스마트홈 전기관리 시스템

소프트웨어 아키텍처 명세

작성자: 이유훈

Revision History

|  |  |  |
| --- | --- | --- |
| Version | Date | Summary |
| 1 | 8/22 | Interim report 작성 |
| 2 | 9/6 | Prefinal report 작성 |
| 3 | 9/16 | Final report 작성 |

내용

[1. Business Drivers 6](#_Toc208946391)

[1.1. Business Goals 6](#_Toc208946392)

[1.1.1. Stakeholders List 6](#_Toc208946393)

[1.1.2. Business Goals 6](#_Toc208946394)

[1.2. Business Constraints 8](#_Toc208946395)

[2. System Context 9](#_Toc208946396)

[2.1. System Context Diagram 9](#_Toc208946397)

[2.2. External Entity List 9](#_Toc208946398)

[2.3. External Interface List 10](#_Toc208946399)

[3. Architectural Drivers 14](#_Toc208946400)

[3.1. Use Case Model 14](#_Toc208946401)

[3.1.1. Use Case Diagram 14](#_Toc208946402)

[3.1.2. Actor List 14](#_Toc208946403)

[3.1.3. Use Case List 14](#_Toc208946404)

[3.1.4. UC-01 전력정보 업데이트 17](#_Toc208946405)

[3.1.5. UC-02 AI 기반 요금예측 및 최적화 제안 18](#_Toc208946406)

[3.1.6. UC-03원격 기기 제어 19](#_Toc208946407)

[3.1.7. UC-04실시간 사용량 조회 20](#_Toc208946408)

[3.1.8. UC-05이상 상태 감지 및 알림 21](#_Toc208946409)

[3.2. Quality Attribute Scenario 22](#_Toc208946410)

[3.2.1. QA Scenario List 22](#_Toc208946411)

[3.2.2. QA-01 원격 제어 응답 속도 Scenario 24](#_Toc208946412)

[3.2.3. QA-02 신속한 시스템 장애 감지 및 자동 복구 Scenario 24](#_Toc208946413)

[3.2.4. QA-03 신규 IoT 프로토콜 지원 확장 Scenario 25](#_Toc208946414)

[3.2.5. QA-04 사용자 데이터 접근 제어 Scenario 25](#_Toc208946415)

[3.2.6. QA-05 사용자 증가에 따른 성능 유지 Scenario 26](#_Toc208946416)

[3.2.7. QA-06 인터넷 중단 시 로컬 기능 연속성 Scenario 26](#_Toc208946417)

[3.3. Architectural Constraint 27](#_Toc208946418)

[4. High Level Structure Design Description 28](#_Toc208946419)

[4.1. Domain Modeling 28](#_Toc208946420)

[4.1.1. Conceptual Class List 28](#_Toc208946421)

[4.1.2. Dynamic View 28](#_Toc208946422)

[4.1.3. Static View 32](#_Toc208946423)

[4.2. Quality Driven Architectural Design 32](#_Toc208946424)

[4.2.1. Exploring Architectural Options for *QA-01 원격 제어 응답 속도* 32](#_Toc208946425)

[4.2.2. Exploring Architectural Options for *QA-02 신속한 시스템 장애 감지 및 자동 복구* 38](#_Toc208946426)

[4.2.3. Exploring Architectural Options for *QA-05 사용자 증가에 따른 성능 유지* 42](#_Toc208946427)

[4.3. Component & Connector View 48](#_Toc208946428)

[4.3.1. UML Component Diagram 51](#_Toc208946429)

[4.3.2. Component List 52](#_Toc208946430)

[4.3.3. Connector List 54](#_Toc208946431)

[4.4. Deployment View 56](#_Toc208946432)

[4.4.1. Deployment Diagram 58](#_Toc208946433)

[4.4.2. Artifact Definition Diagram 64](#_Toc208946434)

[5. Detailed Component Design Description 68](#_Toc208946435)

[5.1. Design Description for *GatewayController* 68](#_Toc208946436)

[5.1.1. Overview 68](#_Toc208946437)

[5.1.2. Quality Driven Component Design 69](#_Toc208946438)

[5.1.3. Module View 74](#_Toc208946439)

[*5.2.* Design Description for *DeviceControlService* 79](#_Toc208946440)

[5.2.1. Overview 79](#_Toc208946441)

[5.2.2. Quality Driven Component Design 79](#_Toc208946442)

[5.2.3. Module View 82](#_Toc208946443)

[5.3. Design Description for *DataService* 86](#_Toc208946444)

[5.3.1. Overview 86](#_Toc208946445)

[5.3.2. Quality Driven Component Design 87](#_Toc208946446)

[5.3.3. Module View 90](#_Toc208946447)

[6. Architectural Evaluation 96](#_Toc208946448)

[*6.1.* Architectural Evaluation for *QA-01 원격 제어 응답 속도 (성능)* 96](#_Toc208946449)

[6.1.1. List of Risks 97](#_Toc208946450)

[6.1.2. List of Sensitivities 97](#_Toc208946451)

[6.1.3. List of Tradeoffs 97](#_Toc208946452)

[6.1.4. List of Nonrisks 97](#_Toc208946453)

[*6.2.* Architectural Evaluation for *QA-02 신속한 시스템 장애 감지 및 자동 복구* 97](#_Toc208946454)

[6.2.1. List of Risks 98](#_Toc208946455)

[6.2.2. List of Sensitivities 99](#_Toc208946456)

[6.2.3. List of Tradeoffs 99](#_Toc208946457)

[6.2.4. List of Nonrisks 99](#_Toc208946458)

[*6.3.* Architectural Evaluation for *QA-05 사용자 증가에 따른 성능 유지* 100](#_Toc208946459)

[6.3.1. List of Risks 101](#_Toc208946460)

[6.3.2. List of Sensitivities 101](#_Toc208946461)

[6.3.3. List of Tradeoffs 101](#_Toc208946462)

[6.3.4. List of Nonrisks 101](#_Toc208946463)

[7. Appendix 103](#_Toc208946464)

[7.1. Detailed Component Specification for C&C View 103](#_Toc208946465)

[7.1.1. Component 1 GatewayController 103](#_Toc208946466)

[7.1.2. Component 2 CustomerUI 103](#_Toc208946467)

[7.1.3. Component 3 DataService 104](#_Toc208946468)

[7.1.4. Component 4 NotificationSystem 104](#_Toc208946469)

[7.1.5. Component 5 DeviceControlService 105](#_Toc208946470)

[7.1.6. Component 6 HeartbeatMonitor 105](#_Toc208946471)

[7.1.7. Component 7 PredictionService 105](#_Toc208946472)

[7.1.1. Component 8 DeviceMonitor 106](#_Toc208946473)

[7.2. Interface specifications for C&C View 106](#_Toc208946474)

[7.2.1. IDeviceMonitor Interface Specification 106](#_Toc208946475)

[7.2.2. IGatewayService Interface Specification 106](#_Toc208946476)

[7.2.3. IMQTTBroker Interface Specification 107](#_Toc208946477)

[7.2.4. IHeartbeatMonitor Interface Specification 107](#_Toc208946478)

[7.2.5. IDeviceControlService Interface Specification 107](#_Toc208946479)

[7.2.6. IDataService Interface Specification 108](#_Toc208946480)

[7.2.7. ISlaveDatabase Interface Specification 108](#_Toc208946481)

[7.2.8. IMasterDatabase Interface Specification 108](#_Toc208946482)

[7.2.9. ILocalStorage Interface Specification 108](#_Toc208946483)

[7.2.10. ICache, IAppCache Interface Specification 109](#_Toc208946484)

[7.2.11. IPredictionService Interface Specification 109](#_Toc208946485)

# Business Drivers

본 장에서는 스마트홈 전기관리 시스템의 이해관계자, 비즈니스 목표, 비즈니스 제약사항에 대해 설명합니다. 본 시스템은 가정 내 전기 사용을 실시간으로 모니터링하고 자동 제어 및 최적화 기능을 통해 에너지 효율성을 높이며 전력 소비를 절감하는 IoT 기반 관리 시스템입니다.

## Business Goals

### Stakeholders List

|  |  |
| --- | --- |
| 이해관계자 | 역할 및 관심사항 |
| 사용자 | **[역할]** 시스템을 통해 가정 내 전력 사용량을 확인하고 가전기기를 제어하며, 에너지 절감 혜택을 받는 최종 주체  **[관심사항]** 전기 요금 절감, 원격 제어를 통한 생활의 편리성 증대, 과부하·누전 등 전기 안전사고 예방, 직관적이고 사용하기 쉬운 애플리케이션 |
| 경영진 | **[역할]** 비즈니스 전략 수립, 수익 모델 개발, 시장 경쟁력 확보 및 투자 유치  **[관심사항]** 시장 점유율 확대, 서비스 유료화 또는 데이터 기반 부가 사업을 통한 수익 창출, 경쟁사 대비 차별화된 기능(AI 최적화 등) 제공, 브랜드 이미지 제고 |
| 개발팀 | **[역할]** 시스템의 설계, 개발, 테스트 및 배포  **[관심사항]** 안정적이고 확장 가능한 시스템 아키텍처 구축, 다양한 IoT 기기 프로토콜 연동의 복잡성 해결, 실시간 데이터 처리 성능 확보, 보안 취약점 없는 견고한 시스템 개발 |
| 운영팀 | **[역할]** 배포된 전체 시스템의 생태계를 관리하고 지원  **[관심사항]** 사용자 동의 하에 수집된 익명화된 데이터 기반의 전체 시스템 문제점 분석, 무중단 시스템 운영 목표 |
| 전력 공급사 | **[역할]** 시스템에 전력 단가, 요금제, 전력 수급 상황 등 외부 정보 제공  **[관심사항]** 정확한 데이터 연동을 통한 요금 예측 정확도 확보, 국가적 에너지 수요 관리 정책과의 연계, 자사 고객에게 부가 서비스 제공 |
| 가전기기 제조사 | **[역할]** 자사 제품(스마트 가전)이 시스템과 호환되도록 기술 규격 준수 및 연동 지원  **[관심사항]** Matter, Zigbee 등 표준 IoT 프로토콜 지원, 자사 제품의 시스템 연동을 통한 스마트홈 생태계 확장, 연동 안정성 및 보안성 확보 |

### Business Goals

|  |  |  |  |
| --- | --- | --- | --- |
| 이해관계자 | 비즈니스 목표 | | |
| ID | 기술 | 중요도\* |
| 사용자 | BG-01 | **[목표]** AI 기반 요금 예측 및 최적화 제안을 통해 평균 15%의 전기 요금 절감과 타사 대비 빠른 원격 제어 기능을 통해 높은 편의성을 제공하고 과부하 및 누전 등의 이상상태 감지 및 알림 기능을 통해 차별화된 사용자 경험을 제공하며 서비스 만족도 90%를 달성한다,  **[아키텍처와의 연관성]** 사용자의 전력 사용 패턴을 분석하고, 제어 명령을 2초 이내의 지연 시간으로 IoT 기기에 전달할 수 있는 고성능 데이터 처리 및 저지연 메시징 아키텍처가 필요하다.  **[중요도 산정 근거]** 전기 요금 절감과 편의성 증대는 사용자가 서비스를 선택하고 지속적으로 사용하는 가장 핵심적인 동기이다. 사용자의 직접적인 만족도가 서비스 성공의 기반이 되므로 가장 높은 중요도를 부여한다. | 5 |
| 경영진 | BG-02 | **[목표]** 높은 사용자 만족도를 바탕으로 출시 2년 내 스마트홈 에너지 관리 시장 점유율 25%를 달성하고, 이를 이용해 데이터 기반 부가 사업을 통해 출시 3년 차부터 연간 5억 원의 추가 수익을 창출한다.  **[아키텍처와의 연관성]** 대규모 사용자 및 기기 증가에 유연하게 대응할 수 있는 확장 가능한(Scalable) 아키텍처가 요구된다. 또한, 데이터 판매 수익 모델을 위해 대규모 데이터를 안전하게 수집, 저장, 비식별화하여 분석할 수 있는 데이터 플랫폼 아키텍처가 필요하다.  **[중요도 산정 근거]** 시장 점유율 확보와 신규 수익 모델 창출은 기업의 생존과 성장을 위한 최우선 과제이다. 지속 가능한 비즈니스를 구축하는 데 있어 핵심적인 지표이므로 가장 높은 중요도를 부여한다. | 5 |
| 운영팀 | BG-03 | **[목표]** 시스템 장애 발생 시 자동 복구 메커니즘 등 장애 처리 수동 개입을 감소 시키고 연간 운영 비용 20% 절감을 달성한다. 또한 연간 시스템 가동률 99.5% 이상을 달성함으로써 안정적인 사용자 경험을 제공한다.  **[아키텍처와의 연관성]** 하나의 컴포넌트에서 발생한 장애가 다른 컴포넌트로 전파되지 않도록 격리하는 구조를 가져야 한다. 또한 문제 발생 시 자동으로 복구할 수 있는 능력을 갖추어야 한다.  **[중요도 산정 근거]** 서비스의 안정성은 사용자의 신뢰를 얻고 이탈을 방지하는 기본 조건일 뿐만 아니라 장애 대응 자동화를 통해 직접적인 운영 비용 절감 효과를 가져온다. 이는 기업의 수익성에 기여하고 장기적인 비즈니스 지속 가능성을 높인다. 다만 점유율 확보(BG-02)나 핵심 기능 제공(BG-01)보다는 후순위로 판단하여 중요도 4로 설정한다. | 4 |
| 개발팀 | BG-04 | **[목표]** 표준 프로토콜 지원 및 모듈화 설계를 통해 신규 IoT 기기 연동에 소요되는 개발 공수를 50% 단축하고, 코드 재사용성을 높여 유지보수 비용을 30% 절감한다.  **[아키텍처와의 연관성]** 새로운 통신 프로토콜을 지원하는 모듈을 추가할 때, 기존 시스템의 다른 부분에 미치는 영향을 최소화하는 확장 가능한 구조를 가져야 한다. 각 기능은 독립적으로 개발하고 배포할 수 있도록 모듈화되어야 함  **[중요도 산정 근거]** 개발 생산성과 유지보수 효율성은 장기적인 비용 경쟁력과 직결된다. 하지만 서비스 초기에는 시장에 성공적으로 안착하는 것이 더 중요하므로, 다른 목표 대비 상대적으로 낮은 중요도를 부여한다. | 3 |
| 가전기기 제조사 | BG-05 | **[목표]**  시스템과 원활하게 연동되는 IoT 가전기기를 공급하여 시장 점유율을 20% 이상 확대한다. 또한, 다양한 제조사 간 호환성을 보장하는 표준 API 지원을 통해 파트너십을 강화하고, 장기적으로 연간 매출 15% 성장률을 달성한다.  **[아키텍처와의 연관성]**  IoT 가전기기와 시스템 간의 원활한 연동을 위해 표준화된 프로토콜 및 확장 가능한 API 게이트웨이가 필요하다. 또한, 다양한 제조사의 기기가 안정적으로 연결될 수 있도록 유연한 디바이스 관리 아키텍처와 보안 인증 체계가 필수적이다.  **[중요도 산정 근거]**  가전기기 제조사와의 협력은 스마트홈 생태계의 핵심 성공 요소이다. 다양한 IoT 기기와의 호환성은 더 많은 사용자를 유치하여 시장 점유율을 확대(BG-02)하는 데 직접적으로 기여하며 표준 API를 통한 파트너십 강화는 장기적인 관점에서 안정적인 수익 모델을 구축하고 기술 생태계를 주도하는 데 필수적이다. 다만, 서비스 초기 단계에서는 플랫폼 자체의 안정성과 핵심 기능 제공(BG-01, BG-03)이 더 시급한 과제이므로, 중요도 4를 설정하였다. | 5 |

\* 중요도 << 1~5 또는 (최상, 상, 중, 하, 최하)로 구분 >>

## Business Constraints

|  |  |
| --- | --- |
| 비즈니스 제약사항 | |
| ID | 기술 |
| BC-01 | **[시간 제약]**  최초 버전은 10개월 이내에 개발 및 테스트를 완료하고 시장에 출시해야 한다.  **[아키텍처 영향]**  개발 기간 단축을 위해 검증된 오픈소스 프레임워크와 기술 스택을 우선적으로 채택한다. 초기에는 핵심 기능에 집중하고, 복잡한 AI 모델보다는 규칙 기반의 최적화 로직을 먼저 구현한 후 점진적으로 고도화하는 전략을 고려해야 한다. |
| BC-02 | **[법적 제약]**  사용자의 전력 사용 패턴 데이터는 개인정보보호법에 따라 민감 정보로 취급될 수 있으며, 모든 데이터 수집, 저장, 처리 과정에서 법규를 엄격히 준수해야 한다.  **[아키텍처 영향]**  민감 데이터의 전송 및 저장 시에는 강력한 암호화 메커니즘을 적용해야 한다. 또한, 허가된 사용자만이 역할에 맞는 데이터에 접근할 수 있도록 견고한 인증 및 권한 부여 체계를 아키텍처에 반영해야 한다. |

# System Context

## System Context Diagram

아래 다이어그램은 서비스 관점에서 '스마트홈 전기관리 시스템'과 상호작용하는 외부 개체를 나타낸다. 중앙의 '스마트홈 전기관리 시스템'은 서비스 플랫폼 전체를 의미한다. 본 시스템은 총 사용자 30만 명과, 이들이 사용하는 홈 게이트웨이 15만 개가 연결되는 대규모 서비스를 가정한다. 텍스트, 스크린샷, 폰트, 직사각형이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

## External Entity List

|  |  |  |
| --- | --- | --- |
| Name | Description | Related Stakeholder |
| 사용자 | **[역할]** 시스템을 원격으로 제어하고 데이터를 모니터링하며, 설정 변경 및 알림을 수신한다.  **[사용환경]** 인터넷에 연결된 모바일 기기(스마트폰, 태블릿)의 전용 앱 또는 웹 브라우저를 통해 시스템에 접속한다. **[전문성]** 대부분의 사용자는 일반 가정의 구성원으로, IT 기술에 대한 전문 지식이 없으며 스마트폰 앱 사용에 익숙한 수준이다. | 고객 |
| 전력 공급사 | **[역할]** 전력 요금 및 단가 정보, 정전 공지 등을 시스템에 제공하며, 시스템으로부터 데이터를 수신할 수 있다.  **[HW 사양]**   * 대규모 데이터 처리가 가능한 서버 시스템.   **[SW 사양]**   * 데이터 연동 및 전송을 위한 API를 포함한 SW. * 데이터베이스 시스템   **[품질 수준]**   * 정확성 : 요금 정보 및 공지 내용의 정확성 보장. * 가용성 : 데이터 제공 API의 24/7 운영. * 성능 : API 응답시간 500ms 이내. | 전력공급사 |
| 홈 게이트웨이 시스템 | **[역할]** 각 가정에 설치되는 물리적 장치로, 로컬 환경에서 스마트 미터 및 가전기기와 직접 통신하여 데이터를 수집하고 제어 명령을 전달한다. 또한, 인터넷을 통해 클라우드 플랫폼과 연결되어 데이터를 동기화하고 원격 명령을 수신하는 브릿지(Bridge) 역할을 수행한다.  **[HW 사양]** 데이터 처리를 위한 MCU, 통신 모듈(Zigbee, Wi-Fi 등), 계량 센서, 전원 제어 모듈 등을 포함한 임베디드 시스템이다.  **[SW 사양]** 실시간 데이터 수집/전송 및 기기 제어를 위한 펌웨어, 통신 프로토콜 스택, 데이터 암호화 기능을 포함하며 오픈소스(Home Assistant) 기반으로 구현되어 수정이 가능하다.  **[품질 수준]**   * 상호 운용성: 다양한 제조사의 IoT 기기들과의 호환성을 보장한다. * 응답성: 로컬 제어 명령 수행 지연 시간은 2초 이내이다. * 신뢰성: 데이터 수집 성공률은 99% 이상을 보장한다. | 개발팀, 운영팀 |

## External Interface List

|  |  |
| --- | --- |
| Name | Description |
| customerInterface | **[역할]** 사용자에게 전기 사용 현황 조회, 가전기기 원격 제어, 알림 수신 등 시스템의 주요 기능을 제공합니다.  **[공통 특성]**   * 암호화 방식 : SSL/TLS * 인증 방식 : OAuth2.0   1. 사용자 인증  **[입출력]:**   * **1단계 인증** * 입력 : 인증 유형(Enum) , ID(String,50B), PW(String,50B) * 출력 : 응답코드, 2단계 임시 토큰 * **2단계 인증** * 입력 : 인증 유형(Enum), 인증 코드(String,6B) * 출력 : 응답코드, 액세스 토큰(JWT)   **[특성]:**   * 주기/빈도 : 초당 15만 건 (처리량 확보 목표인 30만명 수준의 사용자가 동시에 로그인 요청 가정) * 응답 시간 : 사용자가 로그인 결과를 받기까지의 응답 시간은 2초 이내여야 함.   **[오류처리 방안]**  - ID/PW 불일치 : 오류 코드 반환 및 메시지 표시  - 인증코드 시간 초과 : 오류 코드 반환 및 메시지 표시, 재인증 유도  2. 전력 사용 현황 조회  **[입출력]:**   * 입력 : 액세스 토큰(JWT), 장치 아이디(Optional, 미지정 시 전체 사용량 조회), 조회 기간 유형(Enum), 시작일, 종료일 * 출력 : 응답 코드, 사용량 데이터(총 사용량, 기간별 상세 데이터 리스트), 전력 관련 정보(전력 요금제, 전력 단가, 공급사 공지)   **[특성]:**   * 입출력 크기 : 입력 최대 500byte / 출력 최대 300byte * 주기/빈도 : 초당 15만 건 (처리량 확보 목표인 300만명 수준의 사용자 중 5%가 동시 요청 가정) * 응답 시간 : 사용자가 결과를 조회하기까지의 응답 시간은 2초 이내여야 함.   **[오류처리 방안]**  - 조회 기간 내 데이터 없음 : 오류 코드 반환 및 메시지 표시  - 잘못된 기간 설정 : 오류 코드 반환 및 메시지 표시, 기간 재설정 요청  - 유효하지 않은 장치 아이디 : 오류 코드 반환 및 메시지 표시   * 5초 이상 응답 지연 시 : 오류 코드 반환 및 메시지 표시, 재시도 유도   - 액세스 토큰 만료/권한 없음 : 오류 코드 반환 및 메시지 표시,재로그인 유도  3. 가전기기 원격 제어 화면  **[입출력]:**   * 입력 : 액세스 토큰(JWT), 장치 아이디, 제어 명령 객체(cmdId, subId, length, values) * 출력 : 응답 코드, 기기 상태(명령 수행 후 변경된 최종상태)   **[특성]:**   * 주기/빈도 : 초당 30만 건 (처리량 확보 목표인 30만명 수준의 사용자가 동시 요청 가정) * 지연 시간 : 1초 이내로 시스템에 전달되어야 함   **[오류처리 방안]**  - 기기 오프라인/무응답 : 오류 코드 반환 및 메시지 표시  - 기기에서 명령 처리 실패 : 오류 코드 반환 및 메시지 표시  - 지원하지 않는 명령 : 오류 코드 반환 및 메시지 표시  4. 설정 화면   * **설정 조회** * 입력 : 액세스 토큰(JWT) * 출력 : 응답코드, 설정 객체(앱/문자/푸시 알림, 에너지 절감 모드 On/Off) * **설정 변경** * 입력 : 액세스 토큰(JWT) * 출력 : 응답코드, 설정 객체   **[특성]:**   * 입출력 크기 : 입력 32byte + JWT, 출력 : 32byte * 주기/빈도 : 초당 30만 건 (처리량 확보 목표인 30만명 수준의 사용자가 동시 요청 가정) * 응답 시간 : 조회 및 변경 요청 응답은 1초 내 완료되어야 함   **[오류처리 방안]**  - 유효하지 않은 값 입력(변경 시) : 오류 코드 반환 및 메시지 표시, 올바른 값 예시 가이드  - 저장/조회 실패 : 오류 코드 반환 및 메시지 표시  - 토큰 만료/권한 없음 : 오류 코드 반환 및 메시지표시, 재로그인 유도 |
| gatewayInterface | **[역할]** 각 가정에 설치된 홈 게이트웨이가 시스템과 통신하기 위한 인터페이스이다. 데이터 동기화, 원격 제어 명령 수신 등의 역할을 수행한다.  **[공통 특성]**   * 암호화 방식: SSL/TLS   1. 데이터 동기화  **[역할]**  시스템이 홈게이트 웨이에 수집한 데이터를 요청하는 역할을 수행한다. **[입출력]**   * 입력 : 인증 토큰, 사용자 식별자 * 출력 : 응답 코드, 전력 정보 및 상태 데이터(JSON)   **[특성]**   * 인증 방식 : OAuth2.0 * 주기/빈도: 초당 15만 건(처리량 확보 목표인 15만개 수준의 홈 게이트에서 동시 요청 가정)   2. 기기 제어 명령 전달 **[역할]** 시스템이 사용자의 원격 제어 요청을 받아, 홈 게이트웨이에 제어 명령을 전달하고 결과를 수신하는 역할  **[입출력]**   * 입력 : 인증 토큰, 사용자 식별자, 장치 아이디, 제어 명령 객체 * 출력 : 응답 코드, 기기 상태   **[특성]**   * 주기/빈도 : 초당 30만 건 (처리량 확보 목표인 30만명 수준의 사용자가 동시 요청 가정) * 지연 시간 : 1초 이내 * 인증 방식 : WPA2 / OOB 등 (IoT 프로토콜에 따라 다름) |
| supplierInterface | **[역할]** 시스템과 외부 개체인 전력공급사 시스템 간에 전력 사용량 데이터, 요금 정보, 긴급 공지 등을 안전하게 교환하기 위한 B2B연동 규약이다.  1. 전력 사용량 데이터 제출  **[역할]** 시스템이 수집 및 취합한 전력 사용량 데이터를 주기적으로 전력공급사 시스템에 전송하는 역할을 한다.  **[입출력]**   * **입력**: 인증 토큰(OAuth 2.0), 각 미터기별 사용량 데이터 배치(Batch) (JSON 배열) * **출력**: 응답 코드, 데이터 처리 결과(성공/실패 건수, 배치 ID)   **[특성]**   * 통신 프로토콜: HTTPS/REST API (POST 방식) * 암호화 방식 : SSL/TLS * 인증 방식: OAuth 2.0 * 주기/빈도: 매시간 정각, 1시간 분량의 데이터를 배치로 전송.   **[오류처리 방안]**   * 데이터 형식 오류: 오류 코드 반환 * 전송 실패: 네트워크 또는 서버 문제로 전송 실패 시, 지수 백오프(Exponential Backoff)를 적용하여 최대 3회 재시도한다 |

# Architectural Drivers

본 장에서는 기능 요구사항과 비기능 요구사항을 구체적으로 정의하고, 이를 바탕으로 아키텍처 설계 시 핵심적으로 고려해야 할 주요 드라이버들을 설명한다. 각 요구사항이 시스템에 미치는 영향을 분석하고, 효과적인 설계를 위한 필수 요소들을 상세히 다룬다.

## Use Case Model

### Use Case Diagram

텍스트, 도표, 스크린샷, 스케치이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### Actor List

|  |  |
| --- | --- |
| Name | Description |
| 사용자 | 시스템에 제어 명령을 내리고, 상태 정보를 조회하며, 설정을 변경하는 주체이다. |
| 홈 게이트웨이 시스템 | 각 가정에 설치되어 스마트 미터 및 가전기기와 직접 통신하며, 수집된 데이터를 클라우드 플랫폼으로 전송하고 원격 제어 명령을 수신하여 실행하는 주체이다. |
| 전력 공급자 | 시스템에 요금제, 단가 등 전력 관련 정보를 제공하고, 시스템으로부터 사용량 데이터를 전달받는 외부 시스템이다. |
| 타이머 | 시스템 내부에서 일정 주기마다 특정 동작이 실행될 수 있게끔 Triggering을 해주는 시스템이다. |

### Use Case List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Name | Description | Priority | | Business Drivers |
| BV | AI |
| UC-01 | 전력 정보 업데이트 | **[개요]** 홈 게이트웨이는 스마트 미터와 가전기기로부터 실시간 전력 사용량 정보를 수신하여 취합한다. 이 데이터는 시스템에 동기화되어 사용자의 조회 및 분석에 사용되며, 전력공급사 시스템에도 전송되어 요금 정산에 활용된다.  **[BV 설정근거]**  시스템의 모든 데이터 기반 서비스(실시간 조회, 요금예측 등)를 가능하게 하는 가장 근본적인 데이터 수집 기능이다. BG-01의 사용자 가치 제공과 BG-02의 데이터 기반 사업 모델의 필수 조건이지만 본 유스케이스 만으로는 타사 대비 차별점을 주거나 이익을 창출하는 행위가 어렵기 때문에 중요도 중으로 결정하였다.  **[AI 설정근거]**  다수의 로컬 장치(스마트 미터, 가전기기)로부터 데이터를 수집하고, 이를 원격의 두 시스템(클라우드, 전력공급사)으로 전송하는 복합적인 데이터 파이프라인 설계가 필요하다. 데이터의 정합성, 신뢰성 있는 전송을 보장해야 하므로 아키텍처 전반에 미치는 영향이 매우 크기에 중요도 최상으로 결정하였다. | 중 | 최상 | BG-01,  BG-02 |
| UC-02 | AI 기반 요금예측 및 최적화 제안 | **[개요]** 사용자의 전력 사용 패턴을 분석하고 외부 전력 공급사의 요금 정보를 결합하여 월말 예상 요금을 제공하고, 절약 가이드를 제안한다.  **[BV 설정근거]** 요금 예측 및 최적화 제안을 통해 전기요금 절감으로 사용자 만족도에 기여하며 이를 바탕으로 점유율 확보에 도움이 될 만한 Killer Feature이다. 또한 전력 사용 패턴 데이터와 생성한 AI 모델을 통해 데이터 기반 부가 사업을 도모하여 추가 수익을 창출할 수 있으므로 중요도 최상으로 설정하였다. (\*중요도가 5인 BG-01, BG-02의 비즈니스 목표와 연결됨)  **[AI 설정근거]** 사용자(웹/앱), 시스템, 전력공급사 시스템 등 거의 모든 내/외부 컴포넌트와의 복합적인 상호작용이 필요하다. 특히 시스템 내부에 대규모 데이터 파이프라인과 AI 모델 서빙 인프라를 구축해야 하므로 아키텍처에 미치는 영향이 매우 크므로 중요도 최상으로 설정하였다. | 최상 | 최상 | BG-01 |
| UC-03 | 원격 기기 제어 | **[개요]** 사용자가 시간과 장소에 구애받지 않고 앱을 통해 집 안의 가전기기를 제어하고, 스케줄을 설정하여 자동으로 동작하게 한다.  **[BV 설정근거]** '높은 편의성 제공'을 통해 사용자 만족도를 달성하는 가장 핵심적인 기능이다. 서비스의 성패를 가를 정도로 핵심적이지만 타 시스템 대비 비교우위로 만족도를 주기에는 어려운 기능이기에 중요도 상으로 설정하였다. (\*중요도가 5인 BG-01의 비즈니스 목표와 연결됨)  **[AI 설정근거]** 사용자(웹/앱)-클라우드-게이트웨이-가전기기와 같이 이어지는 End-to-End 상호작용을 고려해서 설계해야 하고 각 기기의 상태를 양방향으로 일관성 있게 동기화하는 복잡한 데이터 흐름을 설계해야한다. 그렇게 시스템의 여러 핵심 경계를 넘나드는 복합적인 상호작용을 구현해야 하므로 아키텍처 전반에 미치는 영향이 매우 크기에 중요도 최상으로 설정하였다. | 상 | 최상 | BG-01 |
| UC-04 | 실시간 사용량 조회 | **[개요]** 스마트 미터와 개별 가전기기에서 수집된 데이터를 바탕으로, 가정 전체 및 개별 기기의 실시간 전력 사용량을 웹/앱에서 그래프 형태로 시각화하여 보여준다.  **[BV 설정근거]** 사용자가 자신의 소비 패턴을 직관적으로 파악하여 능동적인 에너지 절약을 유도하고 서비스의 신뢰도를 높이는 필수 기능이지만 UC-01에 비해 만족도를 높이는 차별화 포인트로서의 중요도는 덜해 중요도 상으로 설정하였다. (\*중요도가 5인 BG-01의 비즈니스 목표와 연결됨)  **[AI 설정근거]** 스마트미터, 가전기기, 클라우드, 전력공급사로 구성되는 대규모 실시간 데이터 파이프라인 구축이 필요하며, 데이터 처리 성능에 직접적인 영향을 미치므로 중요도 최상으로 설정하였다. | 상 | 최상 | BG-01 |
| UC-05 | 이상 상태 감지 및 알림 | **[개요]**  시스템이 홈 게이트웨이 데이터 중 스마트 미터나 가전기기로부터의 데이터를 분석하여 과부하 등 이상 패턴을 감지하고, 즉시 사용자에게 푸시 알림을 발송한다.  **[BV 설정근거]** 전기 안전사고 예방이라는 명확한 부가 가치를 제공하여 사용자 신뢰를 확보하고 서비스 이탈률을 낮추는 데 기여한다. 사고 예방이라는 차별화된 사용자 경험을 제공하여 서비스 만족도 향상 및 이를 통한 시장 점유율 확보에 기여할 수 있으므로 중요도 최상으로 설정하였다.(\*중요도가 5인 BG-01의 비즈니스 목표와 연결됨)  **[AI 설정근거]** 시스템의 엣지 컴퓨팅 기능과 클라우드의 푸시 알림 시스템 간 연동이 필수적이며, 이벤트 기반 아키텍처 설계를 요구하지만 타 UC 대비 구현 난이도 및 컴포넌트 영향이 덜 하기 때문에 중요도 상으로 설정하였다. | 최상 | 상 | BG-01 |

\* BV: Business Value, AI: Architectural Impact << 최상 상 중 하 최하로 구분 >>

### UC-01 전력정보 업데이트

|  |  |
| --- | --- |
| Pre Condition | • 홈 게이트웨이의 전원이 켜져 있고 정상적으로 동작 중인 상태이다.  • 스마트 미터와 가전기기가 시스템(홈 게이트웨이)에 정상적으로 등록 및 연결된 상태이다.  • 시스템은 인터넷을 통해 홈 게이트웨이 및 전력공급사 시스템과 통신이 가능한 상태이다. |
| Post Condition | • 스마트 미터와 가전기기에서 발생한 특정 기간의 전력 사용량 데이터가 시스템에 저장되었다.  • 시스템에 저장된 데이터가 전력공급사 시스템에 성공적으로 전송되었다. |
| Basic Flow | 본 유스케이스는 타이머가 일정 주기(5초)마다 시스템에 전력 정보 업데이트를 진행하도록 Trigger하며 시작된다.  1. 타이머는 홈 게이트웨이에 전력 정보 업데이트를 하도록 Trigger한다. 2. 홈 게이트웨이는 지금까지 수신된 데이터를 취합하고 유효성을 검사한다.  3. 홈 게이트웨이는 취합된 데이터를 시스템으로 전송한다.  4. 시스템은 데이터 수신 성공을 홈 게이트웨이에 응답한다.  5. 시스템은 취합된 데이터를 전력공급사 시스템으로 전송한다.  6. 전력공급사 시스템은 데이터 수신 성공 여부를 시스템에 응답함으로써 본 유스케이스를 종료한다. |
| Alternative Flow | A1: 데이터 유효성 검증 실패     2a. Basic Flow 2단계에서, 특정 기기로부터 수신된 데이터의 형식이 유효하지 않다.     2b. 시스템은 해당 데이터를 폐기하고 오류를 로그에 기록한다.     2c. 유스케이스는 Basic Flow 3단계로 이어진다.  A2: 시스템 통신 실패     3a. Basic Flow 3단계에서, 홈 게이트웨이가 시스템으로 데이터 전송에 실패한다.     3b. 홈 게이트웨이는 전송할 데이터를 내부 큐(Queue)에 임시 저장한다.     3c. 홈 게이트웨이는 다음 전송 주기에 큐에 저장된 데이터와 함께 재전송을 시도한다.  A3: 전력공급사 통신 실패     5a. Basic Flow 5단계에서, 시스템이 전력공급사로 데이터 전송에 실패한다.     5b. 시스템은 전송할 데이터를 내부 큐에 임시 저장하고, 정해진 재시도 정책에 따라 전송을 시도한다. |

### UC-02 AI 기반 요금예측 및 최적화 제안

|  |  |
| --- | --- |
| Pre Condition | • 사용자가 시스템에 로그인하여 웹/앱의 메인 화면에 접속해 있는 상태이다.  • 시스템은 사용자의 전력 사용량 데이터를 일정 기간(예: 최소 7일) 이상 수집하여 저장하고 있는 상태이다.  • 시스템은 전력공급사로부터 최신 전력 요금제 정보를 수신하여 저장하고 있는 상태이다. |
| Post Condition | • 시스템이 제안한 요금예측 결과와 최적화 제안 내용이 저장되었다. |
| Basic Flow | 본 유스케이스는 사용자가 웹/앱에서 요금예측 기능을 요청하는 것으로 시작한다.  1. 사용자가 웹/앱의 요금예측 화면으로 진입한다.  2. 시스템은 해당 사용자의 과거 전력 사용 패턴 데이터를 조회한다.  3. 시스템은 전력공급사로부터 최신 전력 요금제 정보를 조회한다.  4. 시스템의 AI 모델이 사용 패턴과 요금 정보를 분석하여 월말 예상 요금을 계산한다.  5. 시스템의 AI 모델이 요금 절약을 위한 최적화된 기기 사용 스케줄을 제안으로 생성한다.  6. 시스템은 분석된 예상 요금과 최적화 제안을 사용자의 웹/앱 화면에 표시함으로써 본 유스케이스를 종료한다. |
| Extension Point | • 최적화 제안 실행: Basic Flow 6단계 이후, 사용자가 제안된 최적화 스케줄을 적용하기로 선택하면, 해당 스케줄을 시스템에 자동으로 설정한다. |
| Alternative Flow | A1: 분석 데이터 부족     2a. Basic Flow 2단계에서, 시스템에 축적된 사용자의 데이터가 예측 모델을 실행하기에 충분하지 않다.     2b. 시스템은 "데이터가 부족하여 예측이 어렵습니다. 며칠 후 다시 시도해 주세요." 라는 안내 메시지를 사용자에게 표시한다.     2c. 본 유스케이스를 종료한다.  A2: 최신 요금 정보 부재     3a. Basic Flow 3단계에서, 시스템에 유효한 최신 요금 정보가 존재하지 않는다.     3b. 시스템은 "전력공급사로부터 요금 정보를 가져올 수 없어 예측이 불가능합니다." 라는 안내 메시지를 사용자에게 표시하고, 운영팀에 알림을 전송한다.     3c. 본 유스케이스를 종료한다.  A3: 최적화 제안 미생성     5a. Basic Flow 5단계에서, AI 모델이 사용자의 패턴에서 의미 있는 절약 방안을 찾지 못했다.     5b. 시스템은 예상 요금만 표시하고, "현재 매우 효율적으로 사용 중입니다." 라는 메시지를 사용자에게 표시한다. |

### UC-03원격 기기 제어

|  |  |
| --- | --- |
| Pre Condition | • 사용자는 시스템에 로그인하여 제어 가능한 기기 목록을 보고 있는 상태이다.  • 제어 대상 가전기기는 시스템(홈 게이트웨이)에 정상적으로 등록되어 전원이 연결된 상태이다. |
| Post Condition | • 가전기기에 사용자의 제어 명령이 성공적으로 반영되었다.  • 가전기기의 변경된 상태가 시스템에 업데이트되었고, 사용자의 웹/앱 화면에도 반영되었다. |
| Basic Flow | 본 유스케이스는 사용자가 웹/앱에서 특정 가전기기를 제어하는 것으로 시작한다.  1. 사용자가 웹/앱에서 가전기기의 전원을 변경한다.  2. 시스템은 사용자 요청을 수신하고, 해당 사용자가 기기를 제어할 권한이 있는지 확인한다.  3. 시스템은 제어 명령을 홈 게이트웨이로 전송한다.  4. 홈 게이트웨이는 수신한 명령을 가전기기가 이해할 수 있는 프로토콜로 변환하여 전달한다.  5. 홈 게이트웨이는 가전기기로부터 수행 결과를 전달받는다. 6. 홈 게이트웨이는 가전기기의 최종 상태를 시스템에 동기화하고, 사용자의 웹/앱 화면에 전원 상태를 표시함으로써 본 유스케이스를 종료한다. |
| Extension Point | • 스케줄 기반 자동 제어: 사용자가 미리 설정한 특정 시간에 도달하면, 시스템이 주체가 되어 Basic Flow 2단계부터의 제어 로직을 자동으로 실행한다. |
| Alternative Flow | A1: 기기 오프라인     4a. Basic Flow 4단계에서, 시스템이 가전기기와 통신에 실패한다.     4b. 시스템은 "기기가 오프라인 상태입니다. 연결을 확인해주세요." 라는 오류 메시지를 사용자의 웹/앱에 표시한다.     4c. 본 유스케이스를 종료한다.  A2: 원격 서버 통신 실패     3a. Basic Flow 3단계에서, 시스템이 홈 게이트웨이와의 통신에 실패한다.     3b. 시스템은 웹/앱에 ‘통신에 실패했습니다’ 오류 메시지를 출력한다.     3c. 본 유스케이스를 종료한다. |

### UC-04실시간 사용량 조회

|  |  |
| --- | --- |
| Pre Condition | • 사용자는 시스템에 로그인하여 웹/앱의 메인 화면에 접속해 있는 상태이다.  • 시스템은 스마트 미터와 가전기기로부터 전력 사용량 데이터를 정상적으로 수집하고 있는 상태이다.  • 수집된 데이터는 시스템에 동기화되어 조회 가능한 상태이다. |
| Post Condition | • 사용자가 요청한 실시간 전력 사용량 데이터가 웹/앱 화면에 그래프 형태로 성공적으로 표시되었다. |
| Basic Flow | 본 유스케이스는 사용자가 웹/앱에서 실시간 전력 사용량 조회를 요청하는 것으로 시작한다.  1. 사용자가 웹/앱에서 '실시간 사용량' 메뉴를 선택한다.  2. 시스템은 사용자의 데이터 조회 요청을 수신하고, 데이터베이스에서 해당 사용자의 최신 사용량 데이터를 조회한다.  3. 시스템은 조회된 데이터를 웹/앱에서 그래프로 시각화하기 용이한 포맷(JSON)으로 가공한다.  4. 시스템은 가공된 데이터를 사용자의 웹/앱으로 전송한다.  5. 사용자는 웹/앱 화면에 표시된 실시간 사용량 그래프를 확인함으로써 본 유스케이스를 종료한다. |
| Extension Point | • 개별 기기별 사용량 조회: Basic Flow 1단계 이후에 사용자가 전체 사용량 그래프 화면에서 특정 가전기기를 선택하면, 해당 기기만의 상세 사용량 데이터를 조회하는 흐름이 시작된다. |
| Alternative Flow | A1: 데이터 동기화 지연     2a. Basic Flow 2단계에서, 시스템이 조회한 데이터가 현재 시각과 일정 시간 이상 차이가 난다.     2b. 시스템은 가장 최근에 동기화된 데이터를 그래프로 표시하며, "N분 전 데이터입니다."와 같이 데이터의 시점을 함께 안내한다.  A2: 조회 데이터 없음     2a. Basic Flow 2단계에서, 시스템의 데이터베이스에 해당 사용자의 데이터가 전혀 존재하지 않는다.     2b. 시스템은 "아직 수집된 전력 사용량 데이터가 없습니다." 라는 안내 메시지를 사용자에게 표시한다.     2c. 본 유스케이스를 종료한다. |

### UC-05이상 상태 감지 및 알림

|  |  |
| --- | --- |
| Pre Condition | • 시스템은 스마트미터 및 가전기기와 연결된 홈 게이트웨이로부터 실시간 전력 데이터를 정상적으로 수신하고 있는 상태이다. • 사용자는 웹/앱 설정에서 이상 상태 알림 수신에 동의한 상태이다. |
| Post Condition | • 감지된 이상 상태 정보가 시스템 로그에 기록되었다.  • 사용자의 웹/앱으로 이상 상태에 대한 알림이 성공적으로 발송되었다. |
| Basic Flow | 본 유스케이스는 타이머가 일정 주기(5초)마다 홈 게이트웨이에 이상 상태를 분석하도록 Trigger하며 시작된다.  1. 홈 게이트웨이에 스마트 미터 혹은 가전기기로부터 수신한 데이터에서 과부하 패턴(미리 정의된 임계값 초과)을 분석한다.  2. 홈 게이트웨이는 감지된 이상 상태 정보를 즉시 시스템으로 전송한다.  3. 시스템은 수신된 이상 상태 정보를 바탕으로 사용자에게 보낼 푸시 알림 메시지를 생성한다.  4. 시스템은 사용자의 기기로 알림을 발송한다.  5. 사용자가 자신의 웹/앱을 통해 푸시 알림을 수신하고 내용을 확인함으로써 본 유스케이스를 종료한다. |
| Alternative Flow | A1: 수신 데이터가 정상일 때    1a. Basic Flow 1단계에서 분석 결과가 정상이다.    1b. 알림을 발송하지 않고 본 유스케이스를 종료한다 A2: 푸시 알림 발송 실패     4a. Basic Flow 4단계에서, 시스템이 통신에 실패한다.     4b. 시스템은 SMS와 같은 2차 알림 수단으로 사용자에게 알림을 재전송한다.     4c. 모든 알림 수단이 실패할 경우, 해당 이벤트를 운영팀이 확인할 수 있도록 시스템에 위험 등급 로그를 기록한다.     4d. 본 유스케이스를 종료한다. |

## Quality Attribute Scenario

### QA Scenario List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Type | Priority | | Business Drivers |
| BV | AI |
| QA-01 | **[개요]**  원격 제어 응답 속도  **[BV 설정근거]**  빠른 응답 속도는 BG-01의 핵심 목표인 '높은 편의성 제공'과 직결되어 사용자 만족도를 결정하는 핵심 지표이므로 중요도 최상으로 설정하였다. **[AI 설정근거]**  사용자(웹/앱)에서부터 시스템(클라우드 플랫폼), 홈 게이트웨이에 이르기까지 End-to-End 통신 경로 전체의 성능 설계를 요구한다. 특히 각 단계에서의 상태를 일관성 있게 관리해야 하므로 아키텍처의 여러 경계를 넘나드는 복잡한 상호작용이 요구되기에 중요도 최상으로 설정하였다. | 성능 | 최상 | 최상 | BG-01 |
| QA-02 | **[개요]**  신속한 시스템 장애 감지 및 자동 복구 **[BV 설정근거]**  중단 없는 서비스는 사용자의 신뢰와 직결되며 BG-03의 '시스템 가동률 99.5%' 목표를 달성하기 위한 필수 조건이고 자동 복구 또한 BG-03의 운영비 20% 절감 목표를 달성하기 위한 필수조건이나 점유율 확보 등의 목표를 달성하기는 어렵기 때문에 중요도 상으로 설정하였다. **[AI 설정근거]**  장애 감지 및 자동 복구는 단일 컴포넌트가 아닌, 시스템 내 모든 핵심 서비스와 홈 게이트웨이에 걸쳐 구현되어야 하는 횡단 관심사이다. 이를 위해 모니터링 전용 컴포넌트가 각 서비스 컴포넌트와 지속적으로 상호작용해야 하며, 장애 발생 시 로드 밸런서, API 게이트웨이 등 인프라 수준의 여러 컴포넌트가 연계하여 복잡한 복구 절차를 수행해야 하므로 아키텍처 전반에 미치는 영향이 매우 크기 때문에 중요도 최상으로 설정하였다. | 가용성 | 상 | 최상 | BG-03 |
| QA-03 | **[개요]**  신규 IoT 프로토콜 지원 확장 **[BV 설정근거]**  빠르게 변화하는 시장에 민첩하게 대응하여 비용 경쟁력을 확보하는 BG-04의 핵심 목표를 만족시키고 다양한 제조사 간 호환성을 빠르게 지원함으로써 점유율 확보 및 매출 성장률을 높이는 BG-05의 핵심 목표를 만족시키기 때문에 중요도 최상으로 설정하였다. **[AI 설정근거]**  새로운 IoT 프로토콜을 지원하려면, 주로 홈 게이트웨이 내부의 프로토콜 변환을 담당하는 어댑터 컴포넌트와 가전기기 제어 컴포넌트의 수정 및 추가가 필요하다. 이 변경은 시스템(클라우드)이나 사용자 애플리케이션과의 직접적인 상호작용 변경을 최소화하는 방식으로 격리될 수 있으므로, 아키텍처 전반보다는 특정 부분(시스템, 홈 게이트웨이)에 영향을 미치기 때문에 중요도 중으로 설정하였다. | 변경용이성 | 최상 | 중 | BG-04, BG-05 |
| QA-04 | **[개요]**  사용자 데이터 접근 제어  **[BV 설정근거]**  데이터 유출 사고는 비즈니스의 존속을 위협하는 심각한 리스크이며 BC-02 법적 제약사항을 만족시켜야 하므로 중요도 최상으로 설정하였다. **[AI 설정근거]**  사용자 데이터 접근 제어는 인증 컴포넌트에서 시작하여, API 게이트웨이를 거쳐 데이터에 직접 접근하는 모든 백엔드 서비스 컴포넌트와 데이터베이스에 이르기까지 아키텍처의 여러 계층에 걸쳐 일관되게 구현되어야한다. 이처럼 다수의 핵심 컴포넌트에 걸쳐 보안 정책을 설계하고 적용해야 하므로 중요도 상을 설정하였다. | 보안 | 최상 | 상 | BC-02 |
| QA-05 | **[개요]**  사용자 증가에 따른 성능 유지 **[BV 설정근거]**  BG-02의 '시장 점유율 확대' 목표를 달성하기 위해 필수적인 비기능 요구사항으로, 비즈니스의 성장을 직접적으로 지원하므로 중요도 상으로 설정하였다. **[AI 설정근거]**  사용자 증가는 로드 밸런서, API 게이트웨이, 모든 핵심 서비스 컴포넌트, 데이터베이스, 캐시 등 사용자 요청을 처리하는 모든 경로에 직접적인 부하를 가합니다. 이 품질 목표를 달성하려면 각 컴포넌트가 수평적 확장이 가능하도록 아키텍처 전체를 구성해야 합니다. 이는 컴포넌트 설계부터 배포 전략까지 아키텍처 전반에 영향을 미치므로 중요도 최상을 설정하였다. | 확장성 | 상 | 최상 | BG-02 |
| QA-06 | **[개요]**  인터넷 중단 시 로컬 기능 연속성  **[BV 설정근거]**  인터넷 연결 상태와 무관하게 핵심 자동화 기능이 동작하여 BG-03의 '안정적인 사용자 경험'을 보장한다. 차별점이 되기는 힘드나 예기치 않은 미동작/오동작으로 불만족을 느끼기 쉬운 요소이기에 중요도 중으로 설정하였다.  **[AI 설정근거]**  홈 게이트웨이가 클라우드 플랫폼과의 연결 없이 독립적으로 동작할 수 있도록 엣지 컴퓨팅 능력을 갖추도록 강제한다. 이를 위해서는 기기 제어 등 핵심 로직 일부가 홈 게이트웨이 컴포넌트 내부에 구현되어야 하며, 로컬 상태 저장소 등 클라우드와 분리된 별도의 상호작용 흐름을 설계해야 합니다. 이는 홈 게이트웨이 아키텍처의 복잡성을 크게 증가시키므로 중요도를 최상으로 설정하였다. | 가용성 | 중 | 최상 | BG-03 |

### QA-01 원격 제어 응답 속도 Scenario

|  |  |
| --- | --- |
| QA Type | 성능 |
| Description | 사용자가 원격지에서 가전기기 제어를 요청했을 때, 시스템이 해당 명령을 2초 이내에 기기까지 전달하여 상태를 변경하는 시나리오이다. |
| Source of Stimulus | 사용자 |
| Stimulus | 가전기기 제어 요청 |
| Artifact | 스마트홈 전기관리 시스템 전체 (웹/앱, 클라우드 플랫폼, 홈 게이트웨이 시스템, 가전기기) |
| Environment | 시스템이 정상 운영 중인 상태. 클라우드 플랫폼은 전체 사용자(30만)의 5%가 동시 접속하여 API 요청을 보내고 있는 평균 부하 상태이다. |
| Response | 시스템은 제어 명령을 순차적으로 처리하여 최종적으로 가전기기의 물리적인 상태를 변경하고, 변경된 결과를 웹/앱에 업데이트하여 표시한다. |
| Response Measure | 사용자가 웹/앱에서 제어 명령을 발생시킨 시점(Stimulus)부터, 해당 기기의 변경된 상태가 웹/앱 화면에 표시될 때(Response)까지의 End-to-End Latency를 측정하여, 95% 이상의 요청이 2초 이내에 처리되어야 한다. |

### QA-02 신속한 시스템 장애 감지 및 자동 복구 Scenario

|  |  |
| --- | --- |
| QA Type | 가용성 |
| Description | 시스템을 구성하는 특정 컴포넌트에 장애가 발생했을 때, 시스템이 외부 개입 없이 스스로 이를 감지하고 1분 이내에 정상 상태로 복구하는 시나리오이다. |
| Source of Stimulus | 시스템 내부 컴포넌트 |
| Stimulus | 컴포넌트가 오류로 중단됨 |
| Artifact | 시스템 |
| Environment | 시스템이 정상 운영 중인 상태. 클라우드 플랫폼은 평균 부하 상태에서 동작하고 있으며, 홈 게이트웨이로부터 지속적으로 데이터가 수신되고 있는 상태. |
| Response | 장애가 발생한 '데이터 처리 컴포넌트'가 정상 상태로 복구된다. |
| Response Measure | 컴포넌트의 장애 발생 시점(Stimulus)부터, 해당 컴포넌트가 정상 상태로 복구되어 데이터 처리를 다시 시작할 때(Response)까지의 복구 시간을 측정한다. 이 시간은 1분 이내여야 한다. |

### QA-03 신규 IoT 프로토콜 지원 확장 Scenario

|  |  |
| --- | --- |
| QA Type | 변경용이성 |
| Description | 새로운 통신 프로토콜을 사용하는 IoT 기기를 시스템에 추가해야 하는 상황에서, 개발자가 기존 시스템에 미치는 영향을 최소화하며 2주 안에 신규 기기 연동을 완료하는 시나리오이다. |
| Source of Stimulus | 개발팀 |
| Stimulus | 새로운 프로토콜을 지원하는 제품을 시스템에 연동하라는 개발 요구사항 발생 |
| Artifact | 홈 게이트웨이 시스템 |
| Environment | 시스템을 개발 및 유지보수 하는 상황 |
| Response | 코어 로직 코드 변경 없이 모듈을 추가 |
| Response Measure | 소요 시간은 2주 이내여야 하며, 코어 로직의 코드 변경량은 0%여야 한다. |

### QA-04 사용자 데이터 접근 제어 Scenario

|  |  |
| --- | --- |
| QA Type | 보안 |
| Description | 공격자가 탈취한 인증 정보를 이용해 다른 사용자의 민감한 전력 사용량 데이터에 접근하려 할 때, 시스템이 이를 성공적으로 탐지하고 차단하는 시나리오이다. |
| Source of Stimulus | 외부 공격자 |
| Stimulus | 사용자 데이터 탈취 시도 |
| Artifact | 시스템 |
| Environment | 시스템이 정상 운영 중인 상태 |
| Response | 시스템은 요청을 거부하고, 어떠한 데이터도 공격자에게 반환되지 않는다. 해당 접근 시도는 보안 로그에 기록된다. |
| Response Measure | 권한 없는 데이터 접근 요청에 대한 차단 성공률을 측정한다. 성공률은 100%여야 한다. |

### QA-05 사용자 증가에 따른 성능 유지 Scenario

|  |  |
| --- | --- |
| QA Type | 확장성 |
| Description | 총 사용자 수가 2배로 증가하는 상황에서, 시스템이 자원을 수평적으로 확장하여 기존의 응답 속도와 처리량을 안정적으로 유지하는 시나리오이다. |
| Source of Stimulus | 다수의 사용자 |
| Stimulus | 시스템으로 유입되는 데이터 동기화 및 API 요청 트래픽이 2배로 증가 |
| Artifact | 시스템 |
| Environment | 시스템이 정상 운영 중인 상태. 시스템의 전체 부하가 점진적으로 2배 증가하는 상태. |
| Response | 시스템의 컴퓨팅 자원(서버 인스턴스, 데이터베이스 등)이 자동으로 확장된다. |
| Response Measure | 응답 속도는 기존 목표(2초 이내)를 만족해야 하며, 자원 사용률은 안정적인 임계값(80%) 미만을 유지하고, 에러 발생률은 증가하지 않아야 한다. |

### QA-06 인터넷 중단 시 로컬 기능 연속성 Scenario

|  |  |
| --- | --- |
| QA Type | 가용성 |
| Description | 홈 게이트웨이가 시스템과 통신할 수 없는 오프라인 상태가 되었을 때, 미리 설정된 자동화 규칙(스케줄 등)을 독립적으로 정상 수행하는지 확인하는 시나리오이다. |
| Source of Stimulus | 외부 네트워크 |
| Stimulus | 홈 게이트웨이와 시스템 간의 인터넷 연결 중단됨 |
| Artifact | 홈 게이트웨이 시스템 |
| Environment | 시스템이 정상 운영 중인 상태.  홈 게이트웨이에는 스케줄이 저장되어 있는 상태. |
| Response | 저장된 스케줄대로 기기를 제어한다. 이후 인터넷이 복구되면, 시스템은 오프라인 중에 수행했던 제어 이력을 시스템과 동기화한다. |
| Response Measure | 인터넷이 중단된 상태에서 사전에 설정된 스케줄의 실행 성공률은 100%여야 한다. 또한, 스케줄에 설정된 시간과 실제 명령이 실행된 시간의 오차는 1초 이내여야 한다. |

## Architectural Constraint

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Title | Description | Business Drivers |
| AC-01 | 오픈소스 프레임워크 및 기술 우선 채택 | 10개월 내 최초 버전 출시'라는 시간 제약(BC-01)을 만족시키기 위해, 시스템의 주요 컴포넌트는 검증된 오픈소스를 기반으로 구현한다. IoT 컴포넌트는 SmartThings 등 많은 IoT 허브 등에 사용되는 오픈소스인 HomeAssistant를 기반으로 구현한다. AI 요금 예측 컴포넌트는 Facebook의 오픈소스 모델인 Prophet을 우선적으로 활용하고 필요시 최소한의 추가 학습(Fine-Tuning)을 적용한다. 자체 개발은 필요한 최소 범위로 한정하여 개발 리스크와 기간을 단축한다. | BC-01 |
| AC-02 | 개인정보보호법 준수를 위한 보안 아키텍처 적용 | 사용자의 민감한 전력 사용량 데이터를 보호하기 위해(BC-02), 모든 데이터 전송 구간은 TLS 기반으로 암호화한다. 시스템에 저장되는 모든 사용자 데이터는 암호화된 상태로 보관되어야 하며, 데이터 접근은 역할 기반 접근 제어(RBAC)를 통해 엄격히 통제한다. | BC-02 |
| AC-03 | 클라우드 기반 인프라 활용 | 초기 투자 비용 절감 및 운영 비용 최적화를 위해 클라우드 인프라(AWS, Azure, Google Cloud 중 하나)를 기반으로 구축되어야 한다. 또한 서비스 확장성을 고려하여 시스템 수요 변화에 따라 컴퓨팅 리소스를 자동으로 확장 및 축소할 수 있어야한다. | BG-02,  BG-04 |

# High Level Structure Design Description

본 장에서는 3장에서 도출한 Architectural Driver들을 반영하기 위한 상위 수준의 구조 설계와 관련된 내용을 설명합니다.

## Domain Modeling

3.1장에서 산정한 UC별 BV와 AI를 종합 고려하여, 다음 3개 UC에 대한 Domain 모델링 작업을수행한다:

* **UC-02 AI 기반 요금예측 및 최적화 제안 (BV: 최상, AI: 최상)**
* **UC-03 원격 기기 제어 (BV: 상, AI: 최상)**
* **UC-05 이상 상태 감지 및 알림 (BV: 최상, AI: 상)**

### Conceptual Class List

|  |  |  |  |
| --- | --- | --- | --- |
| Conceptual Class | | | Relevant Use Cases |
| ID | Name | Type |
| CC-01 | UserInterface | boundary | UC-02, UC-03 |
| CC-02 | GatewayInterface | boundary | UC-03, UC-05 |
| CC-03 | PowerCompanyInterface | boundary | UC-02 |
| CC-04 | NotificationInterface | boundary | UC-05 |
| CC-05 | PredictionController | control | UC-02 |
| CC-06 | DeviceControlController | control | UC-03 |
| CC-07 | AnomalyDetectionController | control | UC-05 |
| CC-08 | DataManager | control | UC-02 |
| CC-09 | PowerUsageData | entity | UC-02, UC-05 |
| CC-10 | PricingPlan | entity | UC-02 |
| CC-11 | controlSchedule | entity | UC-02, UC-03 |
| CC-12 | DeviceStatus | entity | UC-03 |
| CC-13 | AnomalyLog | entity | UC-05 |
| CC-14 | UserProfile | entity | UC-02, UC-03 |
| CC-15 | PredictionModel | application logic | UC-02 |
| CC-16 | OptimizationEngine | application logic | UC-02 |
| CC-17 | AnomalyDetectionModel | application logic | UC-05 |
| CC-18 | DeviceInterface | boundary | UC-03 |
| CC-19 | GatewayController | control | UC-03, UC-05 |
| CC-20 | CommandExecutor | application logic | UC-03 |
| CC-21 | ScheduleManager | control | UC-02, UC-03 |
| CC-22 | GatewayLocalMemory | control | UC-01, UC-05 |

### Dynamic View

#### *UC-02 AI 기반 요금예측 및 최적화* Dynamic Domain Model

UC-02의 alternative flow는 단순 실패에 관한 예외 처리에 대한 것으로 아키텍쳐 관점에서 중요도가 높지 않아 가독성을 저해하지 않도록 별도로 표기하지 않는다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Seq No. | Description |
| 1단계: 요금 예측 및 최적화 제안 | |
| 1, 1.1 | 사용자가 UserInterface를 통해 요금 예측을 요청하면, PredictionController가 사용자 ID와 함께 요청을 전달받아 전체 프로세스를 시작합니다. |
| 1.2-1.5 | PredictionController는 DataManager를 통해 사용자의 과거 전력 사용량 및 프로필 정보를, PowerCompanyInterface를 통해 최신 요금제 정보를 순차적으로 조회합니다. |
| 1.6 | PredictionController는 수집된 데이터를 PredictionModel에 전달하여 월말 예상 요금(predictedFare)을 계산합니다. |
| 1.7 | 이어서 OptimizationEngine에 사용자의 데이터(powerUsageData, userProfile)를 전달하여 요금 절감을 위한 최적화 제안(optimizationSuggestion)을 생성합니다. |
| 1.8-1.9 | PredictionController는 계산된 예상 요금과 최적화 제안을 UserInterface로 전달하고, UserInterface는 이 정보를 가공하여 사용자 화면에 표시합니다. |
| 2단계: 최적화 제안 수락 및 스케줄 저장 | |
| 2 | 사용자가 제안된 최적화 스케줄을 수락하는 경우,  UserInterface를 통해 requestAcceptSuggestion 요청을 보냅니다. |
| 2.1 | UserInterface는 PredictionController에게 사용자의 제안 수락 사실을 알립니다. |
| 2.2 | PredictionController는 제안에 포함된 스케줄 정보(scheduleList)를 새로운 ScheduleManager에게 전달하여 저장을 요청합니다. ScheduleManager는 전달받은 스케줄을 controlSchedule 엔티티로 저장하여 관리합니다. |

#### *UC-03 원격 기기 제어* Dynamic Domain Model

UC-03의 alternative flow는 단순 실패에 관한 예외 처리에 대한 것으로 아키텍쳐 관점에서 중요도가 높지 않아 가독성을 저해하지 않도록 필요한 수준에서만 표기한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Seq No. | Description |
| 1단계: 제어 프로세스 시작 | |
| 2 | **(스케줄에 의한 제어)** ScheduleManager가 사전에 등록된 일정(시간, 명령)에 따라 DeviceControlController에게 processControl 메시지를 보내며 제어를 시작. |
| 1 | **(사용자에 의한 제어)** 또는, 사용자가 UserInterface를 통해 특정 기기(deviceID)에 대한 제어 명령(command)을 요청하며 제어를 시작할 수도 있다. |
| 2단계: 제어 실행 및 결과 처리 (공통) | |
| 1.2/ 2.1 | DeviceControlController는 DataManager를 통해 해당 기기에 대한 제어 권한확인 |
| 1.3 /2.2 | **(권한이 있는 경우)** DeviceControlController는 홈 게이트웨이의 GatewayController에게 원격 제어 명령을 전송하고, 최종 상태(newStatus) 반환을 기다립니다. GatewayController는 내부의 CommandExecutor를 통해 명령을 실행하고 DeviceStatus를 갱신한 후, 최종 결과를 반환 |
| 1.3A/2.1A | **(권한이 없는 경우)** DeviceControlController는 UserInterface에 권한 없음 알림(notifyAuthFailure)을 보내고, 사용자에게 오류 메시지가 표시된 후 시나리오가 종료 |

#### *UC-05 이상 상태 감지 및 알림* Dynamic Domain Model

UC-03의 alternative flow 중 단순 실패에 관한 예외 처리에 대한 것은 아키텍쳐 관점에서 중요도가 높지 않아 가독성을 저해하지 않도록 필요한 수준에서만 표기한다.

**텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.**

|  |  |
| --- | --- |
| Seq No. | Description |
| 1-1.1 | **(홈 게이트웨이 내부)** 시나리오는 타이머가 홈 게이트웨이의 GatewayController가 실시간 전력 사용량 데이터를 AnomalyDetectionModel에 전달하여 분석을 요청하도록 Trigger하면서 시작합니다. |
| 1.1.1 | GatewayController는 GatewayLocalMemory에 저장된 실시간 데이터(realtimeUsage)를 가져옵니다. |
| 1.2 | AnomalyDetectionModel은 실시간 데이터(realtimeUsage)를 이용해 과부하, 누전 등의 이상 패턴을 분석합니다. |
| 1.3 | **(이상 상태 감지 시)** 분석 결과 이상 상태(anomalyResult)가 감지되면, GatewayController는 클라우드 플랫폼의 AnomalyDetectionController에게 해당 사실을 알립니다. |
| 1.3.1 | 클라우드의 AnomalyDetectionController는 DataManager를 통해 발생한 이상 상태 이벤트의 상세 내역을 AnomalyLog 엔티티에 기록하도록 요청합니다. |
| 1.3.2 | 동시에, AnomalyDetectionController는 NotificationInterface를 호출하여 사용자에게 보낼 알림 메시지(message) 전송을 요청합니다. |
| 1.3.2.1 | NotificationInterface는 최종적으로 사용자의 기기로 푸시 알림을 발송합니다. |

### Static View

텍스트, 도표, 스크린샷, 평행이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

## Quality Driven Architectural Design

3.2장에서 산정한 QA별 BV와 AI를 종합 고려하여 다음 3개 QA에 대한 Architectural Design을 수행한다:

* **QA-01 원격 제어 응답 속도 (BV: 최상, AI: 최상)(성능)**
* **QA-02 신속한 시스템 장애 감지 및 자동 복구 (BV: 최상, AI: 최상)(가용성)**
* **QA-05 사용자 증가에 따른 성능 유지 (BV: 상, AI: 최상)(확장성)**

### Exploring Architectural Options for *QA-01 원격 제어 응답 속도*

#### Quality Driven Architectural Decision Structure

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Design Decisions for *설계 이슈1 클라우드-홈 게이트웨이 간 통신 최적화 방안*

##### ***클라우드-홈 게이트웨이 간 통신 최적화 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 원격 제어 명령은 사용자의 앱에서 출발하여 클라우드 플랫폼을 거쳐 각 가정의 홈 게이트웨이로 전달되어야함. 홈 게이트웨이는 사설망 내에 위치하므로, 클라우드 플랫폼이 게이트웨이에 직접 연결을 시작하기 어려움. 따라서 게이트웨이가 먼저 클라우드에 연결을 요청하는 방식으로 통신이 이루어져야함. |
| 이슈 정의 | 사용자의 제어 요청을 홈 게이트웨이까지 최소한의 지연으로 전달하기 위한 최적의 통신 방식은 무엇인가? |

##### **Design Options for *클라우드-홈 게이트웨이 간 통신 최적화 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Client-Server 스타일 적용 | 스마트홈 전기관리 시스템'에서 각 가정의 홈 게이트웨이가 클라이언트가 되고 클라우드 플랫폼이 서버가 되는 구조임. 홈 게이트웨이는 1초와 같은 정해진 간격으로 클라우드 서버의 특정 API 엔드포인트에 HTTP GET 요청을 보내 자신에게 온 새로운 원격 제어 명령이 있는지 확인하고, 명령이 있을 경우 수신하여 처리함. |
| Publish-Subscribe 스타일 적용 | 스마트홈 전기관리 시스템'의 클라우드 플랫폼이 Publisher 역할을, 각 홈 게이트웨이가 Subscriber 역할을 수행함. 각 홈 게이트웨이는 시스템에 등록될 때 자신의 고유 ID를 포함하는 특정 주제를 메시지 브로커에 구독함. 사용자가 앱에서 원격 제어 명령을 내리면, 클라우드 플랫폼은 해당 홈 게이트웨이의 주제로 제어 명령 메시지를 발행함. 메시지 브로커는 이 메시지를 즉시 해당 주제를 구독하고 있는 홈 게이트웨이에게 전달함. |

##### **Decision and Rationale for *클라우드-홈 게이트웨이 간 통신 최적화 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Client-Server 스타일 적용 | [구현 용이성] 표준 HTTP 기반으로 구현이 간단함. | [QA-01 성능] 주기적인 Polling 간격만큼의 지연이 필연적으로 발생하여 2초 이내 응답 목표 달성이 어려움. | Client-Server 스타일의 Polling 방식은 실시간성이 떨어져 QA-01(성능) 목표를 만족시키기 어려움. 요청 주기를 줄이면 불필요한 트래픽과 서버 부하가 급증하여 QA-05(확장성)에 악영향을 줌. |
| Publish-Subscribe 스타일 적용 (Selected) | [QA-01 성능] Publisher와 Subscriber가 메시지 브로커를 통해 비동기적으로 통신하므로 명령 전달 지연이 최소화됨. [결합도 감소] Publisher(클라우드)와 Subscriber(게이트웨이)가 서로를 직접 알 필요가 없어 시스템 유연성이 증가함. | [복잡성] 메시지 브로커라는 중간 계층이 추가되어 시스템의 복잡도가 증가함. | Publish-Subscribe 스타일은 실시간 제어라는 핵심 요구사항을 만족시키는 가장 확실한 방법 중 하나임. 아키텍처 복잡도 증가는 감수해야 할 Trade-off지만, QA-01(성능) 목표를 달성하고 Publisher와 Subscriber 간의 결합도를 낮춰 변경용이성까지 확보할 수 있으므로 이 옵션을 선택함.  \*QA-01(성능)과 QA-05(확장성)의 설계 이슈가 밀접한 관계가 있음. 규모가 확장되어도 성능을 만족해야 하기에 관련 추가 설계 이슈는 4.2.3장에서 다룸. |

#### Design Decisions for *설계 이슈2 클라우드 플랫폼 내부 처리 효율화 방안*

##### ***클라우드 플랫폼 내부 처리 효율화 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 클라우드 플랫폼은 사용자 인증, 권한 확인, 로그 기록 등 제어 명령을 전달하기 전에 여러 내부 작업을 수행해야 함. 피크 타임에는 대규모 동시 요청이 발생할 수 있음(QA-05). |
| 이슈 정의 | 다수의 동시 제어 요청이 발생했을 때, 플랫폼 내부 처리 과정이 병목이 되지 않고 빠르게 요청을 처리하여 전체 응답 시간에 미치는 영향을 최소화할 방안은 무엇인가? |

##### **Design Options for *클라우드 플랫폼 내부 처리 효율화 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Introduce Concurrency 택틱 적용 | 사용자의 원격 제어, 상태 조회 등 모든 API 요청은 클라우드 플랫폼의 API 게이트웨이를 통해 들어옴. 들어온 요청은 단일 작업 큐(FIFO)에 순서대로 쌓임. 미리 생성된 스레드 풀의 여러 워커 스레드들은 이 큐에서 작업을 하나씩 가져와 동시에 병렬로 처리함. 예를 들어, 100개의 스레드가 있다면 동시에 100개의 사용자 요청(제어, 조회 등)을 처리할 수 있음. |
| Introduce Concurency 택틱 기반의 Prioritize Events 택틱 적용 | Introduce Concurrency 택틱을 기반으로 하되, 단일 작업 큐 대신 여러 개의 우선순위 큐를 둠. 예를 들어, '원격 제어'나 '이상 상태 알림'과 같이 즉각적인 처리가 필요한 요청은 '높은 우선순위 큐'에, '단순 전력 사용량 조회'나 '로그 기록' 같은 요청은 '낮은 우선순위 큐'에 넣음. 스레드 풀의 워커 스레드들은 항상 '높은 우선순위 큐'를 먼저 확인하고, 해당 큐가 비어있을 때만 '낮은 우선순위 큐'의 작업을 처리함. |

##### **Decision and Rationale for *클라우드 플랫폼 내부 처리 효율화 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design OptionTitle | Pros | Cons | Rationale |
| Introduce Concurrency 택틱 적용 | [QA-01 성능, QA-05 확장성] 여러 요청을 병렬로 처리하므로 단일 스레드 방식에 비해 전체 처리량(Throughput)이 높고 사용자 증가에 대응하기 용이함. | [QA-01 성능] 모든 요청을 공평하게 FIFO 방식으로 처리하므로, 중요도가 낮은 다수의 요청(예: 로그 기록)이 큐에 쌓여있으면 긴급한 제어 명령의 처리가 지연될 수 있음. | Introduce Concurrency 택틱은 동시성 처리를 가능하게 하여 기본적인 성능과 확장성을 제공하지만, 모든 요청을 동일하게 취급하는 한계가 있음. 이로 인해 피크 부하 시, 긴급한 원격 제어 요청이 2초 이내 응답 목표를 만족하지 못할 위험이 큼. |
| Introduce Concurency 택틱 기반의 Prioritize Events 택틱 적용 (Selected) | [QA-01 성능, QA-05 확장성] 시스템 부하가 증가하는 상황에서도 원격 제어와 같은 핵심 기능의 응답 시간을 안정적으로 유지할 수 있음. 전체 처리량을 효율적으로 관리하여 확장성을 높임. | [복잡성] 우선순위 큐 관리 및 이벤트 처리 로직 구현으로 아키텍처의 복잡도가 증가함.  [파생 이슈]  낮은 우선순위 이벤트의 기아 상태 발생 가능성이 있음 | Prioritize Events 택틱은 이러한 한계를 극복하고 핵심 기능의 응답 시간을 안정적으로 보장해 줌. 따라서 시스템의 핵심 품질 목표인 QA-01(성능)을 확실하게 만족시키기 위해 이 옵션을 선택함.  [단점보완 – 복잡성]  표준 라이브러리(예:Java의 PriorityBlockingQueue)를 활용하여 구현 복잡도를 최소화  [파생 이슈 해결] 설계이슈 3에서 해결책 제시. |

#### Design Decisions for *설계 이슈3 낮은 우선순위 이벤트의 처리 보장 방안*

##### ***낮은 우선순위 이벤트의 처리 보장 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 설계 이슈 2에서 대규모 동시 요청 처리를 위해 ' Introduce Concurency 택틱 기반의 Prioritize Events 택틱'을 채택하기로 결정함. 이로 인해 시스템은 우선순위가 높은 이벤트(예: 원격 제어)를 먼저 처리하게 됨. |
| 이슈 정의 | 만약 우선순위가 높은 이벤트가 지속적으로 대량 유입될 경우, 우선순위가 낮은 이벤트(예: 상태 로그 기록, 단순 조회)가 무기한으로 지연되거나 전혀 처리되지 못하는 상황이 발생할 수 있음. 이러한 문제를 방지하고 시스템의 모든 기능이 최소한의 실행을 보장받을 수 있는 방안은 무엇인가? |

##### **Design Options for *낮은 우선순위 이벤트의 처리 보장 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Limit Event Response 택틱 적용 | 낮은 우선순위 큐'에 최대 크기(예: 10,000개)를 설정함. 만약 '높은 우선순위' 요청이 계속 밀려와 '낮은 우선순위 큐'가 가득 차게 되면, 이후에 들어오는 '단순 조회', '로그 기록' 등의 요청은 시스템에서 거부(Drop)하고 사용자에게 "잠시 후 다시 시도해주세요"와 같은 에러를 응답함. 이를 통해 시스템이 낮은 순위의 작업들로 인해 과부하되는 것을 방지함. |
| Schedule Resources 택틱 적용 | 작업 큐에서 작업을 가져가는 스케줄러에 '가중치 기반 공정 스케줄링' 정책을 적용함. 예를 들어, '높은 우선순위 큐'와 '낮은 우선순위 큐'의 처리 비율을 4:1로 설정함. 이렇게 하면 워커 스레드는 '높은 우선순위 큐'에서 4개의 작업을 처리한 후에는, 큐에 작업이 남아있더라도 반드시 '낮은 우선순위 큐'에서 1개의 작업을 처리함. 이를 통해 아무리 긴급한 요청이 많아도 로그 기록 같은 작업이 최소한의 실행을 보장받게 됨. |

##### **Decision and Rationale for *낮은 우선순위 이벤트의 처리 보장 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Limit Event Response 택틱 적용 | [안정성] 시스템 과부하 상황에서 낮은 순위의 작업 유입을 막아 전체 시스템의 안정성 유지에 도움 | [기아 상태] 이미 큐에 대기 중인 낮은 순위 이벤트의 실행을 보장해주지는 못하며, 지속적인 고부하 상황에서는 기아 상태가 지속 가능. | Limit Event Response는 과부하 방지에는 효과적이나, 기아 상태 문제의 근본적인 해결책이 아님. |
| Schedule Resources 택틱 적용 (Selected) | [공정성 및 처리 보장] 낮은 우선순위의 이벤트도 일정 비율로 처리될 기회를 보장받으므로 기아 상태를 근본적으로 방지할 수 있음. [예측 가능성] 모든 종류의 요청이 예측 가능한 시간 내에 처리될 것을 보장하여 시스템의 신뢰도를 높임. | [구현 복잡성] 단순한 우선순위 큐보다 정교한 스케줄러 구현이 필요하여 복잡도가 다소 증가함. | Schedule Resources 택틱은 낮은 우선순위 작업에도 최소한의 실행 기회를 보장하여 공정성을 확보하고 모든 기능이 최소한의 서비스를 제공하도록 보장함. 이는 시스템의 전반적인 신뢰성과 예측 가능성을 높이는 데 더 적합한 해결책임. 따라서 이 옵션을 선택함. |

#### Design Decisions for 설계 이슈4 *단일 장애점 제거 방안*

##### ***단일 장애점 제거 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 설계 이슈 2와 3에서 이벤트의 우선순위를 관리하고 공정하게 처리하기 위한 스케줄링 메커니즘을 도입하기로 결정함. 만약 이 스케줄러가 시스템 전체에 하나만 존재하는 중앙 집중형 컴포넌트로 구현될 경우, 이 컴포넌트가 시스템의 핵심 의존성이 됨. |
| 이슈 정의 | 중앙 스케줄러에 장애가 발생하거나 성능 병목이 생기면, 모든 App 서버의 요청 처리가 중단되어 시스템 전체의 성능 저하 및 서비스 마비를 유발함. 이러한 단일 장애점(SPOF)을 제거하고, 장애가 전파되지 않도록 격리할 수 있는 방안은 무엇인가? |

##### **Design Options for *단일 장애점 제거 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| 이중화된 중앙 스케줄러 | 스케줄러를 별도의 서비스로 구현하되, Active-Passive 또는 Active-Active 방식으로 이중화하여 구성함. 모든 App 서버는 이 중앙 스케줄러 클러스터에 작업을 요청하고, 스케줄러 중 하나에 장애가 발생하면 다른 스케줄러가 즉시 그 역할을 이어받아 처리하는 방식임. |
| 분산된 내부 스케줄러 | 중앙 스케줄러를 두지 않고, 각각의 App 서버 인스턴스가 자신만의 독립적인 스케줄러(우선순위 큐 포함)를 내장하는 방식임. 로드밸런서가 App 서버로 요청을 분산하면, 각 서버는 완전히 독립적으로 자신의 내부 스케줄러를 통해 작업을 처리함. |

##### **Decision and Rationale for *단일 장애점 제거 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| 이중화된 중앙 스케줄러 | [중앙 관리]  전체 시스템의 작업 큐 상태를 한 곳에서 모니터링하고 제어하기 용이함. | [복잡성 및 병목]  이중화된 스케줄러 간의 상태 동기화가 매우 복잡하며, 모든 요청이 중앙으로 몰리기 때문에 여전히 논리적인 병목 지점이 될 수 있음. | 중앙 스케줄러를 이중화하는 것은 SPOF를 완화할 수는 있지만, 구조적인 복잡성과 잠재적 병목 문제를 완전히 해결하지는 못함. |
| 분산된 내부 스케줄러 (Selected) | [SPOF 완벽 제거]  중앙 집중 지점이 없으므로 단일 장애점이 원천적으로 존재하지 않음.  [완벽한 확장성]  App 서버 인스턴스를 늘리면 스케줄링 처리 능력도 그에 비례하여 자동으로 함께 확장됨.  [장애 격리]  한 서버의 스케줄러에 문제가 생겨도 다른 서버에는 전혀 영향을 주지 않음. | [전역 상태 파악의 어려움]  전체 시스템에 대기 중인 작업의 총량을 한눈에 파악하기는 어려움. | 분산된 내부 스케줄러 방식은 우리가 채택한 Active Redundancy 및 Load Balancing과 완벽하게 일치하는 자연스러운 확장 모델임(4.2.3장에서 선택하였음). 각 서버가 독립적으로 동작하므로 장애가 격리되고(QA-02 가용성), 서버를 추가하는 것만으로 스케줄링 처리 용량도 함께 늘어나 확장성(QA-05)이 뛰어남. 따라서 SPOF를 가장 확실하게 제거하고 시스템의 전반적인 견고성을 높이는 이 옵션을 선택함. |

### Exploring Architectural Options for *QA-02 신속한 시스템 장애 감지 및 자동 복구*

#### Quality Driven Architectural Decision Structure

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Design Decisions for *설계 이슈1 장애 탐지 방안*

##### ***장애 탐지 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 시스템은 다수의 분산된 컴포넌트(예: 클라우드 서비스, 홈 게이트웨이)로 구성됨. 이 중 하나의 컴포넌트라도 응답하지 않거나 비정상 종료되는 장애가 발생할 수 있음. |
| 이슈 정의 | 시스템을 구성하는 컴포넌트의 장애를 1분 이내의 목표 시간 안에 신속하게 탐지할 수 있는 방안은 무엇인가? |

##### **Design Options for *장애 탐지 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Ping/Echo 택틱 적용 | 클라우드 플랫폼의 모니터링 컴포넌트가 홈 게이트웨이 또는 다른 클라우드 내부 서비스들의 상태 확인 API(Health Check API)에 주기적으로 Ping을 보냄. 대상 컴포넌트는 정상일 경우 Echo(응답)를 반환함. 만약 정해진 시간 내에 응답이 없으면 장애로 간주함. |
| Heartbeat 택틱 적용 | 홈 게이트웨이 및 클라우드의 각 서비스 컴포넌트들이 자신의 상태가 정상임을 알리기 위해 주기적으로 Heartbeat 메시지를 클라우드의 모니터링 컴포넌트로 전송함. 모니터링 컴포넌트는 특정 컴포넌트로부터 정해진 시간 동안 Heartbeat 메시지를 받지 못하면 해당 컴포넌트에 장애가 발생한 것으로 판단함. |

##### **Decision and Rationale for *장애 탐지 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Ping/Echo 택틱 적용 | [탐지 정확성] 모니터가 능동적으로 상태를 확인하므로, 대상 컴포넌트의 일시적인 네트워크 문제와 실제 장애를 구분하기에 상대적으로 용이함. | [성능 부하] 모니터링 대상이 많아질수록(예: 15만 게이트웨이) Ping 요청으로 인한 네트워크 트래픽과 모니터의 부하가 증가함. | Ping/Echo의 네트워크 부하 문제는 대규모 시스템에서 확장성을 저해할 수 있음 |
| Heartbeat 택틱 적용 (Selected) | [성능 효율성] 모니터는 수신만 하므로, 감시 대상이 많아져도 모니터링 시스템의 부하가 적음. [구현 용이성] 감시 대상은 자신의 상태만 주기적으로 보내면 되므로 로직이 간단함. | [탐지 정확성] Heartbeat 메시지 유실(예: 일시적 네트워크 끊김)과 실제 컴포넌트 장애를 구분하기 위한 추가적인 로직이 필요할 수 있음. | Heartbeat 택틱은 모니터링 시스템 입장에서 수동적으로 메시지를 기다리기만 하면 되므로, 수많은 컴포넌트(특히 홈 게이트웨이)를 동시에 감시해야 하는 우리 시스템에 더 효율적임. 따라서 리소스 효율성과 확장성을 고려하여 Heartbeat 택틱을 선택함.  [단점보완 : 탐지 정확성]  모니터 내부에 실제 컴포넌트 장애 구분 로직 추가(패턴 인식 등) |

#### Design Decisions for *설계 이슈2 장애 복구 방안*

##### ***장애 복구 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 설계 이슈 1의 장애 탐지 메커니즘을 통해 특정 컴포넌트의 장애가 감지된 상황임. |
| 이슈 정의 | 탐지된 장애에 대응하여 서비스 중단을 최소화하고, 1분 이내에 시스템을 정상 상태로 자동 복구하기 위한 방안은 무엇인가? |

##### **Design Options for *장애 복구 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Active Redundancy 택틱 적용 | 클라우드 플랫폼의 핵심 서비스(예: DeviceControlController)를 최소 2개 이상의 동일한 인스턴스로 항상 실행시켜 둠. 로드 밸런서가 이 인스턴스들로 요청을 분산함. 하나의 인스턴스에 장애가 발생하면 로드 밸런서가 이를 감지하고 트래픽을 나머지 정상 인스턴스로만 보내어 서비스 중단을 방지함. |
| Passive Redundancy 택틱 적용 | 핵심 서비스를 주(Active) 인스턴스와 보조(Standby) 인스턴스로 구성함. 평상시에는 주 인스턴스만 요청을 처리하고, 처리 과정에서 발생하는 중요 상태 변경(예: 사용자 설정 변경)을 보조 인스턴스에 지속적으로 복제함. 주 인스턴스에 장애가 발생하면, 보조 인스턴스를 주 인스턴스로 승격시켜 서비스를 재개함. |

##### **Decision and Rationale for *장애 복구 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Active Redundancy 택틱 적용 (Selected) | [QA-02 가용성] 장애 발생 시 서비스 전환(Failover)에 걸리는 시간이 거의 없어 서비스 중단이 사실상 없음. 1분 이내 복구 목표를 초과 달성함. | [비용] 모든 인스턴스가 항상 활성 상태이므로 Passive 방식보다 더 많은 컴퓨팅 자원을 소모함. | QA-02의 목표는 '신속한' 자동 복구임. Active Redundancy는 장애 발생 시 거의 즉각적인 복구를 통해 서비스 중단 시간을 최소화하므로, '1분 이내 복구'라는 목표를 가장 확실하게 만족시킴. 자원 소모가 크다는 단점은 AC-03(클라우드 기반 인프라) 제약사항에 따라 필요할 때 동적으로 자원을 할당하고 회수하는 방식으로 완화할 수 있음. 따라서 가용성을 최우선으로 고려하여 이 옵션을 선택. |
| Passive Redundancy 택틱 적용 | [비용] 보조 컴포넌트는 대기 상태이므로 Active 방식보다 자원 효율성이 높음. | [사용자 만족도]  장애 발생 후 보조 컴포넌트를 활성화하고 상태를 완전히 인계받는 데 시간이 소요되어, 짧은 서비스 중단이 발생하고 이는 사용성 저하로 이어져 만족도 저하. | Active Redundancy 택틱 대비하여 QA-02의 목표인 신속한 자동복구를 만족시키기에 지연 시간이 존재하며 비용적인 장점이 존재하지만 비용 장점이 단점에 비해 크지 않으며 Active Redundancy 택틱 또한 이 점을 보완할 수 있기 때문에 적합하지 않다고 판단하였음. |

#### Design Decisions for *설계 이슈3 다중화 비용 축소 방안*

##### ***다중화 비용 축소 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 설계 이슈 2에서 높은 가용성(QA-02)을 확보하기 위해 Active Redundancy 택틱을 채택함. 이 방식은 동일한 서비스를 수행하는 서버 인스턴스를 최소 2개 이상 동시에 운영하므로, 필연적으로 인프라 비용이 증가함. |
| 이슈 정의 | Active Redundancy를 구현하면서도 불필요한 자원 낭비를 줄이고, 장기적인 관점에서 총 소유 비용(TCO)을 최적화할 수 있는 효율적인 시스템 구조화 방안은 무엇인가? |

##### **Design Options for *다중화 비용 축소 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| 서버리스 아키텍처  적용 | 각 마이크로서비스의 기능을 더 작은 '함수(Function)' 단위로 분해하고, AWS Lambda와 같은 서버리스 플랫폼에 등록하는 방식임. 평소에는 아무것도 실행되지 않다가, 특정 요청(이벤트)이 발생했을 때만 클라우드가 자동으로 함수를 실행하고, 실행이 끝나면 즉시 종료됨. 가용성을 위한 다중화는 플랫폼이 자동으로 보장함. |
| 마이크로 서비스  아키텍처 적용 | 시스템의 각 기능이 독립적인 작은 서비스(예: Prediction Service, DeviceControl Service)로 분리되어 개발되고 배포되는 구조임. Active Redundancy를 위해 각 마이크로서비스를 필요한 만큼만 독립적으로 복제하여 운영함. |

##### **Decision and Rationale for *다중화 비용 축소 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| 서버리스 아키텍처 | [극대화된 비용 효율성]  코드가 실행되는 시간(밀리초 단위)과 횟수에 대해서만 비용을 지불하므로 유휴 비용이 전혀 발생하지 않음. | [콜드 스타트(Cold Start) 지연]  오랫동안 호출되지 않은 함수가 처음 실행될 때, 실행 환경을 준비하는 데 수 초의 지연이 발생할 수 있음. 이는 성능(QA-01)에 치명적임.  [실행 시간 제약]  플랫폼마다 최대 실행 시간이 정해져 있어, 향후 복잡한 AI 분석(UC-02) 기능이 중단될 리스크가 있음.  [상태 관리의 복잡성]  근본적으로 Stateless이므로, 여러 단계에 걸친 작업의 상태를 관리하기 위한 별도의 복잡한 설계가 필요함. | 서버리스 아키텍처는 유휴 비용을 없애준다는 점에서 매우 매력적이지만, 우리 시스템의 핵심 품질 요구사항인 성능(QA-01)과 가용성(QA-02) 측면에서 치명적인 단점을 가짐. 특히 '콜드 스타트'로 인한 지연은 '2초 이내 원격 제어 응답'이라는 우리의 핵심 성능 목표를 위협할 수 있는 중대한 리스크임. 또한, 복잡한 AI 분석 기능이 최대 실행 시간을 초과할 가능성도 무시할 수 없음. |
| 마이크로서비스 아키텍처 (Selected) | [비용 효율성]  부하가 높은 서비스(예: 원격 제어)는 3개의 인스턴스로, 부하가 낮은 서비스(예: 알림 발송)는 2개의 인스턴스로 선택적 확장이 가능하여 자원 사용을 최적화하고 비용을 절감함.  [QA-05 확장성]  각 서비스가 독립적으로 확장되므로, 시스템 전체의 부하 변화에 매우 유연하고 신속하게 대응할 수 있음. | [운영 복잡성]  다수의 서비스를 독립적으로 배포, 모니터링, 관리해야 하므로 운영 복잡도가 증가함. | 마이크로서비스 아키텍처는 부하가 많은 서비스만 선택적으로, 그리고 독립적으로 확장할 수 있어 매우 효율적임. 이는 클라우드 환경(AC-03)의 탄력성을 극대화하여 장기적인 관점에서 운영 비용을 최적화하는 가장 합리적인 방식임. 약간의 유휴비용을 감수하더라도 일관된 성능과 높은 안정성을 보장할 수 있기 때문에 이 옵션을 선택함.  \*유휴비용 감소 방안 : 오토스케일링 기능을 활용하여 평소에는 최소한의 인스턴스(2개)를 운영하다가 트래픽 급증 시 인스턴스를 늘려 비용 최소화 |

### Exploring Architectural Options for *QA-05 사용자 증가에 따른 성능 유지*

#### Quality Driven Architectural Decision Structure

텍스트, 스크린샷, 폰트, 포스트잇 노트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Design Decisions for *설계 이슈1 대규모 동시 요청 처리를 위한 부하 분산 방안*

##### ***대규모 동시 요청 처리를 위한 부하 분산 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 시스템은 초기 30만 사용자에서 2배 증가한 60만 사용자까지 지원해야 함(QA-05). 사용자 증가는 API 요청 트래픽의 선형적인 증가로 이어짐. |
| 이슈 정의 | 단일 서버 인스턴스로는 증가하는 트래픽을 감당할 수 없으므로, 다수의 서버 인스턴스로 부하를 효과적으로 분산하여 기존의 응답 속도와 처리량을 안정적으로 유지하기 위한 방안은 무엇인가? |

##### **Design Options for *대규모 동시 요청 처리를 위한 부하 분산 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Multi-tier 스타일 및 Load Balancing 적용 | Multi-tier 스타일을 기반으로, 클라이언트(사용자 앱)의 요청을 처리하는 Application Tier를 여러 개의 동일한 서버 인스턴스로 구성함. 그 앞단에 로드 밸런서(Load Balancer)를 배치하여, 사용자의 '원격 제어'나 '전력 사용량 조회'와 같은 모든 수신 트래픽을 각 서버 인스턴스에 라운드 로빈(Round-Robin) 등으로 균등하게 분배함. |
| Dispatcher 스타일 적용 | Dispatcher 스타일을 적용하여, 클라이언트와 서버 사이에 중개자인 Dispatcher 컴포넌트를 둠. 사용자의 앱(클라이언트)은 기능 요청 시 먼저 Dispatcher에게 현재 가장 부하가 적거나 응답이 빠른 Application Server의 주소를 문의함. Dispatcher로부터 서버 정보를 받은 후, 클라이언트는 해당 서버와 직접 연결하여 통신을 수행함. |

##### **Decision and Rationale for *대규모 동시 요청 처리를 위한 부하 분산 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Multi-tier 스타일 및 Load Balancing 적용 (Selected) | [단순성 및 표준성] 현대 클라우드 환경에서 가장 보편적이고 표준화된 확장 방식임(AC-03). [QA-05 확장성] 트래픽 증가에 따라 서버 인스턴스 수를 늘리는 수평적 확장이 매우 용이함. | [지능성 부족] 단순한 라운드 로빈 방식의 로드 밸런서는 개별 서버의 실제 부하 상태를 정교하게 반영하지 못할 수 있음. | Multi-tier 구조에서 로드 밸런서를 사용하는 방식은 클라우드 네이티브 환경(AC-03)에서 가장 효과적이고 검증된 확장 솔루션임. 로드 밸런서는 클라우드 플랫폼에서 고가용성 서비스로 제공되므로 안정성과 관리 용이성 측면에서 더 우수함. 따라서 이 옵션을 선택함.  [단점 보완:지능성 부족]  단순 라운드 로빈 방식에서 벗어나 부하 가중치 방식을 도입, 부하에 따라 로드 밸런싱 적용하여 보다 정교하게 반영 가능. |
| Dispatcher 스타일 적용 | [유연성] 서버의 위치나 상태에 따라 동적으로 최적의 서버를 선택하여 연결할 수 있음. | [QA-02 가용성] Dispatcher 자체가 단일 장애점이 될 수 있어, Dispatcher의 이중화가 별도로 요구됨. [복잡성] 구현 및 관리가 로드 밸런서 방식보다 복잡함. | Dispatcher 스타일은 서버 위치 투명성을 제공하는 장점이 있으나, Dispatcher 자체가 단일 장애점(Single Point of Failure)이 될 수 있으며 구현이 더 복잡함. |

#### Design Decisions for *설계 이슈2 데이터 접근 부하 증가 대응 방안*

##### ***데이터 접근 부하 증가 대응 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 사용자 수 증가는 API 요청 증가뿐만 아니라, 데이터베이스에 대한 읽기/쓰기 작업의 부하 증가로 직접 이어짐. 특히 읽기 작업(전력 사용량 조회 등)의 빈도가 쓰기 작업보다 훨씬 높을 것으로 예상됨. |
| 이슈 정의 | 데이터베이스가 병목 지점이 되어 시스템 전체 성능을 저하시키는 것을 방지하고, 증가된 데이터 접근 요청을 효율적으로 처리하기 위한 방안은 무엇인가? |

##### **Design Options for *데이터 접근 부하 증가 대응 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| 자원 증대 (Increase Resources) | Performance Tactics 중 하나인 Increase Resources를 적용함. 백엔드 데이터베이스 서버(DataTier의 하드웨어 스펙(CPU, RAM, 디스크 I/O)을 높이거나(Scaling Up), 동일 스펙의 서버 인스턴스(DataManager)를 추가하여(Scaling Out) 처리 용량을 증대한다. |
| Caching (Maintain multiple copies of data) 택틱 적용 | Performance Tactics 중 하나인 Caching을 적용함. 전력 공급사의 '요금제 정보'나 사용자의 '개인 설정'과 같이 자주 조회되지만 변경 빈도가 매우 낮은 데이터를 데이터베이스보다 훨씬 빠른 인메모리(In-memory) 캐시에 저장함. DataManager와 같은 컴포넌트는 데이터 조회 요청 시, 먼저 캐시를 확인하고 데이터가 있으면 DB에 접근하지 않고 즉시 반환하여 응답 시간을 최소화하고 DB 부하를 원천적으로 감소시킴. |

##### **Decision and Rationale for *데이터 접근 부하 증가 대응 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| 자원 증대 (Increase Resources)  택틱 | [직관적인 성능 증가 (QA-01)] 하드웨어 자원을 늘리거나 인스턴스를 추가하는 만큼 시스템의 처리 능력과 동시 요청 처리량이 증가하여 QA-01 (성능)을 직접적으로 향상함. 이는 [부하 감소]에 기여하는 기본적인 방법론임.  [단순성] 캐싱 로직 추가와 같은 소프트웨어적인 복잡성 없이 인프라 레벨에서 비교적 단순하게 적용 가능함.  [확장성 (QA-05)] Scaling Out의 경우, 인스턴스 수를 늘려 수평 확장을 통해 증가하는 부하에 유연하게 대응할 수 있음. | [높은 비용] 고성능 하드웨어 스펙은 비용이 매우 높으며, Scaling Up은 특정 시점에서 물리적 한계에 도달함.  [선형적 성능 증가의 한계] 자원 투자 대비 성능 증가가 선형적(혹은 그 이하)일 수 있으며, 특정 병목(예: DB 락 경합)은 자원 증대로 해결되지 않을 수 있음.  [낭비 가능성] 트래픽이 낮은 시간대에도 항상 증대된 자원이 유지되므로 자원 낭비가 발생할 수 있음. | 캐싱과 달리 Increase Resources는 하드웨어 투자에 비례하여 부하를 감소시키는 기본적인 방법이다. 특히 즉각적인 성능 향상과 단순성 측면에서 강점이 있지만, 비용 효율성과 비선형적인 성능 최적화에는 한계가 있다. 캐싱을 통해 읽기 부하를 1차적으로 줄인 후, 그럼에도 불구하고 발생하는 병목이나 쓰기 부하에 대해서는 Increase Resources를 통해 보완하는 것이 가장 합리적인 전략임. |
| Caching 택틱 적용 (Selected) | [QA-01 성능] 인메모리에서 직접 데이터를 반환하므로, DB 접근보다 응답 속도가 월등히 빠름. [부하 감소] DB로 전달되는 읽기 요청의 수를 크게 줄여 DB 부하를 근본적으로 감소시킴. | [데이터 일관성] 캐시의 데이터와 DB의 원본 데이터 간의 불일치가 발생할 수 있어, 이를 관리하기 위한 정교한 캐시 무효화(Cache Invalidation) 전략이 필요함. | Increase Resources는 선형적인 성능 증가를 가져오지만, 캐싱은 데이터 접근 패턴(읽기 빈도)에 따라 훨씬 더 큰 폭의 비선형적인 성능 향상을 가져올 수 있다. 특히 읽기 작업이 많은 DataService의 특성상 캐싱은 데이터베이스의 부하를 근본적으로 감소시키는 데 가장 효과적인 소프트웨어적 해결책이다. 또한, 비용 효율성 측면에서도 고가의 데이터베이스 리소스 증대보다 유리하며 Data replication(설계이슈 3)은 캐시에서 처리하지 못하는 대규모 읽기 요청을 분산 처리하고 데이터베이스 자체의 가용성과 확장성을 보장함. 이 두 가지를 조합하면 QA-01(성능), QA-02(가용성), QA-05(확장성) 요구사항을 모두 높은 수준으로 만족함. 따라서 비용효과 및 개발 기간을 종합적으로 고려하여 Caching 선택. |

#### Design Decisions for *설계 이슈3 대규모 읽기 요청 처리 방안*

##### ***대규모 읽기 요청 처리 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 사용자 수 증가는 데이터 티어의 부하, 특히 읽기 요청(전력 사용량 조회 등)의 급증으로 이어짐. |
| 이슈 정의 | 데이터베이스가 병목 지점이 되어 시스템 전체 성능을 저하시키는 것을 방지하고, 증가된 데이터 읽기 요청을 효율적으로 처리하기 위한 최적의 데이터 접근 구조는 무엇인가? |

##### **Design Options for *대규모 읽기 요청 처리 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| Data Replication | 설계이슈 2에서 선택한 Caching에 Data Replication을 더해 대규모 읽기 요청 등의 문제를 보완함. 사용자의 데이터 조회 요청을 다계층으로 처리하는 전략임.  1단계 (캐싱): DataManager는 요청을 받으면 먼저 인메모리 캐시(예: Redis)에서 데이터 존재 여부를 확인. 데이터가 있으면 DB 조회 없이 즉시 반환.  2단계 (복제 DB 조회): 캐시에 데이터가 없으면(Cache Miss), 다수의 읽기 전용 복제 데이터베이스(Slave DB) 중 하나에 조회를 요청함. 조회된 데이터는 캐시에 저장한 후 사용자에게 반환.  쓰기/수정 요청은 항상 주 데이터베이스(Master DB)로 전달되며, 변경된 내용은 모든 Slave DB로 자동 복제됨. |
| Database Sharding 적용 | 데이터를 특정 규칙(예: 사용자 ID의 해시값)에 따라 여러 개의 독립된 데이터베이스(Shard)에 수평적으로 분할하여 저장하는 방식임. 특정 사용자의 데이터 조회 요청은 해당 데이터가 저장된 특정 Shard로만 전달되므로, 전체 읽기/쓰기 부하가 여러 서버로 분산됨. |

##### **Decision and Rationale for *대규모 읽기 요청 처리 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Data Replication(Selected) | [QA-01 성능] Caching을 통해 가장 빈번한 요청을 ms 단위의 초저지연으로 처리함. [QA-05 확장성] 읽기 부하가 증가할 때 캐시 노드와 Slave DB 인스턴스를 각각 독립적으로 수평 확장할 수 있음. [QA-02 가용성] Master-Slave 구조는 DB 장애 시 Failover를 지원하여 데이터 티어의 가용성을 보장함. | [데이터 일관성] Master에서 Slave로 데이터가 복제되는 데 약간의 지연(Replication Lag)이 발생할 수 있음. | 대규모 읽기 요청을 처리하기 위해 단일 택틱만으로는 한계가 명확함. Caching은 가장 빠른 응답 속도를 제공하지만 모든 데이터를 담을 수 없고, Data Replication은 DB의 읽기 부하를 분산하지만 캐시만큼 빠르지 않음. 따라서 이 두 택틱을 조합하는 것은 선택이 아닌 필수적인 설계 결정임.  우리 시스템은 쓰기보다 읽기 작업이 훨씬 빈번한 읽기 중심(Read-heavy) 워크로드를 가짐. Data Replication은 이러한 시나리오에서 읽기 부하를 분산시키는 데 가장 간단하고 효과적인 방식임.  현재 우리의 요구사항은 읽기 성능 확장이 핵심이므로, 더 간단하고 명확하며 가용성(QA-02) 확보에도 유리한 Data Replication 택틱을 선택하는 것이 더 합리적임. |
| Database Sharding 적용 | [최고 수준의 확장성] 읽기뿐만 아니라 쓰기 부하까지 여러 서버로 분산시키므로, 거의 무한에 가까운 수평적 확장이 가능함. | [구현 복잡성] 데이터를 분산하고 조회하는 로직, 여러 Shard에 걸친 트랜잭션 처리 등이 매우 복잡하여 개발 및 운영 난이도가 높음. | Database Sharding은 쓰기 부하까지 분산시키는 강력한 기술이지만, 구현과 운영이 매우 복잡하고 한 번 적용하면 되돌리기 어려움.  우리 시스템의 주된 부하는 읽기이기 때문에 쓰기 부하까지 줄이기 위해 시스템의 복잡도를 더 높일 필요가 없음. |

#### Design Decisions for *설계 이슈4 로드 밸런싱 지능성 보완 방안*

##### ***로드 밸런싱 지능성 보완 방안***

|  |  |
| --- | --- |
| 배경 및 가정 | 설계 이슈 1에서 대규모 트래픽 분산을 위해 로드 밸런서를 도입하기로 함. 하지만 가장 단순한 라운드 로빈(Round Robin) 방식의 로드 밸런서는 각 서버의 실제 부하 상태나 사용자의 세션 연속성을 고려하지 않고 순차적으로 요청을 분배함. |
| 이슈 정의 | 일부 서버에 계산량이 많은 요청이 몰려 과부하가 걸리거나, 사용자의 요청이 매번 다른 서버로 전달되어 세션 정보가 유실되는 비효율을 방지해야 함. 이를 위해, 서버의 실제 상태와 사용자의 세션 연속성을 고려하여 트래픽을 보다 지능적으로 분배할 방안은 무엇인가? |

##### **Design Options for *로드 밸런싱 지능성 보완 방안***

|  |  |
| --- | --- |
| Design Options | |
| Title | Description |
| 중앙 세션 저장소(Centralized Session Store) 적용 | 로드 밸런서는 Least Connections와 같은 순수 부하 분산 알고리즘에 따라 자유롭게 요청을 분배함. 대신, 모든 App 서버 인스턴스들은 세션 데이터를 자신의 메모리가 아닌, 별도의 중앙 세션 저장소(예: In-memory Cache인 Redis)에 저장하고 조회함. |
| Sticky Session (고정 세션) 적용 | 특정 사용자의 첫 번째 요청을 처리한 서버가 이후의 모든 요청도 계속해서 처리하도록 로드 밸런서가 세션을 고정하는 방식임. 이는 사용자의 세션 정보(예: 로그인 상태, 장바구니)를 각 서버의 메모리에 저장하고 사용할 때 유용하며, 매번 다른 서버로 요청이 전달되어 발생하는 데이터 불일치나 재인증 문제를 방지함. |

##### **Decision and Rationale for *로드 밸런싱 지능성 보완 방안***

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| 중앙 세션 저장소 적용 | [가용성, 세션 일관성]  특정 App 서버에 장애가 발생해도, 다른 서버가 중앙 저장소에서 세션 데이터를 이어받아 처리할 수 있어 세션 유실이 없음.  [부하 분산]  로드 밸런서가 자유롭게 트래픽을 분산할 수 있어, 서버 간의 부하를 더 균등하게 유지할 수 있음. | [성능]  세션 데이터를 조회할 때마다 외부 저장소와의 네트워크 통신이 발생하여, Sticky Session 방식보다 응답 지연이 발생함.  [복잡성 및 비용] 세션 관리를 위한 별도의 중앙 저장소(Redis 등)를 구축하고 운영해야 하므로 아키텍처가 더 복잡해지고 비용이 추가됨. | 중앙 세션 저장소 방식은 서버 장애 시에도 세션이 유지되는 등 가용성 측면에서 큰 장점이 있지만, 모든 세션 조회 시마다 외부 캐시 서버와의 네트워크 통신이 추가되어 응답 지연이 발생할 수 있음. 우리 시스템의 핵심 품질 목표 중 하나는 빠른 응답 속도(QA-01)이기 때문에 부적합. |
| Sticky Session (고정 세션) 적용 (Selected) | [세션 일관성]  사용자의 모든 요청이 동일한 서버에서 처리되므로, 서버 메모리에 저장된 세션 정보를 안정적으로 유지하고 사용할 수 있음.  [성능 개선]  세션 정보를 매번 중앙 저장소에서 조회할 필요가 없어 응답 속도가 향상됨. | [부하 불균형 가능성]  특정 사용자가 계산량이 많은 요청을 지속적으로 보낼 경우, 해당 사용자가 연결된 서버에만 부하가 집중될 수 있음. | Sticky Session은 사용자의 요청을 일관되게 동일한 서버로 보내주어 서버 측에서 사용자 상태를 효율적으로 관리할 수 있게 함. 이는 불필요한 데이터베이스 조회나 재인증 과정을 줄여 성능(QA-01)을 향상시키고, 사용자 경험을 개선하는 데 더 유리함. 따라서 사용자 세션의 안정적인 관리를 위해 이 옵션을 선택함.  [단점 보완 :부하 불균형 가능성]  단기간에 과도한 트래픽 발생 사용자를 필터링. 일정 수준 넘어가면 LeastConnection 방식을 부분적으로 차용해 서버부하 분산. |

## Component & Connector View

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design Option Title | Reflection in the C&C View |
| QA-01  (성능)  원격 제어 응답 속도 | Publish-Subscribe 스타일 | [반영 방식] Pub-Sub (MQTT)라는 이름의 커넥터로 명시적으로 표현됨. 이 커넥터는 Cloud Service와 Home Gateway 및 CustomerUI를 연결하는 비동기 통신 경로를 나타냄.  [만족 QA 부분] 사용자의 제어 요청 후 2초 이내에 앱 화면에 결과가 표시되어야 하는 End-to-End Latency 최소화 요구사항을 만족시킴. Push 방식으로 서버가 클라이언트에 즉시 결과를 전달하여 불필요한 지연을 제거함. |
| Schedule Resource 택틱 | [반영 방식] TaskScheduler라는 process 컴포넌트로 표현됨. 이 컴포넌트는 Cloud Service(DataService, DeviceControlService, PredictionService) 내부에서 요청을 우선순위에 따라 처리하는 로직을 담당함을 의미함.  동일한 기능을 하는 여러 개의 App 서버 마다 TaskScheduler가 독립적으로 위치함에 따라 TaskScheduler가 단일 장애점이 되지 않도록 함.  [만족 QA 부분] 평균 부하 상태에서 제어 명령과 같은 긴급한 요청이 다른 요청 때문에 지연되지 않도록 보장함. 이를 통해 2초 이내 응답(Response Measure)이라는 성능 목표를 안정적으로 달성하도록 지원함. |
| Introduce Concurency 택틱 기반의 Prioritize Events 택틱 적용 |
| 분산된 내부 스케줄러 |
| QA-02  (가용성)  신속한 시스템 장애 감지 및 자동 복구 | Heartbeat 택틱 | [반영 방식] Heartbeat Monitor 컴포넌트와, 다른 서비스들로부터 이 컴포넌트로 향하는 gRPC 연결로 명확하게 표현됨.  [만족 QA 부분] 컴포넌트 오류가 발생했을 때 이를 신속하게 탐지하는 장애 감지 요구사항을 만족시킴. 이는 1분 이내 복구의 선결 조건임. |
| Active Redundancy 택틱 | [반영 방식] LoadBalancer 커넥터와 다중화된 Cloud Service 인스턴스들의 조합으로 표현됨. 이 구조 자체가 Active Redundancy 택틱의 구현을 의미함.  [만족 QA 부분] 장애 발생 시 서비스 중단 없이 정상 인스턴스로 즉시 요청을 전환하여 자동 복구를 수행함. 이를 통해 1분 이내 복구 및 99.5% 가동률 목표를 달성함. |
| 마이크로 서비스 아키텍처 | [반영방식] 각 기능을 서비스(DataService,NotificationService…) 단위로 분해하여 나타냄. 시스템이 서버 단위로 확장도 가능하지만 부하에 따라 서비스 단위로도 확장 가능함을 나타냄.  [만족 QA 부분] 시스템 장애 발생 시 장애가 발생한 서비스를 Fail Over를 통해 서비스의 가용성(QA-02)를 만족시킬 수 있으며 사용자 증가는 물론 특정 서비스 이용에도 더 유연하게 대응, 서비스의 확장성(QA-06)을 만족시킴과 동시에 비용적인 측면에도 유리. |
| QA-05  (확장성)  사용자 증가에 따른 성능 유지 | Multi-tier 스타일 및 로드밸런싱 | [반영 방식] 다이어그램 전체가 User Space, Cloud Platform, Home Space라는 논리적 계층으로 분리되어 있으며, LoadBalancer가 명시적으로 표현됨.  [만족 QA 부분] 사용자 트래픽이 2배로 증가할 때, LoadBalancer를 통해 Application Tier에 해당하는 Cloud Service 인스턴스 수를 유연하게 늘려 대응할 수 있는 수평적 확장성을 보장함. |
| Caching 택틱 | [반영 방식] DataTier 내부에 In-memory Cache라는 data store 컴포넌트로 명확하게 표현됨. DataService는 DB 접근 전에 이 캐시를 먼저 조회함. Cache Miss의 경우 읽기 전용 DB인 SlaveDB를 조회함.  [만족 QA 부분] 사용자 증가로 인한 데이터 조회 부하를 캐시에서 흡수하여 데이터베이스의 병목 현상을 방지함. 이를 통해 트래픽이 증가하는 환경에서도 기존의 응답 속도와 처리량을 안정적으로 유지할 수 있음. |
| Sticky Session (고정 세션) | [반영 방식] C&C 뷰에서는 다이어그램에 직접적으로 표현되지는 않지만, 로드 밸런서와 서버가 사용자 세션 정보를 가지고 있고 동일 사용자가 지속 요청 시 로드 밸런서가 서버를 고정하는 방식임.  [만족 QA 부분] 사용자의 연속적인 요청이 동일한 서버로 전달되도록 보장하여, 세션 데이터의 일관성을 유지하고 불필요한 데이터 조회를 줄여 응답 속도를 안정적으로 유지하는 데 기여함. |
| Data Replication(Master/Slave)  택틱 | [반영 방식] DataTier가 Master DB와 Slave DB라는 별도의 data store 컴포넌트로 분리되어 표현됨. DataService는 쓰기 요청은 Master로, 읽기 요청은 Slave로 보내는 흐름을 통해 이 택틱의 구현을 나타냄. [만족 QA 부분] Cache Miss가 발생한 대규모 읽기 요청을 여러 Slave DB로 분산 처리하여 데이터베이스 레벨에서의 읽기 성능과 확장성을 보장함. 또한 Master DB 장애 시 Slave를 승격시켜 데이터 서비스의 가용성(QA-02)도 함께 만족시킴. |

### UML Component Diagram

텍스트, 스크린샷, 디스플레이, 도표이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

시스템은 Multi-Tier 스타일을 적용함에 따라 DataTier와 ApplicationTier로 나누어 표현하였다. 이러한 계층 분리는 각 영역의 역할을 명확히 하고 서로에게 미치는 영향을 최소화하여 시스템의 변경 용이성과 확장성을 높인다.

Application Tier는 사용자의 요청을 직접 처리하는 계층으로 마이크로 서비스 아키텍처가 적용됨에 따라 PredictionService, DeviceControlService 등 각 기능을 독립적인 컴포넌트로 분리했다.

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.TaskScheduler는 단일장애점이 되지 않기 위해 각 서비스(DataService, PredictionService, DeviceControlService)마다 개별적으로 위치하여 동작한다. 예를 들어, 사용자의 원격 제어 요청은 높은 우선순위를 갖고 DeviceControl Service의 TaskScheduler에 의해 즉시 처리된다. 이 과정에서 필요한 데이터 조회 요청은 DataService로 전달되는데, 이 요청은 상대적으로 낮은 우선순위를 가질 수 있다. 이는 시스템 부하가 높을 때도 핵심 기능(기기 제어)의 빠른 응답 속도를 보장한다. LoadBalancer 커넥터가 여러 서비스 인스턴스로 요청을 분산하는 구조를 통해 일부 서비스에 장애가 발생해도 중단없이 서비스를 제공하고 트래픽 증가 시 서버를 유연하게 늘릴 수 있도록 하여 Active Redundancy & Load Balancing 을 적용하여 설계했다. DataTier는 대규모 데이터를 안정적으로 처리할 수 있도록 설계되었다. DB를 쓰기 전용(Master)와 읽기 전용(Slave)로 분리하고 자주 찾는 데이터를 DB까지 가지 않고 Cache 메모리에 저장하여 매우 빠르게 제공할 수 있도록 설계했다. Master 장애시에도 Slave가 그 역할을 대신하여 가용성을 보장한다. Pub-Sub (MQTT) 커넥터는 클라우드, 홈 게이트웨이, 클라이언트 간의 실시간 통신을 담당한다. 클라우드에서 홈 게이트웨이로 제어 명령을 보내고, 홈 게이트웨이와 클라우드에서 클라이언트로 최종 처리 결과를 전달하는 양방향 비동기 통신 구조를 통해, 사용자는 2초 이내에 빠른 피드백을 받을 수 있다. 또한 홈 게이트웨이 내부에 로컬 메모리를 두어 인터넷 중단 시에도 로컬 기능 연속성을 보장한다. 보안성(QA-04)과 개인정보보호법 준수(AC-02)를 위해 외부와의 모든 통신에는 암호화가 적용된 HTTPS를 사용하였다. 클라우드 내부 서비스 간의 빠른 통신을 통한 성능(QA-01) 확보를 위해, 경량의 바이너리 프로토콜인 gRPC를 채택하였다. 원격 제어의 실시간 응답성을 위한 성능(QA-01)과 향후 다양한 디바이스 추가를 고려한 변경용이성(QA-03)을 만족시키기 위해 MQTT 기반의 Publish-Subscribe 통신 방식을 사용하였다. Connection Pool과 함께 JDBC를 사용하여 데이터베이스 접근 오버헤드를 최소화함으로써 성능(QA-01)과 대규모 요청 처리를 위한 확장성(QA-05)을 지원하도록 설계하였다.

### Component List

|  |  |  |  |
| --- | --- | --- | --- |
| Component  Name | Component  Kind | Property Description | Relevant ADs |
| CustomerUI | process | **[기능 요구사항]** 사용자에게 시스템의 모든 기능을 제공하는 인터페이스 역할을 수행함(UC-01~05). **[품질 요구사항]** (QA-01 성능 부분만족) Pub/Sub 커넥터를 통해 제어 결과를 비동기적으로 수신하여 사용자에게 신속하게 피드백을 제공함으로써 응답 속도 향상에 기여함. | UC-01~05 QA-01 |
| Prediction Service | process | **[기능 요구사항]** AI 기반 요금 예측 및 최적화 제안(UC-02) 로직을 수행하며, 내부적으로 OptimizationEngine과 PredictionModel을 포함함. **[품질 요구사항]** (QA-02, QA-05 완전만족) Active Redundancy 및 수평 확장을 통해 가용성과 확장성을 보장함.  **[제약사항]** (AC-01 오픈소스 완전만족) Facebook의 오픈소스 모델인 Prophet을 활용하여 제약사항을 만족함. | UC-02 QA-02, QA-05,  AC-01, AC-03 |
| DeviceControl Service | process | **[기능 요구사항]** 원격 기기 제어(UC-03), 이상상태 감지 및 알림(UC-05) 및 스케줄 실행 요청을 처리하는 핵심 비즈니스 로직을 수행함. **[품질 요구사항]** (QA-01 성능 완전만족) 내부 TaskScheduler를 통해 긴급한 제어 명령을 우선 처리함. (QA-02 부분만족, QA-05 완전만족) Active Redundancy 및 수평 확장을 통해 가용성과 확장성을 보장함. 이와 같은 인스턴스 레벨의 상위 장애 이외에도 일시적인 지연 등으로 인한 장애에도 대응할 수 있도록 상세 설계가 필요함.  **[제약사항]** (AC-03, 클라우드 기반 완전만족) 클라우드 기반의 MQTT Broker를 통해 홈 게이트웨이와 통신하여 제약사항을 만족함. | UC-03,  UC-05, QA-01, QA-02, QA-05, AC-03 |
| TaskScheduler | process | **[기능 요구사항]** ApplicationTier로 유입되는 요청들을 우선순위에 따라 내부 큐에 분배하고 처리 순서를 관리함. **[품질 요구사항]** (QA-01 성능 완전만족) Introduce Concurency 택틱 기반의 Prioritize Events 택틱 및 Schedule Resources 택틱을 구현하여 핵심 기능의 응답 시간을 보장하고 우선순위 가 낮은 작업의 기아 상태를 방지함. (QA-02 가용성 부분만족) 각 서비스 내부에 분산 배치되어 단일 장애점을 제거함. | QA-01, QA-02, AC-03 |
| Heartbeat Monitor | process | **[기능 요구사항]** 시스템 내 다른 컴포넌트들의 상태를 감시하여 장애 발생 여부를 탐지함. **[품질 요구사항]** (QA-02 가용성 완전만족) Heartbeat 택틱을 구현하여 각 컴포넌트의 장애를 신속하게 탐지하고 자동 복구 프로세스를 트리거함. | QA-02, AC-03 |
| Notification Service | process | **[기능 요구사항]** 이상 상태 감지(UC-05) 등 시스템의 주요 이벤트 발생 시 사용자에게 알림을 발송하는 역할을 전담함. **[품질 요구사항]** (QA-02, QA-05 완전만족) Active Redundancy 및 수평 확장을 통해 안정적인 알림 발송을 보장함. | UC-05 QA-02, QA-05, AC-03 |
| DataService | process | **[기능 요구사항]** 데이터 관련 요청을 처리하고, 데이터베이스 및 캐시와의 상호작용을 전담하는 데이터 접근 계층 역할을 수행함. **[품질 요구사항]** (QA-01, QA-05 완전만족) Caching 및 Data Replication 택틱을 구현하여 데이터 접근 성능과 확장성을 보장함.  (QA-04, 보안 미충족) RBAC 충족 방안과 암복호화 방안이 상세설계 단계에서 구현되어야 함.  **[제약 사항]** (AC-02 개인정보보호법 부분만족) 데이터의 전송구간과 저장은모두 암호화 됨. 데이터 접근은 역할 기반 접근 제어(RBAC)를 통해 엄격히 통제되도록 상세설계 되어야함. | UC-01, UC-04 QA-01, QA-02,  QA-04, QA-05,  AC-02, AC-03 |
| In-memory Cache | data store | **[기능 요구사항]** 자주 조회되는 데이터를 메모리에 임시 저장하여 데이터 접근 속도를 높임. **[품질 요구사항]** (QA-01, QA-05 완전만족) Caching 택틱을 적용하여 DB 부하를 원천적으로 줄여, 시스템 전반의 응답 속도를 향상시키고 확장성을 지원함. | QA-01, QA-05, AC-03 |
| Master DB / Slave DB | data store | **[기능 요구사항]**시스템의 모든 영구 데이터를 저장하고 관리함. **[품질 요구사항]** (QA-05 확장성 완전만족) Data Replication(Master-Slave) 택틱을 적용하여 읽기 요청을 분산 처리함. (QA-02 가용성 완전만족) Master DB 장애 시 Failover를 통해 데이터 서비스의 가용성을 보장함. | QA-02, QA-05, AC-03 |
| Gateway Controller | process | **[기능 요구사항]** 홈 게이트웨이 내부의 모든 동작을 총괄함. 클라우드와의 통신, 로컬 기기 데이터 수집 및 제어 흐름을 관리함. **[품질 요구사항]** (QA-06 가용성 부분만족) LocalMemory에 명령 등이 저장되어 연속성 부분 만족. 구체적으로 어떤 방식으로 로컬 기능의 연속성을 보장할 지 상세 설계 되어야 함.  (QA-03 확장성 부분만족) GatewayController와 HomeAssistant가분리되어 확장에 용이한 구조 설계되어 부분 만족함. 신규 IoT 프로토콜 지원 등의 경우는 어떻게 할 지 상세 설계 되어야 함.  **[제약 사항]** (AC-01 오픈소스 완전만족) Home Assistant 기반으로 구현됨. | UC-01, UC-03, UC-05,  QA-03, QA-06, AC-01 |
| DeviceMonitor | process | **[기능 요구사항]** 수집된 데이터를 바탕으로 홈 게이트웨이 내에서 실시간으로 이상 상태를 감지(LocalAnomalyDetector)하는 엣지 컴퓨팅 로직을 수행함(UC-05). | UC-05, QA-06 |

### Connector List

|  |  |  |
| --- | --- | --- |
| Connector  Name | Property Description | Relevant ADs |
| API Gateway | **[관련 기능]** 모든 외부 요청에 대한 단일 진입점(Single Entry Point) 역할을 하며, 인증/인가 및 요청 라우팅을 수행함. **[데이터 포맷]** JSON **[품질 요구사항]** (QA-02 가용성 부분만족) LoadBalancer와 연계하여 Active Redundancy 구조의 일부로 동작함. (QA-05 확장성 부분만족) LoadBalancer를 통해 백엔드 서비스로 트래픽을 분산시키는 관문 역할을 수행함. **[제약 사항]** (AC-02 개인정보보호법 완전만족) JWT 기반 인증 및 역할 기반 접근 제어(RBAC)를 통해 모든 요청에 대한 권한을 검사하여 보안 제약을 만족함. (AC-03 클라우드 기반 완전만족) 클라우드 플랫폼이 제공하는 관리형 API Gateway 서비스를 활용하여 제약사항을 만족함. | UC-02,  UC-03,  UC-04,  QA-02, QA-05 AC-02, AC-03 |
| LoadBalancer | **[관련 기능]** 사용자로부터의 모든 요청(UC-02, UC-03, UC-04 등)을 처리하는 Cloud Service들의 부하를 분산시키는 데 사용됨. **[품질 요구사항]** (QA-02 가용성 완전만족) Health Check 기능으로 장애가 발생한 인스턴스를 제외하여 Active Redundancy를 구현함. (QA-05 확장성 완전만족) 트래픽 분산을 통해 수평적 확장성을 지원함. | UC-02,  UC-03,  UC-04,  QA-02, QA-05 |
| Pub-Sub (MQTT) | **[관련 기능]** 주로 원격 기기 제어(UC-03) 명령 전달 및 결과 수신, 이상 상태 감지(UC-05) 알림 전파 등 실시간 양방향 통신이 필요한 기능에 사용됨. **[데이터 포맷]** JSON **[품질 요구사항]** (QA-01 성능 완전만족) 저지연 통신을 가능하게 함. (QA-02 가용성 완전만족) 버퍼 역할을 하여 일시적인 수신자 장애에 대응함. (QA-03 변경용이성 완전만족) 컴포넌트 간 결합도를 낮춰 변경을 용이하게 함. | UC-03,  UC-05  QA-01, QA-02, QA-03 |
| HTTPS | **[관련 기능]** 사용자 인증, 앱의 모든 기능 요청(UC-02, UC-03 등), 외부 전력공급사와의 데이터 교환(UC-01, UC-02) 등 보안이 중요한 모든 외부 통신에 사용됨. **[데이터 포맷]** JSON **[품질 요구사항]** (QA-04 보안성 완전만족) TLS 암호화를 통해 통신 구간의 데이터를 보호함. **[제약 사항]** (AC-02 개인정보보호법 완전만족) 보안 통신을 통해 법적 제약을 만족함. | UC-01~05,  QA-04, AC-02 |
| gRPC | **[관련 기능]** 클라우드 내부 서비스 간의 모든 동기적 상호작용을 지원함. **[데이터 포맷]** Protocol Buffers **[품질 요구사항]** (QA-01 성능 완전만족) HTTP/2 기반의 바이너리 직렬화를 통해 내부 통신의 응답 속도를 극대화함. | UC-01~05,  QA-01, QA-02 |
| JDBC | **[관련 기능]** 데이터의 영구 저장이 필요한 모든 기능(UC-01~05)에서 DataService가 Main Database에 접근하기 위해 사용됨. **[데이터 포맷]** SQL **[품질 요구사항]** (QA-01 성능 완전만족) Connection Pool과 함께 사용되어 DB 접근 성능을 최적화함. (QA-05 확장성 부분만족) 데이터베이스의 확장성을 지원함. | UC-01~05,  QA-01, QA-05 |
| Redis | **[관련 기능]** 실시간 사용량 조회(UC-04)와 같이 반복적인 데이터 조회가 많은 기능에서 DataService가 In-memory Cache에 접근하기 위해 사용됨. **[데이터 포맷]** Redis String, Hash 등 **[품질 요구사항]** (QA-01 성능 완전만족) 인메모리 데이터 처리에 최적화되어 있어 매우 빠른 응답 속도를 보장함. (QA-05 확장성 완전만족) DB 부하를 원천적으로 감소시켜 확장성을 크게 향상시킴. | QA-01, QA-05 |

## Deployment View

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design  Option Title | Reflection in the Deployment View |
| QA-01 (성능) 원격 제어 응답 속도 | Publish-Subscribe 스타일 | [반영 방식] MessageBroker Server 노드가 명시적으로 존재하며, Application Server와 HomeGateway, Client(User Phone/PC) 노드 간에 «MQTT» 통신 경로가 연결됨. [만족 QA 부분] 이 구조는 클라우드와 클라이언트/게이트웨이 간의 저지연 양방향 통신을 가능하게 하여, 2초 이내 응답이라는 핵심 성능 목표 달성을 보장함. |
| Schedule Resource & Introduce Concurency 택틱 기반의 Prioritize Events 택틱 | [반영 방식] Application Server 노드 내부에 배포되는 deviceControlService.jar와 같은 서비스 아티팩트의 내부 로직으로 구현됨. 다이어그램 상에서는 직접 표현되지 않으나, Node Specification에서 그 역할을 정의함.  [만족 QA 부분] 평균 부하가 높은 상황에서도 긴급한 제어 명령이 다른 작업에 의해 지연되지 않도록 보장하여, 2초 이내 응답 목표를 안정적으로 만족함. |
| 분산된 내부 스케줄러 | [반영 방식] Kubernetes Cluster 내에 PredictionService Container, deviceControlService Container 등 다수의 서비스 컨테이너에 독립적으로 배포된 구조를 통해 표현됨. [만족 QA 부분] 스케줄러가 중앙 병목 지점이 되는 것을 방지하여, 시스템 전체가 안정적인 응답 시간을 유지하도록 함. |
| QA-02 (가용성) 신속한 시스템 장애 감지 및 자동 복구 | Heartbeat 택틱 | [반영 방식] Kubernetes Cluster 내에 HeartbeatMonitorContainer가 별도의 실행 환경으로 명확하게 표현됨. [만족 QA 부분] 컴포넌트 오류를 신속하게 탐지하는 장애 감지 요구사항을 만족시켜, 1분 이내 복구 목표의 선결 조건을 충족함. |
| Active Redundancy 택틱 | [반영 방식] AWS ApplicationLoad Balancer 노드가 Kubernetes Cluster 내의 다중 서비스 컨테이너로 트래픽을 분산하는 구조 전체를 통해 표현됨. [만족 QA 부분] 장애 발생 시 서비스 중단 없이 자동 복구를 수행합니다. 이를 통해 1분 이내 복구 목표를 달성함. |
| 마이크로서비스 아키텍처 | [반영 방식] Application Server 노드가 PredictionService, DeviceControlService 등 여러 개의 독립된 Service Container로 분해되어 배포된 모습으로 표현됨. [만족 QA 부분] 특정 서비스의 장애가 다른 서비스로 전파되는 것을 막는 장애 격리를 통해 시스템 전체의 안정성과 99.5% 가동률 목표 달성에 기여함. |
| QA-05 (확장성) 사용자 증가에 따른 성능 유지 | Multi-tier 스타일 및 로드밸런싱 | [반영 방식] Client, Cloud service(Application/Data), HomeGateway라는 논리적 계층이 물리적으로 분리된 노드들로 표현됨. AWS ApplicationLoad Balancer 노드가 명시적으로 존재함. [만족 QA 부분] 사용자 트래픽이 2배로 증가할 때, Application Server 노드 그룹을 독립적으로 확장하여 기존 응답 속도를 안정적으로 유지할 수 있음. |
| Caching 택틱 | [반영 방식] 클라우드 내에 Cache Server라는 별도의 노드가 명확하게 존재하며, Application Server와 «Redis» 프로토콜로 연결됨. [만족 QA 부분] 사용자 증가로 인한 데이터베이스 조회 부하를 흡수하여, DB 병목 현상을 방지하고 안정적인 처리량을 유지. |
| Sticky Session (고정 세션) | [반영 방식] AWS ApplicationLoad Balancer 노드의 내부 설정으로 반영됨. LoadBalancerConfig.yaml 아티팩트가 이 설정을 정의함을 의미함. [만족 QA 부분] 세션의 일관성을 보장하여 불필요한 데이터 조회를 줄임으로써, 부하가 높은 환경에서도 안정적인 응답 속도 유지에 기여함. |
| Data Replication(Master/Slave) 택틱 | [반영 방식] Database Server 노드가 1(Master)와 \*(Slave) 구성으로 명확히 나뉘어 표현되어 있으며, 이들 간에 Data Replication 통신 경로가 존재함. [만족 QA 부분] 읽기 요청을 여러 Slave 노드로 분산시켜, 사용자 증가에 따른 DB 읽기 성능의 수평적 확장을 가능하게 합니다. |

### Deployment Diagram

텍스트, 스크린샷, 평행, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

본 다이어그램은 4.2절의 설계 결정 과정을 통해 도출된 '스마트홈 전기관리 시스템'의 최종적인 물리적 배포 구조를 나타낸다. 클라우드 내부의 Application Tier는 마이크로서비스 아키텍처를 채택하여, DeviceControl Service, Prediction Service 등 각 기능이 독립된 컨테이너로 쿠버네티스 클러스터 내에 배포된다. 그 앞단에는 AWS Application Load Balancer 노드를 두어 다중화된 서비스 인스턴스로 트래픽을 분산시킴으로써, 많은 동시 접속 사용자에 대한 높은 처리량(확장성, QA-05)과 일부 인스턴스 장애 시에도 중단 없는 서비스를 제공하는 고가용성(QA-02)을 확보하고 있다. 특히 로드밸런서에는 Sticky Session 설정을 적용하여 사용자의 세션 연속성을 보장한다. Data Tier는 Master/Slave 구조로 이중화된 Database Server 노드와, 별도의 Cache Server 노드로 분리하여 물리적으로 배포된다. 이는 Data Replication과 Caching 택틱을 구현한 것으로, 대규모 읽기 요청을 효율적으로 분산 처리하여 확장성(QA-05)을 만족시키고, 데이터베이스 장애 시 Failover를 통해 가용성(QA-02)을 보장한다. 클라우드와 외부 컴포넌트 간의 실시간 통신은 Message Broker 노드를 중심으로 하는 Publish-Subscribe 스타일로 구조화되어 있다. 클라우드 서비스, 홈 게이트웨이, 클라이언트가 MQTT 프로토콜을 통해 비동기적으로 메시지를 교환하여, 원격 제어 명령과 그 결과 피드백을 2초 이내에 처리하는 성능(QA-01) 요구사항을 만족시키고 있다. 또한, 클라우드 내부에 Heartbeat Monitor 컴포넌트를 두어 주요 컴포넌트들의 상태를 감시하는 구조를 통해 장애 탐지 및 복구 용이성(QA-02)을 확보하고 있다.

#### Node Specification

|  |  |
| --- | --- |
| Name | Description |
| User SmartPhone | [역할]사용자가 시스템과 직접 상호작용하는 물리적 기기  [관련 기능]CustomerUI를 실행하여 시스템의 모든 기능을 사용자에게 제공함  [품질기여도]Pub/Sub 커넥터로부터 Push 메시지를 실시간으로 수신하여 UI를 갱신, 빠른 체감 응답 속도(QA-01) 제공  [HW 사양] 일반 스마트폰 사양(쿼드코어 이상 AP, 4GB 이상 RAM) |
| User PC | [역할]웹 브라우저를 통해 시스템에 접근하는 사용자 기기  [관련 기능]웹 기반 CustomerUI를 렌더링하고 실행, 시스템의 모든 기능을 지원함.  [품질 기여도]모바일 앱과 동일하게 실시간 피드백을 받아 체감 응답 속도(QA-01)를 제공  [HW 사양] 일반 상용 개인용 컴퓨터 및 노트북 사양 |
| Home Gateway | [역할]각 가정에 설치된 엣지 컴퓨팅 노드  [관련 기능] 데이터 수집(UC-01), 기기 제어(UC-03), 로컬 이상 상태 감지(UC-05)를 수행함.  [품질 기여도] 인터넷 중단 시 독립동작하여 서비스 연속성(QA-06)을 보장하고 저지연 MQTT 메시지 처리로 빠른 제어 성능(QA-01)에 기여함.  [HW사양] CPU: 1.5GHZ 쿼드 코어(ARM Cortex-A Series)  Memory:2GB PDDR4 / Storage : 16GB |
| Application Server | [역할] 핵심 비즈니스 로직을 수행하는 클라우드의 두뇌  [관련 기능] 요금 예측(UC-02), 원격 제어(UC-03) 등 서버 측 로직을 처리함  [품질 기여도] 다중 인스턴스 배포로 가용성(QA-02)과 확장성(QA-05)을 보장하며, 내부 스케줄러를 통해 성능(QA-01)을 만족시킴.  [HW 사양(가상)] vCPU: 쿼드 코어 이상(Compute Optimized)  Memory : 16GB 이상 |
| Database  Server | [역할] 시스템의 모든 영구 데이터를 저장하는 데이터 저장소  [관련 기능] 모든 UseCase에서 발생하는 데이터를 저장하고 조회함  [품질 기여도] Master-Slave 복제 구조를 통해 데이터의 가용성(QA-02)과 읽기 확장성(QA-05)을 보장함.  [HW 사양(가상)] vCPU:쿼드 코어 이상  Memory : 32GB 이상 /Storage : 고속I/O SSD(High IOPS) |
| Cache Server | [역할] 자주 사용하는 데이터를 임시 저장하는 고속 저장소  [관련 기능] 실시간 사용량 조회(UC-04) 등 반복 조회가 많은 기능의 성능을 향상 시킴.  [품질 기여도] DB 부하를 줄여 시스템 전반의 성능(QA-01)과 확장성(QA-05)을 크게 향상시킴.  [HW 사양(가상)] vCPU: 더블 코어 이상  Memory : 64GB 이상(Memory Optimized) |
| Message Broker Server | [역할] 컴포넌트 간 비동기 메시지 통신을 중개하는 허브.  [관련 기능] 원격 제어(UC-03) 명령 전달 및 결과 수신, 이상 상태 알림(UC-05) 등 실시간 통신을 담당함.  [품질 기여도] 저지연 Push 통신으로 성능(QA-01)을 보장하고, 컴포넌트 간 결합도를 낮춰 변경용이성(QA-03)을 높임. 클러스터 구성으로 가용성(QA-02)을 확보함. |
| AWS API Gateway | [역할] 시스템의 모든 외부 요청에 대한 단일 진입점(Single Entry Point).  [관련 기능] 모든 Use Case(UC-01~05)의 첫 관문 역할을 함.  [품질 기여도] RBAC를 통한 보안(AC-02)을 강화하고, 요청 라우팅 및 캐싱을 통해 성능(QA-01) 향상에 기여함.  [HW 사양] AWS에 의해 관리되며, 트래픽에 따라 자동 확장(Managed & Autoscaled by AWS) |
| AWS Load Balancer | [역할] API Gateway를 통과한 트래픽을 다수의 Application Server로 분산하는 역할. [관련 기능] 모든 Use Case(UC-01~05)의 부하를 분산함.  [품질 기여도] Active Redundancy를 구현하여 가용성(QA-02)을 보장하고, 수평적 확장을 가능하게 하여확장성(QA-05)을 만족시킴.  [HW 사양] AWS에 의해 관리되며, 트래픽에 따라 자동 확장(Managed by AWS) |

#### Execution Environment Specification

|  |  |  |
| --- | --- | --- |
| Node | Name | Description |
| User Phone | Android OS/ iOS | [SW 사양] Android 10(API 레벨 29) 이상, iOS 14 이상  [특성] 모바일 애플리케이션을 실행하는 운영체제임. 운영체제가 제공하는 네이티브 UI 컴포넌트, 푸시 알림 서비스, 로컬 저장소 등을 활용하여 사용자에게 최적화된 경험을 제공함.  [결정 근거]  성능(QA-01): 운영체제 수준에서 WebSocket 및 MQTT 프로토콜을 안정적으로 지원하여, 서버로부터의 실시간 Push 메시지를 저지연으로 수신하고 빠른 피드백을 제공할 수 있음. |
| User PC | Web Browser | [SW 사양]Chrome 등 최신 버전의 Modern Web Browser  [특성] Windows 또는 macOS와 같은 데스크톱 운영체제 위에서 동작하며, 웹 애플리케이션(HTML, JS, CSS)을 렌더링하고 실행하는 환경임.  [결정 근거  성능(QA-01): 최신 웹 브라우저는 WebSocket API를 표준으로 지원하므로, 별도의 플러그인 설치 없이 서버와의 실시간 양방향 통신 채널을 수립하여 Pub/Sub 모델을 구현할 수 있음. |
| Application  Server | Kubernetes Cluster | [SW 사양] Kubernetes v1.28+, Docker Engine v20.10+  [특성] DeviceControl Service, Prediction Service 등 우리의 핵심 서비스 컴포넌트들을 컨테이너 형태로 배포, 확장, 관리하는 컨테이너 오케스트레이션 플랫폼임. [결정 근거]  가용성(QA-02): 컨테이너에 장애 발생 시 자동으로 재시작시키고, 노드 장애 시 다른 노드로 컨테이너를 재배치하여 Active Redundancy 택틱의 구현을 지원함.  확장성(QA-05): CPU 사용량과 같은 부하에 따라 서비스 컨테이너의 수를 자동으로 늘리거나 줄이는 HPA(Horizontal Pod Autoscaling) 기능을 통해 탄력적인 확장성을 제공함. 이는 AC-03(클라우드 기반) 제약사항을 효과적으로 활용하는 방식임. |
| Database Server | MySQL Server | [SW 사양] MySQL v8.0+ (InnoDB 엔진)  [특성] 정형화된 데이터를 안정적으로 저장하고 관리하는 관계형 데이터베이스 관리 시스템임. [결정 근거]  가용성(QA-02) & 확장성(QA-05): Master-Slave Replication 기능을 네이티브로 지원하여, 읽기 부하 분산과 장애 시 Failover를 통한 고가용성 데이터 티어 구축이 용이  AC-01(오픈소스 프레임워크): 널리 사용되고 검증된 오픈소스 RDBMS로, 기술 지원과 개발자 생태계가 풍부하여 안정적인 운영이 가능함. |
| Cache  Server | Redis Server | [SW 사양] Redis v7.0+  [특성] 모든 데이터를 메모리에 저장하여 디스크 기반 데이터베이스보다 월등히 빠른 속도로 데이터를 읽고 쓸 수 있는 키-값 저장소임. [결정 근거]  성능(QA-01) & 확장성(QA-05): Caching 택틱을 구현하기 위한 최적의 솔루션임. DB 접근 횟수를 획기적으로 줄여 시스템 전체의 응답 시간을 단축시키고, 대규모 조회 트래픽을 효율적으로 흡수하여 DB의 부하를 줄여줌. 클러스터링 기능을 통해 캐시 서버 자체의 확장성과 가용성 확보도 용이함. |
| ServiceContainer(Java) | JVM | \*Java ServiceContainer : DataServiceContainer, NotificationServiceContainer,  DeviceControlServiceContainer, HeartbeatMonitorContainer  [SW 사양] OpenSDK 17 LTS  [특성] DeviceControlService.jar와 같이 Java로 개발된 마이크로서비스 아티팩트를 실행하는 가상 머신 환경임.  [결정 근거]  성능(QA-01) & 가용성(QA-02): JVM은 오랜 기간 검증된 JIT(Just-In-Time) 컴파일러와 GC(Garbage Collection) 최적화를 통해, 장시간 안정적으로 높은 성능을 유지해야 하는 서버 애플리케이션에 적합함. LTS버전을 사용하여 안정성을 극대화함. |
| PredictionService Container, HomeGateway | PythonRuntime | [SW 사양] Python3.9+  [특성] Python 기반 아티팩트를 실행하는 환경임.  [결정 근거]  BC-01(시간 제약): Python은 방대한 IoT 및 네트워크 라이브러리 생태계를 갖추고 있어, Pub/Sub 통신 등 복잡한 기능을 빠르게 개발할 수 있도록 지원함.  AC-01(오픈소스 프레임워크): Home Assistant와 Prophet (PredictionService)이 Python 기반이므로, 동일한 언어 및 실행 환경을 사용하여 기술 스택을 통일하고 호환성 문제를 최소화할 수 있음. |
| HomeGateway | Embedded Linux | [SW 사양] Yocto Project 기반 커스텀 Linux  [특성] 홈 게이트웨이와 같은 임베디드 시스템에 최적화된 경량 리눅스 운영체제임. 시스템 운영에 필수적인 커널과 라이브러리만 포함하여 안정적이고 가벼움.  [결정 근거]  변경용이성(QA-03): 검증된 IoT기기를 제어 오픈소스인 HomeAssistant를 OS/Framework 형태로 전부 사용하는 대신 라이브러리 형태로 사용할 수 있도록 전체 시스템은 Yocto 기반의 커스텀 Linux 사용, 이후 신규 프로토콜에도 유연하게 대응 가능.  가용성(QA-06): 안정성이 검증된 리눅스 커널을 기반으로 하여, 24/7 운영이 필요한 홈 게이트웨이의 높은 가용성 요구사항을 만족시킴.  AC-01(오픈소스 프레임워크): Yocto는 오픈소스 프로젝트로, 특정 하드웨어(ARM Cortex-A)에 맞게 OS를 커스터마이징할 수 있어 Home Assistant와 같은 다른 오픈소스와의 호환성을 확보하는 데 유리함. |

#### Communication Path Specification

|  |  |
| --- | --- |
| Path | Description |
| Client Device – Cloud Platform | [프로토콜] HTTPS (REST API) over TCP/IP  [네트워크] Public Internet (Wi-Fi, 5G/LTE)  [보안] TLS 암호화, JWT 및 RBAC 기반 인증/인가  [결정 근거]  보안성(QA-04) & 개인정보보호법(AC-02): 사용자의 모든 요청과 개인정보를 안전하게 보호하기 위해, 표준 웹 보안 프로토콜인 HTTPS를 사용하여 통신 채널 전체를 암호화함. 또한 JWT와 RBAC를 통해 허가된 사용자만이 허가된 기능에 접근하도록 통제함.  상호운용성: 모바일 앱(Android/iOS)과 웹 브라우저 등 다양한 클라이언트 환경에서 공통적으로 사용할 수 있는 표준 프로토콜인 REST API를 채택함. |
| Cloud Platform – Home Gateway | [프로토콜] MQTT over TCP/IP (Publish-Subscribe)  [네트워크] Public Internet  [특징] 홈 게이트웨이가 클라우드 측으로 항상 연결을 유지하는 Persistent Connection 방식임.  [보안] TLS 암호화 및 기기 인증서(Device Certificate) 기반 인증  [결정 근거]  성능(QA-01): 원격 제어 명령을 2초 내에 전달해야 하는 저지연 요구사항을 만족시키기 위해, 경량 메시징 프로토콜인 MQTT를 채택함. Push 방식을 통해 서버가 게이트웨이로 즉시 명령을 전달할 수 있음.  변경용이성(QA-03): Pub/Sub 모델은 송신자(클라우드)와 수신자(게이트웨이) 간의 의존성을 낮추므로, 향후 새로운 유형의 메시지나 기기를 추가하기 용이함. |
| Cloud Internal(Application Tier – Data Tier) | [프로토콜] gRPC (서비스 간), JDBC (DB 연결), Redis Protocol (캐시 연결)  [네트워크] 고속, 저지연의 클라우드 내부 사설망  [결정 근거]  성능(QA-01): 외부 인터넷보다 훨씬 빠르고 안정적인 클라우드 사설망을 사용하여 내부 통신의 지연을 최소화함. 서비스 간 통신에는 JSON/REST보다 가볍고 빠른 gRPC를, DB 통신에는 Connection Pool을 적용한 JDBC를 사용하여 성능을 극대화함.  보안성(QA-04): 외부 인터넷에 노출되지 않은 격리된 사설망을 사용하여, 인가되지 않은 접근으로부터 데이터베이스와 같은 핵심 자산을 보호함. |
| Cloud Platform – External System | [프로토콜] HTTPS (REST API)  [네트워크] Public Internet  [보안] TLS 암호화, OAuth 2.0 기반 서버 간 인증  [결정 근거]  보안성(QA-04): 외부 기관인 전력 공급사와 안전하게 데이터를 교환하기 위해, 통신 채널은 HTTPS로 암호화하고, 서버 간의 신원은 OAuth 2.0 표준을 통해 인증하여 허가된 서버만 통신할 수 있도록 통제함. |
| HomeGateway - ClientDevice | [프로토콜] MQTT over TCP/IP (Publish-Subscribe)  [네트워크] Public Internet  [특징] 홈 게이트웨이가 기기 상태 변경을 감지하고, 해당 결과를 특정 토픽으로 발행(Publish)하면, 이 토픽을 구독(Subscribe)하고 있는 클라이언트에게 메시지 브로커가 즉시 전달(Push)함.  [결정 근거]  성능(QA-01): 사용자가 기기를 제어한 후, 그 결과가 실제로 반영되었는지 여부를 거의 실시간으로 피드백받아야 함. MQTT Push 방식은 Home Gateway가 상태 변경을 감지하는 즉시 클라이언트로 메시지를 전달할 수 있어, 2초 이내 응답이라는 성능 목표를 최종적으로 만족시키는 핵심적인 경로임.  변경용이성(QA-03): Home Gateway는 정해진 토픽으로 상태를 발행하기만 하면 되고, 이 메시지를 누가 수신하는지(모바일 앱, 웹, 운영자 대시보드 등) 알 필요가 없음. 이처럼 송신자와 수신자가 분리되어 있어, 향후 피드백을 수신할 새로운 클라이언트를 추가하더라도 Home Gateway의 코드를 수정할 필요가 없음. |

### Artifact Definition Diagram

Deployment View에 표현된 각 Artifact에 대해 다룬다. 각 Artifact와 C&C뷰에서의 component, connector 간 관계와 역할에 대해 설명한다.

텍스트, 스크린샷, 폰트, 사각형이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Artifact Definition Specification

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Manifested Component | Depends on | Description and Rationale |
| CustomerUI.apk | CustomerUI,  AppCache | HTTPS MQTT | 사용자의 안드로이드 스마트폰에 설치되는 모바일 애플리케이션 설치 파일입니다. |
| WebAppBundle.js | CustomerUI,  AppCache | HTTPS MQTT | 사용자의 PC 웹 브라우저에서 실행되는 웹 애플리케이션을 구성하는 핵심 JavaScript 파일 묶음입니다. |
| GatewayConfig.yaml. | APIGateway | HTTPS | API Gateway 서비스의 라우팅 규칙, 인증, 백엔드 연동 등 동작 방식을 정의하는 설정 파일. |
| LoadBalancerConfig.yaml | LoadBalancer | - | 클라우드 환경의 로드 밸런서 동작을 정의하는 설정 파일입니다. 그리고 서버의 정상 상태를 확인하는 Health Check 방식과 Sticky Session 적용 여부 등의 규칙이 기술됩니다. |
| HomeGatewayApp.py | Gateway Controller, , LocalMemory, | MQTT | 각 가정의 홈 게이트웨이 장치에 배포되는 실행 가능한 Python 애플리케이션 실행 파일입니다. 홈 게이트웨이의 주된 로직을 제어합니다.  Docker 이미지로 패키징되어 배포됩니다. |
| DeviceMonitor.py | DeviceMonitor, LocalAnomalyDetector | gRPC | 각 가정의 홈 게이트웨이 장치에 배포되는 실행 가능한 Python 애플리케이션 실행 파일입니다.  기기 모니터링 및 이상 상태 감지를 담당합니다. Docker 이미지로 패키징되어 배포됩니다. |
| HomeAssistant.pyz | HomeAssistant | MQTT,  HomeAssistant Lib | 각 가정의 홈 게이트웨이 장치에 배포되는 실행 가능한 Python 애플리케이션 아카이브 파일입니다. |
| DeviceControlService.jar | DeviceControl Service, TaskScheduler | gRPC JDBC MQTT | 원격 기기 제어 및 스케줄링 로직을 담고 있는 Java 마이크로서비스 실행 파일입니다. Docker 이미지로 패키징되어 배포됩니다. |
| OptimizationEngine.py | OptimizationEngine | gRPC | 규칙 기반 최적화 제안을 하는 로직을 담고 있는 Python 애플리케이션 실행 파일입니다. Docker 이미지로 패키징되어 배포됩니다. |
| [PredictionService.py](http://PredictionService.py) | Prediction Service, PredictionModel | gRPC Prophet Lib | AI 기반 요금 예측 로직을 담고 있는 Python 애플리케이션 실행 파일입니다. Docker 이미지로 패키징되어 배포됩니다. |
| DataService.jar | DataService | gRPC JDBC Redis | 데이터베이스 및 캐시와의 통신을 전담하는 데이터 접근 계층 마이크로서비스의 Java 실행 파일입니다. |
| NotificationService.jar | Notification Service | gRPC SMS API | 사용자에게 푸시 알림이나 SMS를 발송하는 알림 전용 마이크로서비스의 Java 실행 파일입니다. |
| HeartbeatMonitor.jar | Heartbeat Monitor | gRPC | 다른 서비스들의 동작 상태를 감시하는 모니터링 서비스의 Java 실행 파일입니다. |
| MainDBSchema.sql | Master DB Slave DB | JDBC | Master/Slave 데이터베이스의 테이블, 인덱스 등 전체 스키마 구조를 정의하는 SQL 스크립트 파일입니다. |
| MQTT-Broker-config.yaml | MQTT Broker(Pub/sub) | MQTT | 클라우드에서 동작하는 MQTT Broker의 토픽, 보안 정책 등을 정의하는 설정 파일. |
| Redis-config.conf | In-memory Cache | Redis | In-memory Cache(Redis) 서버의 동작(메모리 정책, 클러스터링 등)을 정의하는 설정 파일. |
| Redis | Redis | - | DataService가 In-memory Cache 서버와 통신하기 위해 사용하는 클라이언트 라이브러리입니다. |
| JDBC | JDBC | - | Java 기반 서비스들이 관계형 데이터베이스와 통신하기 위한 표준 API 및 드라이버 라이브러리입니다. |
| MQTT | MQTT | - | 각 컴포넌트(서비스, 게이트웨이, UI)가 MQTT Broker와 통신하기 위해 사용하는 클라이언트 라이브러리입니다. (예: Paho MQTT) |
| gRPC | gRPC | - | 클라우드 내부 마이크로서비스 간의 고성능 RPC 통신을 위해 사용되는 프레임워크 및 라이브러리입니다. |
| HTTPS | HTTPS | - | 클라이언트 및 외부 시스템과 안전한 통신을 하기 위해 사용되는 표준 프로토콜 라이브러리입니다. (일반적으로 언어/프레임워크에 내장됨) |
| SMS API | SMS API | - | NotificationService가 외부 SMS 발송 업체 시스템과 연동하기 위한 API 규격 또는 클라이언트 라이브러리입니다. |
| Prophet | Prophet | - | PredictionService가 시계열 데이터 기반의 전기 요금을 예측하기 위해 사용하는 머신러닝 라이브러리입니다. |
| HomeAssistant Lib | HomeAssistant Lib | - | HomeAssistant가 다양한 IoT 기기들과의 통신 및 제어를 쉽게 하기 위해 사용하는 핵심 IoT 프레임워크 라이브러리입니다. |

# Detailed Component Design Description

4장에서 작성한 내용을 토대로 시스템에서 핵심적인 세 컴포넌트의 상세설계, 즉 구현 레벨에서의 설계 이슈 등을 도출하고 적합한 디자인 옵션을 선택하도록 하며 선택된 디자인 옵션이 포함된 클래스 다이어그램 및 시퀀스 다이어그램을 작성한다.

## Design Description for *GatewayController*

### Overview

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 개요 | 홈 게이트웨이 내에서 클라우드와 IoT 디바이스 간의 통신을 중재하고, 디바이스 제어 및 상태 모니터링을 담당하는 핵심 컴포넌트이다. |
| 컴포넌트 기능 요구사항 | [Provided Interface]   * IGatewayService: 클라우드로부터 수신한 명령을 처리하고, 디바이스의 상태를 클라우드로 보고하는 등 게이트웨이의 핵심 비즈니스 로직 흐름을 정의하는 인터페이스.   [Required Interface]   * IMQTTBroker: 클라우드의 Message Broker와 비동기 메시지를 주고받기 위한 통신 인터페이스. * IDeviceControl: HomeAssistant 라이브러리를 통해 실제 IoT 디바이스를 제어하고 상태를 조회하기 위한 인터페이스. * IDeviceMonitor: 연결된 디바이스의 상태를 감시하는 DeviceMonitor 컴포넌트와의 상호작용을 위한 인터페이스. * ILocalStorage: 게이트웨이의 로컬 저장소에 데이터를 저장하고 조회하기 위한 인터페이스. |
| 컴포넌트 품질 요구사항  및 제약사항 | QA-03 (변경용이성 - 다양한 IoT 디바이스 지원): 새로운 종류의 IoT 디바이스가 추가되거나 기존 디바이스의 통신 프로토콜이 변경될 경우, HomeAssistant Lib를 활용하여 최소한의 코드 변경으로 해당 디바이스를 시스템에 통합할 수 있어야 한다.  QA-06 (가용성 - 로컬 기능 유지): 클라우드와의 통신 단절 시에도 LocalAnomalyDetector와 연계하여 기본적인 디바이스 모니터링 및 로컬에서 설정된 비상 제어는 중단 없이 수행할 수 있어야 한다.  [제약 사항] (AC-01 오픈소스 완전만족) Home Assistant 기반으로 구현. |

### Quality Driven Component Design

#### Exploring Design Options for *QA-03 신규 IoT 프로토콜 지원 확장*

* + - * 1. **Quality Driven Component Design Structure**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Design Decisions for *설계이슈1 IoT 디바이스 연동 로직의 복잡성 관리 및 확장 방안*

###### Definition for *설계이슈1 IoT 디바이스 연동 로직의 복잡성 관리 및 확장 방안*

**배경 및 가정:** GatewayController는 HomeAssistant 라이브러리를 통해 다양한 제조사의 IoT 디바이스를 제어해야 한다. 현재 우리가 다루려는 기능은 기기의 전원 제어 (켜기/끄기)와 상태 정보(전원 상태, 전력 사용량) 획득이 전부이다. HomeAssistant 라이브러리는 다양한 프로토콜과 디바이스를 지원하지만, 그 인터페이스는 GatewayController가 직접 사용하기에 범용적이지 않거나 불필요하게 복잡할 수 있다. (QA-03 확장성 완전만족)

**이슈 정의:** HomeAssistant 라이브러리의 복잡성을 GatewayController의 핵심 로직으로부터 숨기고(캡슐화), 향후 새로운 IoT 프로토콜이 추가되더라도 GatewayController의 변경을 최소화할 수 있는 디자인 패턴은 무엇인가?

###### Design Options for *설계이슈1 IoT 디바이스 연동 로직의 복잡성 관리 및 확장 방안*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| Adapter 패턴 적용 | GatewayController의 IDeviceControl 인터페이스에 맞춰 HomeAssistant 라이브러리 내의 특정 디바이스(예: 조명, 스위치)나 프로토콜 인터페이스를 '변환'하는 HomeAssistantDeviceAdapter 클래스를 구현하는 방식. GatewayController는 이 Adapter를 통해서만 HomeAssistant가 제어하는 디바이스와 상호작용한다. |
| Facade 패턴 적용 | HomeAssistant 라이브러리의 복잡한 서브 시스템(전원 제어, 센서 조회 등)을 감싸는 단순하고 통합된 DeviceFacade 인터페이스를 제공하는 방식. GatewayController는 이 Facade를 통해서만 디바이스를 제어한다. |

##### Decision and Rationale for *설계이슈1 IoT 디바이스 연동 로직의 복잡성 관리 및 확장 방안*

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Adapter 패턴 적용(Selected) | [인터페이스 호환성 및 단순화] HomeAssistant의 다양한 디바이스별 인터페이스를 IDeviceControl의 표준화된 메서드로 변환하여 일관된 방식으로 디바이스를 제어할 수 있게 한다. 이는 실제 요구사항인 "전원 제어 및 상태 정보 획득"에 부합  [확장성 (QA-03)] 향후 새로운 IoT 프로토콜이나 HomeAssistant가 지원하는 새로운 유형의 디바이스가 추가되더라도, 해당 디바이스/프로토콜에 맞는 새로운 Adapter만 구현하여 IDeviceControl 인터페이스를 맞추면 되므로, GatewayController의 핵심 로직 변경 없이 QA-03(확장성)을 보장한다. | [초기 구현 복잡도] 각 디바이스 유형별 또는 프로토콜별 Adapter 클래스를 구현해야 하므로, 초기 설계 및 구현에 약간의 오버헤드가 발생할 수 있다.  [재사용성 제약] Adapter가 특정 HomeAssistant 디바이스/프로토콜에 맞춰 구현되므로, 다른 컨텍스트에서의 재사용이 제한적일 수 있다. | 우리의 핵심 요구사항은 표준 IDeviceControl 인터페이스와 HomeAssistant의 다양한 디바이스별 인터페이스 간의 "호환성"을 확보하고, 이를 통해 "새로운 프로토콜 지원 시 변경 최소화"를 달성하는 것이다. Adapter 패턴은 이러한 "인터페이스 불일치 해결"과 "확장성"이라는 두 가지 목표를 가장 정확하고 효율적으로 만족시킨다. HomeAssistant 라이브러리의 복잡한 서브 시스템 전체를 감싸는 Facade보다는, 필요한 특정 기능(전원 제어, 상태 획득)에 대한 인터페이스를 변환하는 Adapter가 현재 요구사항에 더 적합하다고 판단된다. 따라서 Adapter 패턴을 선택한다.  [단점 보완 : 재사용성]  범용적인 Generic Adapter를 설계하여 여러 유사한 Adapteee에 재사용하거나 Adapter 내부에서 전략 패턴 |
| Facade 패턴 적용 | [단순화 및 결합도 감소] HomeAssistant의 복잡한 내부 구조를 알 필요 없이, 단순한 메서드만 호출하면 된다. 이를 통해 라이브러리와의 결합도를 크게 낮춘다. [확장성 (QA-03)] 향후 HomeAssistant가 아닌 다른 IoT 라이브러리를 추가로 사용하게 되더라도, 새로운 Facade를 구현하여 교체하면 되므로 핵심 로직은 변경되지 않는다. | [유연성 제한] Facade가 제공하지 않는 HomeAssistant의 세부적인 기능을 사용하려면 Facade를 수정해야 하는 번거로움이 있을 수 있다. | Facade 패턴은 향후 라이브러리의 버전이 업그레이드되거나 내부 구조가 변경되더라도 Façade 층에서 완화 된다는 장점이 있지만 IoT 기기 특성 상 표준 프로토콜을 이용하여 구현되기 때문에 신규 프로토콜의 경우가 아니라면 내부 서브 시스템이 바뀌는 경우가 적을 것으로 판단됨. |

#### Exploring Design Options for *QA-06 인터넷 중단 시 로컬 기능 연속성*

* + - * 1. **Quality Driven Component Design Structure**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

#### Design Decisions for *설계이슈1 인터넷 중단 시 로컬 기능 연속성 보장 방안*

###### Definition for *설계이슈1 인터넷 중단 시 로컬 기능 연속성 보장 방안*

**배경 및 가정:** GatewayController는 클라우드 MQTT Broker와 통신하여 제어 명령을 수신하고 디바이스 상태를 보고한다. 인터넷 연결이 중단될 경우, 수신되었거나 전송해야 할 명령/상태 데이터가 유실되지 않고 로컬에서 보존 및 관리되어야 한다. (QA-06 가용성 완전만족)

**이슈 정의:** 네트워크 단절 상황에서도 GatewayController가 처리해야 할 명령 및 상태 데이터를 안전하게 저장하고, 네트워크 복구 시 이를 자동으로 재처리하여 로컬 기능의 연속성과 데이터 무결성을 보장하는 디자인 패턴은 무엇인가?

###### Design Options for *설계이슈1 인터넷 중단 시 로컬 기능 연속성 보장 방안*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| Command 패턴 적용 | GatewayController가 수행해야 할 모든 작업(예: 전원, 상태 보고)을 ICommand 인터페이스를 구현하는 독립적인 객체(TurnOnCommand 등)로 캡슐화하는 방식. 이 Command 객체들을 로컬 큐(ICommandQueue)에 저장하고, 네트워크 연결 상태에 따라 큐의 Command들을 실행하거나 대기시킨다. |
| Memento 패턴 적용 (상태 저장 및 복원) | GatewayController나 DeviceMonitor와 같은 특정 객체의 내부 상태(예: 미전송 데이터, 현재 제어 상태)를 캡슐화하여 외부(ILocalStorage)에 저장하고 필요할 때 복원하는 방식. |

##### Decision and Rationale for *설계이슈1 인터넷 중단 시 로컬 기능 연속성 보장 방안*

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Command 패턴 적용  (Selected) | [가용성 (QA-06)] 모든 작업이 Command 객체로 캡슐화되어 로컬 큐에 저장되므로, 인터넷 단절 시에도 명령이나 데이터가 유실되지 않고 안전하게 보존된다. 네트워크 연결 복구 시 큐의 Command들을 순차적으로 실행하여 QA-06(로컬 기능 연속성)을 완벽하게 보장한다. [재처리 로직 분리] 네트워크 복구 시의 재처리 로직이 Command 객체 내부에 캡슐화되므로, GatewayController의 핵심 로직은 네트워크 상태 변화에 따른 복잡한 예외 처리를 담당하지 않아도 된다. | [클래스 증가] 각 작업 유형별로 별도의 Command 클래스를 정의해야 하므로, 구현 초기 클래스의 수가 다소 증가할 수 있다. [오버헤드] Command 객체 생성 및 큐 관리, 실행 과정에서 약간의 런타임 오버헤드가 발생할 수 있다. | Command 패턴은 외부 통신에 의존하는 모든 명령과 보고를 Command 객체로 캡슐화하여, 이를 로컬의 영속적인 큐에 저장함으로써 QA-06(가용성)을 완벽하게 만족시킨다. 인터넷 단절 시 Command를 큐에 쌓아두고, 연결 복구 시 이를 꺼내어 실행함으로써 데이터 유실 없이 로컬 기능의 연속성을 보장한다. 이는 임베디드 환경에서 제한된 자원을 효율적으로 사용하면서도, 외부 통신의 불안정성에 강력하게 대응하는 가장 직접적이고 효과적인 방법이다.  \*해당 패턴의 trade-off이나 시스템 특성상 간단한 명령이 주를 이루고 초기 컨셉에서 기기 제어에 대한 명령이 우려될 만큼 많이 추가되지 않을 것으로 판단되며 오버헤드 또한 Object Pooling 등을 통해 완화 가능 |
| Memento 패턴 적용 (상태 저장 및 복원) | [상태 무결성] 게이트웨이의 중요한 상태를 안전하게 저장하고, 재부팅이나 오류 발생 시 이전 상태로 복원하여 시스템의 무결성을 유지할 수 있다. [간단한 복원] 복원 시점의 상태를 한 번에 쉽게 재설정할 수 있다. | [작업 관리의 부재] "수행해야 할 작업" 자체를 저장하고 재실행하는 메커니즘을 직접 제공하지는 않는다. 미전송 명령이나 미처리 이벤트를 관리하기 위해서는 Memento 외에 추가적인 로직이나 패턴이 필요하다. [저장 대상 제한] Memento는 특정 객체의 내부 상태에 초점을 맞추므로, 광범위한 시스템 이벤트나 명령 스트림을 저장하는 데는 부적합하다. | Memento 패턴은 객체의 상태를 저장하고 복원하는 데 적합하지만, "수행해야 할 작업" 자체를 저장하고 재실행하는 데는 직접적인 해답을 주지 못한다. |

#### Design Decisions for *설계이슈2 로컬 저장소의 데이터 영속성 확보 방안*

###### Definition for *설계이슈2 로컬 저장소의 데이터 영속성 확보 방안*

**배경 및 가정:** Command 패턴을 적용하기로 결정했으며, GatewayController는 네트워크 단절 시 Command 객체들을 로컬 큐에 저장해야 한다. 이 로컬 큐에 저장된 Command 객체들은 게이트웨이 장치 재부팅 시에도 유실되지 않고 유지되어야 한다. (QA-06)

**이슈 정의:** 임베디드 환경의 제한된 자원을 효율적으로 사용하면서, 게이트웨이 재부팅 시에도 로컬 큐의 Command 메시지를 안전하게 보존하고 복구할 수 있는 방법은 무엇인가?

###### Design Options for *설계이슈2 로컬 저장소의 데이터 영속성 확보 방안*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| In-memory Data Storage | Command 객체들을 게이트웨이 장치의 RAM에만 저장하는 방식. |
| 파일 기반 Persistent Storage | Command 객체들을 직렬화하여 파일 시스템(예: 플래시 메모리)에 순차적으로 기록함으로써 영속성을 확보하고, 재부팅 시 해당 파일로부터 Command 객체들을 역직렬화하여 로컬 큐를 복원하는 방식. |

##### Decision and Rationale for *설계이슈2 로컬 저장소의 데이터 영속성 확보 방안*

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| In-memory Data Storage | [최고 성능] 메모리에서 직접 데이터를 처리하므로 Command 객체의 저장 및 조회 속도가 매우 빠르다. [간단한 구현] 구현이 매우 단순하며, 추가적인 파일 I/O나 데이터베이스 접근이 없다. | [데이터 유실] 게이트웨이 장치 재부팅 시 큐에 저장된 모든 Command 객체들이 유실된다. 이는 QA-06(로컬 기능 연속성)을 위배하여 인터넷 단절 시 미처리 명령/데이터가 사라지는 치명적인 문제가 발생한다. [메모리 한계] 큐에 많은 Command 객체가 쌓일 경우, 제한된 임베디드 메모리 자원에 부담을 줄 수 있다. | In-memory Data Storage는 성능이 좋지만, 게이트웨이 재부팅 시 Command 객체 유실이라는 치명적인 단점으로 인해 QA-06(가용성)을 만족시킬 수 없다. 로컬 큐에 쌓인 미처리 Command들은 클라우드 연결이 복구되었을 때 반드시 전송/실행되어야 한다. |
| 파일 기반 Persistent Storage (Selected) | [데이터 영속성 (QA-06)] 게이트웨이 장치 재부팅 시에도 로컬 큐에 저장된 Command 객체들이 유실되지 않고 안전하게 보존된다. 이는 QA-06(가용성)을 직접적으로 만족시킨다. [임베디드 적합성] 복잡한 데이터베이스 시스템 없이 파일 시스템을 활용하므로 임베디드 환경에 적합하다. | [성능 오버헤드] 파일 I/O가 발생하므로 In-memory Data Storage 대비 Command 객체 저장 및 조회 속도가 느려질 수 있다. [파일 시스템 관리] 파일 크기 관리, 손상된 파일 처리, 디스크 공간 사용량 등 파일 시스템에 대한 추가적인 고려가 필요하다. | 파일 기반 Persistent Storage를 사용하여 Command 객체들을 비휘발성 저장소(예: 플래시 메모리)에 저장함으로써, 게이트웨이 재부팅 시에도 메시지 유실 없이 안전하게 복구(QA-06)할 수 있도록 한다. 이는 AC-01(오픈소스 프레임워크 사용)에 따라 이미 구현된 경량의 Persistent Queue 라이브러리를 활용하여 BC-01(제한된 개발 시간)을 지킬 수 있다.  \*성능 오버헤드 및 파일 시스템 관리에 관해서는 해당 옵션의 trade-off로 판단되나 본 시스템 특성 상 각 홈 게이트웨이의 로컬 메모리가 Command 객체로 인해 디스크 사용량에 무리가 가지는 않을 것으로 보임. |

### Module View

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design Option Title | Reflection in the Module View |
| QA-03(확장성) | Adapter 패턴 | Adapter는 GatewayController가 요구하는 표준 IDeviceControl 인터페이스를 구현하면서, HomeAssistantAPI라는 특정 라이브러리의 비표준 인터페이스를 IDeviceControl에 맞춰 변환하는 역할로 반영. 이를 통해 라이브러리의 복잡성을 직접 다루지 않고, 표준화된 방식으로 디바이스를 제어할 수 있어 새로운 프로토콜/디바이스 추가 시 GatewayController의 코드 변경을 최소화. |
| QA-06(가용성) | Command 패턴 | 기기제어 및 상태 보고 와 같은 모든 작업을 ICommand 인터페이스를 구현하는 객체로 캡슐화하여 반영. 이 Command 객체들은 PersistentCommandQueue에 저장되어 네트워크 단절 시에도 유실되지 않고 보존되며, 네트워크 복구 시 순차적으로 실행. 이는 로컬 기능의 연속성을 보장하여 QA-06 가용성을 만족. |
| 파일 기반 Persistent Storage | PersistentCommandQueue는 ILocalStorage 인터페이스를 통해 영속적인 데이터 저장 기능을 요청하며, 이 인터페이스는 FilePersistentStorage에 의해 구현. FilePersistentStorage는 Command 객체들을 직렬화하여 파일 시스템에 저장함으로써 게이트웨이 재부팅 시에도 Command 큐의 데이터 유실을 방지. |

#### Class Diagram

본 클래스 다이어그램은 GatewayController 컴포넌트가 확장성(QA-03) 및 가용성(QA-06) 품질 요구사항을 만족시키기 위한 상세 설계를 나타냄. Adapter 패턴 적용을 위해 HomeAssistantDeviceAdapter를 통해 HomeAssistantAPI의 다양한 디바이스 제어 인터페이스가 GatewayController의 표준 IDeviceControl 인터페이스로 변환되었고 Command 패턴 및 파일 기반 Persistent Storage 적용을 위해 기기 제어 및 상태 보고와 같은 작업이 ICommand 객체로 캡슐화 되었음. 이러한 패턴 적용을 통해 GatewayController는 외부 환경 변화에 유연하게 대응하며 안정적인 IoT 게이트 웨이 서비스를 제공할 수 있도록 설계됨. IGatewayService는 GatewayController가 외부에 제공하는 "서비스 계약"을 정의하고, IMQTTBroker는 그 서비스 계약에 따라 주고받는 메시지를 위한 "통신 수단"을 추상화함. 텍스트, 스크린샷, 평행, 흑백이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

##### Element List

|  |  |
| --- | --- |
| Name | Description |
| GatewayController | [역할] 게이트웨이의 핵심 비즈니스 로직을 오케스트레이션하는 주체.  [기능] 클라우드 명령 처리, 디바이스 제어, 상태 모니터링 및 보고, 네트워크 상태 변화 감지 및 대응 등의 전반적인 기능을 총괄함.  [품질] QA-06(가용성)을 위해 ICommandQueue를 활용하여 오프라인 상황에서도 명령 연속성을 보장함. |
| IGatewayService | [역할] GatewayController가 외부에 제공하는 서비스 인터페이스.  [기능] 클라우드 등 상위 시스템이 게이트웨이에 명령을 전달하는 계약을 정의함. |
| IDeviceControl | [역할] GatewayController가 IoT 디바이스를 제어하고 상태를 조회하기 위해 기대하는 표준 인터페이스.  [기능] 디바이스의 전원 제어 및 상태(전원, 전력 사용량) 획득 기능을 추상화함.  [품질] QA-03(확장성)을 위해 GatewayController가 특정 구현체에 직접 의존하지 않도록 하여 유연성을 제공함. |
| IMQTTBroker | [역할] 클라우드 MQTT Message Broker와 비동기 메시지를 주고받기 위한 통신 인터페이스.  [기능] 메시지 발행(publish) 및 구독(subscribe) 기능을 정의함. |
| IDeviceMonitor | [역할] 연결된 디바이스의 상태 변화를 감시하고 GatewayController에 알리는 외부 컴포넌트(DeviceMonitor)와의 상호작용 인터페이스.  [기능] 디바이스 상태 변화를 구독하여 GatewayController가 실시간으로 인지하도록 함.  [품질] QA-01(성능) 및 QA-02(자원 효율성)을 위해 불필요한 폴링을 줄이고 이벤트 기반으로 동작함. |
| ILocalStorage | [역할] 게이트웨이 로컬 저장소에 데이터를 저장하고 조회하기 위한 추상화된 인터페이스.  [기능] 구체적인 저장 방식(파일 시스템, DB 등)에 독립적으로 데이터 접근 기능을 정의함.  [품질] QA-06(가용성)을 위한 데이터 영속성을 제공하는 기반이 됨. |
| HomeAssistantDeviceAdapter | [역할] IDeviceControl 인터페이스를 구현하며, HomeAssistantAPI의 기능을 IDeviceControl 표준에 맞춰 변환하는 Adapter. [기능] HomeAssistantAPI의 저수준 디바이스 제어 및 상태 조회 기능을 IDeviceControl의 표준화된 메서드로 매핑함. [품질] QA-03(확장성)을 위해 GatewayController와 HomeAssistantAPI 간의 인터페이스 불일치를 해결하고, 새로운 프로토콜/디바이스 추가 시 GatewayController의 변경을 최소화함. |
| HomeAssistantAPI | [역할] HomeAssistant 라이브러리의 실제 API를 추상화하거나 직접 호출하는 Adaptee. [기능] HomeAssistant의 저수준 디바이스 제어 서비스 호출 및 상태 조회 기능을 제공함. |
| ICommandQueue | [역할] ICommand 객체들을 저장하고 관리하는 큐의 동작을 정의하는 인터페이스.  [기능] Command 추가, Command 추출, 큐 비어있는지 확인 등의 메서드를 가짐. [품질] QA-06(가용성)을 위한 미처리 Command 객체들의 추상적인 저장소 역할을 정의함. |
| PersistentCommandQueue | [역할] ICommandQueue 인터페이스를 구현하는 구체적인 큐 구현체.  [기능] ILocalStorage를 사용하여 Command 객체들을 로컬에 영속적으로 저장하고 관리하며, 게이트웨이 재부팅 시에도 큐 내용을 복원함.  [품질] QA-06(가용성)을 위해 Command 유실 없이 로컬 기능 연속성을 보장하는 핵심 역할을 수행함. |
| ICommand | [역할] GatewayController가 수행할 수 있는 모든 작업을 캡슐화하는 추상화된 명령 인터페이스.  [기능] execute(), serialize(), deserialize()와 같은 메서드를 정의하여 명령의 실행 및 영속화를 위한 계약을 제시함.  [품질] QA-06(가용성)을 위해 Command 객체 자체를 영속화하여 네트워크 단절 시에도 명령을 보존할 수 있도록 함. |
| DeviceControlCommand | [역할] 특정 디바이스의 전원 제어(켜기/끄기) 명령을 캡슐화하는 구체적인 ICommand 클래스.  [기능] execute() 메서드에서 GatewayController의 IDeviceControl을 사용하여 실제 디바이스 제어 로직을 호출함.  [품질] QA-06(가용성)을 위해 네트워크 단절 시 미처리 디바이스 제어 요청을 보존하고 재실행을 가능하게 함. |
| ReportStatusCommand | [역할] 디바이스의 상태(전원, 전력 사용량)를 클라우드로 보고하는 명령을 캡슐화하는 구체적인 ICommand 클래스.  [기능] execute() 메서드에서 GatewayController의 IMQTTBroker를 사용하여 클라우드로 상태를 전송함.  [품질] QA-06(가용성)을 위해 네트워크 단절 시 미전송 상태 보고 요청을 보존하고 재전송을 가능하게 함. |
| FilePersistentStorage | [역할] ILocalStorage 인터페이스를 구현하는 구체적인 저장소 구현체.  [기능] 실제 파일 시스템을 사용하여 데이터를 저장하고 로드함. [품질] QA-06(가용성)을 위해 Command 객체들의 영속성을 확보하여 데이터 유실 방지 및 시스템 복구를 지원하는 핵심 요소임. |

#### Sequence Diagram

컴포넌트의 주요 기능인 기기제어와 네트워크 연결이 유실되었을 때의 기기제어를 시퀀스 다이어그램으로 표현하였다. 네트워크가 연결되었을 때 MQTT 커넥터를 통해서 command가 오는 경우와 네트워크가 끊긴 상태에서 내부에 저장된(ILocalStorage) 예약 명령이 실행되는 경우의 핵심 시퀀스를 나타내었다. 네트워크가 연결된 경우 MQTT Broker를 통해 명령을 수신하고 실행한다. 연결이 끊긴 상태에서 내부에 저장된 명령(ICommand)이 있는 경우 Local에 저장된 Command Object를 가져와 명령을 수신하고 실행한다. 기기제어는 IDeviceControl을 사용한다.

스크린샷, 도표, 텍스트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

## Design Description for *DeviceControlService*

### Overview

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

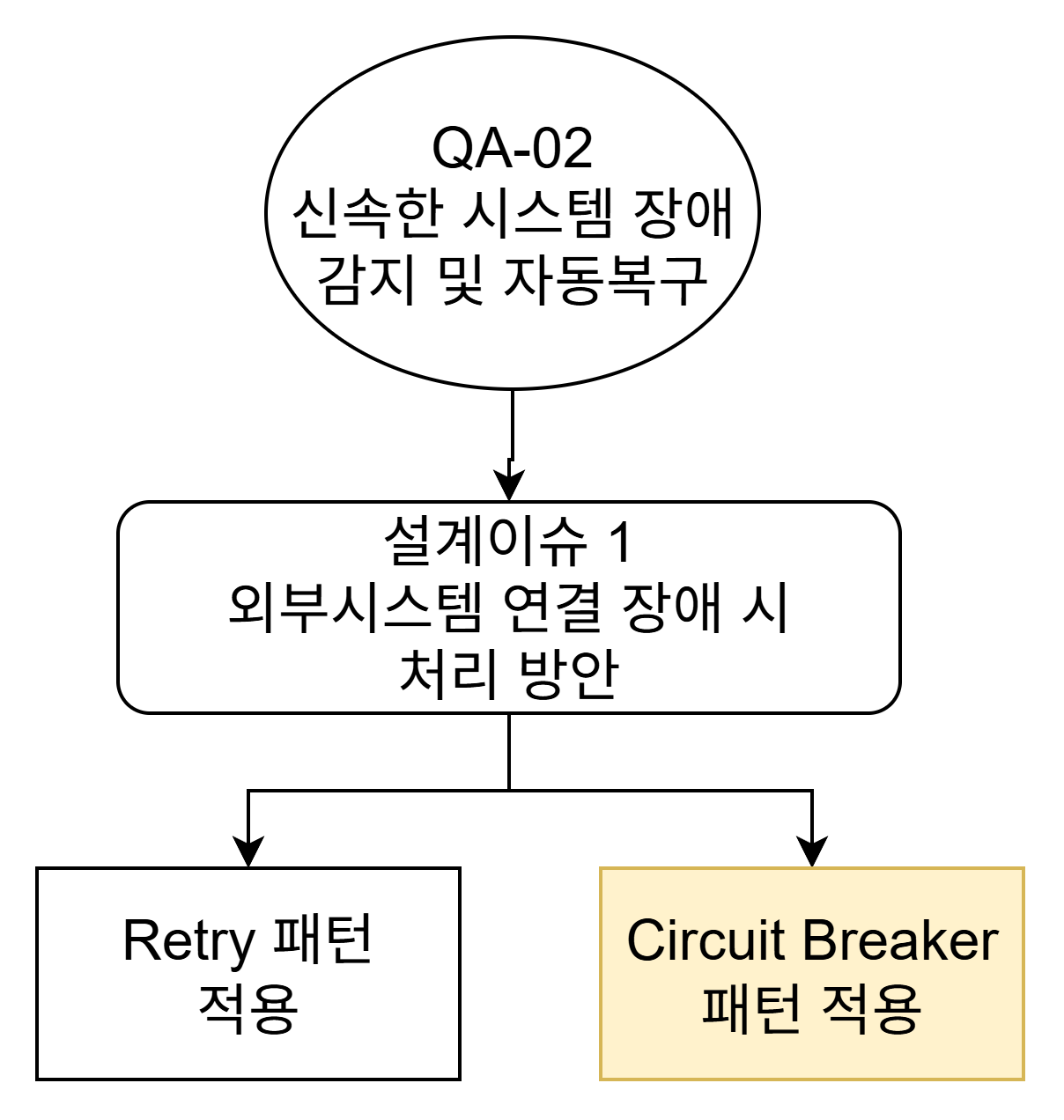
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 개요 | DeviceControlService는 사용자로부터의 원격 기기 제어 요청을 접수하여 처리하고, 그 결과를 Pub/Sub(MQTT) Broker를 통해 홈 게이트웨이와 사용자에게 전달하는 핵심 비즈니스 로직을 수행하는 컴포넌트이다. |
| 컴포넌트 기능 요구사항 | [Provided Interface]   * IDeviceControlService: API 게이트웨이로부터 라우팅된 사용자의 기기 제어 관련 HTTP 요청을 받아 처리하는 인터페이스.   [Required Interface]   * IMQTTBroker: 클라우드의 Message Broker와 비동기 메시지를 주고받기 위한 통신 인터페이스. * IDataService: 데이터베이스 및 캐시와 상호작용하기 위한 인터페이스. (예: 사용자 권한 확인, 기기 상태 업데이트 등) * IHeartbeatMonitor: 자신의 정상 동작 상태를 알리기 위한 인터페이스. |
| 컴포넌트 품질 요구사항  및 제약사항 | QA-01 (성능 - 원격 제어 응답 속도): 사용자 요청 접수 후, 내부 처리 및 Pub/Sub 발행까지의 과정이 매우 짧은 시간 안에 이루어져야 한다. 내부 TaskScheduler를 통해 긴급한 제어 명령을 우선적으로 처리하여 시스템 전체의 2초 내 응답 목표에 기여해야 한다.  QA-02 (가용성 - 신속한 장애 감지 및 복구): Heartbeat 택틱을 통해 자신의 상태를 주기적으로 보고해야 하며, Active Redundancy를 위해 다중 인스턴스로 배포되어 개별 인스턴스의 장애가 전체 서비스 중단으로 이어지지 않도록 설계되어야 한다. 이와 같은 인스턴스 레벨의 상위 장애 이외에도 일시적인 지연 등으로 인한 장애에도 대응하여 가용성을 만족시킨다.  QA-05 (확장성 - 사용자 증가 대응): 상태를 가지지 않는 Stateless 구조로 설계되어, LoadBalancer를 통해 수평적으로 쉽게 확장(Scale-out)할 수 있어야 한다.  AC-03 (클라우드 기반): 클라우드 환경(Kubernetes)에 컨테이너 형태로 배포되는 것을 전제로 설계되어야 한다. |

### Quality Driven Component Design

#### Exploring Design Options for *QA-01 신속한 장애 감지 및 복구 방안*

* + - * 1. **Quality Driven Component Design Structure**



#### Design Decisions for *설계이슈1 외부 시스템 연결 장애 시 처리 방안*

###### Definition for *설계이슈1 외부 시스템 연결 장애 시 처리 방안*

**배경 및 가정:** DeviceControlService는 IMQTTBroker를 통해 홈 게이트웨이와 통신하고, IDataService를 통해 사용자 권한 및 기기 상태 정보를 조회/갱신한다. 이들 외부 시스템은 독립적으로 운영되며, 일시적인 네트워크 문제, 과부하 등으로 인해 응답 지연이나 장애가 발생할 수 있다. 이러한 외부 시스템의 장애가 DeviceControlService의 정상 동작을 방해하거나, 반복적인 실패 요청으로 부하를 가중시켜 연쇄적인 장애를 유발할 수 있다.

**이슈 정의:** DeviceControlService가 의존하는 외부 시스템(MQTT Broker, DataService)의 일시적인 장애로부터 자신을 보호하고, 장애 발생 시 적절하게 대응하여 DeviceControlService의 가용성(QA-02)을 유지할 수 있는 구현 레벨의 디자인 옵션은 무엇인가?

###### Design Options for *설계이슈1 외부 시스템 연결 장애 시 처리 방안*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| Circuit Breaker 패턴 적용 | DeviceControlService가 외부 시스템(MQTT Broker, DataService)을 호출할 때 Circuit Breaker 객체를 사용하여 호출을 감싸는 방식. Circuit Breaker는 호출 실패율이 임계값을 초과하면 회로를 'Open' 상태로 전환하여 일정 시간 동안 모든 후속 호출을 즉시 실패시키고, 외부 시스템의 부하를 줄여 복구할 시간을 제공한다. 이후 일정 시간이 지나면 'Half-Open' 상태로 전환하여 소수의 요청만 허용하여 외부 시스템의 복구 여부를 확인한다. |
| Retry 패턴 적용 | 외부 시스템(MQTT Broker, DataService) 호출이 실패했을 때, 일정 시간 간격으로 미리 정의된 횟수만큼 재시도를 수행하는 방식. IRetriable 객체를 Retryer에 위임하여, Retryer가 설정된 횟수와 간격에 따라 호출 실패 시 자동으로 작업을 재시도하게 함으로써, 일시적인 외부 시스템 오류로부터의 복구력을 높인다. |

##### Decision and Rationale for *설계이슈1 외부 시스템 연결 장애 시 처리 방안*

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Circuit Breaker 패턴 적용 (Selected) | [장애 전파 방지] 외부 시스템의 장애가 DeviceControlService로 전파되어 연쇄적인 장애를 일으키는 것을 효과적으로 방지한다. 이는 DeviceControlService 자체의 가용성을 높이는 핵심적인 방법이다. [빠른 실패(Fail-Fast)] 외부 시스템이 이미 장애 상태임을 인지하고 불필요한 호출 시도를 막아, DeviceControlService의 응답 지연을 줄이고 자원 낭비를 방지한다. [자동 복구 시도] 회로 상태 관리를 통해 외부 시스템이 복구되었을 때 자동으로 다시 호출을 시도하여 서비스의 자율적인 회복력을 높인다. | [구현 복잡도] Circuit Breaker 라이브러리를 적용하고 각 외부 호출 지점에 통합해야 하므로 초기 구현에 약간의 복잡도가 있다. [설정의 중요성] 임계값, 재시도 간격 등 Circuit Breaker의 설정 파라미터를 환경에 맞게 신중하게 튜닝해야 한다. 잘못된 설정은 오히려 장애를 악화시킬 수 있다. | Circuit Breaker 패턴은 지속적인 장애 상황에 대해 DeviceControlService 자신을 보호하고, 외부 시스템에 회복할 시간을 주어 장애 전파를 근본적으로 차단하는 데 매우 효과적이다. 이는 DeviceControlService의 가용성(QA-02)을 확보하기 위한 필수적인 방어 메커니즘이다. 따라서, Circuit Breaker 패턴을 적용하여 IMQTTBroker 및 IDataService 호출 시 발생할 수 있는 외부 시스템 장애로부터 DeviceControlService를 보호하도록 한다.  \*구현 복잡도는 고도화된 솔루션에 대한 Trade-off로 판단함. Retry와 지수 백오프를 제한적으로 조합하여 복구력 높일 수 있음. |
| Retry 패턴 적용 | [일시적 오류 복구] 일시적인 네트워크 문제, 타임아웃, 리소스 부족 등으로 인한 순간적인 오류로부터 자동으로 복구할 수 있다. [구현의 용이성] 대부분의 클라이언트 라이브러리나 프레임워크에서 쉽게 설정하여 사용할 수 있다. | [지속적 장애에 취약] 외부 시스템이 완전히 장애 상태일 경우, 반복적인 재시도는 DeviceControlService에 불필요한 지연을 유발하고, 외부 시스템에 추가적인 부하를 가중시켜 장애 복구를 방해할 수 있다. [장애 전파 가능성] Circuit Breaker 없이 Retry만 사용할 경우, 외부 시스템의 장애가 DeviceControlService로 전파될 위험이 있다. | Retry 패턴은 일시적인 네트워크 끊김이나 타임아웃 같은 순간적인 오류에 적합하지만, 외부 시스템이 지속적으로 장애 상태일 때는 반복적인 재시도가 오히려 부하를 가중시키고 DeviceControlService의 자원을 소모시킨다. |

### Module View

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design Option Title | Reflection in the Module View |
| QA-05(가용성) | Circuit Breaker 패턴 | DeviceControlService 컴포넌트의 클래스 다이어그램에 적용된 Circuit Breaker 패턴은 모듈 뷰 관점에서 외부 시스템 의존성에 대한 DeviceControlService 모듈의 견고성 및 가용성을 크게 향상시킨다. DeviceControlService 모듈은 IMQTTBroker 및 IDataService라는 두 가지 주요 외부 인터페이스에 대한 의존성을 가진다. 일반적인 설계에서는 DeviceControlService가 이들 인터페이스의 구현체를 직접 호출하지만, Circuit Breaker 패턴의 적용으로 중간 계층(ICircuitBreaker 및 CircuitBreaker 구현체)이 삽입되었다. |

#### Class Diagram

이 클래스 다이어그램은 DeviceControlService 컴포넌트의 내부 구조와 외부 인터페이스 간의 상호작용을 포괄적으로 보여준다. DeviceControlService 는 시스템의 핵심 비즈니스 로직을 담당하며, 외부로부터의 디바이스 제어 요청을 처리하고 홈 게이트웨이와 통신한다. IDeviceControlService 인터페이스를 통해 외부 HTTPS 커넥터로부터 사용자 제어 요청을 수신하며, IMQTTBroker, IDataService, IHeartbeatMonitor 인터페이스를 사용하여 다른 시스템 서비스와 상호작용한다. 또한, Circuit Breaker 패턴이 DeviceControlService 내부에 통합되어 IMQTTBroker 및 IDataService와 같은 외부 의존성 서비스의 일시적인 장애로부터 DeviceControlService 자신을 보호하는 메커니즘을 상세히 표현하였다. 이는 시스템의 QA-02 (가용성) 목표를 달성하기 위한 핵심적인 방어 전략이다. TaskScheduler는 비동기 작업을 효율적으로 관리하여 QA-01 (성능) 요구사항을 지원하는 역할을 한다. 전반적으로, 이 다이어그램은 DeviceControlService가 클라우드 환경에서 기기 제어 명령을 안정적이고 효율적으로 처리하기 위한 설계 구조를 보여준다. \*CircuitState 등의 상태 객체에 대해서는 다이어그램의 복잡도로 인해 생략한다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

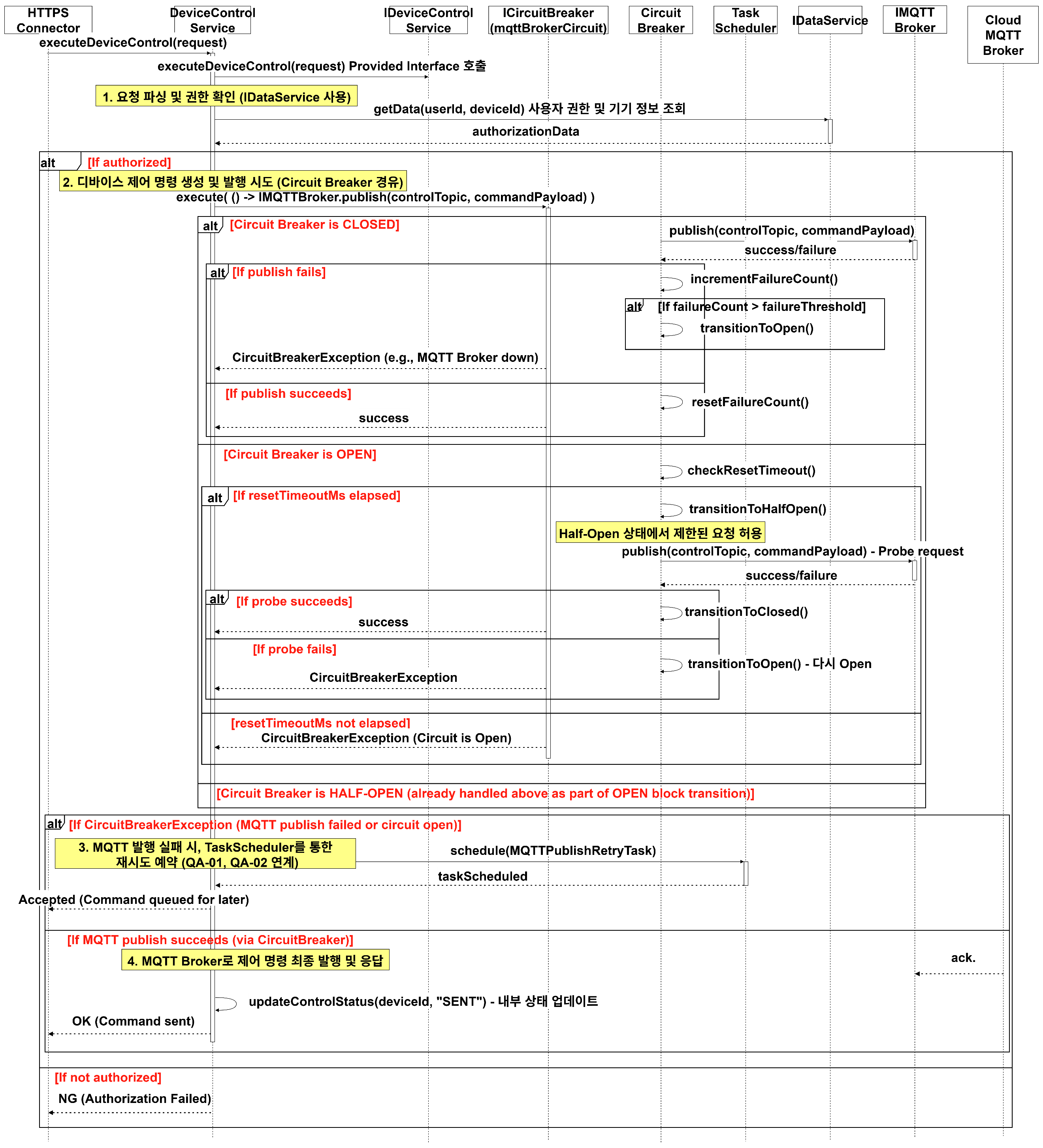
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

##### Element List

|  |  |
| --- | --- |
| Name | Description |
| DeviceControlService | [역할] 시스템의 핵심 비즈니스 로직을 수행하는 클라우드 서비스 컴포넌트임. 사용자로부터 기기 제어 명령을 받아 처리하고, 그 결과를 관리함. [기능] IDeviceControlService 인터페이스를 구현하여 외부 요청을 받아 executeDeviceControl을 실행함. processGatewayResponse로 게이트웨이 응답을 처리하고, reportHeartbeat로 자신의 상태를 보고하며, processScheduledTasks로 예약된 작업을 처리함. [품질] Circuit Breaker를 통해 외부 시스템(MQTT, DataService) 장애로부터 자신을 보호하여 QA-02 가용성을 확보함. TaskScheduler를 활용하여 QA-01 성능을 고려한 비동기 작업을 처리함. |
| ICircuitBreaker | [역할] Circuit Breaker 패턴의 동작 계약을 정의하는 인터페이스임. [기능] 특정 작업을 실행하는 execute(Callable operation) 메서드를 제공함. [품질] DeviceControlService가 외부 시스템 호출 시 장애 전파를 방지하고 빠른 실패를 유도하여 QA-02 가용성에 기여함. |
| CircuitBreaker | [역할] ICircuitBreaker 인터페이스를 구현하는 구체적인 Circuit Breaker 구현체임. [기능] state, failureThreshold, resetTimeoutMs 등의 내부 상태 및 설정을 관리하여 회로의 열림/닫힘/반열림 상태를 제어함. execute 메서드 호출 시 현재 상태에 따라 작업을 실행하거나 즉시 실패시킴. [품질] 외부 시스템의 장애로부터 DeviceControlService를 능동적으로 보호하며, 자동 복구 시도를 통해 QA-02 가용성을 높임. |
| IDeviceControlService | [역할] DeviceControlService가 외부에 제공하는 핵심 API 계약을 정의하는 Provided 인터페이스임. [기능] 사용자의 기기 제어 요청을 접수하는 executeDeviceControl(request) 메서드를 제공함. [품질] 컴포넌트의 외부 노출 기능을 표준화하여 QA-05 확장성에 기여함. |
| IMQTTBroker | [역할] 클라우드 MQTT Broker와 통신하기 위한 계약을 정의하는 Required 인터페이스임. [기능] 메시지를 발행하는 publish(topic, payload)와 구독하는 subscribe(topic) 메서드를 제공함. [품질] 비동기 메시징을 통해 QA-01 성능을 지원하며, Circuit Breaker의 보호를 받아 QA-02 가용성에 기여함. |
| IDataService | [역할] 시스템의 영속적인 데이터(DB, 캐시)에 접근하기 위한 계약을 정의하는 Required 인터페이스임. [기능] 데이터를 조회하는 getData(key)와 저장하는 saveData(key, data) 메서드를 제공함. [품질] AC-02 역할 기반 접근 제어에 필요한 권한 확인 데이터를 제공하며, Circuit Breaker의 보호를 받아 QA-02 가용성에 기여함. |
| IHeartbeatMonitor | [역할] 자신의 동작 상태를 외부에 보고하기 위한 계약을 정의하는 Required 인터페이스임. [기능] 서비스 ID와 상태를 인자로 받아 보고하는 report(serviceId, status) 메서드를 제공함. [품질] DeviceControlService 인스턴스의 장애를 외부에 신속하게 알림으로써 QA-02 가용성 확보에 필수적임. |
| TaskScheduler | [역할] 비동기적이고 지연될 수 있는 작업을 관리하는 내부 스케줄러 컴포넌트임. [기능] 특정 작업을 스케줄링하는 schedule(task)와 작업을 즉시 제출하고 Future를 반환하는 submit(task) 메서드를 제공함. [품질] DeviceControlService가 사용자 요청에 빠르게 응답하고 실제 작업은 백그라운드에서 처리하여 QA-01 성능 목표 달성에 기여함. |

#### Sequence Diagram

이 시퀀스 다이어그램은 DeviceControlService 컴포넌트가 외부 HTTPS Connector (API Gateway)로부터 디바이스 제어 요청을 수신하여 IDeviceControlService 인터페이스의 executeDeviceControl 오퍼레이션을 호출하는 과정부터, 최종적으로 Cloud MQTT Broker를 통해 제어 명령을 발행하는 핵심 흐름을 보여준다. 다이어그램은 특히 Circuit Breaker 패턴이 IMQTTBroker 호출에 어떻게 적용되어 외부 시스템 장애로부터 DeviceControlService를 보호하는지, 그리고 Circuit Breaker의 Closed, Open, Half-Open 상태 전환에 따른 동작 방식을 상세히 시각화한다. 또한, 권한 확인 (IDataService 활용), Circuit Breaker 실패 시 TaskScheduler를 통한 비동기 재시도 예약, 그리고 사용자 요청에 대한 OK 또는 Accepted 응답 처리 방식까지 포함하여 QA-01 (성능) 및 QA-02 (가용성) 요구사항이 컴포넌트 내부에서 어떻게 충족되는지를 구체적으로 기술하였다.



## Design Description for *DataService*

### Overview

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

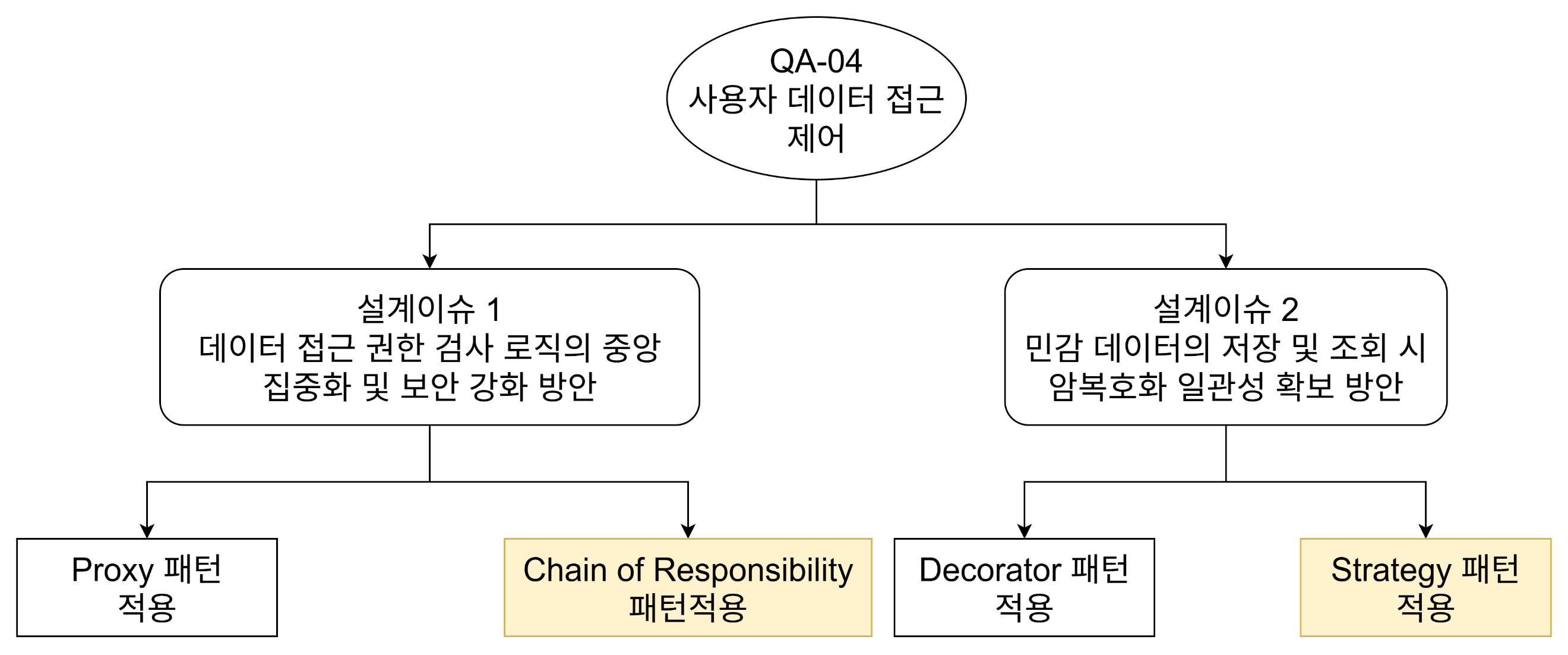
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 개요 | 시스템 내 모든 영속적인 데이터(사용자 정보, 디바이스 상태, 제어 로그 등)의 저장, 조회, 갱신을 담당하는 핵심 컴포넌트이다. 다양한 서비스 컴포넌트로부터 데이터 접근 요청을 처리하고, 캐시 및 데이터베이스 시스템과 상호작용한다. |
| 컴포넌트 기능 요구사항 | [Provided Interface]   * IDataService: 사용자 정보, 디바이스 정보, 제어 이력 등 시스템의 모든 데이터에 대한 조회 및 저장 오퍼레이션을 제공하는 인터페이스.   [Required Interface]   * ICache: 데이터 조회 성능 향상을 위한 인메모리 캐시 시스템과 상호작용하기 위한 인터페이스. (예: Redis) * IMasterDatabase: 데이터 쓰기/갱신 작업을 처리하는 Master 데이터베이스 시스템과 상호작용하기 위한 인터페이스. * ISlaveDatabase: 데이터 읽기 작업을 처리하고 확장성 및 가용성을 높이는 Slave 데이터베이스 시스템과 상호작용하기 위한 인터페이스. * IHeartbeatMonitor: 자신의 인스턴스 상태를 외부에 주기적으로 보고하기 위한 인터페이스. * IPowerCompanyAPI: 전력 공급사로부터 데이터를 연동하거나 요금제 등의 정보를 조회하기 위한 외부 API 인터페이스. |
| 컴포넌트 품질 요구사항  및 제약사항 | QA-01 (성능 - 데이터 조회/갱신 속도): 다양한 서비스 컴포넌트로부터 발생하는 대량의 데이터 조회 및 갱신 요청을 신속하게 처리하여 전체 시스템의 응답 속도를 저해하지 않아야 한다.  QA-02 (가용성 - 데이터 접근의 지속성): 데이터베이스 또는 캐시 시스템의 일시적인 장애에도 데이터 접근이 완전히 중단되지 않거나, 신속하게 복구될 수 있어야 한다.  QA-04 (보안 – 데이터 접근 제어): 데이터베이스 또는 캐시 시스템의 접근에 대해 비인가된 접근을 막고 보안 강화해야 한다.  AC-02 (개인정보보호) 데이터 접근은 역할 기반 접근 제어(RBAC)를 통해 엄격히 통제되어야 한다. |

### Quality Driven Component Design

#### Exploring Design Options for *QA-04 사용자 데이터 접근 제어*

* + - * 1. **Quality Driven Component Design Structure**



#### Design Decisions for *설계이슈1 데이터 접근 권한 검사 로직의 중앙 집중화 및 보안강화방안*

###### Definition for *설계이슈1 데이터 접근 권한 검사 로직의 중앙 집중화 및 보안 강화 방안*

**배경 및 가정:** DataService는 다양한 서비스(예: DeviceControlService, PredictionService)로부터 데이터 접근 요청을 받는다. 각 요청에 대해 요청자의 역할이나 접근하려는 데이터의 종류에 따라 세밀한 권한 검사(Authorization)가 이루어져야 한다. 이 권한 검사 로직이 DataService의 각 메서드에 분산되면 중복 코드가 발생하고, 보안 정책 변경 시 수정이 어려워 보안 취약점으로 이어질 수 있다.

**이슈 정의:** DataService로 들어오는 모든 데이터 접근 요청에 대해, 중앙 집중적이고 일관된 방식으로 다단계 권한 검사 로직을 적용하여 QA-04 (보안)을 강화하고, 권한 정책 변경에 유연하게 대응할 수 있는 방안은 무엇인가?

###### Design Options for *설계이슈1*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| Chain of Responsibility 패턴 적용 | IAuthorizationHandler 인터페이스를 정의하고, 이를 구현하는 AuthenticationHandler, RoleBasedAccessControlHandler, ResourceOwnershipHandler 등의 핸들러 클래스를 구현하는 방식. DataService는 데이터 접근 요청을 수신하면, 이 핸들러 체인의 시작점에 요청을 전달하고, 각 핸들러는 자신의 권한 검사 책임을 수행한 후 다음 핸들러로 요청을 넘기거나, 검사 실패 시 요청을 거부한다. |
| Proxy 패턴 (접근 제어 프록시) | IDataService 인터페이스를 구현하는 DataServiceSecurityProxy 클래스를 구현하는 방식. 클라이언트는 DataServiceSecurityProxy를 통해 IDataService에 접근하며, Proxy는 실제 DataService 구현체를 호출하기 전에 인증(Authentication) 및 권한(Authorization) 검사를 수행한다. |

##### Decision and Rationale for *설계이슈1*

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Chain of Responsibility 패턴 적용 (Selected) | [중앙 집중적 권한 검사 (QA-04)] 모든 데이터 접근 요청이 일련의 핸들러 체인을 거치도록 하여, 권한 검사 로직을 중앙 집중화하고 일관된 보안 정책을 강제한다. 이는 QA-04 (보안)을 효과적으로 강화한다. [유연한 정책 변경 및 확장] 새로운 권한 검사 조건이 추가되거나 기존 검사 순서가 변경되어야 할 때, 핸들러 체인만 수정하거나 새로운 핸들러를 추가/제거하면 되므로 DataService의 핵심 로직 변경 없이 QA-05 (확장성)에도 기여한다. [책임 분리] 각 핸들러는 단일 권한 검사 책임만 가지므로, 코드의 응집도가 높고 유지보수가 용이하다. | [처리 보장 불가] 요청이 체인 끝까지 도달해도 처리되지 않을 수 있으므로, 최종 처리 핸들러 또는 기본 처리 로직에 대한 명확한 정의가 필요하다. [성능 오버헤드] 체인의 길이가 길어질수록 각 요청이 거쳐야 하는 핸들러의 수가 늘어나 미미한 성능 오버헤드가 발생할 수 있다. | Proxy 패턴은 주로 특정 객체에 대한 접근을 제어(예: 인증 후 접근)하는 단일 관점에 강하지만, DataService의 복잡하고 다단계적인 데이터 접근 권한 검사 로직을 관리하는 데는 Chain of Responsibility 패턴이 더 적합하다. 이 패턴은 AC-02 (역할 기반 접근 제어)와 같은 복합적인 보안 정책을 구현 레벨에서 유연하고 확장 가능하게 적용하여 QA-04 (보안)을 직접적으로 강화한다. 각 권한 검사 단계를 명확히 분리함으로써 코드의 재사용성과 유지보수성도 높인다.  \*성능 오버헤드는 보안성의 trade-off로 판단하되 너무 chain이 길어지지 않도록 한다. 처리를 보장하기 위해 chain 마지막에 default Handler를 두어 요청이 항상 종결되도록 한다. |
| Proxy 패턴 (접근 제어 프록시) | [접근 제어 중앙화 (QA-04)] Proxy가 DataService로의 모든 접근 요청에 대한 인증 및 권한 검사 지점 역할을 수행하여 QA-04 (보안)을 강화한다. 클라이언트는 보안 로직을 알 필요가 없음. [클라이언트 투명성] 클라이언트는 IDataService 인터페이스만 알면 되므로, 보안 로직의 존재가 클라이언트에게 투명하다. | [단일 책임의 한계] Proxy가 모든 보안 로직을 담당하게 되면 Proxy 클래스 자체의 책임이 커지고 복잡해질 수 있다. 다단계적이고 복잡한 권한 검사 로직을 구현할 때는 Chain of Responsibility 패턴만큼 유연하지 않을 수 있다. [확장성 제한] 새로운 권한 검사 단계를 추가하거나 기존 검사 순서를 변경할 때 Proxy 클래스 자체를 수정해야 할 가능성이 높다. | Proxy 패턴 역시 DataService에 대한 접근을 제어하여 QA-04 (보안)을 강화하는 효과적인 방법이다. 특히 외부로부터의 모든 IDataService 호출에 대해 일차적으로 보안 검사를 적용하는 진입점 역할에 강력하다. 하지만 다단계적인 복합 권한 검사보다는 기본적인 인증/인가 검사에 더 적합하며, 더 세분화된 검사는 다른 패턴과 조합될 필요가 있다. |

#### Design Decisions for *설계이슈2 민감 데이터의 저장 및 조회 시 암복호화 일관성 확보 방안*

###### Definition for *설계이슈2 민감 데이터의 저장 및 조회 시 암복호화 일관성 확보 방안*

**배경 및 가정**: DataService는 사용자 개인 정보, 디바이스 제어 민감 정보 등 중요한 데이터를 다룬다. 상위 아키텍처에서 데이터 암호화가 결정되었으므로, IDatabase를 통해 데이터베이스에 저장/조회될 때 이 민감 정보들이 항상 일관된 방식으로 암호화되고 복호화되어야 한다. 암복호화 로직이 여러 곳에 분산되면 오류 발생 가능성이 높고 보안 취약점으로 이어질 수 있다.

**이슈 정의**: DataService 내부에서 민감 데이터의 저장(암호화) 및 조회(복호화) 시, 특정 데이터 필드에 대해 암복호화 로직을 일관되고 유연하게 적용하여 QA-04 (보안)을 강화할 수 있는 방안은 무엇인가?

###### Design Options for *설계이슈2*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
| Strategy 패턴 적용 | IEncryptionStrategy 인터페이스를 정의하고, 이를 구현하는 AES256EncryptionStrategy, NoEncryptionStrategy 등의 클래스를 구현하는 방식. DataService는 민감 데이터 필드 처리 시 적절한 IEncryptionStrategy 구현체를 선택하여 암호화/복호화를 수행한다. |
| Decorator 패턴 (암복호화 기능 투명성) | IDataService 인터페이스를 구현하는 EncryptedDataServiceDecorator 클래스를 구현하여, 기존 IDataService 구현체에 암복호화 기능을 동적으로 추가하는 방식. 클라이언트는 Decorator를 IDataService 구현체로 인식하고 호출한다. |

##### Decision and Rationale for *설계이슈2*

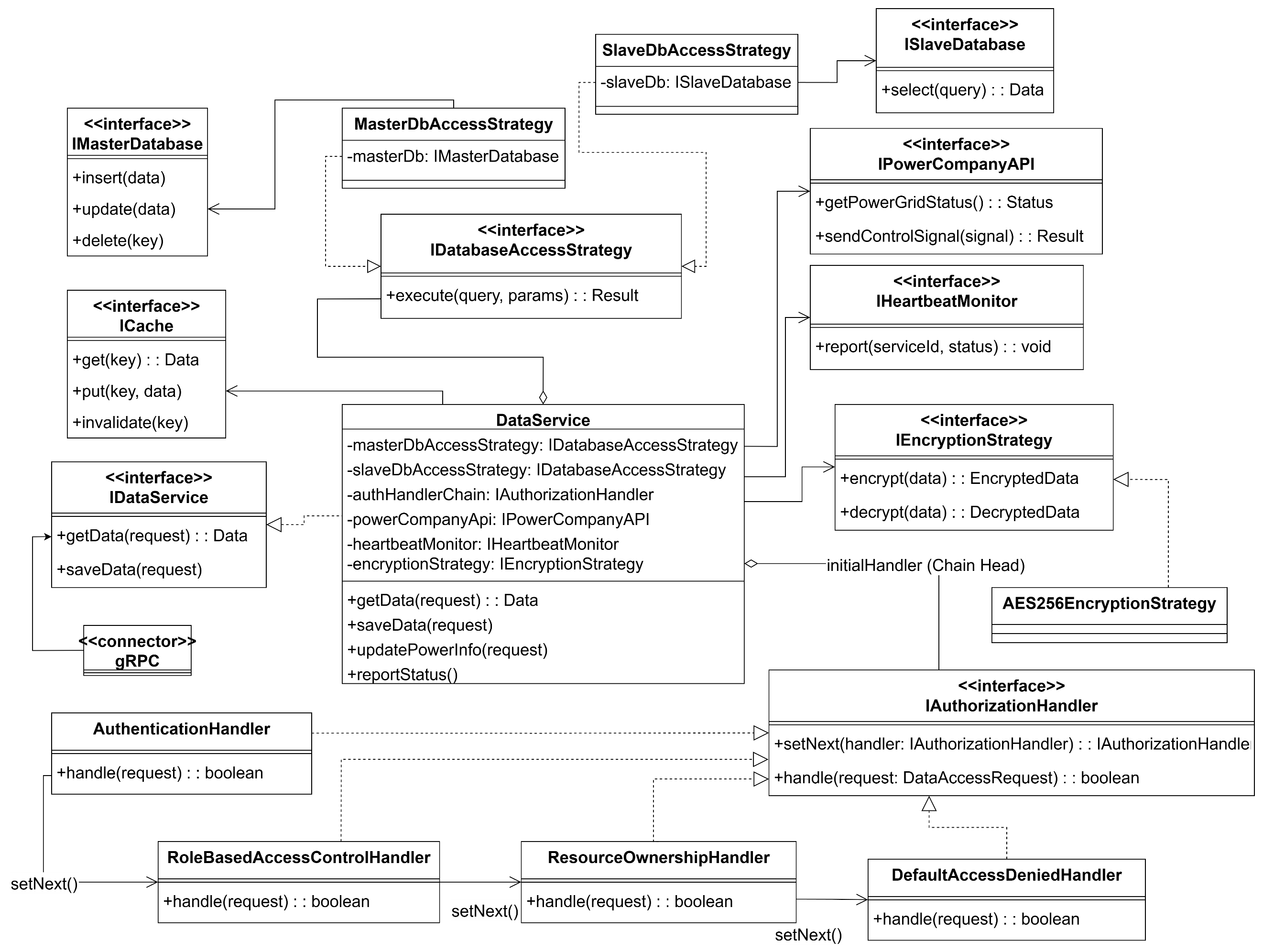
|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Strategy 패턴 적용 (Selected) | [유연한 암복호화 정책] 암호화 알고리즘(AES256, RSA 등)이나 암복호화 방식을 DataService의 핵심 로직 변경 없이 동적으로 교체하거나 확장할 수 있다. 이는 향후 보안 요구사항 변경에 유연하게 대응하여 QA-04 (보안)의 적응성을 높인다. [책임 분리] 암복호화 로직을 별도의 Strategy 객체에 캡슐화함으로써, DataService의 데이터 처리 로직과 보안 로직의 결합도를 낮추고 응집도를 높인다. [일관성 유지] 특정 데이터 타입이나 필드에 대해 어떤 Strategy를 적용할지 DataService가 명확히 결정함으로써, 암복호화 적용의 일관성을 확보할 수 있다. | [클라이언트의 전략 선택 책임] DataService가 어떤 Strategy를 사용할지 판단하고 선택하는 책임이 존재하므로, 초기 구현 시 전략 선택 로직을 명확히 정의해야 한다. [클래스 증가] 각 암호화 방식별로 별도의 Strategy 클래스를 정의해야 할 수 있다. | Decorator 패턴은 기존 기능에 암복호화 기능을 추가하는 데 유용하지만, 특정 데이터 필드에 대한 '암복호화 방식 자체'를 유연하게 변경하고 싶을 때는 Strategy 패턴이 더 적합하다. Strategy를 통해 DataService는 데이터의 민감도에 따라 가장 적절한 암호화 알고리즘을 선택, 적용함으로써 QA-04 (보안)을 직접적으로 강화하고, 향후 보안 표준 변화에도 유연하게 대처할 수 있다. |
| Decorator 패턴 (암복호화 기능 투명성) | [투명한 보안 기능 추가] DataService의 핵심 비즈니스 로직을 변경하지 않고도, 데이터 저장 시 암호화, 조회 시 복호화 기능을 동적으로 투명하게 추가할 수 있다. 이는 QA-04 (보안)을 유지보수성 높게 강화한다. [기능 확장 용이성] 암호화 외에 추가적인 보안 기능(예: 데이터 마스킹, 무결성 검증)을 다른 Decorator로 쉽게 조합하여 적용할 수 있다. | [객체 래핑의 깊이] 여러 Decorator를 적용할 경우 객체 래핑의 깊이가 깊어져 디버깅이 복잡해질 수 있다. [알고리즘 유연성 제한] Decorator 자체는 어떤 암호화 알고리즘을 사용할지 Strategy처럼 유연하게 교체하는 책임을 직접적으로 가지지 않으므로, 알고리즘 변경 시 Decorator 내부 또는 하위 모듈 수정이 필요할 수 있다. | Decorator 패턴 역시 IDataService에 암복호화 기능을 추가하여 QA-04 (보안)을 강화하는 좋은 방법이다. 특히, Decorator는 기존 코드의 변경 없이 보안 기능을 주입할 수 있다는 점에서 유연하며, 다양한 보안 관련 부가 기능들을 조합하여 적용해야 할 경우에 강점을 가진다. |

### Module View

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design Option Title | Reflection in the Module View |
| QA-04  (보안) | Chain of Responsibility 패턴 | [반영 방식]DataService 클래스는 authHandlerChain: IAuthorizationHandler 필드를 통해 권한 검사 체인의 시작점(헤드)을 참조한다. 이 체인은 IAuthorizationHandler 인터페이스를 구현하는 AuthenticationHandler, RoleBasedAccessControlHandler, ResourceOwnershipHandler, 그리고 최종 DefaultAccessDeniedHandler 클래스들이 setNext() 메서드를 통해 순차적으로 연결된 형태로 구성된다.  [효과] 모든 데이터 접근 요청은 이 핸들러 체인을 순차적으로 거치며 다단계 권한 검사를 중앙 집중적이고 일관된 방식으로 적용한다. 이는 QA-04 (보안)을 강력하게 확보하며, 각 핸들러가 단일 책임(인증, 역할 기반 접근, 소유권 확인 등)을 가지므로 보안 정책의 유연한 변경 및 확장을 가능하게 한다. 특히 DefaultAccessDeniedHandler는 요청이 체인을 통과하지 못했을 때 최종적으로 처리(거부 또는 예외 발생)를 보장함으로써 "처리 보장 불가" 단점을 보완한다. |
| Strategy 패턴 | [반영 방식]DataService 클래스는 masterDbAccessStrategy: IDatabaseAccessStrategy, slaveDbAccessStrategy: IDatabaseAccessStrategy 필드를 통해 데이터베이스 접근 전략을, 그리고 encryptionStrategy: IEncryptionStrategy 필드를 통해 암복호화 전략을 주입받아 사용한다. 각 필드는 IDatabaseAccessStrategy (구현체: MasterDbAccessStrategy, SlaveDbAccessStrategy)와 IEncryptionStrategy (구현체: AES256EncryptionStrategy) 인터페이스를 참조한다.  [효과] 이를 통해 DataService는 데이터 조회/갱신 시점에 따라 Master 또는 Slave 데이터베이스 접근 방식을 동적으로 선택하여 QA-01 (성능)을 위한 Read/Write Splitting을 효율적으로 구현한다. 또한, 민감 데이터 저장/조회 시 암복호화 방식을 유연하게 변경할 수 있게 하여 QA-04 (보안)을 강화하고 향후 보안 정책 변경에 용이하게 대응한다. |

#### Class Diagram

이 클래스 다이어그램은 DataService 컴포넌트의 내부 구조, 외부 인터페이스, 그리고 QA-01 (성능) 및 QA-04 (보안)을 강화하기 위해 적용된 디자인 패턴을 상세하게 보여준다. DataService는 시스템 내 모든 영속적인 데이터의 중앙 관리자로서, 다양한 서비스 컴포넌트로부터 데이터 접근 요청을 처리하고 전력 공급사 API와 연동하였다.

**Strategy 패턴**을 적용하여 및 민감 데이터 암복호화 방식을 유연하게 관리하여 성능과 보안을 동시에 강화하였다. **Chain of Responsibility 패턴**을 사용하여 모든 데이터 접근 요청에 대한 다단계 권한 검사 (Authentication, Role-Based Access Control, Resource Ownership)를 중앙 집중적이고 확장 가능한 방식으로 처리하여 QA-04 (보안)을 견고하게 확보하는 등 DataService가 높은 성능으로 데이터를 제공하면서도 중요한 데이터를 안전하게 보호하고 관리하는 견고하고 유연한 구조를 가지도록 하였다.

##### Element List

|  |  |
| --- | --- |
| Name | Description |
| DataService | [역할] 시스템의 모든 영속적인 데이터를 관리하고 제공하는 핵심 서비스 컴포넌트. 다양한 클라이언트의 데이터 접근 요청을 처리하며, 내부적으로 보안 및 성능 최적화 로직을 수행. [기능] IDataService 인터페이스를 구현하여 getData, saveData 등의 핵심 데이터 오퍼레이션을 제공함. updatePowerInfo를 통해 외부 전력 정보를 갱신하고, reportStatus로 자신의 상태를 보고함. 내부적으로 IDatabaseAccessStrategy, IAuthorizationHandler 체인, IEncryptionStrategy 등을 활용함. [품질] Strategy 패턴을 통해 QA-01 성능 및 QA-04 보안을 확보함. Chain of Responsibility를 통해 QA-04 보안의 권한 검사를 중앙 집중화함. |
| IDataService | [역할] DataService가 외부에 제공하는 데이터 접근 API의 계약을 정의하는 Provided 인터페이스임. [기능] getData(request)로 데이터를 조회하고 saveData(request)로 데이터를 저장하는 메서드를 제공함. [품질] PredictionService, DeviceControlService 등 클라이언트 컴포넌트와의 표준화된 인터페이스를 제공하여 QA-05 확장성에 기여함. |
| ICache | [역할] 데이터 조회 성능 향상을 위한 인메모리 캐시 시스템과의 통신 계약을 정의하는 Required 인터페이스임. [기능] get(key)으로 캐시에서 데이터를 조회하고, put(key, data)로 데이터를 저장하며, invalidate(key)로 캐시를 무효화하는 메서드를 제공함. [품질] QA-01 성능을 위해 데이터베이스 부하를 줄이고 빠른 응답을 가능하게 함. |
| IMasterDatabase | [역할] 데이터 쓰기(Write) 작업을 처리하는 Master 데이터베이스 시스템과의 통신 계약을 정의하는 Required 인터페이스임. [기능] insert(data), update(data), delete(key) 등 데이터 변경 오퍼레이션을 제공함. [품질] IDatabaseAccessStrategy와 함께 QA-01 성능을 위한 Read/Write Splitting의 쓰기 전용 통로 역할을 함. QA-02 가용성을 위해 Failover 대상이 됨. |
| ISlaveDatabase | [역할] 데이터 읽기(Read) 작업을 처리하는 Slave 데이터베이스 시스템과의 통신 계약을 정의하는 Required 인터페이스임. [기능] select(query) 등 데이터 조회 오퍼레이션을 제공함. [품질] IDatabaseAccessStrategy와 함께 QA-01 성능을 위한 Read/Write Splitting의 읽기 전용 통로 역할을 함. 읽기 확장성을 제공하여 QA-05 확장성에 기여함. |
| IHeartbeatMonitor | [역할] 자신의 인스턴스 상태를 외부에 주기적으로 보고하기 위한 Required 인터페이스임. [기능] report(serviceId, status)를 통해 DataService 인스턴스의 현재 상태를 모니터링 시스템에 알림. [품질] QA-02 가용성을 위해 DataService 인스턴스의 장애를 신속하게 감지하고 대응하는 데 필수적임. |
| IPowerCompanyAPI | [역할] 외부 전력 공급사 시스템과 연동하기 위한 Required 인터페이스임. [기능] getPowerGridStatus()로 전력망 상태를 조회하고, sendControlSignal(signal)로 전력 제어 명령을 전달하는 메서드를 제공함. |
| IEncryptionStrategy | [역할] 민감 데이터의 암호화 및 복호화 방식을 추상화하는 Strategy 패턴의 인터페이스임. [기능] encrypt(data)로 데이터를 암호화하고, decrypt(data)로 데이터를 복호화하는 메서드를 제공함. [품질] 다양한 암호화 알고리즘을 유연하게 교체할 수 있게 하여 QA-04 보안의 적응성을 높임. |
| AES256EncryptionStrategy | [역할] IEncryptionStrategy 인터페이스를 구현하는 구체적인 암복호화 전략임. [기능] AES256 알고리즘을 사용하여 데이터를 암호화 및 복호화함. [품질] QA-04 보안의 요구사항을 충족하는 강력한 암호화 표준을 제공함. |
| IAuthorizationHandler | [역할] Chain of Responsibility 패턴에서 권한 검사 단계를 추상화하는 인터페이스임. [기능] setNext(handler)로 다음 핸들러를 설정하고, handle(request)로 권한 검사 로직을 수행함. [품질] 다단계 권한 검사 로직을 유연하게 구성하여 QA-04 보안을 강화함. |
| AuthenticationHandler | [역할] IAuthorizationHandler를 구현하는 권한 검사 체인의 첫 번째 핸들러임. [기능] handle(request) 메서드에서 요청자의 인증 상태를 검사함. [품질] QA-04 보안을 위해 모든 데이터 접근 요청에 대한 기본적인 인증 여부를 확인함. |
| RoleBasedAccessControlHandler | [역할] IAuthorizationHandler를 구현하는 역할 기반 접근 제어 핸들러임. [기능] handle(request) 메서드에서 요청자의 역할(Role)에 기반한 데이터 접근 권한을 검사함. [품질] AC-02 (역할 기반 접근 제어)를 구현하여 QA-04 보안을 강화함. |
| ResourceOwnershipHandler | [역할] IAuthorizationHandler를 구현하는 리소스 소유권 검사 핸들러임. [기능] handle(request) 메서드에서 요청자가 접근하려는 데이터/리소스의 소유자인지 검사함. |
| DefaultAccessDeniedHandler | [역할] IAuthorizationHandler를 구현하는 권한 검사 체인의 마지막 핸들러임. [기능] handle(request) 메서드에서 이전 핸들러들이 처리하지 못한 요청을 최종적으로 거부하거나 예외를 발생시킴. [품질] QA-04 보안을 위해 모든 요청이 무시되지 않고 항상 보안 검사를 통해 종결되도록 처리 보장을 제공함. |
| IDatabaseAccessStrategy | [역할] 데이터베이스 접근 방식을 추상화하는 Strategy 패턴의 인터페이스임. [기능] execute(query, params) 메서드를 통해 데이터베이스 쿼리를 실행함. [품질] QA-01 성능을 위한 Master/Slave DB 분리 로직을 유연하게 교체할 수 있게 함. |
| MasterDbAccessStrategy | [역할] IDatabaseAccessStrategy를 구현하는 Master DB 접근 전략임. [기능] IMasterDatabase를 통해 데이터 쓰기(insert, update, delete) 작업을 수행함. [품질] QA-01 성능을 위한 Read/Write Splitting에서 쓰기 전용 경로를 담당함. |
| SlaveDbAccessStrategy | [역할] IDatabaseAccessStrategy를 구현하는 Slave DB 접근 전략임. [기능] ISlaveDatabase를 통해 데이터 읽기(select) 작업을 수행함. [품질] QA-01 성능을 위한 Read/Write Splitting에서 읽기 전용 경로를 담당함. |

#### Sequence Diagram

이 시퀀스 다이어그램은 gRPC 커넥터로부터 DataService의 getData(request) 오퍼레이션이 호출될 때, 내부적으로 Chain of Responsibility를 이용한 권한 검사, IEncryptionStrategy를 통한 데이터 복호화, 그리고 IDatabaseAccessStrategy를 이용한 ISlaveDatabase로부터의 데이터 조회 과정을 보여준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

# Architectural Evaluation

본 장에서는 설계된 홈 IoT 시스템의 아키텍처가 주요 품질 요구사항(QA)을 얼마나 잘 만족하는지 ATAM(Architecture Trade-off Analysis Method)의 원칙에 따라 평가한다. 3.2장에서 산정된 BV(Business Value)와 AI(Architectural Impact)를 종합적으로 고려하여 QA-01 (성능), QA-02 (가용성), QA-05 (확장성) 세 가지 핵심 품질 속성에 대한 평가를 수행한다.

## Architectural Evaluation for *QA-01 원격 제어 응답 속도 (성능)*

* **품질 속성 중요도 (BV): 최상**
* **아키텍처 영향도 (AI): 최상**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| QA 시나리오 | 사용자가 원격지에서 가전기기 제어를 요청했을 때, 시스템이 해당 명령을 2초 이내에 기기까지 전달하여 상태를 변경하는 시나리오. | | | | |
| 품질속성 | 성능 | | | | |
| 자극 소스 | 사용자 | | | | |
| 자극 | 가전기기 제어 요청 | | | | |
| 환경 | 시스템이 정상 운영 중인 상태. 클라우드 플랫폼은 전체 사용자(30만)의 5%가 동시 접속하여 API 요청을 보내고 있는 평균 부하 상태. | | | | |
| 응답 | 시스템은 제어 명령을 순차적으로 처리하여 최종적으로 가전기기의 물리적인 상태를 변경하고, 변경된 결과를 웹/앱에 업데이트하여 표시. | | | | |
| 측정 | 사용자가 웹/앱에서 제어 명령을 발생시킨 시점(Stimulus)부터, 해당 기기의 변경된 상태가 웹/앱 화면에 표시될 때(Response)까지의 End-to-End Latency를 측정하여, 95% 이상의 요청이 2초 이내에 처리되어야 함. | | | | |
| 아키텍처 결정 | | 위험요소 | 민감점 | 절충점 | 비위험요소 |
| Publish-Subscribe 스타일 (MQTT) | | R1 | S1 | T1 |  |
| Introduce Concurrency 기반 Prioritize Events 택틱 + Schedule Resources택틱 | |  | S2 |  | NR1 |
| Command 패턴 | |  | S3 |  | NR2 |
| Reasoning | **[Publish-Subscribe 스타일 (MQTT)]** 클라이언트-게이트웨이-디바이스 간의 효율적인 비동기 메시지 전달 메커니즘을 제공, 제어 명령이 빠르게 전송되고 디바이스 상태 변경이 신속하게 보고될 수 있도록 함. 이는 Latency 단축의 핵심 기반이 됨.  **[Introduce Concurrency 기반 Prioritize Events 택틱 + Schedule Resources 택틱]** 대량의 동시 제어 요청이 발생하는 환경에서, 명령을 병렬로 처리하고, 특히 중요도가 높은 제어 명령에 우선순위를 부여함으로써 병목 현상을 줄이고 핵심 명령의 응답 속도 목표(2초 이내)를 보장함. 또한 기아현상 방지를 위해 Schedule Resources 택틱을 함께 적용하여 우선순위가 낮은 명령 또한 일정 비율로 처리가 될 수 있도록 보장함.  **[Command 패턴 적용]** 클라우드로부터 수신된 제어 명령을 Command 객체로 캡슐화하여 비동기적으로 실행하고, 게이트웨이 내부에 큐를 통해 명령을 보존함. 이는 명령 수신 즉시 클라우드로 응답을 보낼 수 있게 하여 클라이언트가 체감하는 응답 속도를 개선하며, 게이트웨이-디바이스 간의 물리적 지연에 독립적으로 동작하여 성능에 긍정적인 영향을 줌. | | | | |

### List of Risks

* R1 : Broker 장애 시 명령 전달 지연 또는 유실 가능성 있음.
  + 대처 방안 : Broker 클러스터링을 통해 여러 Broker 인스턴스를 클러스터로 묶어 고가용성 확보.
  + 장점 : Broker 단일 장애 지점 제거를 통해 가용성을 향상. 여러 인스턴스로 부하가 분산되어 성능의 Throughput 증가.
  + 단점 : 클러스터 구성 및 관리 복잡도가 증가하고 비용효율성이 떨어지게 됨.
  + 실행 가능성 : 높음. 대부분의 MQTT Broker는 고가용성(HA) 모드 구성을 지원함.
  + 영향 평가 : 부하가 분산되어 정상 상태에서도 응답 속도에 기여, 장애 시 명령 전달 지연 및 유실에 대해 보완함.

### List of Sensitivities

* S1 : MQTT Broker의 메시지 처리 속도 및 메시지 큐 깊이에 따라 응답 속도가 크게 달라짐
* S2 : 이벤트 우선순위 분류 로직의 복잡성 및 처리 속도에 따라 중요 명령의 응답 속도가 결정됨.
* S3 : Circuit Breaker의 임계값(Threshold) 및 재시도 정책에 따라 외부 서비스 장애 발생 시 시스템의 응답성과 안정성이 결정됨.
* S4 : GatewayController의 Command 큐 크기 및 Command 실행 스케줄링 정책에 따라 디바이스 제어의 최종 응답 속도가 달라짐.

### List of Tradeoffs

* T1 : Pub/Sub(MQTT)를 사용함으로써 실시간성은 얻었지만, 메시지 전달의 신뢰성(Reliability)을 일부 희생했다. MQTT의 QoS(Quality of Service) 레벨에 따라 성능과 신뢰도는 반비례 관계에 놓인다. QoS 0는 가장 빠르지만 메시지 유실이 가능하고, QoS 2는 반드시 전달되지만 느리다. 성능을 위해 QoS 1을 채택할 경우, 이로 인해 극히 드문 경우 메시지가 중복 전달될 수 있는 위험을 감수했다.

### List of Nonrisks

* NR1 : 단일 스레드/순차 처리 대비 다수의 동시 제어 명령을 빠르게 처리 가능함.
* NR2: GatewayController는 디바이스 제어의 실제 물리적 지연과 관계없이 클라이언트가 느끼는 응답 속도를 개선함.

## Architectural Evaluation for *QA-02 신속한 시스템 장애 감지 및 자동 복구*

* **품질 속성 중요도 (BV): 최상**
* **아키텍처 영향도 (AI): 최상**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| QA 시나리오 | 시스템을 구성하는 특정 컴포넌트에 장애가 발생했을 때, 시스템이 외부 개입 없이 스스로 이를 감지하고 1분 이내에 정상 상태로 복구하는 시나리오 | | | | |
| 품질속성 | 가용성 | | | | |
| 자극 소스 | 시스템 내부 컴포넌트 | | | | |
| 자극 | 컴포넌트가 오류로 중단됨 | | | | |
| 환경 | 시스템이 정상 운영 중인 상태. 클라우드 플랫폼은 평균 부하 상태에서 동작하고 있으며, 홈 게이트웨이로부터 지속적으로 데이터가 수신되고 있는 상태. | | | | |
| 응답 | 장애가 발생한 '데이터 처리 컴포넌트'가 정상 상태로 복구 | | | | |
| 측정 | 컴포넌트의 장애 발생 시점(Stimulus)부터, 해당 컴포넌트가 정상 상태로 복구되어 데이터 처리를 다시 시작할 때(Response)까지의 복구 시간을 측정한다. 이 시간은 1분 이내여야 함. | | | | |
| 아키텍처 결정 | | 위험요소 | 민감점 | 절충점 | 비위험요소 |
| Microservices Architecture + Active Redundancy | |  |  | T1 |  |
| Heartbeat Monitoring 및 자동 재시작/Failover | | R1 |  |  | NR1 |
| Master-Slave Database Replication | |  |  |  | NR2 |
| Circuit Breaker 패턴 적용 | | R2 | S1 | T2 | NR3 |
| Reasoning | **[Microservices Architecture (MSA) + Active Redundancy]** 시스템을 독립적인 작은 서비스 단위로 분리함으로써, 특정 컴포넌트의 장애가 전체 시스템으로 전파되는 것을 효과적으로 장애 격리한다. 이는 장애 발생 시 영향 범위를 최소화하고, 장애 발생 컴포넌트만 신속하게 교체하여 1분 이내 복구 목표 달성에 직접적으로 기여한다.  **[Heartbeat Monitoring 및 자동 재시작/Failover]**모든 핵심 컴포넌트(서비스, 게이트웨이, 데이터베이스)에 Heartbeat Monitoring을 적용하여 비정상 상태를 자동으로 감지한다. 감지된 장애에 대해서는 미리 정의된 정책(자동 재시작, Failover)에 따라 사람의 개입 없이 즉시 복구 작업을 트리거하여 MTTR(Mean Time To Recovery)을 1분 이내로 단축하는 핵심적인 자동 복구 메커니즘을 제공한다.  **[Master-Slave Database Replication]** Master DB 장애 발생 시 Slave DB가 Master 역할을 인계받을 수 있도록 구성하여 데이터베이스 서비스의 중단을 최소화한다. 이는 데이터베이스라는 핵심 자원의 장애가 전체 시스템 가용성(QA-02)에 미치는 영향을 줄이고, Master DB 장애 발생 시 1분 내 복구에 기여한다.  **[Circuit Breaker 패턴 적용]** 외부 서비스 호출 시 잠재적인 장애나 지연이 발생할 경우, 해당 호출을 신속하게 실패 처리함으로써 내부 시스템으로의 장애 전파를 방지한다. 이는 내부 컴포넌트가 불필요하게 대기하거나 자원 고갈에 빠지는 것을 막아 시스템의 가용성(QA-02)을 보호하고 빠른 복구에 기여한다. | | | | |

### List of Risks

* R1 : Heartbeat 신호 오탐지(False Positive) 또는 미탐지(False Negative) 발생 가능성
  + **대처방안 :** 단일 Heartbeat 신호의 유무에만 의존하지 않고, 여러 번의 연속적인 Heartbeat 실패 또는 추가적인 상태 점검(예: API 응답 상태, 리소스 사용량)을 통해 복합적으로 장애를 검증하는 로직을 도입한다. 또한, 장애 감지 즉시 자동 복구를 트리거하는 대신, 짧은 지연 시간(예: 10초)을 둔 후 복구를 시도하여 일시적인 네트워크 문제나 부하 급증으로 인한 오탐지를 줄인다.
  + **장점:** 오탐지로 인한 불필요한 서비스 재시작을 줄여 가용성(QA-02)의 안정성을 높이고, 실제 장애에 대한 대응 신뢰도를 향상시킴. 복합적인 검증을 통해 미탐지 위험을 줄이고 실제 장애를 더 정확하게 감지 가능.
  + **단점:** 지연된 자동 복구 도입으로 인해 실제 장애 발생 시 가용성(QA-02) 복구 시간이 미세하게 늘어날 수 있음(하지만 1분 이내 목표에는 여전히 부합).
  + **실행 가능성:** 높음. 대부분의 모니터링 시스템이나 클라우드 오케스트레이션 도구(Kubernetes의 Liveness Probe, Readiness Probe 등)는 여러 검증 방식과 initialDelaySeconds, failureThreshold와 같은 지연 및 임계값 설정을 지원하여 이러한 로직을 쉽게 구현할 수 있음.
  + **영향 평가:** 오탐지/미탐지 위험을 줄여 시스템의 안정적인 가용성 유지에 크게 기여함. 불필요한 서비스 중단 방지하여 성능의 연속성 확보 가능.
* R2 : Circuit Breaker가 Open 상태로 너무 오래 유지되거나, 잘못된 임계값 설정으로 인해 정상적인 외부 서비스 호출이 장시간 차단되어 서비스의 가용성(QA-02)이 저하되고, 이로 인해 사용자 요청에 대한 성능(QA-01) 저하(실패 응답)를 경험할 수 있음.
  + 대처방안 : 실패 임계값, Open 상태 유지 시간 등의 파라미터를 시스템의 실제 부하 패턴과 외부 서비스의 응답 특성에 따라 동적으로 조정하거나 최적화된 초기 값을 설정한다.
  + **장점 :** 외부 서비스가 회복되었을 때 불필요하게 Open 상태로 머무르는 시간을 최소화하여 QA-02 가용성 복구 시간을 단축함.서비스 장애의 영향을 최소화하면서도, 서비스 회복 시 트래픽을 빠르게 재개하여 성능(QA-01)의 연속성을 확보.
  + **단점:** 동적 임계값 로직이나 최적화된 Half-Open 정책 구현은 복잡도가 있으며, 지속적인 모니터링과 튜닝이 필요함. 못된 초기 파라미터 설정은 오히려 가용성(QA-02)이나 성능(QA-01)에 악영향을 미칠 수 있음.
  + **실행 가능성:** 높음. 대부분의 Circuit Breaker 라이브러리는 이러한 파라미터 설정 및 Half-Open 정책을 유연하게 구성할 수 있는 기능을 제공합니다. 클라우드 환경에서는 매니지드 서비스 형태로도 지원됨.
  + **영향 평가:** Circuit Breaker의 본래 목적을 더욱 효과적으로 달성하여 외부 서비스 장애로부터 시스템 가용성(QA-02)을 보호하고, 불필요한 차단 시간을 줄여 복구 목표 달성에 기여함. 서비스 회복 시 빠르게 정상적인 트래픽을 처리하여 성능(QA-01)의 연속성을 확보하고, 불필요한 실패 응답을 줄임

### List of Sensitivities

* S1 : Circuit Breaker의 failureThreshold, resetTimeout 등 설정 값은 외부 서비스 장애 감지 및 회복 속도를 결정하여 가용성(QA-02)과 성능(QA-01) 간의 균형에 영향을 미침.

### List of Tradeoffs

* T1 : 장애 격리 및 독립적 배포 확장으로 가용성(QA-02)은 향상되지만 , 서비스 간 통신 오버

헤드로 성능(QA-01) 저하 가능성이 있다

* T2 : Circuit Breaker를 적용하여 외부 서비스 장애 시 가용성(QA-02)을 높이는 대신, 일시적으로 요청 거부로 성능(QA-01) 저하(실패 응답)를 경험할 수 있음.

### List of Nonrisks

* NR1 : 시스템 내부 컴포넌트의 비정상 상태를 자동으로 감지하고, 설정된 정책에 따라 재시작 또는 Failover를 트리거하여 서비스 중단을 최소화함.
* NR2 : Master DB 장애 시 Slave DB가 역할을 인계받아 데이터베이스 서비스의 연속성을 보장하여 QA-02 가용성 목표 달성에 기여함.
* NR3 : 외부 서비스의 장애가 내부 시스템으로 전파되는 것을 막아, 내부 컴포넌트들이 불필요하게 대기하거나 연쇄적으로 실패하는 것을 방지하여 QA-02 가용성을 보호함.

## Architectural Evaluation for *QA-05 사용자 증가에 따른 성능 유지*

* **품질 속성 중요도 (BV): 상**
* **아키텍처 영향도 (AI): 최상**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| QA 시나리오 | 총 사용자 수가 2배로 증가하는 상황에서, 시스템이 자원을 수평적으로 확장하여 기존의 응답 속도와 처리량을 안정적으로 유지하는 시나리오 | | | | |
| 품질속성 | 확장성 | | | | |
| 자극 소스 | 다수의 사용자 | | | | |
| 자극 | 시스템으로 유입되는 데이터 동기화 및 API 요청 트래픽이 2배로 증가 | | | | |
| 환경 | 시스템이 정상 운영 중인 상태. 시스템의 전체 부하가 점진적으로 2배 증가하는 상태. | | | | |
| 응답 | 시스템의 컴퓨팅 자원(서버 인스턴스, 데이터베이스 등)이 자동으로 확장된다. | | | | |
| 측정 | 응답 속도는 기존 목표(2초 이내)를 만족해야 하며, 자원 사용률은 안정적인 임계값(80%) 미만을 유지하고, 에러 발생률은 증가하지 않아야 한다. | | | | |
| 아키텍처 결정 | | 위험요소 | 민감점 | 절충점 | 비위험요소 |
| Microservices Architecture | |  |  |  | NR1, NR2 |
| Multi-tier 스타일 | |  |  |  | NR3 |
| 로드 밸런싱 | |  | S1 |  | NR4 |
| 캐싱 | |  | S2 | T1 | NR5 |
| Database Replication | | R1 | S3 | T2 | NR6 |
| Reasoning | **[Microservices Architecture (MSA)]** 시스템을 독립적인 서비스 단위로 분리함으로써, 사용자 수 증가로 특정 서비스에 부하가 집중될 경우 해당 서비스만 독립적으로 Scale-out할 수 있는 수평 확장성을 제공한다. 이는 전체 시스템을 재배포하거나 다른 부분에 영향을 주지 않고 필요한 부분만 효율적으로 확장하여 확장성(QA-05) 목표 달성에 핵심적으로 기여하며, 클라우드 환경의 자동 확장 기능과 연계될 때 더욱 강력하며 인스턴스 별 자원사용률이 임계값 미만을 유지하도록 한다.  **[Multi-tier 스타일 적용]** 시스템을 Application Tier, DataTier로 논리적/물리적으로 분리함으로써, 각 계층의 독립적인 확장을 가능하게 한다. 예를 들어, 사용자 트래픽 증가로 Web/API 계층의 부하가 높아지면 해당 계층의 서버 인스턴스만 늘릴 수 있고, 데이터 처리량이 많아지면 DataTier의 인스턴스를 확장할 수 있다. 이는 특정 계층이 전체 시스템의 확장성(QA-05) 병목으로 작용하는 것을 방지하고, 리소스의 효율적인 증설을 가능하게 하며 안정적인 자원사용률을 유지하도록 기여한다.  **[로드 밸런싱]** 사용자 증가에 따른 API 요청 트래픽을 다수의 서버 인스턴스(MSA 서비스 인스턴스, Multi-tier의 각 계층 인스턴스)에 균등하게 분배한다. 이는 특정 서버에 부하가 집중되는 것을 방지하고, 모든 컴퓨팅 자원(서버 인스턴스)이 효율적으로 활용될 수 있도록 하여 확장성(QA-05)을 위한 성능(QA-01) 유지를 보장한다. 또한, 새로운 인스턴스가 추가될 때 자동으로 트래픽을 분배하여 Auto-scaling의 효과를 극대화하며 안정적으로 응답속도(2초)를 만족하게 한다.  **[Caching]** 사용자 증가에 따른 데이터 동기화 및 API 요청 트래픽이 2배로 증가할 때, 대부분의 데이터 조회 요청을 캐시에서 처리하여 데이터베이스의 부하를 획기적으로 줄인다. 이는 데이터베이스가 확장성(QA-05)의 병목 지점이 되는 것을 방지하고, 성능(QA-01)을 안정적으로 유지하면서 시스템의 확장성(QA-05)을 확보하는 데 필수적인 역할을 한다. 이를 통해 자원 사용률을 안정적인 임계값(80% 미만)으로 유지하는 데 기여한다.  **[Database Replication]** 읽기 트래픽이 압도적으로 많은 홈 IoT 환경에서, Slave DB를 추가하여 읽기 요청을 수평적으로 분산할 수 있는 데이터베이스 읽기 확장성을 제공한다. 사용자 증가에 따라 Slave DB 인스턴스를 추가함으로써 Master DB의 부하를 줄이고 성능(QA-01)을 유지하며, 데이터베이스 계층의 확장성(QA-05)을 확보하여 트래픽 2배 증가 상황에서도 안정적인 응답 속도를 보장한다. | | | | |

### List of Risks

* R1 : Master DB에 쓰기 작업이 집중될 경우, Master DB 자체가 병목이 되어 확장성(QA-05) 한계에 도달하고 가용성(QA-02)까지 위협받을 수 있음.
  + **대처방안 :** 데이터 샤딩을 통해 각 샤드가 독립적인 Master DB역할을 수행하게 하여 쓰기 트래픽을 분산한다.
  + **장점 :** 쓰기 트래픽을 샤드된 DB 인스턴스에 분산하여 단일 Master DB의 쓰기 병목 현상을 해결(확장성(QA-05)), 특정 샤드에 장애가 발생하더라도 전체 시스템의 가용성에는 영향을 미치지 않도록 장애 도메인을 격리할 수 있음.
  + **단점 :** 데이터 일관성 관리 복잡도가 증가하며 여러 Master DB 인스턴스를 운영하게 되어 운영 비용이 증가할 수 있음.
  + **실행가능성 :** 중간. 기존 시스템에 추가 적용은 매우 어려우므로 초기 설계 단계부터 고려되어야 하며 샤딩 키 선정 및 재분배 전략에 대한 높은 수준의 전문성이 요구됨.
  + **영향 평가 :** MasterDB의 쓰기 병목을 근본적으로 해결하여 사용자 수 2배 증가 이상의 대규모 확장성 목표 달성에 필수적임.

### List of Sensitivities

* S1: 로드 밸런싱 알고리즘 및 헬스 체크(Health Check) 설정은 확장성(QA-05)을 위한 부하 분배 효율성과 성능(QA-01)에 직접적인 영향을 미침.
* S2: 캐시 적중률(Cache Hit Ratio)은 사용자 증가에 따른 데이터베이스 부하 분산 및 성능(QA-01) 유지에 결정적인 영향을 미쳐 확장성(QA-05)에 매우 민감함.
* S3: Slave DB의 수 및 DataService의 읽기 트래픽 분산 로직 효율성은 사용자 증가에 따른 확장성(QA-05)과 성능(QA-01) 유지에 직접적인 영향을 미침.

### List of Tradeoffs

* T1: 캐시(Caching)를 도입하여 데이터베이스 부하를 줄이고 읽기 확장성을 확보했지만, 캐시의 데이터와 원본 DB의 데이터 간에 일시적인 불일치가 발생할 수 있는 데이터 일관성(Consistency) 문제를 감수해야 했다. (성능/확장성 vs. 일관성)
* T2 : 읽기 성능 확장을 위해 DB Replication을 사용했지만, Master DB에서 Slave DB로 데이터를 복제하는 데 지연(Replication Lag)이 발생할 수 있다. 또한, 여러 DB 인스턴스를 유지해야 하므로 운영 비용(Cost)이 증가하는 절충점이 있다. (읽기 확장성 vs. 비용/복제 지연)

### List of Nonrisks

* NR1: MSA는 각 마이크로서비스를 독립적으로 개발, 배포, 확장할 수 있게 하여, 특정 컴포넌트의 부하 증가 시 해당 컴포넌트만 유연하게 Scale-out할 수 있도록 지원하며 확장성(QA-05)에 직접적으로 기여함.
* NR2: 클라우드 환경의 컨테이너 오케스트레이션(Kubernetes 등)과 결합하여 Auto-scaling 기능을 쉽게 구현할 수 있어, 사용자 증가에 따른 리소스 증설을 자동화하여 확장성(QA-05)과 성능(QA-01)을 효과적으로 확보함.
* NR3: 계층 간 명확한 책임 분리는 개발 및 유지보수를 용이하게 하며, 특정 계층의 기술 변경이 다른 계층에 미치는 영향을 최소화하여 유지보수성을 높이고 확장성(QA-05) 구현을 간소화함.
* NR4: 로드 밸런서는 헬스 체크를 통해 비정상적인 인스턴스를 자동으로 트래픽 분배에서 제외하고, 새로운 인스턴스가 추가되면 자동으로 트래픽을 분배하여 가용성(QA-02) 및 확장성(QA-05)을 동시에 확보함.
* NR5: In-Memory Cache는 사용자 증가에 따른 데이터 조회 요청의 대부분을 데이터베이스가 아닌 캐시에서 처리하게 하여, 데이터베이스 부하를 획기적으로 줄이고 성능(QA-01)과 확장성(QA-05)을 동시에 확보함.
* NR6: Replication은 읽기 트래픽이 많은 환경에서 Slave DB를 추가하여 읽기 용량을 무한히 확장할 수 있게 하며, Master DB의 부하를 줄여 성능(QA-01)을 유지하고 확장성(QA-05)을 제공함.

Appendix

## Detailed Component Specification for C&C View

C&C 뷰에서 사용된 주요 Component에 대한 Specification으로 Interface의 이름과 종류, 역할 등에 대해 설명한다.

### Component 1 GatewayController

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

##### Interface List

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IDeviceMonitor | Required | 기기 상태 및 사용량 정보를 모니터링하도록 요청하는 interface |
| IDeviceControl | Required | HomeAssistant 라이브러리를 통해 실제로 기기 제어를 요청하고 상태를 조회하는 interface |
| ILocalStorage | Required | 게이트웨이의 Local 저장소에 저장 및 조회하는 interface |
| IMQTTBroker | Required | 메시지 브로커에 메시지를 구독 및 발행하는 interface |
| IGatewayService | Provided | 클라우드로부터의 명령을 처리하고 디바이스 상태를 보고하는 등  게이트웨이 핵심 비즈니스 로직 흐름을 정의하는 interface |

### Component 2 CustomerUI

텍스트, 스크린샷, 폰트, 상징이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IAppCache | Required | 웹/앱의 Cache에 저장 및 조회하는 interface |
| IMQTTBroker | Required | 메시지 브로커에 메시지를 구독 및 발행하는 interface |
| ICustomerUI | Provided | 주요 시스템의 핵심 비즈니스 로직 흐름에 대해 사용자와 직접 상호작용하고 클라우드와의 통신을 담당하는 interface |

### Component 3 DataService

텍스트, 폰트, 스크린샷, 상징이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IMasterDB | Required | 데이터 쓰기/갱신 작업을 처리하는 MasterDB와 상호작용하기 위한 interface |
| ISlaveDB | Required | 데이터 읽기 작업을 처리하는 SlaveDB와 상호작용하기 위한 interface |
| ICache | Required | InMemory Cache 시스템과 상호작용하기 위한 interface |
| IPowerCompany | Required | 전력 공급사로부터 데이터를 연동하거나 조회하기 위한 interface |
| IHeartbeatMonitor | Required | 자신의 인스턴스 상태를 외부에 주기적으로 보고하기 위한 interface |
| IDataService | Provided | 사용자, 디바이스 정보 및 제어 이력 등 시스템의 모든 데이터에 대한 조회 및 저장 오퍼레이션을 제공하는 interface |

### Component 4 NotificationSystem

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| ISMSSystem | Provided | SMS system을 통해서 메시지를 푸쉬하는 interface |
| INotificationService | Provided | 사용자에게 경고 혹은 알림 메시지를 푸쉬하는 interface |

### Component 5 DeviceControlService

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IDataService | Required | DB 및 Cache와 상호작용하기 위한 interface |
| IHeartbeatMonitor | Required | 자신의 인스턴스 상태를 외부에 주기적으로 보고하기 위한 interface |
| IMQTTBroker | Required | 메시지 브로커에 메시지를 구독 및 발행하는 interface |
| IDeviceControlService | Provided | 사용자의 기기 제어 관련 요청을 처리하는 interface |

### Component 6 HeartbeatMonitor

텍스트, 스크린샷, 명함, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IHeartbeatMonitor | Provided | Monitor 대상 인스턴스가 상태를 보고하기 위한 interface |

### Component 7 PredictionService

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IDataService | Required | DB 및 Cache와 상호작용하기 위한 interface |
| IHeartbeatMonitor | Required | 자신의 인스턴스 상태를 외부에 주기적으로 보고하기 위한 interface |
| IPredictionService | Provided | 전기 요금 예측 및 최적화 제안을 요청하기 위한 interface |

### Component 8 DeviceMonitor

텍스트, 스크린샷, 폰트, 명함이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IDeviceMonitor | Provided | 기기 상태 및 사용량 정보를 모니터링하도록 요청하는 interface |

## Interface specifications for C&C View

C&C 뷰에서 사용된 Interface들에 대한 Specification으로 Interface가 가지는 operation들과 그 책임(Responsibility)에 대해 설명한다.

### IDeviceMonitor Interface Specification

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| registerObserver(observer) : void | 옵저버 등록 |
| unregisterObserver(observer) : void | 옵저버 등독 해제 |
| notifyDeviceStatusChanged(deviceID,string) : void | 기기의 변경된 상태를 notify |

### IGatewayService Interface Specification

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| processCloudCommand(command, payload) : void | 클라우드 명령(기기 제어 등)을 요청 |
| reportDeviceStatus(deviceId, statusData) : void | 기기 상태 및 전력 사용량 보고 |

### IMQTTBroker Interface Specification

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| publish(topic, message) : void | MQTT를 통한 Message 발행 |
| subscribe(topic,string) : void | MQTT를 통한 Message 구독 |
| onMessageReceived(callback) : void | 메시지 수신 시의 callback 등록 |

### 텍스트, 폰트, 스크린샷, 명함이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.IHeartbeatMonitor Interface Specification

|  |  |
| --- | --- |
| Operation | Responsibility |
| report(serviceId, status) : void | 인스턴스의 상태를 보고 |

### IDeviceControlService Interface Specification

텍스트, 명함, 폰트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| executeDeviceControl(request) : Result | request(제어 및 조회) 명령을 gateway에 전달 |

### IDataService Interface Specification

텍스트, 폰트, 스크린샷, 명함이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| getData(key) : Data | 데이터(사용자 정보, 전력 정보 등) 조회 |
| saveData(key, data) : void | 데이터(사용자 정보, 전력 정보 등) 저장 |

### ISlaveDatabase Interface Specification

텍스트, 폰트, 명함, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| select(query) : Data | 데이터(사용자 정보, 전력 정보 등) 조회 |

### IMasterDatabase Interface Specification

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| insert(data) : void | 데이터(사용자 정보, 전력 정보 등) 추가 |
| update(data) : void | 데이터(사용자 정보, 전력 정보 등) 수정 |
| delete(data) : void | 데이터(사용자 정보, 전력 정보 등) 삭제 |

### ILocalStorage Interface Specification

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| saveData(key, data) : void | 사용량, 기기 제어 명령, 기기 상태 등 저장 |
| loadData(key) : string | 사용량, 기기 제어 명령, 기기 상태 등 조회 |
| deleteData(key) : void | 사용량, 기기 제어 명령, 기기 상태 등 삭제 |

### ICache, IAppCache Interface Specification

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| get(key) : Data | 캐시 조회 |
| put(key, data) : void | 캐시 입력 |
| invalidate(key) : void | 오래된 캐시 무효화 |

### IPredictionService Interface Specification

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

|  |  |
| --- | --- |
| Operation | Responsibility |
| processPrediction(userID) : optimizationSuggestion | 사용자 요금 예측 및 최적화 제안 |