AlarmNest - CSCI3310 Project Phase II Report

Group ID:20

Member: CHEN Danggui 1155141529

Layout Flow

Details of the layout and UI style can be found on page 2.

1. Permission Management Part

Only when AlarmNest is launched for the first time on a device, a series of dialogs will guide the user through the necessary authorization steps.

Users will be prompted to enable four essential permissions: `Exact Alarm`, `Activity Recognition`, `Notification`, and `Storage`. After enabling these, Alarmnest needs to be restarted for the permissions to take effect. Subsequent launches of AlarmNest will not display the same permission request dialogs, allowing users to set alarms normally.

2. Clock Setting Part

AlarmNest starts with a `ClockListScreen`, which displays a series of nested alarms integrated in the form of main alarms and sub-alarms. By clicking different components of the alarm card, a dialog will pop up to **allow detailed settings**, such as changing the time, adding descriptions, setting repeat modes, specifying triggering methods, and more.

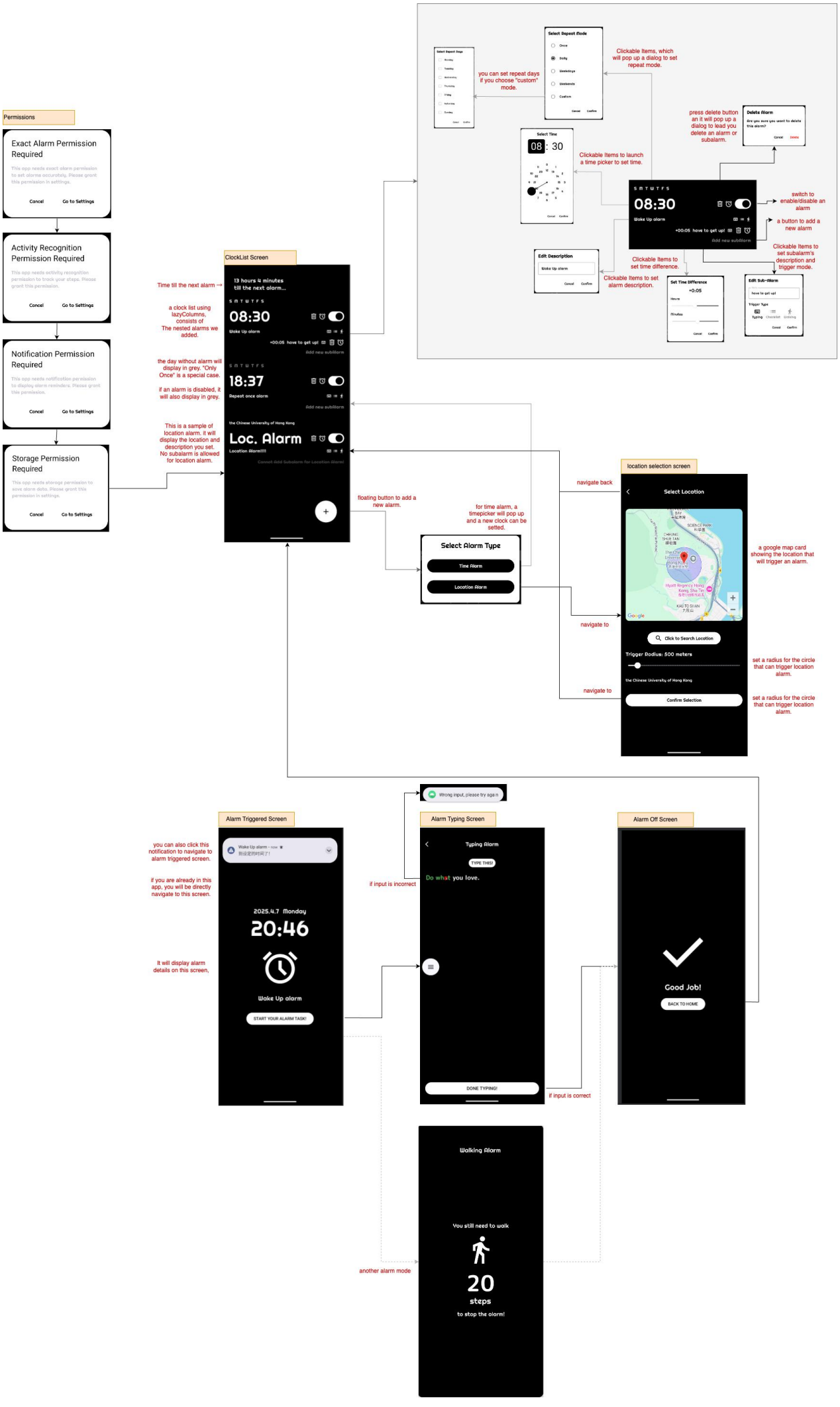
The floating add button located at the bottom right corner allows users to **add either a `time alarm` or a `location alarm`**. When selecting to add a `time alarm`, a time picker dialog appears, where users can pick a time and add a new alarm. When selecting to add a `location alarm`, the app navigates to a new screen where users can set the location and trigger radius. The newly added time or location alarm will be displayed in the `ClockListScreen`.

3. Alarm Triggering Part

When the alarm's time or location meets the triggering conditions, **alarm music** will play, and **a series of notifications** will be sent to prompt the user to return to the `AlarmNest` app to turn off the alarm. It is worth noting that **even if the app is closed, the alarm will still ring when triggered**.

By clicking the notification or opening the app, users will be directed to the `Alarm Triggering Screen`. Users will navigate to the corresponding alarm mode that has been set, such as a walking alarm requiring the user to walk 20 steps or a typing alarm requiring the user to correctly type a specific sentence. **The alarm sound will not stop until the task is completed and the button is pressed**.

After completing the alarm task, users will be directed to the `Alarm Off Screen`. This screen will close and return to the Clock List Screen either after the button is clicked or after waiting for 3 seconds.


Permissions
Exact Alarm Permission Required
This app needs exact alarm permission to set alarms accurately. Please grant this permission in settings.
Cancel
Go to Settings
Activity Recognition Permission Required
This app needs activity recognition permission to track your steps. Please grant this permission.
Cancel
Go to Settings
Notification Permission Required
This app needs notification permission to display alarm reminders. Please grant this permission.
Cancel
Go to Settings
Storage Permission Required
This app needs storage permission to save alarm data. Please grant this permission in settings.
Cancel
Go to Settings
ClockList Screen
Time till the next alarm →
a clock list using lazyColumns, consists of The nested alarms we added.
the day without alarm will display in grey. "Only Once" is a special case.
if an alarm is disabled, it will also display in grey.
This is a sample of location alarm. It will display the location and description you set. No subalarm is allowed for location alarm.
13 hours 4 minutes till the next alarm...
08:30
Wake Up alarm
+00:05 have to get up!
Add new subAlarm
18:37
Repeat once alarm
the Chinese University of Hong Kong
Loc. Alarm
Location Alarm!!!!
Cannot Add Subalarm for Location Alarm!
floating button to add a new alarm.
Select Alarm Type
Time Alarm
Location Alarm
for time alarm, a timepicker will pop up and a new clock can be setted.
navigate back
navigate to
navigate to
location selection screen
Select Location
Click to Search Location
Trigger Radius: 500 meters
the Chinese University of Hong Kong
Confirm Selection
a google map card showing the location that will trigger an alarm.
set a radius for the circle that can trigger location alarm.
set a radius for the circle that can trigger location alarm.
Select Repeat Mode
Once
Daily
Weekdays
Weekends
Confirm
Clickable Items, which will pop up a dialog to set repeat mode.
you can set repeat days if you choose "custom" mode.
press delete button an it will pop up a dialog to lead you delete an alarm or subalarm.
Delete Alarm
Are you sure you want to delete this alarm?
Cancel
Delete
Select Time
08 : 30
Clickable Items to launch a time picker to set time.
S M T W T F S
08:30
Wake Up alarm
+00:05 have to get up!
Add new subAlarm
switch to enable/disable an alarm
a button to add a new alarm
Clickable Items to set subalarm's description and trigger mode.
Edit Description
Wake Up alarm
Cancel
Confirm
Clickable Items to set time difference.
Clickable Items to set alarm description.
Set Time Difference
+0:05
Hours
Minutes
Cancel
Confirm
Edit Sub-Alarm
have to get up!
Trigger Type
Typing
Checklist
Sliding
Cancel
Confirm
Alarm Triggered Screen
you can also click this notification to navigate to alarm triggered screen.
if you are already in this app, you will be directly navigate to this screen.
It will display alarm details on this screen,
Wake Up alarm · now
2025.4.7 Monday
20:46
Wake Up alarm
START YOUR ALARM TASK!
Wrong input, please try again
Alarm Typing Screen
Typing Alarm
TYPE THIS!
Do what you love.
DONE TYPING!
if input is incorrect
if input is correct
Alarm Off Screen
Good Job!
BACK TO HOME
Walking Alarm
You still need to walk
20
steps
to stop the alarm!
another alarm mode

Features Brief Description

1. From the Perspective of Components Usage

1. AlarmNest is built using **single-Activity mode** and `Jetpack Compose` for UI construction, **simplifying lifecycle management**. Additionally, error handling and recovery mechanisms are provided wherever possible to enhance lifecycle management reliability.
2. Internally, AlarmNest primarily uses `Navigation Receiver` to switch pages in a router-like manner across different screens. External requests, such as opening the settings panel, are handled using intents.
3. A `PermissionManager` class has been created to unify permission management, including permissions for precise alarms, activity recognition, notifications, and storage. This ensures a **user-friendly permission request process**.
4. The system's `AlarmReceiver` is used to set precise alarm times, with `PendingIntent` sending alarm-triggering events to the `AlarmReceiver`. Additionally, `AlarmService` is used as a foreground service to acquire a `WakeLock` to prevent sleep, play alarm music, and send notifications. **This ensures that even if the app is killed, the system's `AlarmManager` can still trigger alarms**.
5. The app uses `Room Database` to establish a robust database and entity definitions, providing **convenient CRUD operations**. The WAL mode is enabled, and periodic database maintenance is performed for data cleanup, **optimizing performance**. KSP is utilized to make database operations more efficient and reliable.
6. `Google Maps API` and `Google Places API` are integrated to enable **map-based location tracking and location retrieval** functionality. Meanwhile, `Google Play Service`s' `Geofencing API` is utilized to implement location-based alarm triggering through **geofencing**.
7. For the walking alarm feature, the app can use `Android's Sensor API` to detect steps by listening to either the `TYPE_STEP_DETECTOR` sensor or the `TYPE_ACCELEROMETER` sensor.
8. Integration of `qwen2.5` network requests to optimize automatic time and note settings for alarms **using LLMs**.
9. Use of the system's `MediaStore Content Provider` to upload audio files in suitable formats and sizes for setting custom alarm sounds.
10. By using `WorkManager` to automatically execute tasks on background threads, the framework's intelligent scheduling of background tasks achieves the goal of **Battery Optimization**.

2. From the Perspective of Unique Features

1. Provides a simple **main alarm/sub-alarm architecture** based on "event time differences" to dynamically set alarms, optimizing time management efficiency. By leveraging event associations, it ensures no details are missed.
2. Integrates Google Maps to offer location-based alarm triggers, providing **dynamic reminders** for events with uncertain timing.

> Although the geofencing feature is implemented in the app, in reality, the location alarm has a high risk of failing or not triggering in time. Therefore, location-based alarms are disabled by default.

3. Implements more complex and **harder-to-dismiss Wake-up Alarms,** requiring users to focus during responses, helping them prepare for upcoming tasks in better condition.
4. Integrates large language models to generate **personalized reminder suggestions** based on user habits and historical data. This offers lightweight intelligent Alarm Recommendations, reducing the time spent on manual input and alarm setup.
5. Supports custom ringtone selection and integrates the system's media library, enabling more personalized alarm settings. Multiple wake-up verification methods can be configured, allowing users to **tailor reminder strategies to different scenarios**.

Details of the Top 3 Most Interesting/Technically Challenging Features

1. Main Alarm/Sub-Alarm Architecture

This feature provides a unique approach to time management by offering a main alarm/sub alarm architecture based on **time differences**. It can dynamically reset subalarms based on main alarms' change, optimizing time management efficiency.

The technical challenges involve designing an architecture that can dynamically adjust alarms based on event dependencies, ensuring seamless integration and accurate time management. The design of the database structure, correct and timely synchronization between the frontend and backend, and data preservation and recovery when the app is interrupted or exited are all issues that need careful consideration. This approach enhances the app's functionality by providing users with efficient and reliable alarm settings tailored to their schedules.

2. Triggering Alarms Outside the App

As a core functionality of an alarm app, triggering alarm sounds is both essential and technically challenging. AlarmNest utilizes the system's `AlarmManager` to set precise alarms and employs **`PendingIntent`** for inter-process communication, sending alarm-trigger events to `AlarmReceiver`. Note that whenever the phone restarts, the `PendingIntent` is cleared, so you need to reopen AlarmNest to restore the alarms. To prevent the system from sleeping, a **`WakeLock`** is used to wake up the CPU, while `AlarmService` operates as a foreground service to play alarm sounds.

This feature involves the integration of two Android component types—services and broadcast receivers—along with the need to efficiently schedule multi-threaded tasks, making its technical implementation highly complex.

3. LLM Integration

Applying large language models to traditional applications to improve efficiency is an interesting topic. Here, I integrate **the `qwen2.5-72b` huggingface model** into the alarm app to generate personalized sub-alarm recommendations based on user habits, optimizing alarm times and note settings.

The technical challenges of this feature lie in designing suitable data structures and algorithms in prompts, as well as processing historical data to generate personalized suggestions. Transforming user behavior data into actionable recommendations enhances the app's intelligence and provides a more convenient alarm-setting experience through cutting-edge LLM technology.

Build Instructions

Version of Android Studio: Android Studio Meerkat | 2024.3.1 Patch 1

1. Ensure the project includes `secrets.properties` and `local.defaults.properties` files, and that `MAPS_API_KEY` is correctly configured. Without these keys, Google Maps and Google Places functionalities may not work properly.
2. The testing device used during development is **Medium Phone**, running **Android 16.0 ("Baklava")** on arm64 architecture, with support for Google Play services.
3. Open the project in Android Studio, wait for the Gradle synchronization to complete, and then build the project via Android Studio.
4. Finally, run the application on an emulator or a physical device that meets the specifications above.

Reflection on the Project

Due to time constraints, I decided to temporarily abandon the implementation of the **Checklist Alarm** feature. While it is not particularly difficult to achieve. Actually, it use the same custom components and principles as the Typing Alarm. However, it requires database integration and changes to both the front-end and back-end. Given the limited time, I chose to focus solely on Typing and Walking alarms for now.

For **Location-based alarms**, my initial idea was to use network data to find suitable geofencing information. However, I couldn't find appropriate datasets for implementation. Instead, I used circular regions for triggering alarms. This approach may lead to issues such as alarms not being triggered despite entering the area due to the circle's radius, or alarms being triggered when only nearby activities occur. Although this feature adds innovation to the alarm app, it might not be very **convenient** in practice.

I also attempted to train a lightweight model to deploy within the program, but the results were unsatisfactory. I also tried integrating a DeepSeek API model, but the DeepSeek model might respond too slowly or even fail within the Hong Kong area. As a result, I ultimately switched to using a **HuggingFace inference model of qwen2.5-72b** for intelligent recommendations, ensuring response speed and success probability. Additionally, when the model cannot response, a default label will apply now.

Lastly, although it's difficult to achieve perfection, I made an effort to improve the **Battery Optimization** feature. I utilized the `WorkManager` framework to handle background tasks, allowing `WorkManager` to intelligently schedule these tasks based on system status. Additionally, the advantage of using `WorkManager` is that it eliminates the need to manually create and manage threads, reducing errors potentially caused by threading issues, thus making the program run more smoothly.