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 |  |  |

내용

[1. Business Drivers 5](#_Toc208012876)

[1.1. Business Goals 5](#_Toc208012877)

[1.1.1. Stakeholders List 5](#_Toc208012878)

[1.1.2. Business Goals 6](#_Toc208012879)

[1.2. Business Constraints 8](#_Toc208012880)

[2. System Context 9](#_Toc208012881)

[2.1. System Context Diagram 9](#_Toc208012882)

[2.2. External Entity List 9](#_Toc208012883)

[2.3. External Interface List 10](#_Toc208012884)

[3. Architectural Drivers 15](#_Toc208012885)

[3.1. Use Case Model 15](#_Toc208012886)

[3.1.1. Use Case Diagram 15](#_Toc208012887)

[3.1.2. Actor List 16](#_Toc208012888)

[3.1.3. Use Case List 16](#_Toc208012889)

[3.1.4. UC-01 전력정보 업데이트 19](#_Toc208012890)

[3.1.5. UC-02 AI 기반 요금예측 및 최적화 제안 20](#_Toc208012891)

[3.1.6. UC-03원격 기기 제어 21](#_Toc208012892)

[3.1.7. UC-04실시간 사용량 조회 22](#_Toc208012893)

[3.1.8. UC-05이상 상태 감지 및 알림 24](#_Toc208012894)

[3.2. Quality Attribute Scenario 25](#_Toc208012895)

[3.2.1. QA Scenario List 25](#_Toc208012896)

[3.2.2. QA-01 원격 제어 응답 속도 Scenario 27](#_Toc208012897)

[3.2.3. QA-02 신속한 시스템 장애 감지 및 자동 복구 Scenario 28](#_Toc208012898)

[3.2.4. QA-03 신규 IoT 프로토콜 지원 확장 Scenario 28](#_Toc208012899)

[3.2.5. QA-04 사용자 데이터 접근 제어 Scenario 29](#_Toc208012900)

[3.2.6. QA-05 사용자 증가에 따른 성능 유지 Scenario 29](#_Toc208012901)

[3.2.7. QA-06 인터넷 중단 시 로컬 기능 연속성 Scenario 30](#_Toc208012902)

[3.3. Architectural Constraint 30](#_Toc208012903)

[4. High Level Structure Design Description 32](#_Toc208012904)

[4.1. Domain Modeling 32](#_Toc208012905)

[4.1.1. Conceptual Class List 32](#_Toc208012906)

[4.1.2. Dynamic View 33](#_Toc208012907)

[4.1.3. Static View 37](#_Toc208012908)

[4.2. Quality Driven Architectural Design 37](#_Toc208012909)

[4.2.1. Exploring Architectural Options for *QA-01 원격 제어 응답 속도* 38](#_Toc208012910)

[4.2.2. Exploring Architectural Options for *QA-02 신속한 시스템 장애 감지 및 자동 복구* 45](#_Toc208012911)

[4.2.3. Exploring Architectural Options for *QA-05 사용자 증가에 따른 성능 유지* 49](#_Toc208012912)

[4.3. Component & Connector View 57](#_Toc208012913)

[4.3.1. UML Component Diagram 61](#_Toc208012914)

[4.3.2. Component List 62](#_Toc208012915)

[4.3.3. Connector List 64](#_Toc208012916)

[4.4. Deployment View 66](#_Toc208012917)

[4.4.1. Deployment Diagram 68](#_Toc208012918)

[4.4.2. Artifact Definition Diagram 74](#_Toc208012919)

[5. Detailed Component Design Description 78](#_Toc208012920)

[5.1. Design Description for *ComponentName* 78](#_Toc208012921)

[5.1.1. Overview 78](#_Toc208012922)

[5.1.2. Quality Driven Component Design 79](#_Toc208012923)

[5.1.3. Module View 81](#_Toc208012924)

[5.2. Design Description for *Component 2* 83](#_Toc208012925)

[5.3. Design Description for *Component 3* 83](#_Toc208012926)

[6. Architectural Evaluation 84](#_Toc208012927)

[*6.1.* Architectural Evaluation for *QA-01 Title* 84](#_Toc208012928)

[6.1.1. List of Risks 84](#_Toc208012929)

[6.1.2. List of Sensitivities 84](#_Toc208012930)

[6.1.3. List of Tradeoffs 85](#_Toc208012931)

[6.1.4. List of Nonrisks 85](#_Toc208012932)

[7. Appendix 86](#_Toc208012933)

[7.1. Detailed Component Specification for C&C View 86](#_Toc208012934)

[7.1.1. Component 1 Name 86](#_Toc208012935)

[7.1.2. Component 2 Name 87](#_Toc208012936)

[7.2. Interface specifications for C&C View 87](#_Toc208012937)

[7.2.1. *Interface2\_1* Interface Specification 87](#_Toc208012938)

# 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 한다.  **[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 |
|  |  |  |  |

### 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 | |
| 1.1 | **(스케줄에 의한 제어)** ScheduleManager가 사전에 등록된 일정(시간, 명령)에 따라 DeviceControlController에게 processControl 메시지를 보내며 제어를 시작합니다. |
| 2 | **(사용자에 의한 제어)** 또는, 사용자가 UserInterface를 통해 특정 기기(deviceID)에 대한 제어 명령(command)을 요청하며 제어를 시작할 수도 있습니다. |
| 2단계: 제어 실행 및 결과 처리 (공통) | |
| 1.2 / 2.1 | DeviceControlController는 DataManager를 통해 해당 기기에 대한 제어 권한을 확인합니다. |
| 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 반환합니다. |
| 1.3A/2.1A | **(권한이 없는 경우)** DeviceControlController는 UserInterface에 권한 없음 알림(notifyAuthFailure)을 보내고, 사용자에게 오류 메시지가 표시된 후 시나리오가 종료됩니다. |

#### *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, 폰트이(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. |
| 1.2 | AnomalyDetectionModel은 홈 게이트웨이로부터 실시간 데이터(realtimeUsage)를 가져와 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, 디자인이(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(Private Network) 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 전달(Push)함. |

##### **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. |
| 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 Option Title | 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. |
| 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.  [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 '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 안정성을 유지하는 데 도움이 됨. | [기아 상태] 이미 큐에 대기 중인 낮은 순위 이벤트의 실행을 보장해주지는 못하며, 지속적인 고부하 상황에서는 기아 상태가 지속될 수 있음. | 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 선택함. |
| 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. | [Cold Start 지연]  오랫동안 호출되지 않은 함수가 처음 실행될 때 약간의 지연(Cold Start)이 발생할 수 있어, 항상 즉각적인 응답이 필요한 기능에는 부적합할 수 있음. | 서버리스 아키텍처는 유휴비용을 0에 가깝게 만들 수 있다는 점에서 비용 최적화의 정점에 있으나 기기 제어와 같이 항상 저지연 응답이 필요한 서비스는 상시 대기하는 컨테이너 방식이 더 적합하다고 판단됨. |
| 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. 따라서 이 옵션을 선택함. |

### 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 |
| DB 쿼리/인덱스 최적화 | 데이터베이스 자체의 성능을 개선하는 방식임. '스마트홈 전기관리 시스템'에서 자주 사용되는 조회 조건(예: 기간, 사용량, 요금 정보 등)에 맞게 인덱스(Index)를 생성하고, 비효율적인 SQL 쿼리를 튜닝하여 각 조회 작업의 응답 시간을 단축시키는 데 집중함. |
| 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 |
| DB 쿼리/인덱스 최적화 | [구현 용이성] 애플리케이션 아키텍처의 큰 변경 없이 DB 내부 설정만으로 성능을 개선할 수 있음. | [확장성 한계] 최적화만으로는 물리적인 I/O 한계를 넘을 수 없음. 사용자 수가 임계점을 넘어 증가하면 결국 성능 저하가 발생함. | DB 최적화는 기본적으로 수행해야 할 중요한 작업이지만, QA-05에서 가정한 '사용자 2배 증가'와 같은 대규모 트래픽을 감당하기에는 근본적인 한계가 있음. Caching 택틱은 아예 DB 접근 자체를 생략하게 함으로써, 부하를 원천적으로 차단하고 비교할 수 없이 빠른 응답 속도를 제공함. 이는 성능(QA-01)과 확장성(QA-05)을 동시에 만족시키는 보다 근본적인 아키텍처 해결책이므로, 이 옵션을 선택함. |
| 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. | Caching은 최상위에서 가장 빠른 응답을 제공하여 DB 부하를 원천적으로 줄여주고, Data replication은 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 만족 |

#### 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 |
| Least Connections 알고리즘 적용 | 로드 밸런서가 현재 연결 수가 가장 적은 서버에게 다음 요청을 보내주는 방식임. 이는 각 서버의 현재 처리 중인 작업량을 간접적으로 반영하여, 특정 서버에 과부하가 걸리는 것을 방지하고 서버 간의 부하를 균등하게 유지하는 데 도움이 됨. |
| 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 |
| Least Connections 알고리즘 적용 | [QA-05 확장성, QA-01 성능]  서버 간의 부하를 실시간으로 균등하게 유지하여, 특정 서버의 과부하로 인한 성능 저하를 방지하고 전체 시스템의 처리량을 극대화함. | [세션 관리의 어려움]  사용자의 요청이 매번 다른 서버로 전달될 수 있어, 세션 정보를 중앙에서 관리하지 않으면 사용자 상태가 유실될 수 있음. | 우리 시스템은 로그인 기반 서비스로, 사용자의 상태를 일정 시간 유지해야 할 필요가 있음. Least Connections는 부하 분산에는 효과적이지만 세션 연속성을 보장하지 못함. |
| 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 서버 부하 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 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. |
| Prioritize Events 택틱 |
| 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 대응할 수 있어 서비스의 확장성(QA-06)을 만족시킴과 동시에 비용적인 측면에도 유리함. |
| 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, 도표, 디스플레이이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 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, QA-05 완전만족) Active Redundancy 및 수평 확장을 통해 가용성과 확장성을 보장함.  **[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) Prioritize Events 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. | UC-01, UC-04 QA-01, QA-02, QA-05, 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 완전만족) 인터넷 중단 시 독립적으로 동작하여 로컬 기능의 연속성을 보장함. **[제약 사항]** (AC-01 오픈소스 완전만족) Home Assistant 기반으로 구현됨. | UC-01, UC-03, UC-05, 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 & 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.

#### 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 제공함  [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.<br>[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 & Autoscaled 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 용이함.  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. |
| DataServiceContainer, NotificationServiceContainer,  DeviceControlServiceContainer, HeartbeatMonitorContainer | JVM | [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

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

\* 아키텍처적으로 중요한 컴포넌트 3개 이상에 대해 작성하고, 컴포넌트의 대표적 Behavior를 시퀀스 다이어그램으로 기술해야 함.

## Design Description for *ComponentName*

### Overview



<작성 방법>

|  |  |
| --- | --- |
| 항목 | 설명 |
| 개요 | 이 컴포넌트의 역할을 한/두 줄로 간략하면서 명확하게 서술한다. |
| 컴포넌트 기능 요구사항 | 이 컴포넌트에 부여된 기능 요구사항을 서술한다. Provided Interface를 중심으로 구체적으로 설명한다. |
| 컴포넌트 품질 요구사항  및 제약사항 | 이 컴포넌트가 만족해야 할 품질 요구사항 및 제약사항을 서술한다.   * 시스템의 QA를 충족시키기 위하여 이 컴포넌트에서 만족시켜야하는 QA를 구체적으로 서술한다. * 시스템의 제약사항 중에서 이 컴포넌트에서 만족시켜야 하는 제약사항이 있다면 이를 구체적으로 서술한다. |

<점검 사항>

* 컴포넌트의 요구사항이 명확하게 정의되었는가?

### Quality Driven Component Design

#### Exploring Design Options for *QA-01 Title*

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



<작성 방법>

* 해당 QA를 만족시키기 위해서 설계 시 고려해야할 설계이슈(Design Issue)와 관련된 설계 옵션 (Design Option) 간의 의사결정 구조를 도식화 한다.
* 고려해야할 설계 이슈로는 다음과 같은 것을 고려해 볼 수 있다.
  + 컴포넌트의 모듈 관점의 분해
  + 컴포넌트 간의 상호작용의 모듈 설계
* 해당 Design Issue가 다른 Design Option에서 추가적으로 파생된 문제인 경우에는 이들을 연결시켜서 표현한다.

<점검 사항>

* 품질 요구사항에 대한 컴포넌트 설계 결정 구조가 적절한가?

#### Design Decisions for *설계이슈1*

###### Definition for *설계이슈1*

설계 이슈(*설계이슈1*)에 대한 설명 기술

<작성 방법>

* 이슈가 발생하는 상황과 그 배경을 구체적으로 기술하고, 이슈와 관련된 가정된 조건이 있다면 이를 구체적으로 명시한다.
* 기술된 배경 및 가정하에서 제기된 이슈(문제)가 무엇인지 명확하게 정의하고, 이 문제가 왜 중요한지 설명한다.

###### Design Options for *설계이슈1*

|  |  |
| --- | --- |
| Design Option | |
| Title | Description |
|  |  |
|  |  |
|  |  |

<작성 방법>

* 해당 설계이슈와 관련된 모든 Design Option을 기술한다.
* Design Option의 Title이 설명하고 있는 Description과 일관성이 있어야 한다.
* Design Option의 Description이 실질적이고 세부적인 design이 제시되어야 한다.
  + 적용하고 있는 설계 솔루션(패턴, 택틱 등)에 대한 일반적인 설명이 아니라 해당 솔루션을 이 시스템에 적용할 때의 아키텍처 설계 결정을 구체적으로 제시해야 한다.
  + Layer patter n 적용의 경우: Layer의 수 및 각 layer의 역할, Layer interface 설계 등에 대한 decision이 제시될 필요가 있음
* 가능하면 제시된 설계 옵션들을 명확히 비교해 볼 수 있도록 구체적인 설계결정을 반영한 설계 뷰를 제시하는 것이 좋음

<점검 사항>

* 설계 이슈와 설계 옵션에 대한 설명이 구체적인가?

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

|  |  |  |  |
| --- | --- | --- | --- |
| Design Option Title | Pros | Cons | Rationale |
| Design Option Title1 |  |  |  |
| Design Option Title2 |  |  |  |

<작성 방법>

* 각 Design Option별로 5.1절에서 제시된 모든 관련된 QA 측면 및 필요한 관심사(concerns) 측면에서 장/단점을 제시한다.
  + Pros/cons는 QA와 관심사 관점에서 구체적으로 장단점을 기술해야 한다. 즉 각 QA 관점에서 Response Measure에 대한 유/불리 와 Constraint 충족 여부 등이 명시적으로 제시될 필요가 있음
* 제시된 Design Option들 중에서 가장 적합한 Design Option을 선정하는 근거를 기술한다.

<점검 사항>

* 설계 옵션에 대한 평가와 선택 근거가 타당한가?

#### Exploring Design Options for *QA-02 Title*

### Module View

* 5.1.2절에서 기술된 컴포넌트 설계 결정 위주로 표현한다.

|  |  |  |
| --- | --- | --- |
| QA Scenarios ID | (Selected) Design Option Title | Reflection in the Module View |
| QA-01 | Design Option Title1 |  |
| Design Option Title3 |  |
| QA-02 |  |  |
|  |  |

<작성 방법>

* 4.2절에서 기술한 QA에 대한 아키텍처 설계결정 중에서 Module View에 반영된 부분이 있으면 어느 부분에 어떻게 반영되어 표현되었는 지를 기술한다.

<점검 사항>

* 컴포넌트 설계 결정이 모듈 뷰에 구체적으로 반영되어 있는가?

#### Class Diagram

<작성 방법>

* Component를 구현하기 위하여 필요한 Class 및 Interface들을 모두 식별한다.
* Class 간의 관계(generalization, association, composition/aggregation, dependency)를 표시한다.
* Class의 attribute 및 operation이 구체적으로 명시한다.
* 많은 수의 Class 및 Interface가 식별된 경우 Package Cohesion/Coupling을 고려하여 Packaging한다.

<점검 사항>

* Class Diagram이 구체적이고 올바로 작성되어 있는가?



##### Element List

<작성 방법>

* 위의 Class diagram에 표현된 모든 요소(package, class, interface)에 대한 역할을 설명한다.
* 각 요소가 Component에 부여된 요구사항(기능, 품질, 제약사항 등)에 어떻게 기여하는 지 측면에서 설명한다.

<점검 사항>

* Element의 설명이 구체적인가?

|  |  |
| --- | --- |
| Name | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#### Sequence Diagram

<작성 방법>

* 컴포넌트가 제공하는 인터페이스의 각 오퍼레이션이 어떻게 내부 모듈 인스턴스 간의 상호작용으로 실현되는가를 시퀀스 다이어그램을 이용해서 기술하시기 바랍니다.
* 시작은 컴포넌트 provided IF가 커넥터 모듈로부터 호출되는 방식으로 기술하시기 바랍니다.

<점검 사항>

* Sequence Diagram이 구체적이고 올바로 작성되어 있는가?



## Design Description for *Component 2*

## Design Description for *Component 3*

# Architectural Evaluation

* 우선순위가 높은 QA 3개 이상에 대해 이와 선정된 Architectural Decision에 대한 분석서를 작성한다.

## Architectural Evaluation for *QA-01 Title*

<작성 방법>

* 제시된 양식에 맞춰서 작성한다.

<점검 사항>

* 아키텍처 결정의 민감점, 절충점, 위험요소, 비위험요소 분석이 타당한가?
* 아키텍처 결정에 대한 Reasoning이 타당한가?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QA 시나리오 |  | | | |
| 품질속성 |  | | | |
| 자극 소스 |  | | | |
| 자극 |  | | | |
| 환경 |  | | | |
| 응답 |  | | | |
| 측정 |  | | | |
| 아키텍처 결정 | 위험요소 | 민감점 | 절충점 | 비위험요소 |
|  | R1 | S1 | T1 |  |
|  |  |  |  | NR1 |
|  |  |  |  |  |
| Reasoning | <<아키텍처 결정들이 시나리오가 표현하는 품질속성을 달성하는데 어떻게 공헌하는지에 대한 근거를 설명>> | | | |

### List of Risks

* R1
* …

### List of Sensitivities

* S1
* …

### List of Tradeoffs

* T1
* …

### List of Nonrisks

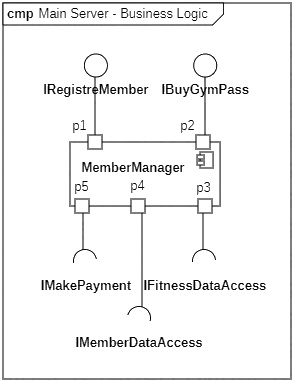
* NR1
* …

Appendix

## Detailed Component Specification for C&C View

### Component 1 Name

<< 아래 그림과 유사하게 UML 컴포넌트 다이어그램으로 컴포넌트의 인터페이스를 모델링하기 바랍니다.>>



##### Interface List

<점검 사항>

* 해당 Component의 모든 interface가 정의되었는가?
* 각 interface의 유형(Provided/Required)가 UML Component Diagram과 부합하는가?
* 각 interface의 역할/기능을 명확하게 설명하였는가?

|  |  |  |
| --- | --- | --- |
| Interface Name | Kind | Description |
| IRegisterMember | Provided | 회원 가입을 요청하는 interface |
| IBuyGymPass | Provided | 이용권 구매를 요청하는 interface |
| IMakePayment | Required | 결제를 요청하는 interface |
| IMemberDataAccess | Required | DB에 저장된 회원 정보에 접근하는 interface |
| IFitnessDataAccess | Required | DB에 저장된 피트니스 정보에 접근하는 interface |

### Component 2 Name

## Interface specifications for C&C View

### *Interface2\_1* Interface Specification

<작성 방법>

* 인터페이스를 구성하는 각 operation 별로 구체적인 인자/반환 타입을 기술한다.

<점검 사항>

* Component에 할당된 모든 기능이 Operation으로 할당되었는가?
* Operation의 이름이 제공되는 기능을 명확히 나타내는가?
* Operation의 인자/반환 타입이 올바르게 정의되었는가?

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| Operation | Responsibility |
| op1() |  |
| op2() |  |