Project Name: OmegaCalendar

December 12th, 2022

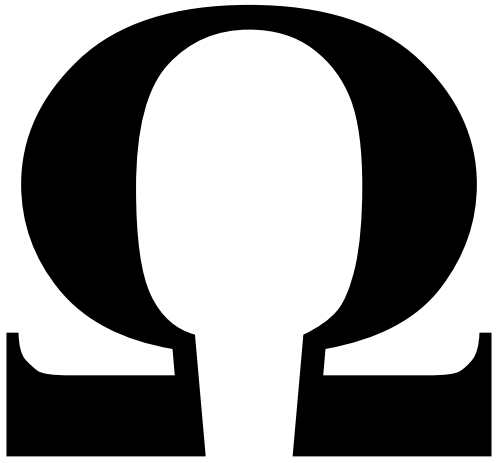
Version 1.0

Team 6

SRS Document For OmegaCalendar project CSCI 150 - Intro to Software Engineering

---

This document serves as a basic reference for usage and handling of the OmegaCalendar App and its internal structure, library usage, and user capabilities/product value.



1. Introduction to Omega Calendar

Product Scope: Omega Calendar is an Android calendar app developed in Kotlin with Discord linkage. Omega calendar allows users to send Discord notifications about their calendar events. Additionally, Omega Calendar is equipped with a custom database. This facilitates ease of use and convenience for the user.

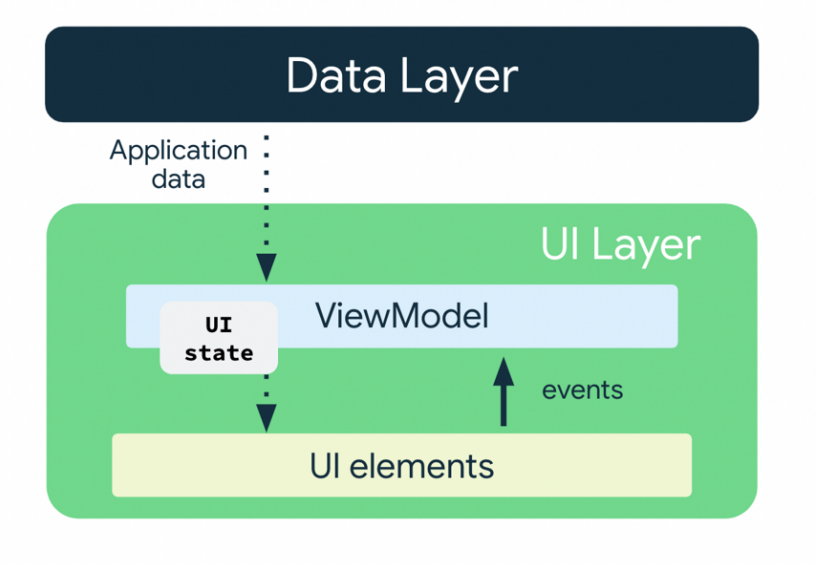
Product Value: OmegaCalendar offers users an extra convenience that other calendars simply do not. The ability to notify Discord servers of events and happenings at the push of a button could prove a valuable tool for organizations and business groups. Additionally, the calendar app itself is easy to use and displays events and information accurately and efficiently which is useful in its own right.

Intended Audience: The intended audience of OmegaCalendar is the average working individual and anyone with events to manage whether they use Discord or not, is able to use the basic functionalities of the calendar. Additionally as mentioned previously it could be a useful tool for organizations that use Discord to coordinate or plan events as announcements could be made easily and readily available for large groups of people to see.

Intended Use: Individuals will use our product to store their daily events and tasks. They can keep a record of what they are going to do and what they have done by inputting events into the calendar. They can then send announcements to Discord servers about their events and will do so.

2. System and Functional Requirements

Basic App Functionality The basic app functionalities including navigation, ui building, and interaction, as well as management of internal processes are handled by jetpack compose. Jetpack compose is an internal software library that supports and facilitates the ease of building android apps and their component systems. The compose UI is what facilitates all of our navigation and information passing. The basic product structure of our app is handled through what is known as a View Model which will be abbreviated as VM. This VM is the core component of our app and holds the current state of the app, information is passed to the view model when an action is made and it will edit the app's state accordingly. There are three primary views within our app, a view is a terminology for a new screen ie in our apps case we have a monthly view which allows the user to see all events for a given month, and we have a day view which allows a user to see specifics times and descriptions for all events of a given day. We also have a final view for adding events. Each of these views are facilitated and swapped between by our view model. Our monthly view is our primary screen and what appears first on opening the app this initialization happens during the Oncreate function using a ViewModel factory to create our component model. Once the model exists it accepts information whether it be in the form of swiping inputs or taps and uses it to facilitate the swapping of screens. The basic navigation can be further understood via the following diagram:



Shown in this figure is the basic data abstraction followed by our app. It can be seen that the user never directly interacts with the data layer rather they interact with UI elements that coexist within the ViewModel and communicate with the Data Layer.

Some primary functions include

- Tapping on any given day will take the user to that day's list of events and their times
- Tapping the add event button will take users to the add event screen where they can input information to create new events
- Swiping back from an internal view (ie day view or add event view) will return the user to the monthly view screen they were on previously
- Tapping on an event while within day view will bring out a popup menu in order for users to access network functions or delete the event

All of this navigation is managed by our view model using jetpack compose and an internal function which can be triggered on input and receives a new screen to pass into.

Database Functionality OmegaCalendar features a fully custom database built using the Room DB library. The important aspect of this library is that it facilitates the ability for persistent storage, meaning the user's events and contents will stay saved locally within the device on construction and destruction of the app (ie opening and closing of the app). Additionally, this database not only facilitates the storage of events but handles all the passing of data from the database to the ViewModel. This is done through the passing of an event-type object. On calling by the ViewModel the database can create event-type objects that hold the information for a given event. It then stores them internally and facilitates the retrieval and passing when called.

Network Protocols the network protocols of this app are facilitated by the RetroFit library and Discord native webhook API. The network functionalities function using simple HTTP requests and parsing of JSON data. When the network function requirements are called they simply accept an object of type event, access its internal data, and place it into JSON format to be passed to the Discord webhook. The retrofit library supports this by allowing the creation of service builders which simply accept all relevant information in order to build a network object which then actually sends the HTTP request. It is required for the user to set up a webhook on their Discord server.

System Info each of these component systems works to make the app function, wherein the ViewModel facilitates communication and navigation. This ViewModel is responsible for accessing our database and placing the information into the UI for the user to see, it also remembers and maintains the general state of the app and facilitates the calling of the network function. This is the internal structure of the app, the UI is built around the usage of the app and all buttons/actions are coded using the compose ViewModel facilitating easy usage and understanding of passing database info and network parameters.

3. External Interface Requirements

System/OS/Hardware Requirements Omega calendar was created in android studio and has only been tested in the emulator of android studio. The primary libraries that were used to create this app were jetpack-compose, retrofit library, Discord webhook API, and room database library. Each of these libraries allowed us to properly structure and create the app as it currently stands. This app can be run currently only through the android native emulator. But if it was placed onto the android app store it would be able to run on any phone with android 10 or above. Preferably though the app would function best on a device running Android 13 as this would be the ideal scenario for all of our libraries. The app itself requires 6.7 MB of space on the user's phone but would recommend 7 MB total in order to preserve storage space for events. While events themselves take up almost no storage space, on the order of bytes it should be the extra 0.3 MB reserved in order to ensure there is never any conflict of storage space and so that the app will always run properly.

Additional External Requirements: The user's phone itself must be equipped with touch screen capabilities it’s primary form of navigation is through touch screen application. Additionally to make use of the network capabilities of this app the user must have stable internet access, and access to a Discord server. A specific Discord account though is **not** required for the usage of this app as it functions through the use of the Discord webhook API described within the introduction. While a Discord account is not required for usage, access to a server equipped with webhooks is required to use the network functionalities.

Communication Interfaces: As mentioned in the previous section this app does use network functionality and communicates with the Discord webhook API. It does this using HTTP passing using the @post event. The information passed using this follows basic JSON format and is as basic as possible.

4. Non - Functional Requirements

Security: OmegaCalendar does not handle sensitive data other than the user's personal events, but in this manner, it is still fairly defensive as all data is stored directly on the user's device so any security risks are mitigated there. Additionally, no extraneous information is passed over the network it is just a simple JSON packet only consisting of the required information that has already been parsed by the app internally so no extra information ever leaves the app's database/the user's phone's internal storage.

Capacity: This app's capacity is discussed in detail in section 3. But to reiterate the app itself should require 7 MB of storage, 6.7 MB of this is the actual app internals and code, but the extra 0.3 MB is to garner additional request space to store events. The events themselves do not take up much space so this is a gross overestimate of how much space would ever be needed by this app but it is such a small amount of storage on a large scale that it becomes a simple ask for only 7 MB of storage for proper usage.

Usability OmegaCalendar was designed to be as simple s possible to use, often times within our UI we have doubled up on navigation features. Due to the simplicity of the app, we wanted to offer users multiple navigation paths based on what they are comfortable with/what is intuitive to users. For example, many of our screens can be left by either a back button or a simple swipe backward and this will send the user back to their previous view. This is a powerful tool as it allows users to select what feels good and intuitive to them. Additionally, navigation is mostly facilitated by tapping the object of interest ie if the user is interested in viewing their daily

view they can simply tap it and it will expand and display all relevant info. This is the case for most usable objects within the app for consistency's sake tapping is the primary form of navigation.

Abbreviations/Jargain:

- VM - View Model, holds state and manages internal components of app
- JSON - Java Script object Notation
- UI - User Interface
- HTTP - Hyper Text Transfer Protocol

End of Official Documentation