[CS350] team 9 SDS

CS350SafehomeProject

SoftwareDesignSpecification(SDS)

20220541	Jien Lee
20240410	Jonghwa An
20240685	Minseok Jo

Korea Advanced Institute of Science and Technology

i. Overview

1. Introduction

This document details the concrete Design Model of the SafeHome System, which is intended to satisfy the Software Requirements Specification (SRS) defined in the preceding report. As the design phase directly impacts the subsequent implementation and critical Security Review Phases, we focus on presenting a stable and well-formed design that meets SafeHome's core non-functional requirements, such as End-to-End Encryption, Real-Time Notifications, and Offline Operation. To clearly illustrate the system's design, this document defines the Distributed 3-Tier Architectural Structure and includes Class Diagrams to show the relationships and attributes of core objects. Furthermore, it incorporates State Diagrams to clarify the transition logic for system modes and device states, and Sequence Diagrams to concretely portray the interaction flows for key use cases, including user authentication, incident generation, and remote streaming.

2. Goal

a. Ensure Complete Requirements Traceability and a Robust Security Architecture: Explicitly define the implementation model for all functional and non-functional requirements, with a critical focus on the End-to-End Encryption (E2EE) and authentication mechanisms.

b. Achieve High Cohesion and Decoupling between Local and Cloud Components: Design the local Hub as a highly cohesive module capable of independent operation during offline mode, ensuring low data coupling with cloud services to maintain system reliability.

c. Maximize Real-Time Reliability and Operational Efficiency: Minimize latency for real-time incident notifications and streaming. Guarantee system reliability through local fail-safe design in the Hub, and ensure data integrity during storage and transmission.

d. Minimize Complexity and Promote Extensibility: Design components with low cyclomatic complexity to improve testability. Utilize reusable design patterns (e.g., Adapter, Observer) to allow for flexible integration of new devices and protocols in the future.

3. How The Design Work Proceed

The design process commenced with a systematic analysis of the Software Requirements Specification (SRS) to identify and elaborate the major use cases and core system functionalities across all domains, including Intelligent Security, Live Surveillance, and Remote Access. Following this requirements analysis, an initial set of design artifacts, including the preliminary class diagrams and CRC (Class-Responsibility-Collaborator) cards, was created adhering to the provided design format and the mandated MVC architectural template. These initial design decisions were directly based on and traceable to the functional requirements within the SRS. Subsequently, a preliminary design evaluation was conducted utilizing established design metrics (Architectural, CK, and MOOD). This quantitative review identified several key structural limitations; for example, the initial model exhibited high coupling (a high CBO metric).. Furthermore, core controllers like the SecurityModeController were found to have moderate functional cohesion (low LCOM / high V(G)) due to complex, mixed responsibilities like sensor validation. Based on these findings, the architecture was systematically refined by applying the Interface Segregation Principle to abstract the StorageManager behind granular interfaces (e.g., IUserRepository, ISettingsRepository) and by extracting complex validation logic into a new, highly cohesive ArmingValidator service class. This refinement was applied to enhance the overall quality, maintainability, and testability of the system, resulting in the more robust final architecture detailed herein.

4. Assumption

▼ Assumptions

1. Architectural Assumptions

   a. Layered Architecture with Domain-Driven Design

      The system adopts a hybrid layered architecture consisting of:

      - Presentation Layer (View classes): UI components for data display and user input.
      - Application Layer (Controller classes): Coordinates use case workflows and delegates to domain services.
      - Domain Layer (Model classes): Encapsulates business logic and domain rules.
      - Infrastructure Layer: Handles persistence, networking, and device protocols.

   b. Hub-Cloud Distributed Architecture

      The system uses a distributed architecture with:

      - SafeHome Hub (Edge Device): Manages local device connections (Zigbee, Wi-Fi), executes real-time operations, maintains local cache, and synchronizes with CloudServer.
      - CloudServer (Central Service): Manages user authentication, session management, persistent data storage, multi-client synchronization, and remote access.

Internet connectivity is required for authentication, multi-device synchronization, remote notifications, and cloud storage.

2. Persistence Assumptions
    a. Dual Storage Strategy

        The system employs dual storage:

        - Local Storage (Hub): Stores recent sensor data, device states, and automation rules for offline operation with limited retention (e.g., 7 days).
        - Cloud Storage (CloudServer + Database): Stores all persistent entities with unlimited retention based on policies, accessible via CloudServer API.

        Database access is abstracted by a Database component accessed only by CloudServer.

3. Authentication & Session Assumptions
    a. Hybrid Token-Based Authentication
        - Access tokens (short-lived, 15 minutes) for API authentication.
        - Refresh tokens (long-lived, 30 days) for obtaining new access tokens.
        - JWT tokens include accountId, userId, role.
        - Session entity stored in CloudServer supports multi-device session management and device info tracking.
    b. Account Lockout Policy
        - Lockout triggered after 3 consecutive failed logins within 15 minutes.
        - Lockout duration: 15 minutes automatic unlock.
        - Lockout counter resets upon successful login, password reset, or timer expiration.
4. Device Management Assumptions
    a. Device Abstraction Hierarchy

        An abstract Device base class with subclasses:

        - Sensor (e.g., AirQualitySensor, PowerMeter)
        - Camera
        - SmartDevice (e.g., SmartLight, VentilationSystem)

        Devices communicate via Zigbee, Wi-Fi, or Z-Wave.

    b. Device Registration and Health Monitoring

        Devices must register with Hub, report status periodically, and undergo health checks.

        Unresponsive devices are marked offline; failure events are logged and users notified.

5. Security Domain Assumptions
    a. Security Mode State Machine

        Three primary security modes with predefined configurations: Disarmed, Armed (Away), Armed (Stay).

        SecurityMode.modeName stored as string for extensibility.

    b. Sensor Configuration and Zone Logic

        Each sensor assigned to exactly one SecurityMode configuration.

        Simplified zone model without overlapping or conflicting zones in initial release.

    c. Arming Process and Sensor Bypass

        Arming is synchronous and can be blocked by faulted sensors.

        Users can bypass faulty sensors temporarily for the current session only.

d. Incident Management

Incidents created on sensor triggers; associated recordings attached automatically.

Alerts sent via NotificationService; incidents resolved manually or auto-resolved after timeout.

6. Surveillance Domain Assumptions
   a. Video Streaming and Recording

   Supports live streaming without storage and recording with configurable retention (default 30 days).

   Recording metadata stored in Database; actual video stored in cloud object storage.

   b. Camera Control

   Supports PTZ if hardware allows, snapshot capture, and stream quality adjustment.
7. Monitoring & Automation Assumptions
   a. Monitor Classes as Domain Services

   Specialized monitors for AirQuality and PowerMeter encapsulate monitoring logic and decouple automation.

   b. Automation Rule Execution

   Automation rules created by users, stored in CloudServer, executed locally by Hub, and synchronized with CloudServer.

   c. Threshold-Based Automation

   Supports simple threshold automations alongside full rule-based automations.
8. Notification Assumptions
   a. Notification Channels

   Supports push notifications, email, and SMS with user-configurable preferences.
9. Data Retention Assumptions

   Retention periods vary by data type:

   - Raw sensor readings: 7 days in Hub local cache
   - Aggregated sensor data: 1 year in CloudServer database
   - Video recordings: 30 days in object storage
   - Incidents, audit logs: 1 year or indefinite depending on type
   - User sessions: until logout or 30 days

   Optional extended retention available.
10. Offline Operation Assumptions

    Hub operates autonomously during internet outages supporting device control, sensor monitoring, and automation.

    Limited or no remote access, multi-device sync, notifications, or cloud storage during outage.
11. Scalability Assumptions

    System designed to support multiple Hubs per Account in the future; initial release supports one Hub per Account.
12. Security & Privacy Assumptions

    Data encrypted at rest (AES-256) and in transit (TLS 1.3).

    User data owned by account holder with GDPR-compliant export and deletion capabilities.

    Video and sensor data never shared without user consent.
13. Error Handling & Resilience Assumptions

    Hub buffers failed operations and retries with exponential backoff.

    Hub marks failed devices offline and notifies users.

System requires explicit user action on device failures.

14. Testing & Validation Assumptions

    Domain classes designed for unit testing with dependency injection and mockable external dependencies.

15. Performance Assumptions

    Latency targets defined for device control, sensor alerts, video streaming, and login under typical network conditions.

    Performance degrades gracefully under load.

16. Assumptions Summary Checklist

    - Architecture, Persistence, Authentication, Device Management, Security, Surveillance, Monitoring, Automation, Notifications, Data Retention, Offline Operation, Scalability, Security & Privacy, Error Handling, Testing, Performance assumptions included and aligned with Class Diagram.

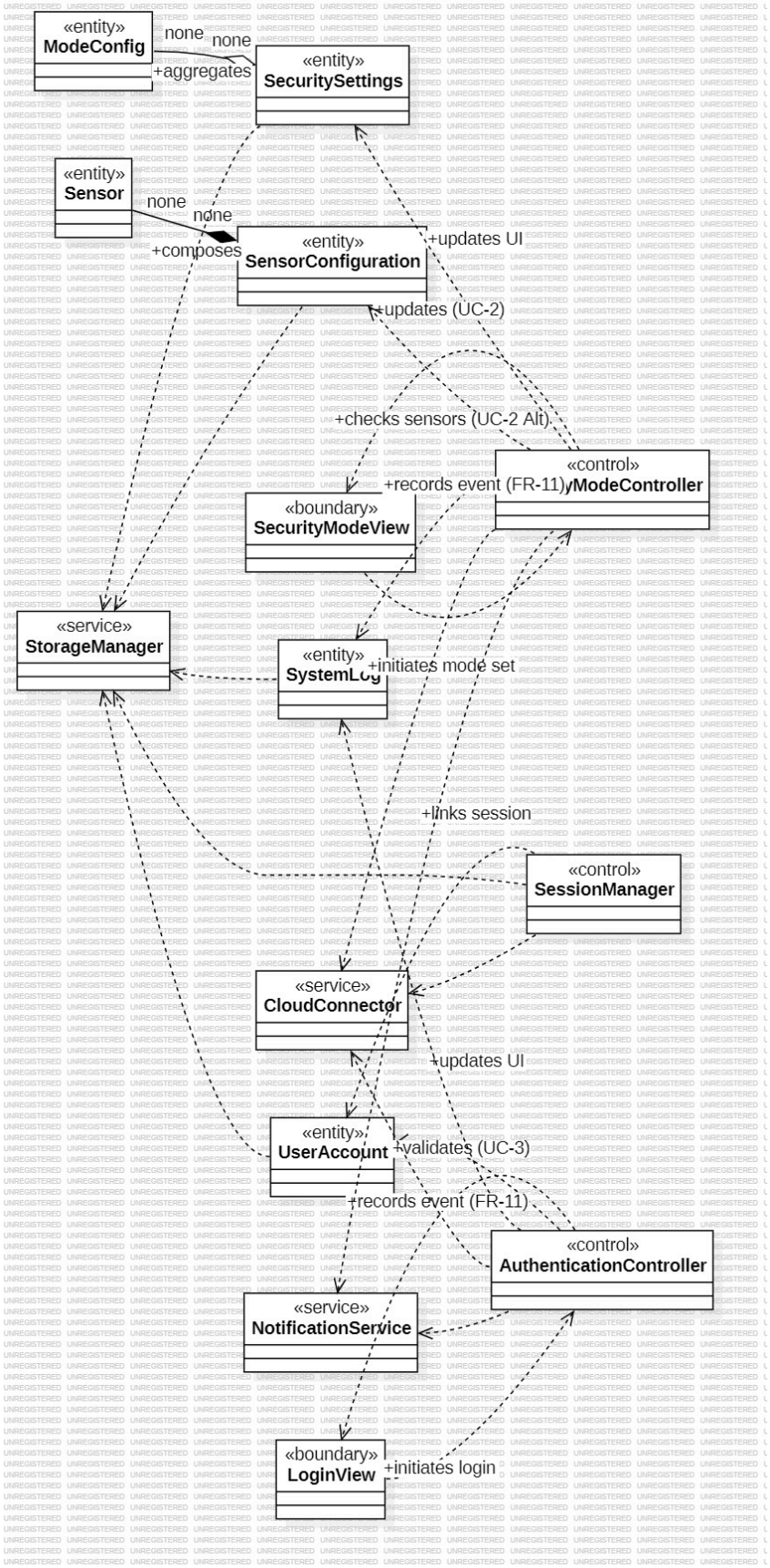
17. Open Questions for Stakeholder Review

    - Video retention duration preferences
    - Multi-Hub support at initial release
    - Acceptability of simplified zone model
    - Offline recording buffer policy
    - Notification throttling policies

ii. Architectural Structure

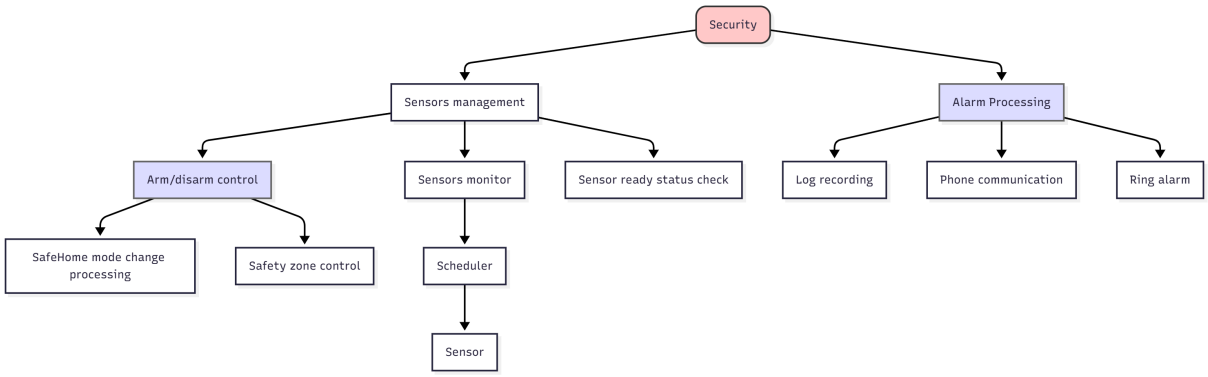
1. Overall Architecture

▼ Overall Architecture Diagram


«entity»
ModeConfig
none
none
+aggregates
«entity»
SecuritySettings
«entity»
Sensor
none
none
+composes
«entity»
SensorConfiguration
+updates UI
+updates (UC-2)
+checks sensors (UC-2 Alt)
«control»
SecurityModeController
+records event (FR-11)
«boundary»
SecurityModeView
«service»
StorageManager
«entity»
SystemLog
+initiates mode set
+links session
«control»
SessionManager
«service»
CloudConnector
+updates UI
«entity»
UserAccount
+validates (UC-3)
+records event (FR-11)
«control»
AuthenticationController
«service»
NotificationService
«boundary»
LoginView
+initiates login

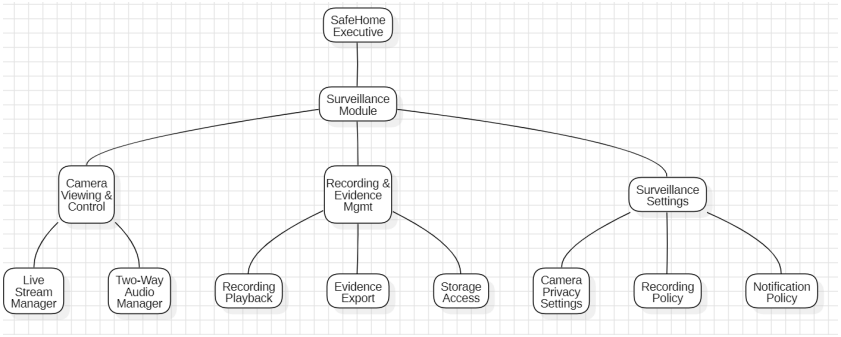
2. Intelligence Security

▼ Intelligent Security Diagram



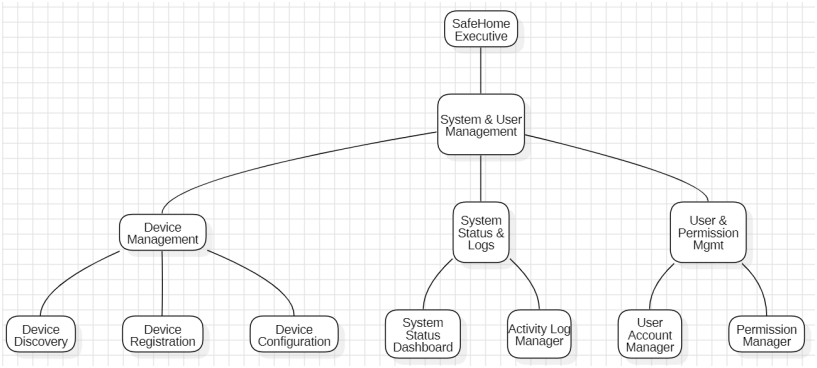
3. Live Surveillance

▼ Live Surveillance Diagram



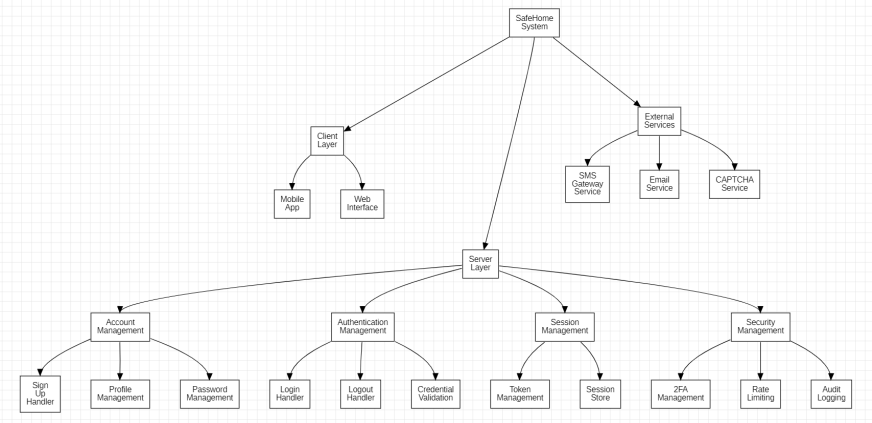
4. System and User Management

▼ System and User Management Diagram



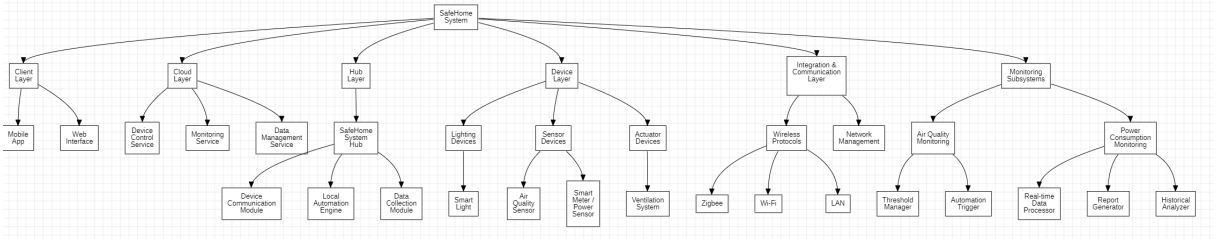
5. Remote access and Account

▼ Remote access and Account Diagram



6. Indoor Monitoring and Device Control

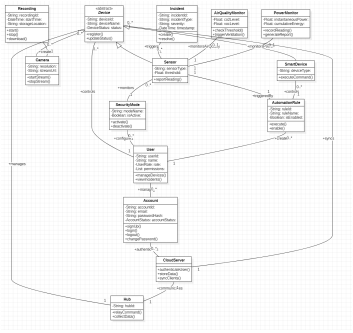
▼ indoor monitoring and device control Diagram



iii. Class Diagram

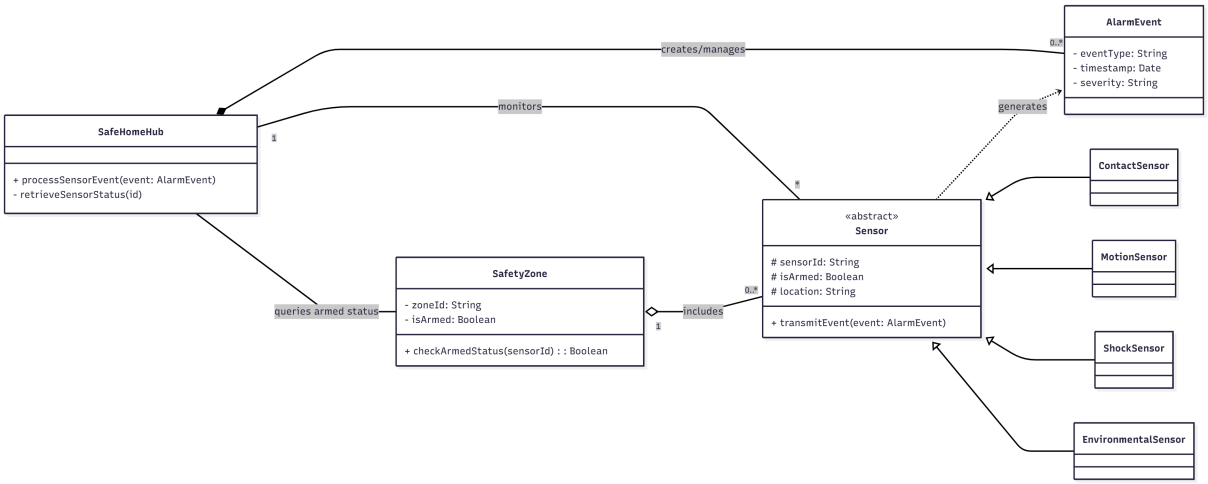
1. Class Diagram - Whole System Overview

▼ overview class diagram

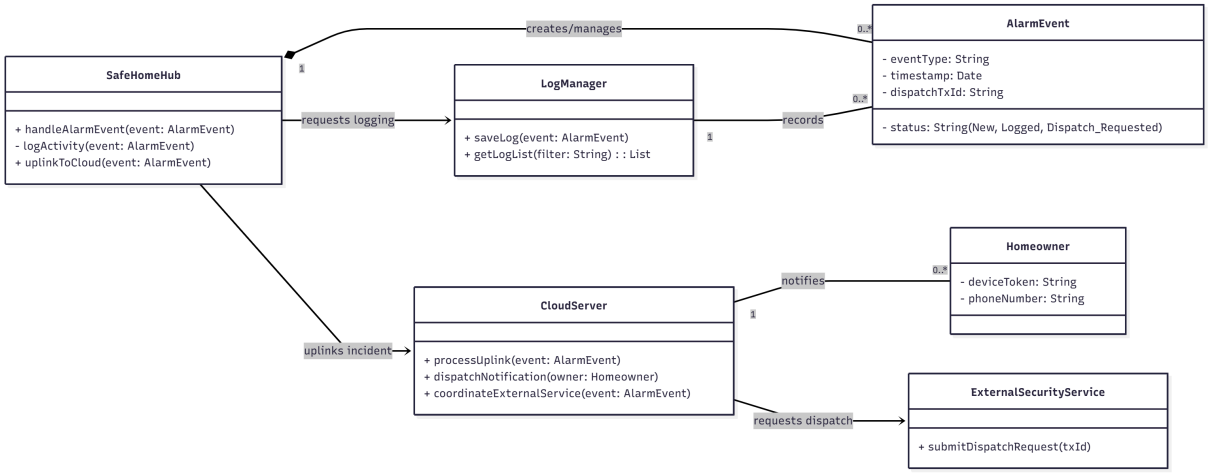


2. Class Diagram - Intelligent Security

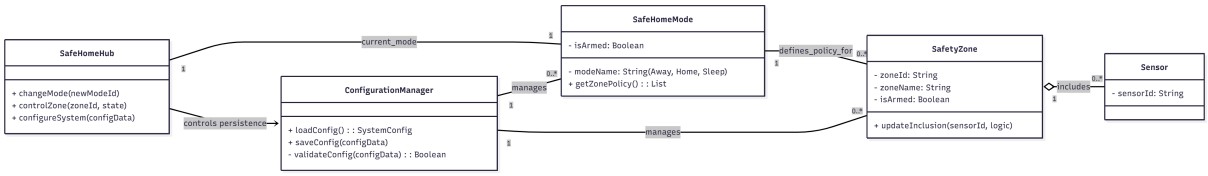
▼ 2.1 Sensor Monitoring



▼ 2.2 Incident Management



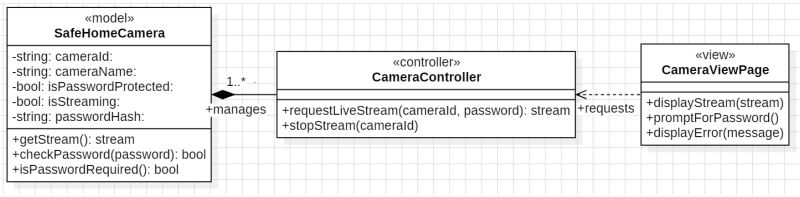
▼ 2.3 Security Mode Control



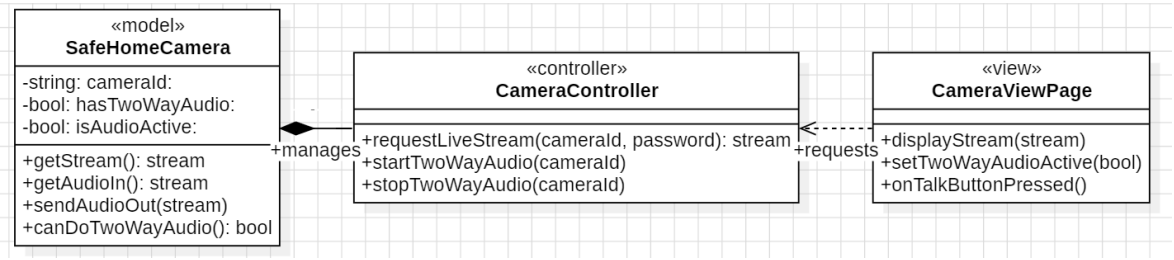
3. Class Diagram - Live Surveillance

▼ 3.1. Camera Viewing and Control

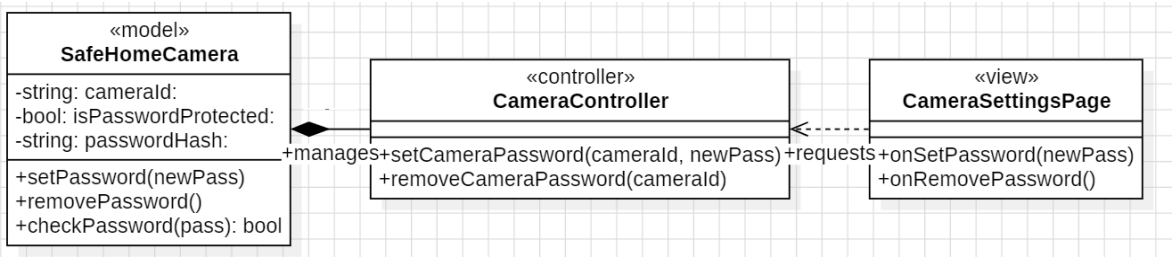
▼ 3.1.1 Single Camera Live View



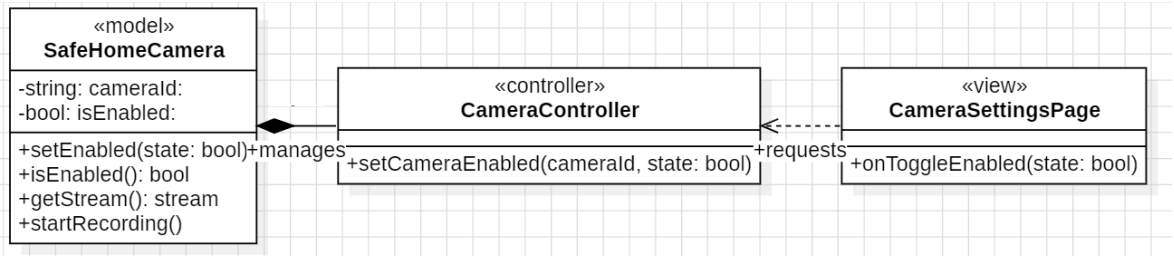
▼ 3.1.2 Two-Way Audio



▼ 3.1.3 Protect Sensitive Camera Feed with a Password

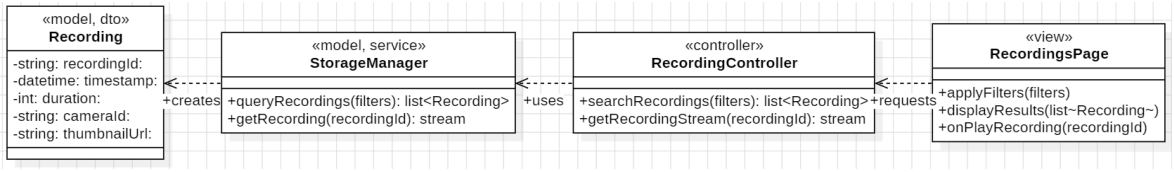


▼ 3.1.4 Camera Activation and Deactivation

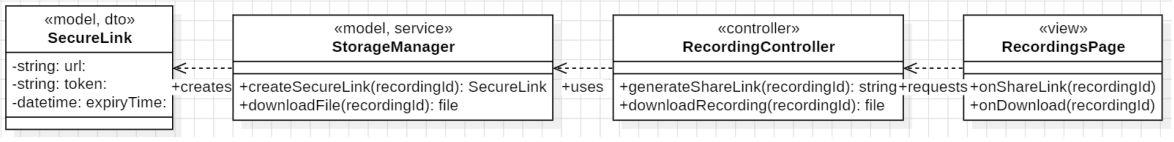


▼ 3.2. Recording and Evidence Management

▼ 3.2.1 Search and Playback Recordings

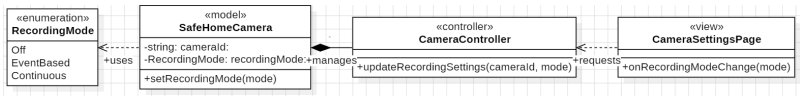


▼ 3.2.2 Evidence Sharing and Export

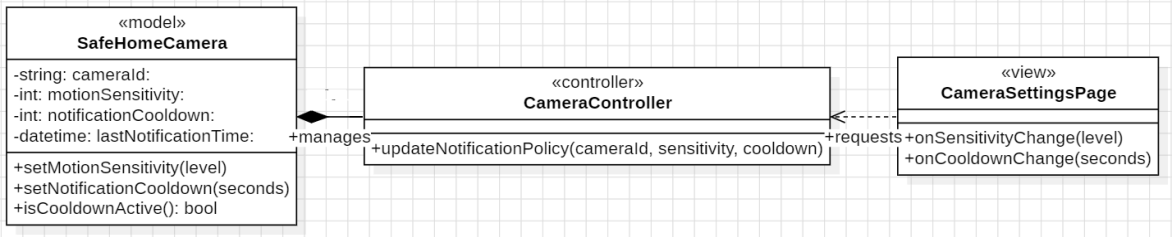


▼ 3.3. Surveillance Settings

▼ 3.3.1 Recording Settings



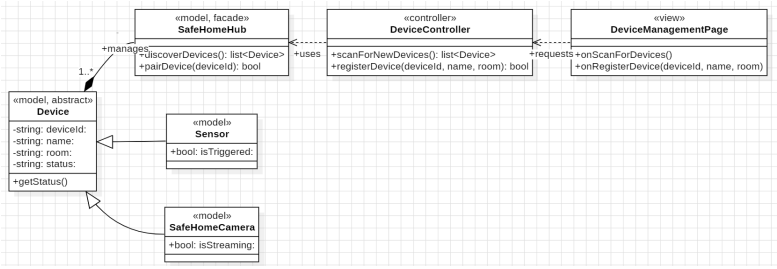
▼ 3.3.2 Notification Policy and Cooldown



4. Class Diagram - System and User Management

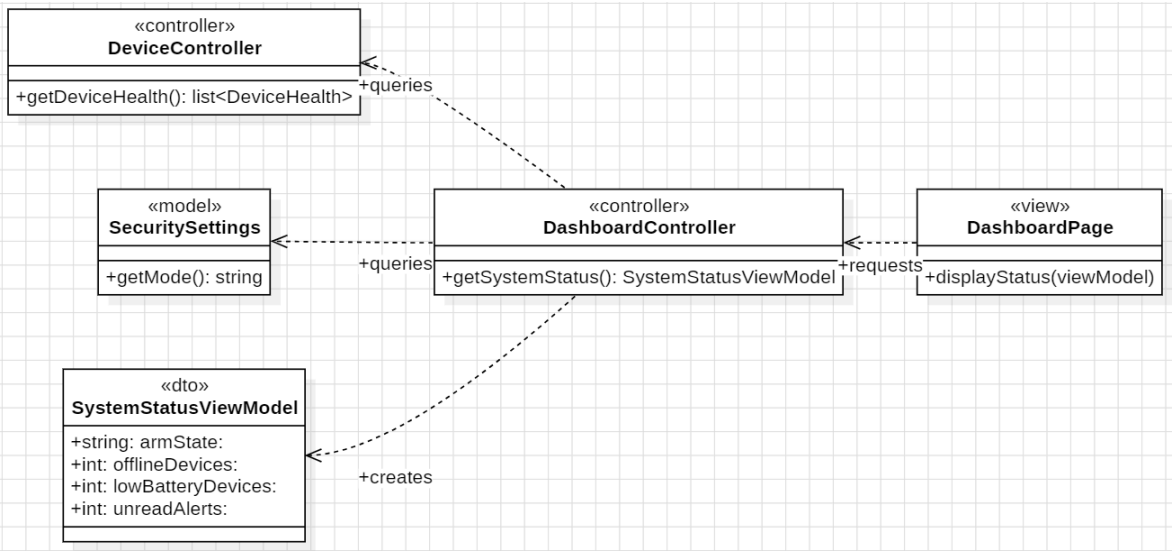
▼ 4.1. Device Management

▼ 4.1.1 Add and Configure New Devices

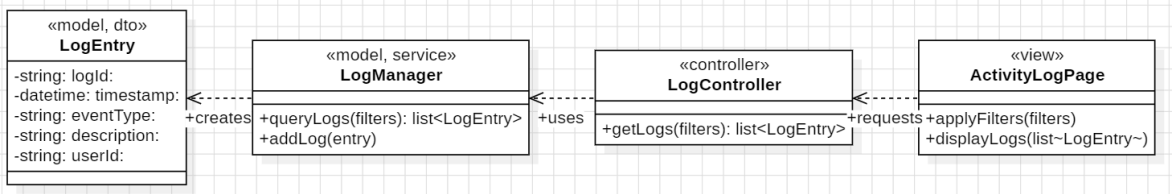


▼ 4.2. System Status and Logs

▼ 4.2.1 System Status Dashboard

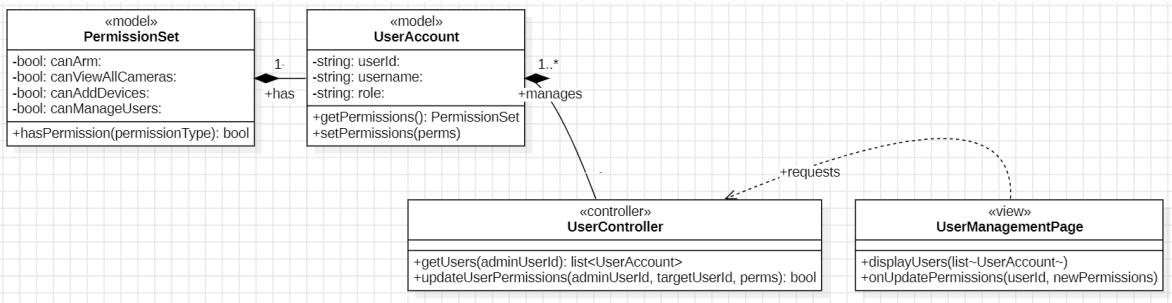


▼ 4.2.2 Activity Logs and Timeline



▼ 4.3. User and Permission Management

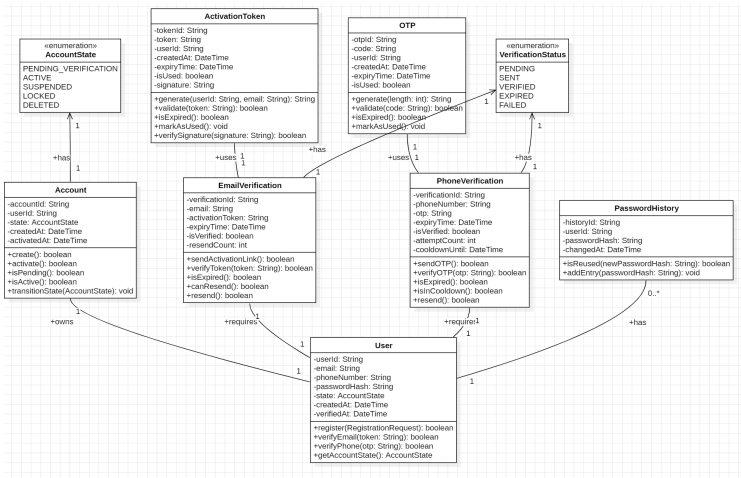
▼ 4.3.1 User Role and Access Control



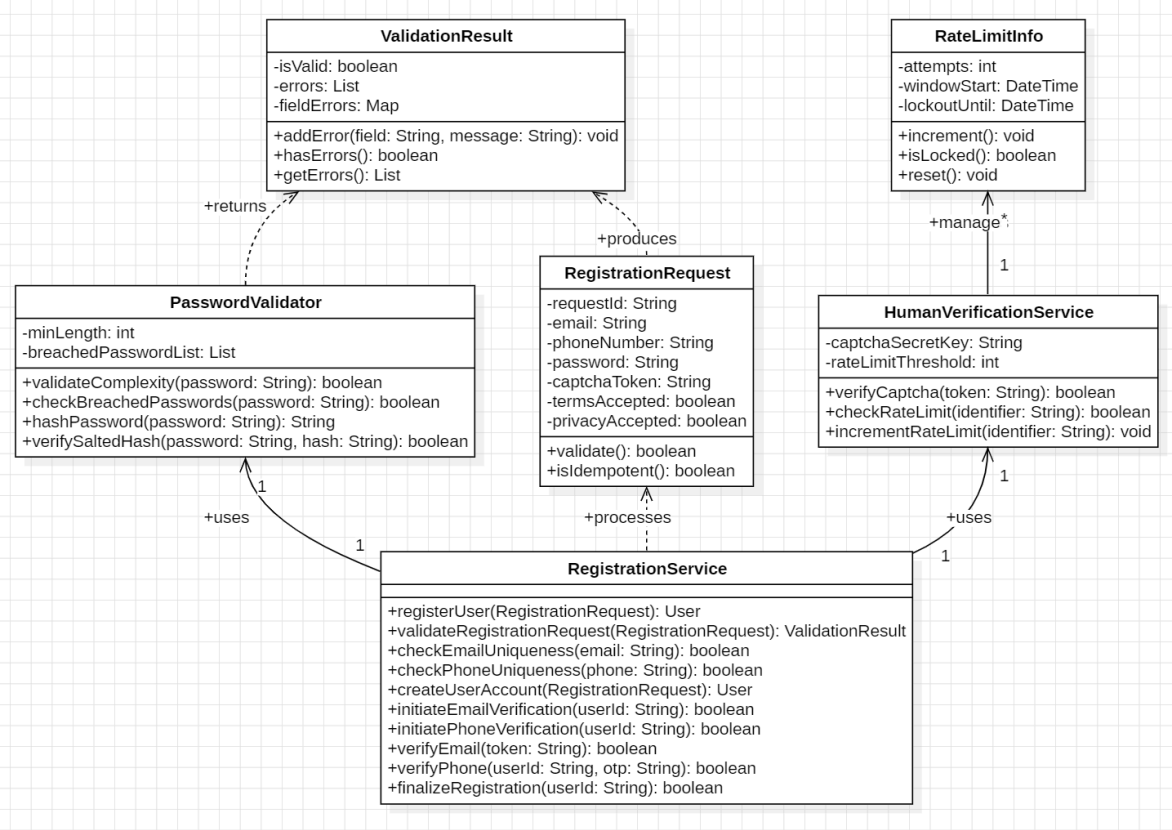
5. Class Diagram - Remote access & account

▼ 5.1 Registration (Sign Up)

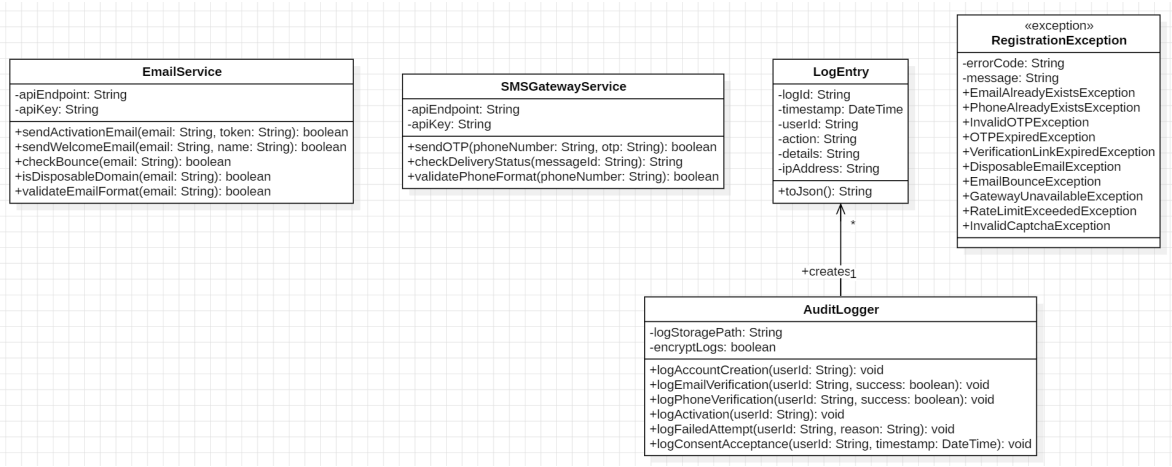
▼ 5.1.1 Domain Layer



▼ 5.1.2 Service Layer

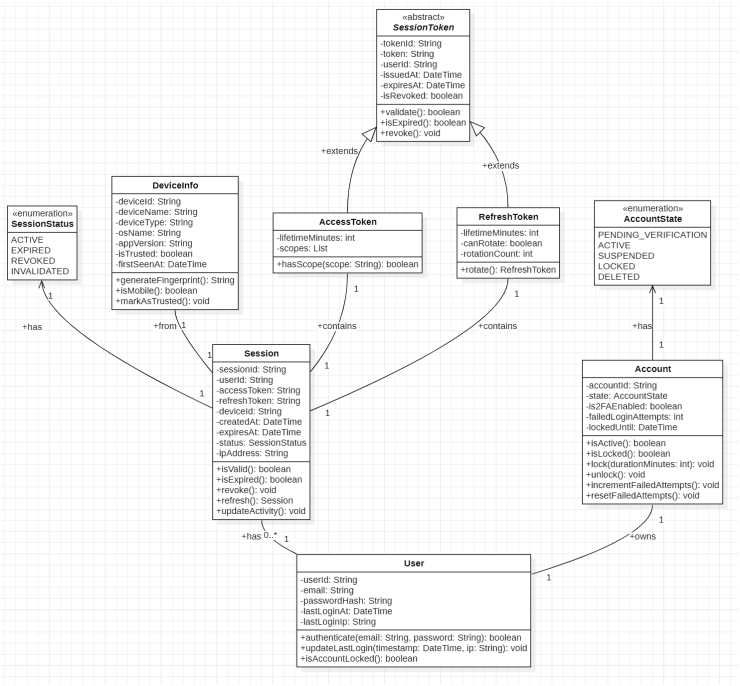


▼ 5.1.3 Infrastructure Layer

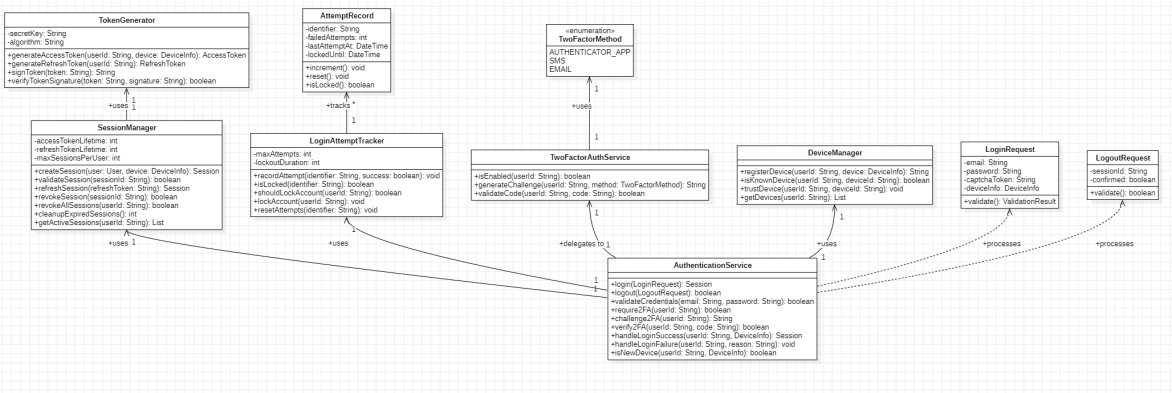


▼ 5.2 Authentication (Log In/Out)

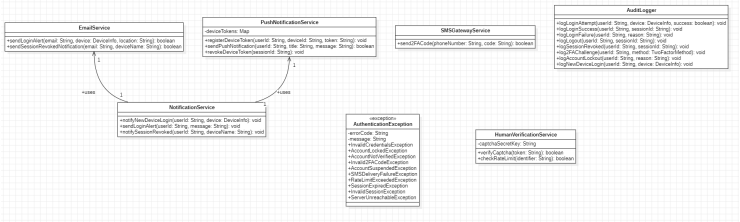
▼ 5.2.1 Domain Layer



▼ 5.2.2 Service Layer

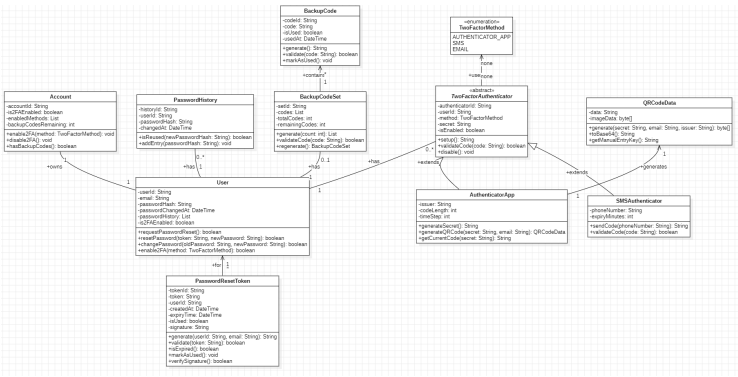


▼ 5.2.3 Infrastructure Layer

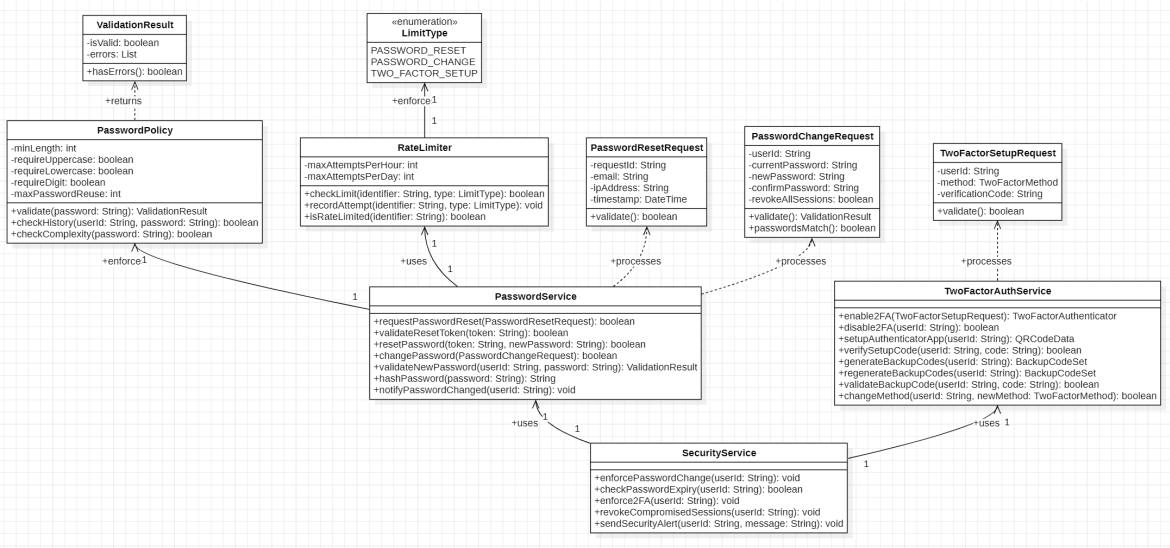


▼ 5.3 Security (Password, 2FA)

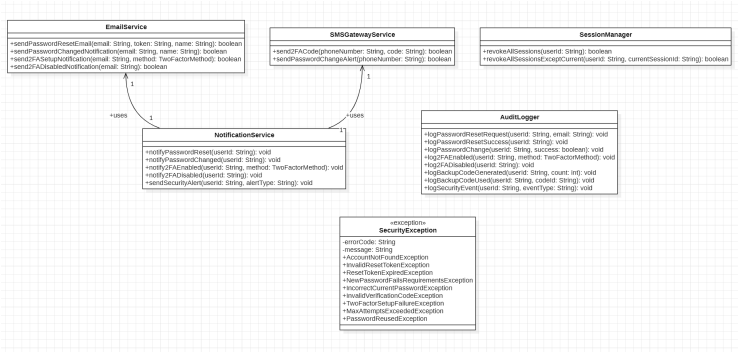
▼ 5.3.1 Domain Layer



▼ 5.3.2 Service Layer

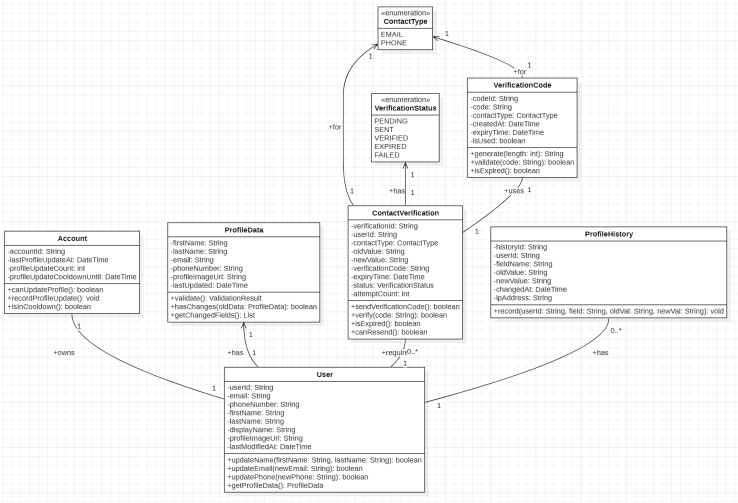


▼ 5.3.3 Infrastructure Layer

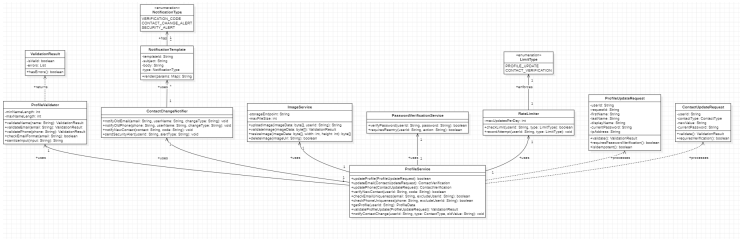


▼ 5.4 Profile Management

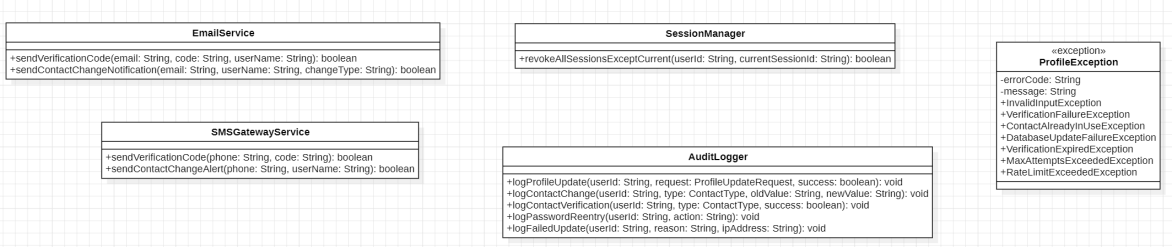
▼ 5.4.1 Domain Layer



▼ 5.4.2 Service Layer



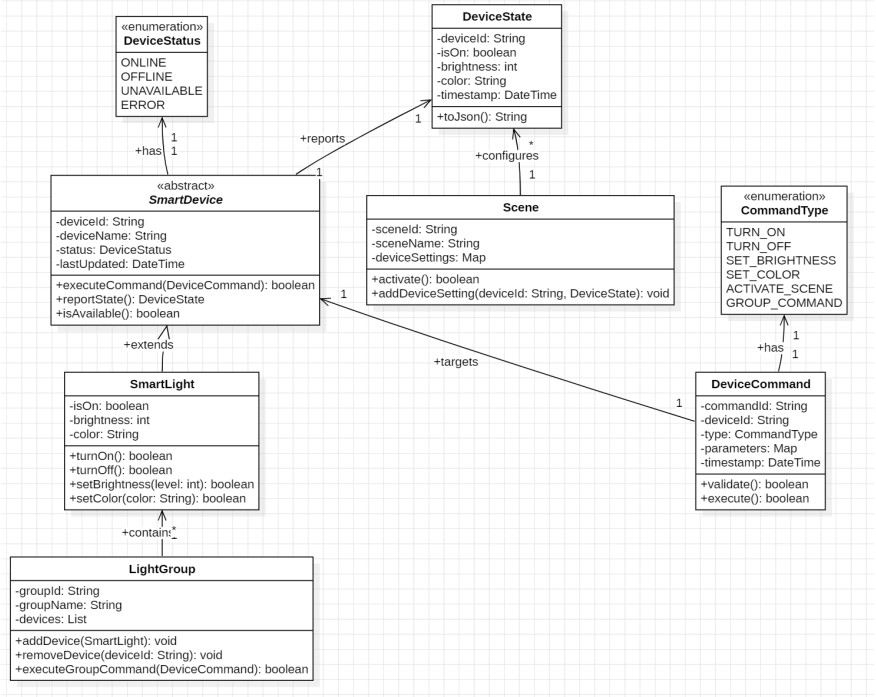
▼ 5.4.3 Infrastructure Layer



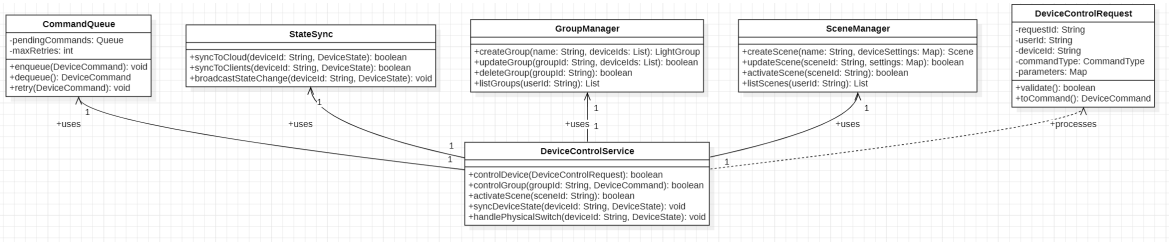
6. Class Diagram - Indoor Monitoring and Device Control

▼ 6.1 Device Control

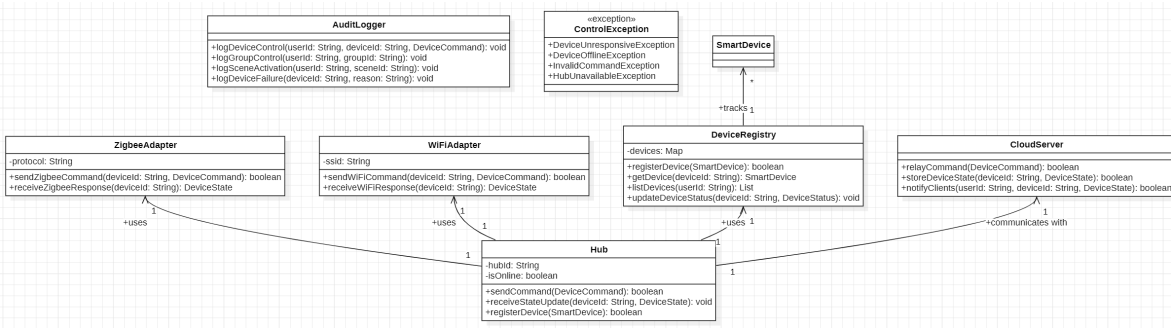
▼ 6.1.1 Device Control - Domain Layer



▼ 6.1.2 Device Control - Service Layer

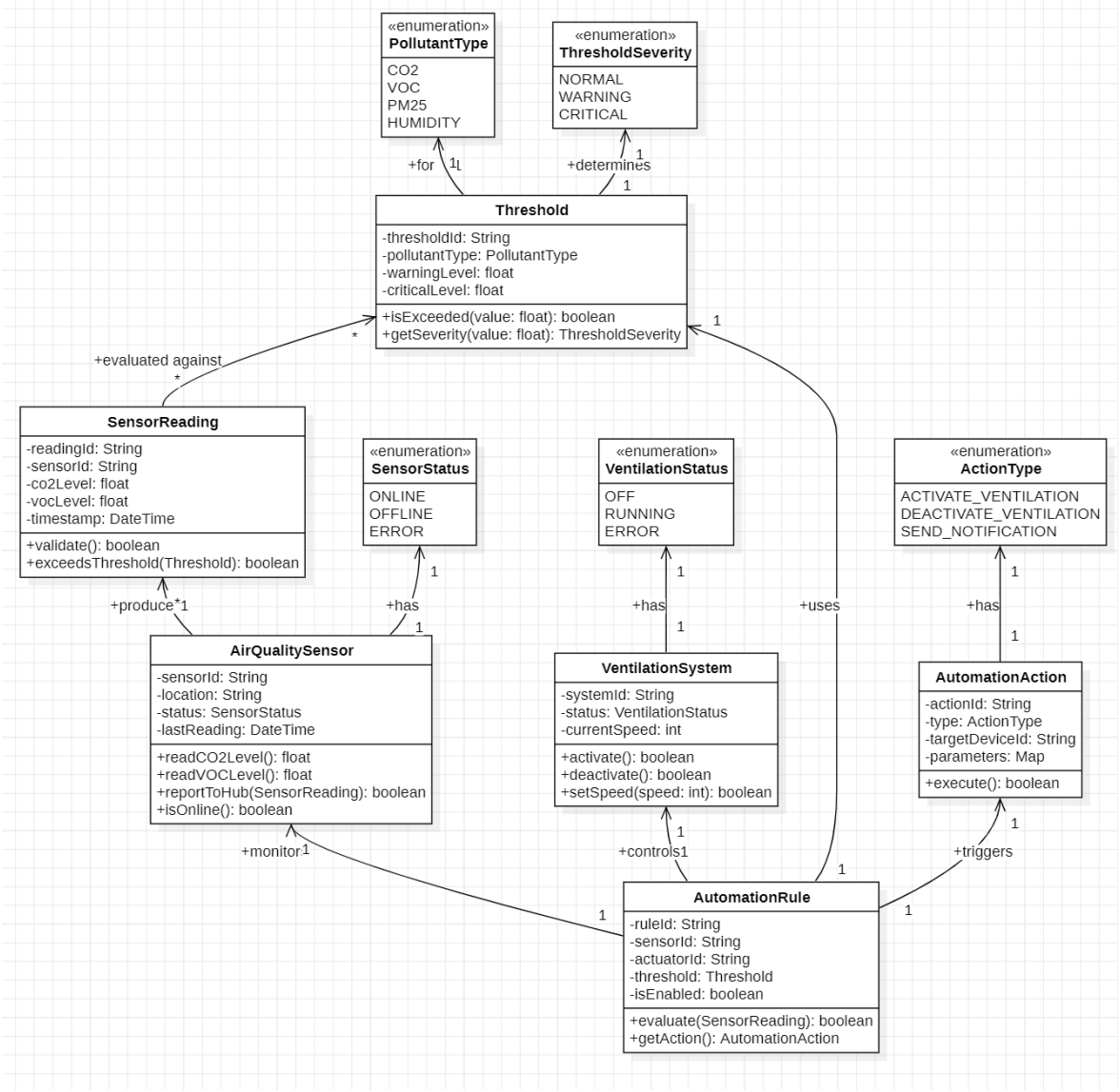


▼ 6.1.3 Device Control - Infrastructure Layer

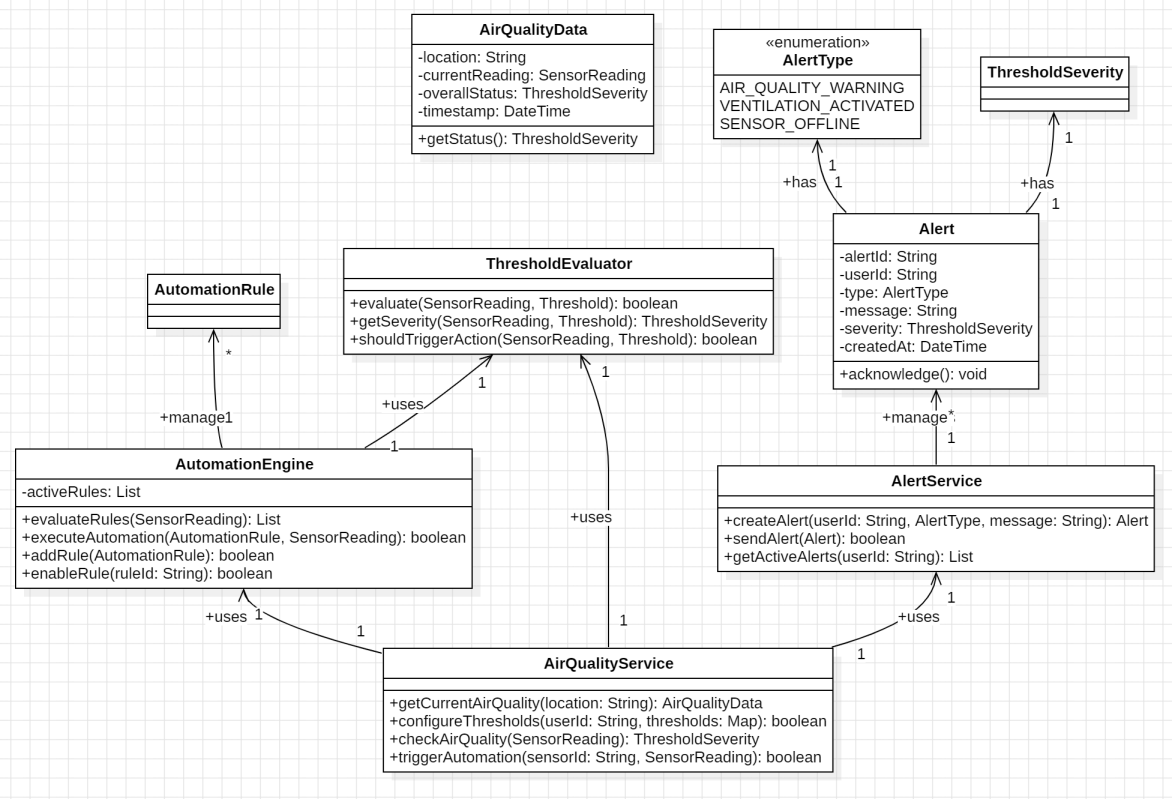


▼ 6.2 Air Quality Monitoring

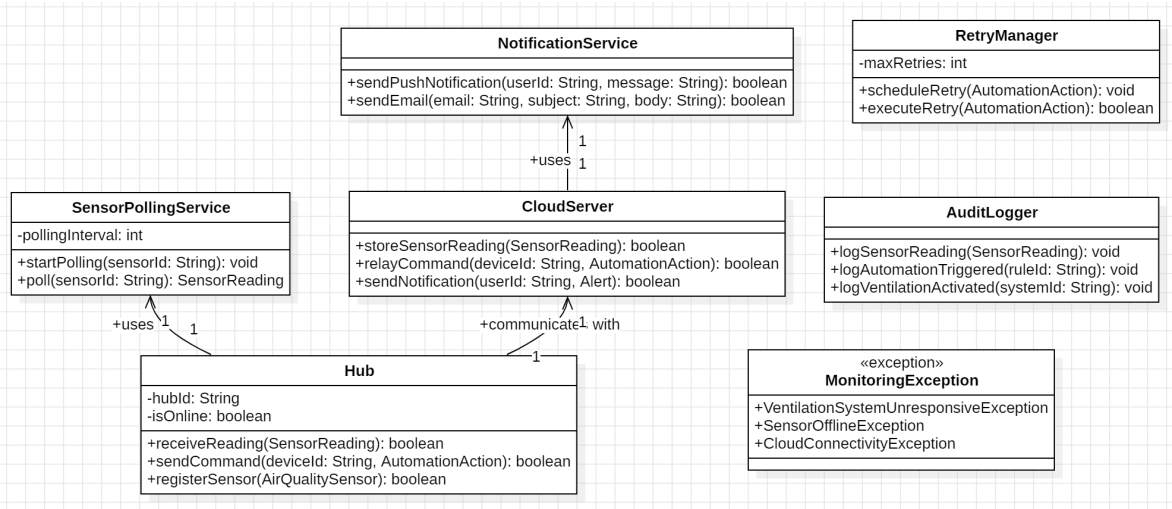
▼ 6.2.1 Air Quality Monitoring - Domain Layer



▼ 6.2.2 Air Quality Monitoring - Service Layer

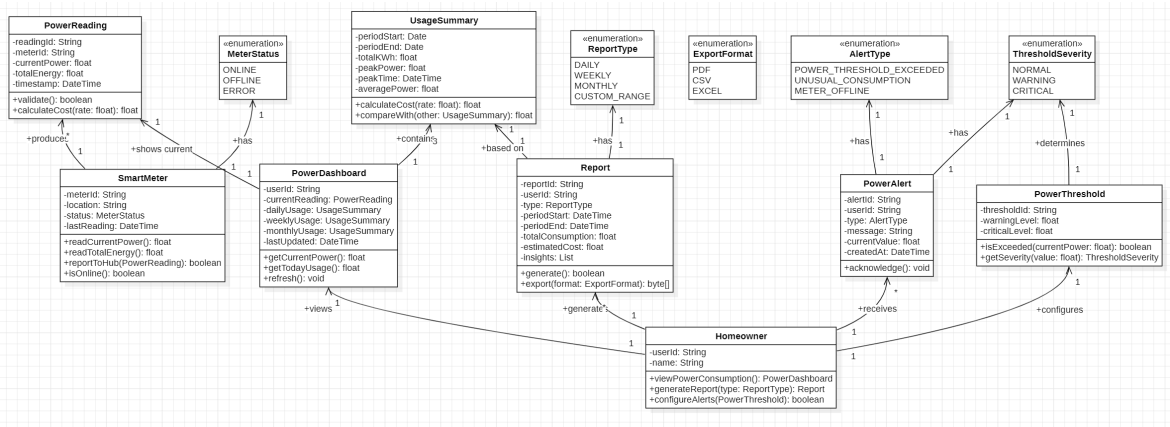


▼ 6.2.3 Air Quality Monitoring - Infrastructure Layer

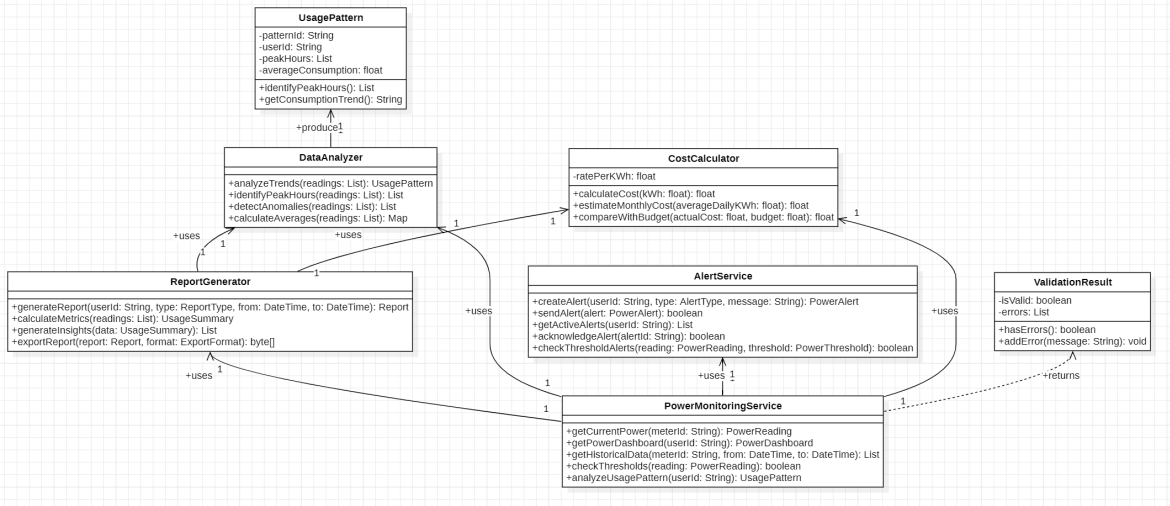


▼ 6.3 Power Monitoring

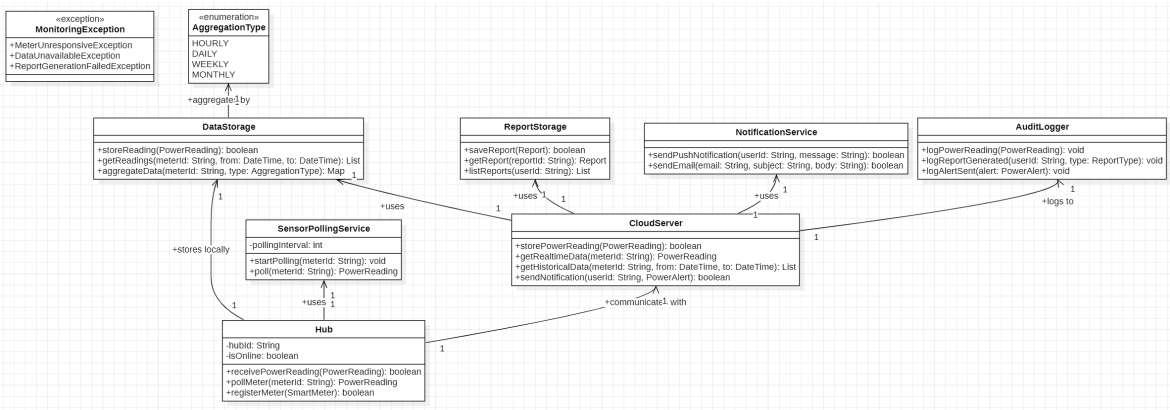
▼ 6.3.1 Power Monitoring - Domain Layer



▼ 6.3.2 Power Monitoring - Service Layer



▼ 6.3.3 Power Monitoring - Infrastructure Layer



iv. CRC Cards

1. CRC Cards - Intelligent Security

▼ CRC Card

Class	Responsibility	Collaborator
SafeHomeHub	Manage the current security state (mode)	**SafeHomeMode**
	Receive, validate, and process all Sensor events (UC 1.1)	**SafetyZone**, **Sensor**
	Create and manage the AlarmEvent instance	**AlarmEvent**
	Initiate local alarm and uplink incident to CloudServer (UC 1.2)	**LogManager**, **CloudServer**
ConfigurationManager	Validate and save all system configuration changes	**SafeHomeMode**, **SafetyZone**
	Load configurations upon system initialization	**SystemStorageManager**
SafeHomeMode	Define sensor activation policy (active/inactive list) per mode	**SafetyZone**
	Provide current armed/disarmed policy to Hub	
SafetyZone	Manage sensor grouping and zone-level armed status	**Sensor**
	Provide armed status check for specific Sensors (UC 1.3)	
AlarmEvent	Store complete incident details (type, time, dispatch status)	
	Provide event data to Hub and CloudServer for processing	
Sensor	Detect environmental input and transmit raw event data (UC 1.1)	**SafeHomeHub**
CloudServer	Dispatch alerts to Homeowner and external services (UC 1.2)	**Homeowner**, **ExternalSecurityService**

2. CRC Cards – Live Surveillance

▼ CRC Cards

Class	Responsibilities	Collaborators
SecurityModeView	• Display current security mode status.• Render mode selection buttons.• Show exit delay countdown timer.• Display sensor fault warnings and bypass options (from ArmingValidator).• Refresh UI when mode changes.	• SecurityModeController (delegates mode changes)• ArmingValidator (requests sensor bypass)• SecuritySettings (displays current mode state)
SecurityModeController	• Orchestrate security mode change workflow. • Receive mode change request from SecurityModeView. • Request sensor/bypass validation from ArmingValidator. • If validation passes, update state in SecuritySettings. • Start and manage exit delay timer. • Notify SecurityModeView to refresh UI. • Log all mode change events via ILogRepository. • Synchronize mode state via CloudConnector.	• SecurityModeView • ArmingValidator • SecuritySettings • ILogRepository • CloudConnector • NotificationService
SecuritySettings	• Store current security mode (Away, Home, Sleep, etc.) • Maintain configuration for each security mode • Track which sensors are bypassed • Store exit delay and entry delay durations • Provide mode configuration to controllers • Persist mode changes to database • Enforce mode-specific sensor activation rules	• SecurityModeController (provides mode configurations) • ModeConfig (contains sensor lists for each mode) • StorageManager (persists security settings data)
ModeConfig	• Define the rules for a single security mode (e.g., "Away"). • List which sensors should be active in this mode.	• SecuritySettings

Class	Responsibilities	Collaborators
SensorConfiguration	• Maintain the complete list of all registered Sensor objects. • Map sensors to safety zones. • Provide the current status of any sensor. • Persist configuration via ISensorConfigRepository.	• ArmingValidator • Sensor • ISensorConfigRepository
Sensor	• Represent the state of an individual hardware sensor (ID, type, location, status). • Report if its status is "Ready" or "Faulted".	SensorConfiguration
ArmingValidator	• **Provide a highly cohesive service for all arming validation logic.** • Check if all required sensors (for a given mode) are "Ready". • Return a list of faulted sensors to the caller. • Manage the logic for sensor bypass requests.	SecurityModeController, SecurityModeView, SensorConfiguration
ISettingsRepository	• Define methods for system settings persistence (e.g., `loadSettings`, `saveSettings`).	StorageManager(Implements), SecuritySettings(Uses)
ISensorConfigRepository	• Define methods for sensor configuration persistence (e.g., `loadSensorConfig`).	StorageManager(Implements), SensorConfiguration(Uses)
ILogRepository	• Define method for saving log entries (e.g., `saveLog`).	StorageManager(Implements), AuthenticationController(Uses), SecurityModeController(Uses), SystemLog(Uses)
SurveillanceStorageManager	• Provide the concrete implementation for all repository interfaces. • Manage database connections and execute SQL queries.	IUserRepository(Implements), ISettingsRepository(Implements), ILogRepository(Implements), ISessionRepository(Implements), ISensorConfigRepository(Implements)
CloudConnector	• Synchronize system state (mode, sessions) with the cloud server. • Handle all remote API communication and network failures.	AuthenticationController, SecurityModeController, SessionManager
NotificationService	• Send push notifications and SMS alerts to users. • Format notification messages.	AuthenticationController, SecurityModeController

3. CRC Cards – System And User Management

▼ CRC Cards

Class	Responsibilities	Collaborators
LoginView	• Display login form (email, password).• Capture user credentials.• Show 2FA code input when required.• Display authentication errors or loading indicators.• Navigate to main application on success.	• AuthenticationController (submits credentials, receives results)
SecuritySettings	• Store and manage the current security mode state (e.g., "Away", "Disarmed"). • Hold the configuration for each mode (via ModeConfig). • Store system-wide settings (e.g., exit delay duration). • Track bypassed sensors. • Persist changes via ISettingsRepository.	• SecurityModeController • ModeConfig • ISettingsRepository
AuthenticationController	• Orchestrate the user authentication workflow. • Receive login request from LoginView. • Request user data from IUserRepository. • Request credential validation from UserAccount entity. • Enforce 2FA (Two-Factor Authentication) if required.	• LoginView • IUserRepository • UserAccount • SessionManager • ILogRepository

Class	Responsibilities	Collaborators
	• Create an authenticated session via SessionManager. • Notify LoginView of authentication result. • Log all authentication attempts via ILogRepository. • Send notifications (e.g., new device login) via NotificationService.	• NotificationService • CloudConnector
SessionManager	• Generate secure session tokens (access, refresh). • Validate active session tokens. • Manage token expiration and refresh logic. • Revoke sessions on user logout. • Persist session data via ISessionRepository.	• AuthenticationController • UserAccount • ISessionRepository • CloudConnector
UserAccount	• **Store and manage user account data (username, passwordHash, role, 2FA settings).** • **Manage its own state (Active, Locked).** • Validate a given password against the stored hash. • Verify a given 2FA code. • **Increment failed login attempts.** • **Lock account if attempts exceed threshold.** • Check if account is currently locked. • Persist changes via IUserRepository.	• AuthenticationController • IUserRepository
SystemLog	• Represent a single log entry (timestamp, eventType, description). • Persist itself via ILogRepository.	• ILogRepository
SystemStorageManager	• Provide database connection abstraction • Execute SQL queries and commands • Perform CRUD operations for all entity classes • Manage database transactions • Handle connection pooling and error recovery	• SecuritySettings (persists security mode data) • SensorConfiguration (persists sensor data) • UserAccount (persists user credentials) • SystemLog (persists event logs) • SessionManager (persists session tokens)
CloudConnector	• Synchronize security mode state to SafeHome Cloud • Synchronize user sessions across devices • Upload system logs to cloud for backup • Send push notifications via cloud service • Handle network failures and retry logic • Maintain connection status	• SecurityModeController (syncs mode changes) • AuthenticationController (syncs session state) • SessionManager (syncs active sessions)
NotificationService	• Send push notifications to mobile devices • Send SMS notifications for critical events • Format notification messages • Handle notification delivery failures • Respect user notification preferences	• SecurityModeController (sends mode change alerts) • AuthenticationController (sends login alerts)
IUserRepository	• Define methods for user data persistence (e.g., `findUserByEmail`, `saveUser`).	• StorageManager (Implements) • AuthenticationController (Uses) • UserAccount (Uses)
ISessionRepository	• Define methods for session data persistence (e.G., `saveSession`, `findSession`).	• StorageManager (Implements) • SessionManager (Uses)

4. CRC Cards - Remote Access And Account

▼ CRC Card

Class	Responsibilities	Collaborators
UserAccount	• Store and manage user profile information (name, email, phone number) • Maintain account state (Active, Pending Verification, Suspended, Locked) • Store hashed password and enforce password security policies • Validate email and phone number verification status • Manage user's 2FA settings and backup codes	• CloudServer • SessionToken • PasswordPolicyManager

Class	Responsibilities	Collaborators
	• Maintain session list and active device tracking • Track account creation date and last login timestamp • Store and manage user preferences and settings	• TwoFactorAuthManager • AuditLogger
CloudServer	• Process and validate signup requests with OTP verification • Authenticate users during login and enforce 2FA • Generate secure session tokens (access and refresh tokens) • Revoke session tokens during logout and invalidate sessions • Handle password reset and change requests with verification • Manage account profile updates with atomic operations • Validate idempotent request IDs to prevent duplicate operations • Enforce rate limiting on sensitive operations (login, signup, password reset) • Implement CAPTCHA and human verification for abuse prevention • Perform final validation of new account uniqueness (email/phone) • Manage account state transitions and logging • Revoke all active sessions on critical account changes (password change, email/phone update) • Retrieve and return secure system responses	• UserAccount • SMSGateway • EmailService • SessionToken • PasswordPolicyManager • TwoFactorAuthManager • AuditLogger • RequestValidator • StorageManager
SessionToken	• Store access token with short expiration (e.g., 15-30 minutes) • Store refresh token with long expiration (e.g., 7-30 days) • Track token creation timestamp and expiration time • Identify associated user account and device • Validate token signature and integrity • Check token expiration status • Support token rotation on refresh • Store session metadata (device type, IP address, location) • Provide token revocation capability	• UserAccount • CloudServer • StorageManager
SMSGateway	• Send OTP verification codes with configurable expiration (e.g., 5 minutes) • Send authentication notifications to user's phone • Handle rate limiting per phone number • Track SMS delivery status and failures • Implement retry logic for failed deliveries • Handle SMS gateway provider errors and timeouts • Support throttling to prevent abuse • Log all SMS delivery attempts • Validate phone number format before sending	• CloudServer • UserAccount • AuditLogger
EmailService	• Send activation/verification emails with time-limited links (e.g., 24 hours) • Send password reset emails with single-use reset links (e.g., 1 hour) • Send notification emails for account changes (password change, email update, new login) • Send 2FA verification codes via email • Handle email delivery failures and bounces • Track email delivery status • Detect and handle disposable/invalid email domains • Implement retry logic for failed email deliveries • Support HTML and plain text email formats • Log all email delivery attempts and failures • Handle email service provider errors and timeouts	• CloudServer • UserAccount • AuditLogger
PasswordPolicyManager	• Validate password minimum length (e.g., ≥12 characters) • Check password complexity rules (uppercase, lowercase, numbers, special characters) • Verify password against breached password databases • Prevent password reuse from previous passwords • Check for common weak passwords and dictionary words • Enforce password history (prevent reuse within N previous passwords) • Validate password confirmation matches • Provide feedback on password strength • Enforce password expiration policies if required • Log password policy violations	• CloudServer • UserAccount • AuditLogger

Class	Responsibilities	Collaborators
TwoFactorAuthManager	• Generate secrets for authenticator app (TOTP) with QR code • Generate backup codes as recovery mechanism • Validate authenticator app codes within time window • Validate OTP codes sent via SMS or email • Track failed 2FA verification attempts • Implement temporary lockout after excessive failed attempts • Manage 2FA method switching (SMS to Authenticator, etc.) • Regenerate backup codes and invalidate old ones • Manage 2FA enablement/disablement • Check 2FA requirement during login • Enforce 2FA verification before critical account changes • Track 2FA enrollment timestamp and last verification	• UserAccount • CloudServer • SMSGateway • EmailService • AuditLogger
RequestValidator	• Validate idempotent request IDs to prevent duplicate processing • Check for CAPTCHA or human verification tokens • Validate request signature and authentication headers • Verify request came from allowed IP addresses (if applicable) • Check request timestamp for replay attack prevention • Validate all input data format and constraints • Detect and reject malformed requests • Rate limit requests per user and IP address • Log suspicious or invalid requests	• CloudServer • AuditLogger
AuditLogger	• Log successful signup events with timestamp and IP address • Log successful login events with device information • Log logout events and session termination • Log password change and reset events • Log profile update events (email, phone, name) • Log 2FA activation/deactivation and configuration changes • Log failed authentication attempts and account lockouts • Log account state transitions (Active, Suspended, etc.) • Log email/phone verification events • Log suspicious activities and policy violations • Log external service failures (SMS, Email gateway) • Maintain tamper-evident audit trail • Support audit log retrieval for compliance • Enforce audit log retention policies • Flag and alert on anomalous activities	• CloudServer • UserAccount • SMSGateway • EmailService • TwoFactorAuthManager • StorageManager
AccessStorageManager	• Atomically create new user accounts in database • Retrieve user account data by email or user ID • Update user profile information • Store password hashes using secure algorithms (Argon2, bcrypt) • Update account state and verification status • Store session tokens with expiration metadata • Retrieve active sessions for a user • Delete revoked session tokens • Store OTP and verification link data with expiration • Store 2FA backup codes and secrets • Persist audit log entries • Handle database transactions for critical operations • Implement database connection pooling • Support data encryption at rest for sensitive fields	• UserAccount • SessionToken • CloudServer • AuditLogger • TwoFactorAuthManager

5. CRC Cards - Indoor Monitoring And Device Control

▼ CRC Card

Class	Responsibilities	Collaborators
DeviceController	• Store device metadata (device ID, name, device type, current state)• Maintain device status (online/offline, battery level, signal strength)• Track device group associations and scene memberships• Validate control commands before transmission• Store device-specific settings and configurations• Manage device communication protocols (Zigbee, Wi-Fi, Z-Wave)• Handle command execution and state updates• Support real-time state synchronization• Maintain command history and logs per device• Detect and report unresponsive devices	• SafeHomeHub• CloudServer• DeviceStateManager• CommandQueue• NotificationService• AuditLogger
SmartLightDevice	• Store light-specific properties (brightness level, color, color temperature) • Track light on/off state and current brightness percentage • Maintain light group membership • Support brightness adjustment (0-100%) • Support color control (RGB, warm/cool temperature) • Report current status to controller • Execute light control commands (on, off, dim, color change) • Implement fade-in/fade-out effects • Support scheduled commands • Log light state changes with timestamps	• DeviceController • SafeHomeHub • LightGroupManager • SceneManager
LightGroupManager	• Create and store light groups with custom names (e.g., "Living Room Lights", "Downstairs Lights") • Add/remove lights from groups • Store group membership and relationships • Execute commands across all lights in a group simultaneously • Manage group-level settings and defaults • Support nested groups or hierarchies • Validate group compositions • Track which lights are available within a group • Handle partial failures when controlling group (some lights responsive, some not) • Provide group status based on constituent lights	• SmartLightDevice • DeviceController • SceneManager • CloudServer
SceneManager	• Create and store scenes (e.g., "Movie Mode", "Good Morning", "Bedtime") • Define scene composition (which devices are affected and to what state) • Store scene trigger conditions and schedules • Execute all scene commands atomically • Manage scene activation and deactivation • Support scene customization by homeowners • Handle scene rollback or state restoration • Track active scenes and conflicting commands • Support scene scheduling and automation • Log scene execution with timestamps	• DeviceController • SmartLightDevice • LightGroupManager • SchedulingService • CloudServer
AirQualityMonitor	• Receive and store air quality sensor readings (CO_2, VOCs, PM2.5, humidity, temperature) • Store sensor data with timestamp and accuracy information • Maintain historical air quality trends • Track current and peak pollutant levels • Compare readings against configured thresholds (healthy, moderate, unhealthy) • Detect threshold violations and trigger alerts • Store air quality baseline and seasonal patterns • Manage multiple sensor data aggregation • Provide air quality recommendations • Support configurable thresholds per pollutant type	• AirQualitySensor • AutomationEngine • ThresholdManager • NotificationService • CloudServer
VentilationController	• Store ventilation system status (on, off, speed level) • Receive activation/deactivation commands from automation engine • Send ventilation control commands to hub • Track ventilation runtime and operational status • Support variable speed control (low, medium, high)	• AirQualityMonitor • AutomationEngine • SafeHomeHub • CommandQueue • NotificationService

Class	Responsibilities	Collaborators
	• Store ventilation schedules and quiet hours • Handle ventilation system failures and timeouts • Implement retry logic for unresponsive systems • Track energy consumption of ventilation • Provide ventilation status feedback	
AutomationEngine	• Evaluate air quality thresholds against sensor readings • Trigger ventilation activation when thresholds are exceeded • Manage automation rule evaluation and execution • Support local automation execution when cloud is unavailable • Execute automation commands without user intervention • Store automation rules and configurations • Handle rule priorities and conflict resolution • Log automation triggers and actions • Support temporary override of automated actions • Manage automation scheduling and timing	• AirQualityMonitor • VentilationController • ThresholdManager • SafeHomeHub • CloudServer
ThresholdManager	• Store and manage air quality thresholds (CO_2, VOCs, particulates) • Define healthy, moderate, and unhealthy threshold levels • Support homeowner customization of thresholds • Validate threshold configurations • Apply health standards recommendations (WHO, EPA) • Store threshold presets for different room types • Support time-based threshold variations (day/night) • Persist threshold history and changes • Provide threshold recommendations • Support threshold backup and recovery	• AirQualityMonitor • AutomationEngine • CloudServer • AuditLogger
SmartMeter	• Store and report real-time power consumption (watts) • Aggregate power readings over time periods • Track total daily, weekly, and monthly energy usage (kWh) • Identify peak consumption times and patterns • Store consumption history with granular timestamps • Support multiple rate periods (time-of-use pricing) • Report meter status and connectivity • Validate power readings for anomalies • Store meter calibration data • Support power generation tracking (solar panels)	• SafeHomeHub • EnergyAnalyzer • CloudServer
EnergyAnalyzer	• Retrieve power consumption data from smart meter • Calculate peak usage hours and off-peak periods • Generate daily, weekly, and monthly consumption reports • Compare current usage with historical baselines • Identify energy-saving opportunities and anomalies • Support appliance-level analysis (if sub-metering is available) • Generate exportable reports (PDF, CSV format) • Calculate estimated costs based on rate structures • Support trend analysis and forecasting • Provide energy efficiency recommendations	• SmartMeter • ReportGenerator • CloudServer • NotificationService
CommandQueue	• Queue device control commands for transmission • Maintain command priority ordering • Store pending commands with metadata • Implement command retry logic for failed transmissions • Track command status (queued, sent, acknowledged, failed) • Support command deduplication • Timeout and purge stale commands • Log all queued commands with timestamps • Support batch command submission • Handle queue overflow gracefully	• DeviceController • SafeHomeHub • CloudServer • AuditLogger
DeviceStateManager	• Store current state of all connected devices • Maintain state history with timestamps • Detect and log state change events	• DeviceController • CloudServer

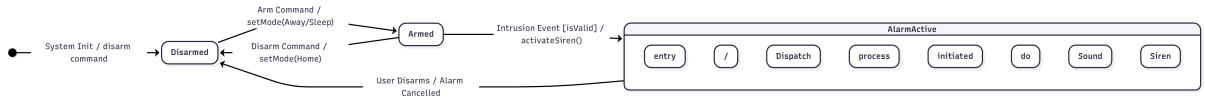
Class	Responsibilities	Collaborators
	• Support state synchronization across clients • Implement optimistic updates for responsive UI • Handle conflicting state updates from multiple sources • Provide state consistency validation • Cache frequently accessed state • Support state rollback and recovery • Notify subscribers of state changes	• WebSocketManager • AuditLogger
SafeHomeHub	• Maintain connection with all registered devices (Zigbee, Wi-Fi, Z-Wave) • Forward commands from cloud to devices • Receive and relay device status updates to cloud • Poll devices for status periodically • Execute local automation when cloud is unavailable • Buffer commands during connectivity loss • Support device pairing/discovery • Manage wireless network configuration • Log all hub-to-device communications • Monitor hub health and connectivity	• DeviceController • AutomationEngine • CloudServer • CommandQueue • AirQualityMonitor • VentilationController
CloudServer	• Receive device control commands from mobile/web apps • Relay commands to SafeHome hub for device execution • Aggregate sensor and device data from hub • Push real-time status updates to connected clients • Manage user sessions and permissions for device control • Store historical data and generate reports • Coordinate cloud-based automations • Handle notification distribution • Support data synchronization across devices • Implement rate limiting on control commands • Log all control operations and state changes	• DeviceController • SafeHomeHub • DeviceStateManager • WebSocketManager • NotificationService • ReportGenerator • AuditLogger • StorageManager
WebSocketManager	• Establish WebSocket connections to mobile and web clients • Push real-time device status updates to connected clients • Support multiple concurrent client connections • Handle client connection/disconnection events • Implement message queuing for offline clients • Authenticate client connections • Broadcast updates to all connected clients • Manage connection timeouts and keepalive • Support client message subscriptions (specific devices or groups)	• CloudServer • DeviceStateManager • NotificationService
NotificationService	• Send notifications for device status changes • Alert on device failures and unresponsive devices • Send air quality alerts when thresholds are exceeded • Notify homeowner of ventilation system activation • Send power consumption alerts • Support multiple notification channels (push, email, SMS) • Queue notifications for delivery retry • Store notification preferences per user • Track notification delivery status • Implement notification rate limiting to prevent spam	• DeviceController • AirQualityMonitor • EnergyAnalyzer • VentilationController • CloudServer • AuditLogger
ReportGenerator	• Generate daily/weekly/monthly energy consumption reports • Create device usage summaries • Produce comparison reports (current vs. historical) • Format reports in multiple formats (PDF, CSV, JSON) • Include visualizations and charts • Calculate cost estimates based on rate structures • Generate appliance-specific breakdowns • Support scheduled report generation • Email reports to homeowner • Store report generation logs	• EnergyAnalyzer • SmartMeter • CloudServer • StorageManager

Class	Responsibilities	Collaborators
AuditLogger	• Log all device control commands with user and timestamp • Record device state changes and transitions • Log automation triggers and executions • Track sensor reading anomalies • Record notification dispatch events • Log cloud connectivity events and outages • Track threshold configuration changes • Maintain tamper-evident audit trail • Support audit log retrieval for compliance • Flag suspicious patterns or unauthorized access attempts	• CloudServer • DeviceController • AutomationEngine • NotificationService • StorageManager
ControlStorageManager	• Persist device configurations and metadata • Store device state history with timestamps • Archive sensor readings and air quality data • Store power consumption data from smart meter • Persist automation rules and scene definitions • Store user preferences and thresholds • Archive generated reports • Implement data retention policies • Support efficient time-series data querying • Handle database transactions for consistency • Implement data backup and recovery mechanisms	• DeviceController • CloudServer • EnergyAnalyzer • AuditLogger • ReportGenerator

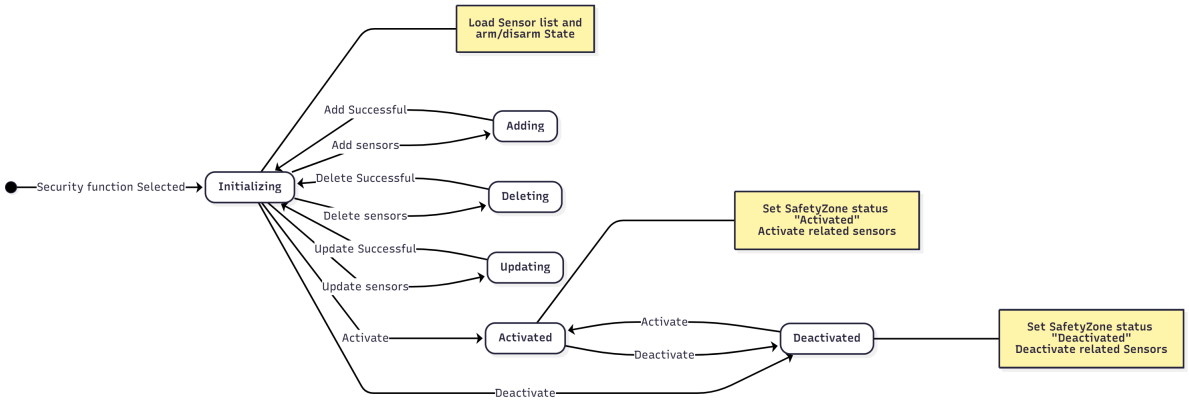
v. State Diagrams

1. State Diagram - Intelligent Security

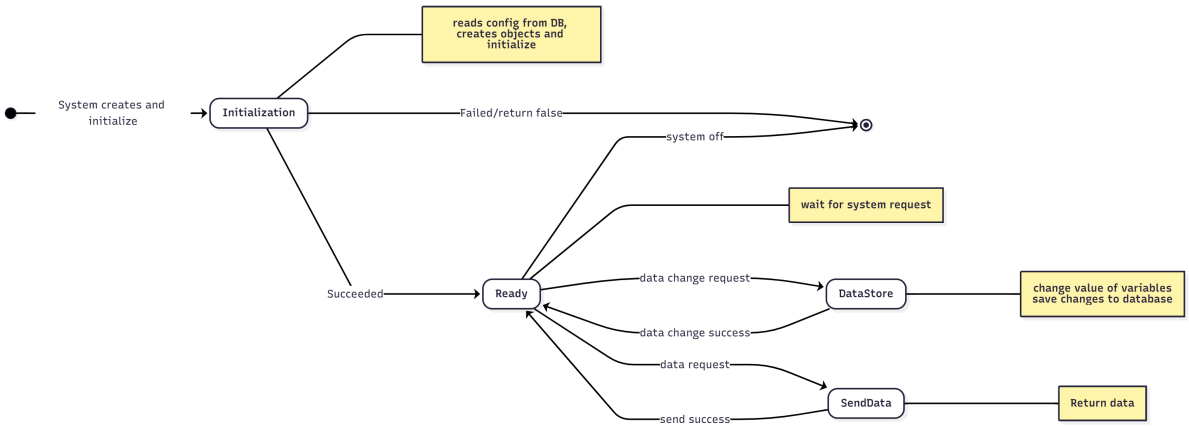
▼ 1.1 SafeHomeHub



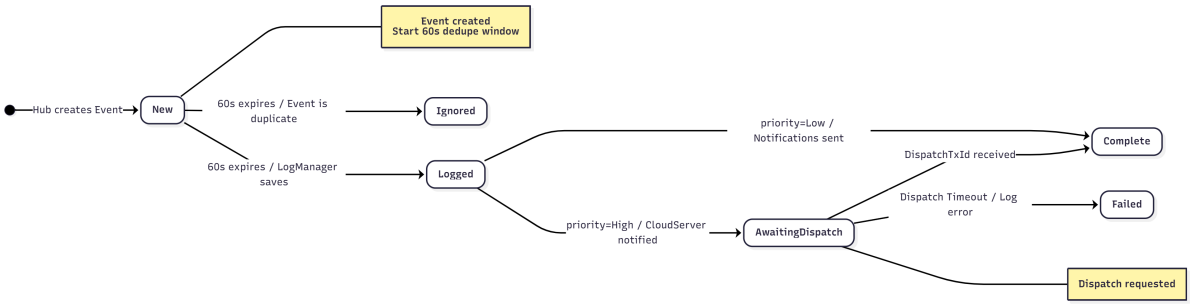
▼ 1.2 SafetyZone



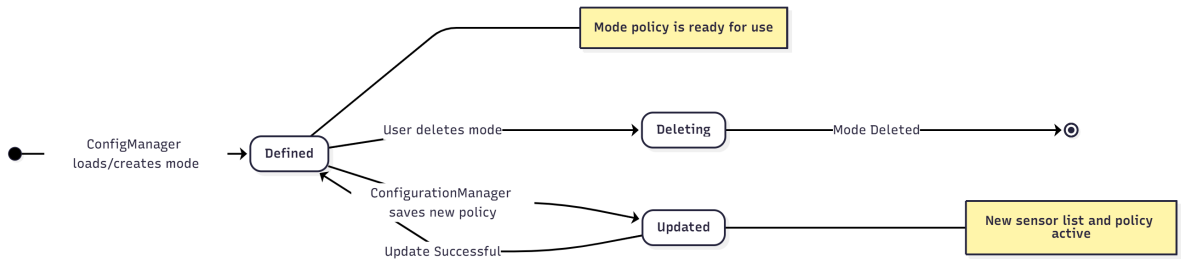
▼ 1.3 ConfigurationManager



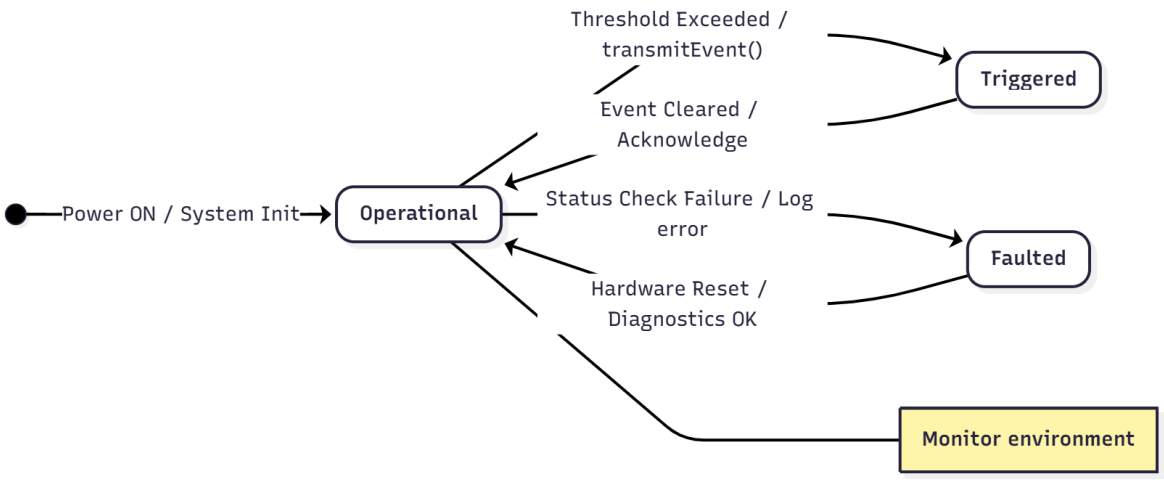
▼ 1.4 AlarmEvent



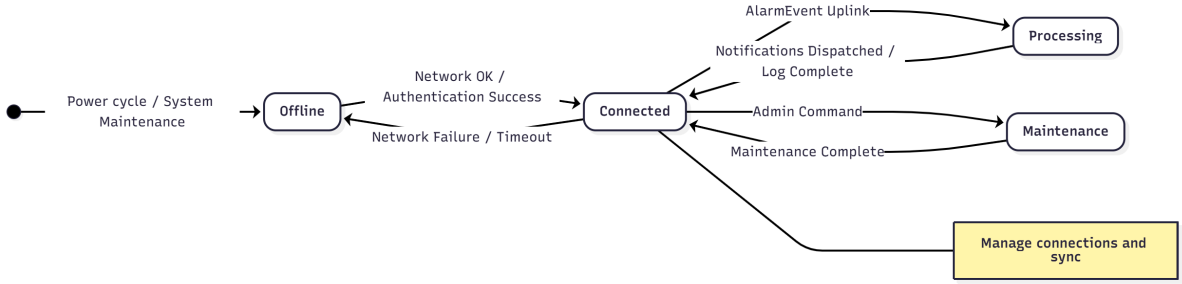
▼ 1.5 SafeHomeMode



▼ 1.6 Sensor



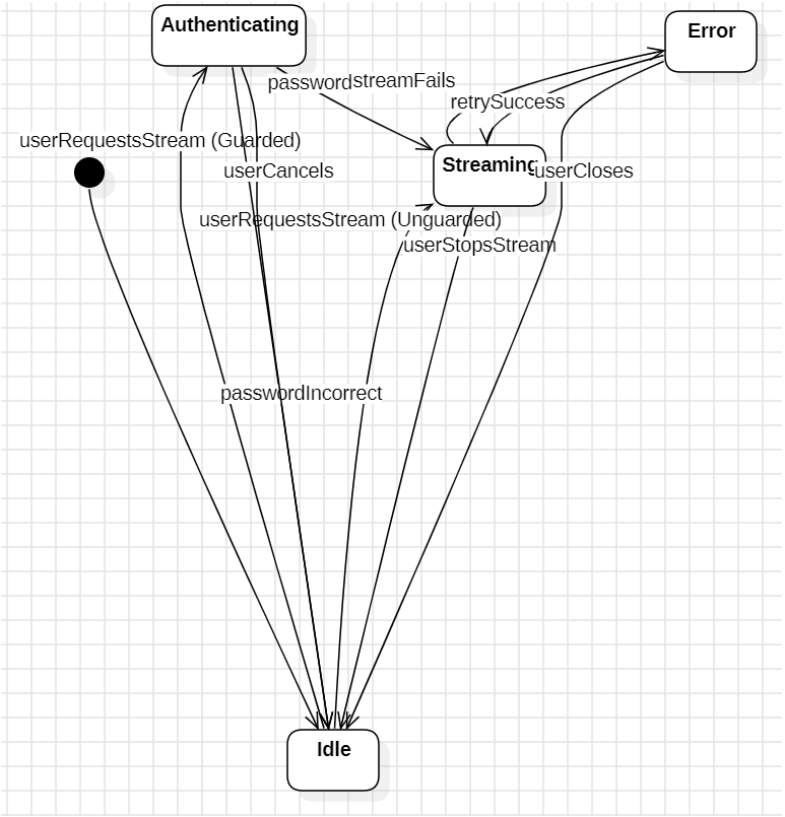
▼ 1.7 CloudServer



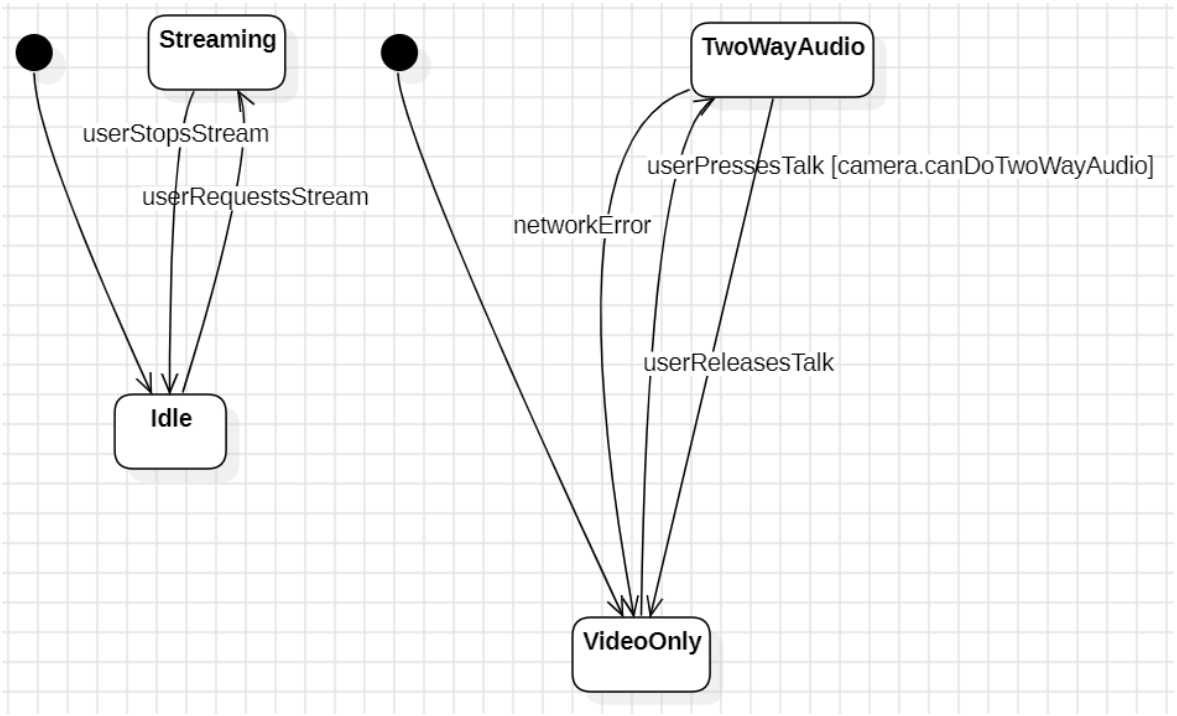
2. State Diagram - Live Surveillance

▼ 2.1. Camera Viewing and Control

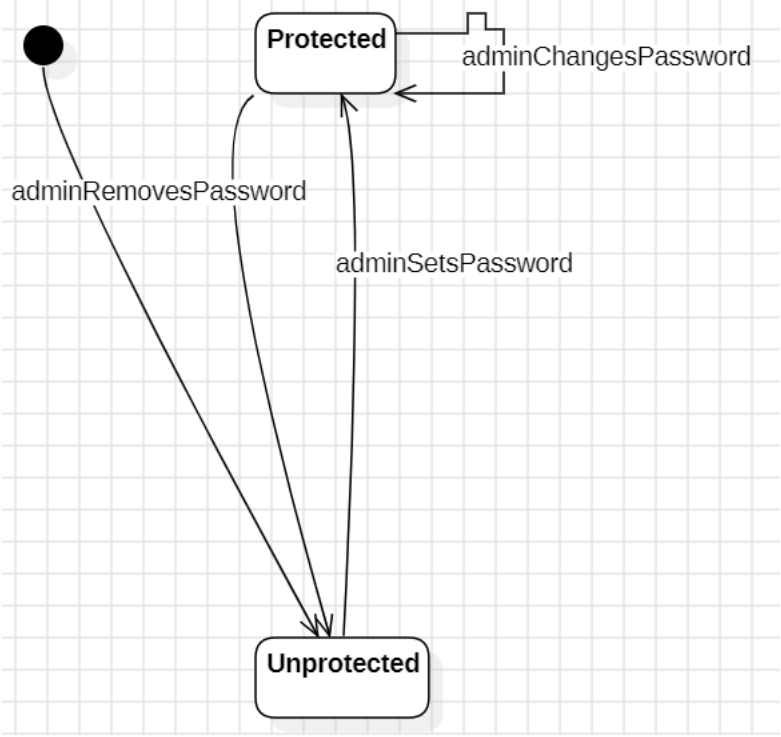
▼ 2.1.1 Single Camera Live View



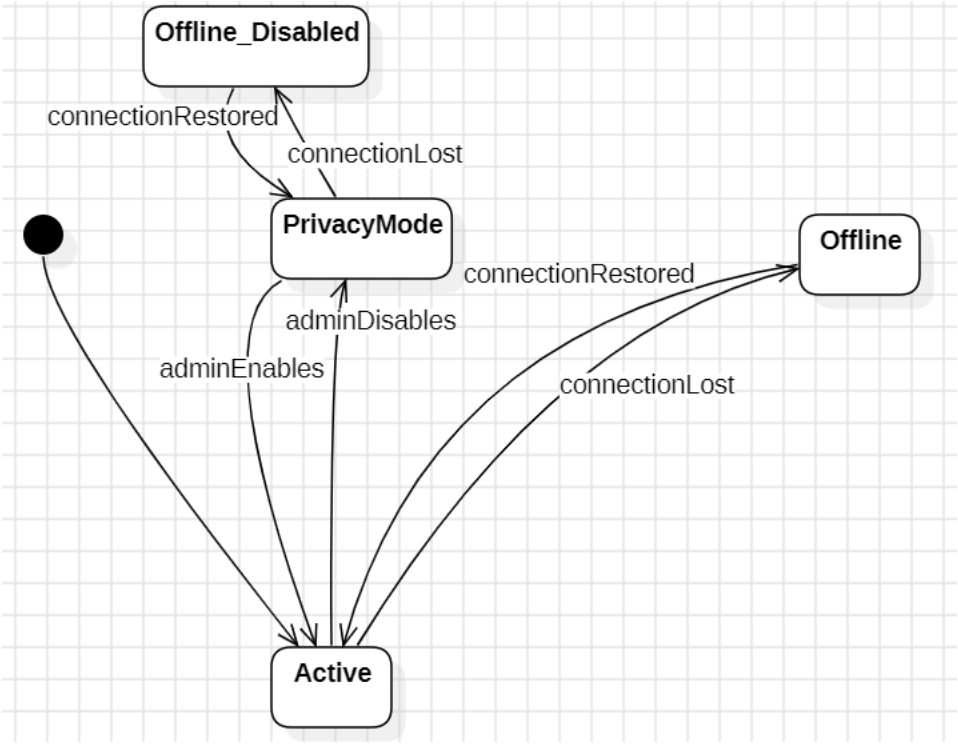
▼ 2.1.2 Two-Way Audio



▼ 2.1.3 Protect Sensitive Camera Feed with a Password

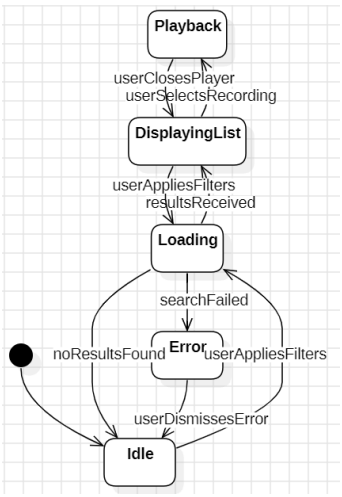


▼ 2.1.4 Camera Activation and Deactivation

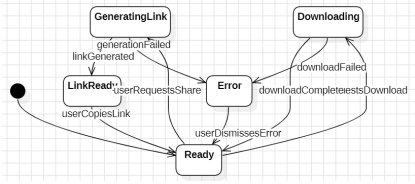


▼ 2.2. Recording and Evidence Management

▼ 2.2.1 Search and Playback Recordings

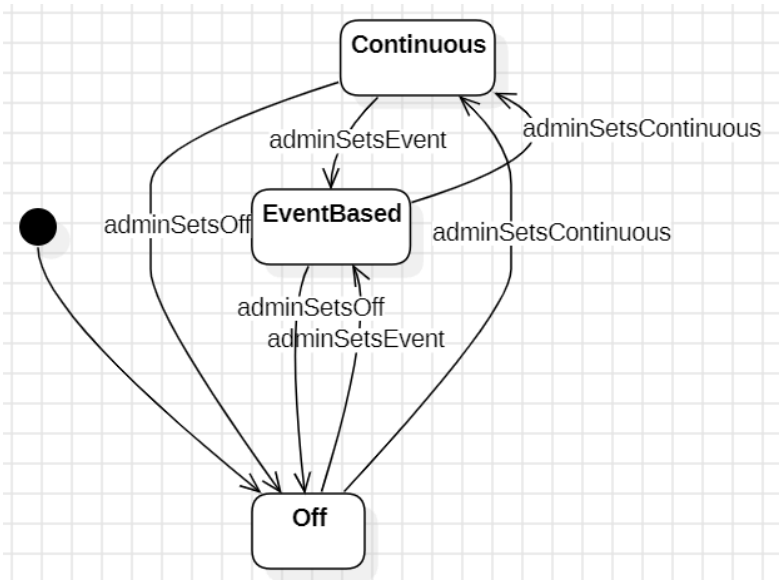


▼ 2.2.2 Evidence Sharing and Export

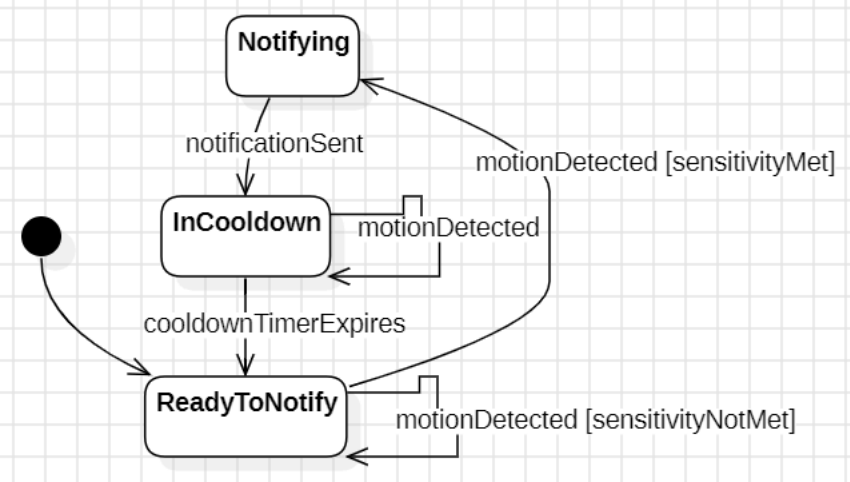


▼ 2.3. Surveillance Settings

▼ 2.3.1 Recording Settings



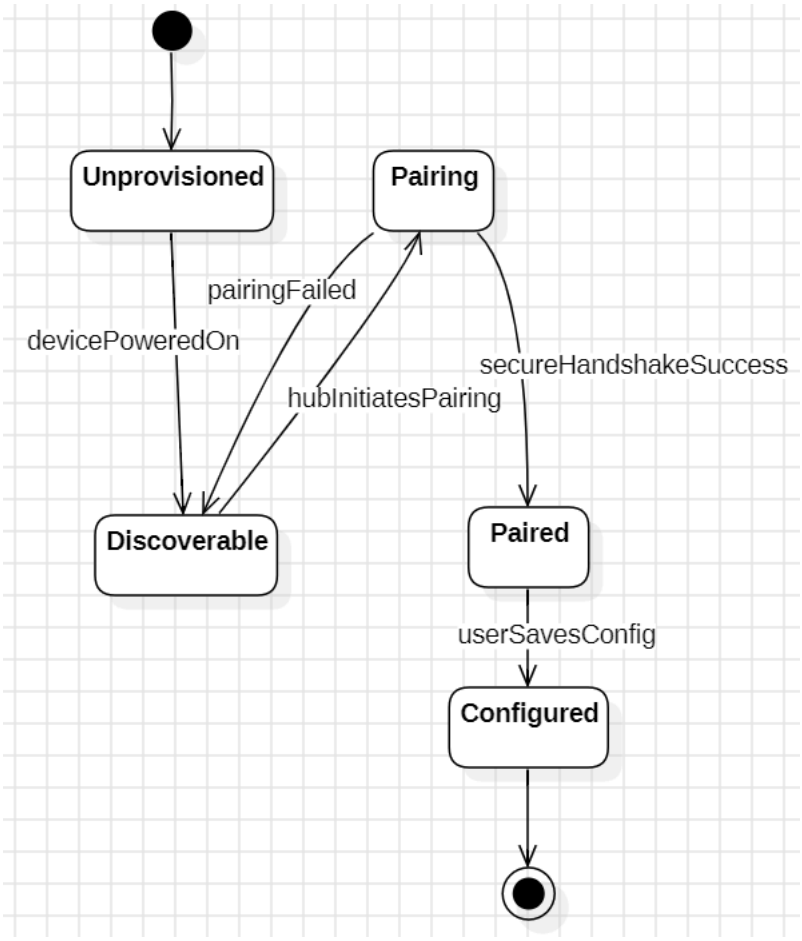
▼ 2.3.2 Notification Policy and Cooldown



3. State Diagram - System and User Management

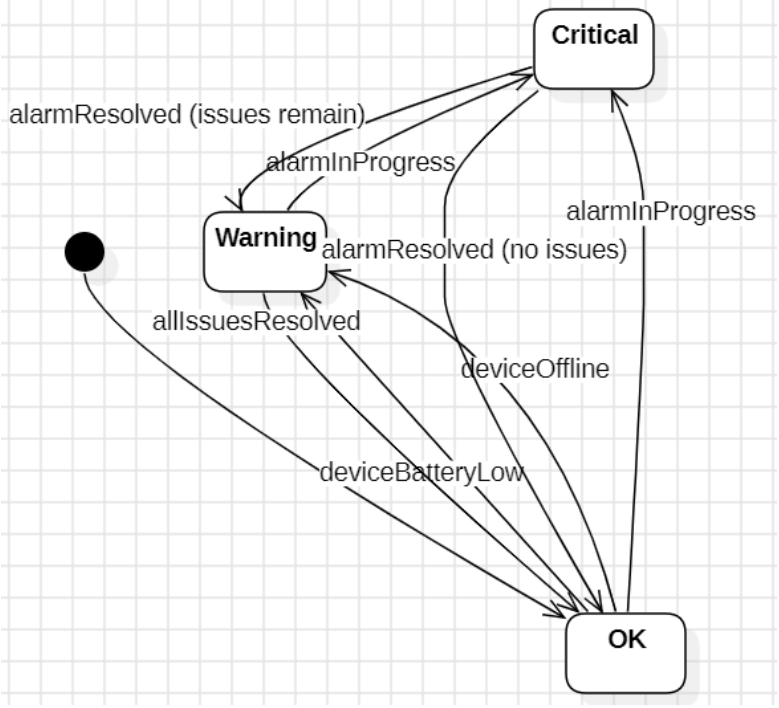
▼ 3.1. Device Management

▼ 3.1.1 Add and Configure New Devices

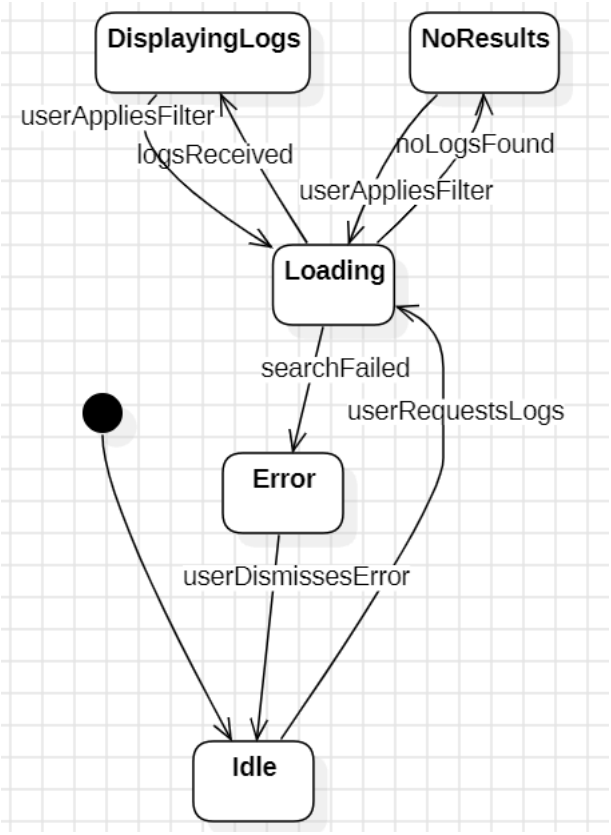


▼ 3.2. System Status and Logs

▼ 3.2.1 System Status Dashboard

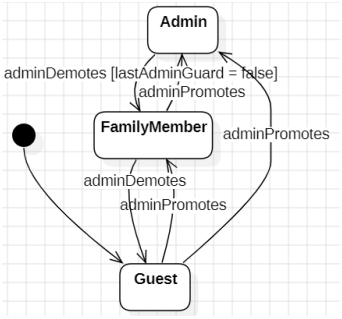


▼ 3.2.2 Activity Logs and Timeline



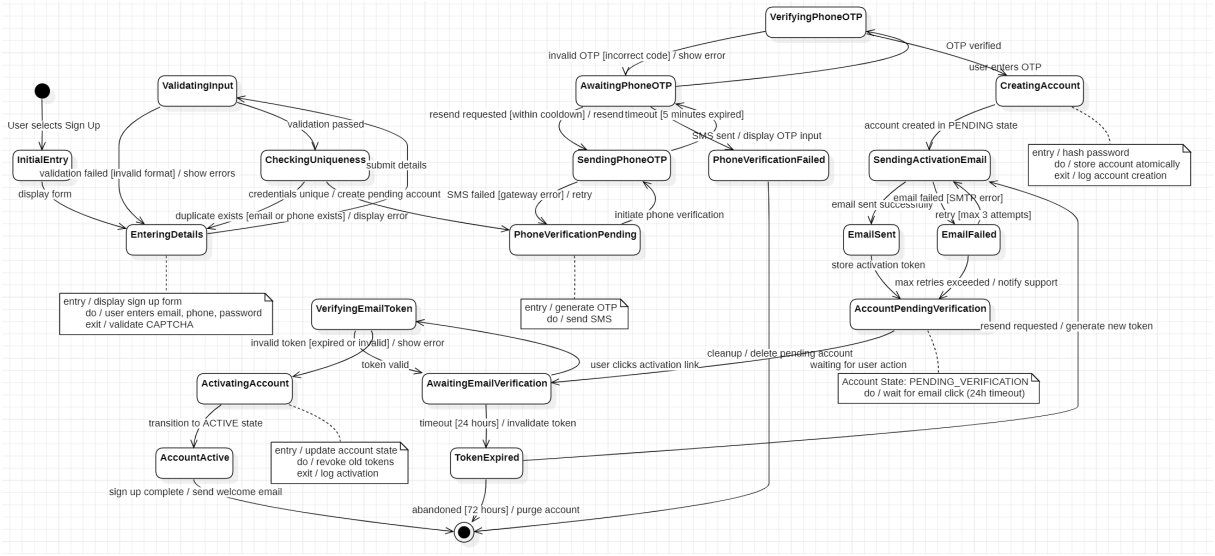
▼ 3.3. User and Permission Management

▼ 3.3.1 User Role and Access Control

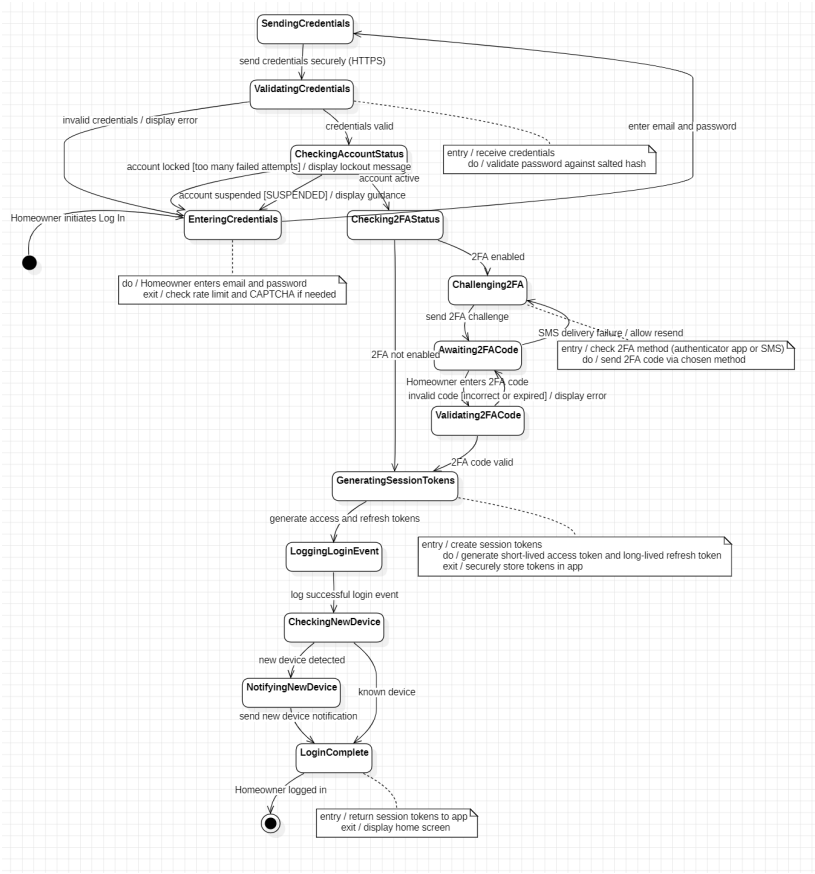


4. State Diagram - Remote Access And Account

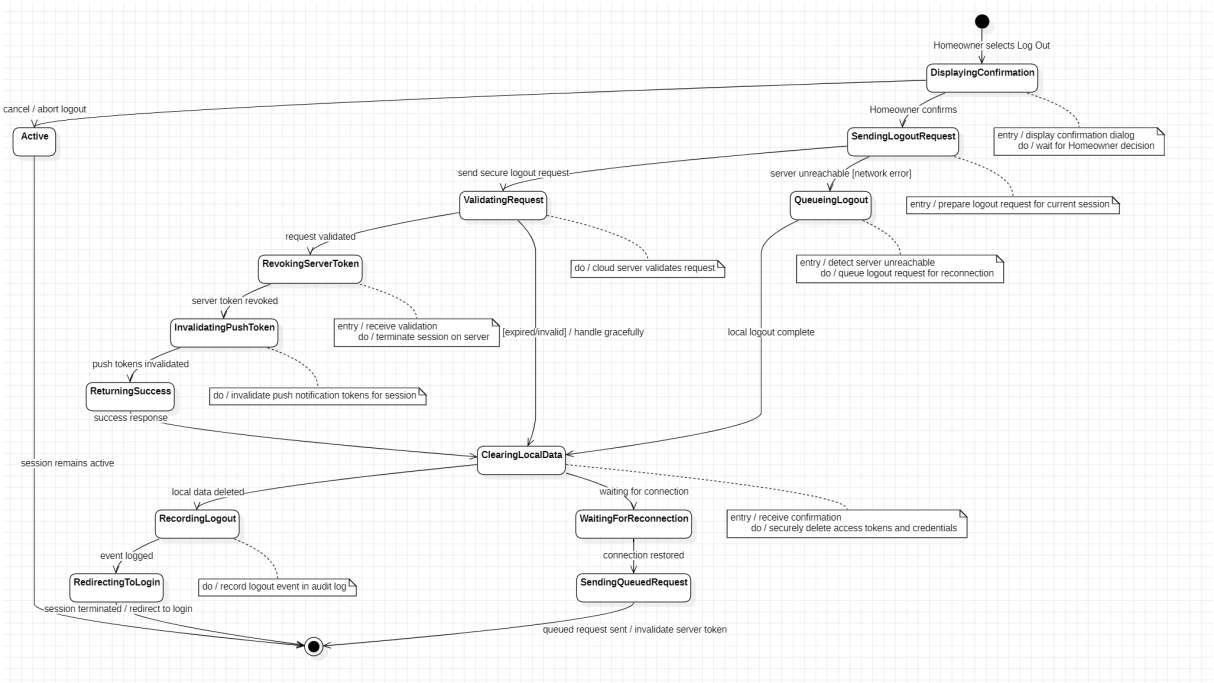
▼ 4.1 Sign Up



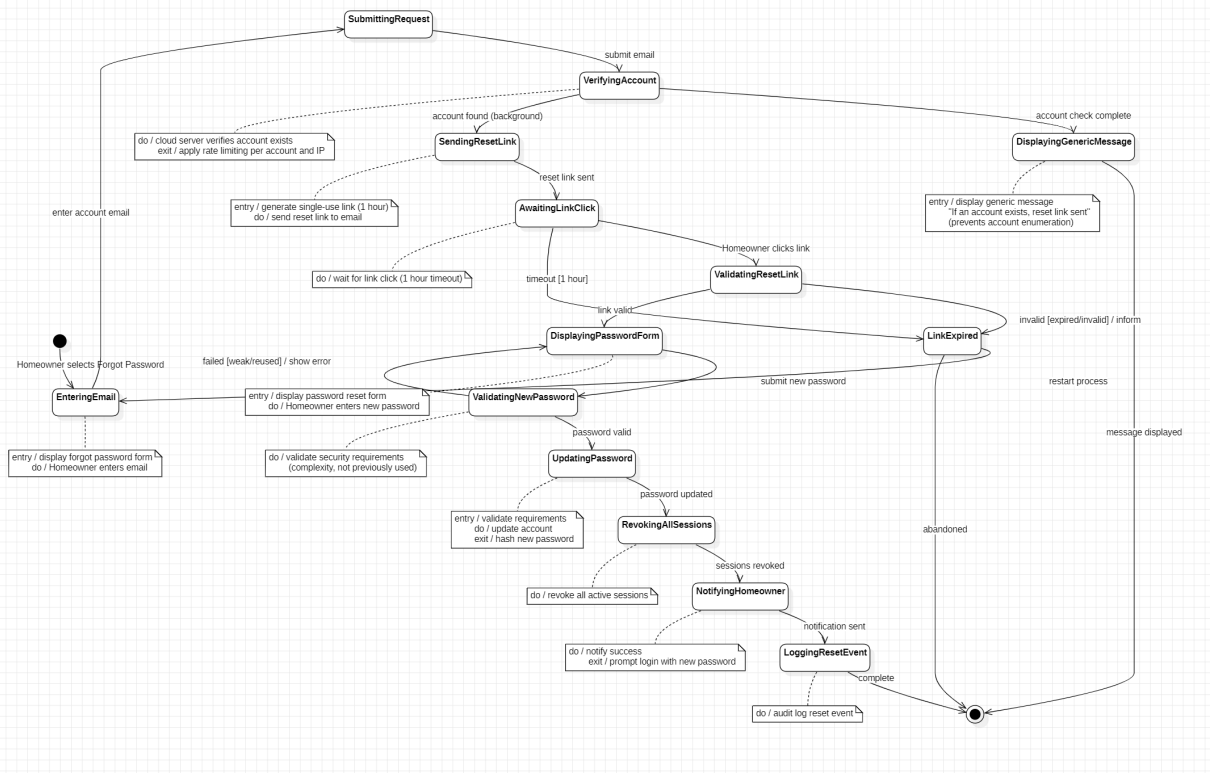
▼ 4.2 Log In



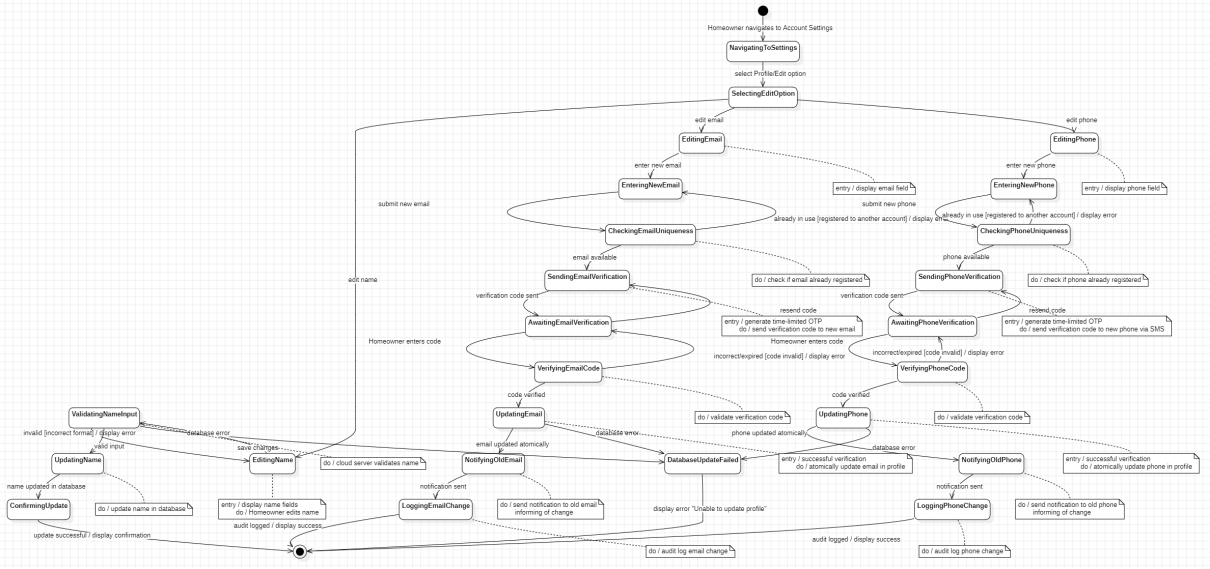
▼ 4.3 Log Out



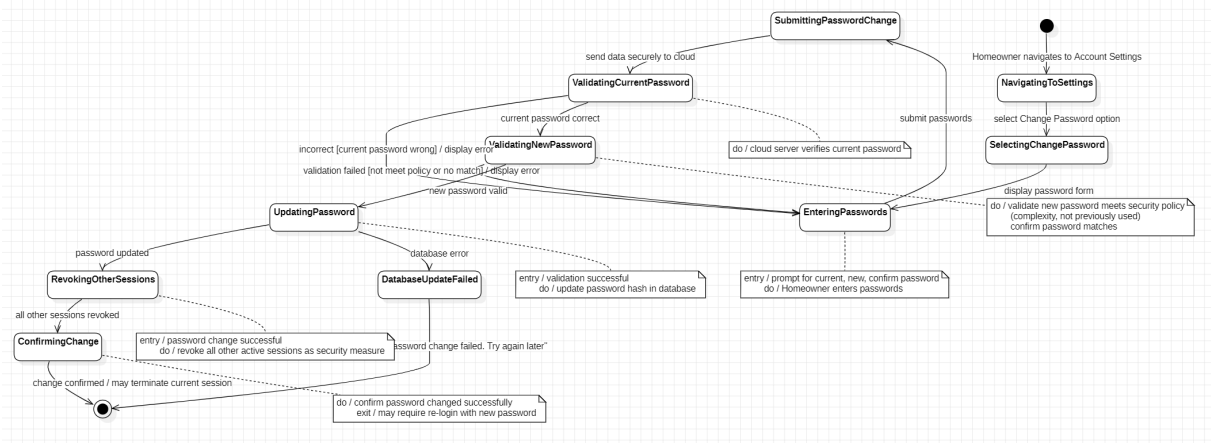
▼ 4.4 Password Recovery and Reset



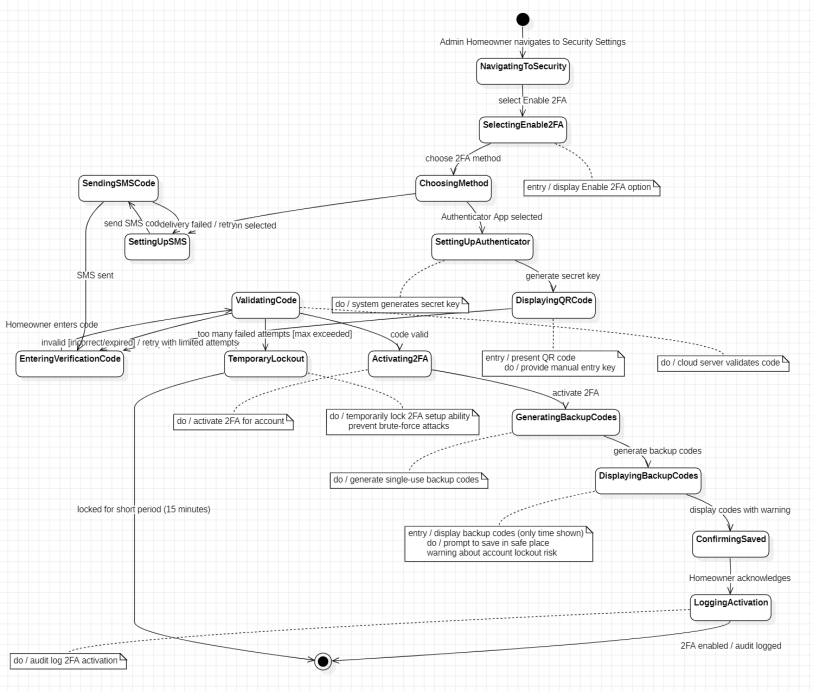
▼ 4.5 Edit Profile Information



▼ 4.6 Change Password

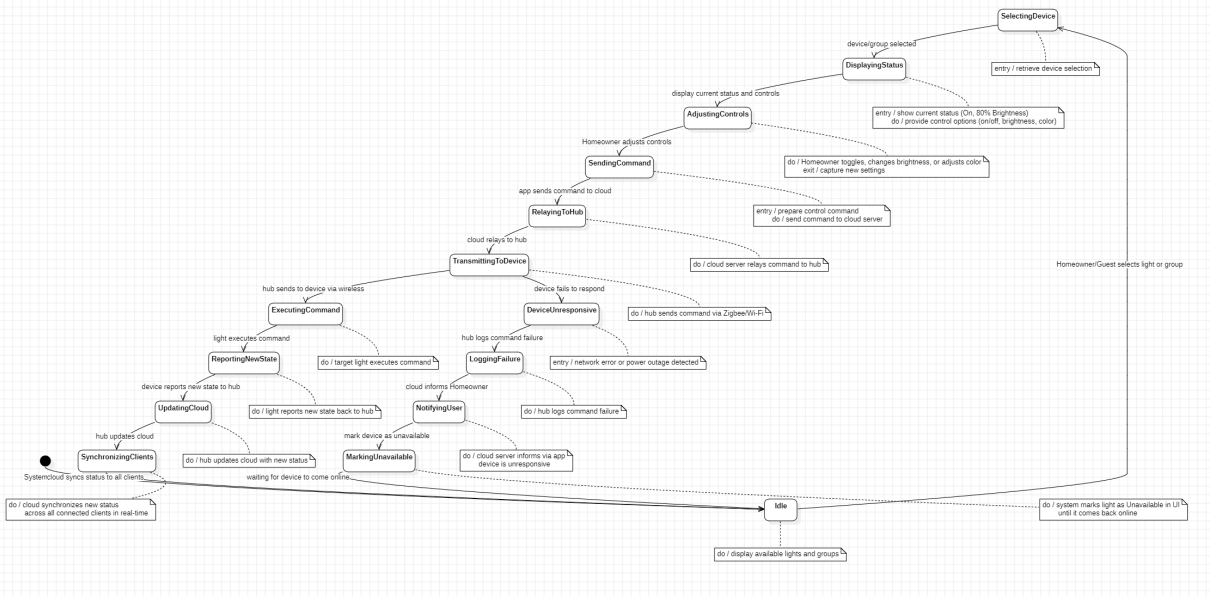


▼ 4.7 Two-Factor Authentication Management

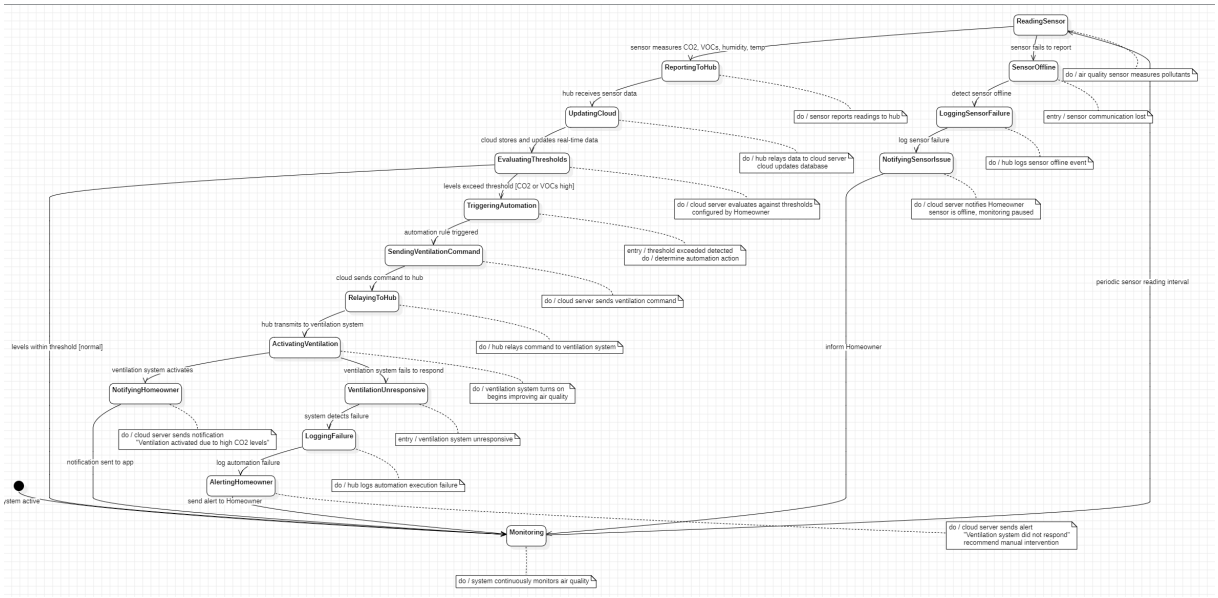


5. State Dagram - Indoor Monitoring And Device Control

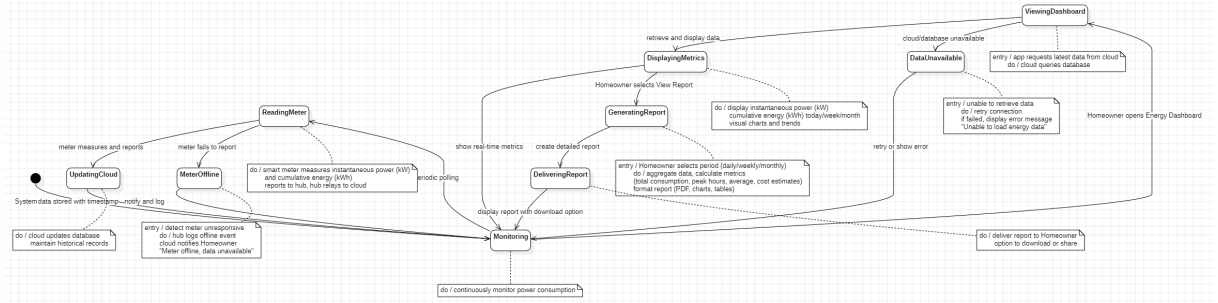
▼ 5.1 Indoor Device Control



▼ 5.2 Air Quality Monitoring



▼ 5.3 Power Consumption Monitoring



vi. Design Evaluation

1. Architectural Design Metric

Fenton's simple morphology metrics (Class Diagram - Whole System Overview)

- node = 14
- arc = 18
- size = 32
- depth = 7
- width = 4
- arc-to-node ratio = 1.29

2. CK Metrics

▼ CK Metrics Detail

1. Account

Methods:

- signUp()
- login()
- logout()
- changePassword()

Attributes:

- accountId, email, passwordHash, accountStatus

Metrics:

- **WMC** = 4 (4 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: User, Session)
- **RFC** = 4 + 2 = 6
  - Own methods: 4
  - Called methods: Session.create() (login), Session.revoke() (logout)
- **LCOM** = 0 (all methods use accountId, email, passwordHash → high cohesion)

---

2. User

Methods:

- manageDevices()
- viewIncidents()

Attributes:

- userId, name, role, permissions

Metrics:

- **WMC** = 2 (2 methods)

- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 4 (coupled to: Account, Device, SecurityMode, AutomationRule)
- **RFC** = 2 + 4 = 6
  - Own methods: 2
  - Called methods: Device.updateStatus(), SecurityMode.activate(), AutomationRule.execute(), Incident.view() (implied)
- **LCOM** = 0 (both methods use userId → high cohesion)

---

3. Device (Abstract)

Methods:

- register()
- updateStatus()

Attributes:

- deviceId, deviceName, status

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (root of hierarchy)
- **NOC** = 3 (children: Sensor, Camera, SmartDevice)
- **CBO** = 2 (coupled to: Hub, CloudServer)
- **RFC** = 2 + 2 = 4
  - Own methods: 2
  - Called methods: Hub.relayCommand(), CloudServer.syncClients()
- **LCOM** = 0 (both methods use deviceId, status → high cohesion)

---

4. Sensor (extends Device)

Methods:

- reportReading()

Attributes:

- sensorType, threshold

Metrics:

- **WMC** = 1 + 2 (inherited) = 3 (1 own + 2 inherited)
- **DIT** = 1 (inherits from Device)
- **NOC** = 2 (children: AirQualitySensor, PowerMeter - implied)
- **CBO** = 3 (coupled to: Device, Incident, AirQualityMonitor)
- **RFC** = 1 + 3 = 4
  - Own methods: 1
  - Called methods: Incident.create(), AirQualityMonitor.checkThreshold(), Device.updateStatus() (inherited)
- **LCOM** = 0 (reportReading uses sensorType, threshold → high cohesion)

---

5. Camera (extends Device)

Methods:

- startStream()
- stopStream()

Attributes:

- resolution, streamUrl

Metrics:

- **WMC** = 2 + 2 (inherited) = 4 (2 own + 2 inherited)
- **DIT** = 1 (inherits from Device)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: Device, Recording)
- **RFC** = 2 + 2 = 4
  - Own methods: 2
  - Called methods: Recording.start(), Device.updateStatus() (inherited)
- **LCOM** = 0 (both methods use resolution, streamUrl → high cohesion)

---

6. SmartDevice (extends Device)

Methods:

- executeCommand()

Attributes:

- deviceType

Metrics:

- **WMC** = 1 + 2 (inherited) = 3 (1 own + 2 inherited)
- **DIT** = 1 (inherits from Device)
- **NOC** = 0 (no children in this diagram)
- **CBO** = 2 (coupled to: Device, AutomationRule)
- **RFC** = 1 + 2 = 3
  - Own methods: 1
  - Called methods: AutomationRule.execute(), Device.updateStatus() (inherited)
- **LCOM** = 0 (executeCommand uses deviceType → high cohesion)

---

7. SecurityMode

Methods:

- activate()
- deactivate()

Attributes:

- modeName, isActive

Metrics:

- **WMC** = 2 (2 methods)

- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 3 (coupled to: User, Sensor, AutomationRule)
- **RFC** = 2 + 3 = 5
  - Own methods: 2
  - Called methods: Sensor.reportReading(), AutomationRule.execute(), User.notify() (implied)
- **LCOM** = 0 (both methods use modeName, isActive → high cohesion)

---

8. Incident

Methods:

- create()
- resolve()

Attributes:

- incidentId, incidentType, severity, timestamp

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: Sensor, Recording)
- **RFC** = 2 + 2 = 4
  - Own methods: 2
  - Called methods: Sensor.getReading() (implied), Recording.start() (for evidence)
- **LCOM** = 0 (both methods use incidentId, timestamp → high cohesion)

---

9. Recording

Methods:

- start()
- stop()
- download()

Attributes:

- recordingId, startTime, storageLocation

Metrics:

- **WMC** = 3 (3 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: Camera, Incident)
- **RFC** = 3 + 2 = 5
  - Own methods: 3
  - Called methods: Camera.captureSnapshot(), CloudServer.storeData()

- **LCOM** = 0 (all methods use recordingId, startTime, storageLocation → high cohesion)

---

10. AirQualityMonitor

Methods:

- checkThreshold()
- triggerVentilation()

Attributes:

- co2Level, vocLevel

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: Sensor, SmartDevice - implied VentilationSystem)
- **RFC** = 2 + 2 = 4
  - Own methods: 2
  - Called methods: Sensor.reportReading(), SmartDevice.executeCommand() (for ventilation)
- **LCOM** = 0 (both methods use co2Level, vocLevel → high cohesion)

---

11. PowerMonitor

Methods:

- recordReading()
- generateReport()

Attributes:

- instantaneousPower, cumulativeEnergy

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 1 (coupled to: Sensor - PowerMeter)
- **RFC** = 2 + 1 = 3
  - Own methods: 2
  - Called methods: Sensor.reportReading() (PowerMeter)
- **LCOM** = 0 (both methods use instantaneousPower, cumulativeEnergy → high cohesion)

---

12. AutomationRule

Methods:

- execute()
- enable()

Attributes:

- ruleId, ruleName, isEnabled

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 3 (coupled to: User, Sensor, SmartDevice)
- **RFC** = 2 + 3 = 5
  - Own methods: 2
  - Called methods: Sensor.reportReading(), SmartDevice.executeCommand(), User.notify() (implied)
- **LCOM** = 0 (both methods use ruleId, isEnabled → high cohesion)

---

13. Hub

Methods:

- relayCommand()
- collectData()

Attributes:

- hubId

Metrics:

- **WMC** = 2 (2 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 2 (coupled to: Device, CloudServer)
- **RFC** = 2 + 4 = 6
  - Own methods: 2
  - Called methods: Device.register(), Device.updateStatus(), CloudServer.storeData(), CloudServer.syncClients()
- **LCOM** = 0 (both methods use hubId → high cohesion)

---

14. CloudServer

Methods:

- authenticateUser()
- storeData()
- syncClients()

Attributes:

- serverId

Metrics:

- **WMC** = 3 (3 methods)
- **DIT** = 0 (no inheritance)
- **NOC** = 0 (no children)
- **CBO** = 3 (coupled to: Account, Device, Hub)
- **RFC** = 3 + 3 = 6

    - Own methods: 3
    - Called methods: Account.login(), Device.updateStatus(), Hub.collectData()
- **LCOM** = 1 (methods have some independence, but moderate cohesion)

▼ Summary Table

Class	WMC	DIT	NOC	CBO	RFC	LCOM
Account	4	0	0	2	6	0
User	2	0	0	4	6	0
Device	2	0	3	2	4	0
Sensor	3	1	2	3	4	0
Camera	4	1	0	2	4	0
SmartDevice	3	1	0	2	3	0
SecurityMode	2	0	0	3	5	0
Incident	2	0	0	2	4	0
Recording	3	0	0	2	5	0
AirQualityMonitor	2	0	0	2	4	0
PowerMonitor	2	0	0	1	3	0
AutomationRule	2	0	0	3	5	0
Hub	2	0	0	2	6	0
CloudServer	3	0	0	3	6	1
Session	3	0	0	1	4	0

System-Wide Averages:

- **Average WMC** = 2.6
- **Average DIT** = 0.4
- **Average NOC** = 0.5
- **Average CBO** = 2.3
- **Average RFC** = 4.6
- **Average LCOM** = 0.07

▼ Comparison with Industry Standards

The design quality evaluation in this report was conducted according to the [1]CK metrics proposed by Chidamber & Kemerer.

Metric	Threshold	Our Value	Evaluation
WMC	≤ 10	2.6	Excellent
DIT	≤ 5	0.4	Excellent
NOC	≤ 10	0.5	Excellent
CBO	≤ 5	2.3	Excellent
RFC	≤ 50	4.6	Excellent
LCOM	< 0.5	0.07	Excellent

Conclusion: All metrics are well within acceptable ranges, indicating high-quality design.

3. MOOD Metric

▼ MIF(Method Inheritance Factor)

	Class	Md(Ci)	Mi(Ci)	Ma(Ci)
1	Account	4	0	4
2	User	2	0	2
3	Device	2	0	2
4	Sensor	1	2	3
5	Camera	2	2	4
6	SmartDevice	1	2	3
7	SecurityMode	2	0	2
8	Incident	2	0	2
9	Recording	3	0	3
10	AirQualityMonitor	2	0	2
11	PowerMonitor	2	0	2
12	AutomationRule	2	0	2
13	Hub	2	0	2
14	CloudServer	3	0	3
15	Session	3	0	3

MIF = Σ Mi(Ci) / Σ Ma(Ci) = 6 / 39 ≈ 0.154

▼ CF (Coupling Factor)

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	0	0	0	1
3	0	0	0	0	0	0	0
4	0	1	0	1	0	0	0
5	0	0	1	0	1	0	0
6	0	0	1	0	0	0	0
7	0	1	0	1	0	0	0
8	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0
10	0	0	1	0	1	0	0
11	0	0	1	0	0	0	0
12	0	1	0	1	0	1	0
13	0	1	0	0	0	0	0
14	1	1	1	0	0	0	0
15	1	0	0	0	0	0	0

Total coupling relationships = 34
CF = 34 / (15^2 - 15) = 34 / 210 ≈ 0.162

vii. Who Did What

▼ Table

Minseok Jo
1. Foundational Architecture (UC 4.0/5.0): Developed the Architectural Structure Diagrams defining the modular decomposition for the Remote Access and Account (UC 4.0) and Indoor Monitoring and Device Control (UC 5.0) domains.

2. Static Modeling: Created the complete Class Diagrams for both UC 4.0 and UC 5.0, defining the static structure and relationships of critical entities (UserAccount, Session, AutomationRule, SmartDevice, etc.). 3. Responsibility Specification: Completed the detailed CRC Cards for all classes within UC 4.0 and UC 5.0, ensuring responsibilities were assigned according to Low Coupling principles (e.g., using IUserRepository interfaces). 4. Behavioral Modeling: Authored the comprehensive set of State Diagrams for critical classes in both domains, modeling the dynamic logic for account status, session lifecycle, and device operational states. 5. Design Evaluation & Metrics: Undertook the entire Design Evaluation section, which involved calculating and interpreting the Architectural Design Metrics, CK Metrics, and MOOD Metrics (MIF, CF). This work provided the quantitative proof that the final design achieved High Cohesion and Low Coupling. 6. Meeting Log Documentation: Recorded and maintained detailed meeting logs throughout the project, ensuring all decisions and progress were transparently tracked. 7. Assumption Specification: Drafted and refined all system and project-related assumptions, clarifying constraints and shared expectations to reduce ambiguity in the documentation. 8. Appendix Completion & Improvement: Compiled and enhanced the Appendix sections, making sure that all supporting materials, references, and supplementary content were well-organized and complete.
Jonghwa An
1. Authored the "How The Design Work Proceed" section, detailing the systematic analysis of the SRS, the creation of initial design artifacts (like CRC cards and UML diagrams), and the subsequent refinement process based on design metric evaluations. 2. Designed the Overall Architecture diagram within the "Architectural Structure" section to provide a high-level view of the system's components and their interactions. 3. Developed the complete set of design artifacts for the "Live Surveillance" domain, including its specific Architectural Structure diagram, all associated Class Diagrams, the full set of CRC Cards, and all State Diagrams. 4. Developed the complete set of design artifacts for the "System and User Management" domain, including its Architectural Structure diagram, all associated Class Diagrams, the full set of CRC Cards, and all State Diagrams. 5. Following the Design Evaluation, led the effort to remodel all CRC Cards across the entire project, refining their content and organizing them into a clear, standardized tabular format to improve system-wide cohesion and clarity.
Jien Lee
1. Project Foundation & Goals: Authored the Introduction and Goal sections of the SDS document, clearly defining the project's scope, vision, and core design objectives. 2. Architectural & Static Modeling (UC 1.0): Developed the foundational Architectural Structure Diagram and the complete Class Diagram for the Intelligent Security (UC 1.0) domain, thereby establishing the system's core static model and hierarchical relationships. 3. Behavioral & Responsibility Definition (text{UC 1.0): Completed the full set of CRC Cards and State Diagrams for all UC 1.0 classes, providing concrete specifications for the system's security logic and dynamic behavioral flows. 4. Document Quality Control: Undertook the critical task of document formatting and quality control, ensuring the final SDS document maintained consistency and professional standards throughout.

viii. Meeting Logs

▼ **11.05 Meeting: Task Allocation and Diagram Planning**

Category	Details
Date & Time	November 5, 2025 (Wednesday), 9PM - 10PM (1h)
Attendees	Minseok Jo, Jonghwa An, Jien Lee
Purpose	Discuss task division for functional sections, diagram creation assignments (architecture, class, CRC, state diagrams), and preparation for system overview documentation.
Discussion Summary	- Proposed dividing tasks among members: one person handles Intelligent Security, Intro, and Goal sections; another handles Live Surveillance and System/User Management; last covers Remote Access, Account Management, Indoor Monitoring, and Device Control. - Identified key diagram requirements: architecture structure diagram, class diagram, CRC cards, state diagrams. - Discussed using AI tools like StarUML connected with cloud services for easier diagram creation. - Agreed that system overview and overall architecture documentation can be drafted after diagrams are completed. - Considered some previously drafted content (dog barking sensor) and agreed to include relevant items. - UI documentation not required unless for clarity in implementation. - Task assignments and deadlines set with understanding that initial drafts may need refinement.
Decisions Made	- Task division per major functional area confirmed. - Diagram creation split by function with use of AI tools. - Overview documentation to follow diagram completion. - UI section excluded for now. - Individual efforts to be tracked via contribution logs.
Assignments	- Each team member to produce diagrams and documentation for their assigned areas. - Prepare system overview after diagram drafts. - Each member to submit individual contribution statements. - Review and integrate materials at next meeting. Minseok Jo - 4. REMOTE ACCESS AND ACCOUNT / 5. INDOOR MONITORING AND DEVICE CONTROL / how the design work proceeded / Meeting log Jonghwa An - 2. Live Surveillance, 3.System and User Management Jien Lee - 1. Intelligence Security / intro, goal
Next Meeting	Tuesday, November 12, 2025, 12:00 AM for review and preparation before submission deadline.

▼ 11.11 Meeting: Diagram Integration and Document Organization

Category	Details
Date & Time	November 11, 2025 (Tuesday), 12:00 AM – 2:00 AM (2h)
Attendees	Minseok Jo, Jonghwa An, Jien Lee
Purpose	To integrate individual diagrams into a unified format, review feedback, finalize document templates, and coordinate next steps and roles.
Discussion Summary	- Agreed to merge diagrams based on the template being developed by a team member Jonghwa An. - Feedback from prior reviews was shared and positively received. - Intro and Goal sections mostly finalized; some content may be adjusted with SRS references. - Discussed AI tool token usage and document collaboration tools (Google Docs, Notion). Notion’s PDF export and ease of use were highlighted. - Planned to postpone "Who Did What" and assumptions sections to final stages. - Made assignments for document and diagram completion with emphasis on quick turnaround using AI assistance. - Agreed on deadlines and next meeting schedule. - Discussed preferences and logistics for collaboration tools and meeting times.

Category	Details
Decisions Made	- Finalize and format diagrams according to chosen template. - Leverage AI tools for efficient document generation. - Adopt Notion or Google Docs as primary collaboration platform. - Maintain clear records of individual contributions. - Stick to agreed timeline for submission preparation.
Assignments	- Template lead to finalize format and oversee diagram integration. - Team members to complete assigned diagrams and attach to template. - All to contribute to documentation and attend next review meeting.
Next Meeting	Wednesday, November 12, 2025, 12:00AM – 2AM

▼ 11.12 Meeting: Diagram Completion Status and Document Finalization Planning

Category	Details
Date & Time	November 12, 2025 (Wednesday), 12:00 AM – 2:00 AM (2h)
Attendees	Minseok Jo, Jonghwa An, Jien Lee
Purpose	Check diagram completion status, discuss document refinement, AI tool usage, and finalize writing assignments.
Discussion Summary	- Confirmed all members completed diagrams but noted complexity requiring content review. - Discussed streamlining use case diagrams by grouping related actions. - Shared experiences with AI tools (Claude, Gemini) and token management. - Confirmed finalization of diagrams, meeting logs, appendices, and Who Did What sections. - Clarified UI documentation scope: limited to implementation if necessary. - Talked about document collaboration tools such as Google Docs and Notion, favoring online editable formats for ease. - Agreed on deadline management: diagrams and documents to be merged and formatted by next meeting. - Light discussion on subscription costs and AI capabilities. - Planned next steps on content editing and formatting tasks.
Decisions Made	- Use AI tools to accelerate diagram finalization. - Employ collaborative platforms for document management. - Maintain detailed contribution logs. - Ensure strict adherence to deadlines.
Assignments	- Individuals finalize their diagrams and documentation sections. - Document editor to integrate and format the final draft. - Team to review and polish work prior to scheduled meeting.
Next Meeting	November 13, 2025 (Thursday) at 6:00 PM for comprehensive review and submission preparation.

▼ 11.13 Meeting: Final Document Edits & Diagram Review

Category	Details
Date & Time	November 13, 2025 (Friday), 19:00 PM – November 14, 2025 2:00 AM (7 h)
Attendees	Minseok Jo, Jonghwa An, Jien Lee
Purpose	Finalize documentation and diagram edits before project submission, review roles and responsibilities, address last-minute changes and quality criteria.
Discussion Summary	- Will finish the design evaluation after reviewing the lecture; may ask team members for help on section details. - Complete appendix, assumptions, and “How the design proceeded” sections using AI support. - Clarified the glossary and terminology in the appendix, suggested starting with the assumptions. - Shared tips for exporting documents to PDF and managing collaborative files. - Reviewed diagram content and edits as assigned. - Addressed efficiency and complexity issues with AI tools (Claude, Gemini) for diagram generation. - Consolidated and simplified overlapping or inefficient items in CRC cards, class diagrams, and storage manager entities.

Category	Details
	- Checked quality metrics per instructor's guidelines (e.g., coupling, inheritance). - Reinforced division of work for overview diagrams, assumptions/appendix, and design evaluation.
Decisions Made	- Each member will finalize their part: assumptions, appendices, and evaluation. - Encourage maximum use of AI, consolidate and simplify diagrams/document structure. - Thorough review of quality metrics and removal of redundant entities. - Considered potential presenters and English-language presentation. - Confirmed all documents and diagrams will be merged before submission, clarified next project timeline.

ix. Appendix A. Glossary

▼ Glossary

A. Architectural & Design Terms

1. **MVC (Model-View-Controller)**

   A software architectural pattern dividing an application into three interconnected components: the Model (data and business logic), the View (user interface), and the Controller (workflow and input handling). This design combines MVC with Domain-Driven Design (DDD), where domain entities encapsulate business logic.

2. **Domain-Driven Design (DDD)**

   A design approach emphasizing the core business domain and logic. Domain entities (e.g., Account, Incident) enclose business rules and invariants, differing from traditional MVC Models by including behavior.

3. **Hub-Cloud Architecture**

   A distributed system architecture where SafeHome Hub manages local devices and time-sensitive operations, while CloudServer provides authentication, storage, and synchronization, supporting offline operation.

4. **Boundary Class**

   A UML stereotype representing classes that interact with external actors, handling UI presentation and user input without business logic (e.g., LoginView).

5. **Control Class**

   A UML stereotype for classes managing and orchestrating use case workflows, coordinating between Boundary and Entity classes (e.g., AuthenticationController).

6. **Entity Class**

   A UML stereotype representing classes modeling persistent business data and logic, enforcing invariants (e.g., UserAccount).

7. **Service Class**

   A UML stereotype for classes providing reusable functionality or infrastructure services (e.g., NotificationService).

8. **Cohesion (High Cohesion)**

   A principle describing the relatedness of responsibilities within a class; high cohesion implies focused, single-purpose classes.

9. **Coupling (Low Coupling)**

   A principle describing minimal dependencies between modules, facilitating modularity and flexibility.

10. **CRC (Class-Responsibility-Collaborator)**

    A method to define key classes by specifying their responsibilities and collaborators, ensuring clear, focused roles.

11. **State Diagram**

    A UML diagram modeling the dynamic states of an object and transitions triggered by events (e.g., SecurityMode state transitions).

B. System Components

1. **SafeHome Hub**

   Local edge device managing connected devices, sensor data, automation rules, and synchronization with CloudServer using protocols like Zigbee and Wi-Fi.

2. **CloudServer**

   Central cloud service offering authentication, persistent storage, synchronization, remote access, and notification delivery. It exclusively accesses the Database.

3. **Database**

   The system's persistent storage layer abstracting various database technologies, storing accounts, devices, incidents, sensor data, and automation rules.

4. **Control Panel**

   A physical touchscreen UI in the home, providing local system access and communicating with the Hub.

C. Device Types

1. **Device (Abstract)**

   The base class for all physical devices, defining common interfaces for registration, status updates, and health checks. Subclasses implement specific behaviors.

2. **Sensor**

   Wireless devices monitoring environmental conditions (e.g., motion, air quality), reporting readings to the system.

3. **Camera**

   Devices capturing video streams and recordings with features like PTZ and motion-triggered recording, storing video in cloud storage and metadata in the Database.

4. **SmartDevice**

   Controllable devices executing user or automation commands (e.g., SmartLight).

5. **AirQualitySensor**

   Specialized sensors measuring indoor air quality indicators such as CO2 levels and VOCs.

6. **PowerMeter**

   Sensors measuring instantaneous power draw and cumulative energy consumption.

D. Security Domain

1. **SecurityMode**

   Defines active sensors and response behavior with modes like Disarmed, Armed (Away), and Armed (Stay), supporting extensibility for custom modes.

2. **Incident**

   A security event triggered by sensors, containing metadata, severity, status, attached evidence, and lifecycle state.

3. **Alert**

   Notifications sent to users about incidents or system events via push, email, or SMS.

4. **Bypass**

   User action to temporarily disable a sensor during arming, applicable only for the current session.

5. **Exit Delay**

   A configurable timer enabling users to leave after arming Away mode, typically 30 seconds; not applicable in Stay mode.

6. **Entry Delay**

   A configurable timer started upon entry during Away mode, allowing users to disarm before alarm activation.

7. **Account Lockout**

   A security measure locking accounts after repeated failed login attempts, with automatic or manual recovery options.

8. **Safety Zone**

   Concept deferred for initial release; sensors assigned directly to SecurityMode configurations without zone logic.

E. Surveillance Domain

1. **Recording**

   Stored video files captured by cameras, triggered by incidents or manually, with defined retention periods.

2. **VideoStream**

   Real-time camera video streamed to user devices without default recording.

F. Monitoring & Automation Domain

1. **AirQualityMonitor**

   Domain service observing air quality sensors and triggering ventilation automation upon threshold breaches.

2. **PowerMonitor**

   Domain service recording power meter data and generating energy consumption reports.

3. **EnergyReport**

   Reports summarizing power consumption over specific periods, formatted as downloadable PDFs.

4. **AutomationRule**

   User-defined rules activating actions based on triggers and conditions, executed locally by the Hub.

5. **Schedule**

   Time-based automation rules triggering actions at specified times or days.

G. Account & User Management

1. **Account**

   Represents a customer account with attributes and authentication operations.

2. **User**

   Individuals authorized within an Account, with defined roles and permissions.

3. **Session**

   Authenticated user connection tokens managing access and lifespan for multiple devices.

4. **Two-Factor Authentication (2FA)**

   Additional security layer requiring secondary verification via authenticator apps or SMS.

H. Infrastructure & Protocols

1. **Zigbee**

   Low-power wireless protocol for battery-powered devices with mesh networking.

2. **Wi-Fi**

   High-bandwidth wireless protocol supporting video streaming devices.

3. **HTTPS (TLS 1.3)**

Encrypted communication protocol for secure data transfer between clients and servers.

4. **JWT (JSON Web Token)**

   Compact token format used for authentication, containing user and account info with digital signatures.

5. **Cloud Object Storage**

   Scalable storage for large files, storing video recordings with metadata in the database.

I. Data & Retention

1. **Aggregated Data**

   Summarized sensor data stored for long-term historical analytics with reduced volume.

2. **Audit Log**

   Immutable records of system events for compliance and security monitoring.

J. Non-Functional Requirements

1. **Latency**

   Target maximum response times for different system actions under normal network conditions.

2. **Offline Operation**

   Hub autonomous operation during internet outages with degraded remote features and event buffering.

3. **Scalability**

   System capacity limits per Hub and Account with cloud scalability for millions of users.

4. **GDPR (General Data Protection Regulation)**

   Compliance framework ensuring user data protection, consent, export, and deletion rights.

K. Design Patterns

1. **Observer Pattern**

   Behavioral pattern notifying dependent objects of state changes, promoting decoupling.

2. **Mediator Pattern**

   Centralized communication controller reducing direct dependencies among components.

3. **Template Method Pattern**

   Algorithm skeleton defined in base class with variable steps overridden in subclasses.

4. **Dependency Inversion Principle (DIP)**

   High-level modules depend on abstractions rather than concrete implementations.

L. Acronyms & Abbreviations

1. **API (Application Programming Interface)**

   Interfaces enabling software components to communicate.

2. **REST (Representational State Transfer)**

   HTTP-based web architecture for resource interaction.

3. **JSON (JavaScript Object Notation)**

   Lightweight, human-readable data interchange format.

4. **UUID (Universally Unique Identifier)**

   Globally unique 128-bit identifier for entity identification.

x. Appendix B. Reference

[1] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," in IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994, doi: 10.1109/32.295895.