Demo Three

Cal-Task

Brad Stiff

Table Of Contents

Scope

Project Description

Goal

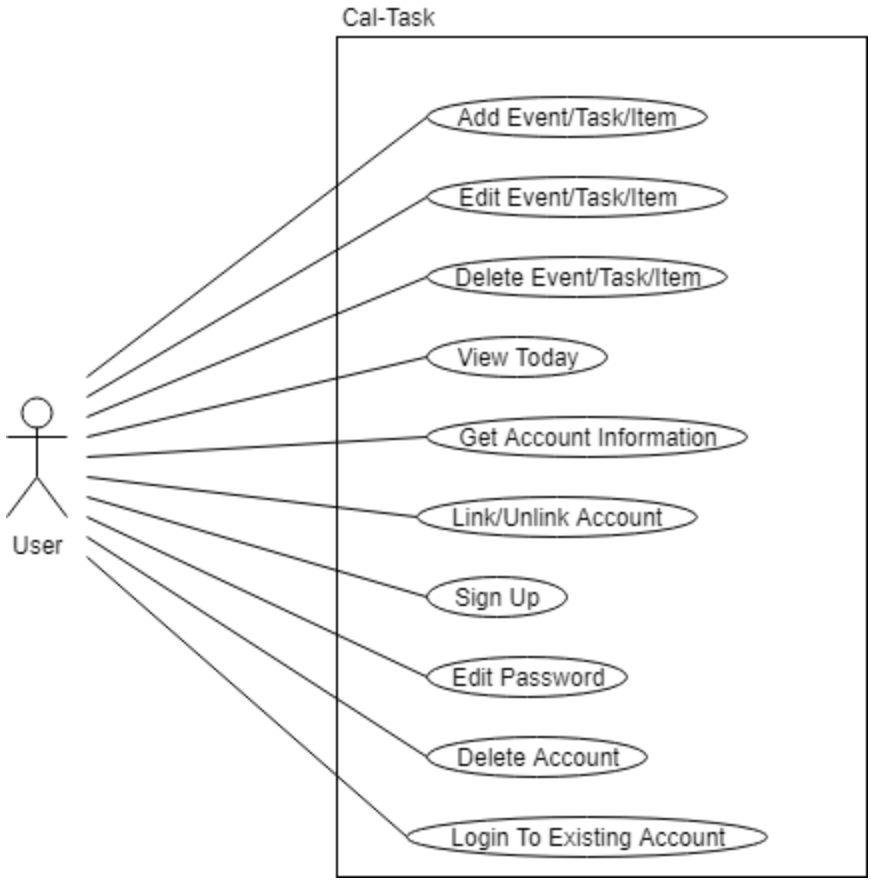
This project will consist of creating an android application that allows linked users to share tasks, events, and grocery lists. The user will be able to list these objects as private or public among the linked users.

Deliverables

- Android Application
- Monthly Demo Reports

Github: https://github.com/neoxis/IS_SUMMER_2018

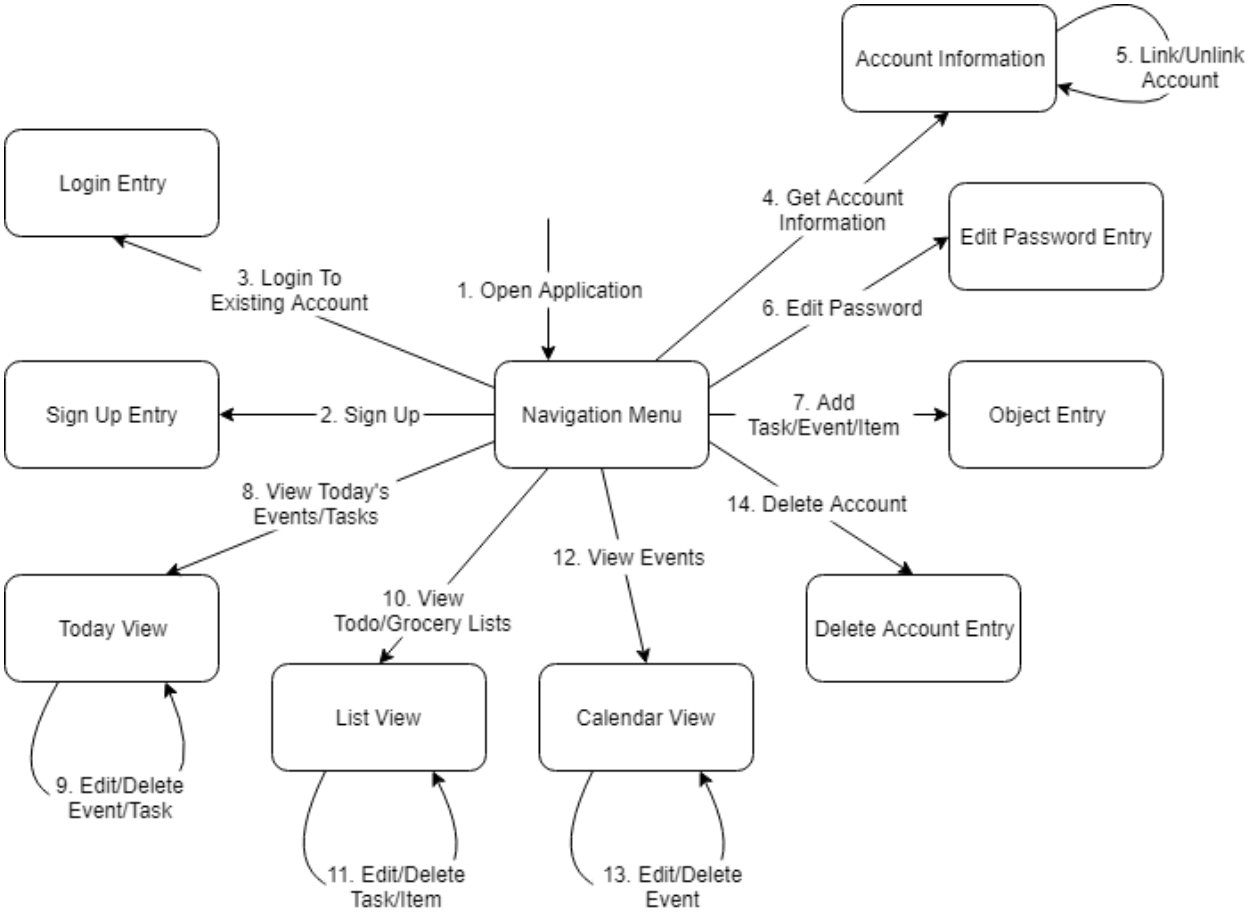
Use Case Diagram



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.
Link/Unlink Account	**Link:** Allows the user to connect to another user's object table and any public object created will be shared by both users. **Unlink:** Allows the user to disconnect from a shared table. **Note:** As of current implementation, the user's old object table will continue to exist. Unlinking returns the user to their original table. Future implementations will migrate public items and remove original table.
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 8. 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 7. 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 8. View Today's Events/Tasks -or- 10. View Todo/Grocery Lists 9. Edit/Delete Event/Task -or- 11. 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 8. View Today's Events/Tasks -or- 12. View Events 9. Edit/Delete Event/Task -or- 13. 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 10. View Todo/Grocery Lists 11. 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.
Link/Unlink Account	1. Open Application 4. Get Account Information 5. Link/Unlink Account **Description:** Below the account information mentioned above, is an entry field for a username and link ID to connect to the entered account.

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 6. 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 14. 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

Updating Fragments

Problem

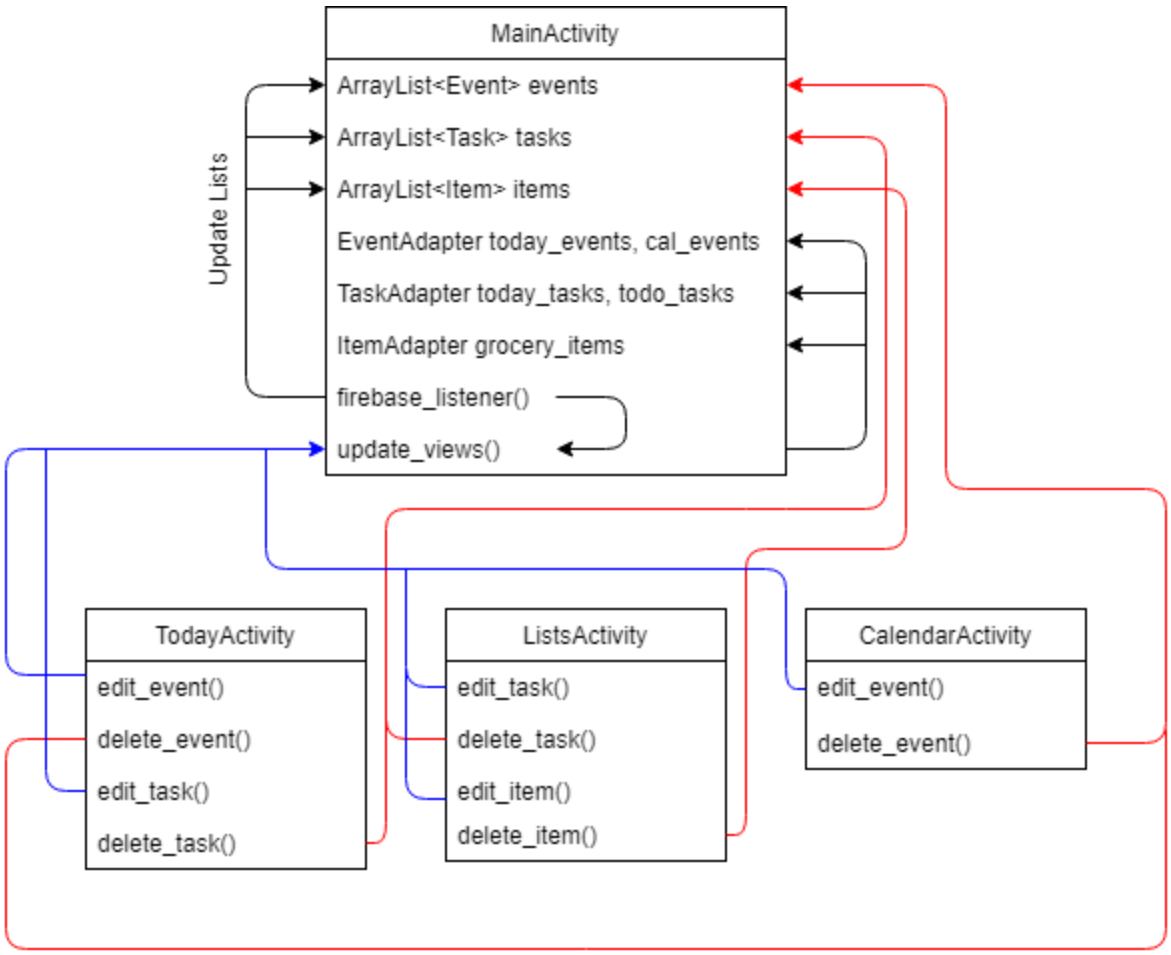
The different tabs of the application store data within them. This data is slightly shared between the activities and this shared data needs to be reflected on the appropriate tabs. For example, the Today View and the List View both can share a task if the task’s due date is the day of viewing. If the user is to complete the task for the Today View, the List View should reflect the change and vise-versa.

Solutions

Static Objects

Description

For the last demo, the choice was to put the firebase listener into the MainActivity and sort each into a static list of events, tasks, and items. Then the custom ListView Adapters would also be static, so the main menu could update the adapters upon item creation. This also proved useful for updating the ListViews from inside the fragments because of the static accessibility.



The firebase objects would be sorted into the ArrayLists held in the MainActivity. It would then call update_views() whenever an add, edit, or delete occurred. The firebase objects would update automatically due to the listener being triggered by data changes. Each class had access to the local database, so whenever a firebase object was changed, the views would be updated to display the current information, but local objects needed the static access to update.

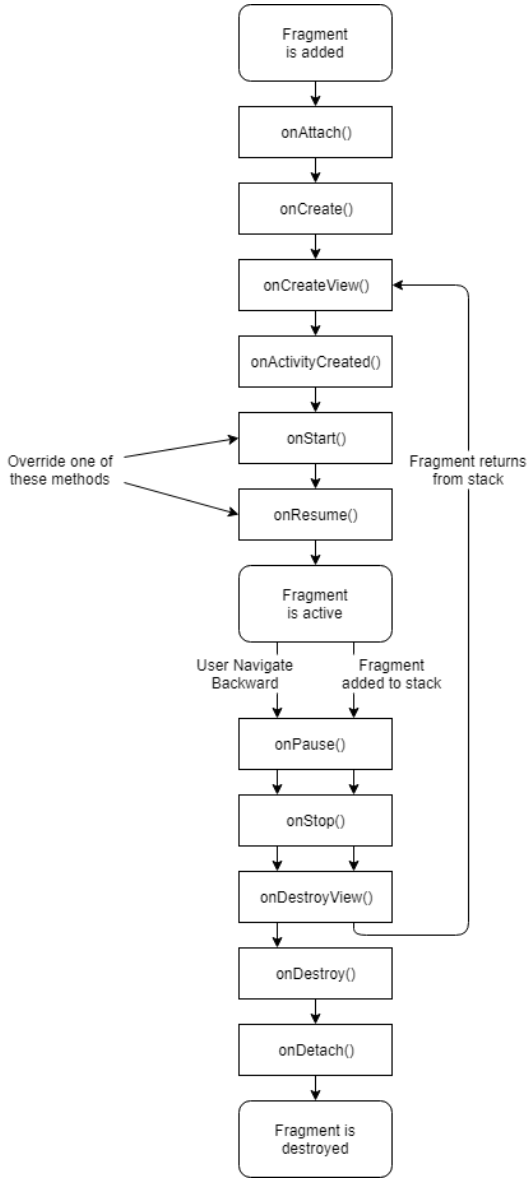
Pros

Splitting the information into object typed lists, storage, and updating can all be done in one location. Each class has static public access to all the objects that it needs to update.

Cons

Passing the application context to a static object cause many errors. Moving the Firebase listener to the main menu ruined my object editing functions due to the context not updating correctly. This also caused the lists to completely disappear if the tablet was turned into landscape or vise-versa. To negate the tablet orientation problems, the application was locked in portrait mode.

Override Method in Fragment Life Cycle



Description

When navigating through the tabs of the application, the fragments holding the various activities have the life cycle shown to the left. When a fragment comes back into view, the onStart() and onResume() methods will be called. If they were overridden, super would have to be called to retain needed functionality, but we can add other things inside these methods.

```
@Override
public void onResume()
{
    super.onResume();
    //do stuff
}
```

Pros

Although the views won't be updated simultaneously, they will still have the appearance of doing so. This method can call a function inside the activity to just before the fragment becomes visible to the user.

Cons

This did work in various tests, but whenever implementing the function to update the ListView it would crash. To use this option, I would have to only use it to do some background calls outside of updating the user interface. This ultimately makes this option useless.

ViewPager Listener

Description

The ViewPager handles the changing of the views when a tab is selected. By attaching an OnPageChangeListener here in the main menu, a method can be called to updated the views similar to the method mentioned above. The updates will appear to happen simultaneously but will only do so during page change. The MainActivity already has references to each of the fragment activities because of the SectionPagerAdapter that creates the tabs returns the class object.

```
public class SectionsPagerAdapter extends FragmentPagerAdapter {

    SectionsPagerAdapter(FragmentManager fm) { super(fm); }

    @Override
    public Fragment getItem(int position) {
        switch (position)
        {
            case 0:
                ta = new TodayActivity(); //grab reference
                return ta;
            case 1:
                la = new ListsActivity(); //grab reference
                return la;
            case 2:
                return new CalendarActivity();
            default:
                return null;
        }
    }
```

Pros

There are no static objects created, so the context is updated properly returning functionality to the editing objects methods and device orientation change.

Cons

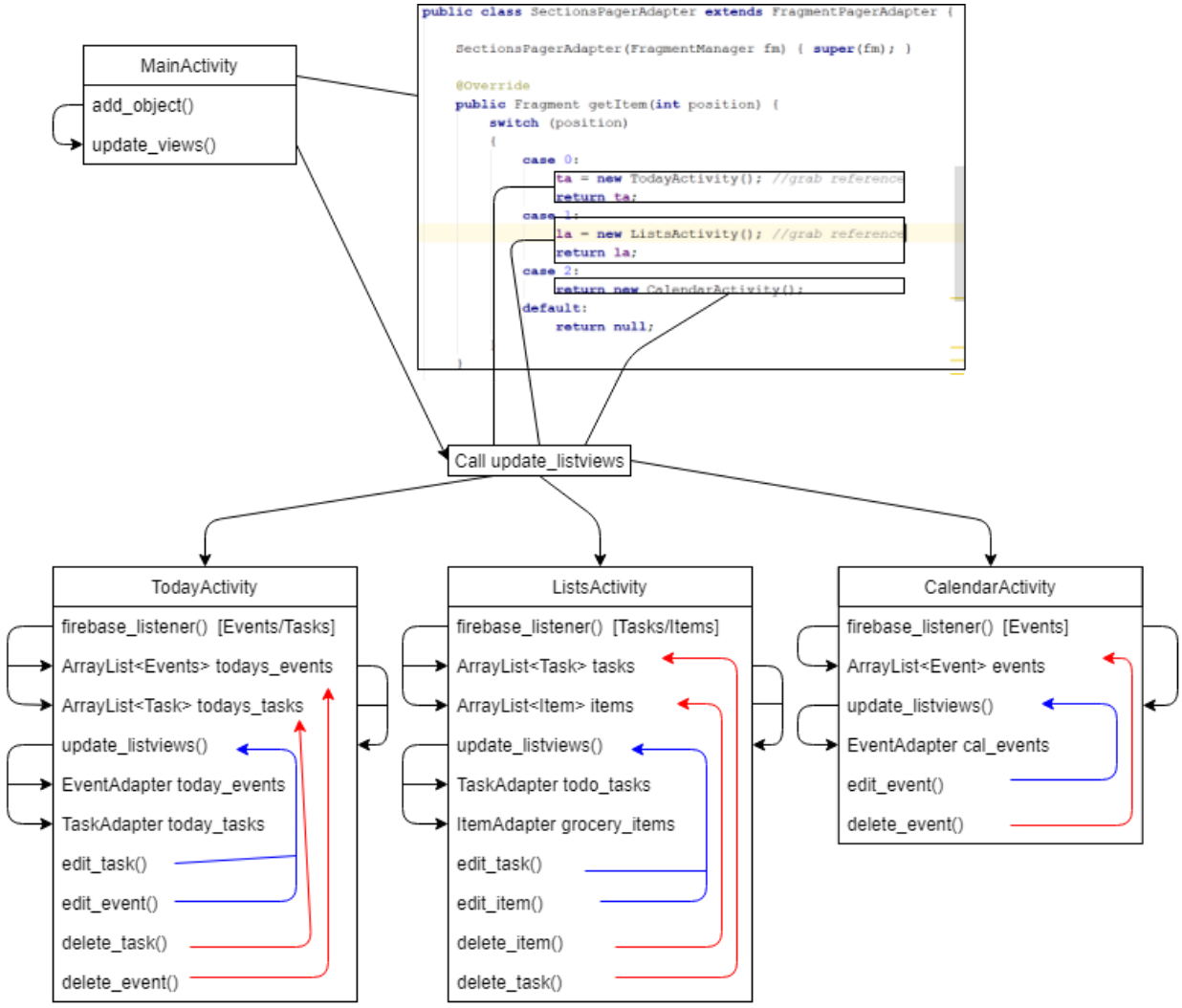
The ViewPager has two onPageChangeListener, one for setting the tab layout and the other handles the fragment updates. I don't know if it is bad practice to do so, but it doesn't appear to cause any problems.

Design Decision

The choice was made to move a firebase listener to each of the three sub-activities. This was done to make the activities more modular and the three different listeners help to individually update the activities simultaneously. When a public object is created, it triggers every listener attached at that location in the firebase. Since each activity has their own listener, each activity is updated upon a public creation. Giving each activity an instance of the LocalDBAdapter class allows the activity to pull the local objects upon firebase updates. To handle the updating of local objects, the ViewPager listener was used to update each fragment.

The current fragment was updated with a method call to update the activity's ListView. This is the same method used when the firebase listener is triggered.

The references to each sub-activity held in the MainActivity is used to update each view when a local object is deleted. This method does not need to be called when a public object is deleted due to it triggering the firebase listener which in turn calls for an update of the ListView. The only thing that needs to be done, is have the object removed from the ArrayList of objects in the activity. This will remove the object from the list, so it will not appear after the next call to update.



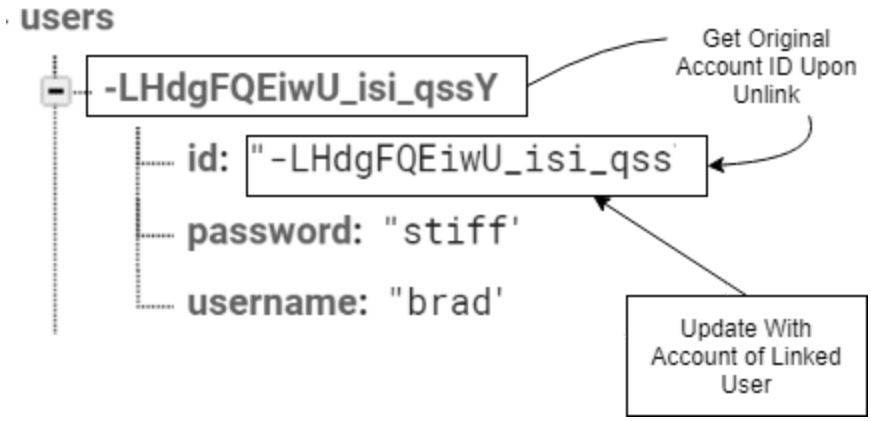
Whenever an object is created, the MainActivity calls the appropriate activity's update_listviews() method. It obtains a reference to each fragment activity by using the SectionPageAdapter's getItem method. Inside here, is a switch that returns the activity associated with that position. All of the editing and deleting happens inside the each of the individual activities. The firebase updates are handled by the listener attached on application open, login, or sign up. To update the other fragments that could share an object, a ViewPager Listener was added to the MainActivity. When the user switches to another fragment, the MainActivity calls the update_views() method.

When deleting public items, the items need to be removed from the arraylist or else it remains after the delete. This phantom object will be successfully deleted from the firebase, but will still be in the arraylist of the activity. This is because the firebase listener only emptied the arraylist on new item entries. Local objects are just pulled from the SQLite database by giving each activity access to the LocalDBAdapter class. Whenever an update_listviews() is called, it pulls the local database objects of the desired type and adds the activity contained arraylist to it and updates the ListView.

Linking Accounts

Problem

The decisions made in the previous demo weren't as fruitful as intended. The original idea was to change the account ID when linking two users to the the same public table, but that ended up destroying the functionality of some previously implemented features. Another problem was how exactly to find the other user's account ID without giving away their password, but requiring more than solely a username to link. I could not just have users guess usernames until they gained access to another user's table.



Solutions

Use Account ID as Link ID

Description

This option would be able to use the structure defined above, with the addition of one extra tuple to hold the original account ID. To implement, login, the account table, and account information methods would have to be changed.

Pros

Using the firebase generated unique ID would prevent me from having to create multiple tuples for a link ID. Here, we could just save the original ID and only create one new tuple.

Cons

Entering in the huge ID that firebase generates would be kinda ridiculous. Everyone also would have the ID of an account, and could possibly tamper with it. Past methods would have to be rewritten to ensure proper operation. Having to only check against the account ID matches to gain access to another user's stuff isn't very secure. A user could randomly enter account IDs to find a viable link.

Create Link ID and Original Account ID Fields

Description

Creating two new tuples in the account table, one for the link ID, and the other to store the original account ID. The link number will be a random four-digit number. In order for a user to link accounts, the other user would need the username and the link ID of the user that they wish to connect to.

Pros

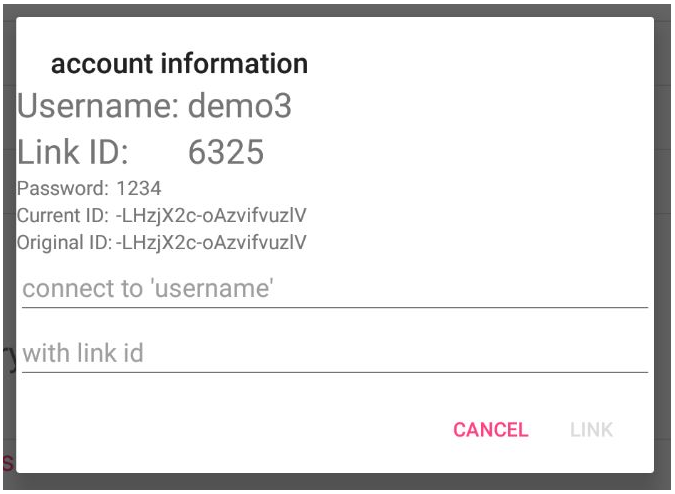
Past methods will retain their functionality and will not have to be re-written. Using this two-step verification is more secure for connecting users. Entry will be a lot easier as well, due to entering only the username and four-digit linkID instead of the account ID.

Cons

The account table and the account information methods will both need to be re-implemented.

Design Decision

Since linking is so closely related to account information, and because that method needs to be re-written, it will be displayed in an alert window instead of a Toast pop-up. Since it is in an alert window now, the user can leave the information on the screen as long as needed for the other user to link.

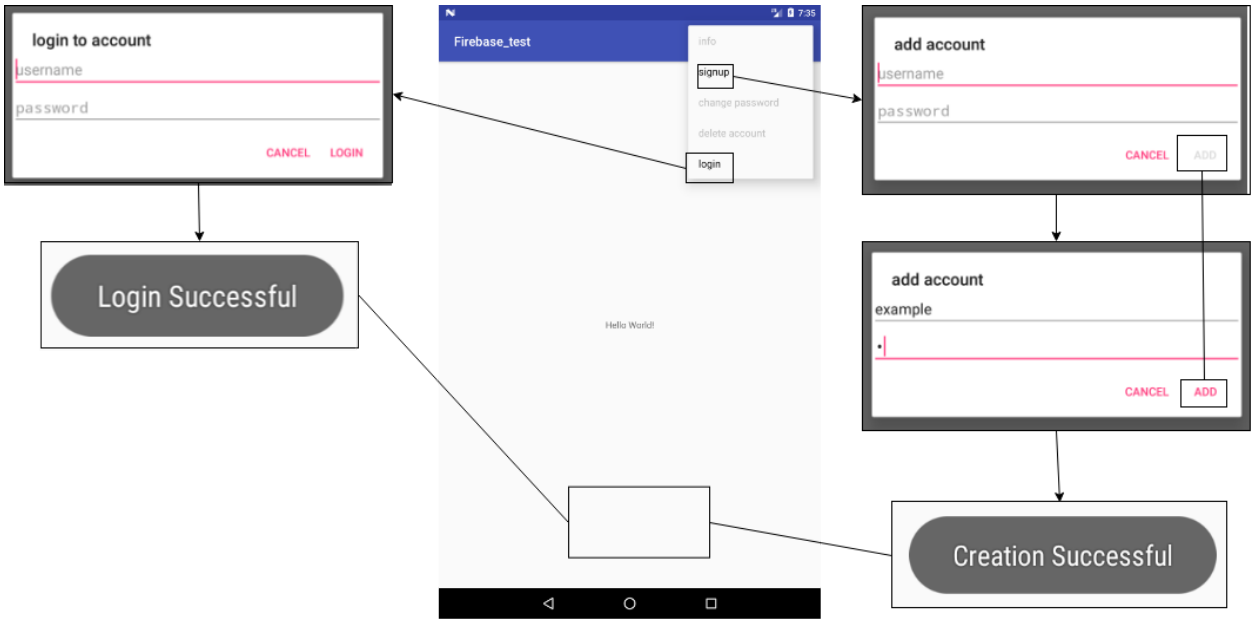


The password and IDs will not show up in the final product, but is there for testing purposes. The link button remains disabled until both fields have something in them and the link ID has four-digits. Once the credentials are accepted, the current ID field will change in the databases.

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. This also attaches the firebase listeners.

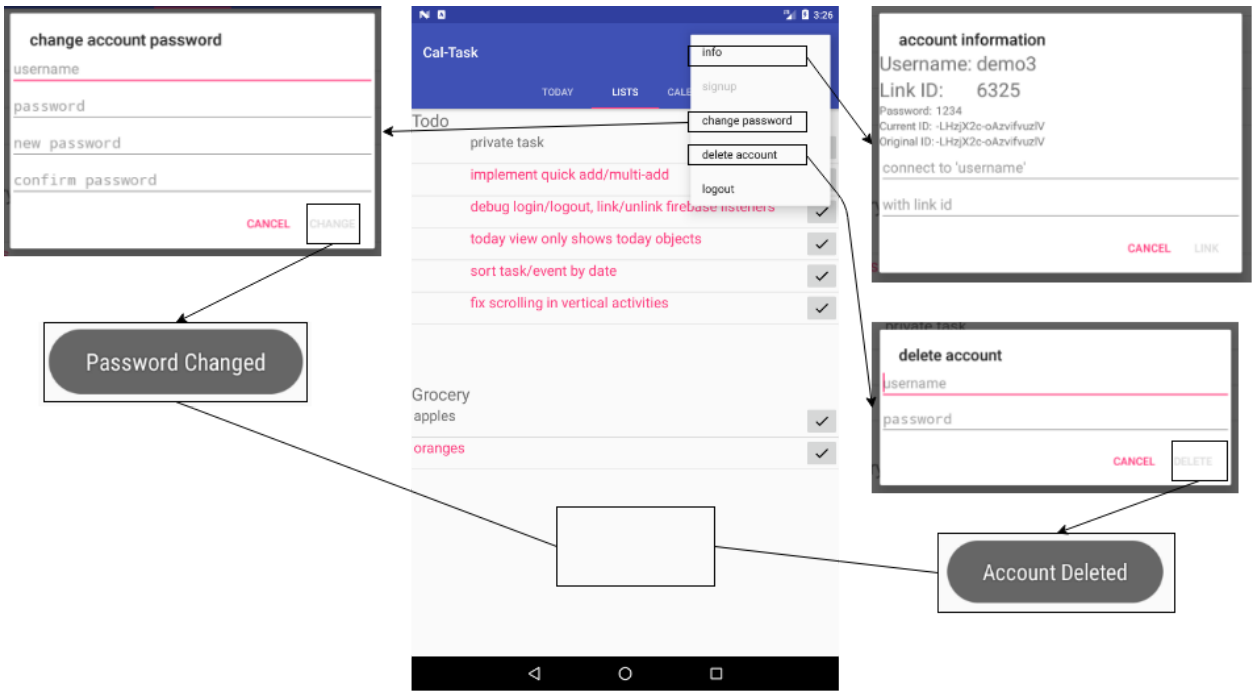
Account Creation

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.

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



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 an alert window. Currently it displays the account’s username, password, unique account id assigned by Firebase, original account id, and a four-digit link id.

   [NOTE] This displays the account password and account IDs 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.

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.

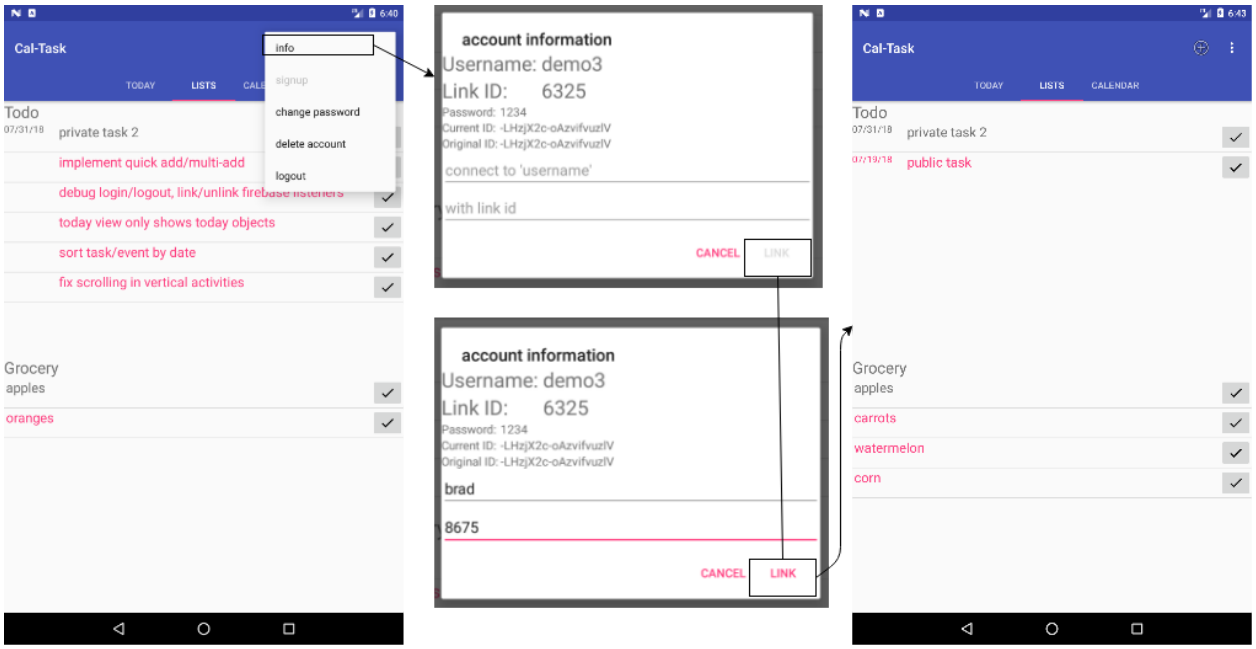
2. **User selects “delete account”**
   Upon clicking, displays an alert window asking the user for the username and password of the desired account to delete. If either field is empty, the “DELETE” button will not be enabled.

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.

Account Linking And Unlinking

Linking Accounts

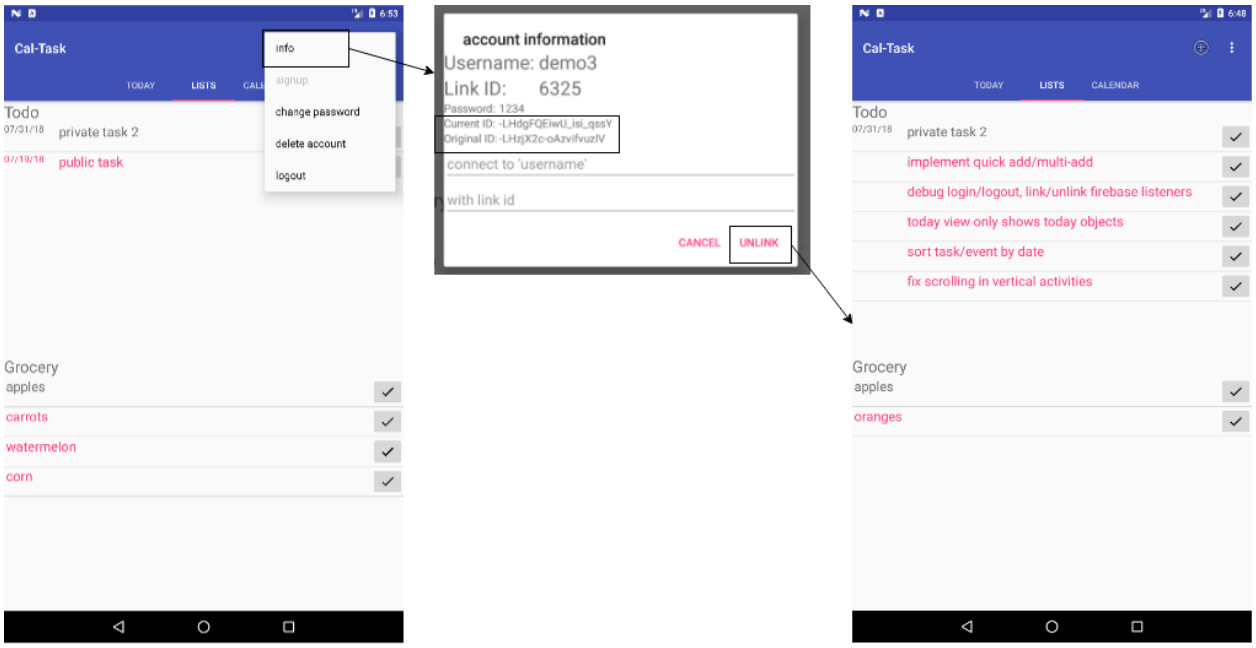


1. **User clicks main menu**
   This activates a dropdown menu allowing the user to make a selection. If an account is in the applications local database, only the "info", "change password", and "delete account" options will be available.

2. **User selects "info"**
   Displays the account information stored in the account table of the local SQLite database via an alert window. Currently it displays the account's username, password, unique account id assigned by Firebase, original account id, and a four-digit link id.

3. **User clicks "LINK" positive action**
   The user enters in the username and link id for the desired account to connect to. Once the account is linked, the firebase listeners are refreshed along with the account's current id.

Unlinking Accounts

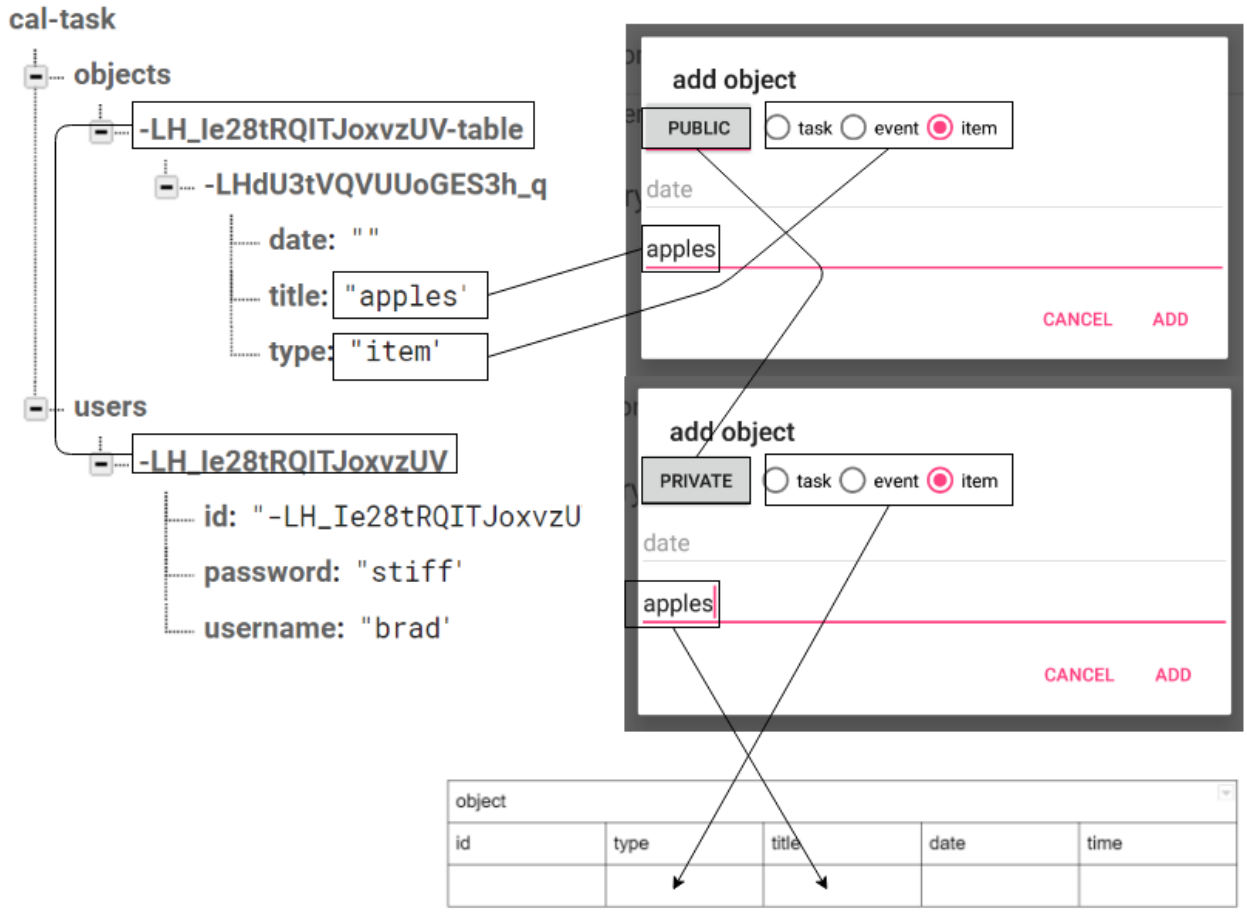


1. **User clicks main menu**
   Same as linking accounts.

2. **User selects “info”**
   Same as linking accounts

3. **User clicks “UNLINK” positive action**
   The user simply clicks unlink, and they listviews will refresh to reflect the account’s original objects in the firebase database. Private tasks are local to the device, so linking and unlinking do not affect those items.

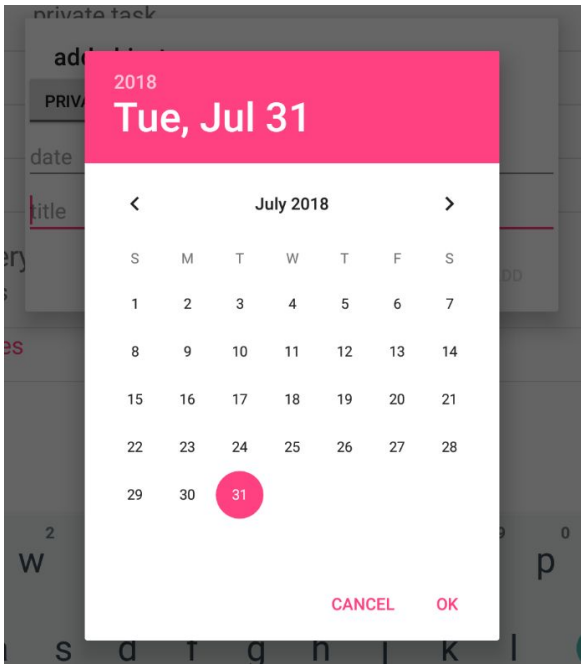
Creating Objects



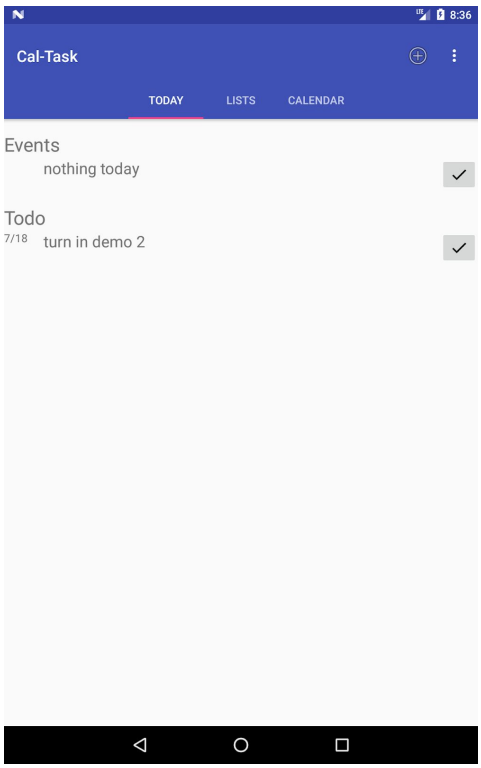
1. **User clicks “+” in main menu**
   This activates an alert window that displays the options for adding an object. Upon opening, the user can only select the public/private toggle button, the radio buttons, or cancel.

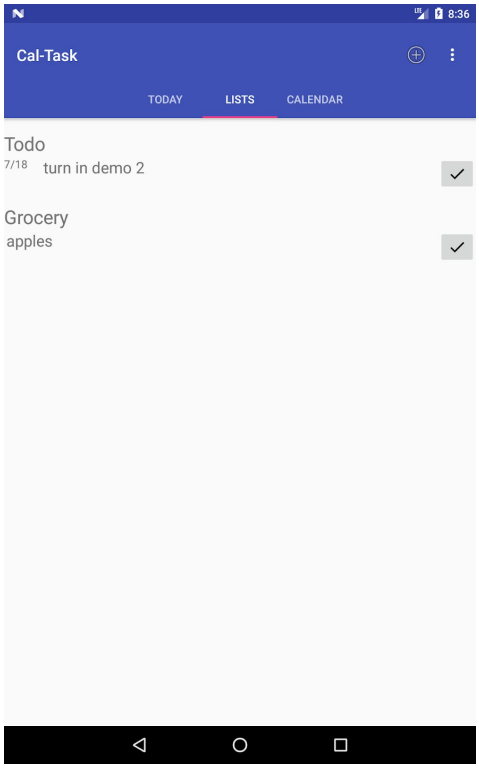
2. **User selects “public” or “private”**
   If the user selects public, the item will be stored in the Firebase database. Selecting private, stores the object in the SQLite database.

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. If the object is a task or an event, clicking the data entry fields will produce a pop-up datepicker object.

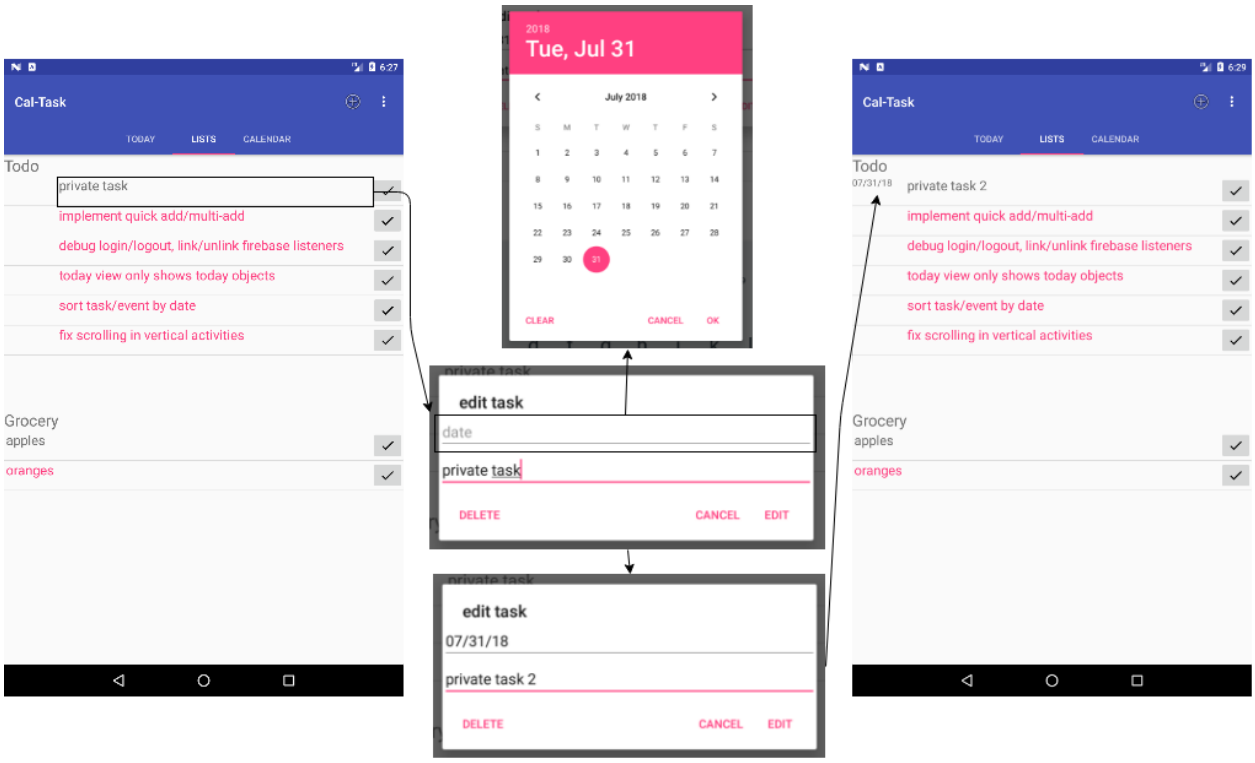


4. **User clicks “ADD” positive action**
   Once the title field is not empty, the “ADD” positive action becomes enabled. Once the item is added to the database, the listViews on the various tabs will update to reflect the database addition.





Editing Objects

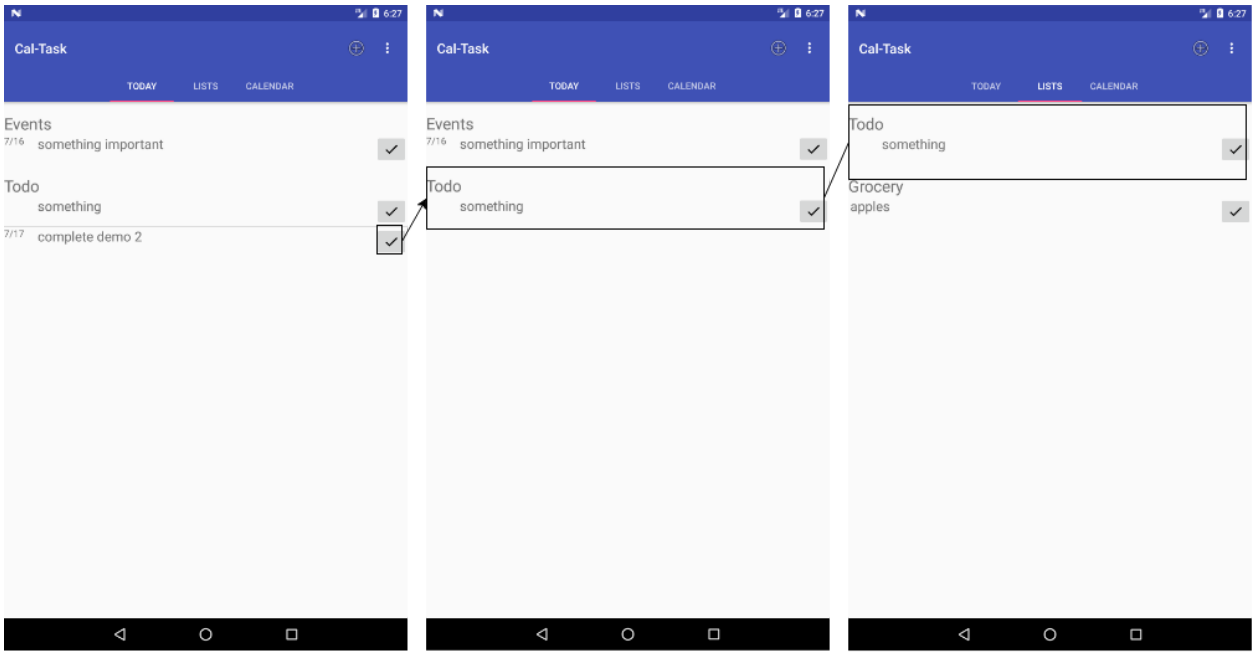


1. **User clicks object's title**
   This activates an alert window with the fields to be edited. The fields will be populated with the information held inside the object.

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

[NOTE] The pink text displays the public items in the listview. Also, the clear option for the date, is only available for tasks since an event can not be created without a date.

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. ViewPager Listener
   https://developer.android.com/reference/android/support/v4/view/ViewPager.OnPageChangeListener

2. Add DatePicker to EditText
   https://stackoverflow.com/questions/14933330/datepicker-how-to-popup-datepicker-when-click-on-edittext

3. Checking for Username in Firebase
   https://stackoverflow.com/questions/47893328/checking-if-a-particular-value-exists-in-the-firebase-database?rq=1

4. Search Firebase by Child Value
   https://stackoverflow.com/questions/39135619/java-firebase-search-by-child-value

5. How to enable/disable AlertDialog Positive button
https://stackoverflow.com/questions/8238952/how-to-disable-enable-dialog-negative-positive-buttons

6. Firebase CRUD Operations
https://www.simplifiedcoding.net/firebase-realtime-database-crud/

7. Display Firebase Child Objects in ListView
https://stackoverflow.com/questions/39558397/display-firebase-child-objects-in-listview

8. Android Fragments
https://developer.android.com/guide/components/fragments

9. Android Menus
https://developer.android.com/guide/topics/ui/menus

10. Firebase Listeners
https://firebase.google.com/docs/database/admin/retrieve-data

11. Fix Orientation Change Application Crash
https://stackoverflow.com/questions/16806856/orientation-change-crash-application

12. Fixed Fragment Vertical Scroll
https://developer.android.com/guide/topics/ui/layout/linear