Lit Commit: A Study-Help App

Alan Koruth

University of North Carolina  
801 Coker Dr

Chapel Hill, NC 27517  
+1 (919) 909-6839

koruth@live.unc.edu

Franz Dominno

University of North Carolina  
700 Bolinwood Dr, Apt 40A  
Chapel Hill, NC 27514  
+1 (919) 225-7172

dominno@live.unc.edu

Victor Cui

University of North Carolina  
4257 Laurel Ridge Circle  
Weston FL, 33331  
+1 (954) 646-9764

victorvi@live.unc.edu

**ABSTRACT**

We introduce our app Lit Commit, which is designed to help a person increase their productivity while studying. We plan to achieve this by delivering three different activities that would assist someone in efficiently planning out their time. This paper will overview the app itself and expands on each individual activity that contributes the app's purpose.

**Keywords**

Pomodoro technique; database queries; calendar

# INTRODUCTION

One common issue among students is that many don't know how to study properly. Some have trouble paying attention long enough to complete their assignment or to study the required material for an exam. Some have trouble being productive the entire time they sit down to study. Some have an issue prioritizing work and what needs to get done. Whatever the issue is, Lit Commit, a study-help app, aims to increase a user's productivity while studying.

The app does this specifically by having the user write down all the things they need to do. Once they have established their "to-do list" the user can move on to one of two things. The user is free to look over their tasks or to begin studying.

As a user explores the app, they will encounter 7 different layouts. Upon opening the app, they will be greeted by a welcome screen. The layout for the MainActivity class fulfills this role by showing a user a welcome message, our app icon, and two buttons. The second button allows a user to login with their Facebook account. This functionality serves no purpose within this app particularly as of yet. The Facebook SDK was used in a previous version of the project when the app had a different idea in mind. The idea had to be scrapped when functionality failed initial expectations. Implementation of the Facebook SDK was carried over to the current project and allows the app to be expanded on. The first button overlooks this issue and allows a user to continue on to use the app without having to login to their Facebook account. Both of these button results in the starting of the Main2Activity class and a finish of the MainActivity class

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee..

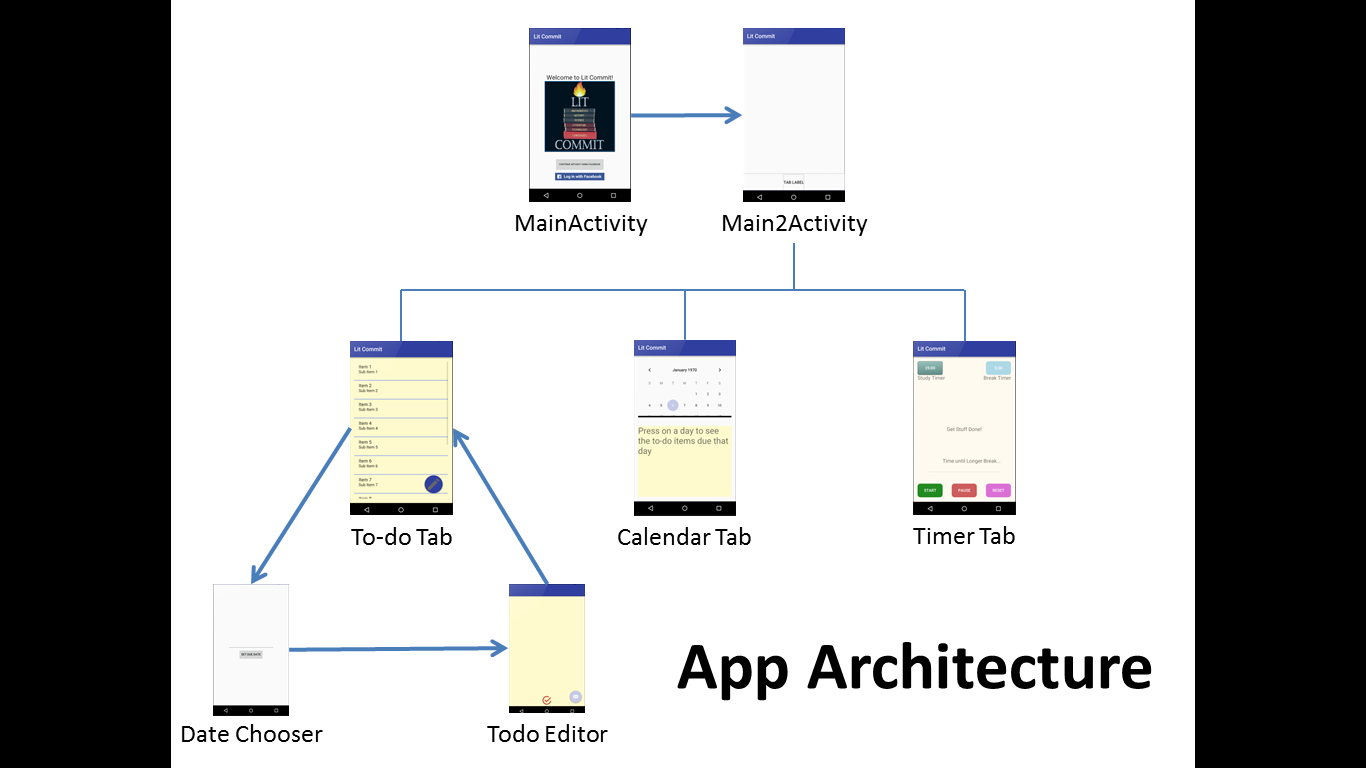
Copyright 2016 ACM 1-58113-000-0/00/0010 …$15.00.

DOI: http://dx.doi.org/10.1145/12345.67890

The Main2Activity class has a tabbed layout containing three different tabs. These tabs are the to-do tab, the calendar tab, and the timer tab. Once the intent has been passed from the MainActivity class, the user first encounters the to-do tab. The primary function for this tab is for users to have a place to input their tasks and serves as a storage place for all of the tasks they have made. The second tab is the calendar tab. The main functionality of the calendar tab is to allow the user to select any particular day and find all the tasks that they have created that are due on that day they selected. The third tab contains the timer tab. This tab allows a user to set timers for themselves for periods of studying and for taking a break.

Standard progression through the app is shown in Figure 1.1 and is as follows:

1. Welcome Screen
2. To-do Tab
3. Calendar Tab
4. Timer Tab



**Figure 1.1** App Architecture

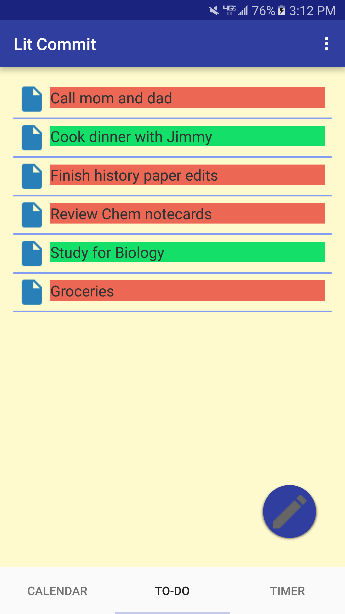
To start using the app a user will start at the to-do tab and create entries for their "to-do list." This is done with three different layouts. The initial layout, "to-do tab," shows the user their current list and allows them to press a button to create a new task. The second layout makes the user insert a due date for that task before allowing them to move on and title the task. The third layout allows them to title the task and give it a short description if they desire. Once a user has made all their entries, they will go to the calendar tab and see all the relevant tasks due on any particular day by choosing different dates in the calendar. Once a user has gotten perspective on what they need to get done, the user moves on to the timer tab to being studying.

# TO-DO TAB

The to-do tab of the Lit Commit app provides another use for those looking to maximize their productivity whilst studying. Organization is a crucial value to have as a student. If you look at some of the leaders in the business world, one thing you will find in common is that their organizational skills are top notch. More than likely, they will have a calendar, detailed to the day, outlining the tasks for the day. This way, time is not lost figuring out what needs to be done or where to allocate resources. Now, scale that down to the needs of a college student. A typical one will have class, assignments, papers, club meetings, team practices, social activities, and many more to balance. It can all be very stressful for an 18 year old to manage all this. The best way to manage this stress, is to organize oneself; list out the things that need to be done and when. Since one day varies so much from the next for a student and commitments come up unexpectedly, a calendar can be of less use. Instead creating a to-do list the night before a busy day becomes useful. This is where the to-do tab of this app comes into play.

This tab is designed like a to-do list. A user can add a to-do item to the list by pressing the add button in the lower right corner. This brings up the editor mode. The user can then type whatever it is they need and as much as they need. Pressing the back button then saves the item. As more and more items are added the main activity turns into a scrollable list. All items in the list are sorted in chronological order with the oldest items at the bottom. The main activity only shows the first line of each to-do item. This way the user can go into more explanation of what they want in the actual note itself. Pressing on an existing note takes the user to the editor mode once again. They can update the note from here, but cannot delete the note in editor mode. This way if the user accidentally deletes all text and presses back, his to-do item is not gone. Additionally the delete options in the main activity encourage the user to finish an item on the to-do list before throwing it out.

Another feature implemented is the marked read or not characteristic, similar to email. When updating an existing note, and the user should do this only while updating, the user can press a check mark on the bottom of the screen. If the check mark is green, it marks the item finished. If red, the item is still to be finished. This translates to the main activity that all unfinished items are highlighted red while all finished items are highlighted green.



To-Do Database

(Internal Storage)

Main To-do Activity

Editor Activity

**Figure 2.1** To-Do tab main

Delete options are available in the menu of the main activity toolbar. The options are “delete all checked” and “delete list”. The “delete all items” is self-explanatory but the other option deletes all to-do items that have been marked finished, or highlighted green.

## Implementation

The To-do items were tracked using a SQLite database. Android has an integrated SQLite implementation that stores the database internally which made the implementation easier in the sense that an external connection to a database was not needed. Although this did make debugging database operations difficult as the database could not be visually seen. The schema of the database was organized as follows: a field for item id, which was auto incremented, a field for item text, a field for date created using SQLite’s timestamp, and a field for whether the item had been marked read or not.

This entire portion of the project was handled with traditional structure of android app implementing backends. A custom version of SQLite’s OpenHelper was used to create the database schema and the database itself. Then, a content provider was used to provide access to the physical database. The custom content provider used in the to-do tab encapsulated all necessary database operations so that other running code could delegate these operations. All data passed to the content provider or read calls were organized with URI objects. Uri’s were structured so that the default path to the database was stored while the last segment of the path was the parameter with which a record could be identified.

All read/write calls to the database were made from the main to-do activity and the secondary editor activity. A custom implementation of the cursor adapter was used to append all to-do items to the list view. This implementation included the logic to update text when a new note is pressed and to change if the current item, retrieved by a cursor, had been finished or not.

Android apps that use backends could potentially have databases with large amounts of data. It is not wise for these apps to implement database operations on the main thread as it could force the app to crash. With the scale of lit commit, this issue doesn’t occur as data is small but to comply with good practice all database operations were moved from the main thread. The main to-do activity implements the loader class. All insert, delete, and update, calls are handled by the loader on its own asynchronous thread. After the database has been updated, the loader is restarted and the cursor adapter will be swapped with the new data it can iterate through and display in a list view.

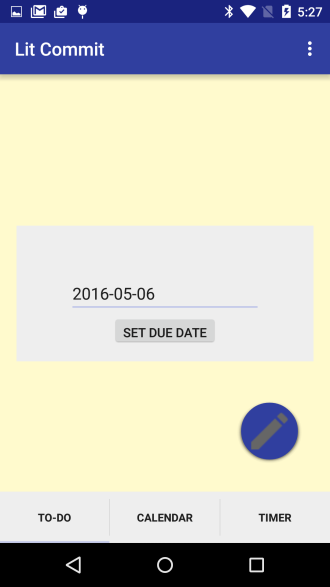
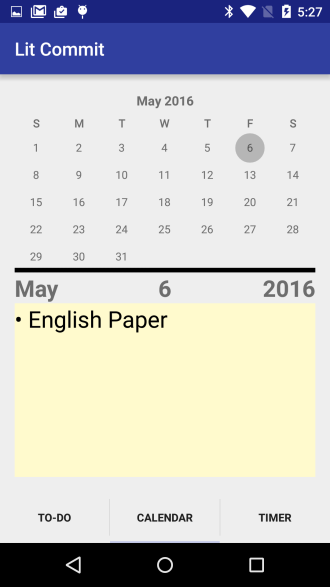
**Figure 2.2** Project Flow

1. **CALENDAR TAB**

When one thinks of a way to efficiently plan out their time, they typically think of setting an agenda of the day-to-day. This usually involves looking at a calendar. Planning out your schedule has proven to be helpful at increasing productivity and efficiency. It all starts off with a to-do list which has already been explained above.

While it is important to keep a to-do list of all the tasks that need to be completed, if one has to too many to go through they may lose perspective on which tasks are more important or pressing. This is where the app's calendar tab comes into play. Building upon what the to-do tab already provides, it minimizes the scope of the tasks that the user is looking at. It achieves this by only showing the relevant tasks that have the same due date that is selected in the calendar. This is accomplished by allowing the user to input a due date to each task. When a new task is made, the due date is stored with it in the database created by the to-do tab.

Once a user switches tabs to the calendar tab, they immediately have the ability to select any day in the calendar by swiping up and down to traverse between different months and tapping on the desired day they want to see if they have tasks due on that day. Those items are taken from the database by conducting a query where all the items have the matching date that was selected. They are then added into a yellow "notepad" placed below the calendar.

1. **Implementation**

To start with implementation, the welcome screen and the tabbed layout had to be made to hold the desired activities diagrammed in the introduction. Franz made an activity that originally housed the login activity of the previous project idea which has now turned into the welcome screen of the current app. This activity retained the Facebook SDK that Victor implemented into it for the previous project idea. Adding on to that Franz added another button to skip Facebook login. To move on to the next activity, Franz made Main2Activity to house the tabbed layout of the project. The tabbed layout is simply a Tab View that is provided as one of the view in the presets in Android Studio. Then from there, the tabs were set within the Main2Activity.java class. Each tab was set by adding a reference to the different java classes that represented each tab. Each of those classes would then set their content layout in it's onCreate() method. Each tab would show what each class set as its content view.

Once setting up the project was finished, the project was passed on to Alan to implement an interactive to-do list that implemented a working database. The calendar tab functions by accessing the items made in the to-do tab. Alan's initial implementation did not include a due date for each to-do item. To solve this, Franz went and modified Alan's code to incorporate a due date. This was done by adding a column to the table that was to be made to house all the to-do items. From there, a user needed a way to enter a due date for each task. Initially Alan's tab had a two-way street between the to-do tab layout and the to-do editor layout. To let a user enter a date, a third layout had to enter the mix. Franz added in a date chooser layout that allowed a user to enter a due date. This is implemented by instead of passing the intent from TodoTab.java to TodoEditor.java, it is passed from TodoTab.java to DateChooser.java. The DateChooser class makes a user enter a date to continue on to make the actual task that is stored in the database. If a user backs out before entering a date, they are returned to the TodoTab class. The DateChooser class also ensures that the user inserts a valid date by testing if the date entered follows the standard date format. Once the date has been entered, the user progresses through the todo tab as before, without the DateChooser activity.

To implement the actual calendar, Franz used the preset calendar view that Android Studio provides. This view has a method called setOnDateChangedListener() that allowed Franz create a new onDateChangeListener to override a method called onSelectedDateChange() to run a chunk of code every time a new day was selected in the calendar. Inside this overridden method is where Franz implemented the functionality of the calendar tab. This is accomplished by generating a valid date string from the parameters of the method. Using this generated string, it searches the database for all todo items that have that matching date. Using a cursor adapter, it iterates through each item that was found and adds them to a vertical linear layout that is held inside a scroll view. When a new day is selected the linear layout is cleared of all views to allow the addition of the items for the new day that was selected.

1. **Querying Issues**

During implementation, difficulty arose when trying to generate a query to receive the items that had a matching due date. Whenever a query was made with a where clause containing the due date generated from the selected day on the calendar it would return zero results. To remedy this issue, Franz had to get all entries in the database and create a new cursor adapter to iterate through each item in the database and check if the value in its due date column matched the date generated. If it passed this check, it would then be added to the linear layout.

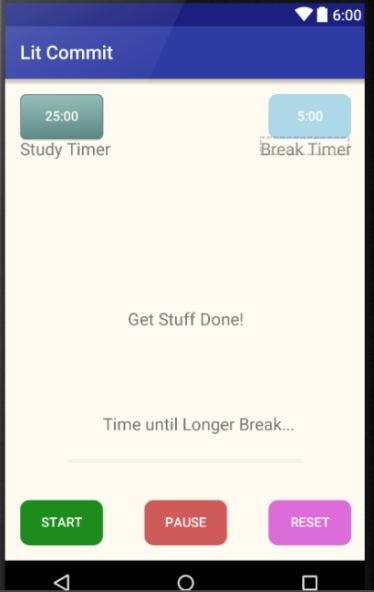
Another issue that arose was generating a valid date. Several modifications had to be made to the parameters that are a part of the onSelectedDateChange method. The month is returned as a number from 0 to 11, with zero corresponding to January and the other months corresponding to the following numbers. This was handled with a switch statement. Each case had a two digit string made for each month returned. The day of the month would also be returned as a number. This proved to be the largest issue of maintaining a valid date format. Integers that were less than 10 had to have a "0" added on as a prefix in the generated date string where the day of the month resided. All dates had to have the format of "yyyy-mm-dd".

1. **TIMER TAB**

The timer tab of lit commit app provides a streamlined way for students to pace themselves while studying. When facing a new or difficult subject, students can feel overwhelmed when they think about how much time they need to truly understand all the material. Oftentimes students will try to cram long study sessions in which they try to study non-stop for five or six hours, but end up having a headache, feeling burned out, and end up playing video games or on their phones updating their Twitter account. Numerous studies have shown that our ability to retain information diminishes after ~25-30 minutes, so it’s best to break up our long study sessions into multiple, smaller sessions [2]. I personally have tried to study for hours on end, only to feel completely burned out after 45 minutes of non-stop studying. Since realizing my problem, I’ve been searching for a way to maximize my productivity while studying, when I discovered the Pomodoro technique.

The Pomodoro technique is a time-management method developed in the late 1980s by esteemed developer, entrepreneur, and author Francesco Cirillo while he was still a college student. This technique will help students power through distractions, hyper-focus, and get things done in short bursts, while taking frequent breaks to take a deep breath and relax. This way, students are extremely productive without feeling overwhelmed. The technique is simple: When faced with an important task or series of tasks, break the work down into short, timed intervals (called “Pomodoros”) spaced out by short breaks. The whole method consists of 5 steps.

1. Choose a task to be accomplished
2. Set the Pomodoro to 25 minutes (this is your study timer)
3. Work intensely on the task until the Pomodoro rings, then put a check on a sheet of paper
4. Take a 5-minute break
5. Every 4 Pomodoros take a longer break (~15-30 minutes)



**Figure 4.1** User interface of Timer Tab

1. **Implementation**

I implemented my version of the Pomodoro timer using two instances of Android’s built-in CountDownTimer class, which schedules a countdown until a specified time in the future. One instance was used to implement the Study Timer countdown button, and another instance was used to implement the Break Timer countdown button. The CountDownTimer class handles the messy business of creating timers on separate threads, thereby not blocking the main thread. When the app first launches and the start button is pressed, an instance of the study timer is created, the study timer button starts counting down from twenty-five minutes, and the text within that button updates every second. When that timer ends, the callback method onFinish() (of the CountDownTimer class) is automatically called. Within that method, I used the RingtoneManager class to play the default notification sound, and the Vibrator class to repeat three vibrations. This notifies the user that it’s time to set aside the books and watch five minutes of mindless television.

Above the start, pause, and reset buttons there’s a progress bar that lets the user know when he’s completed four Pomodoro time intervals & therefore deserves a longer break. At this point, the progress bar is 25% filled up. The text of the button changes to “Start Break.” When the user presses “Start break”, a new instance is created, the break timer button starts counting down from five minutes, and the text within that button updates every second. When the break ends, I again use the RingtoneManager class to play the default notification sound, but this time the Vibrator class plays two longer vibrations, thereby distinguishing the end-of-break notification from the end-of-study one. At this point, one Pomodoro interval has been completed. When four intervals are complete, the progress bar will be completely filled, and the textView above the progress bar will change from “Time until Longer Break…” to “You deserve a longer break!” Now, when the “Start Break” button is pressed, a new instance of a CountDownTimer is created with a ten-minute countdown. To distinguish between the count down timer and the break timer, I simply used a boolean variable.

1. **Issues / Debugging**

One issue I had was at first when the user pressed the start button multiple times, that would start multiple threads and mess up my timer. To solve this, I simply used the Button class’s .setEnabled(false) method on the start button whenever the user pressed the start button once. This blocks the button from responding to touch events. At the end of onFinish(), I call .setEnabled(true) on the start button so the user is able to press start again. Likewise, I call .setEnabled(false) on the pause button and the reset button whenever they’re pressed, and call .setEnabled(true) on the onFinished() method of both buttons.

Another issue was getting the pause button to work properly. At first when I pressed pause then hit start, the timer would restart back at 25 minutes or 5 minutes because I was creating a new instance of the CountDownTimer class. In order to correct this, I added a boolean variable signifying whether the state of either timer was paused. If it’s true, I start a new instance of CountDownTimer with the old value of the old timer saved in an instance variable.