# Capstone Pitch: Dailies App

California State University, Fullerton

Jay Vang

CPSC 491

Fall 2023

Github Link: https://github.com/JVng36/Dailies_Reminder_Mobile_App

## Proposal/Abstract

This capstone pitch will be a mobile application that can track and remind the user of their inputted tasks. It will also serve as a learning experience for mobile development.

Table of Contents

## Introduction

Dailies will serve to be a free application that can hold the user’s tasks and reminders. The goal is to be a digital planner that is remote and virtual. The purpose of this document is to detail the development of the Dailies app. This document will provide the software architecture, UI/UX designs, specifications, modeling techniques, metrics,etc. that is necessary for the development and success of the project.

## Project Plan

### 490

This project was created in 491, so there aren't any plans from 490. However, it is a modified and different version from 491. In 490, the goal was to create a mobile application, and that is still the goal now, just that the concept is different. So, some things from 490 were done for this project such as metrics, requirements, specifications, tools, etc.

### 491 First Half

This project will need to finish the technical document. Then it needs to start building the application.

### 491 Second Half

This project will continue development of the application and finish this document. The project should at least be functional with much of the documentation finished and set. There should also be a powerpoint presentation ready to present the application, what has been achieved, what needs to be fixed and worked on, as well as what is to come in the future for this project.

### Beyond 491

There are plans for continuing this project past 491 as there is still much to be done and it was originally a project meant for learning android mobile development. If this app's development goes well, it will likely go into the Google Play Store.

# Metrics

This document will check requirements to determine the progress of the project.

**Functional Requirements:**

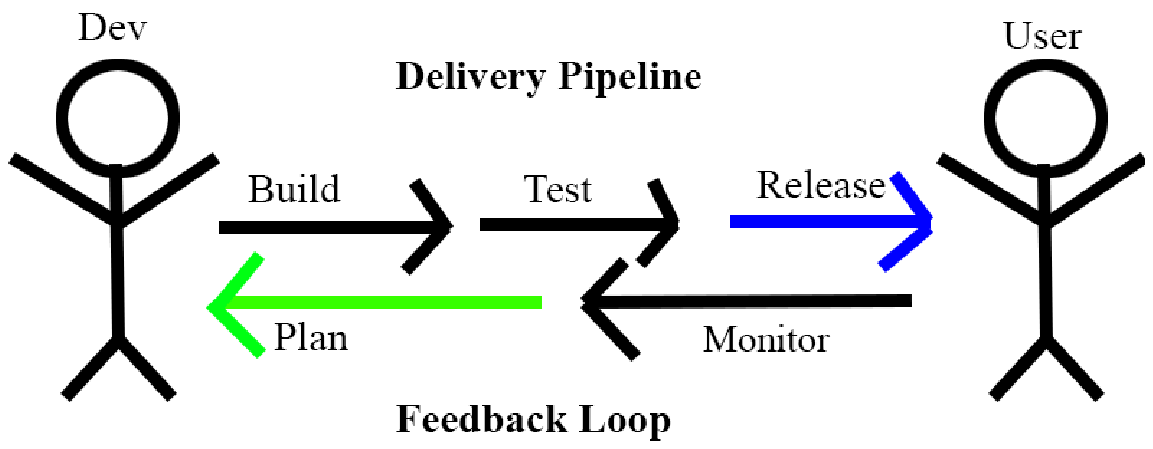
1. Create tasks/reminders
    a. Due dates added
2. Edit or delete tasks/reminders
3. Give the user a notification
4. Allow the user to check off a task/reminder
5. Have an expired bin for the tasks/reminders that have not been done
6. Give the user options on the kind of tasks/reminders
7. User has options for overall functions
    a. Dark/light mode
    b. Choose to use the expired bin or not
    c. Layout
8. Calendar view

## Operational and Support Plan

This mobile application will be used by users locally. There will be no operators involved. Admins such as the developer will handle maintenance and management of the project, making sure the application is backed up and available. The developer will handle security, making sure that the source code cannot be accessed by external malicious individuals through GitHub and personal access tokens.

This mobile application is offline and stores its data locally. User's inputted data will be stored in a database on their phone. Should a user lose their system, it may not be possible for an admin to retrieve their information for them. The application is created in Android Studio and will be using its tools listed later in this document to monitor performance, build, test, and develop the application.

In the event of a bug or issue, users are encouraged to inform the developer of them. Currently, this can be done so with GitHub or the developer's email. This is a diagram showing this application's DevOps model.



## Maintenance Plan

This project will be using Github for version control. Maintenance will most likely continue for this project even beyond the course as one of this project's goals is to learn mobile development. As this is a relatively simple mobile app, there may come to a point where continued maintenance may not be as needed, therefore ceasing continued maintenance.

When a new patch is released, there will be a small description of the new patch changes. The new patch description should look something similar to:

| New Patch 2.0 |
|---|
| 1. New bug/error fixes |
| 2. New Feature.. <br> a. Further explanation or image of new feature |
| 3. New precautionary information that may be useful/important to warn about to users |
| 4. Anything else regarding the gains and losses of going to a new patch |

Since the application is still very new, under development, and rapidly changing, there are no current changelogs. However, once a stable release has been finalized, these changelogs will go inside the application and the Github for this project. The first stable release will be considered patch 1.0. For bugs and small features, the application will stay in version 1.xx, where xx are the subversion numbers. Once the application has received a major feature, upgrade, or change in core development, the application may instead be updated to N.xx, where N is the version number.

If there is an update, it will most likely be the preferred and best version to run. So, there are no plans for different versions that are not considered upgrades. However, since this application is still heavy in development, there is currently a dark mode theme available on the github branch. In such cases, there may be different branches and versions of the app. However, these are only meant for testing, and eventually will be implemented and merged into one single update.

## Technical Sections

### Requirements

Phone: Android OS

Space Required: At least 100 MBs

Permission and Notifications enabled on Phone

### Development Requirements

OS: **Android**

Language: **Kotlin,** XML

Tools: **Android Studio**, Github

Libraries: **SQLite**

Android Framework Classes: Broadcast Receiver, AppCompatActivity.

This mobile application will run on Android devices and it will be coded in Kotlin using Android Studio. Android Studio uses XML to define the layout and structure of its UI, and Kotlin to define the logic, behavior, and event-handling. Therefore, this application will do the same. It will be using SQLite to store data that the user inputs in so that it can show the user to them at a later time.

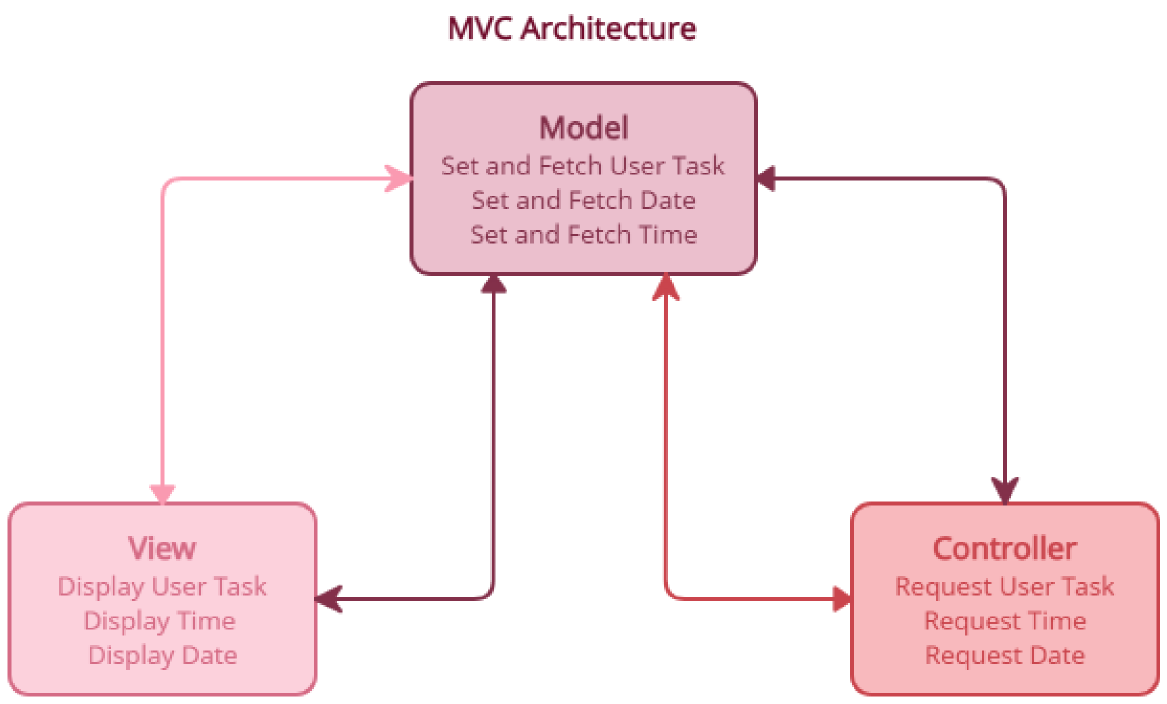
Using Android's built-in Broadcast Receiver library, it will be able to alarm the user when a task is due. AppCompatActivity is used to provide backwards compatibility allowing some new features to be used on older devices. This will allow development for newer devices that can also work with older devices.

SRS Outline for Requirements Specifications

An SRS (Software Requirements Specifications) outline will detail the requirements and specifications of the application. The SRS is a comprehensive description of what software will do, how it will be expected to perform, and how it will fulfill the needs of its users (Gerhard Kruger and Charles Lane 2023).

1. Introduction
    a. The purpose of this outline is to document the requirements, functionalities, and specifications of the application.
    b. The intended audience are people who like to write down their plans.
    c. The intended use for this application is to be a digital planner that can keep track of tasks and remind the user.
    d. The scope of this product is to be a functional application that can be used by anyone, be a good learning experience, and be a proof of knowledge.
2. Overall Description
    a. The user will need to be able to add, edit, and delete tasks. They will also be reminded if they choose so.
    b. This project will be using Android development frameworks. It is going to be made in Android Studio.
3. System Features and Requirements
    a. Functional and System requirements will need the application to store, and take in information from the user. The information needs to be able to be categorized, sorted, edited, and deleted.

b. External Interface Requirements will need the application to have a simple and minimalist view keeping things concise.
c. Nonfunctional Requirements will want the application to have a variety of tweaks and options to the application such as custom reminder/notifications and color of the app’s UI.

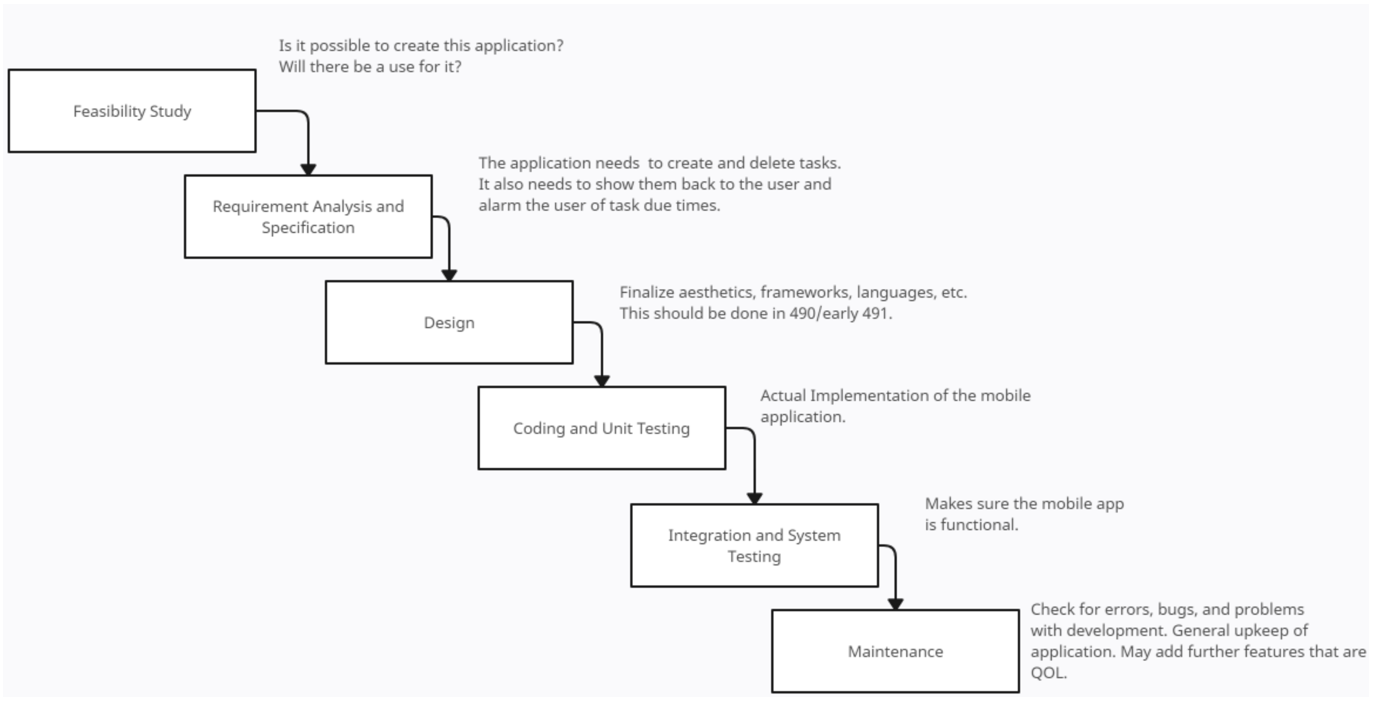


Architecture

The MVC (Model View Controller) architecture splits an application into three parts, the model, the view, and the controller. The model handles all the data-related logic and data itself. The view handles the UI logic, and how and when the data is displayed. The controller handles the business requests and incoming requests (Mishra 2021).

This application will use MVC architecture as many mobile apps do. The model in this application will be the database and database manager classes. The view will be the activities, fragments, and XML layouts. And the controller will be the classes that receive input like buttons, text,fields, and listeners for UI elements.

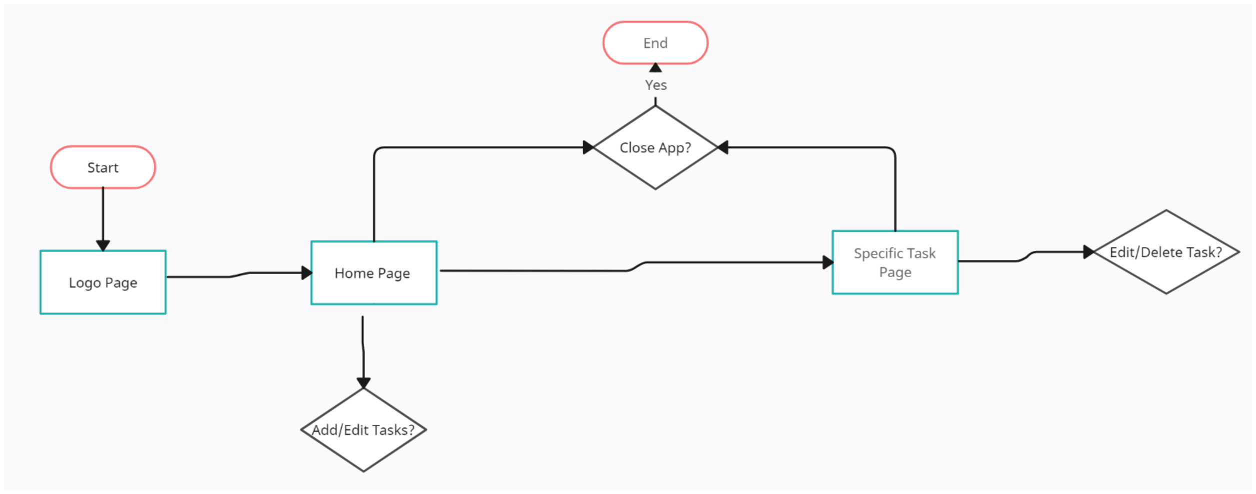
## Waterfall Software Development Life Cycle


Is it possible to create this application?
Will there be a use for it?
Feasibility Study
Requirement Analysis and Specification
The application needs to create and delete tasks.
It also needs to show them back to the user and alarm the user of task due times.
Design
Finalize aesthetics, frameworks, languages, etc.
This should be done in 490/early 491.
Coding and Unit Testing
Actual Implementation of the mobile application.
Integration and System Testing
Makes sure the mobile app is functional.
Maintenance
Check for errors, bugs, and problems with development. General upkeep of application. May add further features that are QOL.

The intended SDLC (Software Development Life Cycle) is the waterfall model. The waterfall model is a linear and sequential SDLC model that splits the life cycle into specific phases known as requirements gathering and analysis, design, implementation, testing, deployment, and maintenance (Pal 2023). The waterfall model is chosen here because it follows the structure of the class.

This project started in 491. In 490, this project had a completely different idea but was still a mobile app. The project builds upon the previous project a bit, but has adopted a lot of it into its own in 491. The waterfall model was chosen because an important aspect of these projects is the documentation which plays a big role in the waterfall SDLC. Its sequential nature also goes along well with the structure of the classes of 490 and 491. A disadvantage of this model is the limited feedback and stakeholder investment which is very much mitigated as this project is a class project that does not currently involve many users and outsiders. It can be tricky to go back during a waterfall SDLC since it is linear and sequential, but the classes 490 and 491 allowed this project to go back and change its identity.

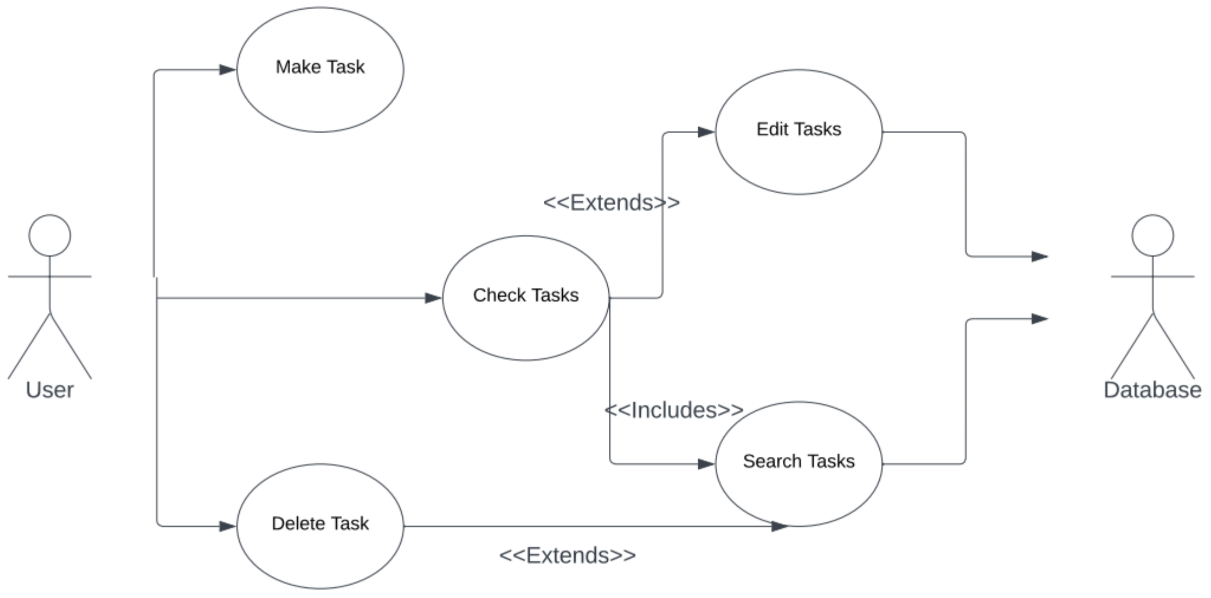
## UX Design



A user flow diagram is an important part of UX design and is used to display an overview of how users can navigate, or show the actual steps that a user will have to follow to do a certain task within an application (Athuraliya 2022).

A user flow diagram was chosen to demonstrate the navigation flow of the application. The user flow diagram allows for easy representation of the portability and accessibility of the application. Users simply open the application and their tasks should be displayed on the homepage. They can then add a task if they want to. They can edit if needed and exit almost whenever they no longer need the application.

UML Use Case Diagram

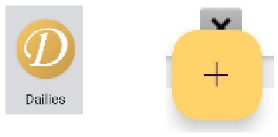


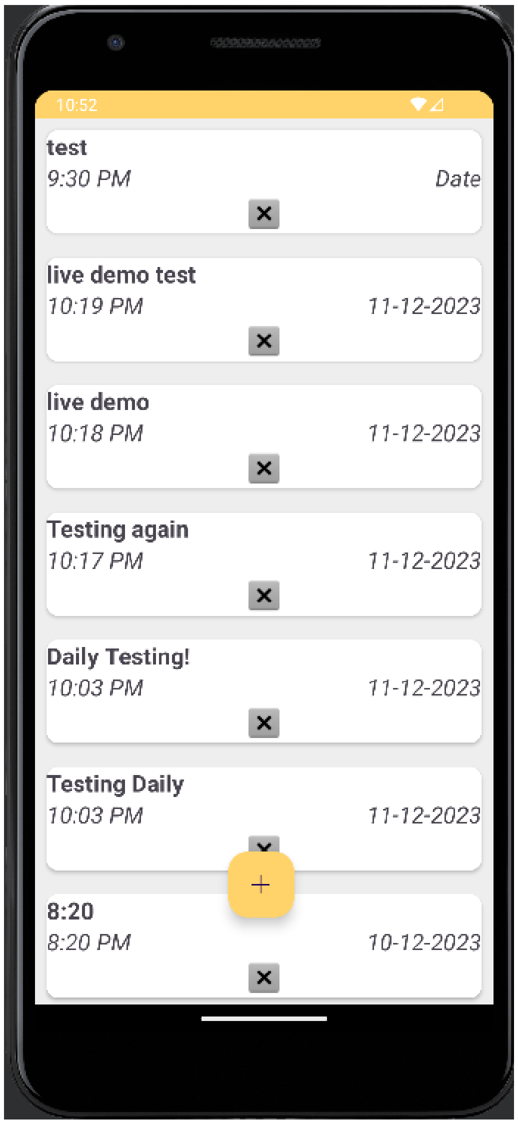
A use case diagram is a behavioral and structural diagram that can model an application's functions and actions through its cases and actors to help display the interactions and functionalities of a system. (Visual Paradigm 2021).

A use-case diagram is chosen for representing the application's behavior. This diagram was chosen since it can provide a high-level view of the application. Starting from the user, they can make or delete a task. This starts the application and has it search and request data from the database. This application is stored locally, so it will mainly communicate with the database stored within the phone.

## Aesthetic

The application will have a simple interface. Its logo will also have a very simple design. The default application should be primarily white and light colors. Its current color theme has yellow accents. In the future, there may be options for different colors, layouts, and a dark mode.This application will try to stray away from bright and unique colors because the goal is to keep everything simple and concise. Here are examples of its color scheme.





test
9:30 PM Date

live demo test
10:19 PM 11-12-2023

live demo
10:18 PM 11-12-2023

Testing again
10:17 PM 11-12-2023

Daily Testing!
10:03 PM 11-12-2023

Testing Daily
10:03 PM 11-12-2023

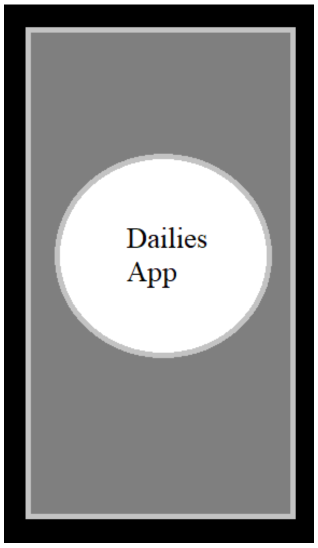
8:20
8:20 PM 10-12-2023

## Personas

Jack is a university student with a part-time job. On top of this, he also tries going to the gym. He wants to try to be as productive as possible. In order to do so, he needs to make sure he does things on time and in an orderly manner. This application will remind him of what he needs to keep track of, what he needs to do, and by when. It will alert him when it is time to do a specific thing he marked to do.

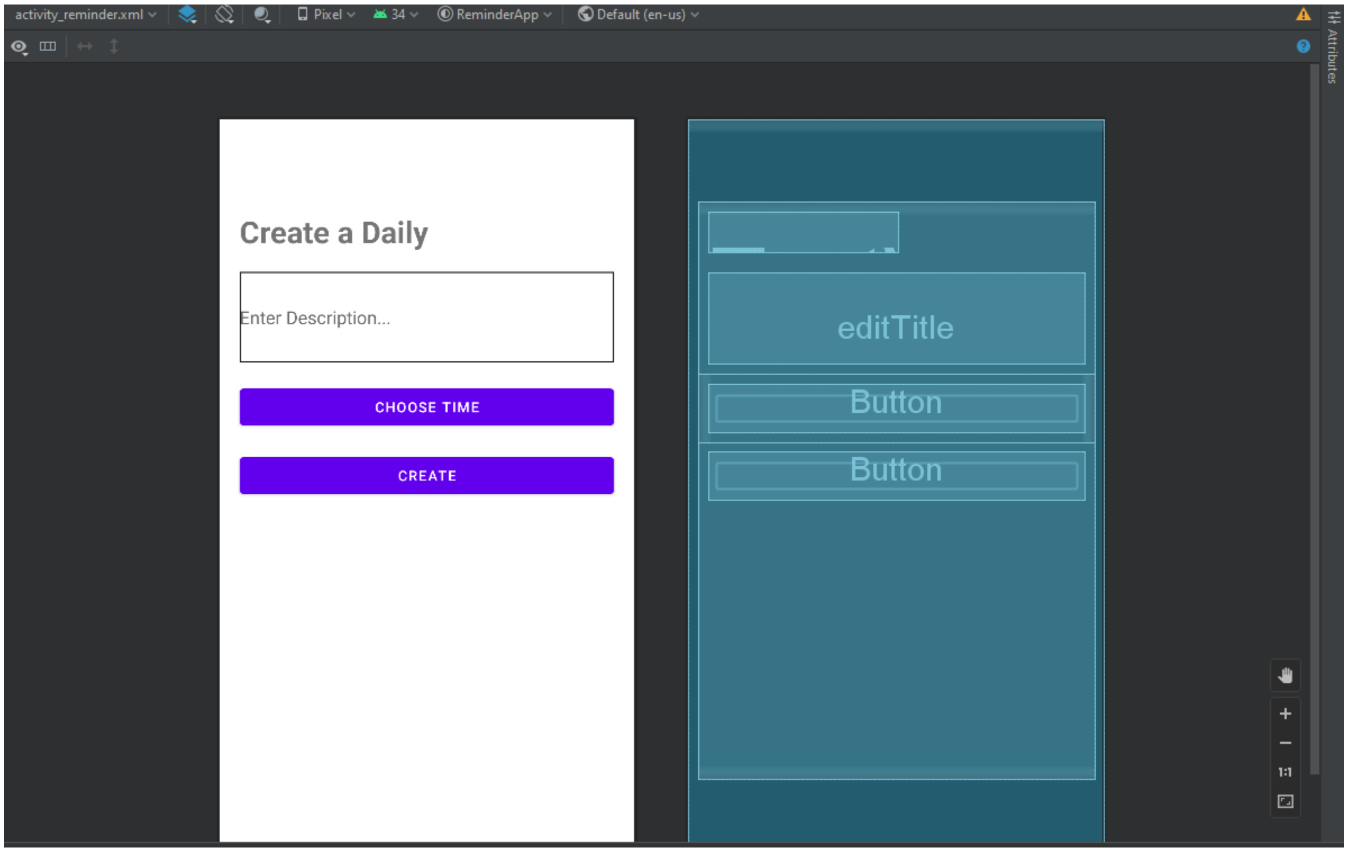
Michelle does not like to carry around books. She already carries around a handful of items. She tries to keep things as digital and remote as possible. This application will help her essentially keep a planner around her so long as she has a phone with her.

## Prototyping

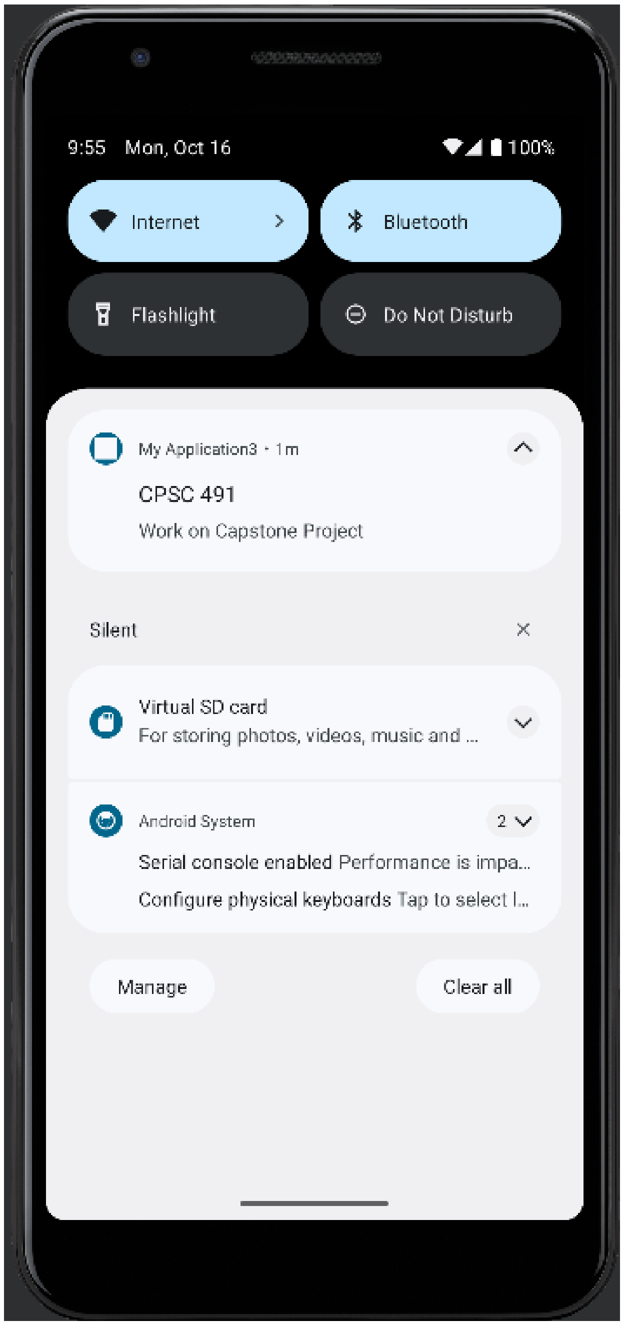


Before the application runs, it should show the logo for a couple of brief moments similar to this.

Example of creating a daily



Example of receiving a notification (Opening the app in Android Studio simulating Pixel 3a)



9:55 Mon, Oct 16 100%

My Application3 • 1m

CPSC 491

Work on Capstone Project

Silent

Virtual SD card

For storing photos, videos, music and ...

Android System 2

Serial console enabled Performance is impa...

Configure physical keyboards Tap to select l...

## Implementation Methods and Coding Standards

General safe and good coding standards will be applied. These include but are not limited to:

1. Object Oriented
2. Efficient, clean coding
3. Proper indentation
4. Comments for appropriate coding sections
5. Maintainable and scalable code
6. Appropriate naming conventions for readability and spacing
7. Automate repetitive tasks
8. Avoid deep nesting
9. Any other coding standard that is good for maintainability, scalability, and efficiency

Linters are tools that can analyze code and find errors, bugs, issues with style, etc. A linter will be used to double-check the code.

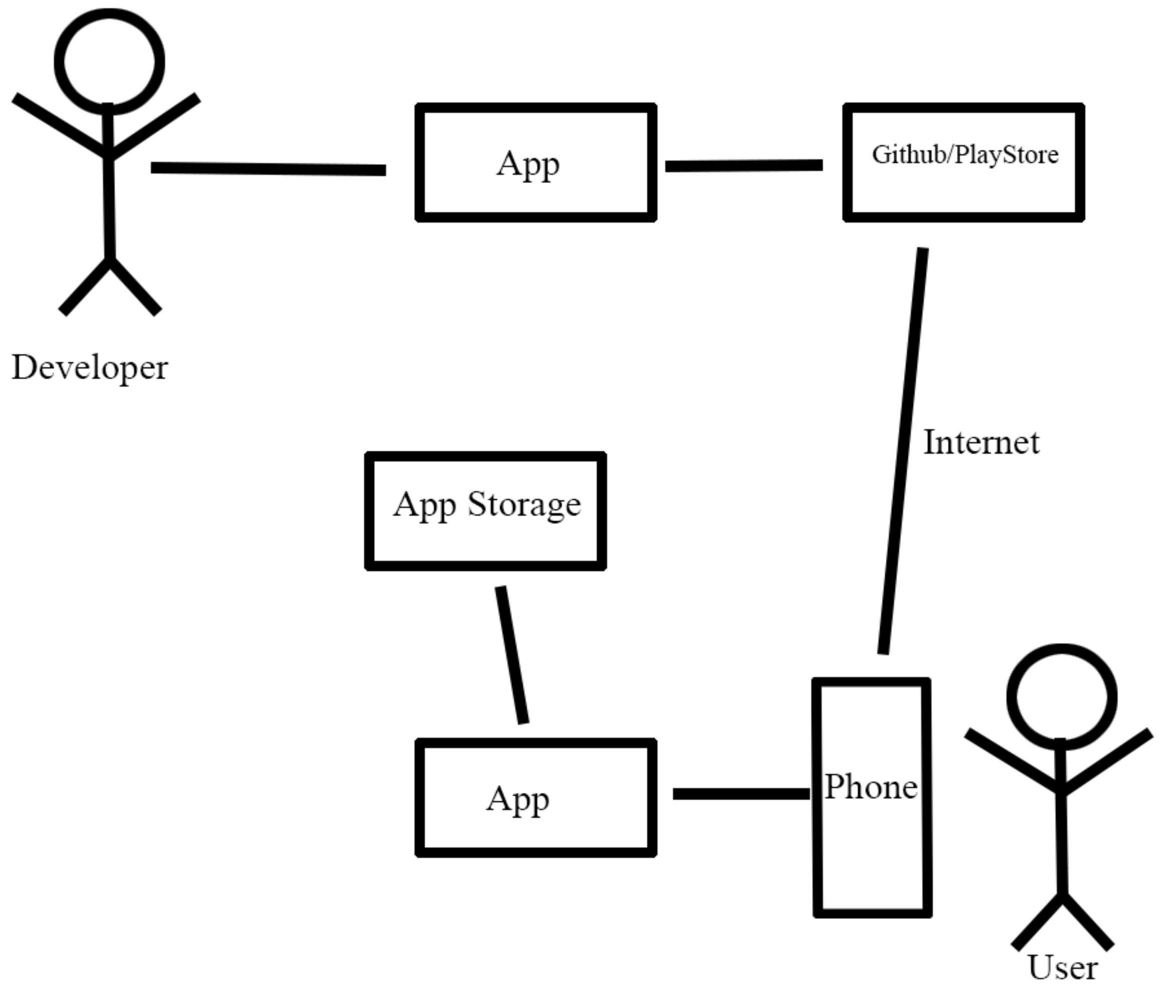
In Android Studio, a lint will be used by using the command in terminal: gradlew lint

## Github

Github Link: https://github.com/JVng36/Dailies_Reminder_Mobile_App

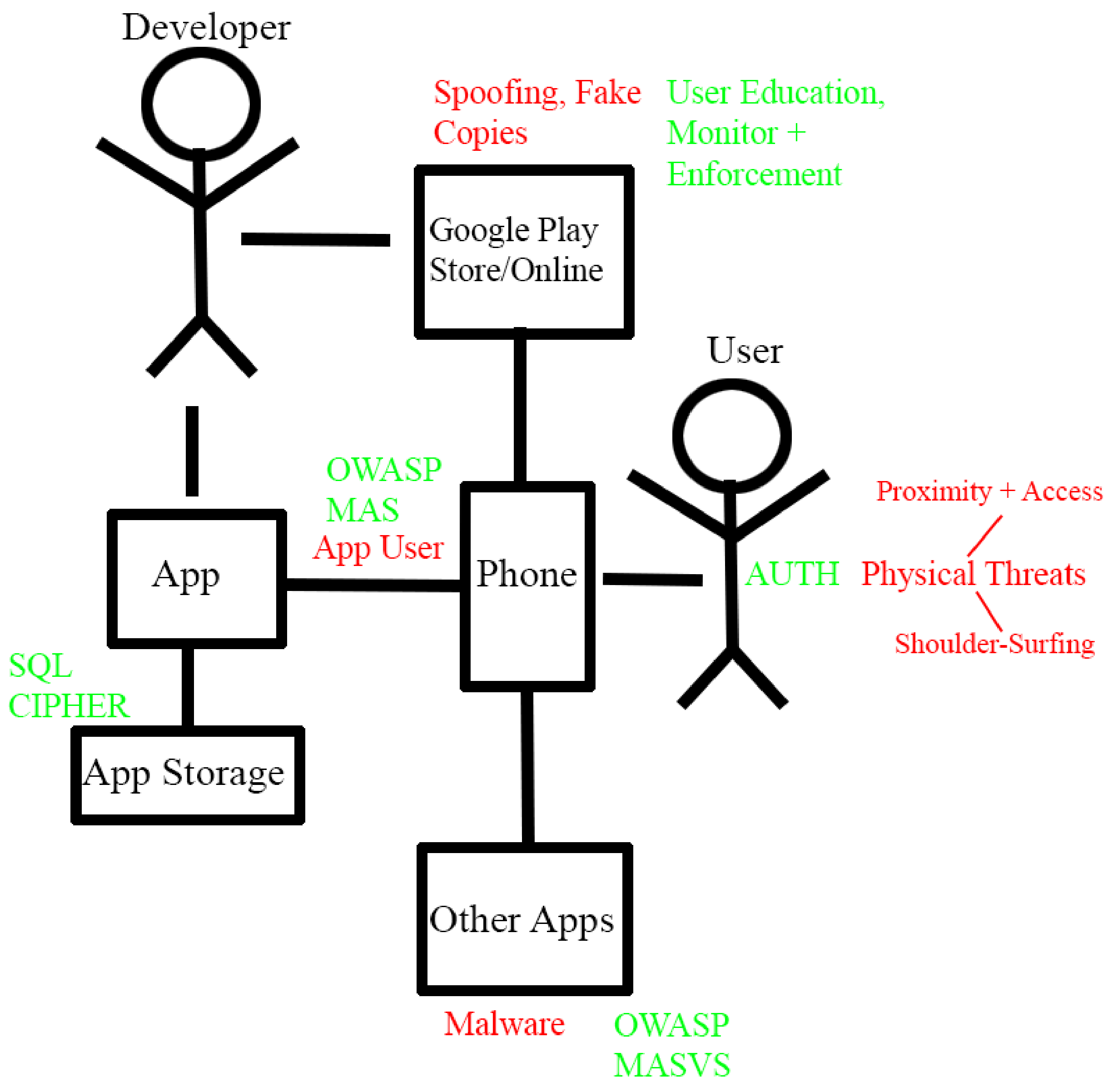
Github will be used for version control. The source code will be available on GitHub. Branches will be used for specific implementations. New Versions of the application will be pushed onto the main branch. Any other information that may be important regarding the application will be available on GitHub.

## Network Analysis



In this diagram, the developer uploads the app onto Github. The user can then download it from Github using the internet and install it on their phone. Once the app is installed, the app will be able to perform its functionality on their device in the app's storage. This app does not make any calls to the network directly. Since the app is currently intended for local and standalone use, there is not much of a network analysis. However, should the application's development continue further, it will most likely be put in the playstore.

Threat Model



This is a mobile application that stores data in a SQLite database locally in the phone's storage. Despite this, there are still some vulnerabilities. This app will use the Open Worldwide Application Security Project (OWASP) Mobile Application Security (MAS) as a basis and guideline to help aid in security. OWASP MAS is a standard that provides checklists, guides, techniques, test cases, and information regarding vulnerabilities ("OWASP Mobile Application

Security" 2023) . It contains three parts, the OWASP MASVS which is the Mobile Application Security Verification Standard, the MASTG which is the Mobile Application Security Testing Guide, and an OWASP MAS checklist.

The first threats are the physical threats. These are threats such as losing a phone, someone else accessing the user's phone, or just someone shoulder-surfing which is looking over the user's shoulders for information. This cannot be fully prevented, but having authentication can help a user in the event that they lose their phone. The next threats are the application users. They can look for sensitive data within the storage or can look for logic vulnerabilities and perform sql injections. Other applications residing within the same phone are viable threats. If they are malware, they will most likely try to access data from the application that can contain the user's personal data. SQLCipher can help by encrypting the SQLite database. Application users and malware can be mitigated with OWASP MAS and OWASP MASVS following their standards and procedures to make more secure code. For more information regarding the OWASP MAS, refer to the Security Tests section.

For dealing with spoofing and fake software, there are many ways, but the way that this project will deal with it for now is user education, monitoring, and enforcement. This is done by making sure the users know to only download from a reputable source such as the original github, and personally making sure there are no other copies out on the internet. Until the app grows and faces bigger threats, these measures will stay the same.

## Open Source and 3rd Party Components Used

This project was done with no prior mobile app development experience or knowledge. So some online resources were utilized and observed to learn how to create a mobile app in Android. Some code, logic, and examples provided here have been modified and translated to fit this mobile app.

1. Official Android Documentation
    a. https://developer.android.com/reference/org/w3c/dom/Document
2. Official Android Development Guides
    a. https://developer.android.com/guide
3. Data-Flair Android Tutorials and Project Guides
    a. https://data-flair.training/blogs/android-studio-tutorial/
    b. https://data-flair.training/blogs/android-broadcast-receiver/
    c. https://data-flair.training/blogs/android-task-reminder-app/
    d. https://data-flair.training/blogs/realm-vs-sqlite/
4. Dev.to Guide
    a. https://dev.to/blazebrain/building-a-reminder-app-with-local-notifications-using-workmanager-api-385f

Some of these links have been referenced APA style in the reference pages.

## Video Demonstration

Google Drive Link:

https://drive.google.com/file/d/1EYDQv5XDIsB4QIFeVgMIb9hF0lwbf6oh/view?usp=sharing

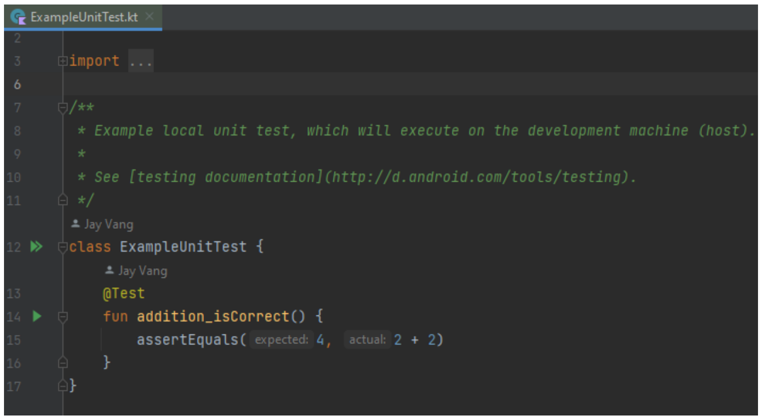
Youtube Mirror Link: https://youtu.be/TI4drLiuaac

Additionally, the video has been converted into a gif file and is in the GitHub repository should the video links go down.

Test Plans and Results

## Unit Testing

Unit testing makes sure that parts of the software are working correctly, specifically the smallest testable code. Some of the common unit testing challenges are dependency management, android-specific components, and asynchronous operations. (Kumar 2023). This app will do unit testing within Android Studio. Android Studio has a directory already built in for unit testing.



ExampleUnitTest.kt

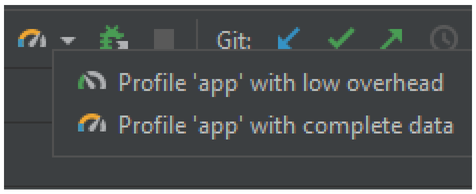
```
import ...

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
Jay Vang
class ExampleUnitTest {
    Jay Vang
    @Test
    fun addition_isCorrect() {
        assertEquals( expected: 4,  actual: 2 + 2)
    }
}
```

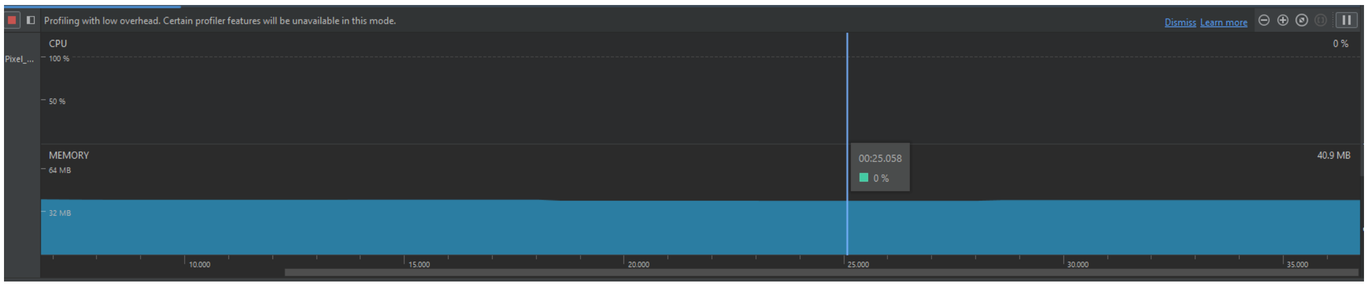
Specific parts of the Android components that will be tested are the Android-specific classes and their components such as activity, broadcast receiver, UI testing, and android-specific behaviors. In activity testing, behavior verification will be done. ActivityScenario, which is a part of AndroidX Test library, will be used to simulate and test the device's activity states. In broadcast receiver testing, there will be tests on the Context and Intent, and the receiver's behavior will be examined. In UI testing, UI elements will be interacted with, and UI testing frameworks will be used to validate UI behavior such as UIAutomator or Roboelectric. Android-specific behaviors will test permissions, resources, and configuration changes.

## Performance Tests

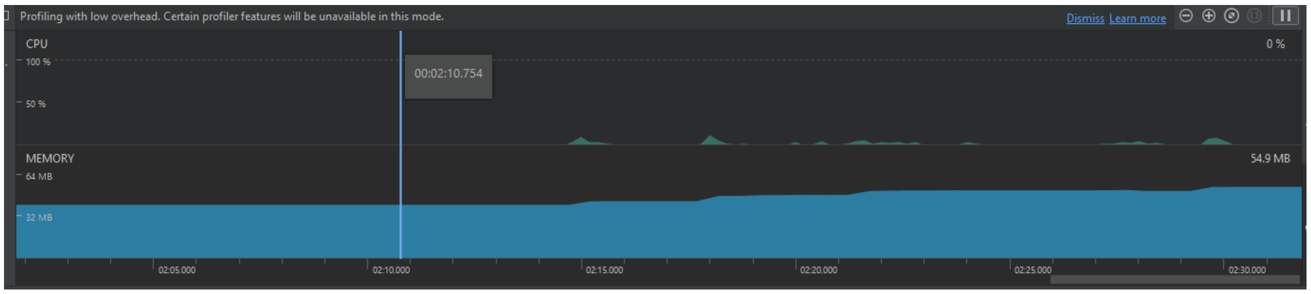
Android Studio has built in performance tests that can profile CPU, memory, and energy levels. These tests will be used to measure how the app is consuming resources. They will make sure the app does not unnecessarily be using too much resources for tasks that do not need it.


Git:
Profile 'app' with low overhead
Profile 'app' with complete data

Simulation 1: Running a low overhead profile while the app is open without doing anything.

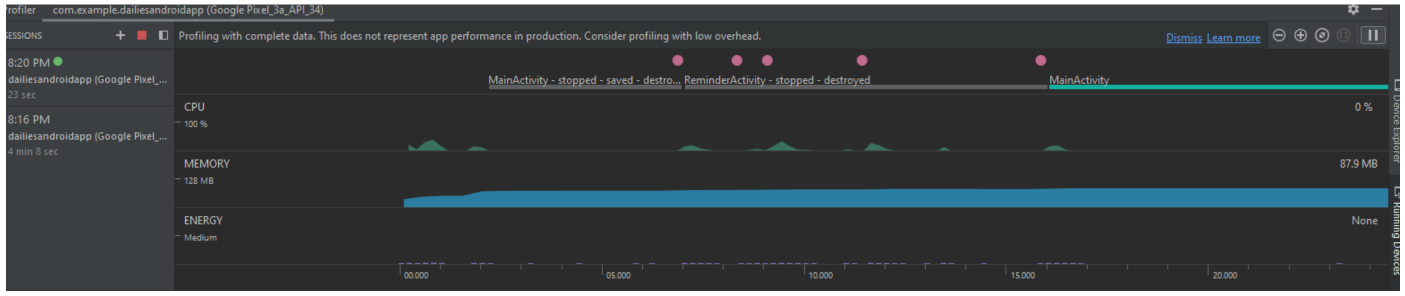

Profiling with low overhead. Certain profiler features will be unavailable in this mode.

Simulation 2: When performing actions within the app such as creating a task, clicking, and typing. Slight CPU usage is needed.


Profiling with low overhead. Certain profiler features will be unavailable in this mode.

These are healthy levels with expected behavior and shows the app is not using too many resources when it shouldn't.

Simulation 3: Running a complete data profile, using the various components of the app and staying idle on MainActivity.


rofiler
com.example.dailiesandroidapp (Google Pixel_3a_API_34)
ESSIONS
8:20 PM
dailiesandroidapp (Google Pixel_...
23 sec
8:16 PM
dailiesandroidapp (Google Pixel_...
4 min 8 sec
Dismiss
Learn more
MainActivity - stopped - saved - destro...
ReminderActivity - stopped - destroyed
MainActivity
CPU
100 %
0 %
MEMORY
128 MB
87.9 MB
ENERGY
Medium
None
00.000
05.000
10.000
15.000
20.000

Profiling with complete data. This does not represent app performance in production. Consider profiling with low overhead.

The app seems healthy as it is only using CPU, memory, and energy when ReminderActivity is run which means the “+” button was clicked and a task was added. Once those actions are finished, the simulation sits on MainActivity and only uses some memory and energy to keep itself up. MainActivity is where the tasks are listed and is the home page view, so there shouldn’t be much resources being used if it is only displaying static data back to the user.

## Security Tests

Security tests have not yet been implemented or tested on the app. However, as stated in the threat model, the OWASP MAS will be used.

In particular, this application should go over these parts of the OWASP MAS:

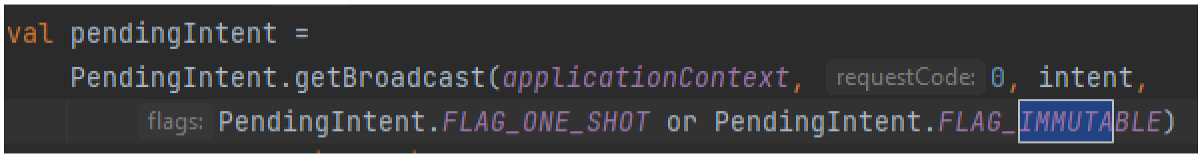
1. App permissions
2. Vulnerable implementation of pendingintent
3. Testing local data
4. Testing device-access-security-policy
5. Prevent leakage of sensitive data
6. Determine whether sensitive data is shared with third parties via embedded services and notifications
7. Finding and testing sensitive data in the keyboard cache, UI, auto-generated screenshots

There are many more topics to go over regarding the app's security, but these are a few of the notable ones. Following the OWASP MAS will require code analysis and testing app behavior. OWASP MAS provides documentation and information on code analysis. The OWASP MASTG will be used as a guideline and reference book for security testing this app.

## Bugs

**Bug 1**: The application will crash once a notification starts.

**Fix**: This bug was fixed by adjusting the code. It seems there was something on Google's end where instead of giving a warning error, it would just crash the application.



```
val pendingIntent =
    PendingIntent.getBroadcast(applicationContext, requestCode: 0, intent,
        flags: PendingIntent.FLAG_ONE_SHOT or PendingIntent.FLAG_IMMUTABLE)
```

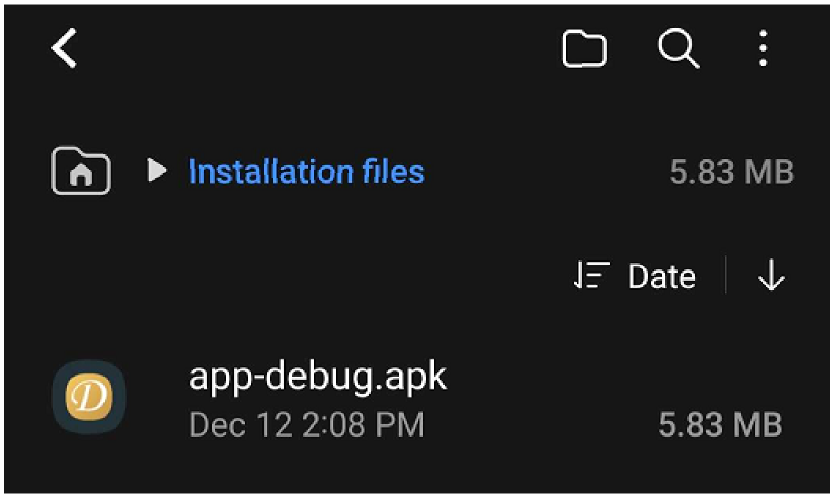
The pendingIntent originally was not set as IMMUTABLE. This issue seems to have stem from a platform security change by Google.

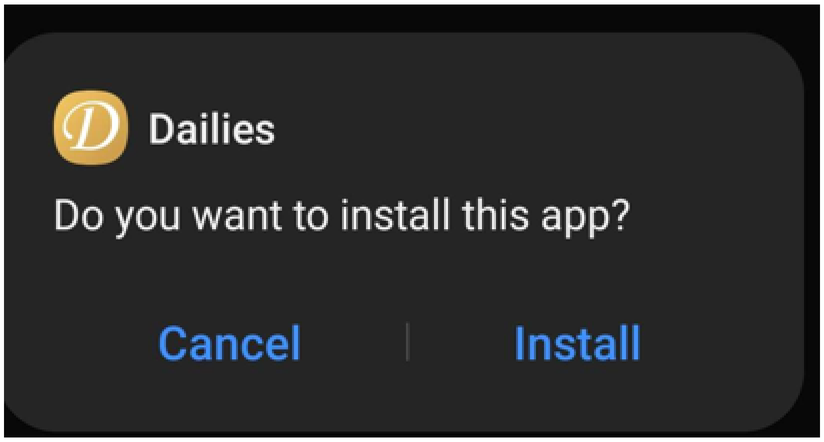
**Bug 2**: Some app functionality seems to be different depending on the API used. Pixel 3a with API 34 plays a sound but pixel 3a with API 33 does not play a sound. Further investigation will need to be done for this bug. It could also not be a bug but rather something set up with the phone instead such as permission to play sounds.

**Bug 3:** Slight delay in receiving alarms. There seems to be a slight delay in receiving alarms sometimes. This could be due to milliseconds as the app only takes in hours and minutes. It could be something with the code and the broadcast receiver. Further investigation will be needed.

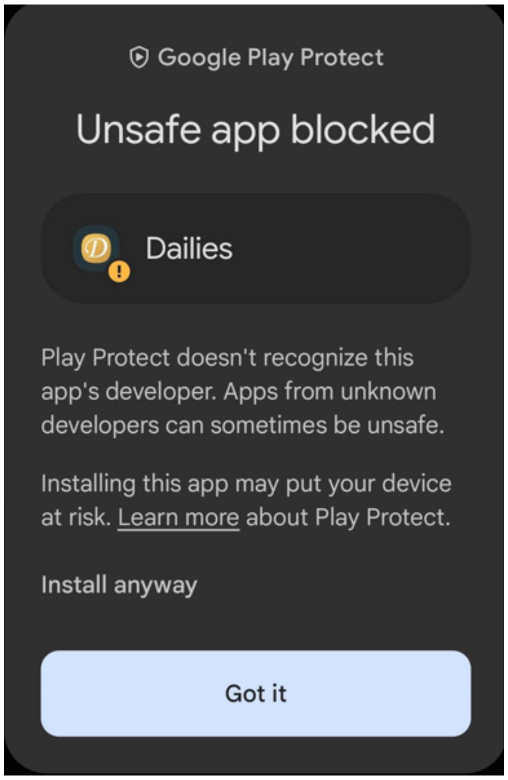
## Installation and Setup Guides

### Installation via Android

1. Download the apk
    a. The apk file can be built using Android Studio and the source code. It will also be available on GitHub.
2. Run and install on phone
    a. Will need to use a file manager program to locate and open the apk file
    b. 

    c. 

3. The user installing will most likely need to give permissions to their phone to allow installation of apk files. Since this app is currently not on the Google Play Store, this is standard procedure as it helps Android users not install malicious software. This step may soon be easier if the app makes it into the Google Play Store.

    a. 

    Google Play Protect

    Unsafe app blocked

    Dailies

    Play Protect doesn't recognize this app's developer. Apps from unknown developers can sometimes be unsafe.

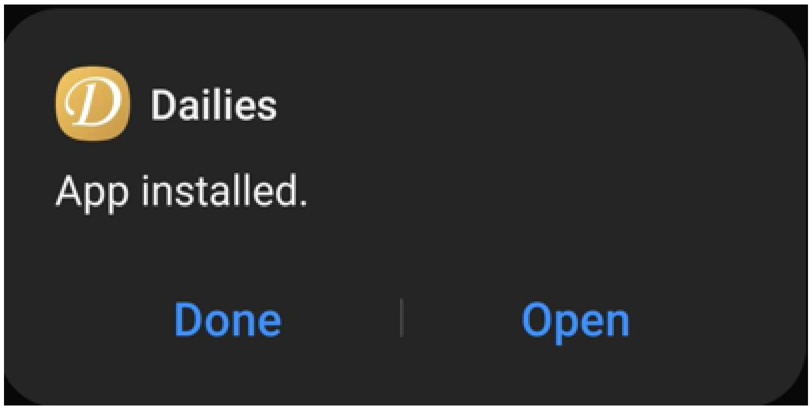
    Installing this app may put your device at risk. Learn more about Play Protect.

    Install anyway

    Got it

    b. Click “Install anyway”

4. Image of installation completion

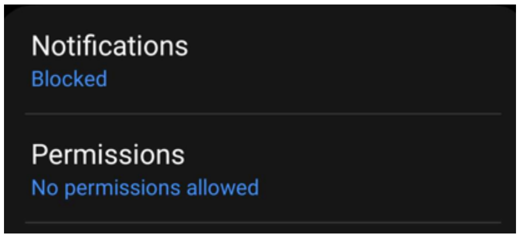
    a. 

    Dailies

    App installed.

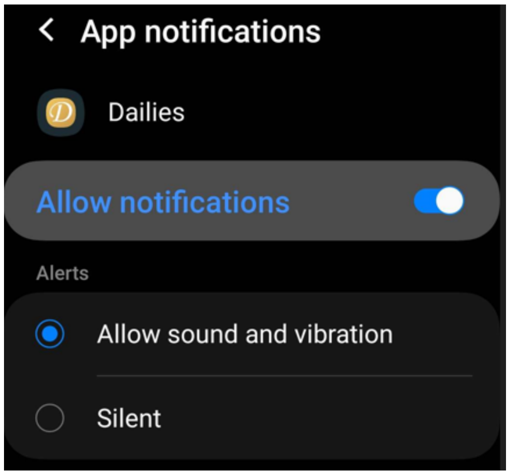
    Done | Open

5. Before using the app, the user needs to provide permissions to allow the app to provide notifications

a. 

Notifications
Blocked

Permissions
No permissions allowed

b. 

Allow notifications

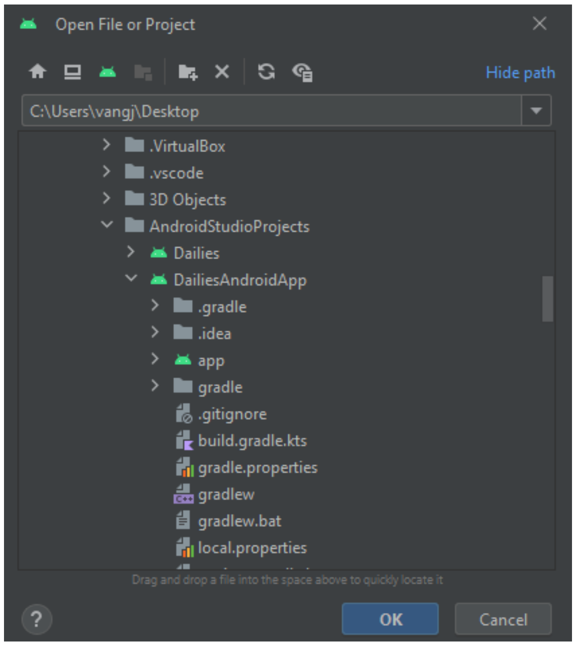
Alerts

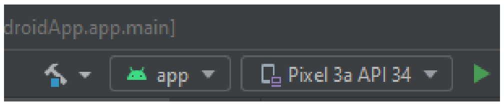
Allow sound and vibration

Silent

## Testing App via Android Studio

Another option to test and run the application is to open the source code in Android Studio.

1. Download and install Android Studio.
    a. Android Studio will need some basic setup and will download dependencies.
2. In Android Studio, File -> Open
    a. Locate the source code.
    b. 

    c. Click “Ok”
3. Once the source code has been loaded and dependencies have been installed, the green play button should be available on the top right.
    a. 

4. This will launch an Android emulator with the application automatically running.

## Run or Operation Books

### Disaster Recovery, Backup, and Restore

This application will be backed up on the developer’s computer, work laptop, and online on GitHub. This is to prevent and mitigate the damage of loss code as much as possible. Further storages to backup the application in the future can be added.

It may be possible for users to retrieve their database file and information through their Android phone’s root. However, this is not recommended because that is a more advanced topic. Due to the mobile application’s simplistic nature and functionality, it is better to not leave any sensitive or personal data within the app to begin with.

Service Levels: SLIs/SLOs/SLAs

## Definitions

SLI (Service Level Indicator) - A metric used to measure the quality and performance of a product. The actual numbers of overall performances such as latency, error rate, wait times, uptimes, etc.

SLO (Service Level Objective) - The objective of the SLI set by developer, team, or organization. SLO is a specified metric such as level of performance for a service.

SLA (Service Level Agreement) - Agreement between the provider and the consumer/customer. Usually about the service meeting a certain criteria, and if unable to do so, there may be some form of penalty or refund. (Judkowitz, Carter 2018)

## App SLI/SLO/SLAs

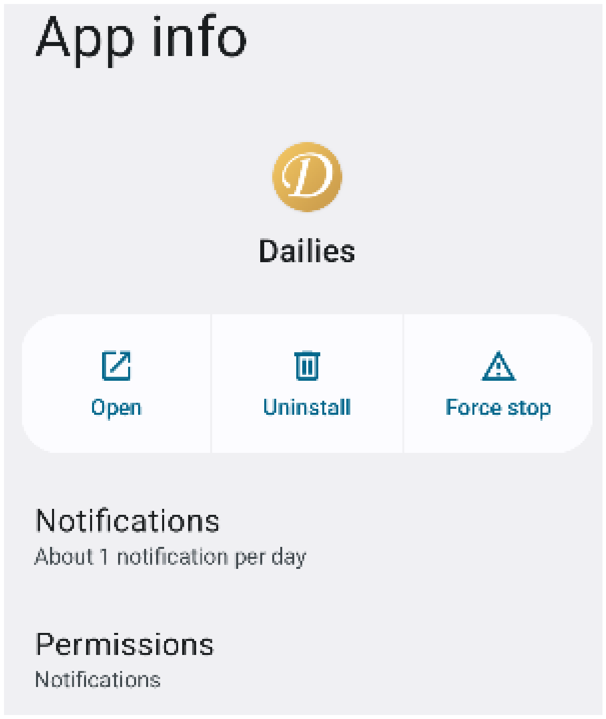
**SLI:** The application launches within 2 seconds of opening the application. Loading into another page takes 2 seconds. Creating a task is within 2 seconds of wait time. A notification is delivered in 2 seconds.

**SLO:** The application will have 99% offline functionality. Application crash rate will be below 10%. 90% of the app should launch within 5 seconds. 90% of screens should load within 2 seconds.

**SLA:** This application will aim to always be available as it is an offline local app. Should there be new upgrades in development, a previous version will always be up for the users unless there is specific reasons such as security. The app will aim to respond to user’s interactions within 3 seconds. If issues arise with bugs or security, users will be able to find out about them within 48 hours of the developer finding out about the issue.

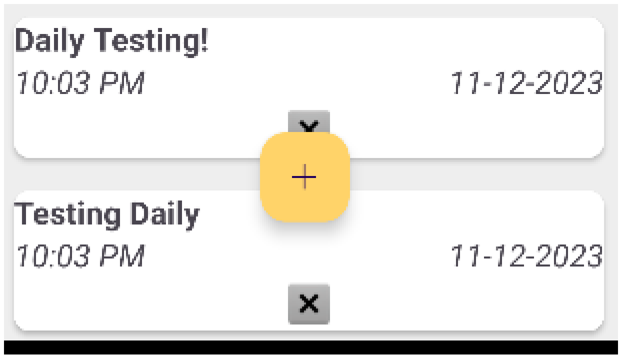
Note that these numbers and agreements are made up and not reliable so far. Only to serve as a basis and learning experience.

User Manuals

1. Refer to the installation page if the app is not installed.
2. Before running the app, make sure the app has permissions and notifications enabled.
   a. 
      App info

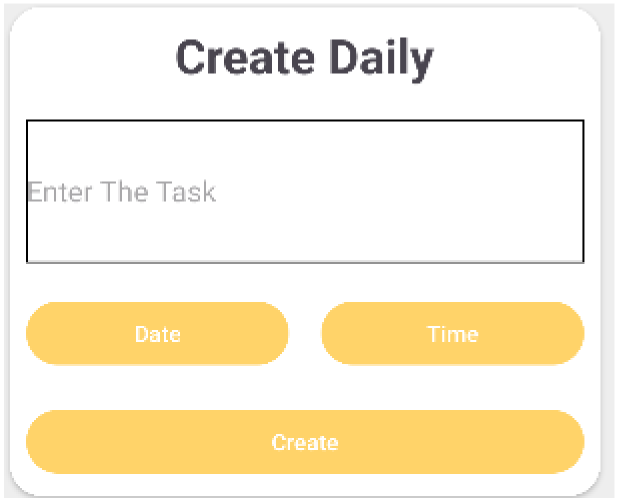
      Dailies

      Notifications
      About 1 notification per day

      Permissions
      Notifications
3. Upon opening the application, created tasks will be shown.
   a. 
      Daily Testing!
      10:03 PM 11-12-2023

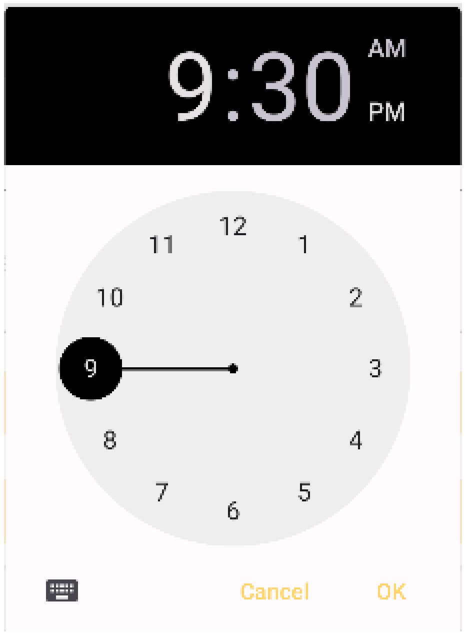
      Testing Daily
      10:03 PM 11-12-2023
4. Press the "+" button to create a task.

5. Enter task name

   a. 

   b. Select Date

   c. Select Time

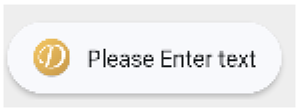
      i. 

      ii. Time can be selected through a clock or typed in

   d. Press the Create button

      i.

ii. If there are issues with what the user has inputted, the app will not create the task and notify the user of it.

1. 
Please Enter text

2. Should the user not want an alarm, the user can opt to not include a date or time and the task will still be created but no alarm will be made.

## Conclusion

The purpose of this project was to create an Android mobile application that can be a user's todolist and alarm. It was also meant to be a learning experience for the developer on learning Android mobile development. It is coded in Kotlin and uses XML as a UI layout in Android Studio. The application stores the user's inputted data into a SQLite database locally on their Android phone.

This project originated from a different mobile app project with similar capabilities and limitations in CPSC 490. In 491, it was changed to this reminder application. This application uses an MVC architecture following a Waterfall SDLC.

This mobile application strives to be simplistic, functional, and portable. It is a digital notebook for its users. It is lightweight and has a minimalistic design of being primarily white with yellow accents. It is intended for the user to simply open the app and get exactly what they need.

The application's basic functionality works. The user is able to create and delete their inputted tasks and also receive notifications for them. However, there is still much to be improved on as there are issues and QOL features that could be implemented to make a more friendly user experience. Overall, it was a valuable learning experience for mobile development, software development, and computer science topics.

## References

1. Athuraliya, A. (2022, December 9). *How to make a user flow diagram*. Creately. https://creately.com/guides/user-flow-diagram/
2. Blazebrain. (2022, February 26). *Building a reminder app with local notifications using WorkManager API*. DEV Community. https://dev.to/blazebrain/building-a-reminder-app-with-local-notifications-using-workmanager-api-385f
3. Courtemanche, M., Mell, E., Gillis, A. S., & Riley, C. (2021, December 22). *What is DevOps? the ultimate guide*. IT Operations. https://www.techtarget.com/searchitoperations/definition/DevOps
4. DataFlair. (2022, January 26). *Android Task Reminder app*. DataFlair. https://data-flair.training/blogs/android-task-reminder-app/
5. Google. (n.d.). *Document  :  android developers*. Android Developers. https://developer.android.com/reference/org/w3c/dom/Document
6. Halder, S. (2023, June 26). *Mobile App Threat Modeling and Security Testing*. Mobile Application Security Testing. https://www.appknox.com/blog/mobile-app-threat-modeling-and-security-testing
7. Judkowitz, J., & Carter, M. (2018, July 19). *Sre Fundamentals: Slas vs slos vs slis | google cloud blog*. Google. https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-slis-slas-and-slos
8. Kumar, P. (2023, June 20). *What is Android Unit Testing?*. BrowserStack. https://www.browserstack.com/guide/android-unit-testing#:~:text=Run%20the%20unit%20tests%3A%20Right,all%20the%20tests%20pass%20successfully.

9. Krüger, G., & Lane, C. (2023, January 17). *HOW TO WRITE A software requirements specification (SRS document)*. Perforce Software. https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#:~:text=Tool%20for%20SRS-,What%20Is%20a%20Software%20Requirements%20Specification%20(SRS)%20Document%3F,stakeholders%20(business%2C%20users).
10. Mishra, R. (2021, January 24). *Android architecture patterns*. GeeksforGeeks. https://www.geeksforgeeks.org/android-architecture-patterns/#
11. *Owasp Mobile Application security¶*. OWASP Mobile Application Security. (n.d.). https://mas.owasp.org/#our-mission
12. Pal, S. K. (2023, May 6). *Software engineering: Classical waterfall model*. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/
13. Visual Paradigm. (n.d.). *What is Use Case Diagram?*. What is use case diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/