Lab5: Persistent storage

(Due Monday, March 2, 2020 @ 3.25pm)

Introduction:

This lab will show you how to store and save information locally so the app can retrieve its state when it restarts. Depending upon the use case and query pattern, one might want to have different types of such storage and retrieval mechanisms. In this lab, we will be looking at 3 different ways to do this. They are:

- **Milestone 1: SharedPreferences** (ideal for small amounts of information, in a key-value format). This information is loaded into the memory when the app starts, and hence provides very fast access.
- **Milestone 2 - Databases:** For information which requires complex queries. e.g.,: *How many notes were created by user 'Jane' after February 2020?*
- **Milestone 3 - Local file storage:** For information which is large and would be expensive to store in a database. This milestone is optional. You **DON'T need to get this checked off** and **you will NOT be graded** on this third milestone.

We will be building a Notes app, where a user can login to the Notes app and create and store notes. You will save some basic information of users in SharedPreferences, store the list of notes in a Database and store the Database and the contents of the notes in a File.

Note:

- The snapshots provided in this lab manual are to help you build the app. They do not represent the complete app. You will find the comments in the code snippet helpful.
- Please read the instructions carefully to implement different components of the app. We have not provided code snippets for components which have been covered in the previous labs.

Github Setup:

To start off let us make a clone of the repository as we have done before:

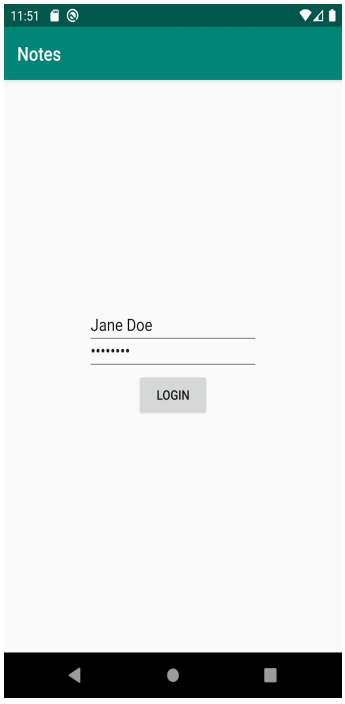
- Clone the Lab 5 repository by selecting File > New > Project from Version Control > Git
- Then click the following link and accept the invitation: https://classroom.github.com/a/YD7wnlMf
- You'll get a link in the form: https://github.com/CS-407-Spring2020/lab-5-yourgithubnamehere

Milestone1: Building the login screens

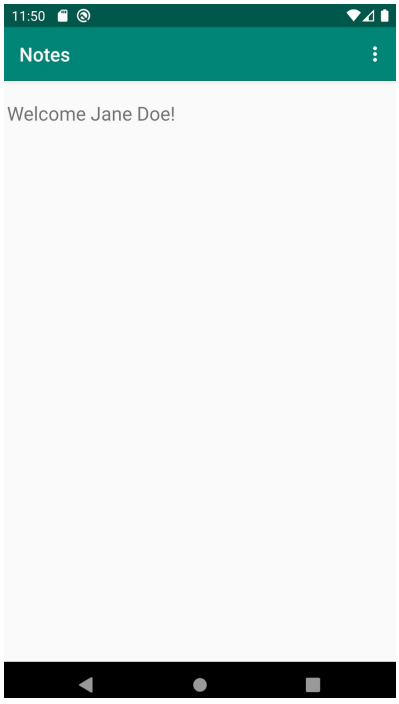
Here, we will be building the first part of the Notes app, that is, user login. We will build two screens:

- First Screen allows the user to log in
- In the Second screen, there is a welcome text and the notes created by the logged in user will be displayed.

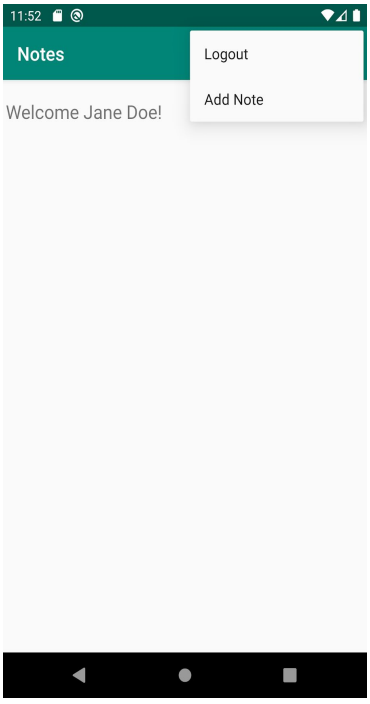
Here are some snapshots showing how the screens look:



S1: Login



S2(a): Displays list of notes (initially empty)



S2(b): Add note, Logout Menu

Building the first screen:

1. Add two EditText views, one for username, the other for password and add a button for the user to login.
2. **You are NOT required to do the authentication as part of this assignment. In other words, a user should be able to login using ANY password.** (Passwords are not saved in plain english anywhere. This login screen is for assignment purpose only)
3. Upon login, a screen like figure S2(a) should open up. So, define a second activity and add code to switch from first to second screen when the user clicks the login button. *(Recall the idea of using Intents to switch between activities.)*

   One way to set this up:

   a. *After you have set up the Login button in Step-1,in the MainActivity class, you will need to define a function say, onButtonClick() which switches to the second screen when the button is pressed.*
   b. *Within the onButtonClick() function, recall the idea of using intents and write code to switch to the second screen.*

Building the second screen:

1. Add a TextView at the top, this will be used to display a welcome text.
2. Add a MenuMain view on the right hand top side of the screen as shown in S2(a) (*Refer to the previous labs for a refresher on adding Menus*.)
2. In the MenuMain view, add two menu options and name them as "Add Note" and "Logout".
3. Clicking the Logout menu item should take you back to Screen 1. (*Hint: You will have to add this functionality in the "onOptionsItemSelected" function of the Menu you created in the previous step. Use intents again to set up this functionality.*).
4. Leave the AddNote button non-functional. We will come back to it in the next milestone.

Adding persistence:

Now that the basic app is ready, you need to add persistence at two points in this app for this milestone using SharedPreferences. We will be using SharedPreferences for the following tasks:

1. If a user closes the app without logging out, the welcome screen (Screen S2(a)) for the logged in user should show up as the first activity and not the login screen (Screen S1). For this, the app needs to REMEMBER the last user that logged in.
2. If the user had logged out before closing the app, the login screen should appear. For this to work, when the user presses the “logout” menu item, you will need to clear out the relevant information from the SharedPreferences and then go back to the login screen.

To help you add these functionalities, we have provided some helper codes in the “Using SharedPreferences” section.

Using SharedPreferences:

1. **Getting SharedPreference instance:**

   First, you need to create/get a shared preferences object - you can do that using getSharedPreferences() call. We do not need to worry if something has been written to it previously or not, the SharedPreference API encapsulates that information from us. You can have a singleton instance of it in code, or you can use the getSharedPreferences method as many times in the code to get new references. **All references will point to the same object, meaning if you write to SharedPreferences in one class, it’s state will change for the second class accessing it.** Here is an example:

```
SharedPreferences sharedPreferences =
getSharedPreferences("<package_name>", Context.MODE_PRIVATE);
```

- For the first parameter you should use a name that's uniquely identifiable to your application, generally it is the *package name of your Activity file*. Look at the package statement at the top of your Activity file if you have trouble finding it.
- Second parameter should be set to MODE_PRIVATE to open the shared preference file in a private mode.

2. **Editing SharedPreferences:**

   Any changes to the SharedPreferences have to be made using the edit() method. We associate the key with a value we wish to store. For the notes app, we need to store the username entered at the time of login. We use a key “username” (you can name the key as you wish) with a value extracted from the editTextView containing username. That is, After the user enters the username at the login screen, extract it and store it in

sharedPreferences. We follow this with an apply() method to complete editing the sharedPreferences. Here is an example:

```
sharedPreferences.edit().putString(“username”, userEditString).apply()
```

The *onButtonClick*() method for login button (shown in Screen-S1) should contain steps shown in the code snippet below:

```
public void onButtonClick(View view) {
    //1. Get username and password via EditText view.
    //2. Add username to SharedPreferences object.
    SharedPreferences sharedPreferences = getSharedPreferences( name: "com.example.lab5ms1", Context.MODE_PRIVATE);
    sharedPreferences.edit().putString("username","<login name you get from edit Text>").apply();
    //3. start second activity.
```

3. **Getting stored data from Shared Preferences:**

Use an instance of SharedPreferences to fetch the data stored in it. In the snippet below, we use the key “username” to obtain its corresponding value. The second argument specifies the default value to return if this key is not found. You can keep this as an empty string.

```
String username = sharedPreferences.getString(“username”, "");
```

Using the ideas shown till now, the OnCreate method of your app for Screen-1, should look something like this:

- In the Screen-1, you check if the username exists in the SharedPreferences object, (which means the user had logged in previously and did not logout) then you start the second activity.
- If the username does not exist in the Sharedpreferences object, the user has to login, so start the first activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String usernameKey = "username";

    SharedPreferences sharedPreferences = getSharedPreferences( name: "c.sakshi.lab5", Context.MODE_PRIVATE);

    if (!sharedPreferences.getString(usernameKey,  defValue: "").equals("")) {
        // "username" key exists in SharedPreferences object which means that a user was logged in before the app close.
        // Get the name of that user from SharedPreferences using sharedPreferences.getString(usernameKey, "").
        // Use Intent to start the second activity welcoming the user.
    } else {
        // SharedPreferences object has no username key set.
        // Start screen 1, that is the main activity.
        setContentView(R.layout.activity_main);
    }

}
```

- Along with this, After starting the second activity, remember that the Screen-2 should display the Welcome Message. You can use the SharedPreferences' .getString() method, in the second activity (Screen-2), to extract the username and display the Welcome message.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);

    // 1. Display welcome message. Fetch username from SharedPreferences.
```

4. **Deleting SharedPreferences keys**

   While logging out you should make sure you remove any data kept in the instance for that particular user.

```
sharedPreferences.edit().remove("username").apply();
```

   This is how you'll implement functionality of the Logout menu option in the Second Activity, which is seen in Screen-S2(b).

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    super.onOptionsItemSelected(item);
    if (item.getItemId() == R.id.logOut) {
        //Erase username from shared preferences.
        Intent intent = new Intent( packageContext: this, MainActivity.class);
        SharedPreferences sharedPreferences = getSharedPreferences( name: "c.sakshi.lab5", Context.MODE_PRIVATE);
        sharedPreferences.edit().remove(MainActivity.usernameKey).apply();
        startActivity(intent);
        return true;
    }
```

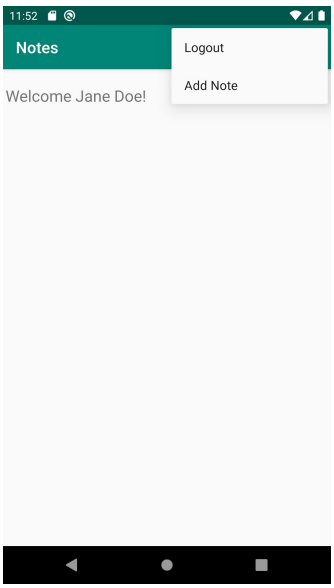
   Great work on getting so many different components to work together! Don't forget to show your instructor your work in this milestone.

Deliverables to be shown to the instructors for Milestone1:

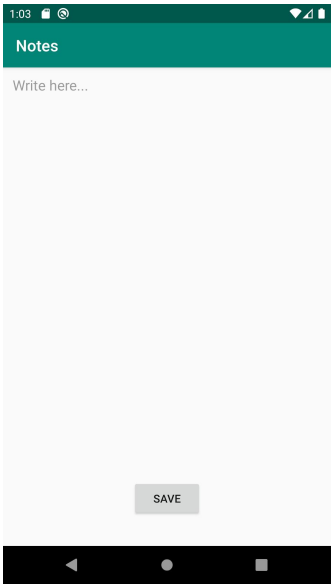
1) Functional login screen.
2) Second screen with options to add notes, logout.
3) If you close the app without logging out, the welcome screen for the logged in user should show up as the first screen.
4) If you close the app after clicking logout, the login screen should be the first to show up.

Milestone 2: Storing the list of notes for all users in a database

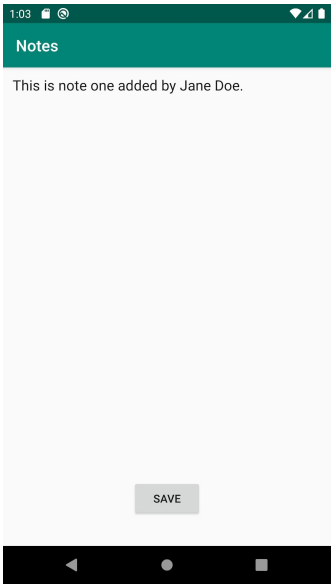
Recall that we wish that the second activity of this app (screen that appears after login) should display all the notes added by that user.



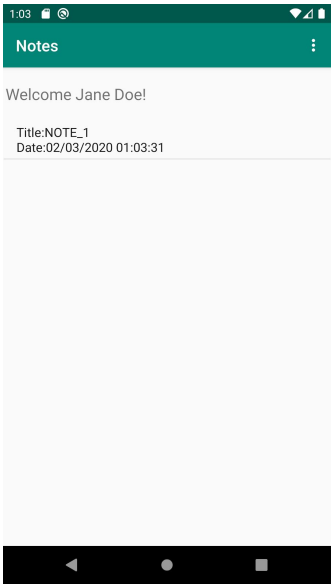
S2(b): Add note, Logout Menu



S3(a): Screen-3 Write note here



S3(b): Screen 3 Added Note



S2(c): Back to Screen 2 after adding the note

Adding components to the second and third screen:

To achieve screens shown above, let's add the following components to our app.

- Add a *ListView* to the second activity (used for Screen-2) using the layout xml file.
- Add another activity, say "**ThirdActivity**" for the screen-3 with a giant Edit text box and a save button. This activity is reachable via two options:
    - First, when the user clicks on the "Add Note" menu button. The "onOptionsItemSelected" method for "Add Note" menu item opens this activity to add a new note (figure S3(a)).
    - Second, when the user clicks on an existing note seen in the ListView of Screen-2, then Screen-3 loads up the content of the note from SQLite and displays it in its EditText view.
- Add an empty onClick method for the "Save" button in Screen-3 above. Use the information in the next section to implement it.

Persisting data using SQLite database:

- Add the following "Note" class to your project.
  This is a standalone class which needs to be added to your project. Recall the idea of creating a new Java class as we did in Lab-3.

```
public class Note {

    private String date;
    private String username;
    private String title;
    private String content;

    public Note(String date, String username, String title, String content) {
        this.date = date;
        this.username = username;
        this.title = title;
        this.content = content;
    }

    public String getDate() { return date; }

    public String getUsername() { return username; }

    public String getTitle() { return title; }

    public String getContent() { return content; }

}
```

Building Second screen

- **Getting SQLiteDatabase instance:**

  Android comes bundled with a database class SQLiteDatabase which provides abstractions on top of an SQL database. We can use this class to create and use databases by firing up SQL commands. The code to get a handler for SQLite database named “notes” in Private mode is shown below:

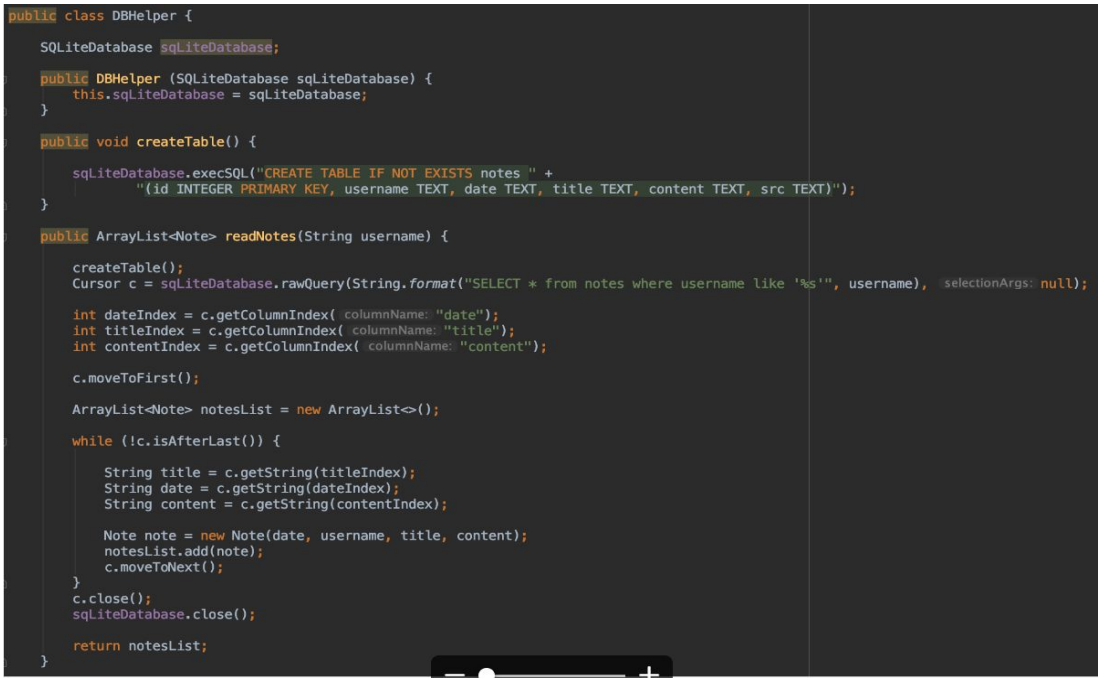
```
Context context = getApplicationContext();
SQLiteDatabase sqLiteDatabase = context.openOrCreateDatabase("notes",
Context.MODE_PRIVATE,null);
```

You will be using this code in a few places, for now just make a note of its usage.

- **DBHelper Class:**

  This is a standalone class which needs to be added to your project. Recall the idea of creating a new Java class as we did in Lab-3.

  This class contains all the helper functions which interact with the SQLiteDatabase object. These include createTable, readNotes, saveNotes, updateNotes. Add the following methods to the DBHelper class. We use SQLiteDatabase class’ `execSQL()` API to run a sql query on the SQLite database.



```
public class DBHelper {

    SQLiteDatabase sqLiteDatabase;

    public DBHelper (SQLiteDatabase sqLiteDatabase) {
        this.sqLiteDatabase = sqLiteDatabase;
    }

    public void createTable() {

        sqLiteDatabase.execSQL("CREATE TABLE IF NOT EXISTS notes " +
                "(id INTEGER PRIMARY KEY, username TEXT, date TEXT, title TEXT, content TEXT, src TEXT)");
    }

    public ArrayList<Note> readNotes(String username) {

        createTable();
        Cursor c = sqLiteDatabase.rawQuery(String.format("SELECT * from notes where username like '%s'", username),  selectionArgs: null);

        int dateIndex = c.getColumnIndex( columnName: "date");
        int titleIndex = c.getColumnIndex( columnName: "title");
        int contentIndex = c.getColumnIndex( columnName: "content");

        c.moveToFirst();

        ArrayList<Note> notesList = new ArrayList<>();

        while (!c.isAfterLast()) {

            String title = c.getString(titleIndex);
            String date = c.getString(dateIndex);
            String content = c.getString(contentIndex);

            Note note = new Note(date, username, title, content);
            notesList.add(note);
            c.moveToNext();
        }
        c.close();
        sqLiteDatabase.close();

        return notesList;
    }
```

```
public void saveNotes(String username, String title, String content, String date) {
    createTable();
    sqLiteDatabase.execSQL(String.format("INSERT INTO notes (username, date, title, content) VALUES ('%s', '%s', '%s', '%s')",
            username, date, title, content));
}

public void updateNote(String title, String date, String content, String username) {
    createTable();
    sqLiteDatabase.execSQL(String.format("UPDATE notes set content = '%s', date = '%s' where title = '%s' and username = '%s'",
            content, date, title, username));
}
```

- **Retrieving information from SQLiteDatabase:**

  When users log into their account, they should see a list of all notes created by them in Screen-2. These can be retrieved from the "notes" database using `rawQuery()` API. Specifically, look at how we are using the `Cursor` class to iterate through the query results. Make sure to `close()` the cursor instance after use.

  - Now that you are ready to read data from SQLite Database, we'll add listing notes functionality to our second screen.
  - Create the following class variable in the activity class of the second screen.

```
public static ArrayList<Note> notes = new ArrayList<>();
```

  - In order to display the notes, The **onCreate()** method of second activity looks like this:
    - Display the notes the user created using ArrayAdapter.
    - If the user clicks on any note in the ListView, we start the third activity and pass the note-ID to the third activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);

    // 1. Display welcome message. Fetch username from SharedPreferences.
    // 2. Get SQLiteDatabase instance.
    // 3. Initiate the "notes" class variable using readNotes method implemented in DBHelper class. Use the username you
    //    got from SharedPreferences as a parameter to readNotes method.
    // 4. Create an ArrayList<String> object by iterating over notes object.

    ArrayList<String> displayNotes = new ArrayList<>();
    for (Note note : notes) {
        displayNotes.add(String.format("Title:%s\nDate:%s", note.getTitle(), note.getDate()));
    }


    // 5. Use ListView view to display notes on screen.
    ArrayAdapter adapter =  new ArrayAdapter( context: this, android.R.layout.simple_list_item_1, displayNotes);
    ListView listView = (ListView) findViewById(R.id.notesListView);
    listView.setAdapter(adapter);


    // 6. Add onItemClickListener for ListView item, a note in our case.
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            // Initialise intent to take user to third activity (NoteActivity in this case).
            Intent intent = new Intent(getApplicationContext(), NoteActivity.class);
            // Add the position of the item that was clicked on as "noteid".
            intent.putExtra( name: "noteid", position);
            startActivity(intent);
        }
    });
}
```

- The “**onOptionsItemSelected**” method for AddNote simply takes the user to ThirdActivity.

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    super.onOptionsItemSelected(item);

   if (item.getItemId() == R.id.addNote) {
        //Go to activity 3.
        return true;
    }
```

Building Third screen

- Add a class variable called “noteid” in the third activity. We will update it later.
- Remember that we will arrive at Screen-3 in two ways:
    - When the user clicks “Add Note” in the menu button of Screen-2
    - User clicks a Note on Screen-2 and then the Note opens up in Screen-3.
- We need to reflect these operations in the **onCreate** method in our third activity. *To distinguish between these two scenarios we use the “noteid” we had added in the Second screen in the Intent.*
    - If no value is present for “noteid”, the user has clicked AddNote.
    - If the user clicked on a certain note, we need to get the contents of the note and display them on our EditText view using the “setText” API of EditText.

```
int noteid = -1;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_note);

    //1. Get EditText view.
    //2. Get Intent.
    //3. Get the value of integer "noteid" from intent.
    //4. Initialise class variable "noteid" with the value from intent.

    if (noteid != -1) {
        // Display content of note by retrieving "notes" ArrayList in SecondActivity.
        Note note = SecondActivity.notes.get(noteid);
        String noteContent = note.getContent();
        // Use editText.setText() to display the contents of this note on screen.
    }

}
```

- The **OnClick()** method of Save button in screen-3 should get the note content entered by the user, save all the relevant details of the note to the SQLite

Database instance and return to Screen-2. The method should look something like the following:

```
public void saveMethod(View view) {
    // 1. Get editText view and the content that user entered.
    // 2. Initialise SQLiteDatabase instance.
    // 3. Initialise DBHelper class.
    // 4. Set username in the following variable by fetching it from SharedPreferences.
    String username = "<get from shared preferences>";
    // 5. Save information to database.
    String title;
    DateFormat dateFormat = new SimpleDateFormat( pattern: "MM/dd/yyyy HH:mm:ss");
    String date = dateFormat.format(new Date());

    if (noteid == -1) { //Add note.
        title = "NOTE_" + (SecondActivity.notes.size() + 1);
        dbHelper.saveNotes(username, title, content, date);
    } else { //Update note.
        title = "NOTE_" + (noteid + 1);
        dbHelper.updateNote(title, date, content, username);
    }

    // 6. Go to second activity using intents.

}
```

(Remember: While initialising the DBHelper class, you first create DBHelper object and then instantiate it using the sqLite Database instance.)

Don’t forget to show your instructor your work in this milestone!

Deliverables to be shown:

1) Clicking on the Add note button should take us to a new screen to add notes.
2) Clicking on the Save button returns us to the home screen for the user which displays the list of all notes for the user.
3) Notes list of a user should be visible after a re-login and after app restart as well.

Milestone 3: Storing notes content in a file (Optional - no need to show this)

As part of milestone 2 you have successfully saved the contents of your note in a database. But think of how big these notes can get. Storing large strings like these in a database can be expensive and provides less value. We will instead be saving the content of notes in a file.

The next milestone of this lab is to create a file for every note and save it to the local file directory. Instead of storing the note text into the database, we will instead store the file name in our database. Use the column src in our table “notes” to store the filename.

This file should be uniquely identifiable for a user even if duplicate note titles exist across users, hence choose a file name carefully.

Code samples:

Add them wherever you are reading writing the database content.

Storing data to a file

Write to file <filename> using the following code snippet:

```
String fileName = "myFile";
String fileContents = "HelloWorld";

try(FileOutputStream fileOutputStream = getApplicationContext().openFileOutput(fileName,
Context.MODE_PRIVATE)) {
   fileOutputStream.write(fileContents.getBytes());
}
```

- **Reading data from a file**

  Open file using `FileInputStream fis = context.openFileInput(filename)`

  Example:

```
public String readContent(String filename) {

   StringBuilder content = new StringBuilder();
   try {
       FileInputStream fis = this.openFileInput(filename);
       InputStreamReader inputStreamReader =
               new InputStreamReader(fis, StandardCharsets.UTF_8);

       BufferedReader reader = new BufferedReader(inputStreamReader);

       String line = reader.readLine();
       while (line != null) {
```

```
            content.append(line).append("\n");
            line = reader.readLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return content.toString();
}
```

Conclusion

Great work getting through this lab! You have successfully persisted all your data in the local storage. You have put one step forward in the direction of recovery. The APIs we mentioned here can do a lot more than we explored in this lab. Feel free to read up more about them in the references section.

References:

- https://developer.android.com/training/data-storage/shared-preferences
- https://developer.android.com/training/data-storage/sqlite
- https://developer.android.com/training/data-storage/app-specific
- https://www.eversql.com/sql-query-formatter/