National University
Of Computer and Emerging Sciences

SmartKnock

A Smart Doorbell System

by TechForge.

CS-3004

SDA

Submitted by:

- Emaan Fatima 22i-0869
- Aden Sial 22i-1313
- Abdul Rehman 22i-0785

"Documentation"
Semester Project

Table of Contents

Scope:

Smart Knock is a mobile application developed by Tech Forge for homeowners looking for a smart and efficient way to monitor visitors.

Following are the **features** of Smart Knock:

- **Platform Development:** A mobile application that supports Android platforms.
- **Real-Time Notifications:** Users will receive notifications whenever the smart bell is pressed, allowing them to remain connected to their home.
- **Image Capturing:** When someone presses the bell, the system captures an image of the visitor and sends it to the user's device.

There are existing products in the market, such as Ring and Nest Hello, but Smart Knock focuses on offering enhanced features like image capturing. Smart Knock is an affordable solution, designed to provide advanced home security features without the high cost typically associated with similar smart home products.

Objective:

The primary objective of the Smart Knock project is to create an intuitive, user-friendly application that:

- Notifies users in real time when the smart bell is pressed.
- Captures and sends images of the visitor to the user.

Problem Statement and Description:

With the increasing need for home security, homeowners are seeking convenient and smart solutions to monitor their front doors, especially when they are away. Traditional doorbells lack the ability to notify users when they are not home, resulting in missed deliveries, guests, or potential security threats. Smart Knock addresses this problem by integrating real-time notifications and image capturing to keep users connected to their home from anywhere.

Smart Knock aims to replace outdated doorbells with an internet-enabled solution that not only notifies users immediately but also captures essential information like visitor photos. This project is feasible due to the widespread availability of smartphones and internet access, along with the growing interest in IoT-enabled smart home systems. By focusing on ease of use and customization, Smart Knock offers a unique and accessible product to improve home security.

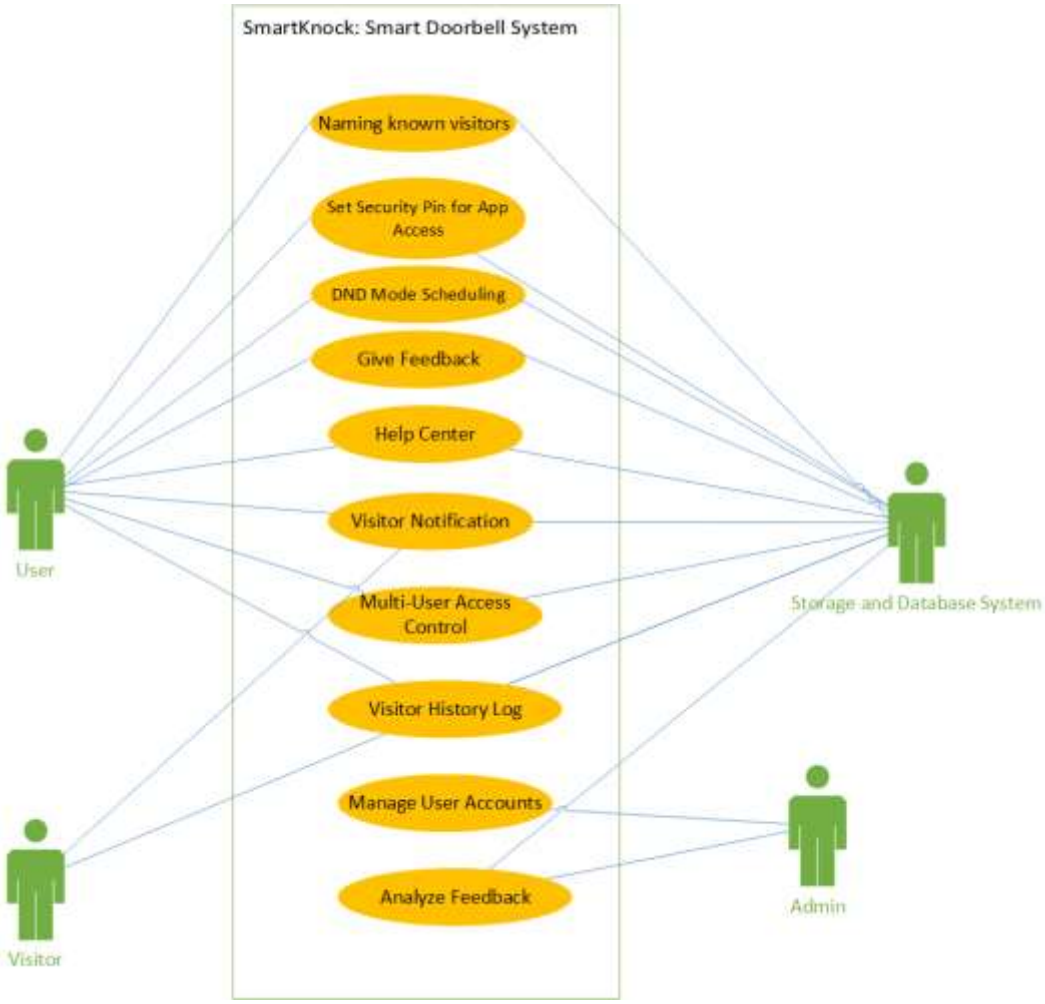
Functional Requirements:

1. **User Management:** Users can create and manage accounts, set up secondary users, and customize notification preferences with secure login options.
2. **Visitor Management**: The system logs visitor events with images and timestamps.
3. **Notifications**: Notifications are sent for doorbell presses.
4. **Feedback System:** Users can submit feedback through the app, which is stored for analysis and future product improvements.

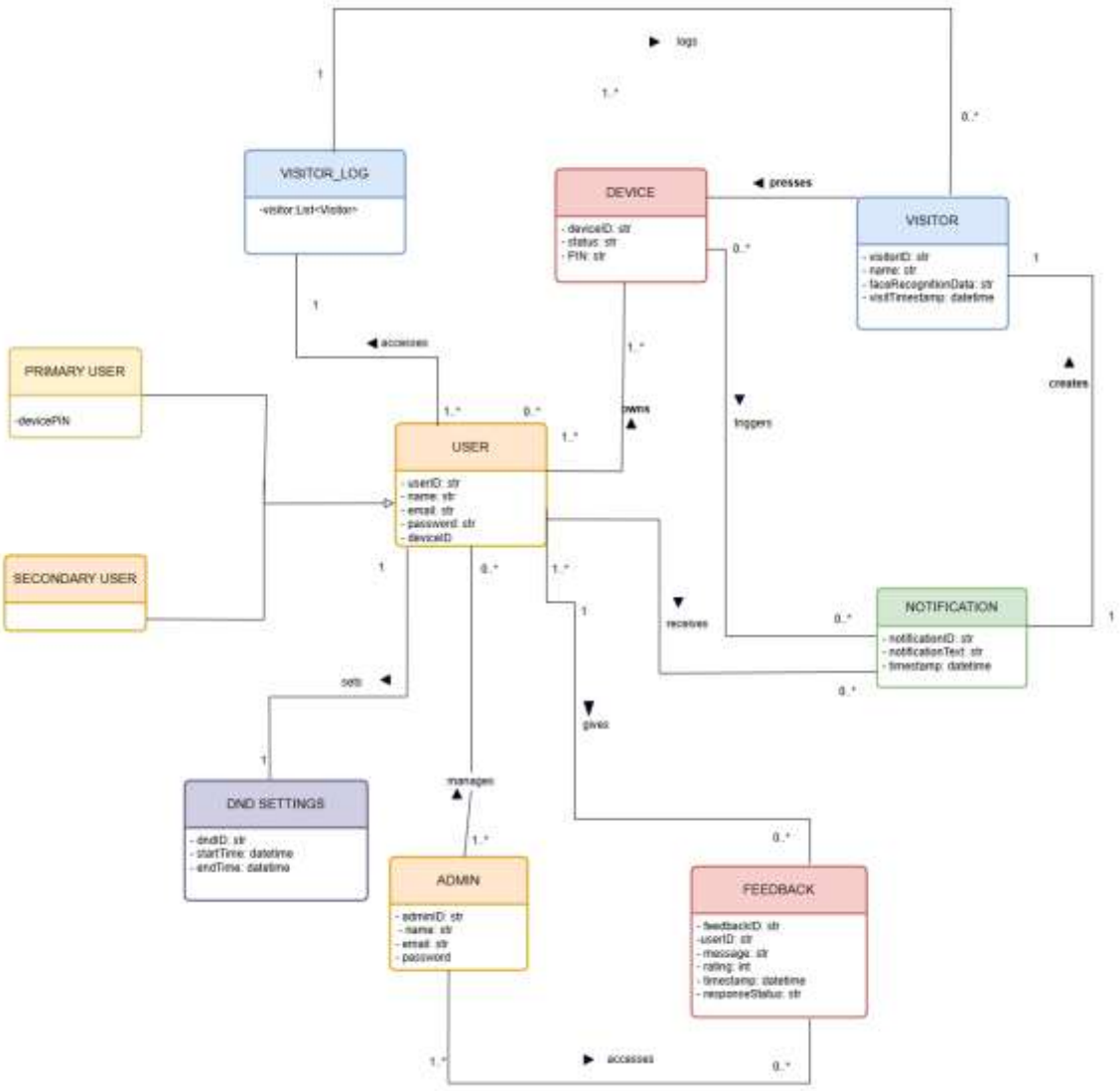
Non-Functional Requirements:

1. **Reusability:** Ensures components like the notification system or authentication can be reused across the system or in other projects, saving time and effort.
2. **Scalability:** Allows the system to handle more devices, users, and data without degrading performance.
3. **Modularity:** Divides the system into independent modules, enabling easier maintenance, testing, and updates without affecting the entire system.
4. **Extensibility:** Facilitates adding new features or integrating with other systems without significant changes to the system's core architecture.

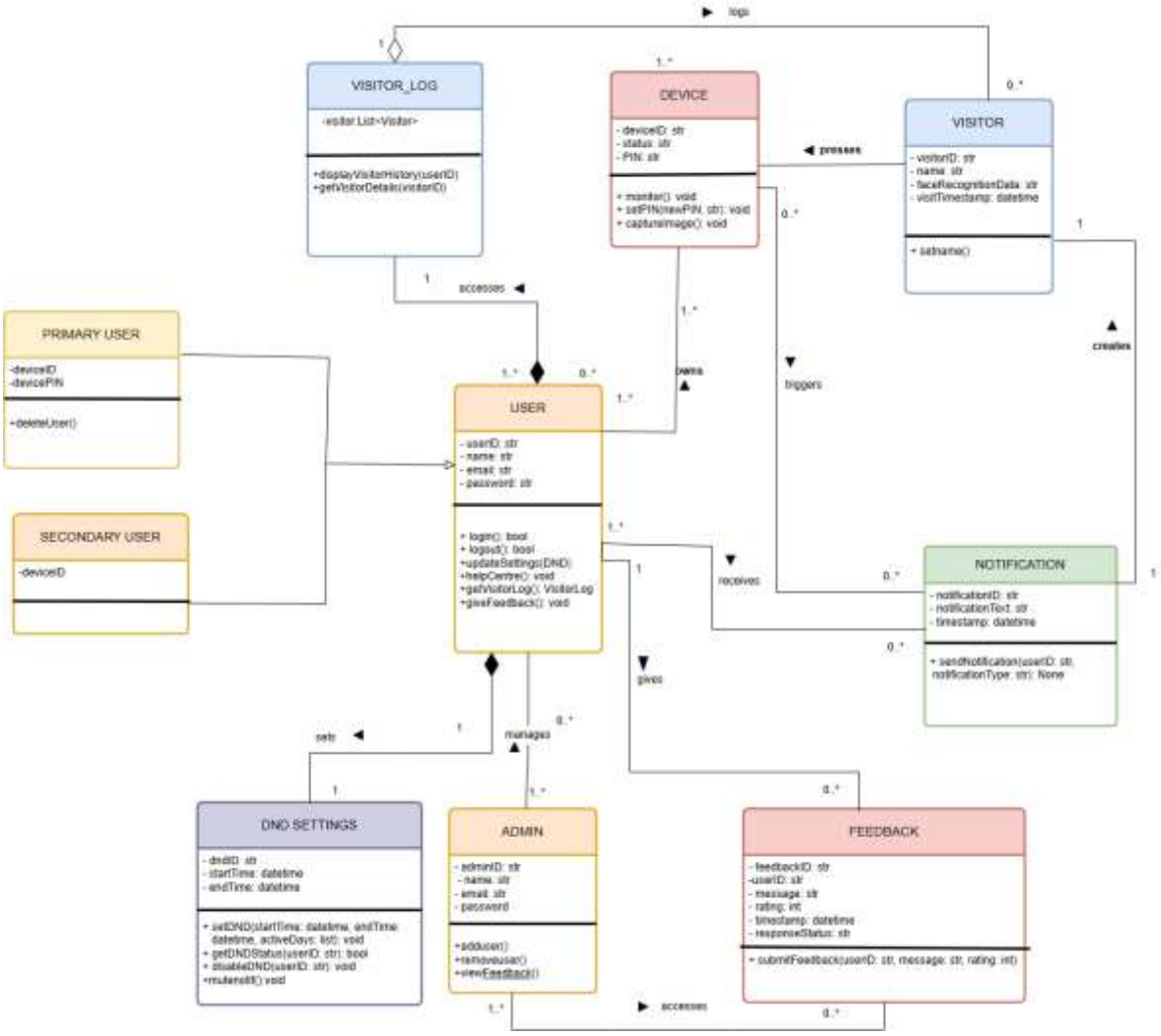
Use Case Diagram:



Domain Model:



Class Diagram:



Fully Dressed Use Cases:

1. Do Not Disturb (DND) Mode Scheduling

Scope: Smart Knock smart doorbell system
Level: User goal
Primary Actor: User (Homeowner)
Stakeholders and Interests:

- User (Homeowner): Wants to avoid disturbances during specific hours.
- Storage System: Ensures that the DND feature functions correctly, logging events without notifying the user.

Preconditions:

- The Smart Knock app is installed and functioning correctly.
- The user has enabled notifications in the app settings.
- The user is logged in to the app and has access to settings.
- The DND mode feature is available and operational.

Postcondition:

- Notifications are suppressed during the chosen time range.
- Visitors during DND hours are logged but not alerted to the user in real-time.
- The user can see the log in application events afterward.

Main Success Scenario:

Actor's Actions	System's Responses
1. The user opens the Smart Knock app and navigates to settings.	2. The system displays DND settings page.
3. The user selects the time range during which notifications should be suppressed.	4. The system validates the selected time range for correctness.
5. The user saves the settings.	6. The system saves the DND schedule and confirms successful configuration.
	7. The system suppresses notifications during the specified hours.
	8. The system continues to log events such as doorbell presses.
9. The user accesses the visitor log to review missed alerts.	10. The system displays a log of all visitors during the DND time range.

	11. Notifications automatically resume outside the DND time range.

Extensions:

3a. User Selects Invalid Time Range:

Actor's Actions	System's Responses
1. The user selects an invalid time range (e.g., start time after end time).	2. The system displays an error message indicating the time range is invalid.
3. The user enters a valid time range.	4. The system validates and accepts the updated time range.

6a. System Fails to Save Settings:

Actor's Actions	System's Responses
1. The user attempts to save the DND settings.	2. The system encounters an error while saving the settings.
	3. The system displays an error message prompting the user to try again.
4. The user retries saving the settings.	5. The system saves the settings successfully after retrying.

5a. User Changes DND Schedule Midway:

Actor's Actions	System's Responses
1. The user accesses the DND settings again during an active DND period.	2. The system displays the current DND schedule settings.
3. The user adjusts the time range.	4. The system validates the new time range for correctness.
5. The user saves the updated schedule.	6. The system applies the updated schedule immediately.

Special Requirements:

- The system must ensure that notifications resume immediately once the DND time range ends.
- The event log must remain accessible to the user at all times, regardless of the DND settings.

Technology and Data Variations List:

- The user may receive notifications depending on app settings, but these will be suppressed during DND hours.
- Events may be logged with timestamps, and the user should be able to filter or sort the log by date and time.

2. Help Center

Scope: Smart Knock smart doorbell system

Level: User goal

Primary Actor: User (Homeowner)

Stakeholders and Interests:

- User (Homeowner): Wants clear guidance on how to navigate the app and use the Smart Knock system effectively.
- Storage and Database System: Ensures the help center and tutorial provide accurate and up-to-date information.

Preconditions:

- The app is installed on the user's device, and the user is logged in.
- The app includes help section accessible to the user.
- The user is connected to the internet to access the FAQs.

Postconditions:

- The user understands how to navigate the app and set up the Smart Knock system.
- The user can troubleshoot common issues using the help center without external assistance.

Main Success Scenario:

Actor's Actions	System's Responses
1. The user accesses the Help Center.	2. The app displays all the FAQs to support the user.
3. The user finds the solution to his problem provided in the FAQ and resolves their issue.	

Extensions:

3a. User Cannot Find the Desired Information in the Help Center

Actor's Actions	System's Responses
1. The user searches for specific guidance in the Help Center but cannot find relevant information.	2. The app provides email to contact customer support.
3. The user sends a mail to that email address addressing the issue faced.	

Special Requirements:

- The tutorial and Help Center must be intuitive and easy to follow.

3. Manage User Accounts

Scope: Smart Knock smart doorbell system
Level: Admin goal
Primary Actor: Admin

Stakeholders and Interests:

- **Admin:** Wants to effectively manage user accounts or delete any account.
- **Users:** Expect their accounts to be created or deleted accurately and securely.
- **Storage and Database System:** Ensures that the system functions smoothly, preventing unauthorized access by checking username and password from the database.

Preconditions:

- The admin is logged into the user management panel with the necessary permissions.
- Users are already registered in the system.
- The system is connected to the database and user account management services.

Postconditions:

- User accounts are successfully created or deleted as requested.
- Any changes are updated in the system, and users are notified.

Main Success Scenario:

Actor's Actions	System's Responses

1. The admin logs into the Smart Knock.	2. The system authenticates the admin and grants access to the panel.
3. The admin goes to the User Management section.	4. The system displays a list of users with their details.
5. The admin adds a new user by entering the required details.	6. The system validates the details and updates the user list.
7. The admin removes an inactive or unauthorized user from the system.	8. The system validates the removal and updates the user list.
9. The admin confirms all changes.	10. The system reflects the updates in real-time.

Extensions:

6a. Invalid User Details Entered

Actor's Actions	**System's Responses**
1. The admin enters invalid details.	2. The system displays an error message indicating the issue.
	3. The system prompts the admin to correct the data.

6b. User Already Exists

Actor's Actions	**System's Responses**
1. The admin attempts to add a user with an email or username that already exists.	2. The system prevents duplication and notifies the admin that the user already exists.

8a. User Account In Use

Actor's Actions	**System's Responses**
1. The admin attempts to delete a user currently active in the system.	2. The system blocks the deletion and notifies the admin that the user is active.

Special Requirements:

- The system must handle a large number of user accounts efficiently.
- Proper validation of user data, especially for critical fields like email and password.

Technology and Data Variations List:

- The admin may access the user management panel via a web interface or mobile app, depending on their system setup.

- User details may include variations based on system requirements (e.g., additional fields like role).

4. Visitor Notification

Scope: Smart Knock smart doorbell system

Level: User goal

Primary Actor: User (Homeowner)

Stakeholders and Interests:

- **User (Homeowner):** Wants to be notified promptly when a visitor arrives, with relevant details such as the visitor's image.
- **Cloud and Storage System:** Ensures visitor images securely stored and efficiently processed for notification purposes.
- **Doorbell:** Detects the visitor press, captures the visitor's image, and triggers the notification system.

Preconditions:

- The Smart Knock doorbell is installed, connected to the user's network, and integrated with the mobile app.

Postconditions:

- The user receives a real-time notification on their mobile device with the visitor's image and timestamp.

Main Success Scenario:

Actor's Actions	System's Responses
1. A visitor presses the Smart Knock doorbell.	2. The system detects the press and triggers the visitor notification system.
	3. The system captures an image of the visitor.
	4. A notification is sent to the user's mobile device, including the visitor's image and timestamp.
5. The user views the notification and decides if he wants to name the visitor in the log for easy accessibility in future.	6. The system records the user's interaction and updates the database.

Extensions:

6a. Notification Fails to Send

Actor's Actions	System's Responses
1. The system encounters an error while sending the notification due to network issues.	2. The system stores the event data and retries sending the notification once connectivity is restored.
	3. The user is notified of the event soon.

Special Requirements:

- The system must ensure that the visitor's image is captured and sent within seconds of the doorbell press.
- The system must comply with privacy regulations, ensuring that visitor data is stored securely and not misused.

Technology and Data Variations List:

- The app may send notifications based on user preferences.

5. Review and Analyze Feedback

Scope: Smart Knock smart doorbell system

Level: Admin goal

Primary Actor: Admin

Stakeholders and Interests:

- Admin: Wants to monitor, analyze, and act on user feedback to improve the product and enhance customer satisfaction.

Preconditions:

- The admin has access to the feedback dashboard through the system.
- Users have submitted feedback through the app, including ratings, comments, and suggestions.

Postconditions:

- The admin gains actionable insights from the feedback, identifies common issues.

Main Success Scenario:

Actor's Actions	**System's Responses**
1. The admin logs into the system and opens the feedback dashboard.	2. The system displays user feedback.
3. The admin filters feedback based on rating.	4. The system updates the feedback view based on the required data.
5. The admin reviews feedback and identifies patterns, such as recurring issues or common positive aspects.	
6. The admin uses this analysis to determine key areas for improvement.	8. The system does not provide a direct response, as this is part of the admin's decision-making.

Technology and Data Variations List:

- Feedback data may include text comments, and ratings.

6. Set Security PIN for Device:

Scope: Smart Knock smart doorbell system

Level: User goal

Primary Actor: User (Homeowner)

Stakeholders and Interests:

- **Homeowner:** Wants to secure their device with a PIN to prevent unauthorized access.
- **Database and Storage System:** Ensures secure storage of the PIN without affecting user experience.

Preconditions:

- The app supports setting a security PIN for the device.
- The user is logged into the app and has access to the device's settings.

Postconditions:

- The device is secured with a 4 digit PIN.
- Unauthorized users cannot access the device without the correct PIN.

Main Success Scenario:

Actor's Actions	System's Responses
1. The user logs in.	2. The user has the option to either set PIN (primary user) or enter a PIN (secondary user).
3. The user selects an option.	4. The system prompts the user to enter a 4 digit PIN.
5. The user enters a 4 digit PIN.	6. The system validates the PIN format (e.g., length, numeric only).
7. The user confirms the entered PIN.	8. The system verifies the PIN matches or sets the initial entry and saves it securely.
	9. The device is now secured, and the user is required to enter the PIN for future access.

Extensions:

3a. Invalid PIN Format

Actor's Actions	System's Responses
1. The user enters a PIN with invalid format (e.g., too short, non-numeric characters).	2. The system displays an error message prompting the user to enter a valid 4 digit numeric PIN.
3. The user re-enters a valid PIN.	4. The system validates and accepts the PIN.

7. Multi-User Access Control:

Scope: Smart doorbell system

Level: User goal

Primary Actor: Primary User (Homeowner)

Stakeholders and Interests:

- **Primary User (Homeowner):** Wants to manage access for other users while retaining full control over the system.
- **Secondary Users (Family Members):** Want to have access to the same device.
- **Database and Storage System:** Ensures multiple users can securely access the system without conflicts.

Preconditions:

- The system supports multiple user accounts.
- The primary user is logged into the app and has full control over system settings.

Postconditions:

- Multiple users can access the same device with their individual notification preferences.
- The primary user maintains control over all access permissions for secondary users.

Main Success Scenario:

Actor's Actions	System's Responses
1. The primary user invites secondary users by sending them device PIN via text.	
2. The secondary users receive the text and enter the same PIN on logging in.	3. The system allows access to the device to secondary user.
4. Each secondary user sets their own notification preferences.	5. The system saves the notification preferences for each secondary user.
6. The secondary users begin receiving notifications independently based on their preferences.	7. The system ensures the primary user can modify or revoke permissions at any time.

Extensions:

6a. Removing a secondary user

Actor's Actions	System's Responses
1. The primary user wants to remove a secondary user.	2. The system displays the list of secondary users.
3. The primary user chooses who to remove	4. The system applies the changes.

8. Visitor History Log:

Scope: Smart doorbell system

Level: User goal

Primary Actor: User (Homeowner)

Stakeholders and Interests:

- **User (Homeowner):** Wants to review and monitor past visitors for security and awareness.

- **Database and Storage System:** Ensures accurate logging of visitor data and easy retrieval of information.

Preconditions:

- The system logs all visitor interactions, including images, timestamps.
- The user is logged into the app and has access to the visitor history section.

Postconditions:

- The user can review detailed information about past visitors, including images and timestamps.
- The user can filter and search the visitor history for specific data.

Main Success Scenario:

Actor's Actions	**System's Responses**
1. The user opens the app and navigates to the visitors section.	2. The system displays a list of all visitors, including images, timestamps.
3. The user scrolls through the list to review visitor details.	
4. The user uses filtering options to sort or search the visitor history by date, time, or visitor identity.	5. The system updates the display to reflect the user's chosen filter.
6. The user selects a specific visitor entry to view detailed information.	7. The system displays detailed visitor information.

9. User Feedback:

Scope: Smart Knock smart doorbell system

Level: User goal

Primary Actor: Homeowner

Stakeholders and Interests:

- **User (Homeowner):** Wants to provide feedback on the device to improve future performance or report issues.
- **Database and Storage System:** Wants to gather user feedback for analysis and system improvements.

Preconditions:

- The feedback feature is enabled and accessible through the app.

- The user is logged into the app and has access to the feedback section.

Postconditions:

- The user's feedback, including ratings and comments, is stored and linked to the specific Smart Knock device.
- The feedback is available for future analysis and product improvements.

Main Success Scenario:

Actor's Actions	System's Responses
1. The user navigates to the "Feedback" section within the app.	2. The system shows "Feedback" section in the app.
3. The user is prompted to provide a rating (e.g., 1-5 stars) and optional comments.	
4. The user submits their rating and comment.	5. The system validates the input (rating and comment).
6. The user successfully submits their feedback.	7. The system stores the feedback.

Extensions:

3a. Incomplete Feedback

Actor's Actions	System's Responses
1. The user submits feedback without providing a rating.	2. The system prompts the user to complete the rating before submission.
3. The user completes the rating and resubmits the feedback.	4. The system validates and saves the completed feedback.

4a. Feedback Submission Error

Actor's Actions	System's Responses
1. The user submits feedback, but the system encounters an error (e.g., no internet connection).	2. The system displays an error message and asks the user to try again later.
3. The user retries submitting the feedback.	4. The system saves the feedback and updates the feedback database.

Technology and Data Variations List:

- Feedback may be submitted via different formats (e.g., star ratings, text comments, or audio feedback).
- Feedback can be linked to specific device versions, allowing for targeted analysis based on device model.

10. Set name for Visitor:

Scope: Smart Knock smart doorbell system
Level: User goal
Primary Actor: User (Homeowner)
Stakeholders and Interests:

- **User (Homeowner):** Wants to improve the readability of the visitor log by naming a visitor, making it easier to identify and track visits.
- **System:** Ensures the name is properly saved and associated with the visitor's record for easy identification in the log.

Preconditions:

- The user is logged into the Smart Knock app.
- A visitor has pressed the doorbell, and their visit is recorded in the system.
- The visitor's image is available in the visitor log.
- The visitor log feature is functioning correctly.

Postconditions:

- The visitor's log entry now includes a name, improving readability and making the log easier to review.

Main Success Scenario:

Actor's Actions	System's Responses
1. The user opens the visitor log in the Smart Knock app.	2. The system displays the list of recorded visits, including details like visitor image and timestamp.
3. The user selects a visitor entry from the log.	4. The system displays the visitor's details, including their image and timestamp.
5. The user chooses to set name for the visitor.	
6. The user sets the desired name for the visitor.	7. The system saves the name and associates it with the visitor's log entry.
11. The user sees the updated visitor log with the named entry.	12. The system updates the log and displays the visitor's name alongside their visit details.

Extensions:

7a. Invalid Name Input

Actor's Actions	System's Responses
1. The user enters an invalid name (e.g., special characters, empty field).	2. The system displays an error message asking the user to enter a valid name.
3. The user corrects the name and submits	4. The system validates and saves the

it again.	corrected name.

10a. System Fails to Save Name

Actor's Actions	System's Responses
1. The user confirms the name entry, but the system encounters an error while saving.	2. The system displays an error message and asks the user to try again.
3. The user retries the action.	4. The system saves the name and updates the visitor log.

Special Requirements:

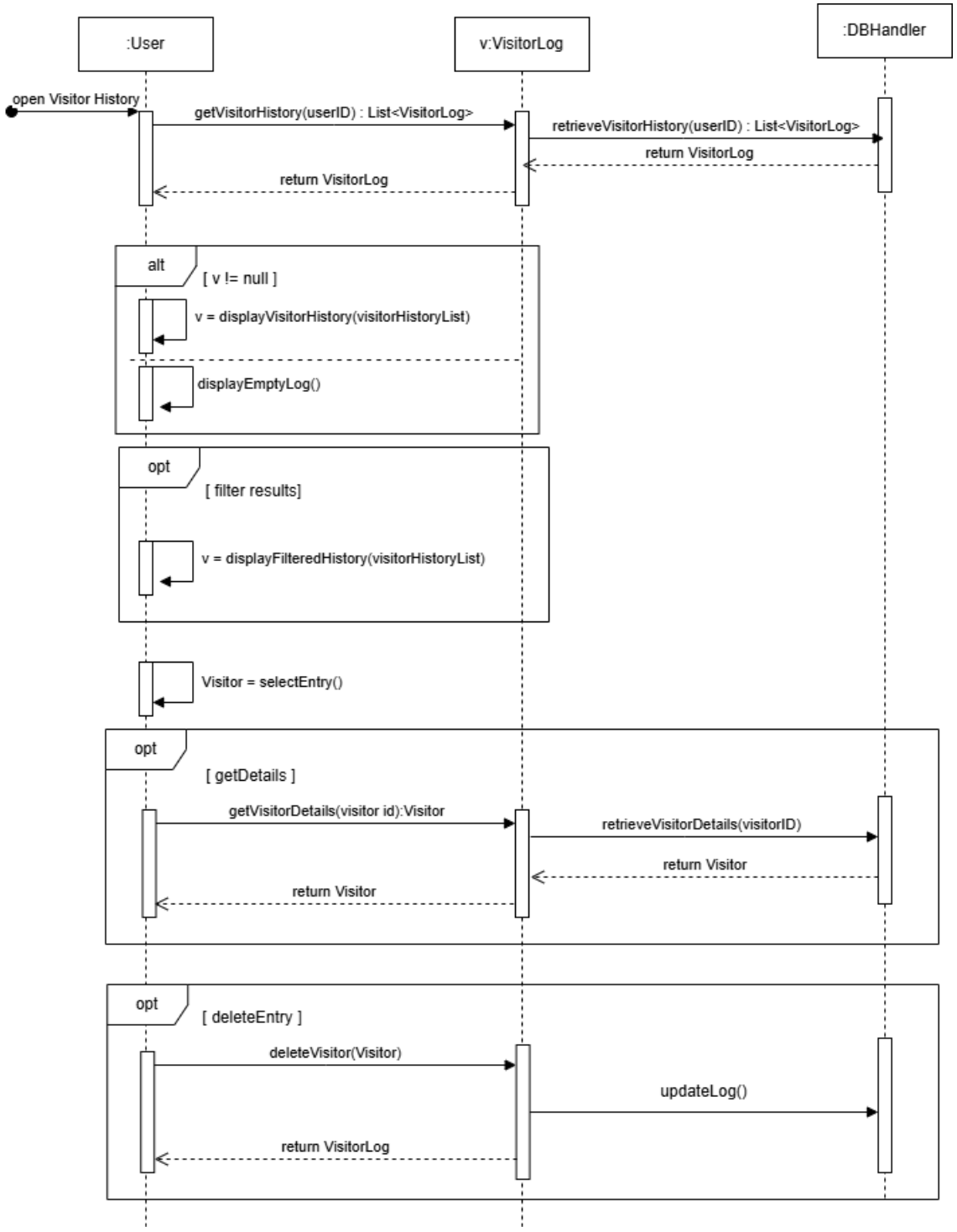
- The system should ensure that names are displayed in a readable format in the visitor log.
- The system must allow the user to edit a visitor's name if necessary.
- The naming feature should be available for all the visitors in the log.

Technology and Data Variations List:

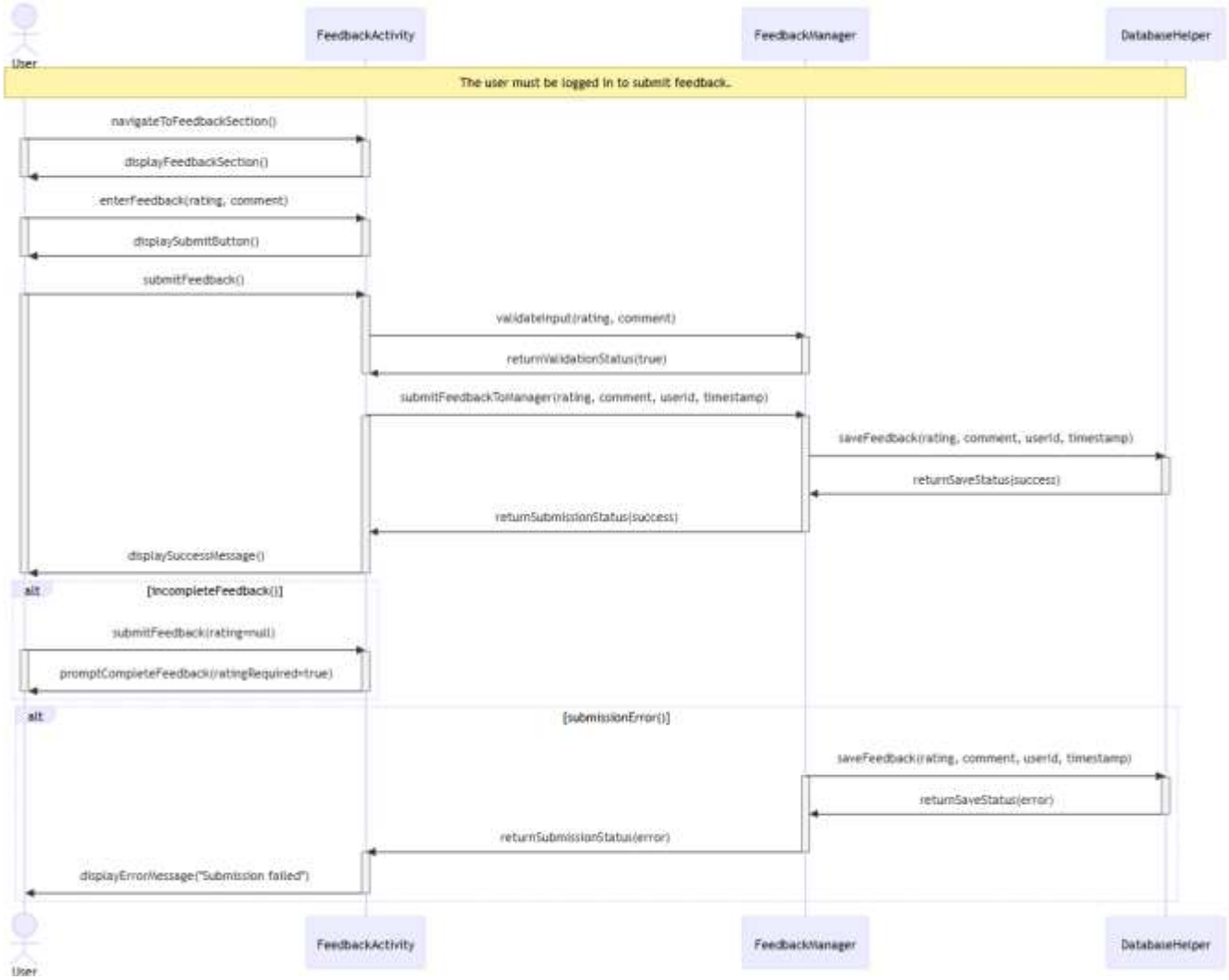
- The visitor log should be synced across all user devices, so any name updates are reflected in real time.

Activity Sequence Diagrams:

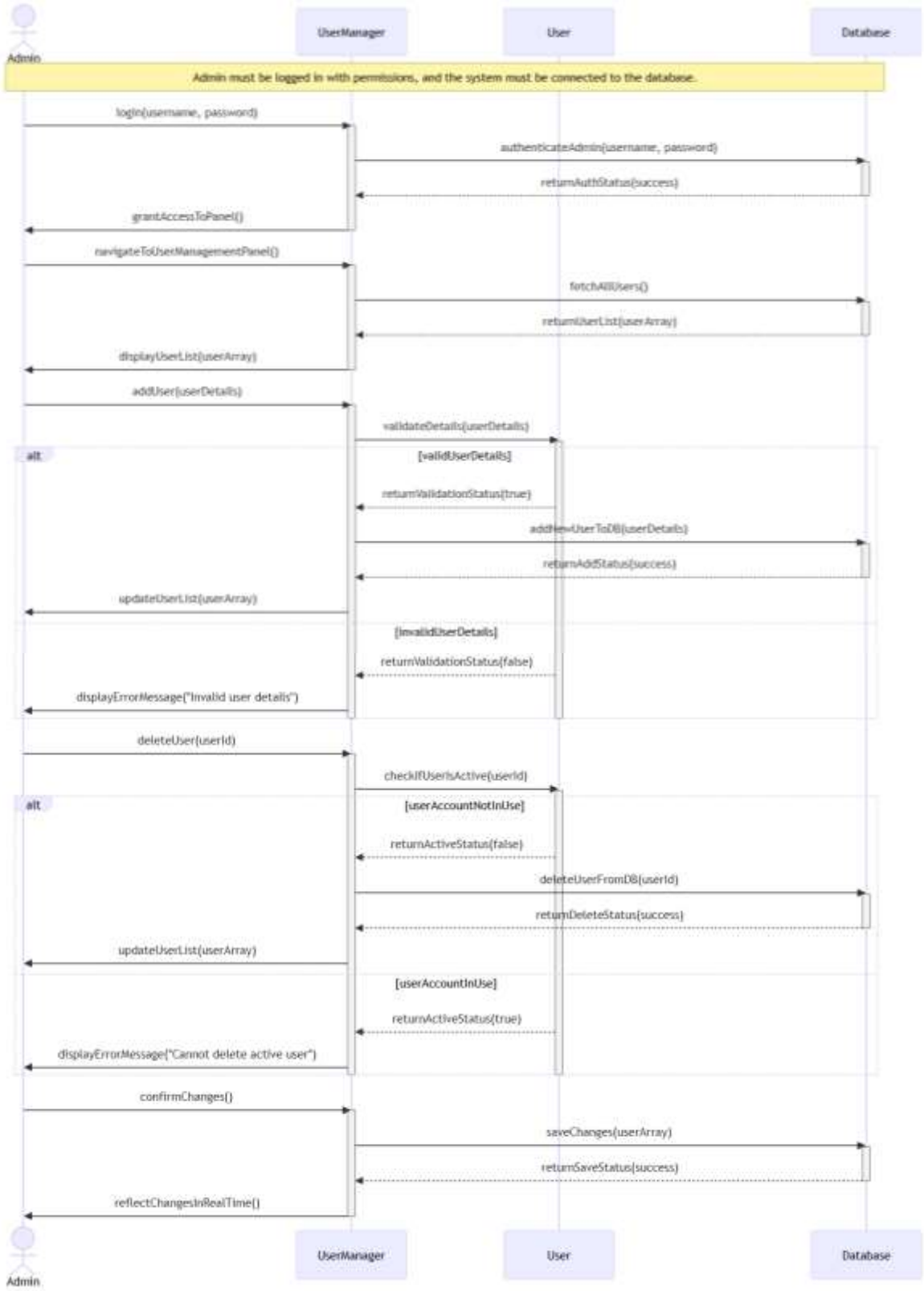
1. **Visitor History Log:**



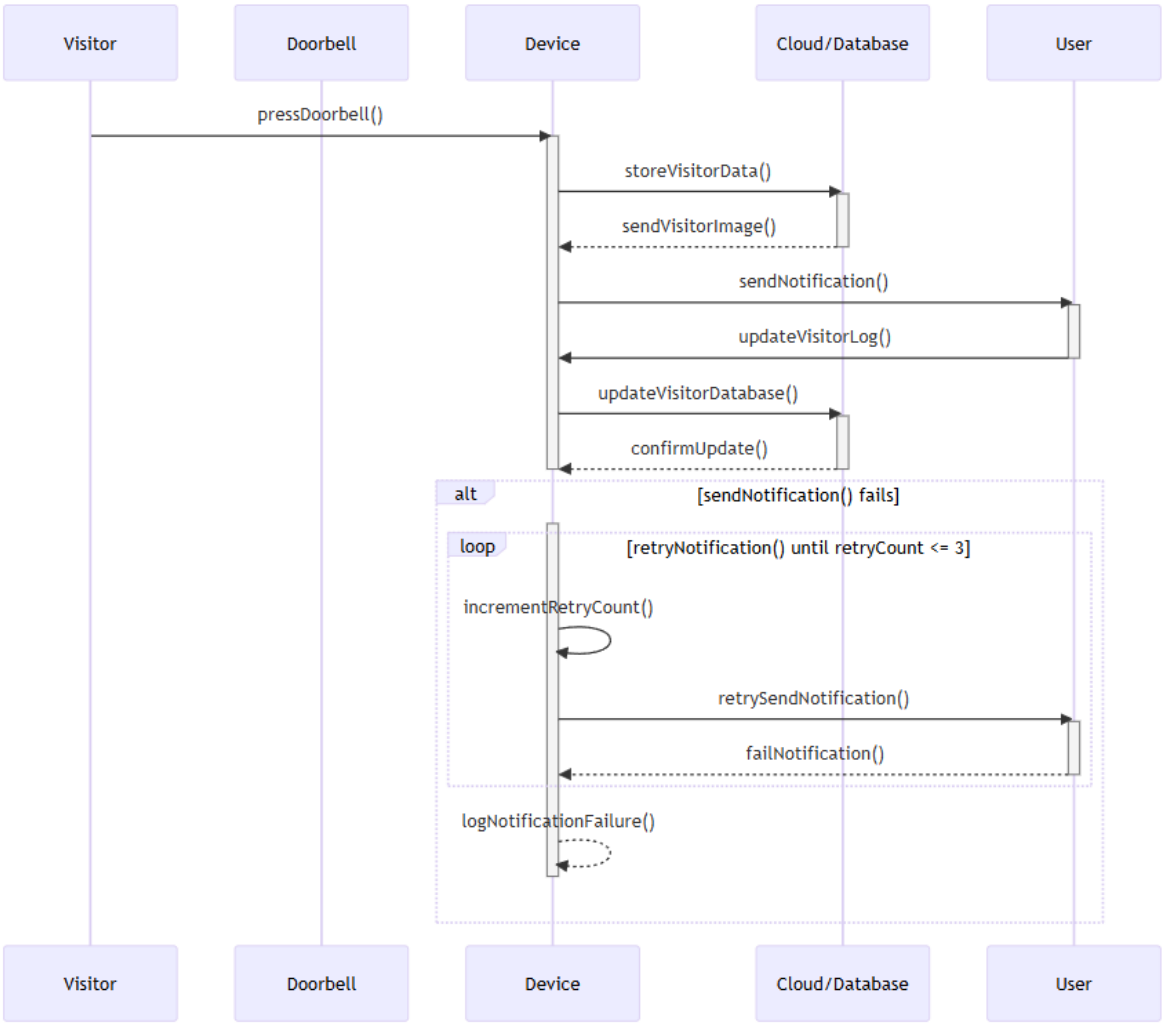
2. <u>User Feedback:</u>



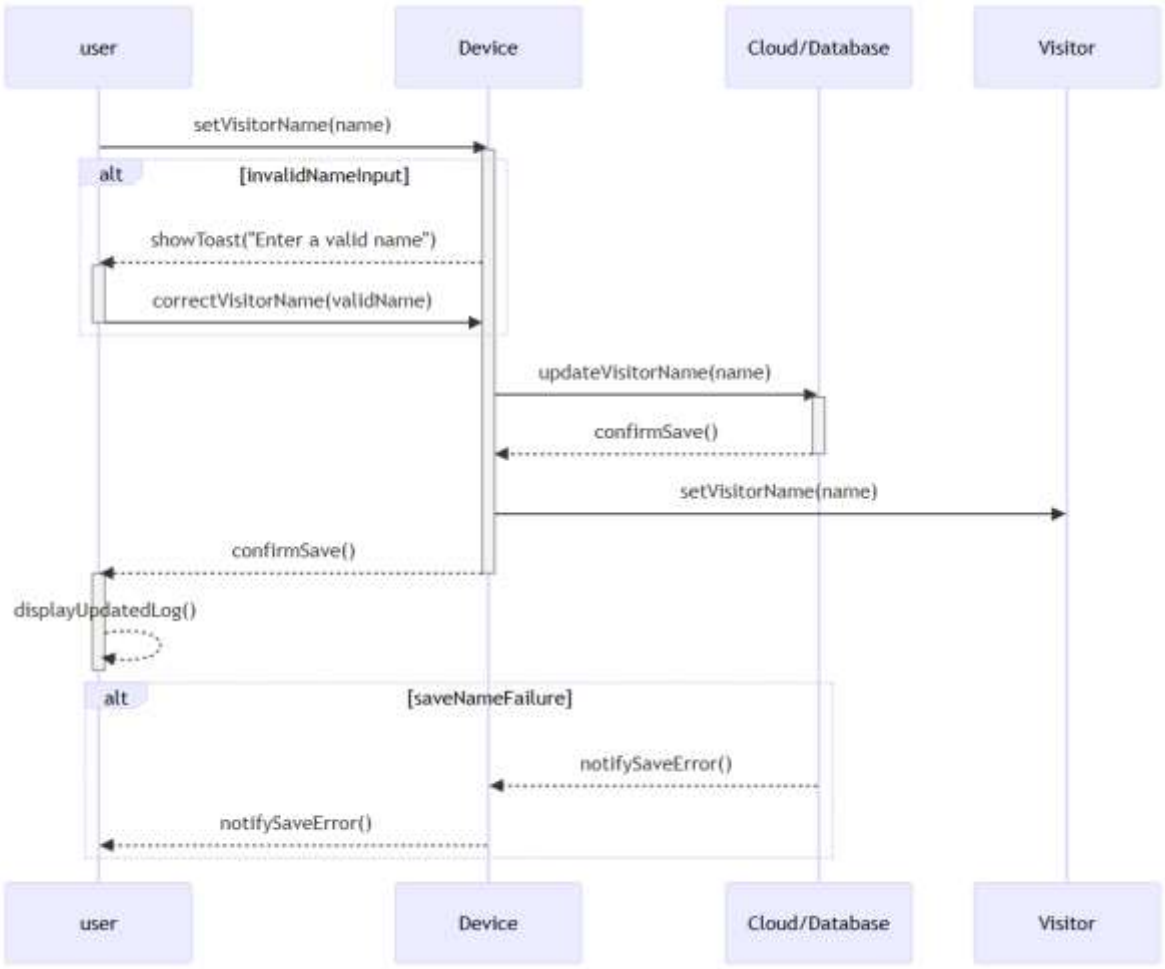
3. <u>Manage User Accounts</u>


Admin
UserManager
User
Database
Admin must be logged in with permissions, and the system must be connected to the database.
login(username, password)
authenticateAdmin(username, password)
returnAuthStatus(success)
grantAccessToPanel()
navigateToUserManagementPanel()
fetchAllUsers()
returnUserList(userArray)
displayUserList(userArray)
addUser(userDetails)
validateDetails(userDetails)
alt
[validUserDetails]
returnValidationStatus(true)
addNewUserToDB(userDetails)
returnAddStatus(success)
updateUserList(userArray)
[invalidUserDetails]
returnValidationStatus(false)
displayErrorMessage("Invalid user details")
deleteUser(userId)
checkIfUserIsActive(userId)
alt
[userAccountNotInUse]
returnActiveStatus(false)
deleteUserFromDB(userId)
returnDeleteStatus(success)
updateUserList(userArray)
[userAccountInUse]
returnActiveStatus(true)
displayErrorMessage("Cannot delete active user")
confirmChanges()
saveChanges(userArray)
returnSaveStatus(success)
reflectChangesInRealTime()
Admin
UserManager
User
Database

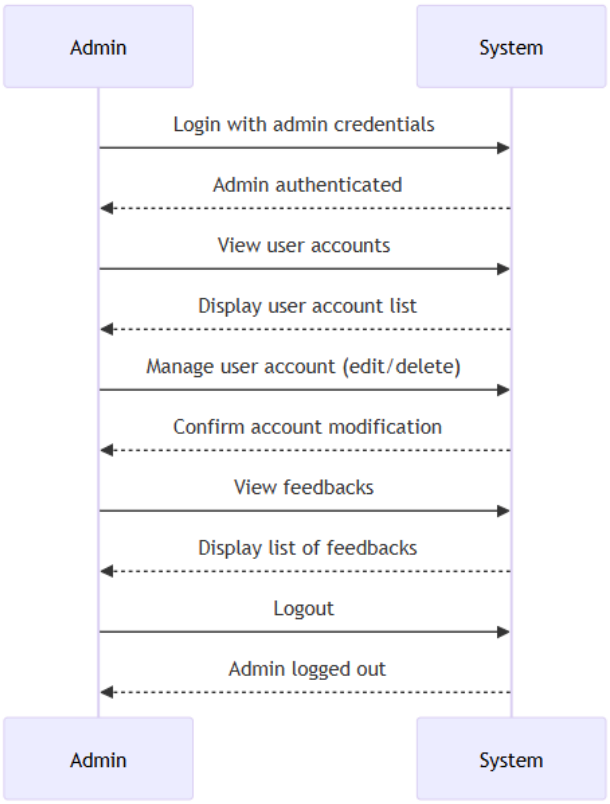
4. Visitor Notification

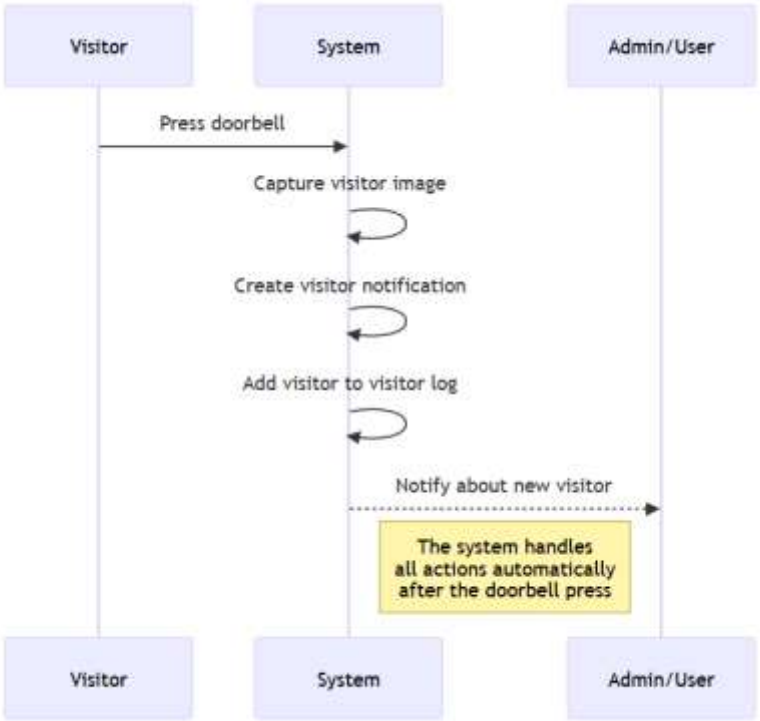


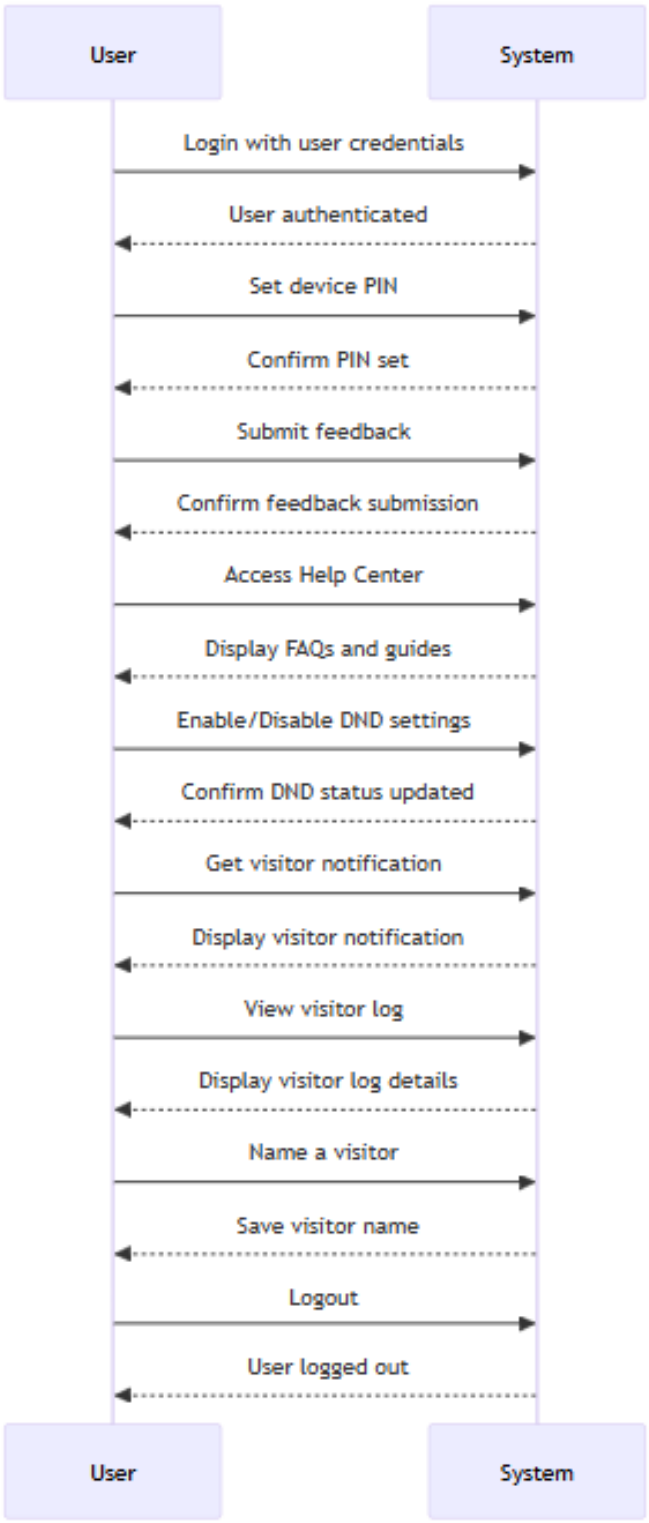
5. Set name for Visitor:



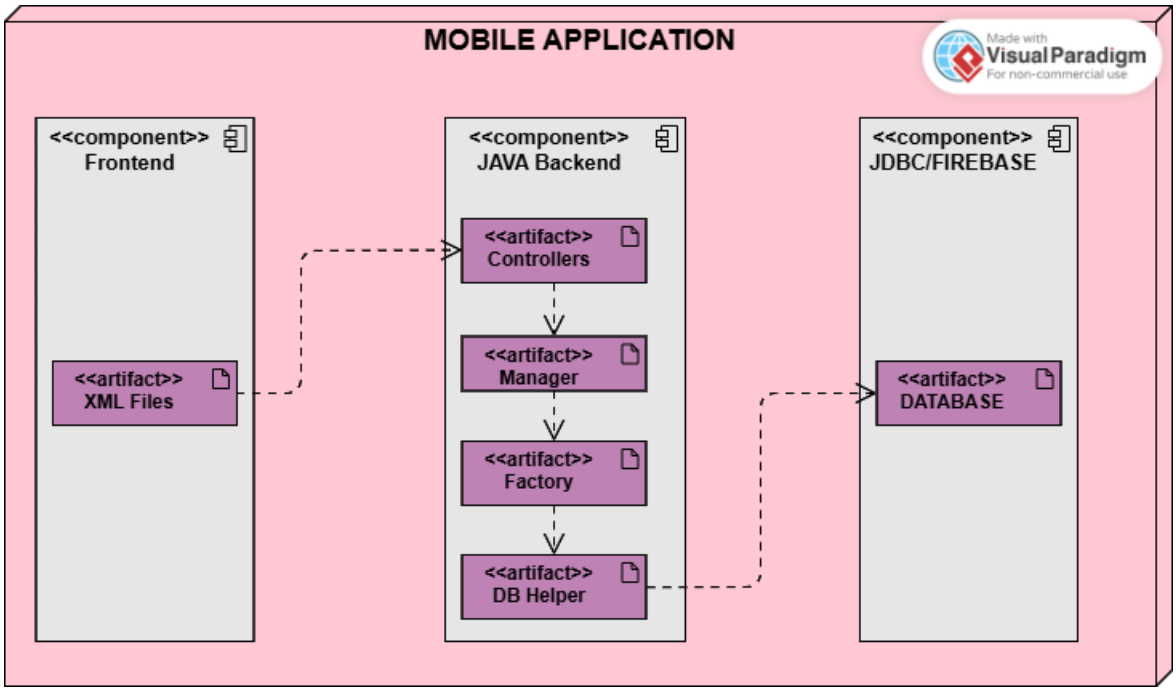
System Sequence Diagrams:



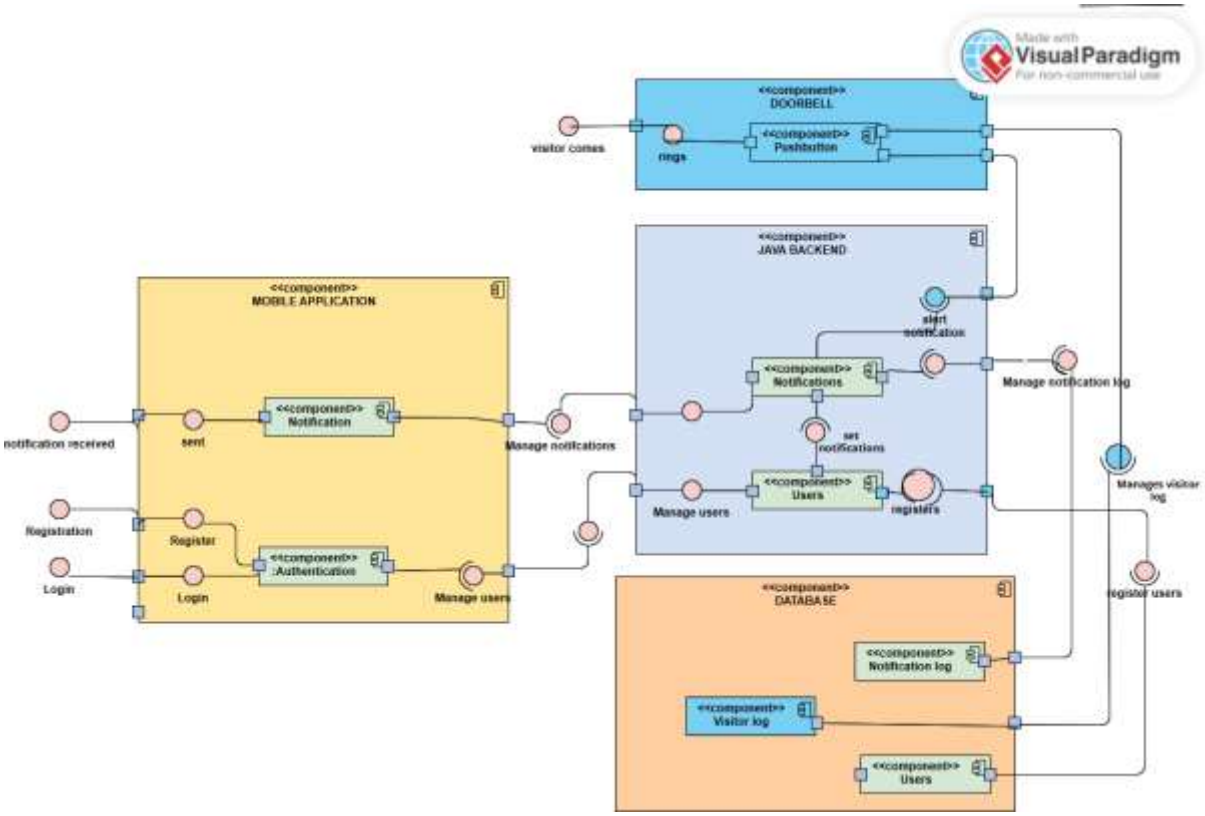



User
System
Login with user credentials
User authenticated
Set device PIN
Confirm PIN set
Submit feedback
Confirm feedback submission
Access Help Center
Display FAQs and guides
Enable/Disable DND settings
Confirm DND status updated
Get visitor notification
Display visitor notification
View visitor log
Display visitor log details
Name a visitor
Save visitor name
Logout
User logged out
User
System

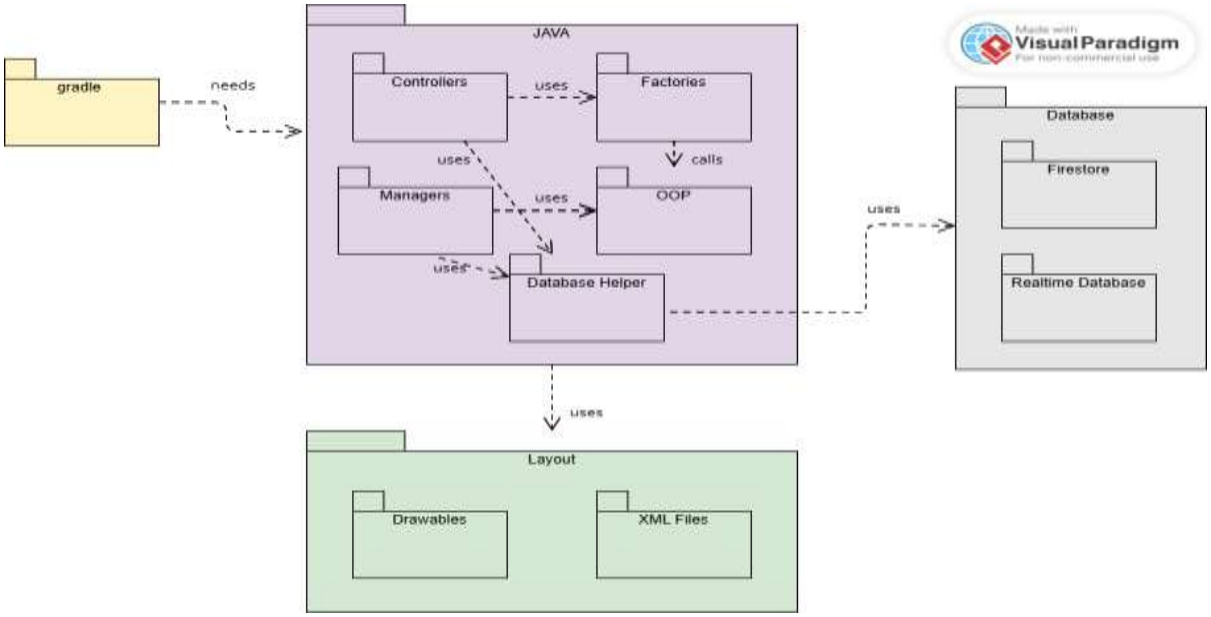
Deployment Diagram:



Component Diagram:



Package Diagram:



Three Tier Architecture:

1. UI Layer (Presentation Layer)

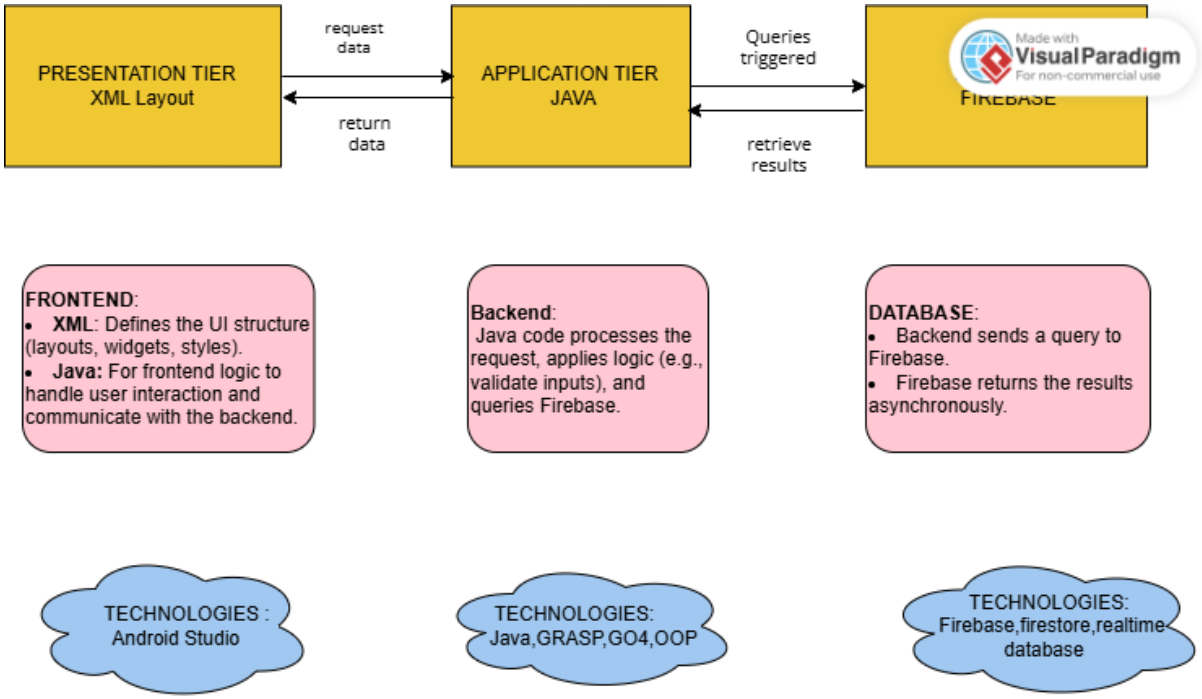
- **User Management**: User activities manage user interactions, such as login, registration, and profile updates.
- **Admin Management**: Admin pages displays the admin panel, with options to manage users, and feedback.
- **Visitor Management**: Visitors Activity displays the list of visitors, while VisitorDetails Activity shows detailed information about each visitor.
- **Feedback Management**: Feedback Activity allows admins to view and manage feedback submitted by users.

2. Business Logic Layer (Application Layer)

- **User Management**: UserController handles the logic for authenticating and updating user data.
- **Admin Management**: AdminController manages the business logic related to admin actions such as adding or removing users, and viewing feedback.
- **Visitor Management**: VisitorController is responsible for managing visitor-related data, including fetching visitor information from Firebase and updating visitor details.
- **Notification Management**: NotificationController coordinates with NotificationManager to monitor Firebase for new events (e.g., new visitor entries) and send notifications to the UI layer.

3. Data Access Layer (Database Layer)

- **User Management**: UserManager manages all interactions with Firebase for user data, including authentication and profile updates.
- **Admin Management**: AdminManager interacts with Firebase to store and manage admin data.
- **Visitor Management**: VisitorManager interacts with Firebase to fetch, add, update, and delete visitor data.
- **Feedback Management**: FeedbackManager manages operations for feedback data, interacting with Firebase for real-time data updates.



GRASP :

1. Information Expert

- **User Management**: The User class holds user-specific data such as name, email, and password, making it the information expert in this domain.
- **Admin Management**: The Admin class manages admin-related operations and holds admin-specific data, acting as an expert in this context.
- **Visitor Management**: The VisitorView class stores details of a visitor (name, date, etc.), making it the information expert for handling visitor-related data.
- **Feedback Management**: The Feedback class encapsulates user feedback, including comments and ratings, ensuring it holds all relevant data for feedback functionality.

2. Creator

- **User Management**: The UserManager class is responsible for creating and managing User instances, ensuring all user data is handled correctly.
- **Admin Management**: The AdminController is responsible for creating and managing admin accounts and delegating tasks to relevant components.
- **Visitor Management**: The VisitorManager creates VisitorView objects when fetching and updating visitor data.
- **Feedback Management**: The FeedbackController creates instances of feedback based on user input.

3. Low Coupling

- The layers of the application are loosely coupled. For example, the UI does not interact directly with Firebase; instead, it communicates with the VisitorController, UserController, and AdminController, ensuring that changes in one class do not require changes in others.

4. High Cohesion

- **User Management**: The UserManager class focuses only on managing user data, including authentication and updates.
- **Admin Management**: The AdminController handles only admin-related logic, ensuring that each class is highly cohesive with a single responsibility.
- **Visitor Management**: The VisitorAdapter binds visitor data to the RecyclerView, and VisitorController manages the visitor-related logic.
- **Notification Management**: The NotificationManager handles all notification logic, ensuring that the class is highly cohesive.

5. Controller and Facade

- **User Management**: The UserController handles the flow of data between the UI and the business logic for user-related operations like login and registration.
- **Admin Management**: The AdminController is responsible for the admin dashboard's logic, managing tasks such as adding, removing, and viewing admin data.
- **Visitor Management**: The VisitorController acts as the controller between the UI and Firebase for fetching, updating, and displaying visitor data.
- **Notification Management**: The NotificationController handles the logic of monitoring Firebase for new visitors and triggering notifications via the NotificationManager.

Design Patterns:

1. Singleton Pattern

- **UserManager, AdminManager, NotificationManager**: These classes are designed as singletons to ensure only one instance of each exists throughout the app, managing user, admin, and notification data in a consistent manner.

2. MVC (Model-View-Controller) Pattern

- User Management:

    - **Model**: User class represents user data.
    - **View**: UserActivity displays the UI for user interactions.
    - **Controller**: UserController manages business logic for handling user operations.
- Visitor Management:
    - **Model**: VisitorView represents visitor data.
    - **View**: Visitors and VisitorDetailsActivity display visitor data.
    - **Controller**: VisitorController manages the business logic for fetching and updating visitor data.
- Admin Management:
    - **Model**: Admin class holds admin-related data.
    - **View**: AdminDashboard displays the admin UI.
    - **Controller**: AdminController handles admin-specific logic.
- Notification Management:
    - **Model**: Notification contains notification-related data.
    - **View**: NotificationActivity displays notifications.
    - **Controller**: NotificationController handles business logic for notification operations.

3. Adapter Pattern

- **Visitor Management**: The VisitorAdapter adapts the visitor data (VisitorView objects) to the format suitable for display in a RecyclerView.
- **Feedback Management**: The FeedbackAdapter adapts feedback data to display it in a list format in the UI.
- **User Management**: User Adapter adapts User data into a displayable format for the admin UI.

4. Observer Pattern

- **Firebase**: Firebase itself acts as the observer, notifying the app of changes in the database (e.g., adding a new visitor or updating feedback) using ChildEventListener or ValueEventListener.
- **Notification Management**: The NotificationManager listens to Firebase changes and updates the notification UI.

5. Callback Pattern

- **Visitor Management**: The VisitorCallback interface allows asynchronous operations such as fetching visitors or updating a visitor's data, and informs the caller (UI) once the operation completes.

- **Feedback Management**: A similar FeedbackCallback interface allows the feedback system to inform the UI when new feedback is available or when feedback submission is successful.

Summary

Concept	Used In	Purpose
Singleton Pattern	UserManager, AdminManager, NotificationManager	Ensures single instances of these managers to maintain consistency.
MVC Pattern	User, Admin, Visitor, Feedback components	Separates concerns between data, UI, and business logic.
Observer Pattern	Firebase Realtime Database, NotificationManager	Notifies the app of real-time changes in data.
Callback Pattern	VisitorCallback, FeedbackCallback	Handles asynchronous operations and notifies UI once tasks are completed.
Adapter Pattern	VisitorAdapter, FeedbackAdapter	Adapts data for display in UI components like RecyclerView.

Conclusion:

These procedures provides a structured approach for the Smart Knock app, ensuring a seamless user experience through personalized security features. By organizing core functionalities around users, devices, and notifications, the model facilitates efficient home security management tailored to user preferences, ultimately enhancing convenience and safety.