Title:

Project Final Report

Name of application:

Calendar Application

Authors:

Kyle Allen,

Mhealyssah Bustria,

Jasper Cavins,

and Favian Quach

Date:

Friday 10 April 2019

Table of Contents

Section 1. Introduction

We set out to create an application that would allow multiple users to schedule tasks that they needed help organizing. The idea was to allow users to create a profile that would allow them to schedule tasks on and view different layouts that would update with each new task they added. We also hoped to add functionality that would allow multiple users to access one single task.

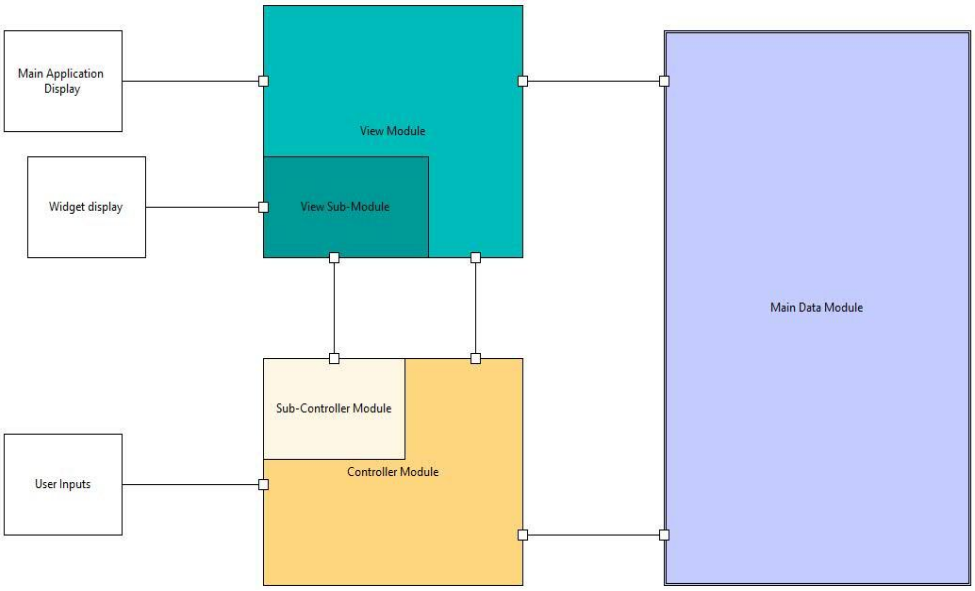
We were required to build the app on a mobile platform.

Section 2. System Functions

We wanted our system to include the following functions of; creating a user, creating a task, changing how to view a task, sharing a task, and adding friends. So far we have implemented every function into our system expect for the ability to share a task with friends. User can on startup create a profile, and then have further ability to change some aspects of that profile. Then they have the option to change between two views, a calendar or a list view, that be the view for their event information. Touching the event on either view will bring up only the due date information and not other information as time ran out before we had the chance to implement it in more detail. User can also on their profile add friends that have also been registered in the app and they will be tied to their profile however, again due to time limitations, there is no way to view or share events created with any friends tied to the user.

Section 3. Architecture

The system uses the Model-View-Controller architecture pattern.



The system has the following main components[3]:

- model: The status of the system.
- view: Draws the appropriate images on the screen.
- controller: Gets user input and keeps track of other non-user-input changes.

We were able to implement this pattern using Android studio (see Section 4.2 for more details). The reason this architecture style was used was because it worked so well for our app. We could have the user inputting data to changing or moving data through the controller which is the application itself and in return it would be either changing or updating the view or it would be sending data to the database for storage if it need to be accessed by the view module again at a different time.

Section 4. Implementation

Section 4.1) Implementation Technologies

Programming Languages	• Java • XML
Frameworks	• Android Studio (IDE)
Software	• Firebase Database/Authorization
Cloud Services	None used

Section 4.2) Implementation Tasks

- **Programming Language: Java**
  Java was used to implement the Model and Controller components of the system. Because of the use of Android Studio Java was a guaranteed must to use.

- **Programming Language: XML**
  XML was used to implement the View component of the system. All UI design needed to be designed in this language.

- **Framework: Android Studio**
  Android Studio was used to was used to compile and run the program as well as a great tool for storing and organizing files as well as

implementing libraries. Android Studio also comes with built in bug test and JUnit support that allowed for easy bug finding.

The Model-View-Controller architecture pattern was implemented using Android Studio[4].

The Model component was implemented using Java classes in Android Studio.

The View component was implemented using Activities and XML files in Android Studio.

The Controller component was implemented using Activities in Android Studio.

- **Software: Firebase**
  Firebase was used to design the database for the system. Because the database deals with a lot of information for the app, Firebase was involved with the Model component of the system. It allowed for use to store data not on the application. Firebase also offers more than just database storage as it allowed for use to create a away to authenticate user and register users on the application as well as securely store all their information.

Section 4.3) Consistency

SRS Consistencies

The following features are consistent with the requirements specifications created in the SRS[2]:

- **The Month View:** A calendar layout displaying a block for each day of the selected month
  - As required: The Month View displays all of the days of one month. The user can navigate to the month of their choice. The user can add an event to the day of their choice.
  - An inconsistency is there is no colored dot displayed on the calendar to indicate that an event exists on that day.
- **To-Do List View:** A layout that lists all events that need to be done.
  - This View was consistent with the SRS.
- **Adding events:** We wanted the user to be able to add events to their calendar.
  - We successfully implement the Events feature. On both the Views, the user can click a button that will add an event, first giving a popup that requires the user to enter information about

the event. Once completed it will be added to the views for the user to see.
    - An inconsistency is that the user can not change the progress status (mark as complete).
- **Multiple users:** the only implementation of this is allowing registered users to add friends.
    - This allows users to add friends through their profile, however it does not follow what we wanted of allowing users to then share events with friends.

SRS Inconsistencies

The following features were requirements specifications created in the SRS, but were not implemented in the system[2]:

- **Year View & Day View**
    - Since implementing Month and List views was already difficult for us, we decided to focus on other tasks instead of spending time figuring out other views.
- **Sharing events**
    - We could not figure out how to allow users to share tasks between their friends.
- **Adding tasks:** We wanted the user to be able to add tasks to their events.
    - We did not implement the Tasks that would be tied to Events, because this would overcomplicate the system.
- **Widget display**
    - There was a lack of knowledge on how to implement this on the phone, but the project owner specified that the widget was not a priority for the app.
    - We originally intended the widget to display a To-Do List, but we already implemented a To-Do List in the app, so a Widget was not necessary.

Architectural Consistencies

The following features are consistent with the architectural design created in the Architectural Document[3]:

- The implementation of the model module exactly follows the original design.
  It is designed for entering and retrieving user data through commands that the controller calls and not allowing direct user manipulation.

- The implementation of the view module exactly follows the original design.
  The View module is the display that the user sees and it has a separate view for the task list "widget"

Architectural Inconsistencies

The following features were architectural designs created in the Architectural Document, but were not implemented in the system[3]:

- The Controller sub module was combined into the main controller module
  - Due to time constraints and ease, we did not separate the sub controller module from the main module.
- We originally planned on having a secondary display that the user could have on their home screen. But we didn't include that and the user only has one display that they can switch to see the different views.
  - We could not figure out how to add this feature, and given more time we could have worked on implementing that feature.

Section 4.4) System Availability

- Source code is available at https://github.com/bustr003/CalendarApplication

- The video demo is available at https://github.com/bustr003/CalendarApplication/blob/master/FinalDemo.mov

- Our final presentation is available at https://github.com/bustr003/CalendarApplication/blob/master/Final%20Presentation.pdf

- Our repository also contains our other documents, such as the requirements documents, presentations, and video demos.

Section 5. Project Management

What each team member has done

Member	**Roles**
Kyle Allen	• database ○ create and manage the database using

	Firebase ○ connect the application to the database ● coding ○ performed many major coding tasks for the project ○ managed the code written by all group members
Mhealyssah Bustria	● documentation ○ managed documents and presentation slides for the project ○ proofreading ● coding ○ implemented "Activities" in Android Studio to get user input (Control) and manage the screen display (View)
Jasper Cavins	● research ○ conducted research to find various methods that could be used for the project ■ conducted research to find out how to effectively use these methods for the project ● coding ○ created and implemented Java classes to be used by the application
Favian Quatch	● architecture ○ managed the design and implementation of the Model-View-Controller architecture pattern ● documentation ○ managed the process of writing documents ● coding ○ assisted with many major coding tasks for the project ○ debugging

Problems and Resolutions

Problem	**How the problem was resolved**
Due to the large number of files in an Android Studio project, an Android Studio project could not be directly uploaded to GitHub.	Uploading a zip file of the Android Studio project to GitHub, and other collaborators having to download and unzip a new zip file each time the code in the repository was updated. This

	made version control more complicated and strenuous.
A null pointer error occurred when trying to use the EventId that was supposed to be read from the database.	We realized that the second function (using the EventId) was being called right after the first function (getting the EventId from the database) was called, but the first function was not done being executed yet. To resolve this issue, we made the program wait until the first function was completed before continuing, by using a while loop.
There were some project-management difficulties as our group evolved throughout the semester. The main issues were trying to figure out everyone's responsibility and getting a matching schedule were everyone could meet and discuss the project.	With effective communication, we were able to work together well. We communicated with each other via text messaging, Zoom meetings (mainly to utilize the screen-sharing, chat, and voice features), online chats, and face-to-face meetings. We were quickly able to learn about each group member's strengths and weaknesses use this knowledge to have better project management within our group.

Section 6. Conclusion

In conclusion, while our app does not do everything we set out to do, it is still a functioning task organizer allowing users to: register and customize their profiles, add events to their available views, check the due date of the events they created, and add friends.

Although all of the members in our group had to learn how to use Java, XML, and Android Studio from scratch, we created a functioning app. Overall, we are proud of the app that we created as amtaeur moblie-app developers.

Section 7. References

[1] "Project Proposal"
Author: Mhealyssah Bustria
https://github.com/bustr003/CalendarApplication/blob/master/Project%20Proposal.pdf

[2] "Application C Requirements Documents"
Authors: Kyle Allen, Favian Quatch
https://github.com/bustr003/CalendarApplication/blob/master/Requirements%20Doc.pdf

[3] "Architectural Document"
Authors: Kyle Allen, Mhealyssah Bustria, Favian Quatch
https://github.com/bustr003/CalendarApplication/blob/master/Assignment%203%20Architecture%20(1).pdf

[4] "Android Architecture Patterns Part 1: Model-View-Controller"
Author: Florina Muntenescu
Date: Nov 1, 2016
https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6