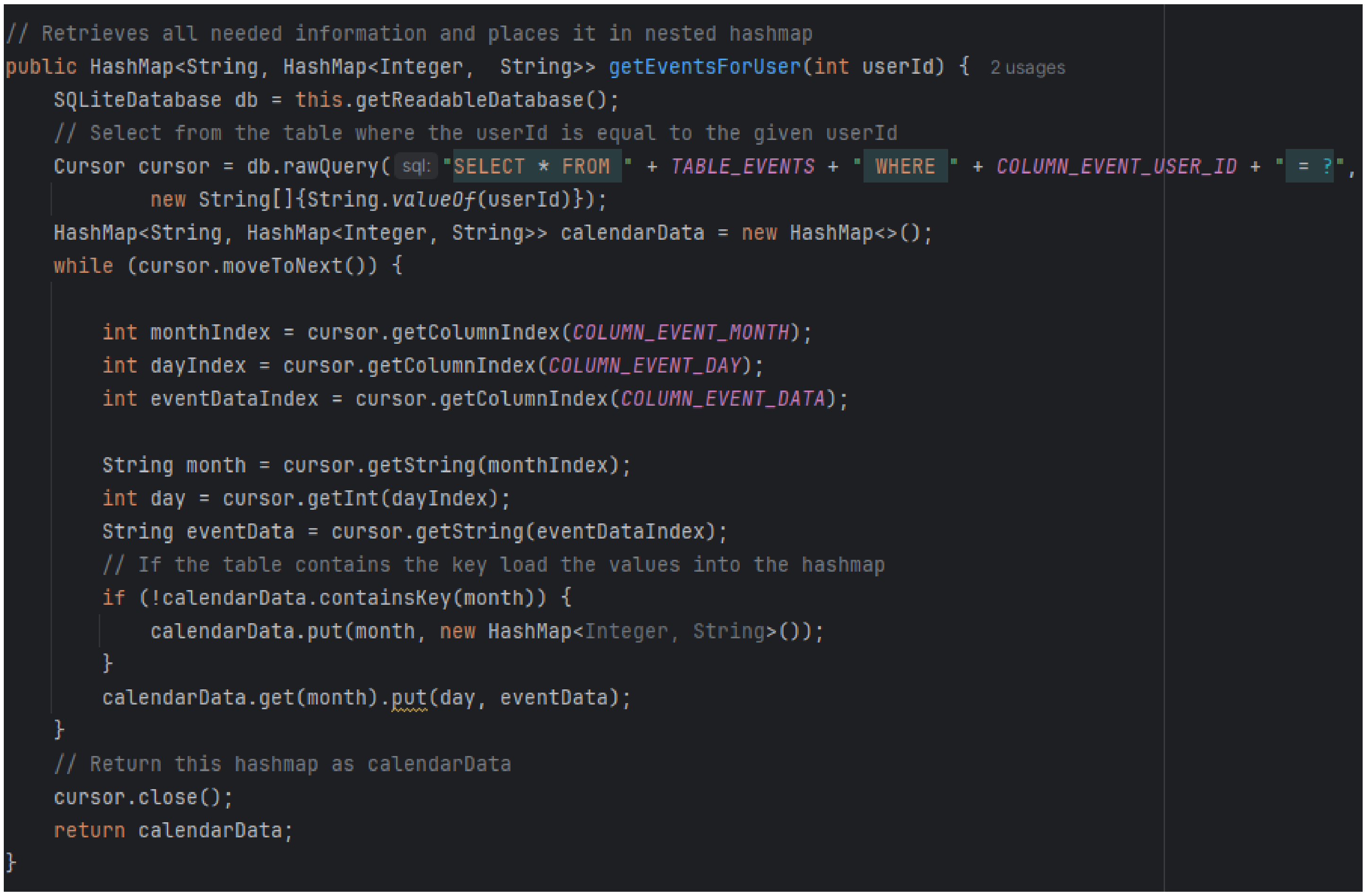
For the second picture, the addEvent call. The bottom line is what is new, there is code under this, but it is not relevant to the database. The bottom line says:

boolean success = dbHelper.addEvent(userId, selectedMonth, selectedDay, inputText);

What this line is doing is taking the addEvent function and filling the variables with what the user has done. These variables can be found in the nested HashMap which is seen above that bottom line. Basically, this file's original purpose was to allow the user to enter text and save it into the nested HashMap. This database enhances uses this HashMap to populate its table for better storage.

Okay, moving onto the second added function of my database which is getEventsForUser. This also has a call but is instead called to the main calendar page. Where the months and days are and where the day can be selected, rather than the user input screen. Here is what that function and call look like:

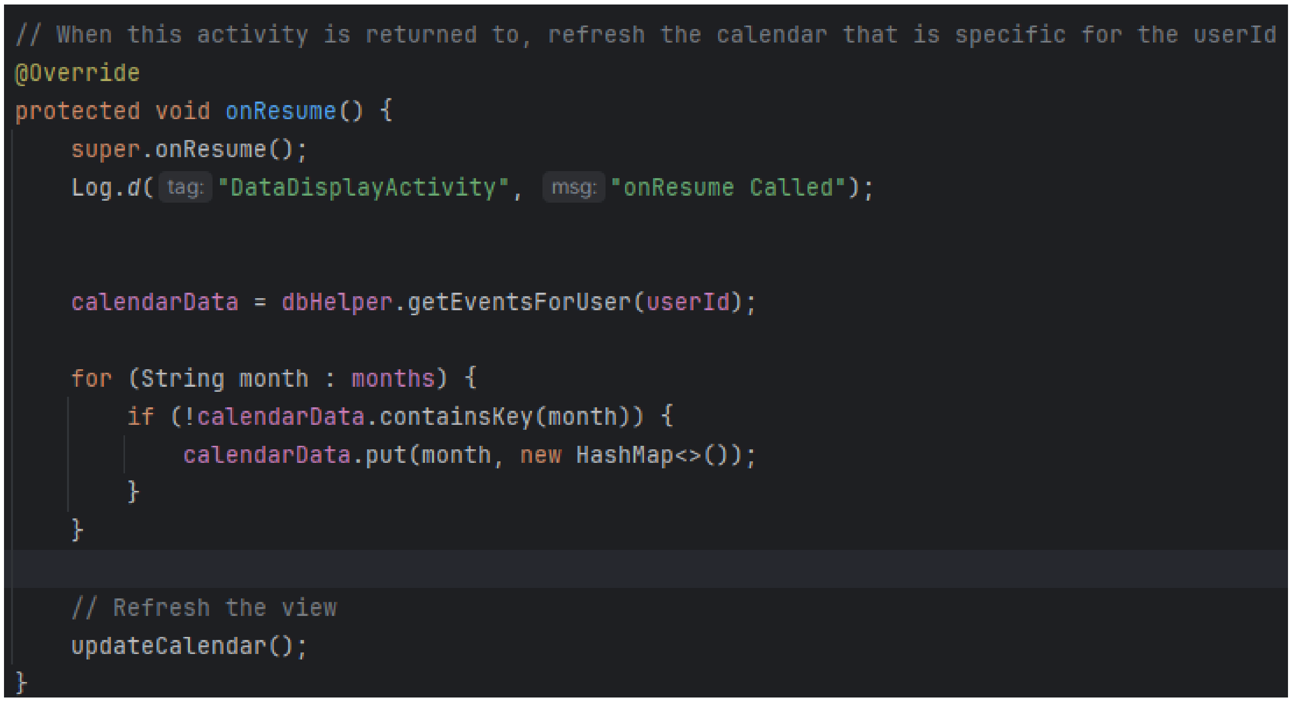
1st Picture getEventsForUser function | 2nd Picture getEventsForUser call



```java
// Retrieves all needed information and places it in nested hashmap
public HashMap<String, HashMap<Integer,  String>> getEventsForUser(int userId) {  2 usages
    SQLiteDatabase db = this.getReadableDatabase();
    // Select from the table where the userId is equal to the given userId
    Cursor cursor = db.rawQuery( sql: "SELECT * FROM " + TABLE_EVENTS + " WHERE " + COLUMN_EVENT_USER_ID + " = ?",
            new String[]{String.valueOf(userId)});
    HashMap<String, HashMap<Integer, String>> calendarData = new HashMap<>();
    while (cursor.moveToNext()) {

        int monthIndex = cursor.getColumnIndex(COLUMN_EVENT_MONTH);
        int dayIndex = cursor.getColumnIndex(COLUMN_EVENT_DAY);
        int eventDataIndex = cursor.getColumnIndex(COLUMN_EVENT_DATA);

        String month = cursor.getString(monthIndex);
        int day = cursor.getInt(dayIndex);
        String eventData = cursor.getString(eventDataIndex);
        // If the table contains the key load the values into the hashmap
        if (!calendarData.containsKey(month)) {
            calendarData.put(month, new HashMap<Integer, String>());
        }
        calendarData.get(month).put(day, eventData);
    }
    // Return this hashmap as calendarData
    cursor.close();
    return calendarData;
}
```



```java
// When this activity is returned to, refresh the calendar that is specific for the userId
@Override
protected void onResume() {
    super.onResume();
    Log.d( tag: "DataDisplayActivity",  msg: "onResume Called");


    calendarData = dbHelper.getEventsForUser(userId);

    for (String month : months) {
        if (!calendarData.containsKey(month)) {
            calendarData.put(month, new HashMap<>());
        }
    }

    // Refresh the view
    updateCalendar();
}
```

The first picture shows the function itself which looks slightly overwhelming. However, let's break it down the first step is to create a new nested HashMap structure, with the same breakdown as the earlier one. Then our function will accept a userId. The Cursor cursor = db.rawQuery line is referencing the TABLE_EVENTS table that contains the column for the userid's. The cursor looks for a userId that is the same as what was given for the function, so if the given userId was 2 it will look in the userId column for 2. Then we will initialize our HashMap with the name calendarData. The while loop will get the index for each column, which is basically getting the data from the table until it sees no more data. Then stores it in the proper variables. The nested if statement checks if there is already a key for the current month, if not it will create a nested HashMap for one. Then will fill the value as the nested HashMap. The statement after this if statement will get any month and place the specific day and eventData. This process will effectively retrieve a full HashMap for all twelve months and any user specific data. Finally, we return this new HashMap which is titled calendarData. This effectively creates our desired HashMap calendarData for a specific userId.

The second picture shows the call of this function. Pay attention to the line:

calendarData = dbHelper.getEventsForUser(userId);

This line is calling the function and sending a userId. The returned HashMap is used for other function within the code to populate the calendars contents.

As for how these help to meet my outcomes. Focusing first on outcome 2, this code has become quite complex so conveying it through this document and the video is part of what I believe to be a solid representation of how I can explain complex content. The comments in the code are helpful guides to the complex functions as well. I hope that my explanations also show how if needed, I could break these explanations down further.

As for the fifth outcome, the user id's and events database is a great way of keeping user privacy at the forefront of my development. There is a check for user accounts to ensure each user has a unique account and that their data is safe. The database also has a solid security to the user data and is a good way to show how I can create an app that is reusable among users while separating their data and interests. Overall, I think both of my proposed outcomes came together well during this enhancement.

Question 4:

This assignment was difficult for a few reasons, firstly the database syntax itself was confusing. When I looked online there are two main websites I used, geeksforgeeks and

AndroidStudioDevelopers. These both help with syntax, but there are many ways to incorporate these database into the code so getting the syntax correct was tricky. Once I realized it was much simpler than I was making it, the functions fell like dominos. As for the hardest part of this enhancement that would be the connection between the files. Maintaining a HashMap and userId across files was tricky both coding wise and mentally. I kept missing the userId and incorporating it incorrectly or not carrying it between files correctly. This in turn was a good lesson at paying attention to the small detail that makes the whole thing work because what good are functions without their variables.

This enhancement was fun to do in the long run and wasn't as stressful as I made it out to be to begin with. There was just something about dealing with a database I had little experience with in a IDE that I am unfamiliar with that seemed overwhelming.