ROVINJ SECURITY DEVELOPER GUIDE

VERSION 1.0

1.0 Introduction and Background

1.1 Introduction

Roving Security, a cutting-edge Android application developed on Android Studio and seamlessly integrated with the Firebase platform, redefines security management paradigms. The motivation behind this application stems from the need to address the limitations observed in existing course management software. Unlike conventional GPS-based tracking systems, Roving Security ingeniously employs QR codes to precisely determine the location of guards, enhancing the overall efficiency and reliability of security protocols.

This feature-rich application enables guards to submit comprehensive reports, complete with image uploads, providing a holistic view of on-site situations. The administrative hierarchy comprises main admins and sub-admins, each entrusted with specific responsibilities, such as managing guards, QR codes, viewing reports, and overall system administration.

1.2 Objectives

The primary objectives of Roving Security are multifaceted, aiming to establish a comprehensive and user-friendly security management system. These objectives include:

- **QR Code Location Tracking**: Utilizing QR codes to pinpoint the exact location of guards, offering a reliable and efficient alternative to traditional GPS systems.
- **Comprehensive Reporting System**: Empowering guards to submit detailed reports, complete with image attachments, ensuring a thorough and accurate account of any unusual incidents.
- **Administrator Functionalities**: Distinguishing between main admins and sub-admins, each equipped with specific privileges to manage users, QR codes, reports, and other critical aspects of the system.
- **Intuitive User Interface**: Designing a user interface with a minimal learning curve, prioritizing ease of use and accessibility for guards and administrators alike.

1.3 Risks

While Roving Security introduces innovative solutions, certain risks accompany its current implementation. The application heavily relies on an exception-based error-handling strategy, which, while effective, may pose memory usage challenges. Suggestions for an alternative boolean-based error-handling approach are presented to mitigate potential memory-intensive issues. Careful consideration of these risks is essential for maintainers.

2.0 Architecture

2.0 Software Architecture

Roving Security boasts a meticulously designed software architecture, opting for a layered model to ensure modularity and scalability. While resembling aspects of the MVC pattern, the application explicitly adopts a Layered Architecture, incorporating distinct layers such as:

- **Presentation Layer:** Housing user interfaces tailored for guards and admins, providing an intuitive and seamless user experience.
- **Logical Layer:** Encompassing core functionalities and business logic to orchestrate intricate operations within the application.
- **Data Layer:** Facilitating real-time data synchronization through interaction with Firebase, ensuring the availability of up-to-date information.

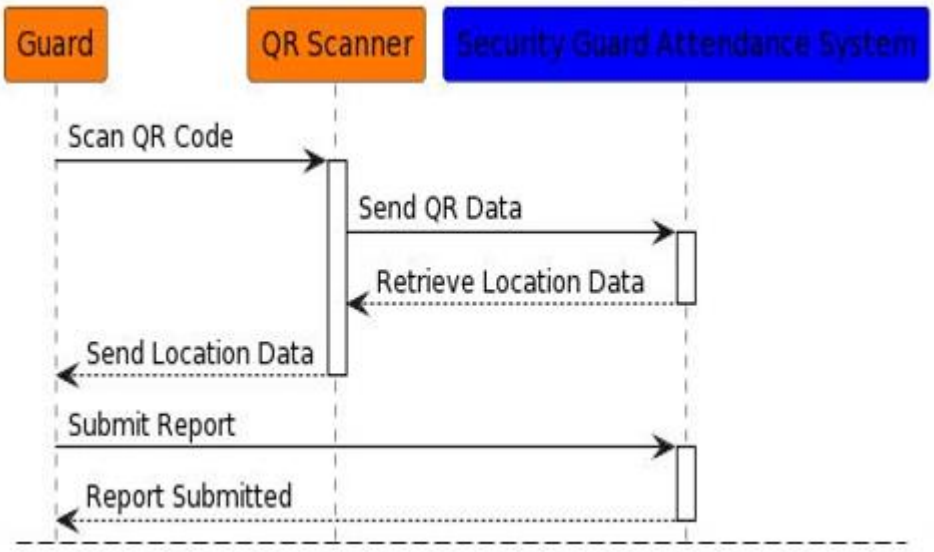


Figure 1: Roving Security Sequence Diagram

2.1 Database Design

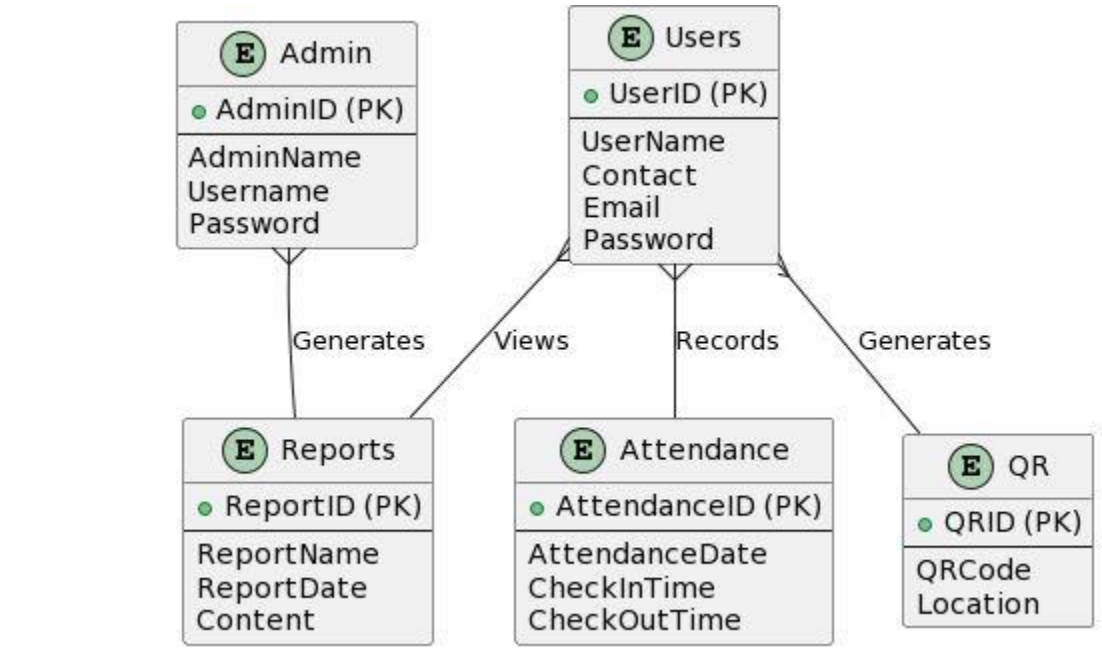
Roving Security's database design adheres to a meticulous relational model approximating third-normal form. Key tables, including Users, QR Codes, Reports, and Admins, are strategically crafted to ensure efficient data storage and retrieval, fostering a robust foundation for the application.

Tables:

- Users: Centralized user information, essential for user authentication and role-based access control.
- QR Codes: Repository of QR codes, pivotal for location tracking of guards.
- Reports: Comprehensive records of incident reports submitted by guards, including associated images.
- Admins: Information repository for system administrators, categorizing main admins and sub-admins.

2.2 Class Level Design

Roving Security's class-level design is organized into distinct folders, each serving a specific purpose and contributing to the application's overall functionality and maintainability.



Cache

- **CacheItem:** Implementation of a versatile key-value pair cache item, offering flexibility in data storage.
- **MemoryCache:** In-memory cache implementation with an aging mechanism, contributing to efficient data management.

Models

- **Guard**: Comprehensive representation of guard information, encapsulating crucial details such as identification and status.
- **Admin**: Model designed to capture and manage administrative information, facilitating a structured hierarchy within the system.

Managers

- **QRCodeManager:** Adept manager handling QR code-related functionalities, ensuring smooth operations in QR code generation and tracking.
- **ReportManager:** Orchestrating the submission and management of reports, contributing to enhanced situational awareness.
- **AdminManager:** Managing various administrative functionalities, catering to the specific needs of both main admins and sub-admins.

3.0 Software Interfaces

Roving Security seamlessly interfaces with external tools and packages, creating a robust development environment for maintainers.

- **Android Studio:** The official Integrated Development Environment (IDE) for Android app development, offering a powerful and intuitive platform for code development and debugging.
- **Firebase:** A real-time database serving as the backend, facilitating efficient data storage, retrieval, and synchronization, crucial for real-time updates.
- **ZXing Library:** An essential tool employed for QR code generation and scanning, enhancing the application's tracking capabilities and ensuring the accuracy of location data.

Maintainers must prioritize the correct installation and integration of these tools to ensure a smooth and efficient development process.

4.0 Conclusion

4.1 Remarks on Implementation

This section reflects on the current implementation of Roving Security, available on GitHub. The application, estimated to be approximately 95% complete, successfully incorporates functionalities outlined in the Requirements Specification. However, certain features, such as advisor overrides for prerequisites and complete theming, remain works in progress. The layout of the user interface, though partially meeting the Design Specification, demands further attention.

Considering time constraints, implementation decisions, particularly in performance-critical areas, may not be optimal. The developer has made commendable efforts to provide comments for every method, though some may be missing.

A few test accounts are available for evaluation, accessible through the login page, featuring varying roles such as Administrator, Instructor, and Student.

4.2 Possible Future Improvements

Roving Security presents ample opportunities for refinement and enhancement:

- **Completion of Developer Comments:** Ensure comprehensive understanding and ease of maintenance by completing developer comments for all methods, providing valuable insights into each component's functionality.

- **GUI Layout Enhancement:** Thorough GUI layout completion or redesign for an improved and aesthetically pleasing user interface, aligning with the application's functionality.

- **DatabaseManager Refactoring:** Strategic refactoring of the DatabaseManager class for improved code organization and enhanced maintainability, ensuring a streamlined approach to database interactions.

- **Method Refactoring:** Simplification and refactoring of complex methods within manager classes to achieve a simpler, shorter form, enhancing code readability and maintainability.

- **Sequence Diagrams Update:** Update sequence diagrams to accurately reflect the current system state, providing an invaluable resource for future developers to understand system interactions.

- **Database Transition:** Move the database to a full-fledged SQL Server instance or Azure instance for centralized data management, scalability, and independent development.

- **Final Grades Implementation:** Add the ability to assign final grades for students in a course, addressing an essential feature for academic tracking within the application.

By meticulously addressing these points, maintainers can elevate the performance, stability, and user experience of the Roving Security application, ensuring its continued evolution and effectiveness in the dynamic landscape of security management applications.