Section 1 - Project Description

1.1 Project
OmegaCalendar

1.2 Description
OmegaCalendar is an android app coded using Kotlin language. It functions as a calendar application to store events and their times. Additionally, the app is integrated with Discord webhook functionalities to send Discord announcements to servers using the Discord webhook API.

Contents

Section 2 - Overview

2.1 Purpose

This section will describe the basic scope of the application as well as system requirements necessary to run the application.

2.2 Scope

OmegaCalendar is an Android calendar app with Discord linkage. OmegaCalendar allows users to send Discord notifications about their calendar events. Additionally, OmegaCalendar is equipped with a custom database. This facilitates ease of use and convenience for the user.

2.3 Requirements

OmegaCalendar 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.

2.3.1 Estimates

#	Description	Hrs. Est.
1	**General Usage and Understanding of Kotlin**	10
2	**General Usage and Understanding of libraries**	12
3	**Building of Day View**	20
4	**Building of Weekly View**	35
5	**Navigational Tolls Building**	15
6	**Building and Linkage of database**	30
7	**Linkage of Views**	10
8	**Network Functions**	15
9	**TOTAL**:	149

2.3.2 Traceability Matrix

SRS Requirement	SDD Module
View Model	5.2
Network Protocol	5.3

Section 3 - System Architecture

Overall system architecture is best described as a communication path and state manager. Wherein we have our primary ViewModel (VM) object which handles all state and message passing operations between different objects and screens, as well as most database pulls. This ViewModel is coded in Kotlin using jetpack compose library. It stores information, displays our screens, swaps through screens, and handles user navigation. Additionally, the database architecture is done using RoomDb library and is handled by creating event-type objects which store basic information regarding the event.

Section 4 - Data Dictionary

Brief description of each element in this module or a link to an actual data dictionary

(template of a database table description)

Table

Field	Notes	Type
VM	System Manager	Kotlin Object
Daily View	Displays events for a given day	activity
Monthly View	Displays days and event count for all days in a month	activity
Event	Database Object	object
Description	The description in event.description	Var string
Start Time	Time event starts in event object	Var int
End Time	Time vent ends in event object	Var int

Section 5 - Software Domain Design

5.1 Software Application Domain Chart

Below follows a list of software use cases

- User opening app for the first time
- User adding event
- User deleting event
- User checking events for a given day
- User sending notification to Discord
- User returning to the home screen (month view)

5.2 Software Application Domain

View Model exists as an object within out Kotlin code and manages the state of all internal affairs of the application. It does so by pulling up activities and placing them onto the screen as well as calling and returning values from necessary database functions.

5.2.1 Monthly View

We have two major view screens that act as user interfaces, these are controlled entirely by the view model. Our default screen ie our monthly view is the home screen for the app, and it is the first screen users see when the app is opened

5.2.2 Daily View

Second of our major view screens this displays when a user taps any given day from the monthly screen, the ViewModel will then navigate the user to the daily view screen by pulling the day view activity and inserting into it all corresponding event type objects from the database that happen on that day.

5.2.2.1 Event type object

The event type object is what holds all necessary information for any given event. Each event object represents a distinct and unique event that the user has added. The database facilitates the storage and retrieval of these objects. Once an object is passed containing all the necessary info the info becomes accessible to the application and necessary components.

5.3 Software Application Domain

The basic network protocol of the app functions with simple HTTP requests. Once the user accesses an event from the day view described in 5.2.2 then they can tap on a given event for the option to send a Discord message of the event to a Discord server.

5.3.1 Domain X

Calling the network function created requires the passing of an event-type object. The necessary members of this object are then extracted and parsed and placed into a JSON format using var strings.

5.3.1.1 HTTP @POST operation

The parsed information is then placed into an HTTP post operation and forwarded to the Discord webhook URL to be placed as a message on the server.

Section 6 – Data Design

The data stored within our database consists of all event information. They are stored as event-type objects which are a custom class built to house the description, start, and end times for a given event. The custom database stores this information persistently.

6.1 Persistent/Static Data

Data is stored in the internal storage of the phone

6.1.1 Dataset

Event Description
Start Time
End Time

6.1.2 Static Data

None

6.1.3 Persisted data

All events stored in the phone are persistent data that exists beyond construction and destruction of the application.

6.2 Transient/Dynamic Data

None

6.3 External Interface Data

No external interface data is logged other than instant commands which come from tapping or swiping. This data is not stored in any way instead it is used to activate functions if a necessary step is taken. Ie if a user taps a given day the ViewModel will change to a daily view.

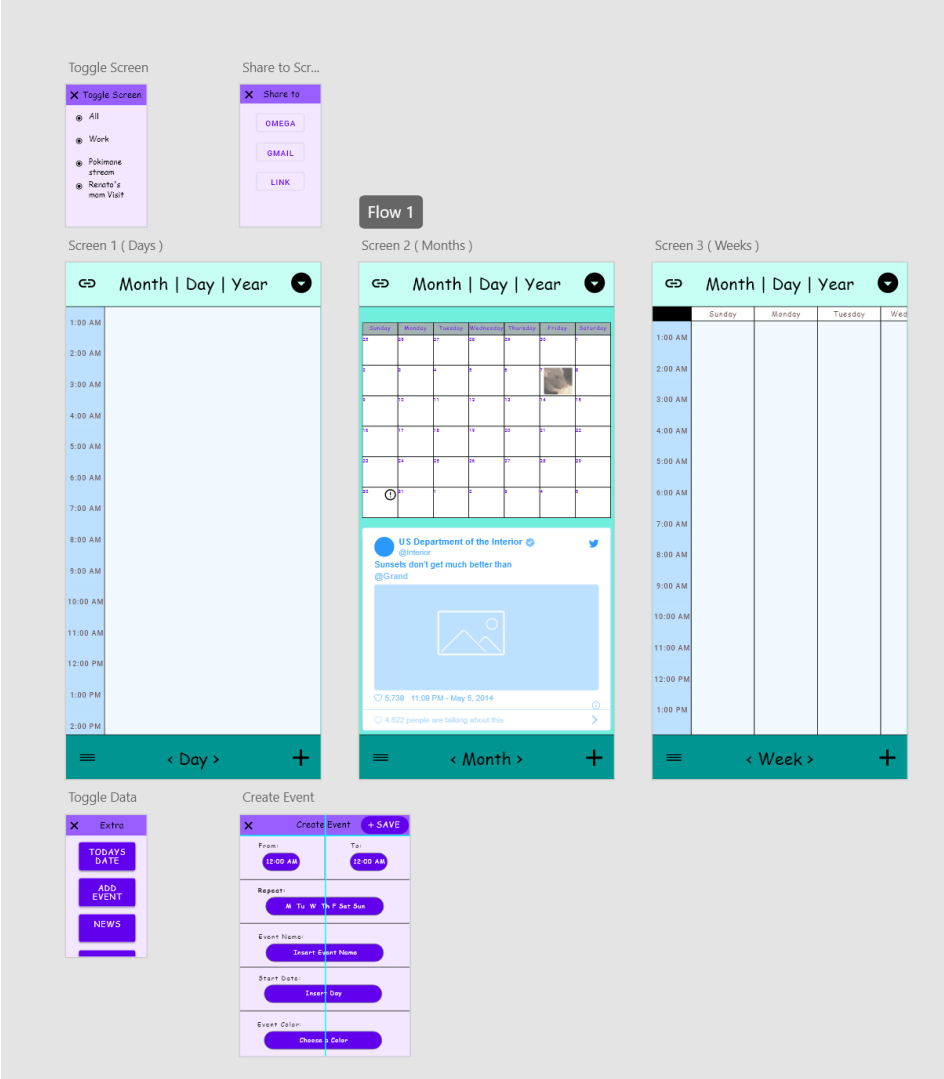
6.4 Transformation of Data

The ViewModel handles all data and information passing so most data remains internal to the ViewModel. Rather than data passing between different segments the ViewModel retains the app;'s current state and then calls different views which can then access and pull information from the current state in order to generate and display the proper information on the screen.

Section 7 - User Interface Design

7.1 User Interface Design Overview

This is the first concept design for the user interface that had created



7.2 User Interface Navigation Flow

For this section there will be abbreviations used for each name wherein

- DV = Daily View
- MV = Month View
- VM = Viewmodel
- DB = DataBase

Upon opening the app the user is opened to the monthly screen, and it displays the current month. The user can then navigate to different months or years using the navigational buttons on the screen. This changes the data displayed on the screen but not the view itself as it still uses the MV activity it simply swaps the information based on internal functions to calculate the date.

If the user is within the MV and taps on a given day then the ViewModel will open the DV activity this gives users the ability to see all the events scheduled for the selected day. Tapping on any of the events gives the user the option to delete the event, post a Discord notification or dismiss the menu. The user can swipe backward to navigate back to the MV screen.

At any point, the user can tap the plus sign in order to enter the add event screen which of course allows users to add events to their calendar. The user can leave this screen by swiping backward and it will return them to the screen they were on previously. Once the user adds the event an event object is created in the DB.

Section 8 - Other Interfaces

This application uses the Discord workbook API which accepts HTTP @post requests with JSON formatting in order to send information to Discord servers

8.1 Interface X

The application sends post requests to the webhook URL. If the post fails then the app will retry 1 extra time before quitting the attempt.