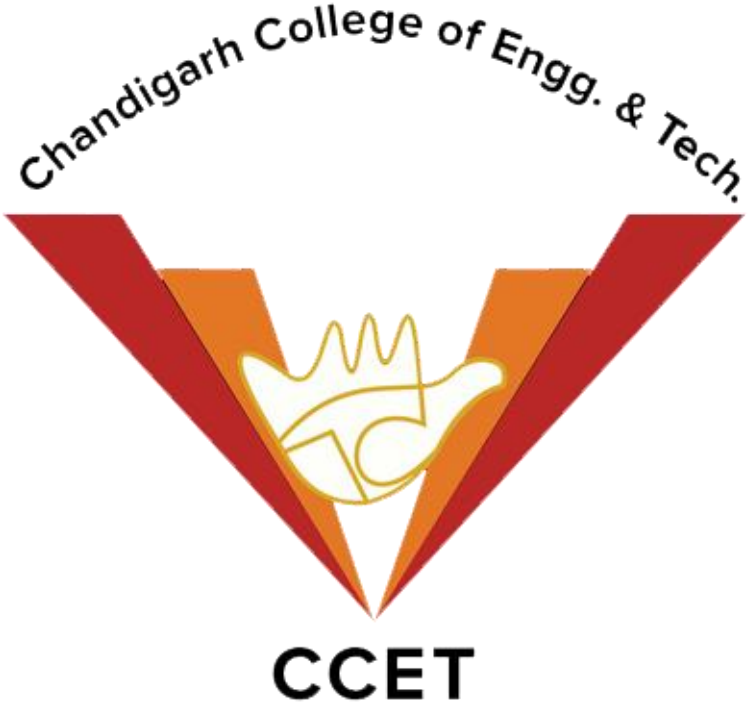
Software Requirements Specification (SRS)

Ultimate Shake Alarm App



Submitted To:

Dr. Santosh Kumar Yadav Lecturer

CSE DEPARTMENT

Submitted By:

Vinay Sharma

6TH SEMESTER

9531/22

Table of Contents

Software Requirements Specification Ultimate Shake Alarm App

1. Introduction

1.1 Purpose

The Ultimate Shake Alarm App is designed to provide users with a reliable and effective wake-up system by integrating interactive alarm dismissal methods. Unlike traditional alarm clocks, which can be snoozed or dismissed with a simple button press, this application requires the user to complete specific physical or mental challenges before the alarm can be turned off.

This document provides a detailed functional and non-functional specification for the Ultimate Shake Alarm App, ensuring that all development, testing, and deployment aspects are clearly defined. It serves as a guide for developers, testers, project managers, and stakeholders.

1.2 Scope

The application provides multiple alarm dismissal methods, including:

- Shake-based dismissal, which requires the user to shake the phone with a specific intensity.
- QR Code Scan, where the user must scan a pre-configured QR code to turn off the alarm.
- Math Puzzle, which requires the user to solve an arithmetic equation to stop the alarm.
- Typing Challenge, where the user must type a predefined phrase correctly.
- Step Counter, which requires the user to walk a certain number of steps before the alarm stops.
- Voice Command, where the user must say a specific phrase out loud.
- GPS-Based Movement, which requires the user to move to a predefined location.

The application ensures accurate alarm scheduling, data persistence, and a customizable user experience through intuitive UI and background services.

1.3 Definitions, Acronyms, and Abbreviations

- SRS refers to Software Requirements Specification.
- UI stands for User Interface.
- API means Application Programming Interface.
- GPS refers to Global Positioning System.
- MVVM is Model-View-ViewModel Architecture.
- Room DB is Android’s Room Database for local storage.

1.4 References

- Android Developer Documentation (developer.android.com)
- Java Official Documentation (docs.oracle.com)
- Room Database Documentation (developer.android.com/training/data-storage/room)
- Material Design Guidelines (material.io)

1.5 Overview

This document provides a comprehensive description of the Ultimate Shake Alarm App, covering system functionality, performance expectations, security considerations, and future enhancements. The goal is to ensure a structured approach for designing, developing, and deploying the application.

---

2. Overall Description

2.1 Product Perspective

The Ultimate Shake Alarm App is a standalone Android application that leverages device sensors such as the accelerometer, microphone, and GPS to ensure the user is fully awake before dismissing the alarm.

The app will integrate with:

- Android Alarm Manager for precise alarm scheduling.
- Room Database for storing user preferences and alarms.
- ZXing Library for QR code scanning.
- Google Speech API for voice command recognition.

2.2 Product Functions

The application provides the following core functionalities:

- Alarm Scheduling, allowing users to create, edit, and delete alarms.
- Multiple Dismissal Methods, requiring users to complete predefined challenges before dismissing the alarm.
- User Preferences, where users can select alarm tones, snooze duration, and UI themes.
- Background Execution, ensuring alarms function even when the app is closed.
- Data Persistence, where alarms and settings are stored locally using Room Database.

2.3 User Characteristics

This application is designed for:

- Students and Professionals who need a strict wake-up routine.
- Heavy Sleepers who often ignore or snooze alarms.
- Fitness Enthusiasts who prefer movement-based dismissals.

2.4 Constraints

The application must adhere to the following constraints:

- It requires Android 6.0 or later.
- It must comply with Google Play Store policies.
- Background execution permissions are necessary for reliable alarm scheduling.

2.5 Assumptions and Dependencies

- The device must have working sensors such as an accelerometer, microphone, and GPS.
- Users must grant necessary permissions for alarm-related features.
- The app assumes a stable internet connection for QR code validation and GPS-based dismissals.

---

3. Specific Requirements

This section outlines the functional and non-functional requirements necessary for the Ultimate Shake Alarm App to operate effectively.

3.1 Functional Requirements

3.1.1 Alarm Scheduling

- The app must allow users to create, edit, delete, and duplicate alarms.
- Users must be able to set one-time or recurring alarms, including daily, weekdays, weekends, or custom schedules.
- The app must store alarm data persistently in Room Database to ensure alarms remain scheduled even after restarting the device.

3.1.2 Dismissal Methods

- Users must select at least one dismissal method when setting up an alarm.
- The app must validate and confirm successful completion of the dismissal challenge before stopping the alarm.
- Available dismissal methods include:
    - Shake-based dismissal, where the user must shake the phone at a specific intensity level.
    - QR Code Scan, where the user must scan a pre-configured QR code.
    - Math Puzzle, requiring the user to solve a randomly generated arithmetic problem.
    - Typing Challenge, where the user must type a predefined phrase correctly.
    - Step Counter, requiring the user to walk a set number of steps before dismissal.
    - Voice Command, where the user must say a predefined phrase detected by voice recognition.
    - GPS-Based Movement, requiring the user to move to a specific predefined location.

3.1.3 Background Execution

- Alarms must ring even when the app is closed or running in the background.
- The app must request exact alarm permissions to ensure timely alarm triggers.
- Background services should be optimized for battery efficiency to prevent excessive power consumption.

3.1.4 User Preferences

- Users must be able to customize alarm tones, selecting from preloaded sounds or custom ringtones.
- Users must be able to adjust snooze duration with options like 5, 10, or 15 minutes.
- The app must support both light and dark mode to enhance user experience.

3.1.5 Notification and Widget Support

- The app must display a persistent notification when an alarm is ringing.
- Users should be able to snooze or dismiss the alarm directly from the notification panel.
- The app should support home screen widgets for quick alarm access and management.

3.2 Non-Functional Requirements

3.2.1 Performance

- The alarm must trigger within 500 milliseconds of the scheduled time.
- The app must process sensor inputs (shake detection, step counting, voice recognition) within 300 milliseconds.
- The background service must consume less than 3% of the device's battery per hour.

3.2.2 Usability

- The app must have an intuitive and user-friendly interface following Material Design Guidelines.
- It should support both portrait and landscape orientations.
- Dismissal challenges should provide real-time feedback, helping users know whether they have successfully completed the task.

3.2.3 Reliability

- The app must ensure 99.9% uptime for alarm triggering.
- It should gracefully handle failures such as sensor malfunctions by providing alternative dismissal methods.

3.3 External Interface Requirements

3.3.1 User Interfaces

The app must include the following screens:

- Alarm List Screen, displaying all scheduled alarms.
- Alarm Creation Screen, allowing users to set the alarm time, sound, and dismissal method.
- Alarm Dismissal Screen, presenting the required challenge to turn off the alarm.
- Settings Screen, allowing users to configure preferences such as themes, default sounds, and snooze durations.

3.3.2 Hardware Interfaces

- The app must use the device’s accelerometer to detect shake gestures.
- The app must use the microphone for voice command dismissal.
- The app must utilize the device's GPS module to validate movement-based dismissal.

3.3.3 Software Interfaces

- The app must integrate with Android's Alarm Manager API for scheduling alarms.
- It must use the ZXing Library for QR code scanning.
- It must incorporate the Google Speech API for voice recognition.

3.4 Data Requirements

3.4.1 Alarm Data Storage

- Each alarm must store details such as time, dismissal method, sound, and snooze duration.
- Alarms should persist across app restarts and system reboots.

3.4.2 User Preferences Storage

- The app must save user preferences such as selected theme (light/dark), default snooze time, and preferred alarm sound.

---

4. System Architecture

4.1 Architectural Overview

The Ultimate Shake Alarm App follows the Model-View-ViewModel (MVVM) architecture to separate concerns and improve maintainability. The system is divided into three layers:

- Model Layer: Handles data storage using Room Database. It manages alarm persistence and user settings.
- ViewModel Layer: Contains business logic for alarm scheduling and dismissal validation. It acts as a bridge between the UI and data.
- View Layer: Displays UI components such as alarm lists, settings, and challenge screens. It handles user interactions and updates based on ViewModel data.

4.2 Data Flow Description

1. The user schedules an alarm through the UI.
2. The alarm details are stored in the Room Database.
3. At the scheduled time, Android's Alarm Manager triggers the alarm.
4. The alarm dismissal screen appears, requiring the user to complete a challenge.
5. If the challenge is completed successfully, the alarm stops ringing and updates the log.
6. If unsuccessful, the alarm continues ringing until the user retries and succeeds.

4.3 Component Overview

- User Interface Components: Includes alarm list, settings, and dismissal screens.
- Business Logic Layer: Handles scheduling, dismissal validation, and user settings management.
- Data Storage Layer: Uses Room Database to store alarm data and SharedPreferences for user preferences.

4.4 Database Storage

Alarm Data

- The system must store details such as alarm time, dismissal method, and selected alarm tone.
- Data should persist even after the app is restarted or the phone reboots.

User Preferences

- The system must store user-selected themes, snooze duration, and default alarm sounds in SharedPreferences.

4.5 API Integration

- The app must use the Alarm Manager API to schedule and trigger alarms.
- It must use the Sensor API to handle shake detection and step counting.
- It must integrate the ZXing Library for QR code scanning.

---

5. Use Case Scenarios

This section describes various scenarios in which users interact with the Ultimate Shake Alarm App.

5.1 Use Case: Creating an Alarm

Actors: User

Description: The user wants to create an alarm with a specific time, sound, and dismissal method.

Flow of Events:

1. The user opens the app and navigates to the "Create Alarm" screen.
2. The user sets the desired time and date for the alarm.
3. The user selects an alarm sound (default or custom).
4. The user chooses one or multiple dismissal methods (Shake, QR Code, Math Puzzle, etc.).
5. The user adjusts snooze settings and repeat options (daily, weekdays, weekends, etc.).
6. The user taps "Save," and the alarm is stored in the Room Database.

Expected Outcome:

- The alarm is successfully created and appears in the alarm list.
- The user receives confirmation feedback via a notification or message.

5.2 Use Case: Alarm Rings & User Dismisses It

Actors: User

Description: At the scheduled time, the alarm rings. The user must complete the selected dismissal challenge to turn it off.

Flow of Events:

1. The alarm notification is triggered at the scheduled time.
2. The wake-up screen opens, displaying the alarm name and dismissal challenge.
3. The user performs the selected dismissal method (shake phone, scan QR code, solve math puzzle, etc.).
4. The system validates the input and confirms if the challenge is completed successfully.

5. If successful, the alarm stops, and the log is updated.
6. If unsuccessful, the user must retry until successful.

Expected Outcome:

- The alarm stops only after successful dismissal.
- If the user fails the challenge, the alarm keeps ringing until correctly dismissed.

5.3 Use Case: Editing an Alarm

Actors: User

Description: The user wants to modify an existing alarm.

Flow of Events:

1. **The user opens the alarm list and selects an existing alarm.**
2. **The user edits the time, sound, or dismissal method.**
3. **The user taps "Save" to apply the changes.**

Expected Outcome:

- **The updated alarm is successfully saved and displayed in the alarm list.**

5.4 Use Case: Deleting an Alarm

Actors: User

Description: The user wants to remove an alarm from the app.

Flow of Events:

1. **The user navigates to the alarm list.**
2. **The user selects an alarm and taps "Delete."**
3. **The app confirms deletion with a pop-up dialog.**
4. **The user confirms, and the alarm is removed from the database.**

Expected Outcome:

- **The alarm is successfully deleted and no longer appears in the alarm list.**

5.5 Use Case: Snoozing an Alarm

Actors: User

Description: The user chooses to snooze the alarm instead of dismissing it.

Flow of Events:

1. **The alarm rings, and the wake-up screen appears.**
2. **The user taps the "Snooze" button.**
3. **The alarm stops temporarily and reschedules itself based on the snooze duration.**

Expected Outcome:

- **The alarm pauses and reactivates after the snooze duration ends.**
- **A snooze limit prevents excessive usage.**

5.6 Use Case: Backup and Restore Alarms

Actors: User

Description: The user wants to back up and restore alarms.

Flow of Events:

1. **The user navigates to "Settings" and selects "Backup Alarms."**
2. **The app exports alarm data to a local file or cloud storage.**
3. **Later, the user selects "Restore Alarms."**
4. **The app retrieves the backup file and restores the alarms.**

Expected Outcome:

- **Alarm data is successfully backed up and restored when needed.**

---

6. Performance Requirements

Performance requirements define how efficiently the Ultimate Shake Alarm App should function under normal and extreme conditions.

6.1 Response Time

- The app must launch within **2 seconds** under normal conditions.
- Alarm activation must occur **within 500 milliseconds** of the scheduled time.
- The system must process **shake detection, QR scanning, step counting, and voice recognition within 300 milliseconds** to provide real-time feedback.
- Dismissal validation must complete **within 1 second** after the user completes a challenge.

6.2 Resource Utilization

- The app should **consume less than 3% battery per hour** while running in the background.
- CPU usage should not exceed **15%** while actively monitoring sensor-based dismissals.
- The app should not exceed **50MB** of RAM usage during peak performance.

6.3 Scalability

- The app should support **up to 100 scheduled alarms per user** without performance degradation.
- The app should efficiently handle **multiple alarms ringing simultaneously** without crashes or UI slowdowns.

6.4 Reliability

- The app must ensure **99.9% uptime** for alarm triggering and dismissal functionality.
- The app should handle **low battery situations** by reducing background sensor usage while maintaining alarm accuracy.
- Alarm notifications must work **even if the app is closed or the device is restarted**.

---

7. Security Requirements

Security is crucial to protect **user data, prevent unauthorized dismissals, and ensure alarm integrity**.

7.1 Data Security

- All **alarm data, user preferences, and challenge completions** must be securely stored in **Room Database** with encryption where applicable.
- User data should **never be shared with third-party services** unless explicitly allowed by the user.

7.2 User Authentication

- If implemented, cloud backup and restoration must require **user authentication** (e.g., Google Sign-In).
- If the app introduces **profile-based alarms**, user credentials must be stored securely using **hashed and salted encryption**.

7.3 Alarm Integrity

- The app must prevent **force-stopping the app from dismissing an active alarm**.
- Background restrictions from Android should not interfere with alarm triggering.
- QR code dismissals should be **encrypted and unique per user** to prevent spoofing.

7.4 Privacy Considerations

- The app must request **only necessary permissions** (e.g., microphone for voice recognition, location for GPS-based dismissal).
- Users should be able to **review and revoke permissions** at any time from the settings.
- Any optional cloud-based backup service should be **opt-in only**.

---

8. Risks and Mitigation

This section identifies potential risks associated with the Ultimate Shake Alarm App and proposes mitigation strategies.

8.1 Sensor Malfunctions

- **Risk:** The device's **accelerometer, microphone, or GPS may fail**, making shake-based or location-based dismissal methods unreliable.
- **Mitigation:** The app must allow **alternative dismissal methods** (math puzzle, typing challenge, etc.) as a fallback option.

8.2 Background Service Restrictions

- **Risk:** Android **may restrict background services**, causing the alarm to fail if the app is not active.
- **Mitigation:** The app must use **Foreground Services** and request **Exact Alarm Permissions** to ensure reliability.

8.3 High Battery Consumption

- **Risk:** Running multiple sensors for dismissal detection may cause **excessive battery drain**.
- **Mitigation:** Optimize battery usage by **disabling unnecessary sensors** when not in use and allowing users to configure sensor sensitivity.

8.4 Alarm Bypassing

- **Risk:** Users might try to **bypass the alarm** by closing the app forcefully.
- **Mitigation:** Implement a **wake-lock feature** to ensure the alarm continues running even if the app is force-closed.

8.5 App Crash During Alarm

- **Risk:** If the app crashes during an active alarm, the alarm may not trigger.
- **Mitigation:** Implement an **auto-restart mechanism** that reopens the app and reschedules alarms if an unexpected crash occurs.

---

9. Future Enhancements

The Ultimate Shake Alarm App has the potential for future improvements and additional features based on user feedback.

9.1 Cloud Backup and Sync

- Allow users to **sync alarms across multiple devices** using **Google Drive or Firebase Cloud Storage**.
- Enable automatic **cloud backups** to restore alarms after reinstalling the app.

9.2 Wearable Integration

- Enable alarm notifications on **smartwatches and fitness bands**.
- Allow users to dismiss alarms using a **shake gesture on their smartwatch**.

9.3 AI-Based Dismissal Challenges

- Implement **AI-generated math problems** that adjust difficulty based on user performance.
- Introduce **adaptive shake detection** that requires more effort if the user repeatedly snoozes alarms.

9.4 Multi-Language Support

- Add support for **multiple languages** to expand the app's reach worldwide.
- Enable **voice recognition for multiple languages** in the voice dismissal method.

9.5 Sleep Pattern Analysis

- Integrate **sleep tracking** features to analyze user wake-up patterns and suggest optimal alarm times.
- Provide insights into **wake-up efficiency based on past alarm dismissals**.

---

10. Conclusion

The **Ultimate Shake Alarm App** is a **feature-rich, sensor-based alarm system** designed to **prevent oversleeping and enhance user engagement**. The app ensures **alarm persistence**, **customizable dismissal methods**, and **user-friendly settings** to offer an effective wake-up solution.

This **SRS document** outlines the app's **functional requirements, performance expectations, security measures, and future enhancements**. By following this specification, the development team can ensure that the app meets the highest standards in usability, reliability, and efficiency.