Demo Two

Cal-Task

Brad Stiff

Table Of Contents

Scope

Project Description

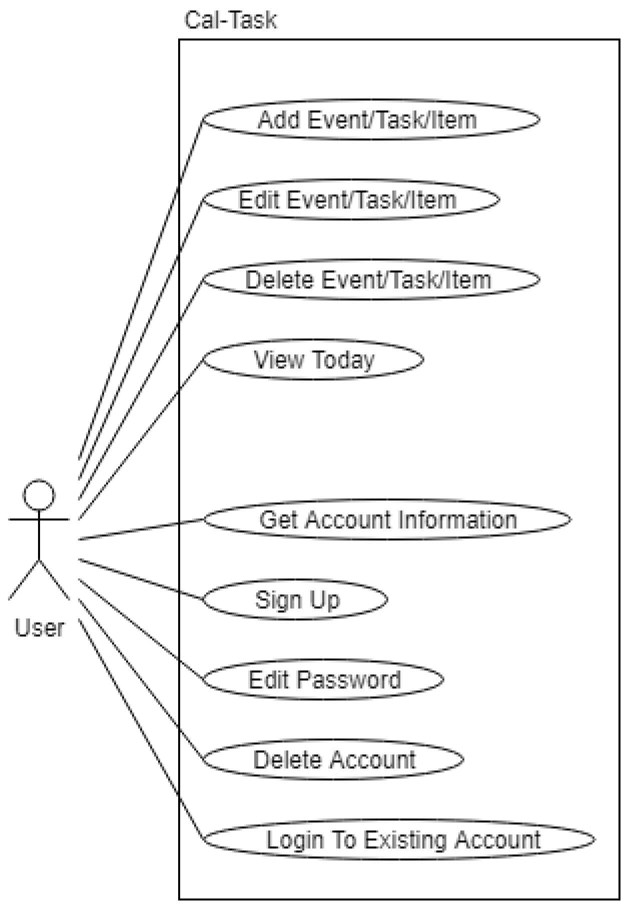
Goal

This project will consist of creating an android application that allows linked users to share tasks, events, and grocery lists. The user will be able to list these objects as private or public among the linked users. A wall screen, essentially a wall calendar, will act as a home entry point for the system, or display the events/lists of the household.

Deliverables

- Android Application
- Monthly Demo Reports
- Wall Screen Running Android

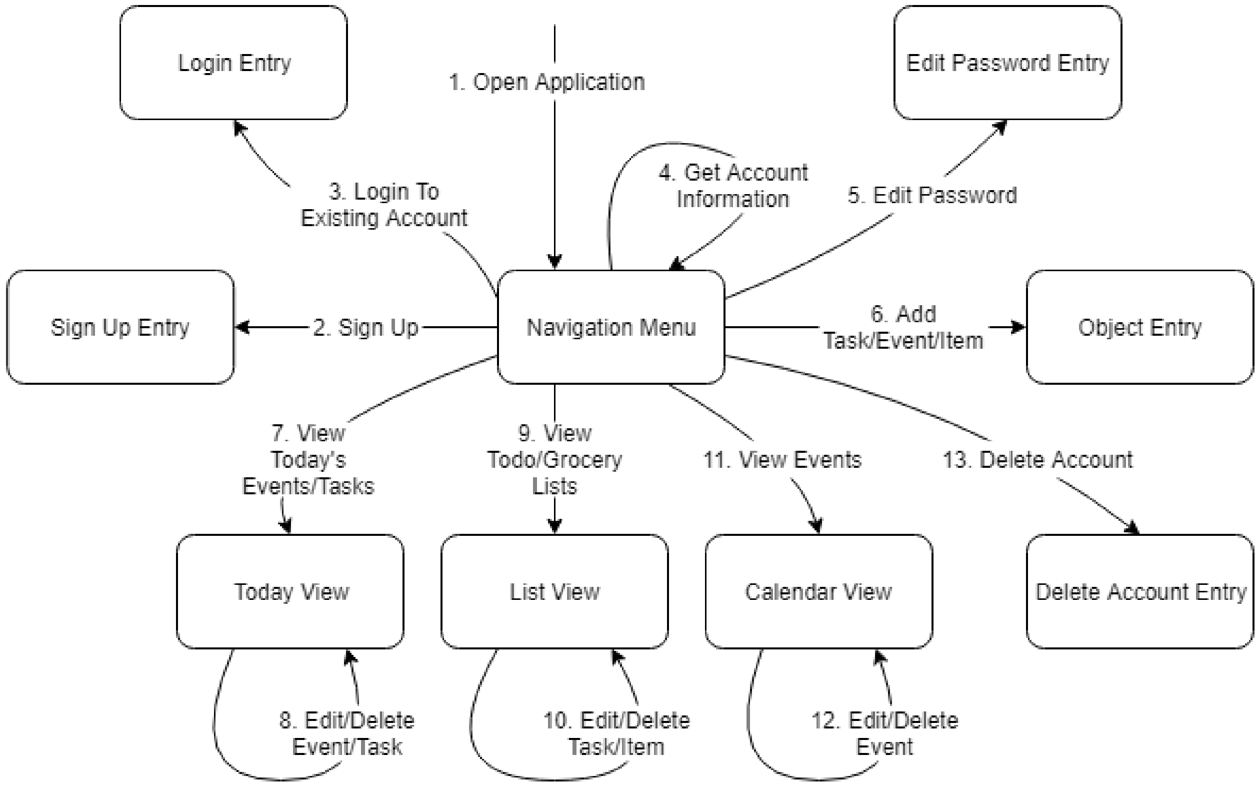
Use Case Diagram



Use Case	Description
Add Task/Event/Item	**Task:** Adds task to database. The user has the option of giving their task entry a due date. If the task has an associated date, it will also appear on the calendar. **Event:** Adds event to the database. The user can add a date, time, and location to the event. **Item:** Adds an item to the database. The item will appear on the Lists tab in the Grocery List Section. **All:** Objects that are listed as public will show up for all linked users. Public objects can be edited by linked users, but can not be changed to private by a linked user.
Edit Task/Event/Item	**Task:** Allows the user to edit the task entry or date. Tasks can be entered without a due date, if a date is added, it will also appear on the calendar. If a date is removed, it will also be removed from the calendar, but remain on the Todo list. **Event:** Allows the user to edit an event, date, time, or location. **Item:** Allows the user to edit a Grocery List item. **All:** If an object is changed from public to private, all linked users outside of the object creator will lose access to said object. Only the object creator can change object back to private.
Delete Task/Event/Item	**Task:** Removes the task from the database. Completing the task essentially does the same action as deleting. **Event:** Removes the event from the database. **Item:** Removes the item from the database. Items that have been picked up essentially does the same action as deleting the item.

	All: Public objects that are deleted are also removed from linked users. Both users can delete objects regardless of object owner.
View Today	Polls the events and tasks with the current date.
Get Account Information	Displays the currently logged in account's information stored in the local database.
Sign Up	Creates an account with the given information.
Edit Password	Allows the user to edit the password to their account by giving the current information along with the updated information to be changed.
Delete Account	Allows the user to remove their online account by giving the current account credentials.
Login To Existing Account	Requests existing account's credentials from online storage and stores in local database.

Communication Diagram



Use Case	**Communication Path/Description**
View Today	1. Open Application 7. View Today's Events/Tasks **Description:** Upon opening the application, the Today tab/view is already selected to show today's events and tasks.
Add Task/Event/Item	1. Open Application 6. Add Task/Event/Item **Description:** On the toolbar above the tabs, there will be a button that displays a pop-up window will allow the user to enter any of the objects into the database. Since it is placed here, all views will have access to creating new objects.
Edit Task/Event/Item	**Task:** 1. Open Application

	7. View Today's Events/Tasks -or- 9. View Todo/Grocery Lists 8. Edit/Delete Event/Task -or- 10. Edit/Delete Task/Item **Description:** When the user is viewing either the Today View or the Lists View, they can click on the task item (row) and choose to edit or delete the object. **Event:** 1. Open Application 7. View Today's Events/Tasks -or- 11. View Events 8. Edit/Delete Event/Task -or- 12. Edit/Delete Event **Description:** When the user is viewing either the Today View or the Calendar View, they can click on the event item (row), or date on the CalendarView object to edit or delete the event. **Item:** 1. Open Application 9. View Todo/Grocery Lists 10. Edit/Delete Task/Item **Description:** When the user is viewing the Lists View, they can click on the grocery item (row) to edit or delete the item.
Delete Task/Event/Item	**All:** Same paths as edit
Get Account Information	1. Open Application 4. Get Account Information **Description:** Selecting the "info" option in the main menu will display the current user's account information stored in the local database.
Sign Up	1. Open Application 2. Sign Up **Description:** Selecting the "sign up" option in the main menu will ask the user for the required information to create an account to post to the online database.
Edit Password	1. Open Application 5. Edit Password

	Description: Selecting the “edit password” option in the main menu will ask the user for the current account information and the new password.
Delete Account	1. Open Application 13. Delete Account **Description:** Selecting the “delete account” option from the main menu will ask the user for the account credentials and remove the account from the database.
Login To Existing Account	1. Open Application 3. Login To Existing Account **Description:** Selecting the “login” option in the main menu will ask the user for the credentials to the existing account and pull the information from online storage to local storage.

Implementation

Object Storage

Problem

The application needs to be able to store objects privately or publicly. The public objects also need to be accessible by more than one user. The application should work without internet connection, besides the sharing aspect, online useage should only be needed when necessary (posting of public objects). When opening the application or posting a public object, the user should not have to enter their account credentials every time. The last requirement is free hosting, due to this project’s workload, paying for hosting would be deemed superfluous.

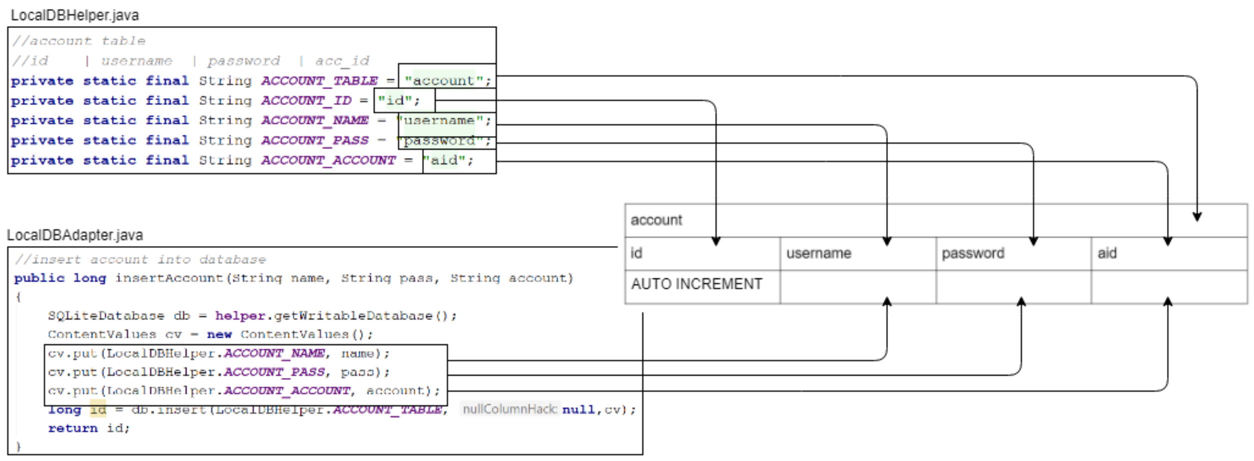
Problems to solve:
- Local & Online Storage
- Free Hosting
- Self contained Account Credentials

Solutions

SQLite Database

Description

Each android device comes preinstalled with SQLite. This feature allows for an SQL “like” database to be stored locally on the phone. Because of this, the system is serverless and self-contained. In order to access the SQLite information, the user has to be physically using the device storing the database.


LocalDBHelper.java

```
//account table
//id    | username  | password  | acc_id
private static final String ACCOUNT_TABLE = "account";
private static final String ACCOUNT_ID = "id";
private static final String ACCOUNT_NAME = "username";
private static final String ACCOUNT_PASS = "password";
private static final String ACCOUNT_ACCOUNT = "aid";
```

LocalDBAdapter.java

```
//insert account into database
public long insertAccount(String name, String pass, String account)
{
    SQLiteDatabase db = helper.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(LocalDBHelper.ACCOUNT_NAME, name);
    cv.put(LocalDBHelper.ACCOUNT_PASS, pass);
    cv.put(LocalDBHelper.ACCOUNT_ACCOUNT, account);
    long id = db.insert(LocalDBHelper.ACCOUNT_TABLE, nullColumnHack: null,cv);
    return id;
}
```

account			
id	username	password	aid
AUTO INCREMENT			

The account “id” tuple is solely used as the unique identification number for the account in local storage used for querying.

Pros

Since the database is stored on the device, the application will always have access regardless of connectivity. Implementation is free, and the database structure is similar to previously created databases. The locally stored objects are secure from the outside world, because only the application on the physical phone can access it.

Cons

Due to the database being stored locally, this option will not work for the publicly posted objects. Public and privately created objects can not be stored in the same location.

MySQL Hosted Database

Description

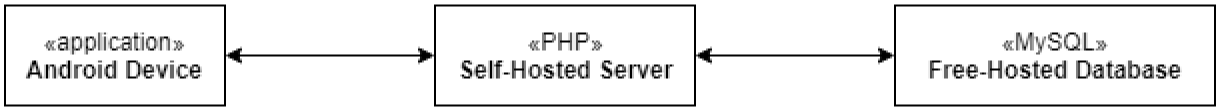
Storage of the accounts and associated object tables could be stored in an online MySQL database. To access the database, the phone would use its internet connection to connect to the server for the requested information.

Pros

The service guarantees up time and provides information backup for free. Implementation of the PHP requests have personally been completed before, so reimplementation should be relatively easy. Databases created can also be managed using the provided phpMyAdmin tool.

Cons

In order to interact and access the hosted database, the application would have to connect through a server. This server will need to either be implemented personally or through another online service. To personally implement a server, a Raspberry Pi could be used, but would only work if the device shared WiFi with the Raspberry Pi. This ultimately defeats the purpose of hosting the database in the first place.



Firebase Database

Description

Google offers a free cloud service called Firebase which allows for many different online services. One of these services is the Realtime Database which is a NoSQL cloud database. It has the ability to sync across all clients and can still be accessed when the application is offline. NoSQL means that it is a technology outside of SQL is used to create the database structure.

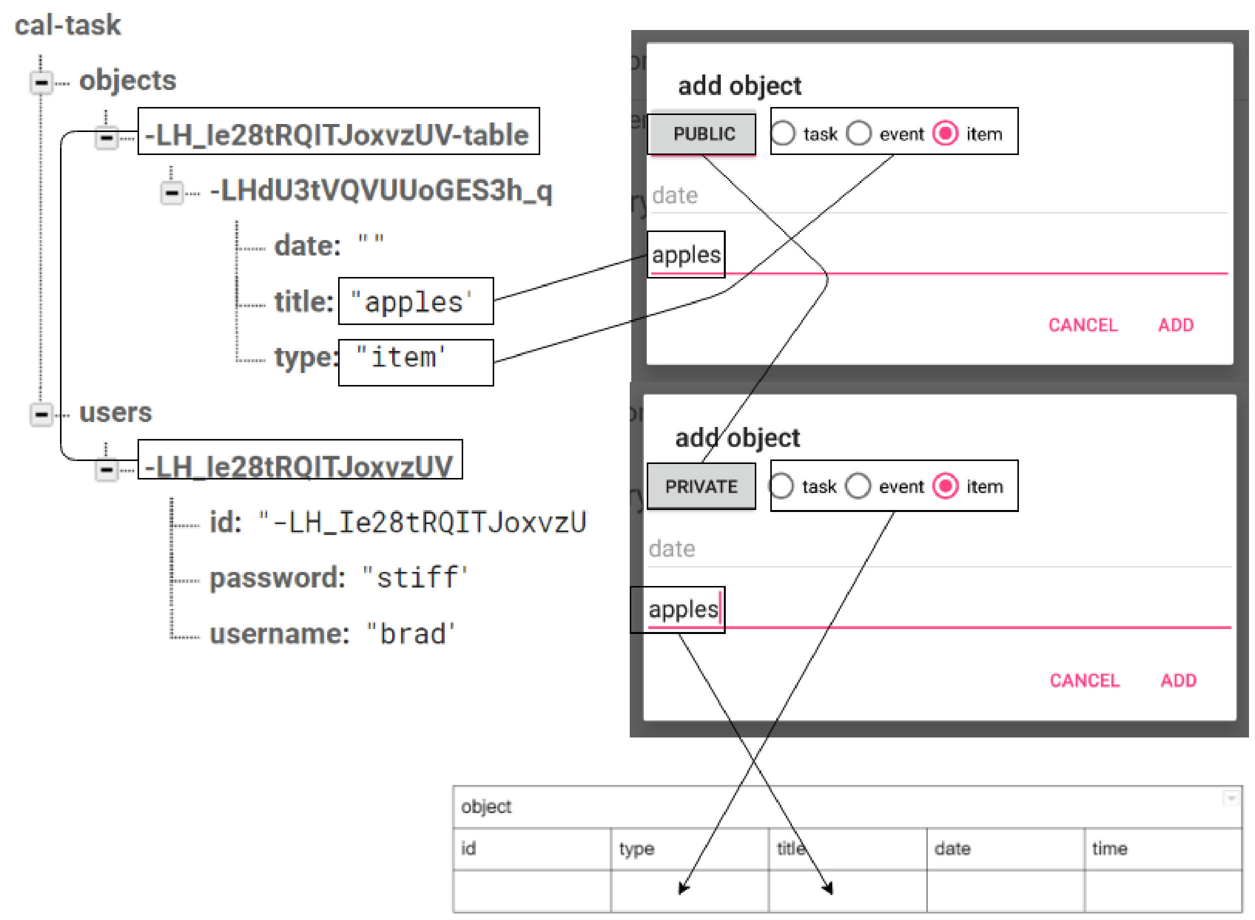
Pros

This option allows for the database to be accessible from anywhere with an internet connection. It also does not require a server with PHP scripts to be implemented for accessing the database. It updates in realtime, so all of the linked users will be updated instantly.

Cons

Although the database suggests that it works offline, it is not the whole truth. It stores your pushed data locally until it connects to the internet again. This could cause problems if trying to create a user account while having no online connection.

Design Decision



Description

A toggle button will separate the the online and offline object creation. If the toggle is set to public, the item is stored in the Firebase online database. With the toggle set to private, it will store the item in the local SQLite database.

Account Creation And Sharing Objects

Problem

One of the main features of the application, is to store created objects and share them with linked users. To be able to do this, there has to be a way to store all account IDs in one central spot to ensure unique account numbers. There also has to be a way for users to link accounts together without being accessible from anyone that has your account number. The application should also have the option to be used entirely offline. In order to share objects, the user will have to create an account.

Problems to solve:

- Unique Account IDs
- Store Account Information Locally
- Linking/Unlinking Accounts

Solutions

SQLite And Firebase Hybrid

Description

This option uses the SQLite database to store all of the privately created objects and Firebase, to store all of the public objects.

Pros

Ability to create unique user IDs by using the auto increment option in the IDs and using it as the account link.

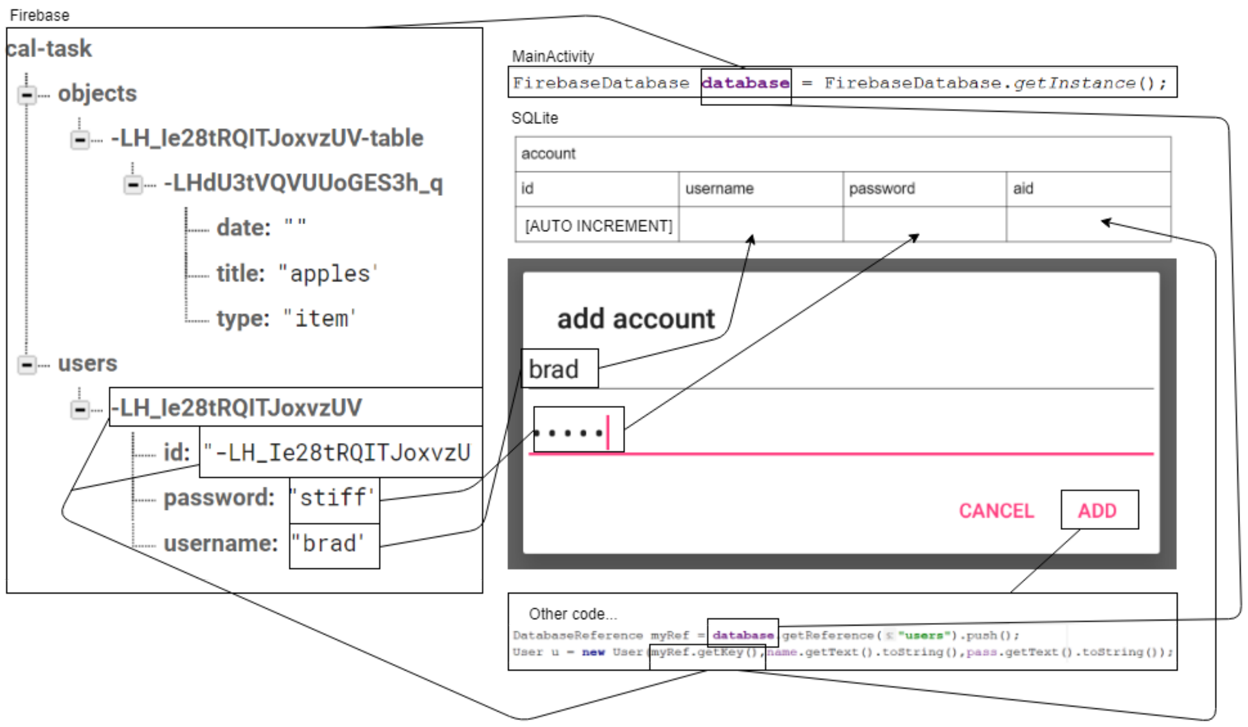
Cons

The two databases are different enough implementation that each query will need to be written twice, once for each system. If an object needs to switch from public to private, deletion will occur in one database while being added to the other without data loss. If an account is unlinked, all public objects need to revert to owner in respected database.

Design Decision

By choosing a hybrid between Firebase and SQLite, the application can provide all of the desired features. Since both of these services are already being used, adding the extra options will not require any new technologies.

A user needs to create an account in order to post objects publicly or to be able to link accounts across devices. When the account is created, Firebase assigns a unique account ID and all of the account credentials are stored in the SQLite database locally. These credentials will be used during public postings to provide account details in the background.


Firebase

```
cal-task
  objects
    -LH_Ie28tRQITJoxvzUV-table
      -LHdU3tVQVUUoGES3h_q
        date: ""
        title: "apples'
        type: "item'
  users
    -LH_Ie28tRQITJoxvzUV
      id: "-LH_Ie28tRQITJoxvzU
      password: 'stiff'
      username: "brad'
```

MainActivity

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
```

SQLite

account			
id	username	password	aid
[AUTO INCREMENT]			

Other code...

```
DatabaseReference myRef = database.getReference("users").push();
User u = new User(myRef.getKey(),name.getText().toString(),pass.getText().toString());
```

Description

When an account is created, a reference to the Firebase database is called. This reference then looks at the “users” table,(if there is not one, one is created) creates a unique key for the soon to be created object. The unique key is then used for the account id and is stored in both the Firebase object and the SQLite database. The SQLite database has an id field for system calls to this information. User class used for account structure shown below.

```
@IgnoreExtraProperties
public static class User
{
    public String id;
    String username;
    String password;

    public User() {}

    public User(String id, String username, String password)
    {
        this.id = id;
        this.username = username;
        this.password = password;
    }

    public String getId() { return id; }

    public void setId(String id) { this.id = id; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }
}
```

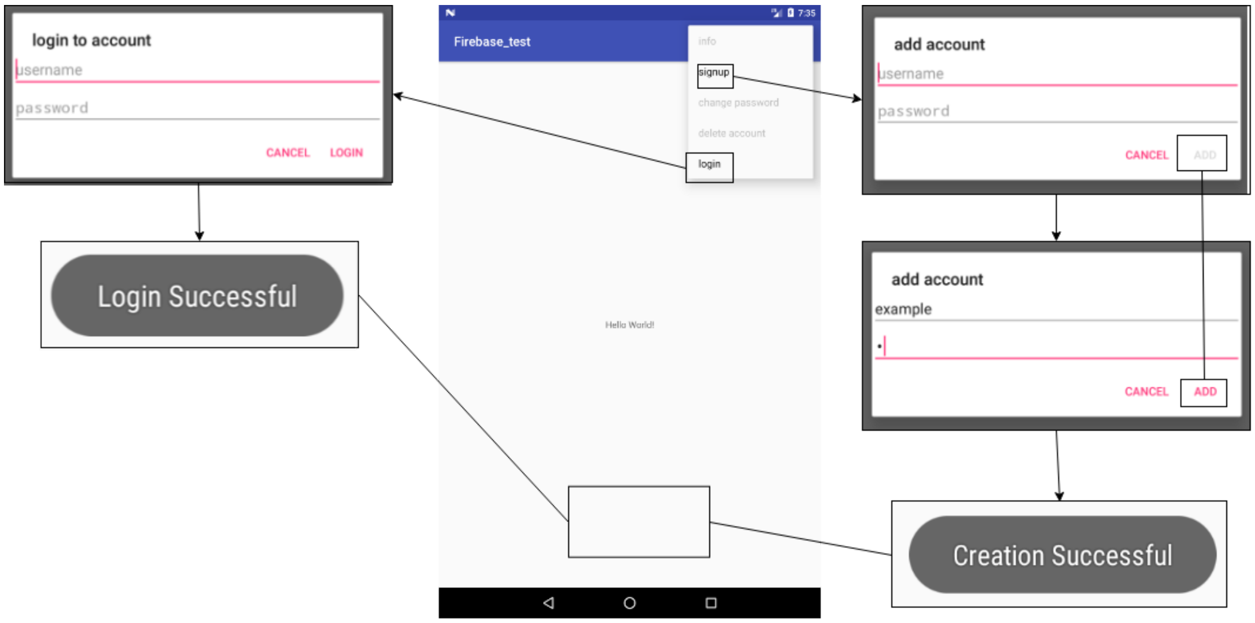
Linking Accounts

Linking accounts will be done by changing the aid in the SQLite database and the user id in the Firebase database to the desired account. When the user unlinks, the is will be reset with the unique key.

Tutorial

User Accounts

Login And Account Creation



Login

1. **User clicks main menu**
   This activates a dropdown menu allowing the user to make a selection. If an account is not in the applications local database, only the “sign up” and “login” options will be available.

2. **User selects “login”**
   Upon clicking the option, an alert window is triggered to ask the user for the username and password for the existing account. The “LOGIN” positive action button of the alert window with be enabled when both fields are not empty.

3. **User clicks “LOGIN” positive action**
   If the given credentials match an account in the Firebase database, the account

credentials will then be saved to the SQLite database. The user interface will then be updated to reflect the public items associated with that account. The user will be notified the results of the request via Toast popup.

Account Creation

1. **User clicks main menu**
   Same as login.

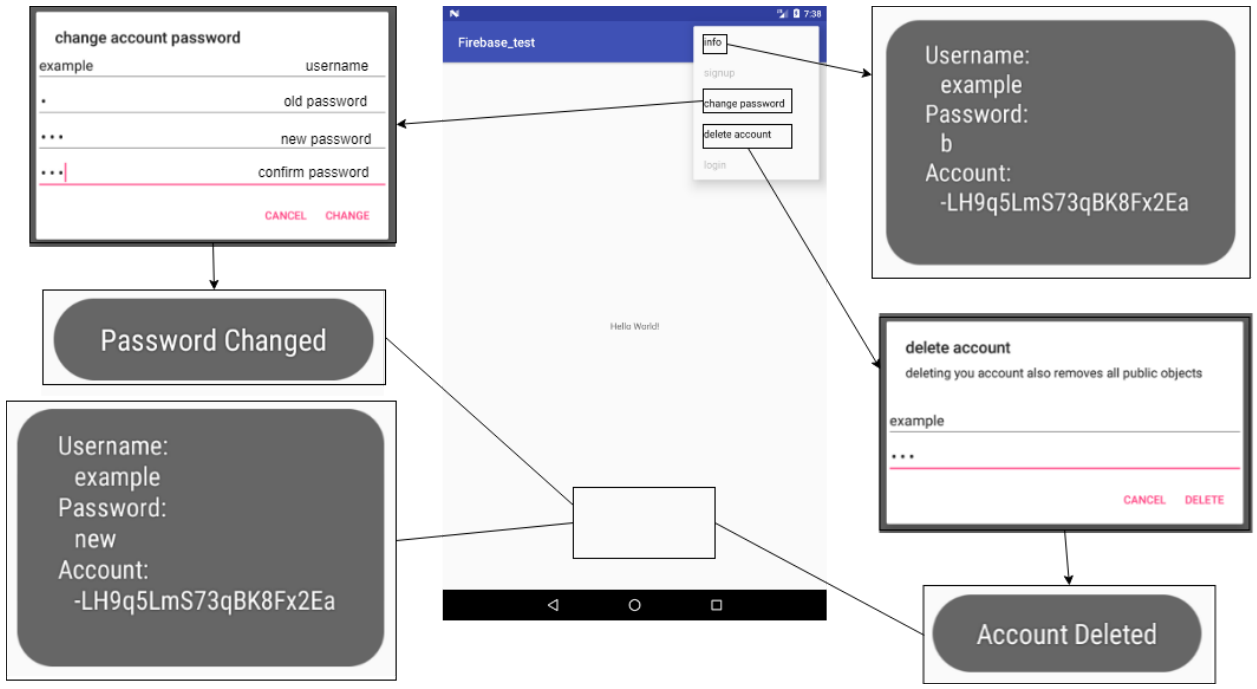
2. **User selects “sign up”**
   Upon clicking this option, an alert window is triggered to ask the user for the desired username and password. If the username is already taken, the username EditText will notify the user with an error. The “ADD” positive action will not enable if either field is empty.

   [PROBLEMS] Currently, a solution could not be found for blocking if the “ADD” positive action if the username is taken. All implemented options did not work to desired effect. The best version would work, but would offset by a keystroke and allow the duplicate username to be created. The setError() works flawlessly though.

   [TEMPORARY SOLUTION] If username and password are both unique, the same usernames can be found as long as the passwords are different. Creating an account with the same credentials will still produce a unique id in the Firebase database, but logging in will only return the first match in the database.

3. **User clicks “ADD” positive action**
   The application takes the given information and creates an account in the Firebase account table. Then it grabs the Firebase key and uses it as the account ID and pushes the information into the SQLite database. The user will be notified the results of the request via Toast popup.

Account Information, Editing Passwords, And Deleting Account



Username:
example
Password:
b
Account:
-LH9q5LmS73qBK8Fx2Ea

Password Changed

Username:
example
Password:
new
Account:
-LH9q5LmS73qBK8Fx2Ea

deleting you account also removes all public objects

Account Deleted

Account Information

1. **User clicks main menu**
   This activates a dropdown menu allowing the user to make a selection. If an account is in the applications local database, only the “info”, “change password”, and “delete account” options will be available.

2. **User selects “info”**
   Displays the account information stored in the account table of the local SQLite database via a Toast pop-up. Currently it displays the account’s username, password, and unique account id assigned by Firebase.

   [NOTE] This displays the account password as well, but only for testing purposes.

Editing Password

1. **User clicks main menu**
   Same as account information.

2. **User selects “change password”**
   Clicking displays an alert window to ask for the current username and password along with the new password and a confirmation of the new password. If the fields are empty, the positive action button is disabled. The confirmation EditText sets an error if it does

not match the given new password.

[PROBLEMS] This has the same problem of blocking the “CHANGE” positive action if the passwords don't match.

3. **User clicks “CHANGE” positive action**
   If the account is found online, the password is changed to the given new password. If not, a Toast pop-up will notify the user of the results of the attempt and reasoning behind its success or failure. The password is change in both the Firebase and SQLite databases.

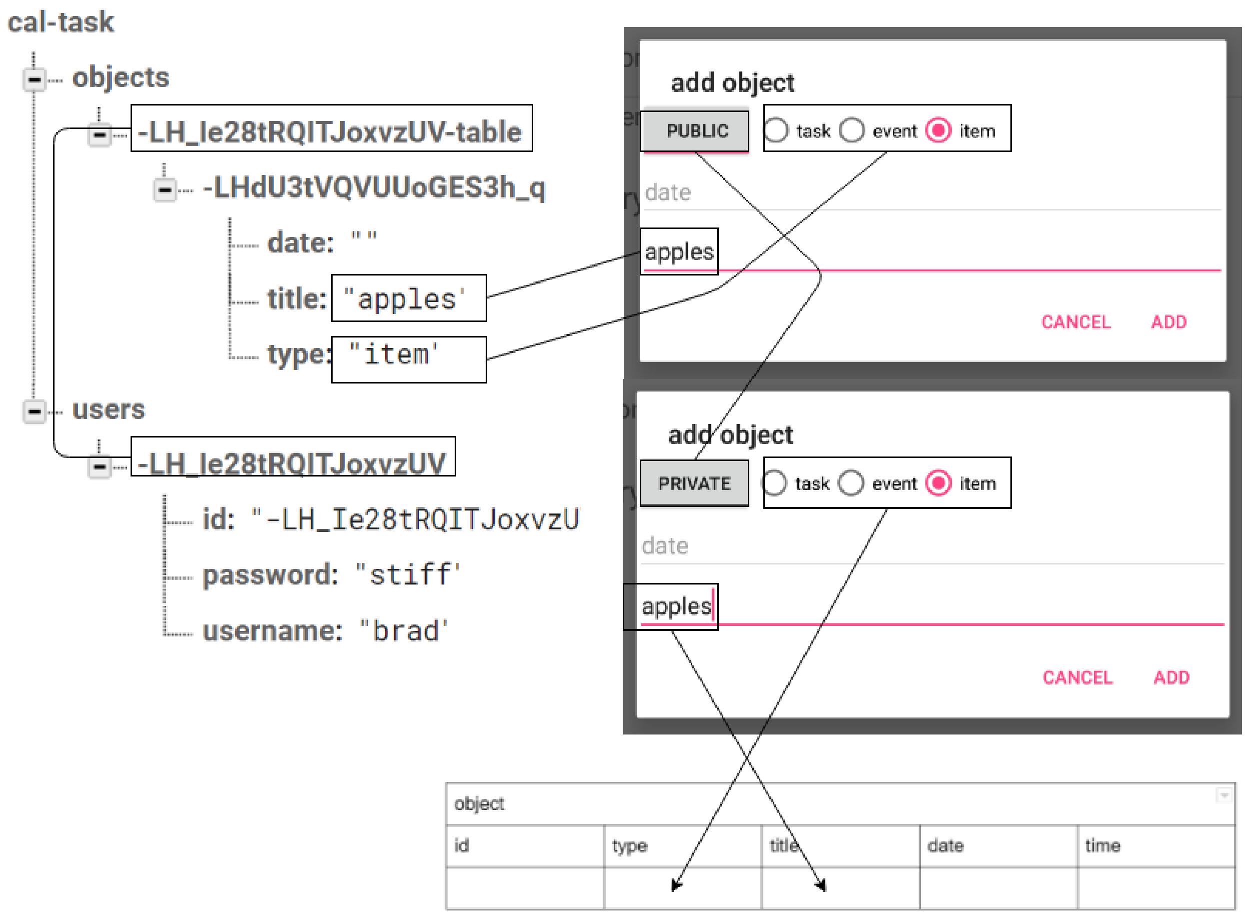
Deleting Account

1. **User clicks main menu**
   Same as account information.

2. **User selects “delete account”**
   Upon clicking, displays an alert window asking the user for the username and password of the desired account to delete. If either field is empty, the “DELETE” button will not be enabled.

3. **User clicks “DELETE” positive action**
   If the account is found in the Firebase database, it deletes the node from the database. Then it drops and re-creates the account table in the SQLite database. This is to ensure that the next account entered will have the first id, allowing the getAccountID methods to work properly. It then sets the main menu back to the state mention in login. Results of the user’s actions will be displayed via a Toast pop-up.

Creating Objects



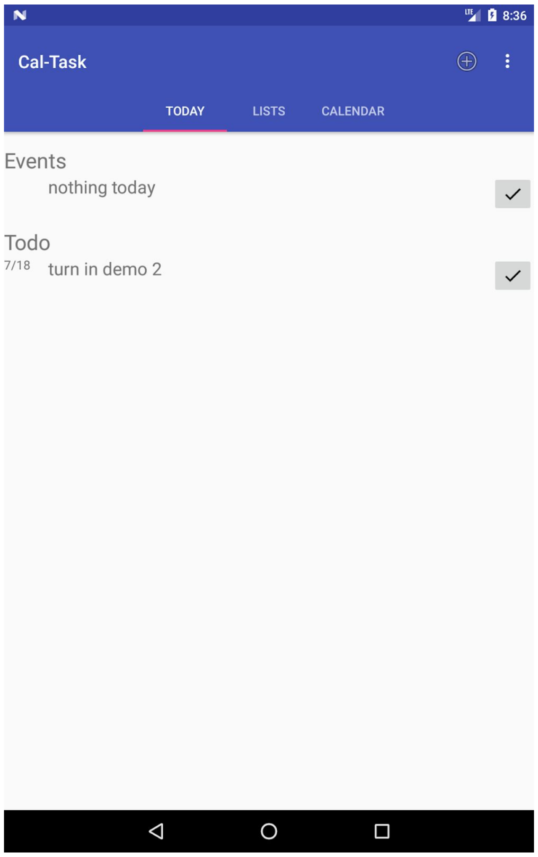
1. **User clicks "+" in main menu**
   This activates an alert window that displays the options for adding an object. Upon opening, the user can only select the public/private toggle button, the radio buttons, or cancel.

2. **User selects "public" or "private"**
   If the user selects public, the item will be stored in the Firebase database. Selecting private, stores the object in the SQLite database.

3. **User selects "task", "event", or "item"**
   Once the user selects one of the radio buttons, the associated date and time fields will become enabled. Changing the selected radio button enables the different fields.

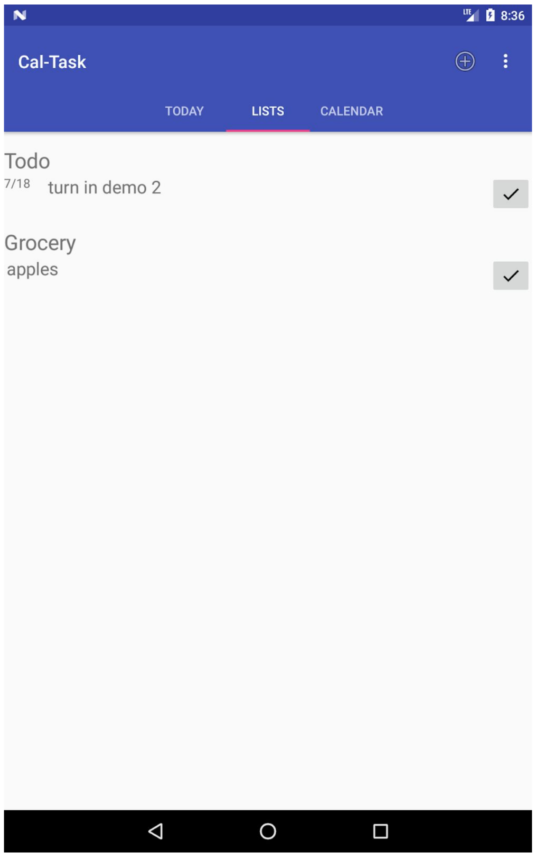
4. **User clicks "ADD" positive action**
   Once the title field is not empty, the "ADD" positive action becomes enabled. Once the item is added to the database, the listViews on the various tabs will update to reflect the

database addition.



Events
nothing today

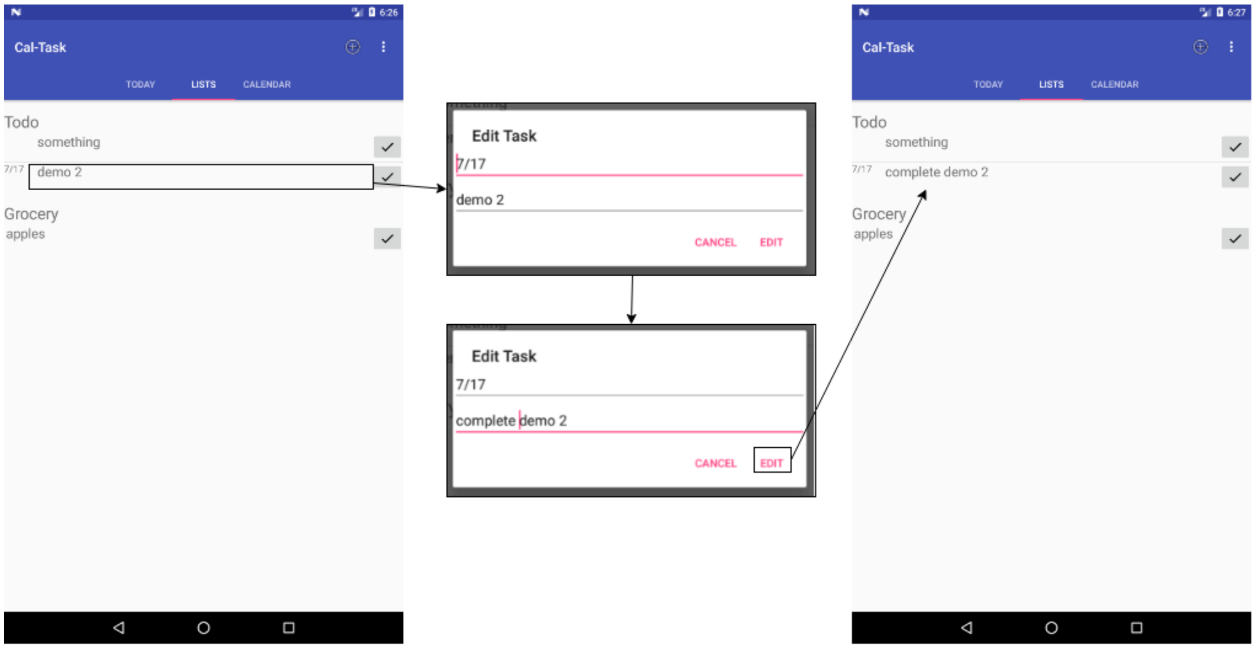
Todo
7/18 turn in demo 2



Todo
7/18 turn in demo 2

Grocery
apples

Editing Objects



Todo
something
7/17 demo 2

Grocery
apples

Edit Task
7/17
demo 2

Edit Task
7/17
complete demo 2

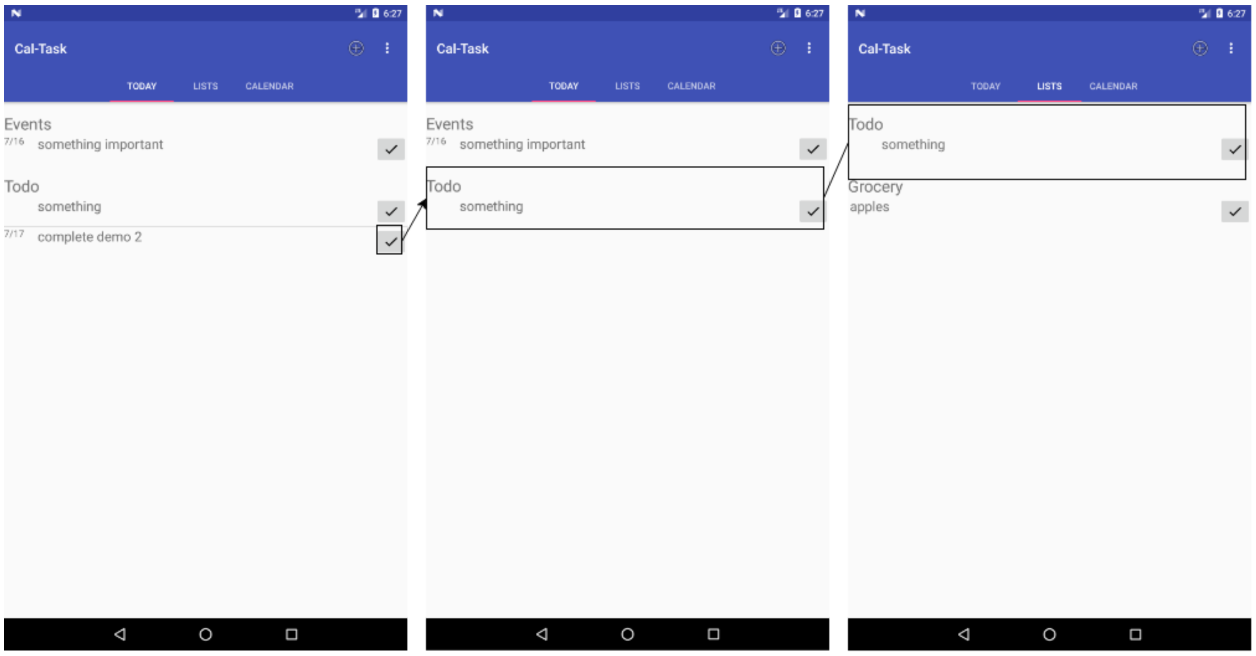
Todo
something
7/17 complete demo 2

Grocery
apples

1. **User clicks object's title**
   This activates an alert window with the fields to be edited. The fields will be populated with the information held inside the object.

2. **User clicks "EDIT" positive action**
   After the user changes the desired field, chicking the "EDIT" positive action will update the object in the appropriate database followed by updating the application's user interface.

Deleting Objects



1. **User clicks object's "✓"**
   Once the user clicks the image button, the object will be removed from the appropriate database followed by updating the user interface.

References

1. ListView Adapters
   https://github.com/codepath/android_guides/wiki/Using-a-BaseAdapter-with-ListView

2. How To Refresh BaseAdapter
   https://stackoverflow.com/questions/13672700/how-to-refresh-custom-listview-using-baseadapter-in-android

3. SQLite Todo List Application Tutorial
https://www.sitepoint.com/starting-android-development-creating-todo-app/

User Accounts

1. Programmatic EditText Password
https://www.android-examples.com/set-edittext-input-type-password-programmatically-android/

2. Programmatically Build Layouts For AlertDialog
https://stackoverflow.com/questions/32026299/programatically-create-layout-to-be-displayed-on-an-alertdialog

3. Get PositiveButton Reference In AlertDialog
https://stackoverflow.com/questions/10931638/get-positive-button-in-dialogpreference

4. Display Keyboard On Focus Change
https://stackoverflow.com/questions/9812258/default-focus-and-keyboard-to-edittext-in-android-alertdialog/13056259

Add Object Window

1. Show Keyboard When Changing EditText Focus
https://stackoverflow.com/questions/8991522/how-can-i-set-the-focus-and-display-the-keyboard-on-my-edittext-programmatical/28596779

2. Check If RadioGroup Has A Selected Button
https://stackoverflow.com/questions/24992936/how-to-check-if-a-radiobutton-is-checked-in-a-radiogroup-in-android

3. How To Add IDs To Programmatically Created Objects
https://stackoverflow.com/questions/1714297/android-view-setidint-id-programmatically-how-to-avoid-id-conflicts

Firebase

1. Realtime Database
https://firebase.google.com/docs/database/

2. Add Firebase To App
https://firebase.google.com/docs/android/setup?authuser=0

3. Read And Write Data
   https://firebase.google.com/docs/database/android/read-and-write?authuser=0