CS350 Safehome Project
Software Design Specification

Team 1

20190813 Minjun Kim
20210145 Jehyung Kim
20210482 Onejune Lee
20210484 Yunje Lee

Table of Contents

I. Overview

1 Introduction

This document provides the comprehensive software design specification (SDS) for the SafeHome system. Acting as the bridge between the requirement analysis and the implementation phases, this specification details the architectural and component-level design. The focus is to create a robust and scalable blueprint for the development team. To comprehensively illustrate the system's design, this document outlines the high-level system architecture, detailed class and object models including CRC cards, and the dynamic behavior of the system through state and sequence diagrams.

2. Goal

The primary objective of this design is to serve as a solid and resilient foundation for the entire lifecycle of the SafeHome system. The following goals are the result of extensive analysis and are established to ensure its long-term success and viability:

1. **High-Fidelity Implementation**: To ensure the design is a faithful and precise translation of the previously established requirements and analysis models, leaving no room for ambiguity.
2. **Architectural Integrity**: To engineer a fundamentally sound architecture built on the principles of high cohesion and low coupling. This promotes a modular structure where components are both internally robust and externally independent.
3. **Embedded Quality Attributes**: To proactively embed critical quality attributes—specifically reliability, maintainability, testability, and efficiency—into the core of the design, rather than treating them as afterthoughts.
4. **Strategic Simplicity and Adaptability**: To aggressively manage and reduce complexity wherever possible, while building a flexible and reusable framework that can adapt to future requirements without significant rework.

3 How the design work proceeded

1) Extraction of classes: The foundation of our design was established by deriving an initial set of classes directly from the system requirements. We used LLM to parsed the narrative of each use case, treating the primary nouns as potential classes and the associated verbs as their principal methods.

2) Creation of architectural structure: The extracted classes served as the foundational building blocks for the system's architecture. By analyzing how these classes collaborate within the context of the main use case scenarios, we established a cohesive architectural structure that maps out the system's core modules and their interactions.

3) Visualization of class diagram: To create a concrete representation of the design, we developed a class diagram. We used LLM to generate mermaid code and created diagrams with mermaid live editor. This visually modeled the relationships between our classes within the defined architecture, incorporating early considerations for the implementation phase.

4) Creation of CRC card: We created CRC cards to simulate object interactions and validate our design. This process was crucial for mapping out collaborations

5) Iterative Refinement based on Validation: The class diagram was not static; it evolved. Through rigorous testing with CRC cards and constant cross-referencing against the original requirements, we refined the model. This iterative process was guided by the core principles of achieving high cohesion and low coupling, resulting in a more modular and robust design.

6) Creation of state diagram: To model the dynamic behavior of objects, we created state diagrams. We utilized an LLM to analyze the class diagram and extract all classes that could potentially have states. For each extracted class, we then analyzed its states and drew the corresponding state diagram.

4. Assumptions

4.1. Scope Additions (Not in SRS)

4.1.1 Dual Frontend Applications

- Decision: Separate SafeHomeHub (tablet) and SafeHomeMobile (phone) applications
- Rationale: SRS implied single interface; we added explicit dual-platform support for better UX
- Impact: Both share same ViewControllers but with different layouts (landscape vs. portrait)

4.1.2 PTZ Control Service

- Decision: Add dedicated PTZControlService with mutex locking mechanism
- Rationale: SRS mentioned PTZ control but didn't specify conflict resolution; we added proper concurrency control
- Impact: Prevents multiple users from controlling same camera simultaneously

4.1.3 Recording Management

- Decision: Add RecordingManager with search, export, and sharing capabilities
- Rationale: SRS mentioned recording but lacked management features; we added complete lifecycle management
- Impact: Users can search recordings by date/camera, export to MP4/AVI, and generate share links

4.1.4 Device Health Monitoring

- Decision: Add DeviceHealthMonitor service for battery, signal strength, and offline detection
- Rationale: SRS implied device status but didn't specify health monitoring; we added proactive monitoring
- Impact: System can detect and alert on low battery, weak signal, or offline devices

4.1.5 Safety Zone Abstraction

- Decision: Create SafetyZone entity to group sensors logically
- Rationale: SRS mentioned zones implicitly; we made them first-class entities for better organization
- Impact: Users can create zones (e.g., "First Floor", "Backyard"), arm/disarm zones independently

4.1.6 Session Management

- Decision: Add explicit Session entity and session lifecycle management

- Rationale: SRS mentioned login but didn't specify session handling; we added proper session tracking
- Impact: Multi-device support, session expiration, and logout functionality

4.2. Security Simplifications

4.2.1 Password Authentication Only

- Decision: Simple password hash verification; no encryption, no OAuth
- Rationale: Complex security is out of scope for 2-week MVP; focus on architecture over security implementation
- Impact: Use bcrypt or SHA-256 for password hashing; no HTTPS/TLS simulation

4.2.2 Two-Factor Authentication (Out of Scope)

- Decision: 2FA mentioned in SRS but excluded from SDS implementation
- Rationale: 2FA requires SMS/email integration or authenticator apps; too complex for MVP
- Impact: LoginManager has 2FA methods in design but marked as "future enhancement"

4.2.3 No Data Encryption

- Decision: Store data in plain text or simple hashing
- Rationale: Encryption adds complexity without demonstrating architectural principles
- Impact: Focus on proper data access patterns rather than encryption

4.2.4 Simple Audit Logging

- Decision: Use console.log() for audit trail instead of database logging
- Rationale: Detailed audit trail is out of scope; console logging demonstrates logging concept
- Impact: UserPermissionManager logs actions but doesn't persist to database

4.3. Concurrency Simplifications

4.3.1 Single-User PTZ Control

- Decision: PTZ locking mechanism exists but assumes single concurrent user
- Rationale: Multi-threaded concurrency control is too complex for 2-week implementation
- Impact: PTZControl tracks lock owner but doesn't handle race conditions

4.3.2 No Streaming Concurrency Control

- Decision: StreamingService allows multiple streams but doesn't enforce bandwidth limits
- Rationale: Bandwidth management requires network-level implementation beyond scope
- Impact: Bandwidth properties exist in design but not enforced

4.3.3 Single Application Instance

- Decision: Assume single running instance; no distributed system support
- Rationale: Distributed systems, load balancing, and clustering are out of scope
- Impact: All managers are singletons; no horizontal scaling considerations

4.4. Data Management Simplifications

4.4.1 In-Memory Data Storage

- Decision: Use in-memory collections (Map, List) instead of database
- Rationale: Database integration is not core to software architecture demonstration
- Impact: Data lost on restart; sufficient for demonstration purposes

4.4.2 No Cloud Server Integration

- Decision: All services run locally; no actual cloud communication
- Rationale: Network communication and cloud infrastructure out of scope
- Impact: NotificationManager simulates push notifications without actual delivery

4.4.3 Simplified Storage Management

- Decision: RecordingManager tracks storage quota but doesn't enforce it
- Rationale: File system management beyond scope of architecture demonstration
- Impact: Storage properties exist but automatic cleanup not implemented

4.5. Device Integration Simplifications

4.5.1 Simulated Device Communication

- Decision: Devices don't actually communicate; trigger() methods simulate events
- Rationale: IoT protocol implementation (Zigbee, Z-Wave) is out of scope
- Impact: Device classes have communication methods but use event simulation

4.5.2 No Firmware Updates

- Decision: Device firmware management excluded

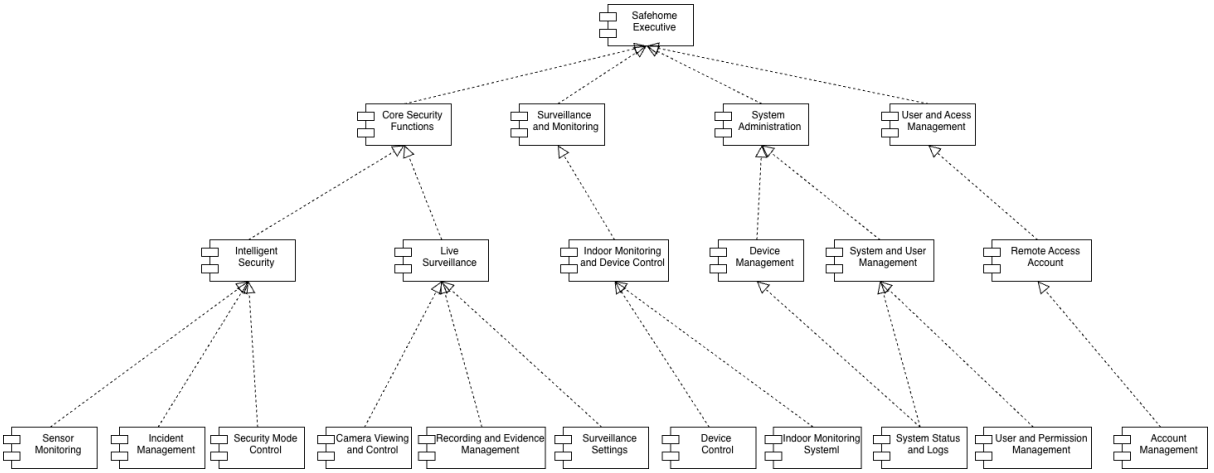
- Rationale: Firmware OTA updates require low-level system programming beyond scope
- Impact: Device has firmware property but no update mechanism

4.5.3 Simplified Device Discovery

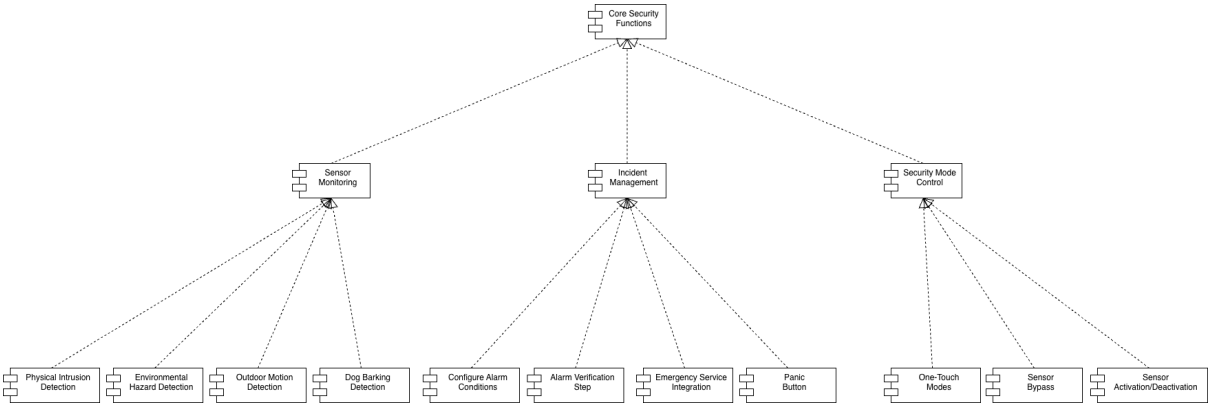
- Decision: DeviceRegistry.discoverDevices() returns pre-configured devices
- Rationale: Network discovery protocols (mDNS, UPnP) are out of scope
- Impact: Manual device registration assumed; no automatic discovery

II. Architectural Structure

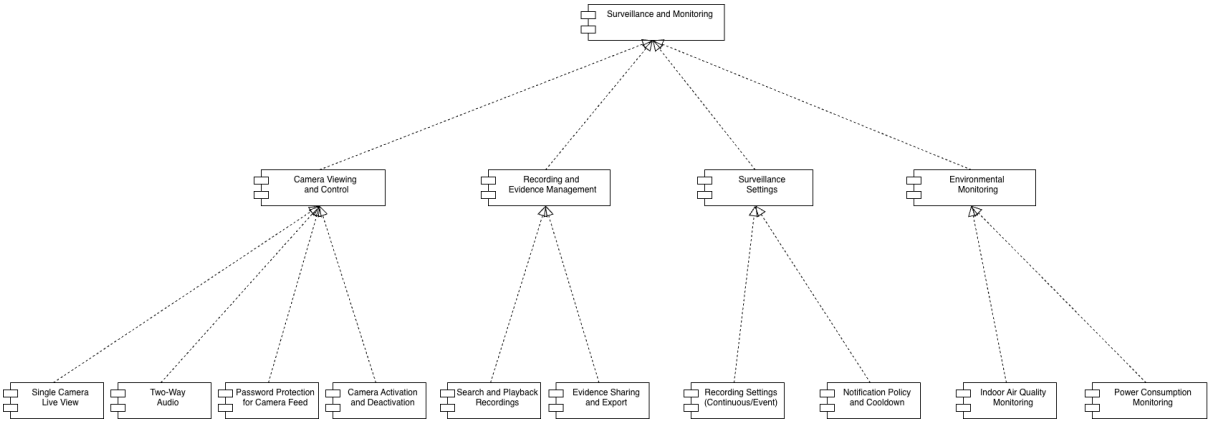
1. Overall Architecture



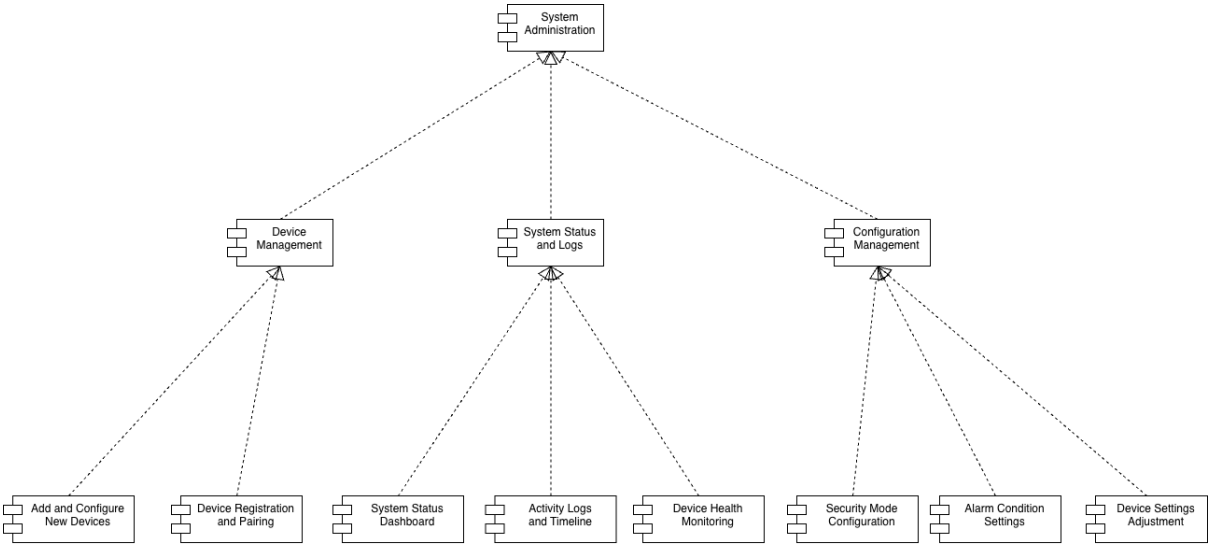
2. Core Security



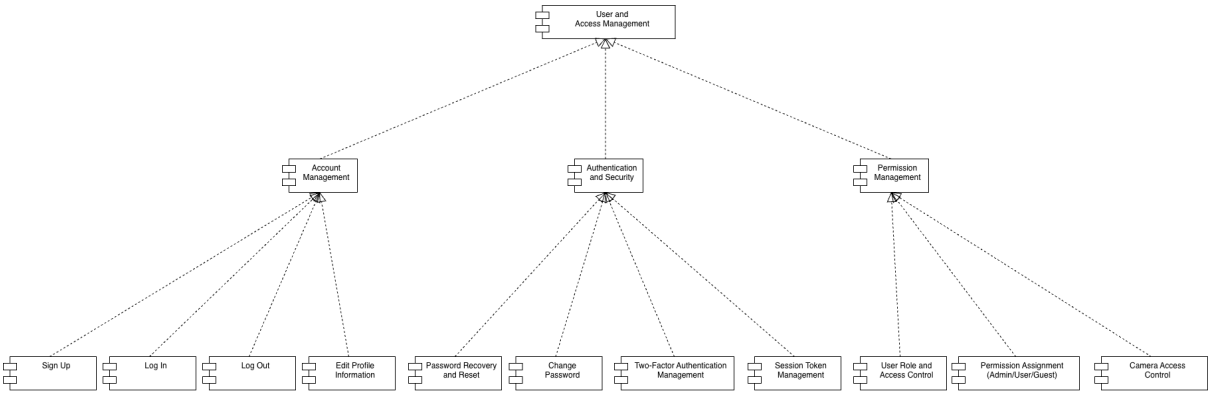
3. Surveillance Monitoring



4. System Admin

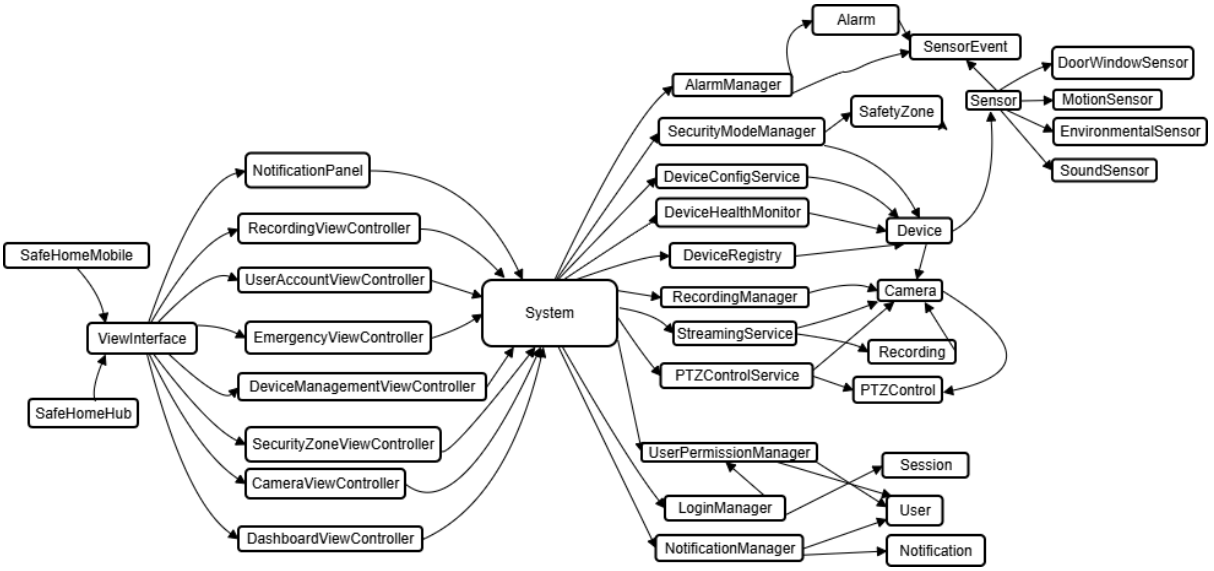


5. User Access

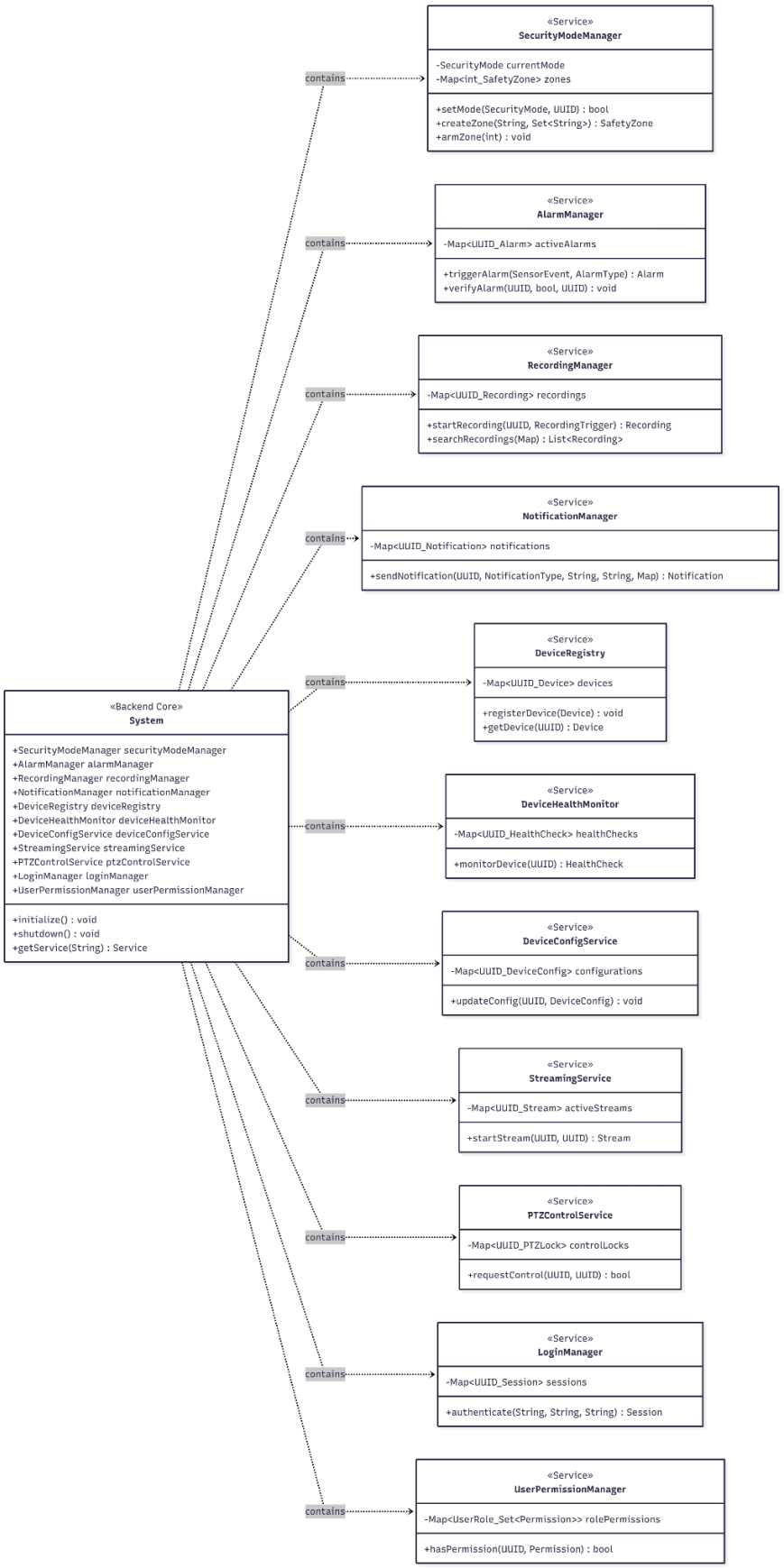


III. Class Diagram

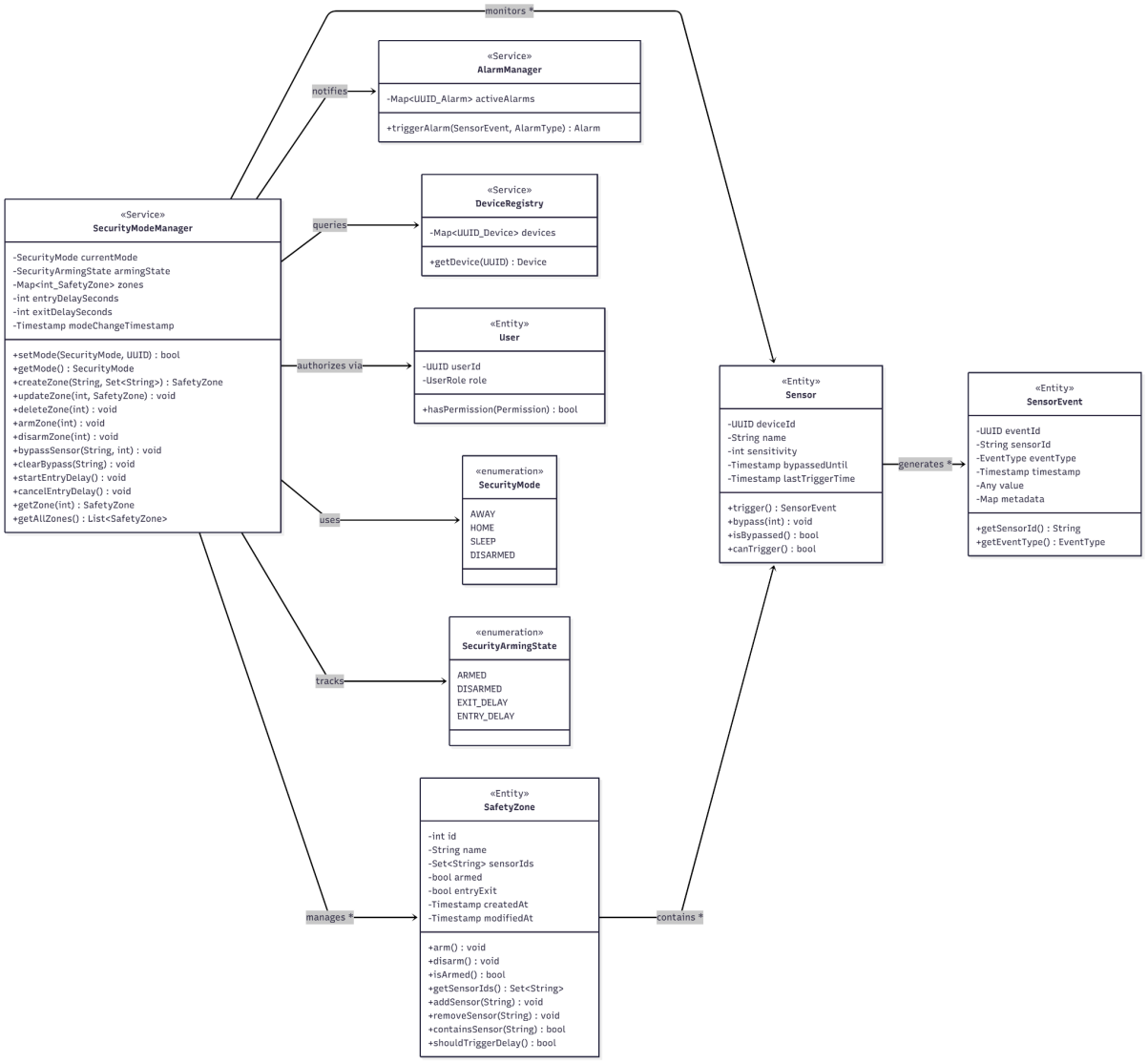
1. Class overview



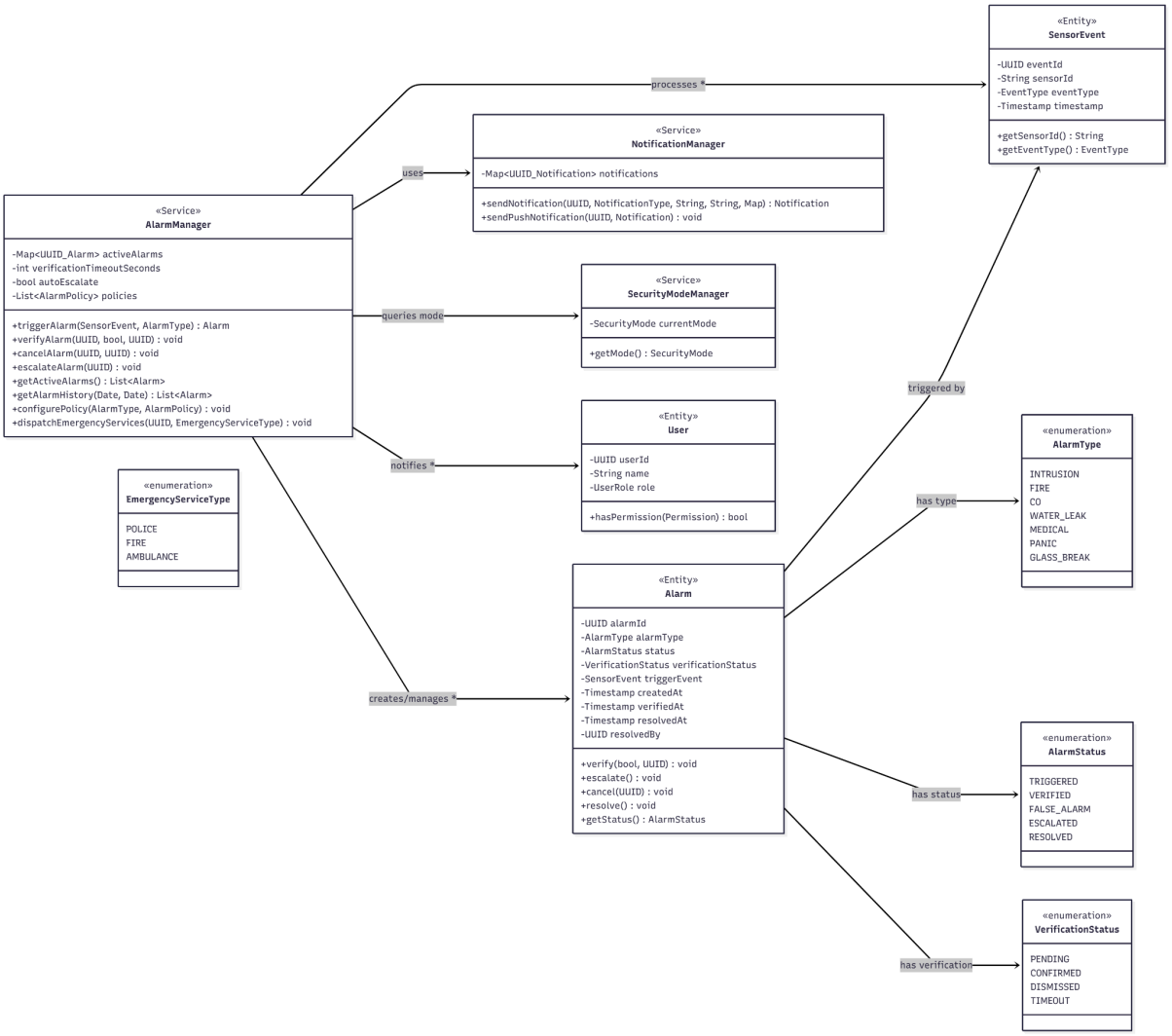
2. Backend Core System



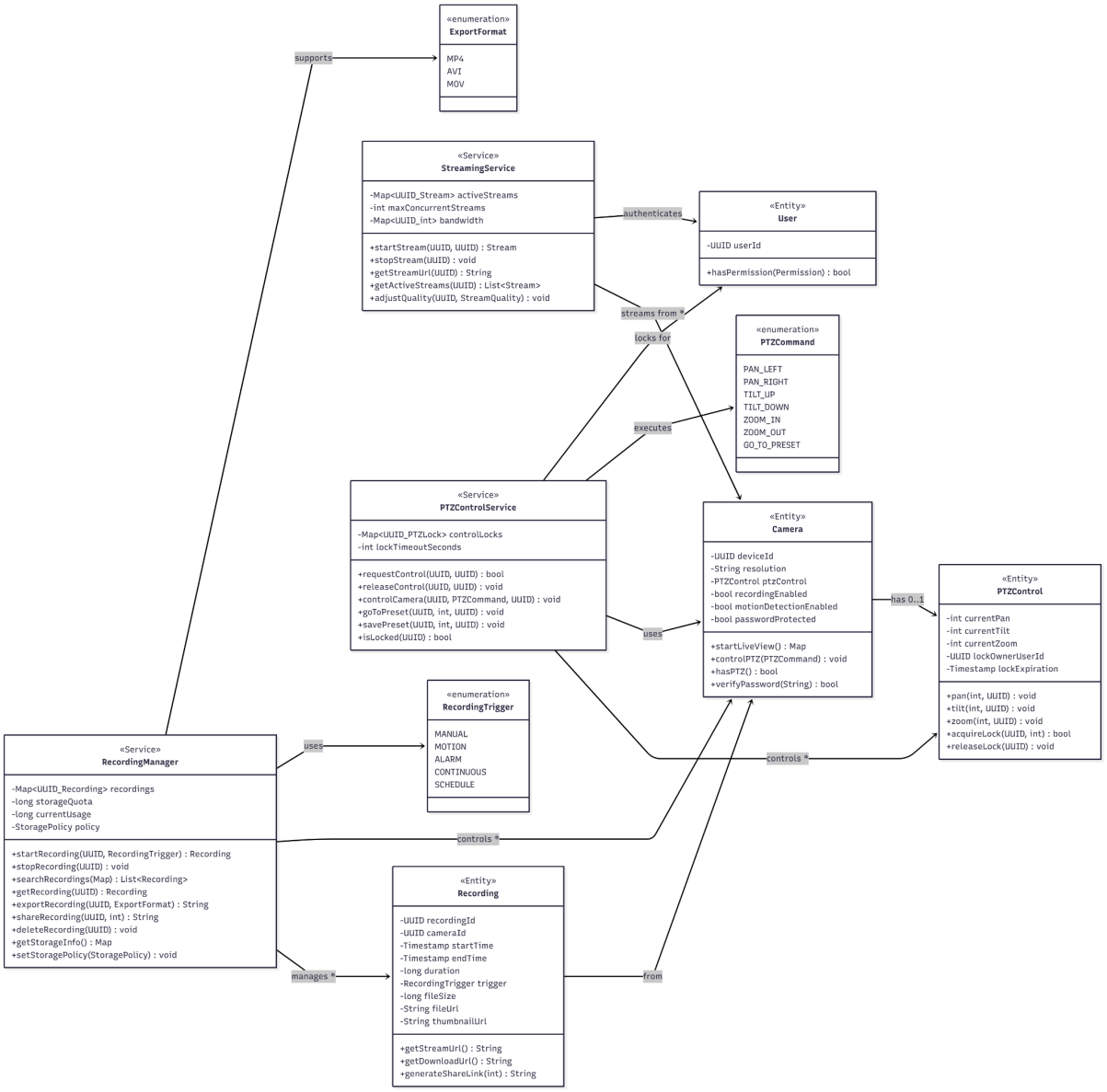
3. Security Management



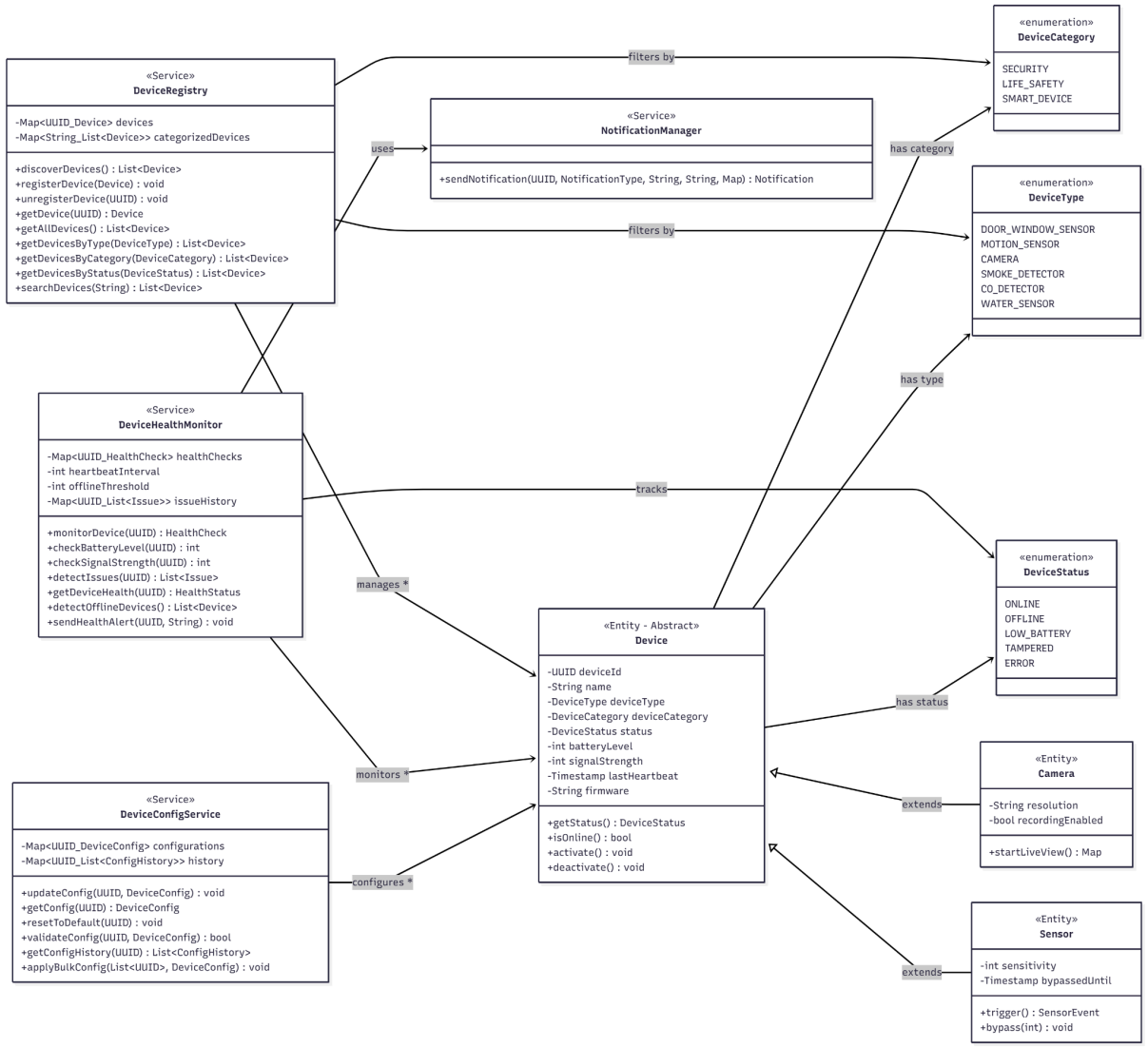
4. Alarm System



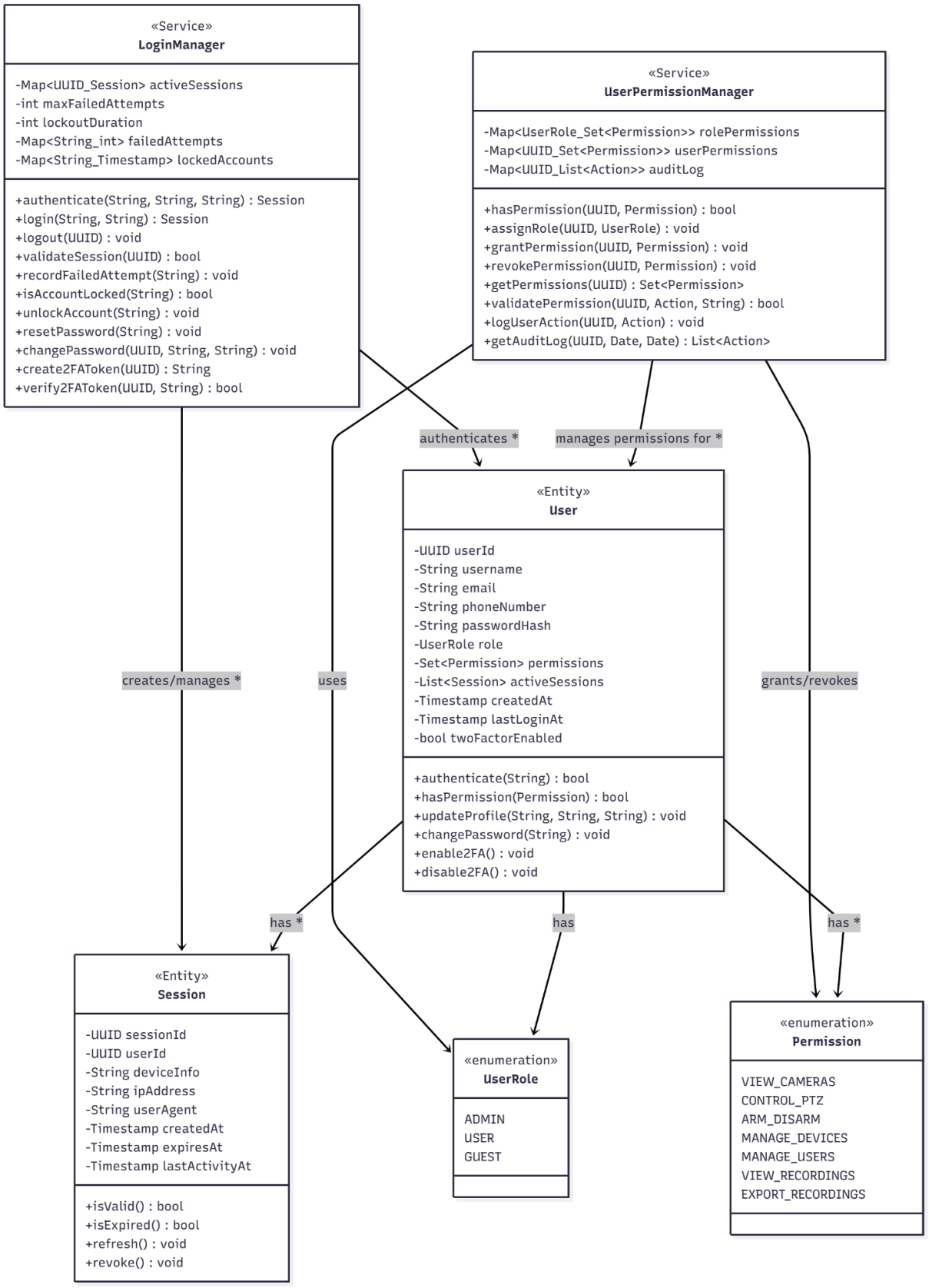
5. Recording & Surveillance



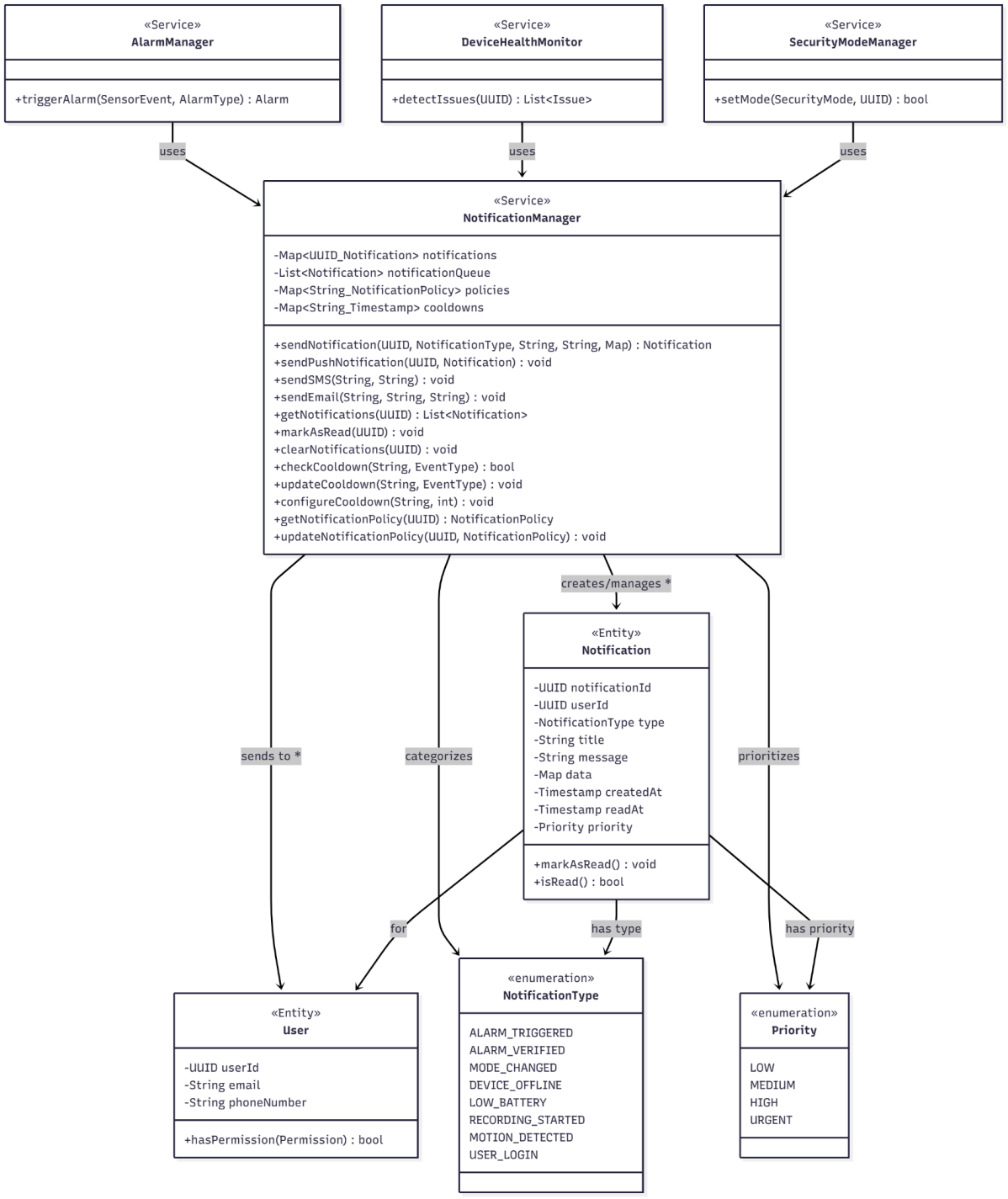
6. Device Management



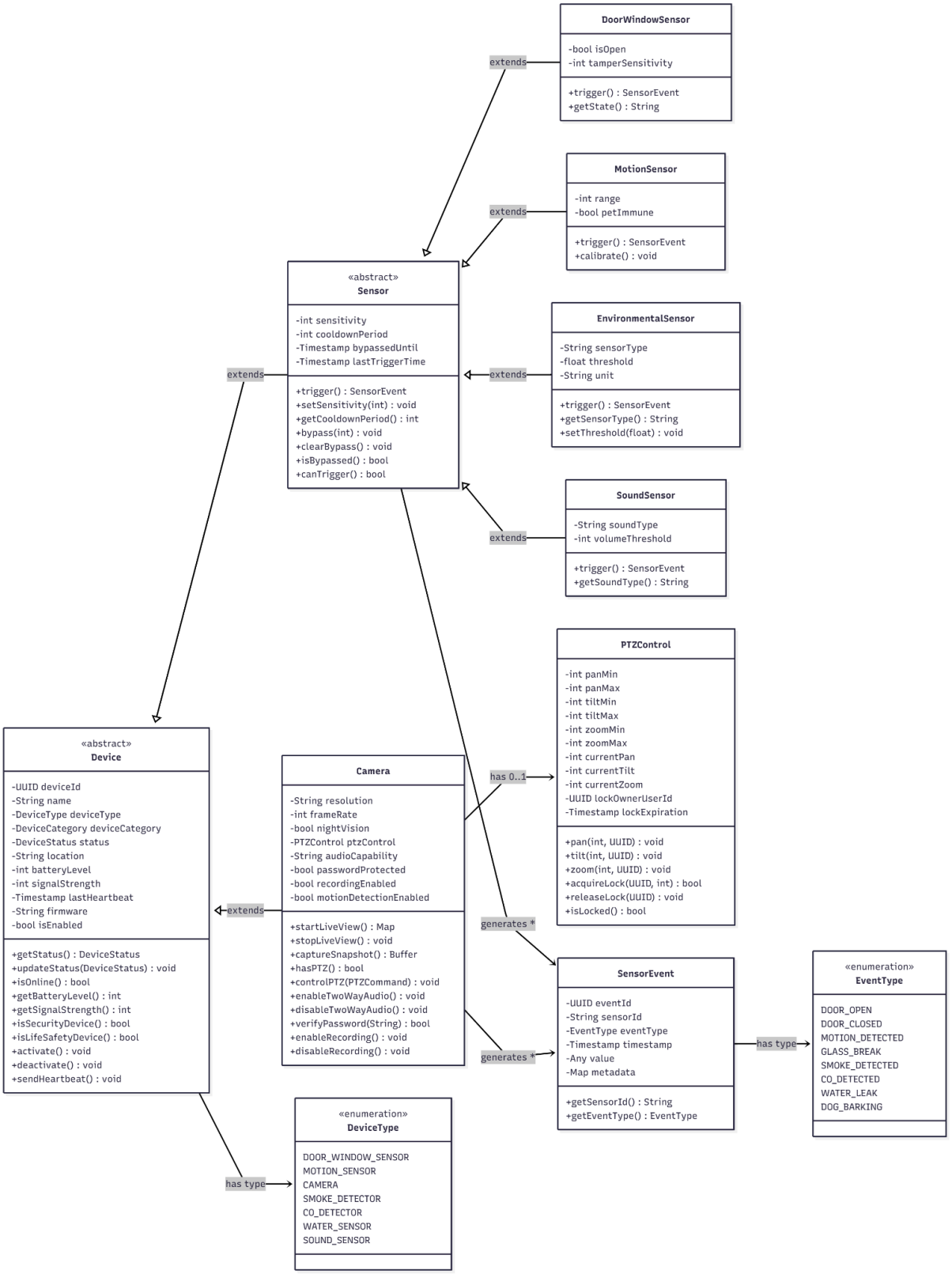
7. User & Authentication



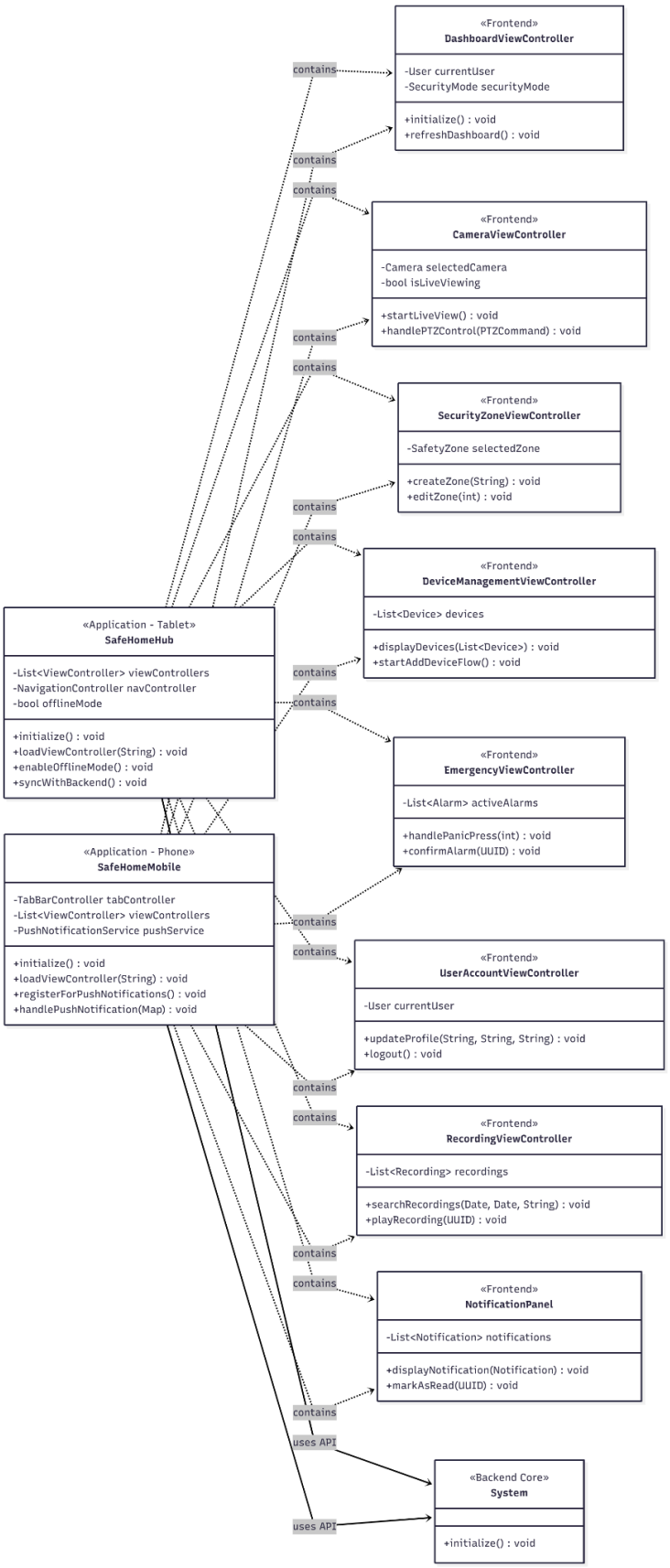
8. Notification System



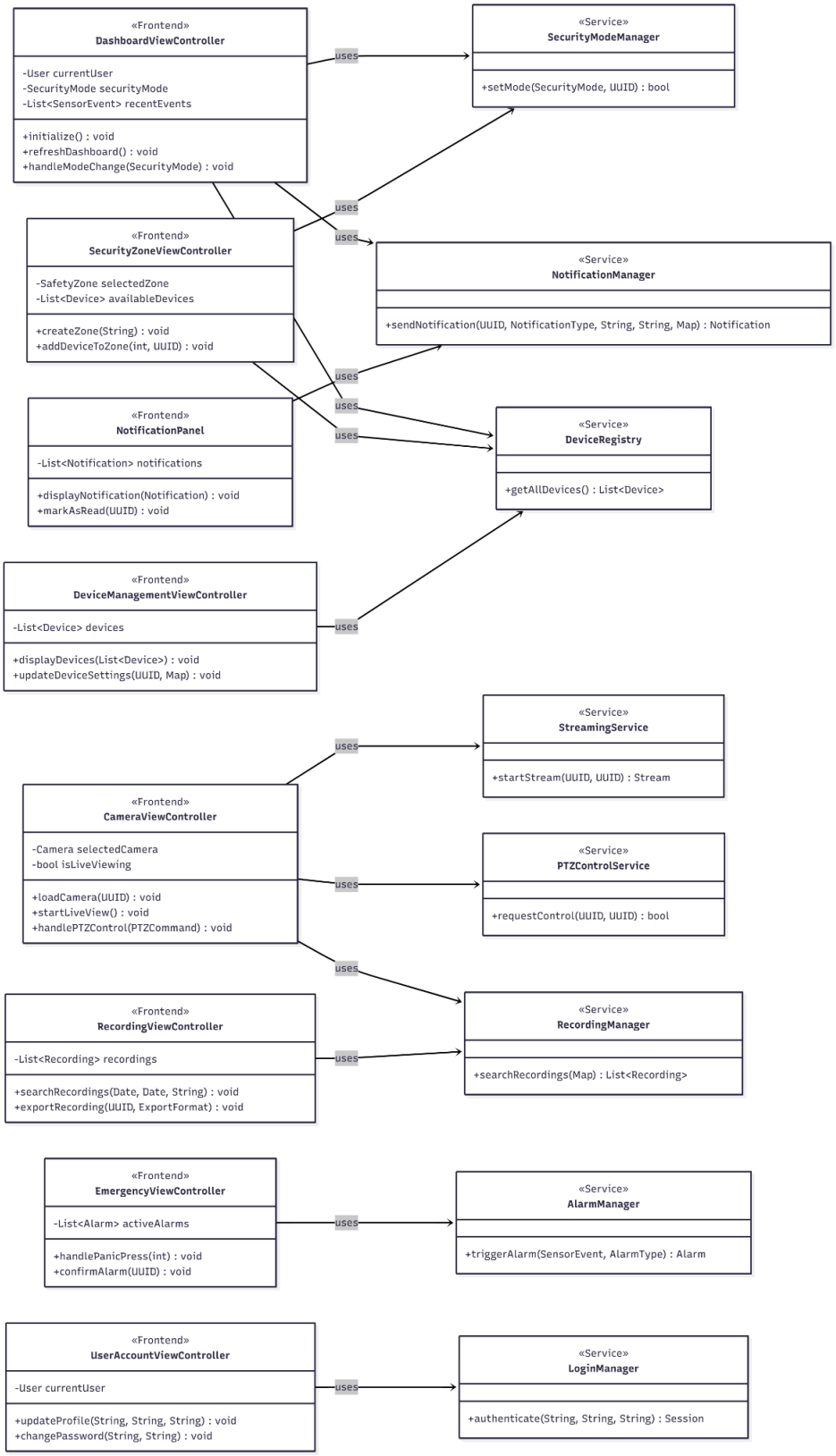
9. Device Hierarchy



10. Frontend Applications



11. ViewControllers Overview



IV. CRC Cards

1. Business Logic Services

Class: System	
Central backend system that initializes and manages all service components, provides unified RESTful API for frontend applications, and coordinates inter-service communication.	
Responsibilities	**Collaborators**
Initialize all manager services	All Manager Services
Provide unified RESTful API	
Route frontend requests to services	All Manager Services
Manage service lifecycle	All Manager Services
Coordinate inter-service communication	All Manager Services
Handle system-level configuration	
Serve both Hub and Mobile apps	SafeHomeHub, SafeHomeMobile
Provide service discovery	

Class: SecurityModeManager	
Manages security modes (Away, Home, Sleep, Disarmed), safety zones, sensor arming/disarming, entry/exit delays, and sensor bypass functionality.	
Responsibilities	**Collaborators**
Set current security mode	SecurityMode, User
Get current security mode	SecurityMode
Track security arming state	SecurityArmingState
Create new safety zone	SafetyZone
Update existing zone	SafetyZone
Delete safety zone	SafetyZone
Arm security zone	SafetyZone
Disarm security zone	SafetyZone
Bypass sensor temporarily	Sensor
Clear sensor bypass	Sensor
Start entry delay countdown	SafetyZone
Cancel entry delay	
Find zone by sensor ID	SafetyZone, Sensor
Trigger alarm when sensor activates	AlarmManager, SensorEvent

Class: AlarmManager	
Manages alarm lifecycle including triggering, verification, escalation, and resolution. Integrates with emergency services and notification system.	
Responsibilities	**Collaborators**
Trigger alarm from sensor event	SensorEvent, Alarm, AlarmType
Track all active alarms	Alarm
Verify alarm (confirm or dismiss)	Alarm, User
Cancel false alarm	Alarm, User
Escalate unverified alarm	Alarm
Get list of active alarms	Alarm
Configure alarm policies	SecurityMode
Dispatch emergency services	EmergencyServiceType
Activate siren	AlarmType
Start verification timer	Alarm
Send alarm notifications	NotificationManager
Process alarm resolution	Alarm, User

Class: RecordingManager	
Manages video recording lifecycle including starting, stopping, searching, exporting, and sharing recordings. Handles storage quota management.	
Responsibilities	**Collaborators**
Start recording from camera	Camera, Recording, RecordingTrigger
Stop ongoing recording	Recording
Search recordings by criteria	Recording
Get specific recording	Recording
Export recording to format	Recording, ExportFormat
Share recording via link	Recording
Delete recording	Recording
Get storage information	
Set storage policy	StoragePolicy
Manage storage quota	

Class: NotificationManager	
Manages multi-channel notification delivery (push, SMS, email) with cooldown policies, notification history, and user preferences.	
Responsibilities	**Collaborators**
Create and send notification	User, Notification, NotificationType
Send push notification	User, Notification
Send SMS notification	User
Send email notification	User
Get user notifications	User, Notification
Mark notification as read	Notification
Clear user notifications	User
Check cooldown period	EventType
Update cooldown settings	EventType
Configure notification cooldown	
Get notification policy	User
Update notification policy	User

Class: DeviceRegistry	
Central device repository that manages device discovery, registration, removal, and provides device search and filtering capabilities.	
Responsibilities	**Collaborators**
Discover new devices on network	Device
Register device in system	Device
Unregister device from system	Device
Get device by ID	Device
Get all registered devices	Device
Filter devices by type	Device, DeviceType
Filter devices by category	Device, DeviceCategory
Filter devices by status	Device, DeviceStatus
Search devices by name	Device

Class: DeviceHealthMonitor	
Monitors device health status including battery levels, signal strength, heartbeat tracking, and issue detection. Generates health alerts.	
Responsibilities	**Collaborators**
Monitor device health status	Device, HealthCheck
Check device battery level	Device
Check signal strength	Device
Detect device issues	Device, Issue
Get device health status	Device, HealthStatus
Detect offline devices	Device
Send health alert	Device, NotificationManager
Track heartbeat intervals	Device

Class: DeviceConfigService	
Manages device configuration updates, validation, and history tracking. Supports bulk configuration updates and reset to defaults.	
Responsibilities	**Collaborators**
Update device configuration	Device, DeviceConfig
Get device configuration	Device, DeviceConfig
Reset device to defaults	Device
Validate configuration changes	Device, DeviceConfig
Get configuration history	Device, ConfigHistory
Apply bulk configuration	Device, DeviceConfig

Class: StreamingService	
Manages live video streaming from cameras including stream initialization, URL generation, quality adjustment, and bandwidth management.	
Responsibilities	**Collaborators**
Start stream from camera	Camera, Stream, User
Stop active stream	Stream
Get stream URL	Stream
Get active streams for user	User, Stream
Adjust stream quality	Stream, StreamQuality
Manage bandwidth allocation	
Handle streaming errors	Camera

Class: PTZControlService	
Manages PTZ camera control with mutex locking mechanism to prevent conflicts. Handles PTZ commands and preset positions.	
Responsibilities	**Collaborators**
Request PTZ control lock	PTZControl, User
Release PTZ control lock	PTZControl, User
Execute PTZ command	Camera, PTZControl, PTZCommand, User
Go to preset position	Camera, PTZControl, User
Save preset position	Camera, PTZControl, User
Check if PTZ is locked	PTZControl
Validate control permissions	User

Class: LoginManager	
Manages user authentication, session lifecycle, failed login tracking, account locking, password reset, and two-factor authentication.	
Responsibilities	**Collaborators**
Authenticate user credentials	User
Create user session	Session, User
Logout user	Session
Validate active session	Session
Record failed login attempt	User
Check if account locked	User
Unlock user account	User
Reset user password	User
Change user password	User

Class: UserPermissionManager	
Implements role-based access control (RBAC), manages user permissions, validates actions, and maintains audit logs of user activities.	
Responsibilities	**Collaborators**
Check if user has permission	User, Permission
Assign role to user	User, UserRole
Grant specific permission	User, Permission
Revoke permission from user	User, Permission
Get user permissions	User, Permission
Validate permission for action	User, Permission, Action
Log user action for audit	User, Action
Get audit log for user	User, Action

2. Domain Entities

Class: User	
User entity storing profile information, credentials, role, permissions, and active sessions. Provides authentication and authorization methods.	
Responsibilities	**Collaborators**
Store user identification	
Store user profile information	
Store password hash	
Authenticate password	
Check user permissions	Permission
Update profile information	
Change password	
Generate backup codes	
Track active sessions	Session
Enable/disable 2FA	

Class: Device	
Abstract base class for all physical devices. Provides common functionality for device status, battery level, signal strength, and heartbeat tracking.	
Responsibilities	**Collaborators**
Store device identification	
Store device name and location	
Get device status	DeviceStatus
Update device status	DeviceStatus
Check if device online	
Get battery level	
Get signal strength	
Identify if security device	DeviceCategory
Identify if life safety device	DeviceCategory
Activate device	
Deactivate device	
Send heartbeat signal	

Class: Sensor	
Base sensor class extending Device. Handles sensor triggering, sensitivity settings, cooldown periods, and bypass functionality.	
Responsibilities	**Collaborators**
Trigger sensor event	SensorEvent
Set sensitivity level	
Get cooldown period	
Set cooldown period	
Bypass sensor temporarily	
Clear bypass	
Check if bypassed	
Validate if can trigger	

Track last trigger time	

Class: Camera	
Camera entity extending Device. Manages streaming, recording, PTZ control, two-way audio, password protection, and motion detection.	
Responsibilities	**Collaborators**
Start live view stream	PTZControl
Stop live view stream	
Capture snapshot image	
Check if has PTZ capability	PTZControl
Control PTZ movement	PTZControl, PTZCommand
Enable two-way audio	
Disable two-way audio	
Set password protection	
Remove password protection	
Verify password	
Enable recording	
Disable recording	
Enable motion detection	
Disable motion detection	

Class: PTZControl	
PTZ control entity managing pan, tilt, and zoom functionality with mutex locking to prevent control conflicts between users.	
Responsibilities	**Collaborators**
Execute pan command	User
Execute tilt command	User
Execute zoom command	User
Acquire control lock	User
Release control lock	User
Check if locked	
Validate PTZ command	PTZCommand
Get current position	
Track lock owner	User
Track lock expiration	

Class: Recording	
Recording entity storing video metadata including camera source, timestamps, duration, trigger type, file information, and sharing capabilities.	
Responsibilities	**Collaborators**
Store recording identification	
Store camera source	Camera
Store start and end timestamps	
Store recording duration	
Store trigger type	RecordingTrigger
Store file size and URL	
Store thumbnail URL	
Get stream URL for playback	

Get download URL	
Generate share link	
Get recording metadata	

Class: Alarm	
Alarm entity managing alarm state, verification status, timestamps, and resolution tracking. Records trigger event and resolving user.	
Responsibilities	**Collaborators**
Store alarm identification	
Store alarm type	AlarmType
Store alarm status	AlarmStatus
Store verification status	VerificationStatus
Track trigger event	SensorEvent
Track timestamps	
Record resolving user	User
Verify alarm	User
Escalate alarm	
Cancel alarm	User
Resolve alarm	
Get alarm status	AlarmStatus

Class: SensorEvent	
Sensor event entity recording individual sensor triggers with event type, timestamp, sensor value, and additional metadata.	
Responsibilities	**Collaborators**
Store event identification	
Store sensor identification	Sensor
Store event type	EventType
Store timestamp	
Store sensor value	

Store event metadata	
Get sensor ID	
Get event type	EventType
Get timestamp	

Class: Session	
User session entity tracking login state, device information, IP address, timestamps, and session validity with expiration handling.	
Responsibilities	**Collaborators**
Store session identification	
Store user identification	User
Store device information	
Store IP address	
Store user agent string	
Track creation timestamp	
Track expiration timestamp	
Track last activity	
Check if session valid	
Check if session expired	
Refresh session	
Revoke session	

Class: Notification	
Notification entity storing message details, type, priority, read status, and timestamps for tracking notification delivery and user engagement.	
Responsibilities	**Collaborators**
Store notification identification	
Store user identification	User
Store notification type	NotificationType
Store title and message	

Store priority level	Priority
Track creation timestamp	
Track read timestamp	
Mark notification as read	
Check if notification read	

Class: SafetyZone	
Security zone entity managing zone configuration including name, sensor membership, armed status, entry/exit delay settings, and sensor validation.	
Responsibilities	**Collaborators**
Store zone identification	
Store zone name	
Store sensor ID list	Sensor
Track armed status	
Track entry/exit delay flag	
Track timestamps	
Arm zone	
Disarm zone	
Check if armed	
Get sensor IDs	
Add sensor to zone	Sensor
Remove sensor from zone	Sensor
Check if contains sensor	Sensor
Determine if delay needed	
Validate sensor compatibility	Sensor

3. Frontend Cores

Class: SafeHomeHub	
Tablet application that hosts all ViewControllers and provides the main user interface for local SafeHome system control with offline mode support and landscape-optimized layout.	
Responsibilities	**Collaborators**
Host all ViewControllers for tablet interface	All ViewControllers
Provide larger touch targets for tablet	
Support offline mode operation	System
Optimize layout for landscape orientation	
Manage tablet-specific UI interactions	System
Initialize application and connect to backend	System
Sync data with backend when online	System

Class: SafeHomeMobile	
Mobile smartphone application with 4-tab navigation (Dashboard, Emergency, Devices, Settings) providing remote access to the SafeHome system with push notification support.	
Responsibilities	**Collaborators**
Host all ViewControllers for mobile interface	All ViewControllers
Implement 4-tab navigation structure	
Handle push notifications	NotificationManager
Optimize layout for portrait orientation	
Manage mobile-specific UI interactions	System
Register for push notifications	NotificationManager
Connect to backend API	System

4. Frontend ViewControllers

Class: DashboardViewController	
Main dashboard screen controller that displays system status, current security mode, recent events, device status, and allows quick mode changes.	
Responsibilities	**Collaborators**
Initialize dashboard view	SecurityModeManager, DeviceRegistry
Display current user information	User
Show current security mode	SecurityMode
Display recent sensor events	SensorEvent
Refresh dashboard data periodically	SecurityModeManager, NotificationManager
Handle security mode changes	SecurityModeManager
Update device status displays	Device, DeviceRegistry
Show notifications to user	Notification, NotificationManager

Class: CameraViewController	
Camera viewing and control UI that manages live video streaming, PTZ camera controls, two-way audio, and provides access to camera recordings.	
Responsibilities	**Collaborators**
Load camera details and settings	Camera
Start live video streaming	StreamingService, Camera
Stop live video streaming	StreamingService
Handle PTZ control commands	PTZControlService, PTZCommand
Toggle two-way audio on/off	StreamingService, Camera
Show camera recordings	RecordingManager
Verify camera password if protected	Camera
Manage live viewing state	

Class: SecurityZoneViewController	
Security zone configuration and management UI that allows users to create zones, assign devices to zones, and manage zone settings.	
Responsibilities	**Collaborators**
Load all safety zones	SecurityModeManager
Create new security zone	SecurityModeManager, SafetyZone
Edit existing zone properties	SafetyZone, SecurityModeManager
Delete security zone	SecurityModeManager
Add device to zone	SafetyZone, Device, DeviceRegistry
Remove device from zone	SafetyZone, Device
Display zone device lists	SafetyZone, Device
Show available devices for assignment	DeviceRegistry, Device
Validate device addition	Device, SafetyZone

Class: DeviceManagementViewController	
Device management UI for registering new devices, viewing device status, updating device settings, and removing devices from the system.	
Responsibilities	**Collaborators**
Display all registered devices	DeviceRegistry, Device
Show detailed device information	Device
Start device addition flow	DeviceRegistry
Update device settings	Device, DeviceConfigService
Remove device from system	DeviceRegistry, Device
Filter devices by category or type	DeviceRegistry
Sort devices by various criteria	

Class: EmergencyViewController	
Emergency response UI that provides panic button functionality, displays active alarms, handles alarm verification, and enables emergency service calls.	
Responsibilities	**Collaborators**
Display panic button interface	
Handle panic button press	AlarmManager
Show active alarms	Alarm, AlarmManager
Display alarm verification UI	Alarm
Confirm alarm as real threat	AlarmManager
Dismiss alarm as false alarm	AlarmManager
Call emergency services	AlarmManager, EmergencyServiceType

Class: UserAccountViewController	
User account management UI for viewing and editing profile information, changing passwords, managing permissions, and handling user logout.	
Responsibilities	**Collaborators**
Load user profile information	User, LoginManager
Update user profile details	User
Change user password	User, LoginManager
Manage user permissions	UserPermissionManager
Handle user logout	LoginManager

Class: RecordingViewController	
Recording management UI for searching recordings, playing back video, exporting recordings, sharing recordings via link, and deleting old recordings.	
Responsibilities	**Collaborators**
Search recordings by date and camera	RecordingManager
Display list of recordings	Recording, RecordingManager
Play selected recording	Recording, RecordingManager
Export recording to file	Recording, RecordingManager, ExportFormat

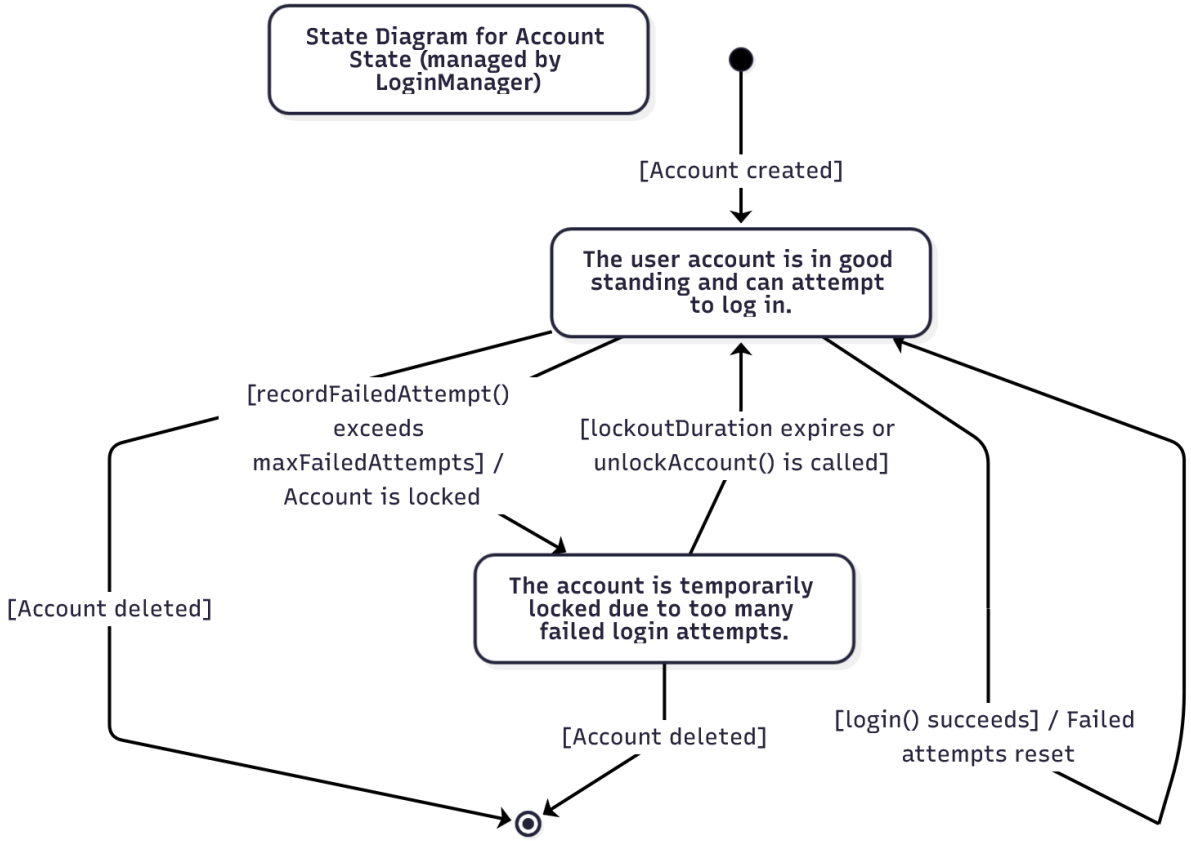
Share recording via link	Recording, RecordingManager
Delete recording	RecordingManager

Class: NotificationPanel	
Real-time notification display component that shows notifications, tracks unread count, allows marking as read, and manages notification settings.	
Responsibilities	**Collaborators**
Display real-time notifications	Notification, NotificationManager
Show unread notification count	Notification
Mark notification as read	Notification, NotificationManager
Clear all notifications	NotificationManager
Update notification settings	NotificationManager
Filter notifications by type	NotificationType

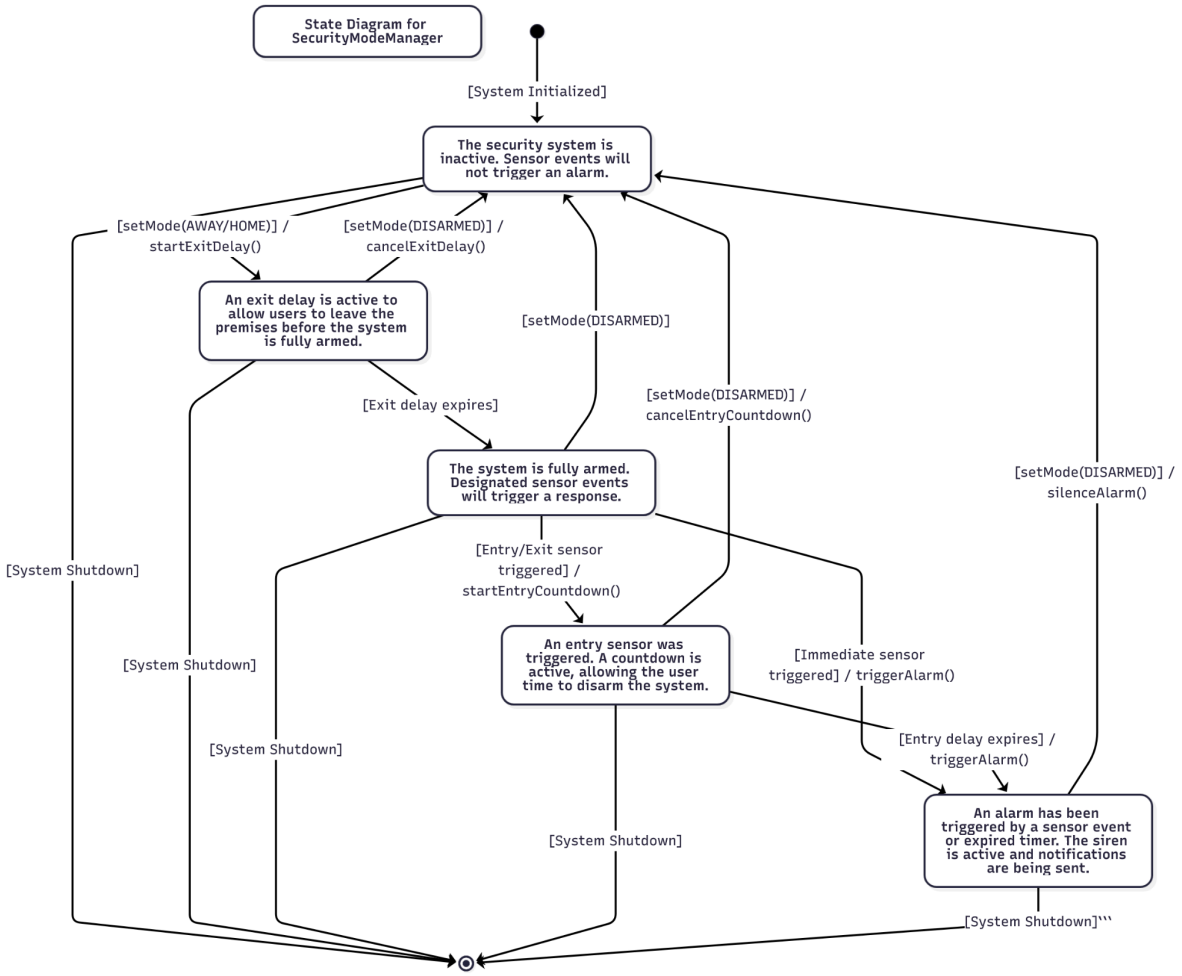
V. State Diagram

1 Business Logic Services

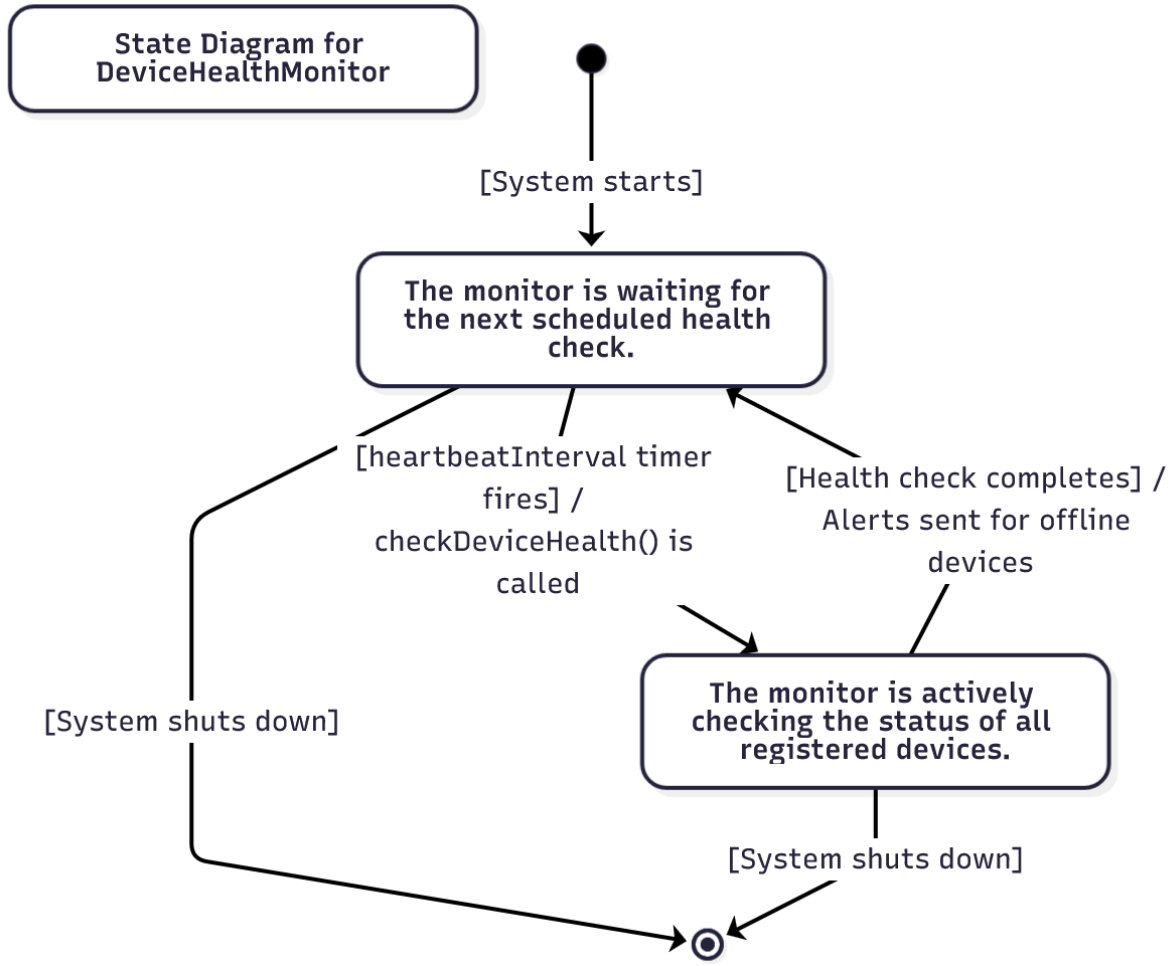
1.1 LoginManager



1.2 SecurityModeManager

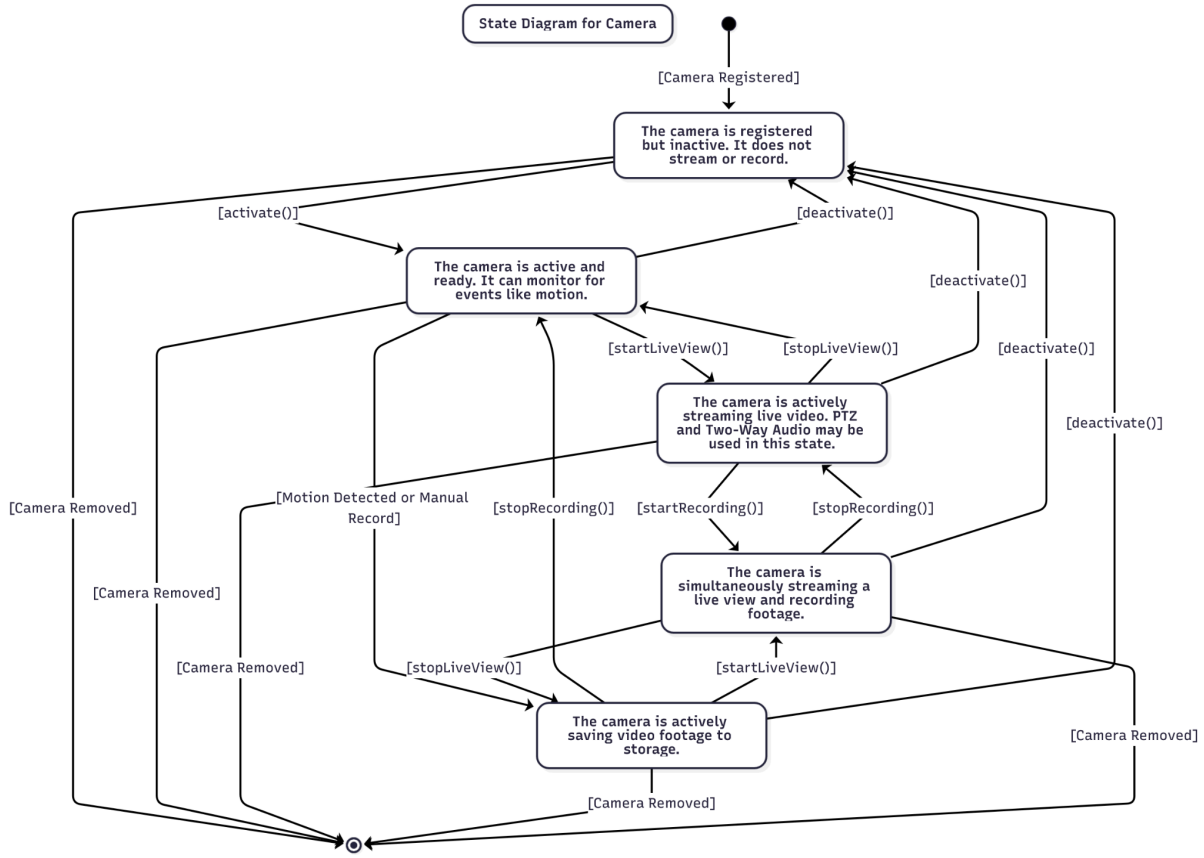


1.3 DeviceHealthMonitor

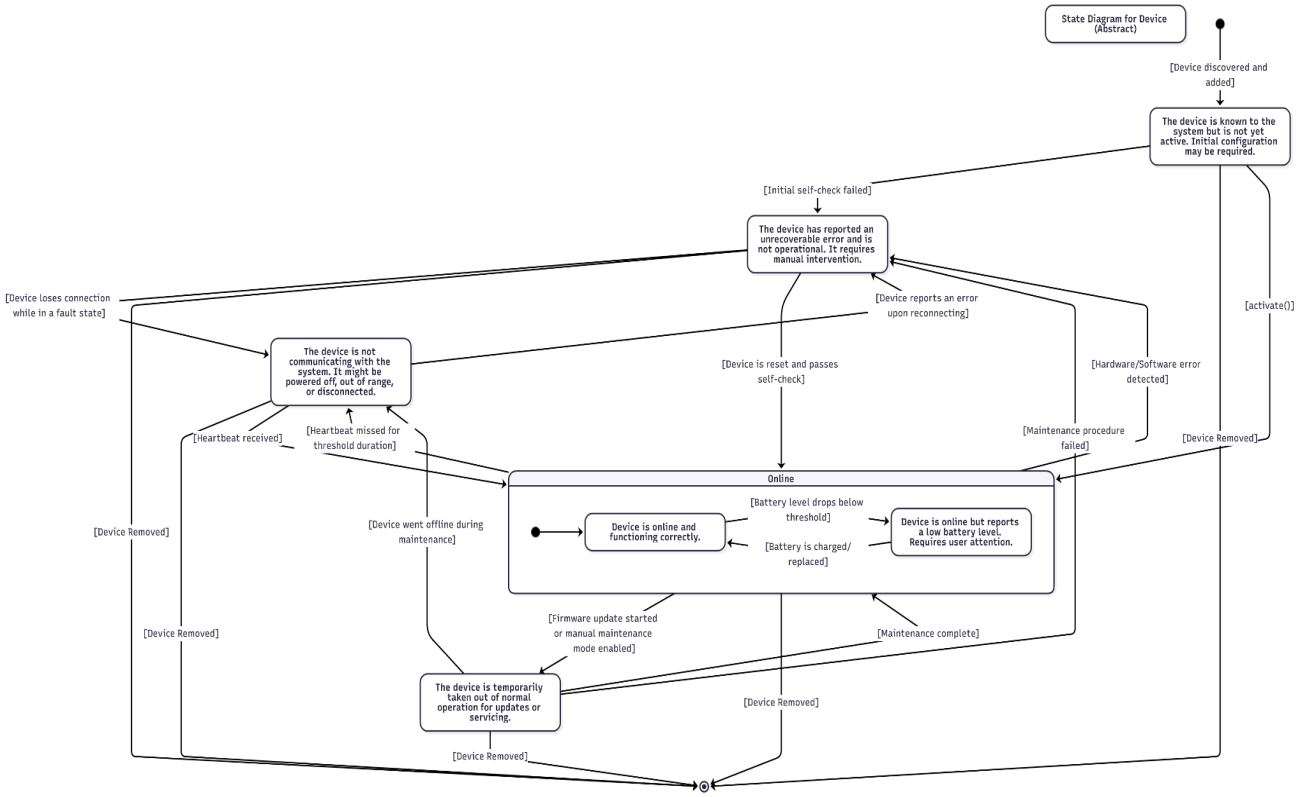


2 Domain Entities

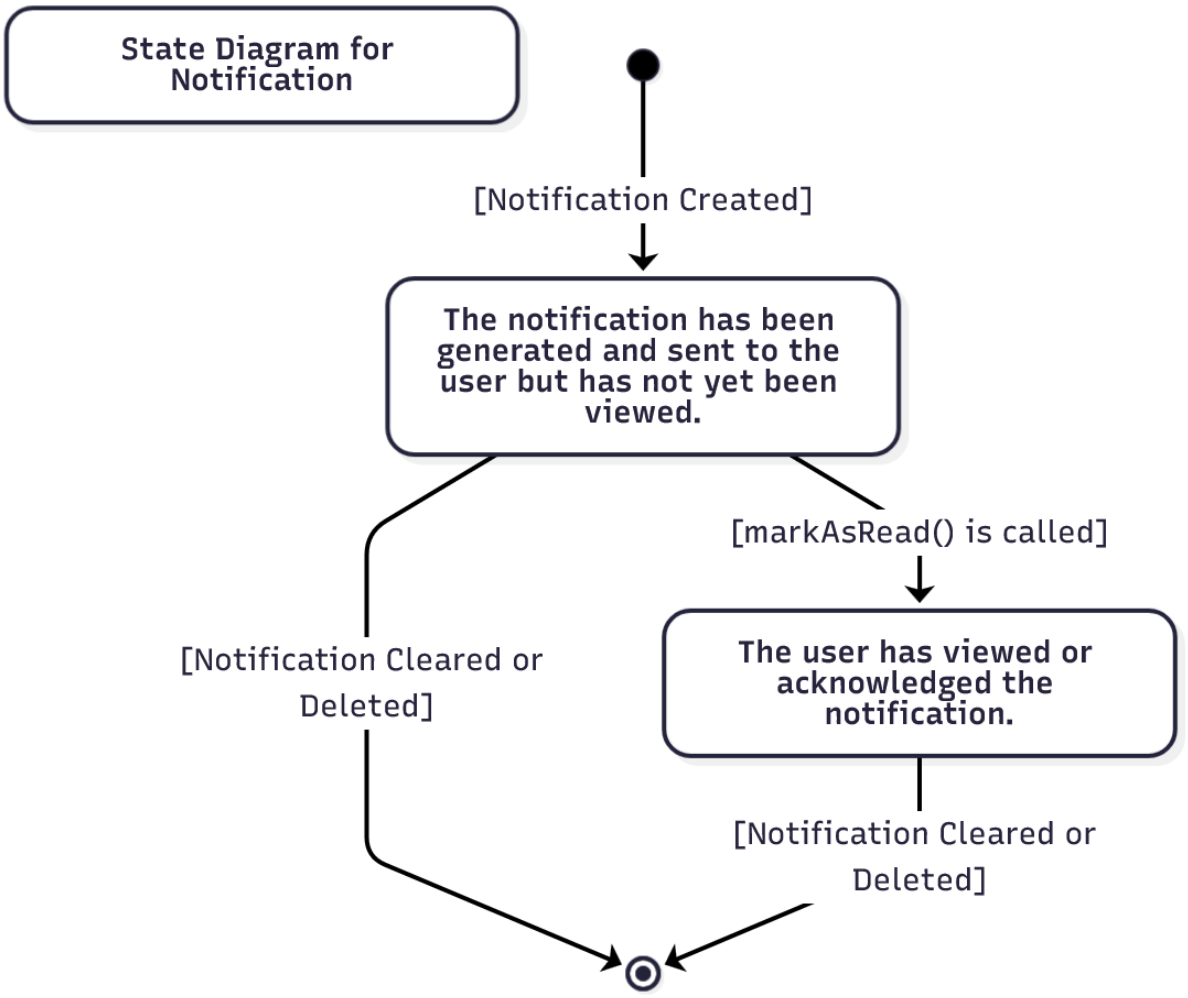
2.1 Camera



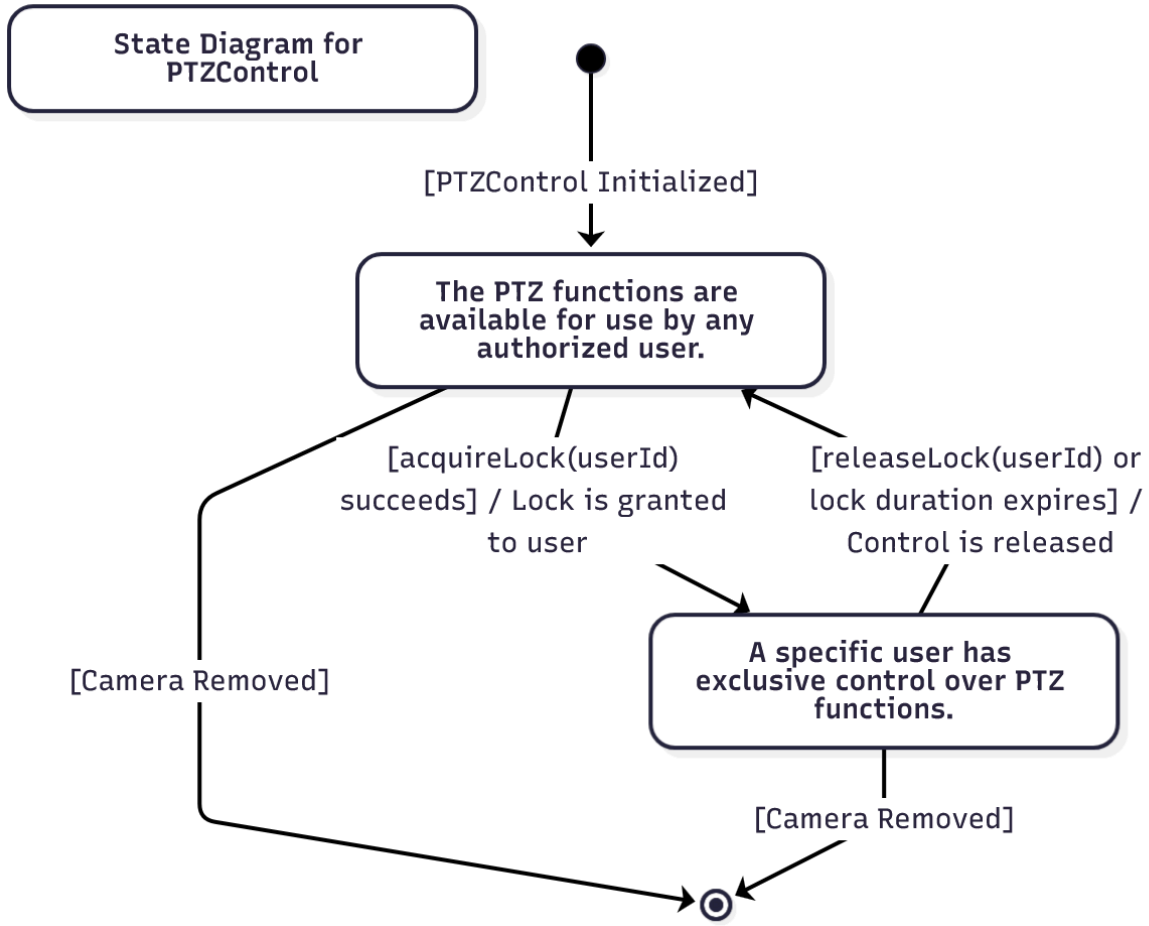
2.2 Device


State Diagram for Device (Abstract)
[Device discovered and added]
The device is known to the system but is not yet active. Initial configuration may be required.
[Initial self-check failed]
The device has reported an unrecoverable error and is not operational. It requires manual intervention.
[Device loses connection while in a fault state]
[Device reports an error upon reconnecting]
[activate()]
The device is not communicating with the system. It might be powered off, out of range, or disconnected.
[Device is reset and passes self-check]
[Hardware/Software error detected]
[Heartbeat received]
[Heartbeat missed for threshold duration]
[Maintenance procedure failed]
[Device Removed]
Online
[Battery level drops below threshold]
Device is online and functioning correctly.
Device is online but reports a low battery level. Requires user attention.
[Battery is charged/ replaced]
[Device Removed]
[Device went offline during maintenance]
[Firmware update started or manual maintenance mode enabled]
[Maintenance complete]
[Device Removed]
The device is temporarily taken out of normal operation for updates or servicing.
[Device Removed]
[Device Removed]

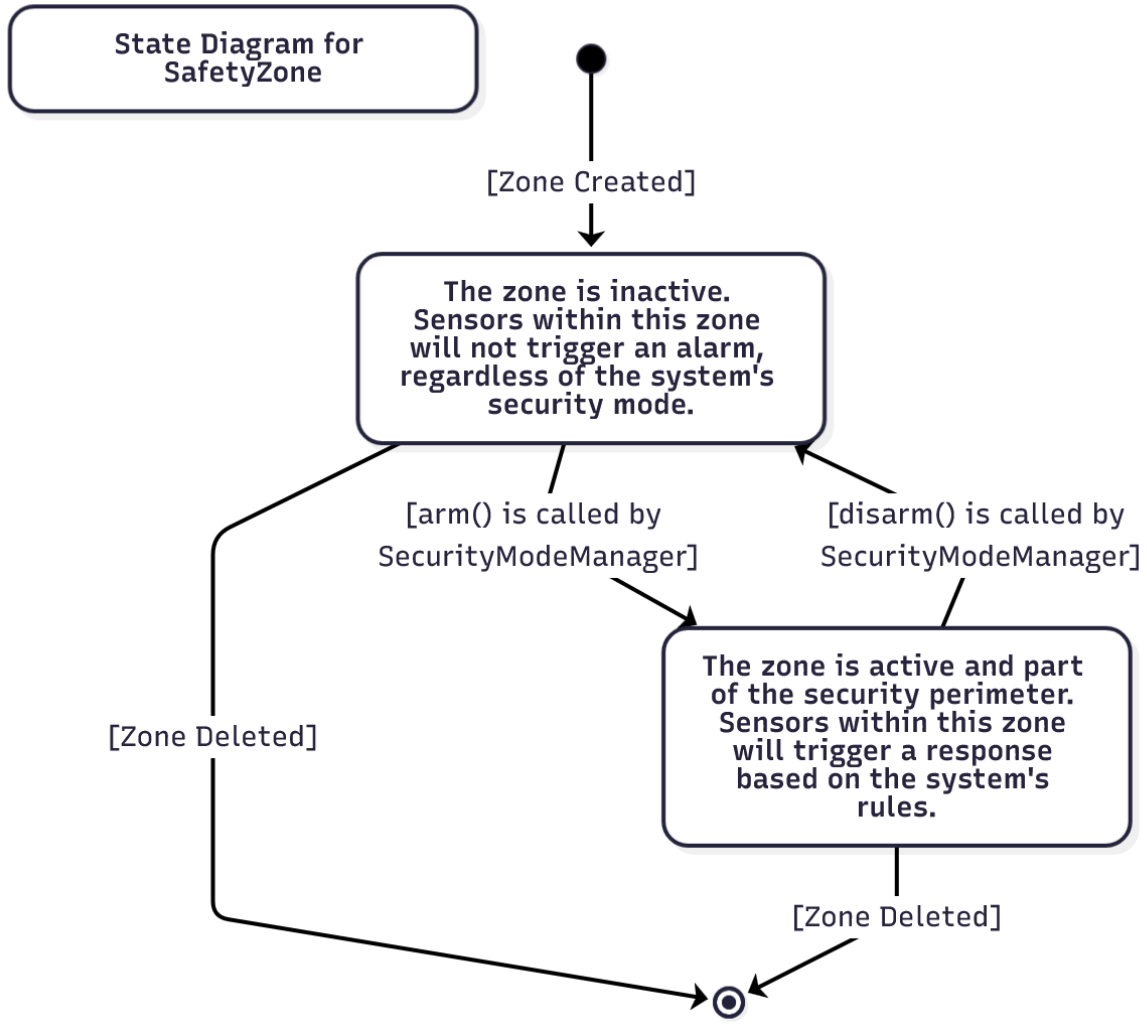
2.3 Notification



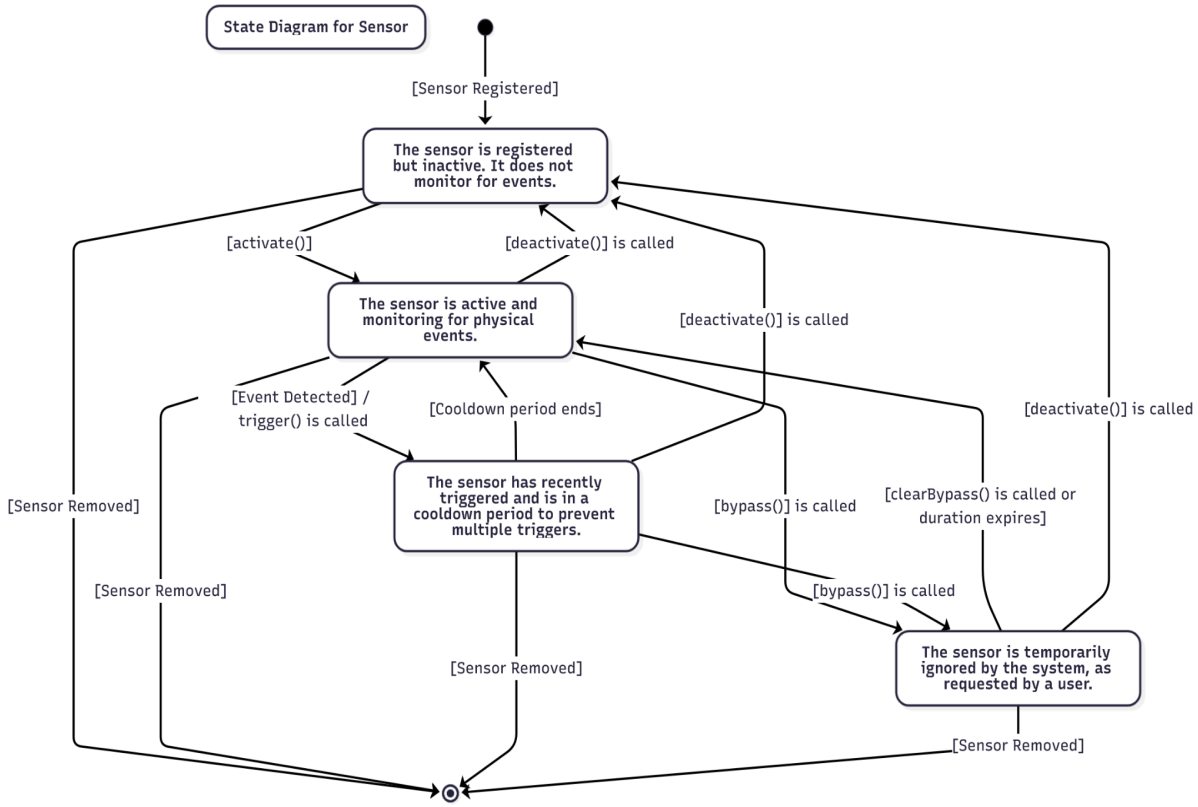
2.4 PTZControl



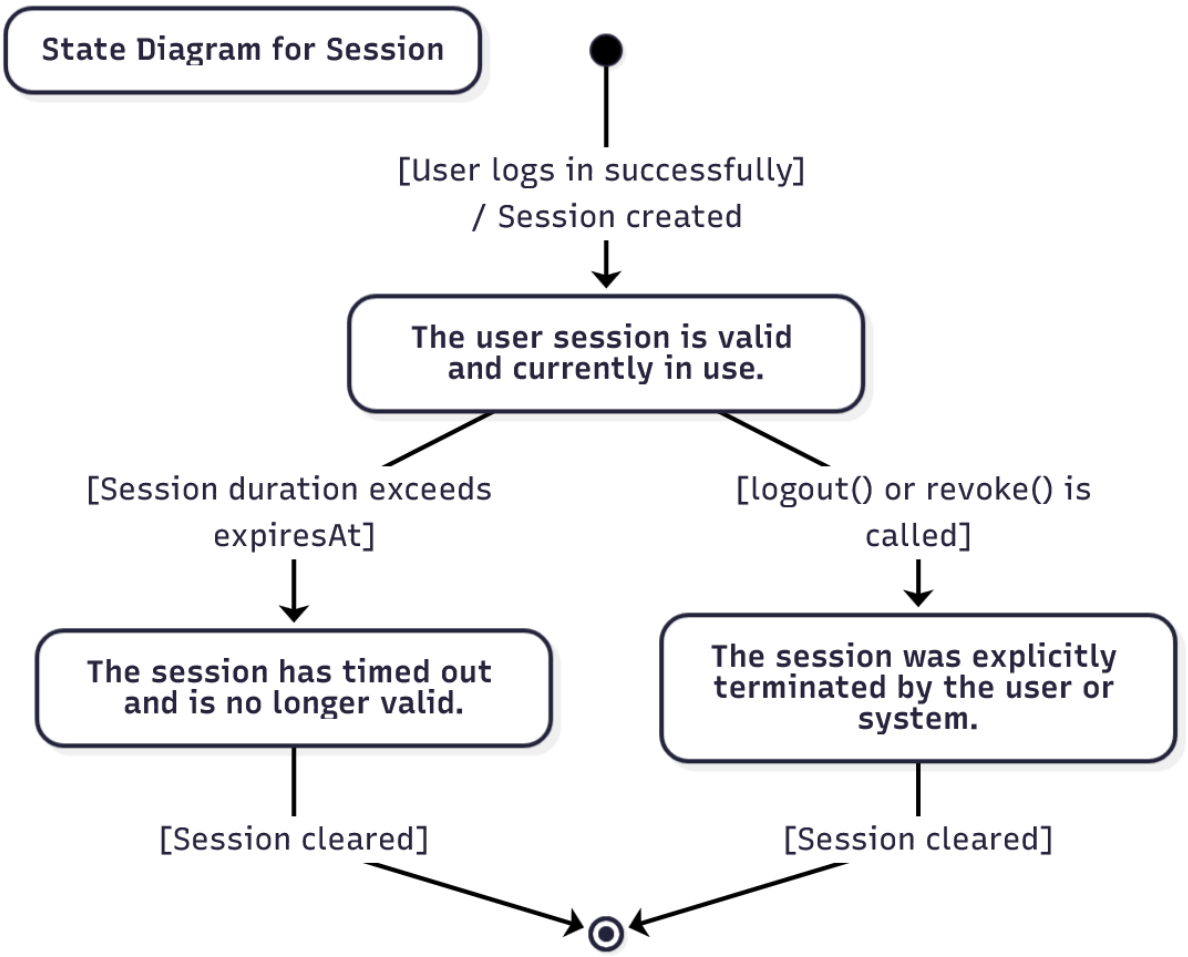
2.5 SafetyZone



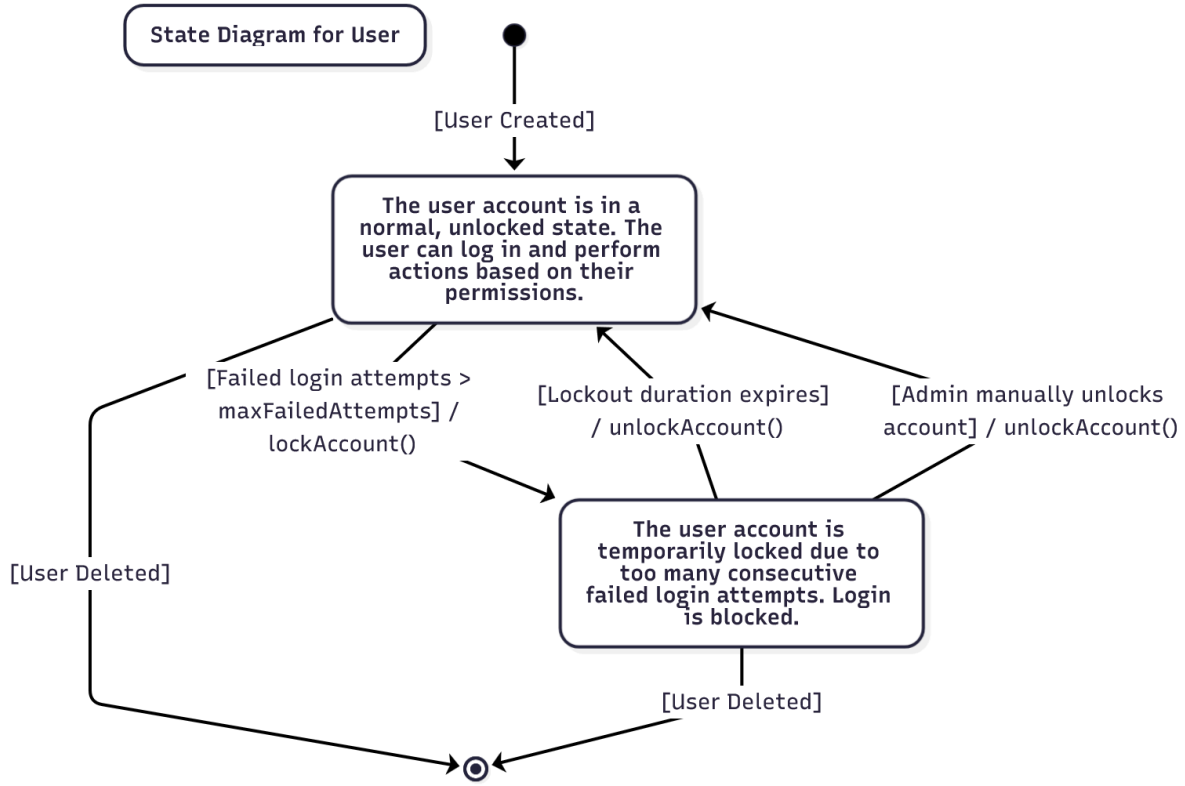
2.6 Sensor


State Diagram for Sensor
[Sensor Registered]
The sensor is registered but inactive. It does not monitor for events.
[activate()]
[deactivate()] is called
The sensor is active and monitoring for physical events.
[deactivate()] is called
[Event Detected] / trigger() is called
[Cooldown period ends]
[deactivate()] is called
[Sensor Removed]
The sensor has recently triggered and is in a cooldown period to prevent multiple triggers.
[bypass()] is called
[clearBypass() is called or duration expires]
[Sensor Removed]
[bypass()] is called
[Sensor Removed]
The sensor is temporarily ignored by the system, as requested by a user.
[Sensor Removed]

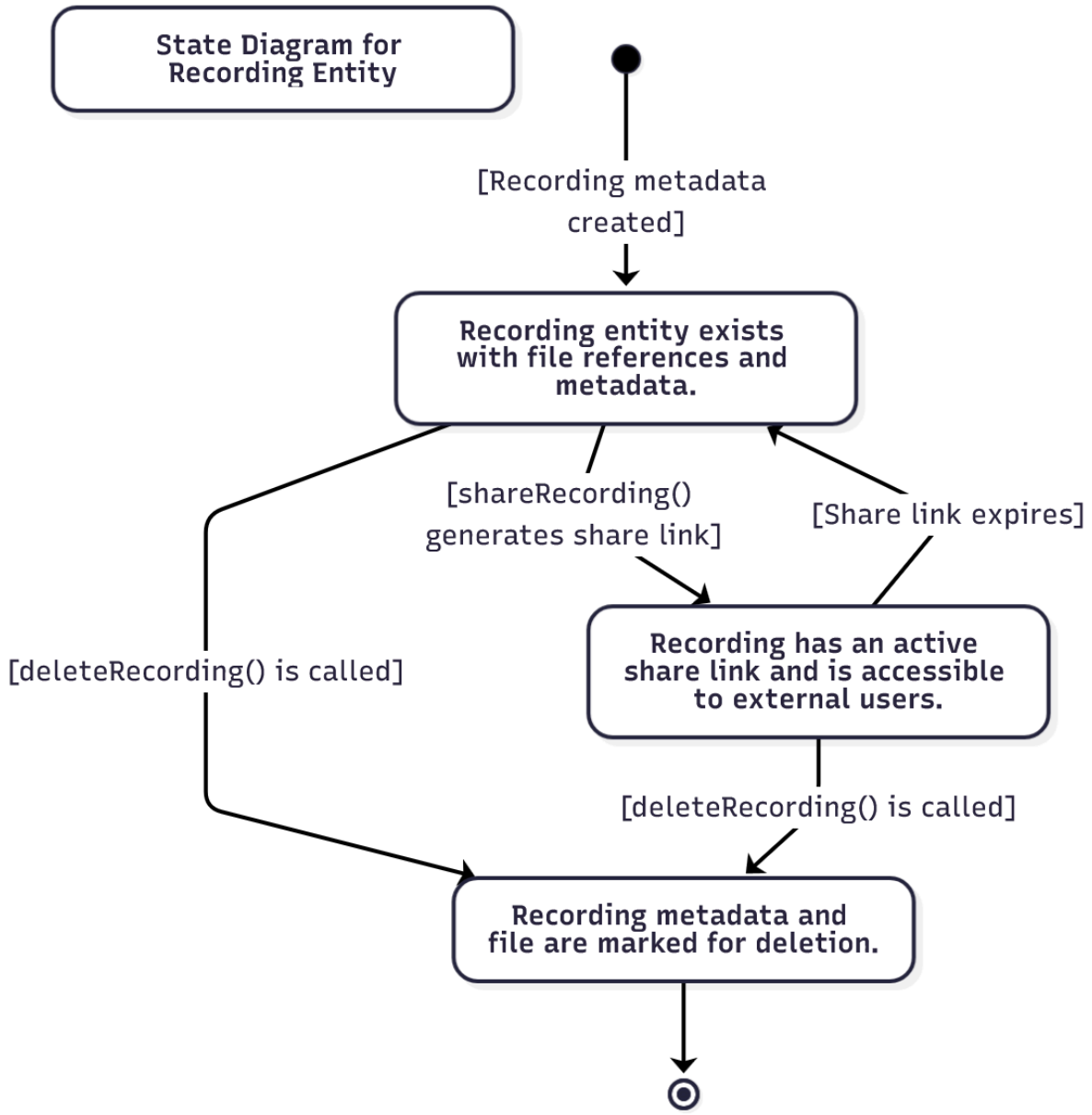
2.7 Session


State Diagram for Session
[User logs in successfully]
/ Session created
The user session is valid and currently in use.
[Session duration exceeds expiresAt]
[logout() or revoke() is called]
The session has timed out and is no longer valid.
The session was explicitly terminated by the user or system.
[Session cleared]
[Session cleared]

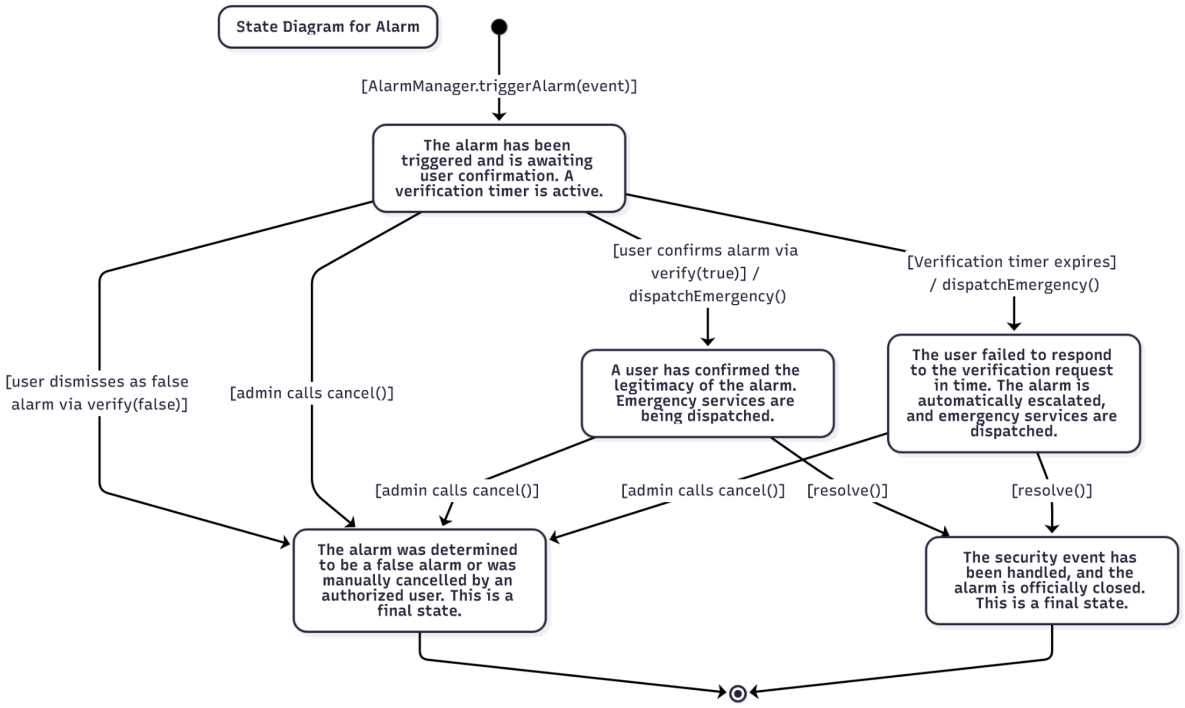
2.8 User


State Diagram for User
[User Created]
The user account is in a normal, unlocked state. The user can log in and perform actions based on their permissions.
[Failed login attempts > maxFailedAttempts] / lockAccount()
[Lockout duration expires] / unlockAccount()
[Admin manually unlocks account] / unlockAccount()
[User Deleted]
The user account is temporarily locked due to too many consecutive failed login attempts. Login is blocked.
[User Deleted]

2.9 Recording

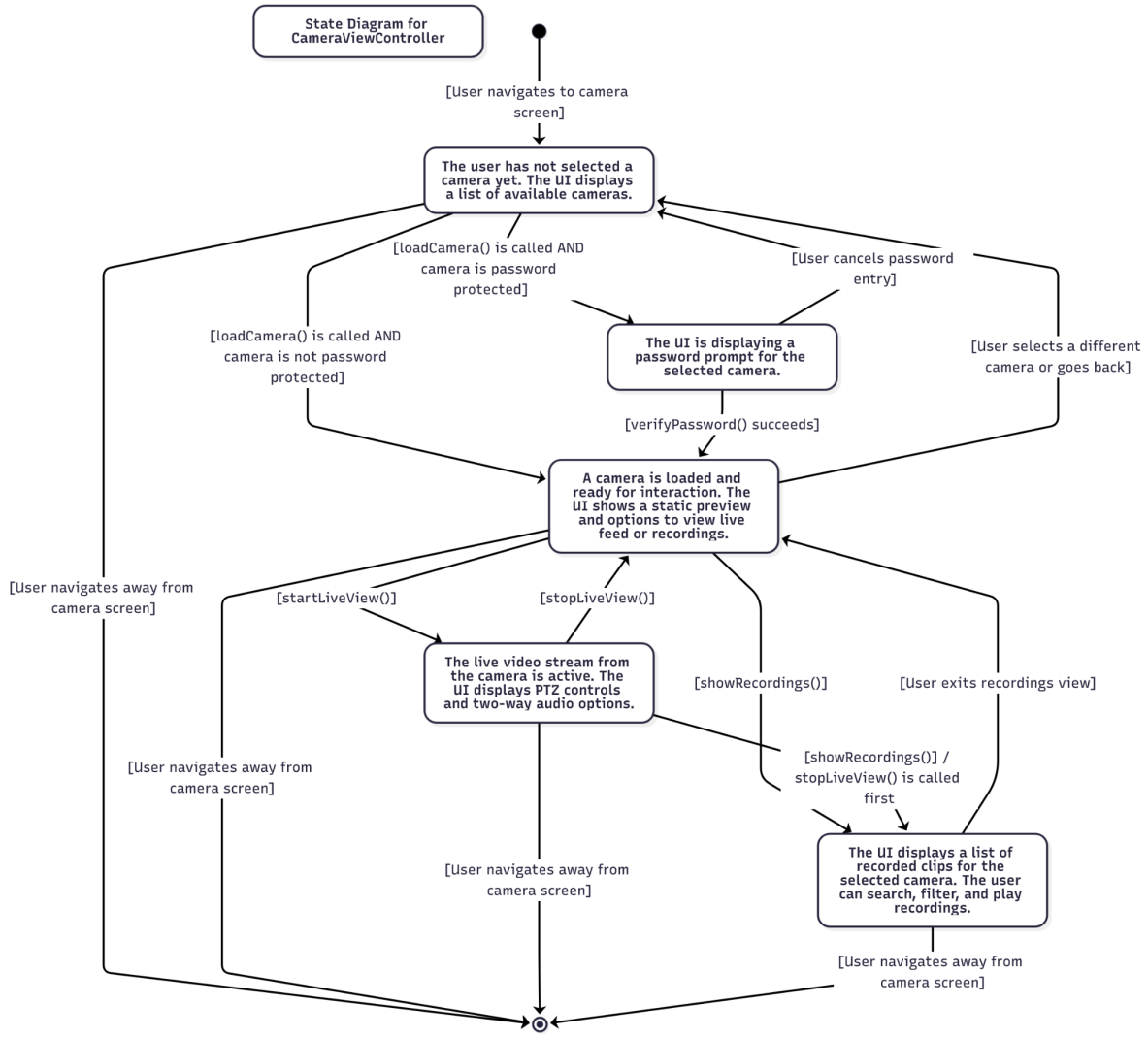


2.10 Alarm

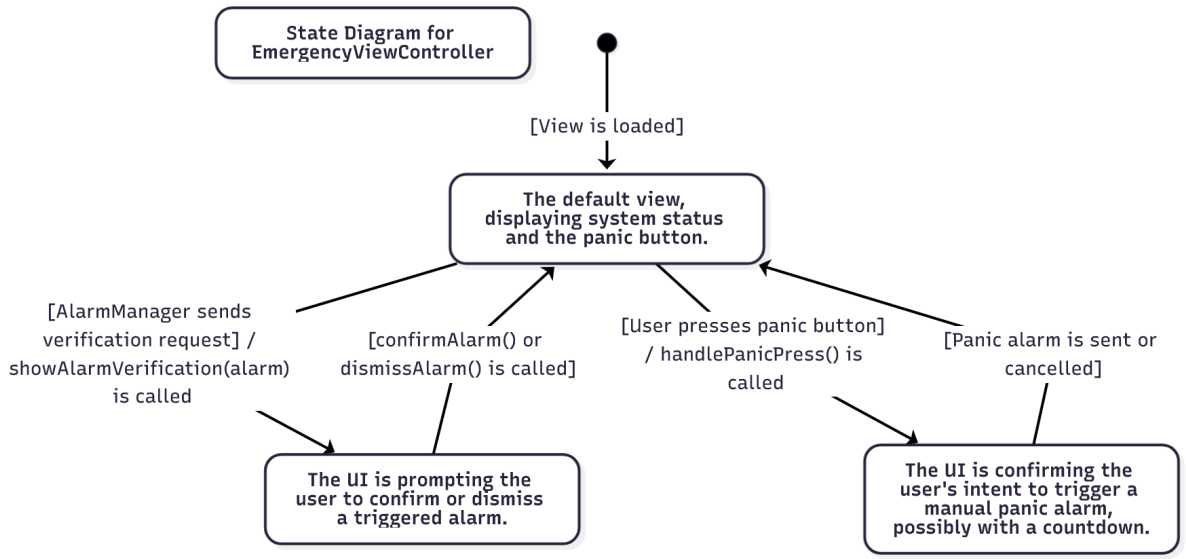

State Diagram for Alarm
[AlarmManager.triggerAlarm(event)]
The alarm has been triggered and is awaiting user confirmation. A verification timer is active.
[user dismisses as false alarm via verify(false)]
[admin calls cancel()]
[user confirms alarm via verify(true)] / dispatchEmergency()
[Verification timer expires] / dispatchEmergency()
A user has confirmed the legitimacy of the alarm. Emergency services are being dispatched.
The user failed to respond to the verification request in time. The alarm is automatically escalated, and emergency services are dispatched.
[admin calls cancel()]
[admin calls cancel()]
[resolve()]
[resolve()]
The alarm was determined to be a false alarm or was manually cancelled by an authorized user. This is a final state.
The security event has been handled, and the alarm is officially closed. This is a final state.

3 Frontend ViewControllers

3.1 CameraViewController



3.2 EmergencyViewController



VI. Design Evaluation

1 Architectural Design Metrics

1.1 Fenton's Simple Morphology Metrics

node	30	Total number of classes
arc	47	Total number of relationships
size	77	node + arc
depth	3	Maximum inheritance depth
width	11	Maximum classes at one level
arc-to-node ratio	1.57	Indicates connectivity

Analysis: Arc-to-node ratio of 1.57 indicates good connectivity without excessive coupling. Depth of 3 shows reasonable inheritance hierarchy.

2 CK Metrics

2.1 Depth of Inheritance Tree (DIT)

Device	0	Base class
Sensor	1	Extends Device
Camera	1	Extends Device
User	0	Base class
SecurityModeManager	0	Service class
Average DIT	0.7	Shallow hierarchy

2.2 Number of Children (NoC)

Device	2	Sensor, Camera
Sensor	4	4 sensor subtypes
Camera	0	Leaf class
User	0	No subclasses
Average NoC	1.2	Balanced

2.3 Coupling Between Object Classes (CBO)

SecurityModeManager	5	Multiple collaborators
AlarmManager	4	Core security functions
RecordingManager	3	Recording management
DashboardViewController	6	Central UI controller
Device	2	Device types
User	3	User management
Average CBO	3.8	Good coupling (< 6)

2.4 Lack of Cohesion in Methods (LCOM)

SecurityModeManager	0.2	High cohesion
AlarmManager	0.1	Very high cohesion
User	0.15	High cohesion
RecordingManager	0.25	Good cohesion
Camera	0.3	Acceptable cohesion
Average LCOM	0.20	Excellent (< 0.5)

3 MOOD Metrics

3.1 Method Inheritance Factor (MIF)

Class	Md (Defined)	Mi (Inherited)	Ma (Total)
Device	10	0	10
Sensor	8	10	18
MotionSensor	1	18	19
DoorWindowSensor	1	18	19
Camera	13	10	23
User	5	0	5
Alarm	5	0	5
Total	43	56	99

MIF = Mi / Ma = 56 / 99 = 0.566

Analysis: MIF of 0.566 indicates good inheritance utilization. About 56% of methods are inherited, showing effective code reuse.

3.2 Coupling Factor (CF)

Class List with Numbers

No.	Class Name	Layer
1	DashboardViewController	Presentation
2	CameraViewController	Presentation
3	SecurityZoneViewController	Presentation
4	DeviceManagementViewController	Presentation
5	EmergencyViewController	Presentation
6	UserAccountViewController	Presentation
7	RecordingViewController	Presentation

8	NotificationPanel	Presentation
9	SecurityModeManager	Business
10	AlarmManager	Business
11	RecordingManager	Business
12	NotificationManager	Business
13	DeviceRegistry	Business
14	DeviceHealthMonitor	Business
15	DeviceConfigService	Business
16	StreamingService	Business
17	PTZControlService	Business
18	LoginManager	Business
19	UserPermissionManager	Business
20	User	Data
21	Device	Data
22	Sensor	Data
23	Camera	Data
24	PTZControl	Data
25	Recording	Data
26	Alarm	Data
27	SensorEvent	Data
28	Session	Data
29	Notification	Data
30	SafetyZone	Data

Coupling Matrix (30 x 30)

Note: Row represents 'from' class, Column represents 'to' class. 1 = dependency exists, 0 = no dependency

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CF Calculation

Total possible couplings: 30 × (30 - 1) = 870

Actual couplings (from design): 41

CF = 41 / 870 = 0.047

Analysis:

- CF of 0.047 indicates low coupling
- Only 4.7% of possible couplings are used
- Target: < 0.1 ✓
- Excellent design with minimal coupling

3.3 Polymorphism Factor (PF)

Base Class	Total Methods	Subclasses	Overridden Methods	Po
Device	10	2	3 (activate, deactivate, sendHeartbeat)	3
Sensor	8	4	2 (trigger)	2

Total Mo (overridable methods): 10 × 2 + 8 × 4 = 52

Total Po (actually overridden): 3 + 2 = 5

PF = Po / Mo = 5 / 52 = 0.096

Analysis: PF of 0.096 is low but acceptable for this domain. Not all methods need polymorphic behavior. Focus is on composition over deep polymorphism.

4 Design Quality Summary

Metric Category	Score	Target	Status
Morphology			
Arc-to-node ratio	1.57	1.0-2.0	✅ Good
Depth	3	≤ 5	✅ Excellent
Width	11	5-15	✅ Good
CK Metrics			
Average DIT	0.7	≤ 3	✅ Excellent
Average NoC	1.2	1-3	✅ Good
Average CBO	3.8	< 6	✅ Good
Average LCOM	0.20	< 0.5	✅ Excellent
MOOD Metrics			
MIF	0.566	0.3-0.7	✅ Good
CF	0.054	< 0.1	✅ Excellent
PF	0.096	0.1-0.5	✅ Acceptable

Overall Design Quality: GOOD ✅

The design demonstrates:

- Low coupling (CF = 0.054)
- High cohesion (LCOM = 0.20)
- Reasonable inheritance (MIF = 0.566)
- Shallow hierarchy (DIT = 0.7)
- Balanced architecture

VII. Who Did What

MinJun Kim

1. Class Diagram Visualization and Design
    - Created professional Mermaid-based class diagrams with clean visual styling that was adopted as the team standard
    - Designed the visual layout and formatting guidelines for all UML diagrams ensuring consistency and readability
    - Implemented automated diagram generation scripts to improve workflow efficiency
2. Presentation Layer Architecture
    - Defined and documented all 8 ViewController classes (DashboardViewController, CameraViewController, SecurityZoneViewController, etc.)
    - Designed the MVC pattern implementation for the presentation layer based on UI/UX expertise
    - Created detailed specifications for user interface components and their interactions with business logic
3. CRC Cards and Documentation
    - Authored CRC cards for all presentation layer classes defining responsibilities and collaborations
    - Contributed to the standardization of class definition format across the entire document

Jehyung Kim

1. Surveillance and Camera System Design
    - Defined complete camera-related business logic classes including StreamingService, PTZControlService, and RecordingManager
    - Designed the Camera and Recording entity classes with comprehensive attribute definitions
    - Specified protocols for camera activation/deactivation and password protection mechanisms
2. Priority Conflict Analysis and Resolution
    - Identified and documented priority conflicts between camera locking, disabling, and viewing operations
    - Proposed resolution strategies for user permission conflicts (Owner vs Guest access)

- Contributed to the development of system-wide command priority rules

3. System Integration and State Management
    - Designed state diagrams for Recording and Session entities showing lifecycle transitions
    - Specified integration points between surveillance components and the broader system architecture

Onejune Lee

1. Project Leadership and Methodology
    - Led all project meetings and coordinated team efforts throughout the SDS development process
    - Proposed and implemented the AI-assisted class extraction methodology from use case scenarios
    - Created the systematic workflow for extracting nouns and verbs from use cases to identify classes and methods
2. System Architecture and Priority Rules
    - Designed the three-layer architecture (Presentation, Business Logic, Data) ensuring separation of concerns
    - Authored the Global System Priority Rule document defining command precedence (Security > User Input > Automation)
    - Defined core business logic classes including SecurityModeManager, AlarmManager, and NotificationManager
3. Terminology Standardization and Quality Assurance
    - Merged and unified all terminology from four team members' extractions into a single master list
    - Standardized inconsistent terms across the document (e.g., 'Sign up' vs 'Create Account', 'Cloud Server' vs 'System Hub')
    - Conducted comprehensive review and quality assurance of the final document
4. Design Metrics and Evaluation
    - Calculated all design metrics including Fenton's morphology, CK metrics, and MOOD metrics
    - Created coupling matrices and dependency analysis for the system architecture
    - Authored the Design Evaluation section with detailed metric interpretations

Yunje Lee

1. User Management and Authentication System
    - Designed the complete authentication and authorization architecture including LoginManager and UserPermissionManager
    - Defined User and Session entity classes with comprehensive security attributes
    - Specified protocols for account creation, login, logout, password recovery, and two-factor authentication
2. Indoor Monitoring and Device Control
    - Designed DeviceRegistry and DeviceHealthMonitor classes for device lifecycle management
    - Created specifications for indoor air quality monitoring and power consumption tracking features
    - Defined the Device entity class hierarchy and device type enumeration
3. Concurrency and Multi-Device Access
    - Identified critical concurrency issues including simultaneous login from multiple devices
    - Proposed session management strategies to handle multi-device authentication scenarios
    - Analyzed and documented potential race conditions in device control operations
4. Use Case to Class Mapping
    - Created detailed mappings between use case scenarios and class definitions for Parts 1 and 5
    - Contributed to the extraction methodology by testing AI prompts for noun/verb identification

VIII. Meeting Logs

Date	2025/11/5
Location	Online(via google meet)
Attendees	Attendee 1: Yunje Lee Attendee 2: Onejune Lee Attendee 3: Minjun Kim Attendee 4: Jehyung Kim
Summary	1. Meeting Objective (Inferred) To address missing features based on professor feedback, discuss and finalize specific methods for designing the class diagram based on the existing SRS (Software Requirements Specification) and use cases, and to identify and explore solutions for priority conflict issues within the system. 2. Key Discussion Points Class Diagram Methodology ● Onejune Lee: Proposed a method to derive the class diagram based on the previously written use case scenarios. ● Onejune Lee: Explained the approach: identify **nouns** from the scenarios as potential **classes or attributes**, and identify **verbs** as potential **methods**. ● Onejune Lee: Suggested that this task be done individually for each member's assigned part, after which they would decide on a tool. He also noted that they could get the hang of it quickly by using the textbook or consulting GPT. Addressing Feedback and Missing Features ● Onejune Lee: Raised the issue that the 'system-wide turn on/off' feature and the 'PTZ (Pan/Tilt/Zoom)' feature seemed to be missing. ● Yunje Lee: Agreed that the PTZ feature was likely dropped during a previous reorganization of the use cases. ● Onejune Lee: Summarized that he would add the simple 'on/off' feature, and that Jehyung could add the PTZ feature

based on existing materials, which would cover most of the feedback.

'Safe Zone' (Security Zone) and Priority Conflicts

- Onejune Lee: Mentioned that the professor had intentionally given vague guidance on the 'Safe Zone' initially but had emphasized its importance in a recent class, indicating it needed to be added.
- Onejune Lee: Described the 'Safe Zone' as a feature for grouping devices like cameras and sensors to turn them on or off at once.
- Onejune Lee: Provided examples of potential priority conflicts this could cause:
    - When a single sensor spans both an "on" zone and an "off" zone.
    - When two zones overlap.
    - When a single camera covers multiple zones with different settings.
- Onejune Lee: Emphasized that if these priority settings weren't handled properly, problems would arise during test cases.
- Minjun Kim: In response, suggested handling these edge cases with assumptions, such as "there will be no overlapping zones" or "if one sensor is off, another sensor will cover its role."
- Onejune Lee: Added that a general review of priorities was also needed, including user permission hierarchies and conflicts between user commands and system alarms.

Scheduling the Next Meeting

- Yunje Lee: Informed the team he would be in Busan from tomorrow (Thursday) through Sunday and would have to join via Zoom if a meeting was held before then.
- Onejune Lee: Proposed scheduling the meeting to avoid his own study session on Friday evening (7 PM - 9 PM).
- Onejune Lee, Yunje Lee: Agreed to hold the next meeting on Friday from 2:00 PM to 4:00 PM.

3. Decisions Made

	1. Each team member will analyze their assigned use case scenarios to extract nouns (classes/attributes) and verbs (methods). 2. The missing features ('system on/off', 'PTZ') will be added to the use cases. 3. The 'Safe Zone (Security Zone)' feature will be defined, and all potential priority conflicts related to it will be reviewed. 4. Each team member will review and document potential priority conflicts within their own part and with other parts. 5. The next meeting will be held on **Friday from 2:00 PM to 4:00 PM**. Yunje Lee will attend via Zoom. 4. Action Items ● **Onejune Lee:** Write the use case for 'system on/off'. ● **Jehyung:** (As mentioned by Onejune Lee) Supplement the use cases with the missing PTZ feature. ● **All Team Members:** ○ Extract and organize nouns/verbs from their respective use cases. ○ Review and document priority issues, classifying them into three types (undefined, conflict within part, conflict with other parts). ○ Attempt to draw one class diagram each (recommending a code-based tool). ● **Prepare for Next Meeting:** Complete the above items by 2:00 PM on Friday.
AI Note	https://clovanote.naver.com/s/YReVHf7ZnEXYoT7qAVFaM8S 비밀번호: wk93zs

Date	2025/11/7
Location	Online(via google meet)
Attendees	Attendee 1: Onejune Lee Attendee 2: Yunje Lee Attendee 3: Minjun Kim Attendee 4: Jehyung Kim
Summary	1. Meeting Objective To review the initial noun/verb extractions and priority analysis assignments from the previous meeting, refine the methodology for identifying classes and methods, and establish a clear plan for consolidating all findings into a unified model for the class diagram. 2. Key Discussion Points Review of Noun/Verb Extraction (Class/Method Identification) ● Initial Review & Format: Onejune Lee (Attendee 1) reviewed the team's initial extractions and admitted he hadn't provided a clear enough format. ● Refined Methodology: Onejune Lee explained that simply extracting nouns and verbs was insufficient. The correct process involves classifying nouns (e.g., Entity, Role, Event) and deciding whether to "Accept" or "Reject" them for the model. ● "Accept/Reject" Criteria: The team was confused about the "Accept/Reject" step. Onejune Lee couldn't find the guiding textbook section (8.7) , but the team hypothesized that "Attributes" should be "Rejected," as the goal is to find "Classes" and "Methods." They also concluded the 2007 example they were following likely involved heavy manual rejection because they lacked AI, whereas the team can use AI to pre-filter for relevant terms. ● Terminology Consolidation: Onejune Lee noted that the use cases have many inconsistent terms for the same concept (e.g., "Sign up" vs. "Create Account"; "Cloud Server" vs. "System Hub"). These must be unified. ● Consolidation Workflow: Jehyung Kim (Attendee 4) pointed out that if everyone extracts terms separately, merging them

would be very difficult. Onejune Lee worried that one person doing everything would be too large a task, as his part alone had ~500 items.

- Workflow Solution: The team agreed on a multi-step process:
    1. Each member re-extracts all nouns/verbs from their *entire* use case (including exceptions) and removes duplicates *within their own part*.
    2. One person (Onejune Lee) will then merge the four resulting lists, unifying the terminology to create a single master list of entities.

Review of Priority Analysis

- Jehyung Kim (Part 2 - Cameras): Found conflicts *within* his use cases, such as the need to define priority between 2.1.1, 2.1.3, and 2.1.4. The main issue is that logic to check if a camera is locked or disabled must precede all other actions. He also identified conflicts with other parts related to user permissions (Owner vs. Guest).
- Minjun Kim (Part 3 - User/Account): His part is foundational, and all other parts depend on its permission structures. It has significant conflicts, especially with Part 1 (sensors).
- Yunje Lee (Part 1 & 5 - Login/Devices): Initially, Yunje Lee (Attendee 2) thought his parts (login, gas, light) had no conflicts as they use separate sensors. However, Onejune Lee prompted him to consider concurrency issues. After checking with AI, Yunje Lee found several critical issues:
    1. Simultaneous login from multiple devices.
    2. How to handle existing sessions on other devices when a password is changed.
    3. Handling 2-Factor Authentication initiation during a separate login attempt.
- Onejune Lee (Part 1 - System): Also identified significant permission-related issues. He proposed creating a "Global System Priority Rule" document to handle major conflicts (e.g., a fire and a break-in occurring simultaneously).

Project Planning

- Deadline: The team confirmed the project deadline is one week

away (next Friday).

- The Core Task: Onejune Lee emphasized that completing the class/method/attribute extraction is the "alpha and omega" of the project. Once this is done, the CRC cards and architecture diagrams can be easily generated from it.
- Diagram Order: Onejune Lee clarified that the detailed, low-level class definitions must come *before* the high-level architecture diagram to prevent dependency and design principle violations.

3. Decisions Made

1. The team will immediately re-do the noun/verb extraction during the meeting.
2. The extraction must cover the *entire* use case text, including all scenarios, exceptions, and other sections.
3. The extraction will be done in English from the start.
4. Each member will extract from their part and perform an initial de-duplication.
5. Onejune Lee will be responsible for merging the four lists and unifying all terminology.
6. The "Accept/Reject" step will be simplified by relying on AI to filter for relevant classes/methods and by classifying attributes as "Rejected."
7. Onejune Lee will create a "Global System Priority Rule" document for the final report.

4. Action Items

- All Attendees: (During the meeting) Re-extract all nouns and verbs from their assigned use case sections using the new, detailed AI prompt (as defined by Onejune Lee in the "SDS" sheet).
- All Attendees: (During the meeting) Internally de-duplicate their own extraction lists.
- Onejune Lee: (Post-meeting) Merge the four de-duplicated lists into one master list, unifying all terminology.
- Onejune Lee: (During the meeting) Create the new extraction template/format in the shared "SDS" Google Sheet.
- Onejune Lee: (Post-meeting) Draft the "Global System Priority

	Rule" document
AI Note	https://clovanote.naver.com/s/uE475cTsfHQUx7QDBF25PWS 비밀번호: dcmmtg

Date	2025/11/9
Location	Online(via google meet)
Attendees	Attendee 1: Onejune Lee Attendee 2: Minjun Kim Attendee 3: Yunje Lee Attendee 4: Jehyung Kim
Summary	1. Meeting Objective (Inferred) To identify problems with the currently defined class diagrams, assign tasks to "flesh out" the class definitions to an implementation-ready level, and decide on a tool to solve diagram layout issues. 2. Key Discussion Points Architecture Diagram Review ● **Onejune Lee:** Shared that he had generated a draft of the architecture diagram (using a mix of GPT and Gemini) and this part seems to be mostly resolved. Problem 1: Class Diagram "Overview" Layout ● **Onejune Lee:** The biggest issue is the "Class Overview" diagram. Code-based tools like PlantUML simply list classes from left-to-right or top-to-bottom, failing to create a balanced, human-like layout as seen in the examples. ● **Onejune Lee:** This problem will get worse when all class definitions are included. ● **Minjun Kim:** Proposed a solution: use a tool like draw.io or Smart Draw (infoUML) that allows for manual drag-and-drop placement of objects after code generation. ● The team accepted this (using a tool with manual layout) as the solution.

	Problem 2: Need for "Implementation-Level" Class Definitions (Fleshing Out) ● Onejune Lee: Pointed out that the current class definitions are based only on the core features extracted from the SRS. ● Onejune Lee: As a result, essential classes for the system's actual operation (e.g., login interfaces, UI components) are missing. ● The team concluded that the priority task is to "flesh out" the current class list to a "pre-coding" level of detail. ● This "fleshing out" must also include the frontend (UI) portion, which is currently undefined. 3. Decisions Made (Task Distribution) The team, realizing their old roles were no longer valid, divided the "class fleshing-out" task into four new parts. 1. Frontend (UI) Detail: Minjun Kim (This was seen as efficient since he already had related code). 2. Backend Part 1 (Detail): Yunje Lee. 3. Backend Part 2 (Detail): Jehyung Kim. 4. Backend Part 3 (Detail): Onejune Lee. ● Process Suggestion (Onejune Lee): He suggested it might be faster and more accurate to have an AI write the *implementation code* first, and then extract the class definitions *from* that code, rather than designing the classes from scratch. 4. Action Items ● All Team Members: Complete the "class fleshing-out" (in document form) for their assigned part before the next meeting. ● Next Meeting: Tomorrow (November 10) at 10:00 PM. ● Next Meeting Objective: To conduct a peer review of the fleshed-out class definitions and discuss strategies for class dependency reduction.
AI Note	https://clovanote.naver.com/s/MHDa89Hkx7sueqKWJWw6ZeS 비밀번호: dwfyzg

Date	2025/11/11
Location	E3-1 2401
Attendees	Attendee 1: Onejune Lee Attendee 2: Yunje Lee Attendee 3: Jehyung Kim Attendee 4: Minjun Kim
Summary	1. Meeting Objective To review the auto-generated class diagrams, finalize the visual style and content, review remaining project tasks, and assign clear ownership for the final document components. 2. Key Discussion Points Finalizing the Class Diagrams ● Formatting: The team agreed that any diagrams included in the final Word document must be legible. Diagrams that are too wide (horizontal) are problematic , while vertical layouts are acceptable. ● Visual Style: Minjun Kim's (Attendee 4) diagrams, generated with Mermaid, were identified as the cleanest and most professional-looking. The team decided to adopt this as the standard. ● Content (What to exclude): ○ Onejune Lee (Attendee 1) reiterated the professor's feedback that they should not include *every single* class in the diagrams. The professor wants them to use their judgment ("al-jal-ttak"). ○ The team agreed to remove enum (enumerator) types from the diagrams. ○ A major issue identified was that the AI tools (GPT, Gemini, etc.) were "hallucinating" and adding extra, unnecessary classes that are not in the team's final list of 37 classes. ○ Examples of these "hallucinated" classes included Audit Log and Storage Manager, which were part of an older version.

	○ After debate, Onejune Lee (Attendee 1) decided to definitively remove Audit Log, reasoning that it's a "service completeness" feature, not a core function, which the professor does not prioritize. ● Content (How to draw): ○ Jehyung Kim (Attendee 3) pointed out an inefficiency: drawing separate diagrams for Notification and Notification Manager is redundant. ○ The team agreed with his suggestion to only draw the diagram for Notification Manager, as it naturally includes Notification anyway, making the diagram more consolidated and logical. Progress and Remaining Tasks ● Completed Work: Onejune Lee (Attendee 1) reported that the main Class Definitions (the 37 classes), the CRC Cards, and the Metric Calculations are all complete. ● State Diagrams: The team confirmed that State Diagrams are not needed for all 37 classes. They only need to be drawn for classes that have changeable states, such as SafetyZone (Active, Inactive) or ConfigurationManager. ● Author/Responsibility Section: The professor previously criticized the "who did what" section for being too vague. The team agreed that for the final document, they must assign each major section to a single, responsible individual instead of just "everyone". 3. Decisions Made 1. All class diagrams will be standardized using the Mermaid tool and style. 2. All diagrams will be re-generated, strictly adhering *only* to the final list of 37 classes and excluding AI-hallucinated classes like Audit Log. 3. The team will create a new "Author" section that clearly assigns individual responsibility for each major component. 4. The team will meet on Thursday to assemble the final document.

	4. Action Items ● Yunje Lee (Attendee 2) & Jehyung Kim (Attendee 3): ○ Create all required **State Diagrams**. ● Minjun Kim (Attendee 4): ○ Create the **Architecture Diagram**. ● Onejune Lee (Attendee 1): ○ Create the high-level **Class Overview Diagram** (showing only class names). ○ Write the "**Assumptions**" section. ○ Compile the final document and handle all other remaining documentation tasks.
AI Note	https://clovanote.naver.com/s/YReVHf7ZnEXYoT7qAVFaM8S 비밀번호: wk93zs

Appendix A. Glossary

This glossary provides definitions for key terms, classes, and concepts used within the SafeHome Software Design Specification. All terms and their definitions are derived exclusively from the provided SDS and CRC Card documentation to ensure consistency and accuracy.

1. Architectural & Project Concepts

- 3-Layer Architecture
    - The architectural pattern for the SafeHome system, consisting of the Presentation Layer, Business Logic Layer, and Data Layer to ensure a clear separation of concerns.
- Business Logic Layer
    - The middle layer of the architecture, responsible for implementing core business rules and logic. It contains all service classes (e.g., `AlarmManager`, `SecurityModeManager`).
- Data Layer
    - The foundational layer of the architecture, responsible for managing the system's data through domain model entities (e.g., `User`, `Device`, `Alarm`).
- MVC Pattern (Model-View-Controller)
    - A design pattern applied within the Presentation Layer to separate the user interface (View), data (Model), and control logic (Controller).
- MVP (Minimum Viable Product)
    - The initial version of the SafeHome system, developed within a 2-week timeline. The design scope was strategically refined to focus on core functionality.
- Presentation Layer
    - The top-most layer of the architecture, responsible for the user interface and user interaction. It is composed of `ViewController` classes.
- SDS (Software Design Specification)

    - This document; a comprehensive description of the system's design that bridges requirement analysis (SRS) and implementation.
- SRS (Software Requirements Specification)
    - The source document containing the functional requirements and use cases from which the initial classes and system scope were derived.

2. Design Metrics & Key Mechanisms

- CF (Coupling Factor)
    - A MOOD metric measuring the degree of coupling between classes. The design achieved a CF of 0.047, indicating a highly modular and maintainable system.
- CRC Cards (Class-Responsibility-Collaborator)
    - A design tool used to define and validate the system's classes by outlining their core Responsibilities and the other classes they Collaborate with.
- DIT (Depth of Inheritance Tree)
    - A CK metric measuring inheritance depth. The design maintains a shallow hierarchy with an average DIT of 0.7.
- LCOM (Lack of Cohesion in Methods)
    - A CK metric measuring how well methods within a class are related. The design achieved an average LCOM of 0.20, indicating high cohesion.
- PTZ (Pan-Tilt-Zoom)
    - Functionality allowing a camera to be remotely moved. Managed by `PTZControlService` and the `PTZControl` entity.
- RBAC (Role-Based Access Control)
    - An access control model implemented by the `UserPermissionManager` to manage user permissions based on assigned roles (e.g., Admin, User, Guest).

3. Class Definitions

(Classes are organized by their architectural layer and listed alphabetically.)

Frontend Application Classes

- SafeHomeHub
    - The tablet application that hosts all ViewControllers and provides the main UI for local system control, featuring offline mode support.
- SafeHomeMobile
    - The mobile smartphone application providing remote access to the system via a 4-tab navigation structure and push notifications.

Presentation Layer (ViewController) Classes

- CameraViewController
    - Manages the camera viewing and control UI, including live video streaming, PTZ controls, two-way audio, and access to recordings.
- DashboardViewController
    - The main dashboard screen controller that displays system status, security mode, recent events, and allows for quick mode changes.
- DeviceManagementViewController
    - Manages the UI for registering, viewing, updating, and removing devices from the system.
- EmergencyViewController
    - Manages the emergency response UI, providing panic button functionality, displaying active alarms, and handling alarm verification.
- NotificationPanel
    - A real-time display component that shows notifications, tracks unread counts, and allows users to manage notification settings.
- RecordingViewController
    - Manages the UI for searching, playing back, exporting, and sharing video recordings.
- SecurityZoneViewController
    - Manages the UI for security zone configuration, allowing users to create, assign devices to, and manage zones.
- UserAccountViewController

    - Manages the UI for viewing and editing user profiles, changing passwords, managing permissions, and handling logout.

Backend Core System

- System
    - The central backend system that initializes and manages all service components, provides a unified RESTful API, and coordinates inter-service communication.

Business Logic Layer (Service) Classes

- AlarmManager
    - Manages the alarm lifecycle, including triggering, verification, escalation, and resolution. Integrates with emergency services.
- DeviceConfigService
    - Manages device configuration updates, validation, and history tracking, supporting bulk updates and resets.
- DeviceHealthMonitor
    - Monitors device health status, including battery levels, signal strength, and heartbeat tracking, to generate health alerts.
- DeviceRegistry
    - Acts as the central repository for devices, managing discovery, registration, removal, and providing search/filtering capabilities.
- LoginManager
    - Manages user authentication, session lifecycle, failed login tracking, account locking, and password resets.
- NotificationManager
    - Manages multi-channel notification delivery (push, SMS, email), cooldown policies, and user preferences.
- PTZControlService
    - Manages PTZ camera control with a mutex locking mechanism to prevent conflicts, handling commands and preset positions.
- RecordingManager

    - Manages the video recording lifecycle, including starting, stopping, searching, exporting, and sharing recordings.
- SecurityModeManager
    - Manages security modes (Away, Home, etc.), safety zones, entry/exit delays, and sensor bypass functionality.
- StreamingService
    - Manages live video streaming from cameras, including stream initialization, URL generation, and quality adjustment.
- UserPermissionManager
    - Implements role-based access control (RBAC), manages user permissions, validates actions, and maintains audit logs of user activities.

Data Layer (Entity) Classes

- Alarm
    - An entity managing alarm state, verification status, timestamps, and resolution tracking.
- Camera
    - An entity extending `Device` that manages streaming, recording, PTZ control, two-way audio, and password protection.
- Device
    - An abstract base class for all physical devices, providing common functionality for status, battery level, signal strength, and heartbeat tracking.
- Notification
    - An entity storing message details, type, priority, read status, and timestamps for tracking notification delivery.
- PTZControl
    - An entity managing pan, tilt, and zoom functionality with a mutex lock to prevent control conflicts between users.
- Recording

    - An entity storing video metadata, including camera source, timestamps, duration, trigger type, and file information.
- SafetyZone
    - An entity managing zone configuration, including name, sensor membership, armed status, and entry/exit delay settings.
- Sensor
    - A base sensor class extending `Device` that handles sensor triggering, sensitivity settings, cooldown periods, and bypass functionality.
- SensorEvent
    - An entity recording individual sensor triggers with event type, timestamp, sensor value, and additional metadata.
- Session
    - An entity tracking user login state, device information, IP address, timestamps, and session validity.
- User
    - An entity storing user profile information, credentials, role, permissions, and active sessions.