CCExtractor: Ultimate Alarm Clock II

Basic Information:

Name	Shlok Upadhyay
University	National Institute of Technology, Tiruchirappalli
Degree	Bachelor of Technology
Timezone	Indian Standard Time (GMT +5:30)
Email	oopsshlok@gmail.com
Resume	SHLOK UPADHYAY - RESUME
Github	oops-shlok
LinkedIn	https://www.linkedin.com/in/shlok-upadhyay-521065104/
Postal Address	676 New Viman Nagar, Kanpur, Uttar Pradesh, India - 208007
Phone Number	(+91) 7607066219

Project Title:

Ultimate Alarm Clock II

Abstract:

Ultimate Alarm Clock is a mobile app that aims to provide features that other alarm clocks don't have such as conditional alarms based on location, phone activity, weather, or shared settings. Besides these, the clock also includes unique features to dismiss the alarm such as scanning a QR/Bar code, solving simple math problems of varying difficulty or shaking the phone. This is an open-source, free, and ad-free project that's built with Flutter. I propose to improve the Ultimate Alarm Clock app by enabling iOS notifications through Swift and Flutter integration, providing an iPhone version of the app, expanding features such as negative condition alarms based on location and weather, shared alarm notifications, improving weather-based alarms and diverse ringtone options. These enhancements will make the app more versatile and

user-friendly across Android and iOS platforms.

Mentors:

Rijuth Menon R, Nishant Singhal

Duration:

350 hrs in 15 weeks

Difficulty Level:

Medium

Motivation:

Since last year, I started to build a keen interest in flutter because of its highly growing community and easy-to-understand documentation. Plus, I love the idea of working on something that's open-source, meaning it's freely available for anyone to use and contribute to. It's a great opportunity to work with other people who share similar interests and goals.
Also, the idea behind this project itself motivates me, as I would be able to learn more while contributing to this project, and this ensures that I commit my maximum time towards this project.

Why CCExtractor:

The wide range of topics and technologies CCExtractor consists of, intrigued me to contribute to this society. By joining the community, there are much greater chances that I would be able to learn more on not just my project but also other topics that include peer to peer, AI, cloud, and other technologies. I have seen the exciting quality discussions on community channels which thereby motivates me to explore more stuff. Lastly, the supportive environment that mentors provide is one of the essential factors that I would love to contribute to this organization.

I’m only applying for CCExtractor for GSoC’24 and have no plans to contribute to any other organization.

Goals:

1. Negative Condition Alarms:

In the current status of the app, there is condition based cancellation of alarm like, alarm cancellation based on specific location of the user, alarm cancellation based on specific weather etc. To improve this feature, we can introduce the functionality to have alarm activation based on conditions like alarm ringing only if the user is within a certain range of the specified location, alarm rings only if the current weather matches the weather chosen by the user etc.

2. Shared Alarm Notifications:

In the shared alarm feature, we can introduce a new feature of sending an in-app notification to the user if he/she is added to a shared alarm. By receiving these notifications, users will stay informed and can actively participate in shared alarms, fostering better collaboration and coordination.

3. Integration with Open-Meteo:

In the current status of the app, for the weather based alarm feature, we are using Open weather which requires an API key to fetch the current weather details. This requires the user to get an API key necessarily to fetch it. To remove this friction, we can use the API provided by the Open-Meteo to fetch the current weather details which doesn't require an API key.

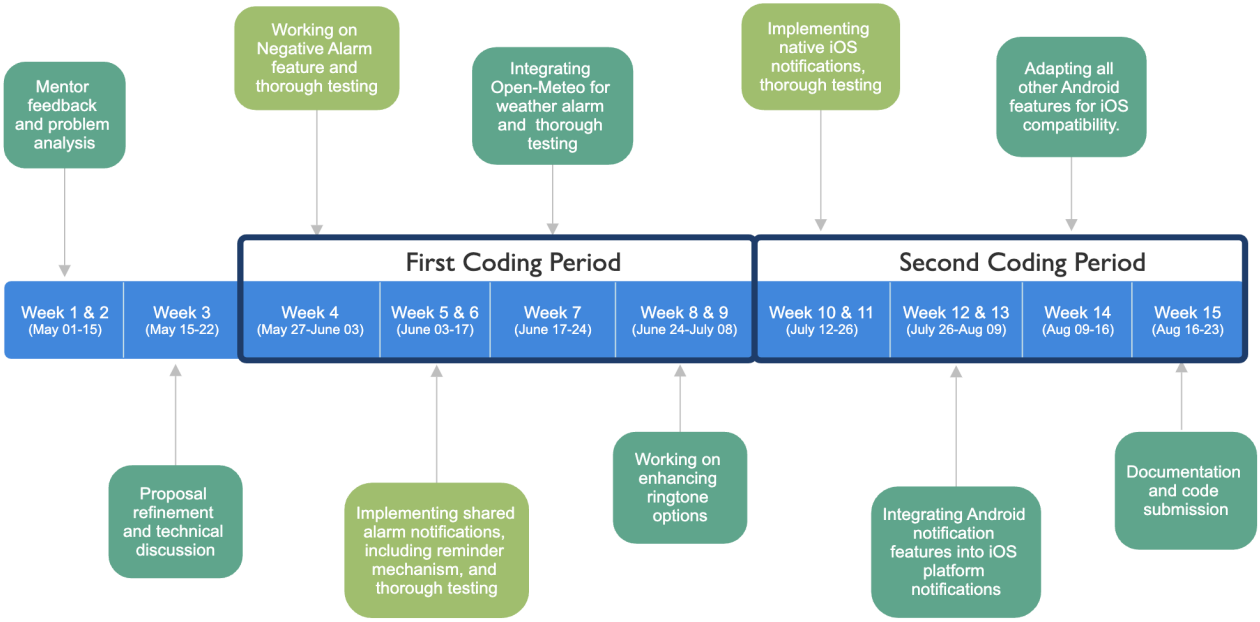
4. Enhanced Ringtone Options:

In the current status of the app, we only have the option to either use the Default ringtone of the device or adding a custom ringtone for the alarm. We can improve this feature by integrating with external providers or sourcing free to use sounds for users to choose from.

5. Alarm Notifications on iOS Devices:

The current state of the Ultimate Alarm Clock app is limited to Android devices, with alarm notifications not functioning on iOS devices. The objective is to address this limitation by implementing native notifications for iOS using Swift, integrated with Flutter. Not only the iOS notifications, but we can also integrate all the features which are currently working on the Android platform, to work on the iOS platform too.

Timeline and Implementation:



Implementation is divided into the following **Milestones (all dates mentioned below are of 2024)**

Community Bonding Period: *(May 01 - May 27)*

I will actively engage with the Ultimate Alarm Clock community, getting to know my mentors and fellow students. I will engage in meaningful discussions with my mentors to refine my proposal based on their valuable feedback. Together, we will thoroughly analyze potential challenges that may arise during feature implementation and brainstorm various edge cases to ensure comprehensive problem-solving. With guidance from my mentors, I will finalize the proposal requirements, ensuring clarity and alignment with project objectives. Additionally, we will discuss the theoretical and technical aspects of each subproblem outlined in the proposal, aiming for a deep understanding to facilitate effective solutions. Specifically, I will collaborate with mentors to discuss optimized approaches for scheduling alarms on iOS devices, with their expertise to provide efficient solutions.

Milestone 1: Negative Condition Alarm Feature: *(May 27 - June 03)*

In the current status of the app, we have a condition to cancel the alarm based on location, weather and phone activity. To complement this existing feature, we can introduce the functionality to ring the alarm only based on specific location and weather.

(i) Location Based: In `location_tile.dart`, we'll add an option for location-based alarm activation. This toggle allows users to enable or disable this feature. In `AddOrUpdateAlarmController`, we'll include a boolean variable, `isLocationActivationEnabled`, to track this setting. A method in the controller will manage location-based activation based on user selection. Users can specify desired alarm locations. Once set, `isLocationActivationEnabled` is marked as true. Using `isWithinRadius` method from `utils.dart`, we'll check if the user is within 500 meters of the specified location, using Haversine formula. If not, `shouldAlarmRing` in `SplashScreenController` will be set to false, ensuring the alarm only rings within the specified location range.

(ii) Weather Based: For weather-based alarm activation, we'll make changes similar to those in `location_tile.dart`. We'll add a boolean variable `isWeatherActivationEnabled` in the `AddOrUpdateAlarmController` to track this feature. A method in the controller will handle weather-based activation based on user preferences. Users can specify desired weather conditions, similar to weather-based cancellation via Checkbox. After specifying, `isWeatherActivationEnabled` will be set to true. We'll use the `checkWeatherCondition` method in `SplashScreenController` to verify if current weather matches the user's selection. A new boolean variable in `SplashScreenController` will segregate weather-based cancellation and activation. In the `onInit` method, if this variable is true, `shouldAlarmRing` will be set to true.

Milestone 2: Shared Alarm Notifications: *(June 03 - June 17)*

This is another feature that we can implement in the app. This feature will provide in-app notifications to the users to accept or decline the request for shared alarms. To implement this, we can follow the below steps:

Step 1: AlarmModel Update:

We can enhance the `AlarmModel` class to accommodate in-app notifications for shared alarm requests. Firstly, we'll add a `notificationStatus` field to signify the status of the associated notification, including states like pending, accepted, or declined. Initially, it'll

be set to pending until the user responds. Secondly, we'll include a `notificationType` field to specify the type of notification, such as a shared alarm request. This allows the application to distinguish between different types of notifications for proper handling. So, we will be adding these two fields in the `alarm_model.dart` file.

Step 2: Updating Firestore Schema:

After updating the AlarmModel, the next step will be updating the Firestore schema. Firestore Schema update would involve modifying the structure of Firestore documents to accommodate new fields for notification status and type in shared alarms. So, we have to include notificationStatus and notificationType fields in the Firestore schema too. To do this, we can update the `addUserToAlarmSharedUsers` and `removeUserFromAlarmSharedUsers` methods to include these two fields in the Firestore document, when a new user is added and removed to the shared alarm respectively.

```
static addUserToAlarmSharedUsers(UserModel? userModel, String alarmID) async {

  // Already implemented logic...


  // Setting notification status and type here
  if (!sharedUserIds.contains(userModelId)) {
    sharedUserIds.add(userModelId);
    await alarmDoc.reference.update(
      {
        'sharedUserIds': sharedUserIds,
        'offsetDetails': offsetDetails,
        'notificationStatus': 'pending',
        'notificationType': 'shared_alarm_request'
      },
    );
  }
  return true;
}

static Future<List<String>> removeUserFromAlarmSharedUsers(
  UserModel? userModel,
  String alarmID,
) async {

  // Already implemented logic..

  // Removing notification status and type here
```

```
  if (sharedUserIds.contains(userModelId)) {
    sharedUserIds.remove(userModelId);
    await alarmDoc.reference.update(
      {
        'sharedUserIds': sharedUserIds,
        'notificationStatus': FieldValue.delete(),
        'notificationType': FieldValue.delete(),
      },
    );
  }

  return sharedUserIds;
}
```

Step 3: Implementing notification handling:

Once the AlarmModel enhancements are complete, we can handle notifications based on notificationStatus and notificationType. When a user adds another user to a shared alarm, we'll trigger a notification to the added user. In the addUserToAlarmSharedUsers method, we'll check if notificationStatus is 'pending' and notificationType is 'shared_alarm_request' before triggering the notification using the in_app_notification package. We'll handle user responses ('Accept' or 'Decline') and update notificationStatus accordingly.

```
if (!sharedUserIds.contains(userModelId)) {
  sharedUserIds.add(userModelId);
  await alarmDoc.reference.update(
    {
      'sharedUserIds': sharedUserIds,
      'offsetDetails': offsetDetails,
      'notificationStatus': 'pending',
      'notificationType': 'shared_alarm_request'
    },
  );

  if (notificationStatus == 'pending' && notificationType == 'shared_alarm_request')
{

    InAppNotification.show(
      title: 'Shared Alarm Request',
      body: 'You have received a shared alarm request.',
      actions: [
        NotificationAction(
          label: 'Accept',
```

```
          onTap: () async {
            await alarmDoc.reference.update({'notificationStatus':'accepted'});
          },
        ),
        NotificationAction(
          label: 'Decline',
          onTap: () async {
            await alarmDoc.reference.update({'notificationStatus':'declined'});
          },
        ),
      ],
    );
  }
}
```

Now, the user can also miss the in-app notification for the first time too. So, to fix this problem, we can implement a mechanism to periodically check for notifications with a status of "pending." When the app detects any pending notifications, it can trigger in-app reminders to notify users about these pending shared alarm requests. I will be discussing with mentors to get a more optimized solution for this issue.

Milestone 3: Integration with Open-Meteo: *(June 17 - June 24)*

Next feature on which we can work is integration with Open-Meteo for weather based alarms. Currently, the weather based alarms are using open weather which requires an API key. Getting an API key for Open Weather can be a bit of a hassle because the user has to sign up and go through some steps to get the API key. So, to avoid this, we can switch to integration with Open-Meteo. Switching to Open-Meteo for weather based alarms could simplify things since it doesn't need an API key. This means skipping the hassle of signing up and managing the key. To integrate Open-Meteo for weather based alarms by replacing the current usage of open weather, we can directly use the API provided by Open-Meteo for fetching the weather details of the user based on his/her location. For integrating the alarm based on weather, we need to get the exact latitude and longitude of the user which we can use to fetch the response from the Open-Meteo API. This we can get from the `setting_controller.dart`'s `getLocation` method which uses the `fl_location` package. From this method, we will be getting the user's current location's latitude and longitude values. Now, Open-Meteo provides different options to fetch the weather details based on latitude and longitude. But, we just need to get the current weather response from the API. To get the current weather details, we can just add "current" as a parameter and give corresponding values to this key. Based on how the weather based alarm feature is there in the app, we just need to get the type of weather (i.e sunny, cloudy, rainy and stormy). But in Open-Meteo, instead of giving

the type of weather for the current weather details, it provides the weather code in the response. So, we can pass “weather_code” as a value to the “current” parameter’s key. Also, for the “Windy” weather type, we can’t get that from the weather code in the response. We need to get the current weather speed to check if the weather is windy or not. So we need to pass another value as “wind_speed_10m” to the “current” param’s key. After this, the one more parameter that is needed to get the successful response is the timezone. However, we can pass “auto” as a value to this parameter’s key which will automatically detect the time zone based on the latitude and longitude values. Here’s how the request URL will look like:

https://api.open-meteo.com/v1/forecast?latitude={user’s latitude value}&longitude={user’s longitude value}&current=weather_code&timezone=auto

Here’s how the response will be like:


```
{
    "latitude": 26.375,
    "longitude": 80.375,
    "generationtime_ms": 0.030994415283203125,
    "utc_offset_seconds": 19800,
    "timezone": "Asia/Kolkata",
    "timezone_abbreviation": "IST",
    "elevation": 127.0,
    "current_units": {
        "time": "iso8601",
        "interval": "seconds",
        "weather_code": "wmo code",
        "wind_speed_10m": "km/h"
    },
    "current": {
        "time": "2024-03-23T00:45",
        "interval": 900,
        "weather_code": 1,
        "wind_speed_10m": 4.2
    }
}
```


Now once we get the weather_code from the response, we can get the type of weather

WMO Weather interpretation codes (WW)

Code	Description
0	Clear sky
1, 2, 3	Mainly clear, partly cloudy, and overcast
45, 48	Fog and depositing rime fog
51, 53, 55	Drizzle: Light, moderate, and dense intensity
56, 57	Freezing Drizzle: Light and dense intensity
61, 63, 65	Rain: Slight, moderate and heavy intensity
66, 67	Freezing Rain: Light and heavy intensity
71, 73, 75	Snow fall: Slight, moderate, and heavy intensity
77	Snow grains
80, 81, 82	Rain showers: Slight, moderate, and violent
85, 86	Snow showers slight and heavy
95 *	Thunderstorm: Slight or moderate
96, 99 *	Thunderstorm with slight and heavy hail

Now, in the weather based alarm feature, we only need to check for Sunny, Cloudy, Rainy, Windy and Stormy weather. So, we can assign the weather code to the weather type like this:

Weather Code	Weather Type
0,1	Sunny
2,3,45,48	Cloudy
51,53,55,56,57,61,64,66	Rainy
65,67,95,96,99	Stormy

This we can convert by modifying the `getWeatherTypesFromInt` method in the `utils.dart` file. Now, to check if the current weather is windy or not, we can just check the wind_speed_10m value from the response and if it more than 15 mph i.e 24 kmph (as we receive the wind_speed_10m value in kmph), we can set the weather type as “Windy”. Primarily, the changes will be in the `weather_tile.dart` file, `weather_api.dart` , `utils.dart` and `splash_screen_controller.dart` file.

Milestone 4: Enhanced Ringtone Options: *(June 24 - July 08)*

Next feature on which we can work is enhancing ringtone options for the users. Currently, users can only choose from a default ringtone or manually add additional audio files. Now, in addition to custom ringtones, we can provide a selection of default alarm tones. These default options will cater to a variety of preferences and ensure that users have a range of options to choose from. To do this, we can follow two approaches.

We can integrate the alarm feature with external providers, such as 'Freesound'. This integration enables users to browse and select from a vast library of ringtones available on these platforms. By leveraging the APIs provided by these services, we can incorporate their functionality into the app. To integrate with Freesound, we can request new access credentials to access their API. Basic API calls can be authenticated using a typical API key mechanism, where we add the key given with the access credentials into every request we make. Freesound provides two options for authentication: Token authentication and OAuth2 authentication. Out of these two options, and based on the requirement of having free ringtones in the app, we can opt for token authentication as it only requires an API key to make a request call. Using OAuth2 authentication follows the authorization code grant, which can be a hassle. Once authenticated, we can then use Freesound's API for searching the ringtones. We can start off by using the Text Search route provided by them. It is a GET method with an endpoint as `/apiv2/search/text/`. The request parameters required for this route are "query, filter (optional), sort (optional)". The route's response is like this:

```
{
  "count": <total number of results>,
  "next": <link to the next page of results (null if none)>,
  "results": [
      <sound result #1 info>,
      <sound result #2 info>,
      ...
      <sound result #page_size info>
  ],
  "previous": <link to the previous page of results (null if none)>
}
```

After getting sound results from this route, the user can select a particular sound from this response. Next step will be retrieving detailed information about the sound. This we can get from the `/apiv2/sounds/<sound_id>/` route. Upon making the request, the API would return comprehensive metadata about the sound, including essential details such as its title, duration, and a URL for downloading the audio file. Next, we can implement the method in the audio_utils.dart file to play this selected sound as an alarm

ringtone. We can also look for different external providers which can be more optimal. I will be discussing this thoroughly with the mentors.

Second approach is “Inclusion of free sounds”. We can provide options to include a selection of free sounds within the app. By curating a collection of high quality free sounds, we can enhance the customization options available to the users. To do this, we will begin by finding a wide range of free sounds suitable for alarms. Once we are done finding the free sounds, we add these audio files in a new `assets/ringtones` directory. After this, we can just implement a method in the `audio_utils.dart` file where we can create a map of ringtone names to their file paths. Then using the `audioplayers` package, which is already integrated within the app, we will be able to play these added ringtones based on user selection.

First Evaluation: *(July 08 - July 12)*

I will share the progress made with my mentor, seeking feedback to ensure the project aligns with the organization's scaling and quality standards. If necessary, I will make improvements or adjustments based on the mentor's suggestions. Additionally, I will submit the midterm report on the GSoC website, detailing the project's status, achievements, and any challenges encountered.

Milestone 5: Alarm Notifications for iOS Devices: *(July 12 - July 26)*

To start off, the primary objective is to do the native implementation of notifications on iOS devices.
For the native implementation of notifications on iOS devices, I would first start making an iOS module using Swift by utilizing the UNUserNotificationCenter framework for managing basic notifications in which the features will be to display the alarm time, Date, option to Snooze the alarm and an option to dismiss the alarm. To do this, I’ll follow these steps:

Step 1: Request Authorisation:

Before scheduling notifications, we can request user authorization to display them. In the AppDelegate.swift file, within the `application(_:didFinishLaunchingWithOptions:)` method, we can use

UNUserNotificationCenter's `requestAuthorization(options:completionHandler:)` method to prompt the user for permission to display notifications.

Step 2: Creating Notification Content:

In AppDelegate.swift, we can use UNMutableNotificationContent to create notification content with title, body, and sound. We can implement actions ('snoozeAction', 'dismissAction') with UNNotificationAction, assigning identifiers, titles, and options. We can then assign a category and include extra information via userInfo.

Step 3: Creating Notification Trigger:

After defining the notification content, we can create a trigger to specify when the notification should be delivered. This trigger, determined by conditions like date, time, or location, can be set using UNCalendarNotificationTrigger. Basic implementation of creating a trigger is like this:

```
import UserNotifications
import CoreLocation

// component for the desired date and time
var dateComponents = DateComponents()
dateComponents.hour = 6
dateComponents.minute = 0
dateComponents.second = 0
// Calendar trigger for the specified date and time
let dateTrigger = UNCalendarNotificationTrigger(dateMatching:dateComponents, repeats:
false)

// Repeating trigger with a time interval
let intervalTrigger =   UNTimeIntervalNotificationTrigger(timeInterval:
"time_interval", repeats: true)

// Defining a region
let region = CLCircularRegion(center: CLLocationCoordinate2D(latitude: "latitude",
longitude: "longitude"), radius: "radius", identifier: "LocationIdentifier")

// location trigger for entering or exiting based on the defined region
let locationTrigger = UNLocationNotificationTrigger(region:region, repeats: false)
```

Step 4: Creating Notification Request:

After defining the notification content and trigger, the next crucial step will be to assemble a notification request. We can do this with the help of UNNotificationRequest. This request will serve as a container for all the essential information required to deliver the notification to the user. It includes unique identifiers, the notification's content, and the trigger that dictates when the notification should be presented.

Step 5: Scheduling Notification:

Once the notification request is ready, we can add it to the notification center using UNUserNotificationCenter. By using the `add(_:withCompletionHandler:)` method, we inform the system about the pending notification, triggering the scheduling process.

Step 6: Handling User Interaction:

After scheduling notifications, we can handle user interactions by conforming to the UNUserNotificationCenterDelegate protocol in AppDelegate.swift. This protocol offers methods to respond to notification events like delivery, opening, snoozing, or dismissal. We can also define actions for when the app is backgrounded or running in the foreground.

Milestone 6: Integration of all features for iOS: *(July 26 - Aug 16)*

Next step will be to integrate the features which are currently working for Android platform, on iOS platform too.

(i) Shared Alarm for iOS platform:

For shared alarm for iOS platform, we can integrate Google Sign-In SDK for iOS by configuring OAuth client credentials from Google Cloud Console, installing GoogleSignIn framework, and setting up URL scheme in Info.plist to handle callbacks. We can implement `addUserToAlarmSharedUsers` and `removeUserFromAlarmSharedUsers` functions, checking alarm ID existence in Firestore, verifying user ownership, and updating Firestore documents accordingly. For `streamAlarms` functionality, we can set up listeners to observe changes in Firestore collection, transform data into AlarmModel objects, and apply sorting logic based on user preferences. Updating notification scheduling for shared alarms using UNUserNotificationCenter framework and setting up real-time updates and

synchronization for shared alarms using Cloud Firestore's listener functions can also be implemented to ensure data consistency across multiple devices.

(ii) Alarm Conditional to Phone Activity:

To track device activity on iOS, we can integrate accessibility notifications in AppDelegate.swift. By registering for notifications like UIAccessibility.elementFocusedNotification, we monitor user interactions system-wide. This helps adjust alarm logic based on user activity. We'll handle these notifications in AppDelegate.swift, updating alarm criteria accordingly. If there's no activity for a while, we'll trigger the alarm for timely notifications.

(iii) Alarm Conditional to location:

To implement location-based alarms for the iOS platform in the app, we'll establish communication between Flutter and iOS using platform channels. We will retrieve latitude and longitude values of the user and we will send it to the AppDelegate.swift file. Upon receiving this data in the AppDelegate.swift file through platform channels, we'll update the app's data model with the location information. Next, we'll update the alarm functionality to consider the user's location when determining whether to trigger the alarm or not. For example, we'll configure alarms to ring or not ring only based on the specified location of the user.

(iv) Alarm conditional to weather:

To do this feature, we can implement a method within the AppDelegate.swift file. When evaluating weather conditions against user-defined alarm criteria, we'll dynamically generate notification content reflecting the current weather status. This customization would ensure that alarm notifications are contextually relevant and informative to users. Next, we'll integrate dynamic alarm scheduling functionality, allowing alarms to be scheduled or skipped based on real-time weather conditions. If the current weather matches the user's specific preferences, the alarm will be scheduled or dismissed accordingly.

(v) Maths Challenge:

To implement this feature, we can organize questions into an array for easy access and retrieve them from the existing Android data source. When scheduling notifications, we'll select questions based on user preferences like difficulty level and desired quantity. These questions will be formatted and included in the notification using

UNMutableNotificationContent. User responses will be compared to correct answers, presenting subsequent questions until all are answered correctly. Upon completion, the alarm will be dismissed, providing an engaging and interactive experience similar to Android.

(vi) Custom Ringtone:

To implement this feature, we can incorporate the user's custom ringtone name into the alarm notification content. We retrieve the name from the app's preferences and integrate it when constructing the notification. This process would involve fetching the custom ringtone name set by the user within the app and then incorporating it into the notification's sound property. By leveraging the UNNotificationSound class, we can ensure that the selected custom ringtone is played when the alarm notification is triggered. Also, we can implement a fallback mechanism to handle scenarios where the custom ringtone is unavailable.

Just like the above features, we will be creating methods for other features like haptic feedback, stopwatch notifications, timer notifications etc. which are currently working on android devices, to make sure that these features work on the iOS platform too.

Milestone 7: Proper Documentation and Testing: *(Aug 16 - Aug 23)*

I'll be documenting the code for the features I have worked on, based on the community standards. I'll clean up the documentation and code and will wrap this up to ensure that it is organized perfectly. I'll finalize and test the application on both the platforms (android and ios) and prepare the configuration file (if required) for successful deployment. I'll add widget tests for all essential components and enough integration tests to cover all the important use cases and better performance profiling. Following this, I will be submitting the code for final evaluation.

Testing:

I believe that tests can very well explain the intention of the programmer for a piece of code, better than comments can do as writing tests ensures that developers catch the regressions at an early stage.

To unit test any particular class in isolation, any external dependencies (classes) need to be mocked so no external noise affects our tests. The mocked object would simulate the behavior of a real object, but knows in advance what's supposed to happen in the test and its intended behavior. We can use the mockito package to add mock classes.

For widget testing of the files on which I'll work in the UI, we can use WidgetTester along with the pump() method for testing Stateful Widgets and would refer to this article.

Towards the end, I'll add the rest of the tests like integration tests to ensure proper performance profiling. The goal would be to verify that all the widgets and services being tested work together as expected. To maintain proper testing standards, I'll refer to the flutter cookbook.

Final Evaluation: *(Aug 26 - Sept 02)*

I will keep my mentor informed about the progress made so far, making any necessary improvements or adjustments based on their feedback. Additionally, I will ensure that all widget tests can be run and that they all pass successfully. Furthermore, I will prepare a document outlining the work in progress and suggesting future enhancements for potential contributors. Lastly, I will compile and submit the final report on the GSoC website, summarizing the project's achievements and contributions.

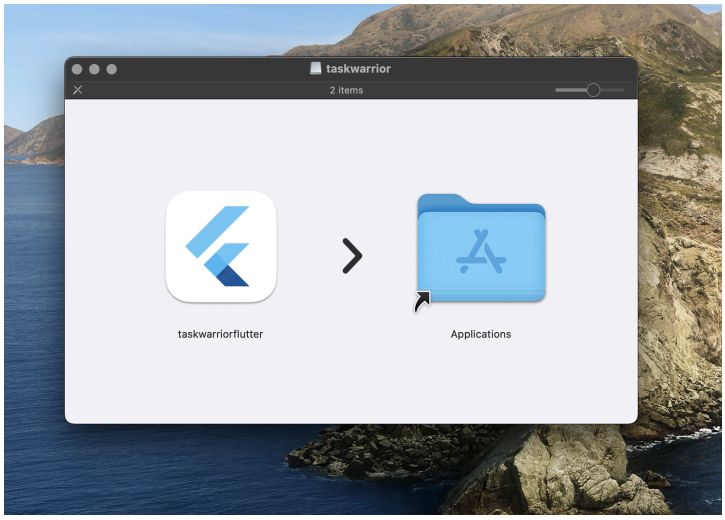
Proof of work:

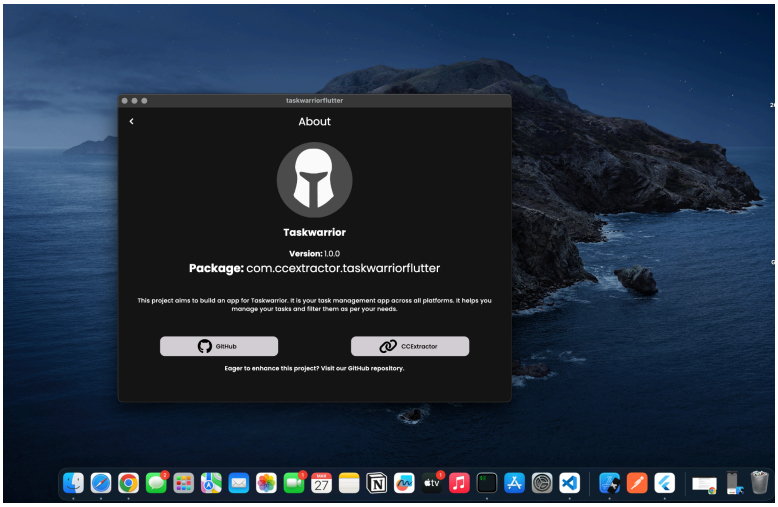
Prerequisite Task:

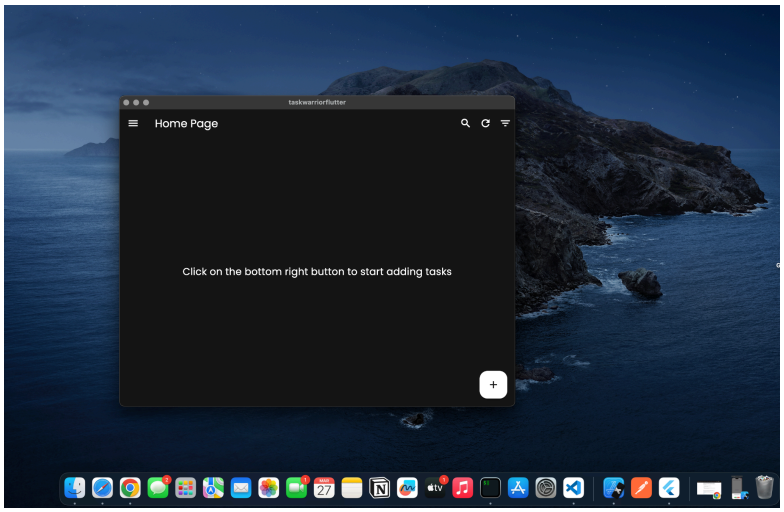
I have completed the **Compile Flutter on Mac** Take-home qualification task. I have prepared distributables for the taskwarrior-flutter app on mac. I've created a convenient DMG file for the Taskwarrior app. The user can download it from the repository provided below and start using Taskwarrior instantly, without any hassle.

Github Link: https://github.com/oops-shlok/taskwarrior-flutter-macOS-distributable
Installation video and screenshots:
https://drive.google.com/drive/folders/1sL0rlX37ngX4PF7Noh3G-n2aOHpeyAsL?usp=sharing

Screenshots:







Contributions to ultimate_alarm_clock Project:

Issues:

- #448 - [Bug] Compatibility Issue with Firebase/CoreOnly on iOS Installation for Mac M1
- #494 - [Bug] Timer Functionality Issue When App is Backgrounded
- #480 - [Bug] Next Alarm Showing Countdown Instead of 'No Upcoming Alarms' on iOS device

Pull Requests:

- #458 - [fix] iOS podfile and podfile.lock version compatibility issue
- #496 - [fix] Timer Functionality Issue When App is Backgrounded
- #481 - [fix] Next alarm showing 'No Upcoming alarm' instead of countdown on iOS device

Contributions to taskwarrior-flutter Project:

Issue:

- #335 - [Bug] MissingPluginException for Permission Status Check on macOS Installation

Pull Request:

- #336 - [fix] macOS build due to permission handler plugin

More about me:

I am a final year student at National Institute of Technology, Tiruchirappalli, Tamil Nadu. A software domain head and treasurer at Spider R&D, the official research and development club of NIT Trichy. I did my summer internship last year at Standard Chartered GBS, Chennai as a SDE Intern.

I like to work, travel, binge watch and listen to music. I enjoy software development and working on challenging projects. I have been involved in software development and have grown to like open source from the past year, because of its vibrant and engaging community. I actively use open-source software and like to contribute back when I can.

I constantly strive to improve my skills, and find new ways of thinking & problem solving to stay on top of recent technological advancements. I prefer to give more importance to practical implementations rather than theoretical studies. I believe in solving problems by reusing the latest technological stuff, rather than reinventing the wheel.

Communication:

I'm comfortable with any of the communication mediums. I can work full-time on weekdays and am usually available between **12 PM IST (6:30 AM UTC) to 2 AM IST (8:30 PM UTC)**. On weekends, I would love to spend time communicating with the team to learn from them while working on whatever issues occur at that time.

I'll also responsibly keep my mentor updated in case of any emergency that occurs with relevant details.

Post GSoC:

If there are things left unimplemented (like documenting the codebase or testing or anything other feature), I would strive to complete them post-GSoC and will keep contributing to the enhancement of this project.

Thanks