2023년도 졸업논문

낙상사고 예방을 위한 스마트 무드등

MQTT, HTTP 프로토콜을 이용한 사물인터넷 시스템 개발

한국공학대학교

전자공학부

김도형

낙상사고 예방을 위한 스마트 무드등

MQTT, HTTP 프로토콜을 이용한 사물인터넷 시스템 개발

Smart Lamp for Fall Accident Prevention
: Development of IoT System Using MQTT and HTTP Protocol

2023年 12月 1日

한국공학대학교

전자공학부

김도형

낙상사고 예방을 위한 스마트 무드등

指導敎授 이 재 명 (인)

이 論文을 졸업논문 으로 提出함.

2023年 12月 1日

한국공학대학교

전자공학부

김도형

2023년도

졸업논문

낙상사고 예방을 위한 스마트무드등

성명 김도형

차 례

초록

오늘날 고령화 시대가 다가오면서, 요양원과 요양병원, 실버타운 등 노인 입소시설이 늘어감에 따라 해당 시설 내에서 발생하는 고령자의 낙상사고의 횟수도 증가하는 추세를 보인다. 또한 고령자들은 작은 낙상사고로도 고관절이나 척추를 다칠 가능성이 크며 중상 이상에 이를 수 있으므로 각별한 주의를 필요로 하고 있다. 이러한 낙상사고의 문제 상황을 사물인터넷 기술을 응용하여 낙상사고 예방과 신속한 사고 대응에 기여하고자 본 연구를 진행하게되었다.

본 연구에서는 야간 보행시 낙상사고에 취약한 환자들을 위한 안전장치로써 스마트 무드등을 연구, 개발하고 그 결과에 대해서 논한다. 스마트 무드등 시스템의 구성 요소에는 크게 3가지로 무드등 디바이스, 모바일 어플리케이션, 클라우드 서버로 구성되어 있다. 무드등 디바이스는 인체감지센서를 통해 보행자의 움직임을 감지하고 LED램프를 점등한다. 또한 인체감지센서 초음파센서를 이용하여 보행자의 낙상사고를 감지한다. 만약 낙상사고가 감지된다면, 무드등 디바이스는 클라우드 서버의 REST API에 HTTP Request를 요청하며 사전에 모바일 어플리케이션을 통해 클라우드 서버에 등록한 사용자 휴대폰 정보를 기반으로 문자 메세지 알림을 전송하는 동작을 수행한다. 이와 같이 본 연구에서는 사물인터넷을 생활안전분야에 접목시켜 일상 생활속에서의 낙상사고 위험도를 낮추며, 낙상사고를 사전에 예방하고 낙상사고 발생에 대한 신속한 대처를 위한 솔루션을 제시한다.

본 연구의 전체적인 실험 결과, 전체적인 실험 결과, 첫 번째로, Wi-Fi Latency와 MQTT Latency를 측정하여 디바이스의 무선 통신 성능을 확인했다. 실험 결과, Wi-Fi Latency는 평균 505ms로 안정적이었지만, MQTT Latency는 평균 363.8ms로 변동이 크게 나타났다. 이는 MQTT 브로커와 클라이언트 간의 통신 최적화가 필요하다는 것을 시사한다. 두 번째로, LED 램프의 밝기 제어 성능을 측정했다. 밝기 단계별로 측정을 진행하고, LED 램프의 밝기는 시그모이드 함수 모양의 곡선을 보였다. 그러나 낮은 단계에서의 측정에는 광학 카메라의 민감도 한계로 어려움이 있었고, 고단계에서의 포화 문제도 확인되었다. 세 번째로, 초음파 센서의 감지 거리와 방사 패턴을 확인했다. 측정 결과, 초음파 센서는 40cm부터 400cm까지의 물체를 감지할 수 있었으며, 중앙을 기준으로 -15도에서 15도까지의 각도 범위에서 원활한 거리 측정이 가능했다. 마지막으로, 인체감지센서의 최대 보행자 감지 거리를 평가했다. 측정 결과, 최대 감지 거리는 평균 396cm로 나타났으며, 이는 초음파 센서의 최대 감지 거리와 유사한 수준이었다. 종합적으로, 이 실험 결과를 토대로 디바이스의 성능을 평가하고 향후 개선 방향을 도출할 수 있었다.

1. 연구 배경 및 목적

연구 배경

첫째, 우리나라 65세 이상 노인 인구는 2020년 815만 명에서 2050년 1,900만 명까지 증가하다가 감소하기 시작하나 노인 인구 비율은 지속으로 증가할 것으로 전망하고 있다(통계청, 2021.12).
둘째, 우리나라의 경우 65세 이상 노인들 중 1/3 이상이 적어도 한 번의 낙상을 경험하고 있으며 한 해 동안 지역사회 재가노인의 경우 25,9%(Kim, 2004), 노인요양시설의 경우 24,2%가 낙상을 경험 하였다(No, 2006).
셋째, 노인의 경우 노화 과정에 따른 균형의 감소, 신경계 기능의 퇴화, 보행능력의 감소 및 근력의 약화와 같은 신체적 변화로 인해 걸려서 넘어지거나 미끄러지는 낙상사고가 쉽게 발생되는데 낙상 후에는 신체적 손상, 심리적 손상, 및 경제적 손실이 발생한다(Lehtola, Koistinen, & Luukinen, 2006).
넷째, 본 논문의 저자인 본인 또한 2021년부터 2023년도까지 요양시설에서 사회복무요원으로 근무하면서 이러한 유형의 낙상사고를 다수 목격한 바 있다. 요양시설 종사자가 24시간 요양시설 입소자들의 일거수일투족을 파악하기 힘들다는 현실을 바탕으로 본 연구를 진행한다.

연구 목적

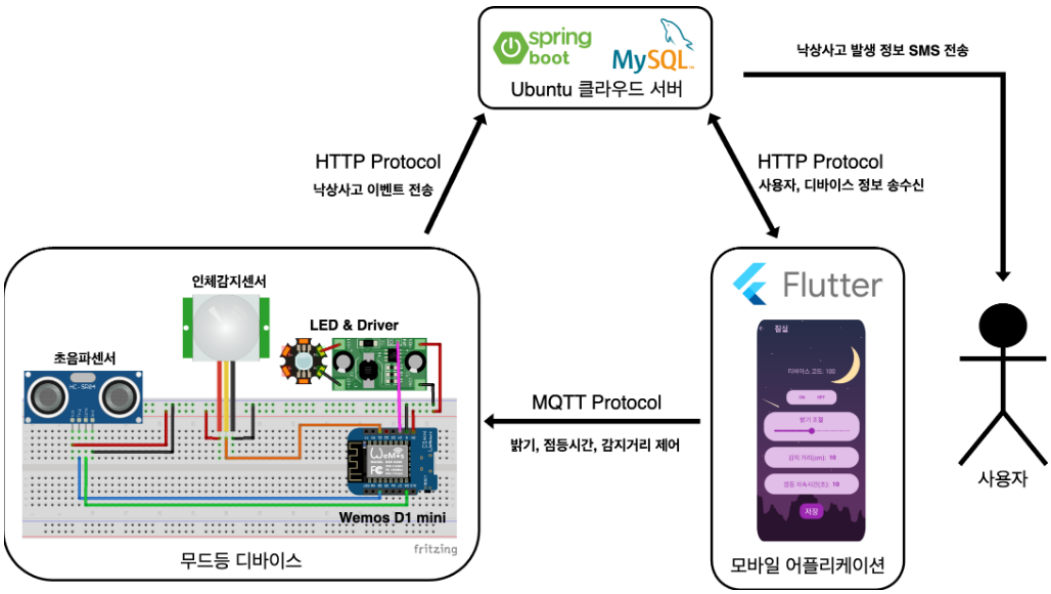
첫째. 본 연구에서는 고령자 또는 야뇨증 환자와 같이 실내 야간 보행 시 낙상사고에 노출되기 쉬운 사회적 약자들을 위한 스마트 무드등을 개발하여 실내 야간 보행 시 낙상사고 위험도를 낮추는 방안을 제시한다.
둘째. 낙상사고가 발생했다면 보호자와 관찰자에게 신속하게 사고 정보를 전달하는 낙상사고 솔루션의 필요성에 의해 본 연구가 진행되었다.
셋째. 야간 보행 중 발생하는 낙상사고는 노약자에게 심각한 건강 문제를 초래하는데, 이를 예방하고 사회적 약자들의 삶의 질을 향상시키는 것이 이 연구의 핵심 목표이다.
넷째. 본 연구 결과는 사회적 약자들에게 낙상사고에 대한 안전과 편의를 제공함으로써 사고 예방과 대처에 중요한 역할을 할 것으로 기대된다.

2. 연구방법

제 1 장 아키텍처 구성

본 장에서는 연구를 진행한 스마트 무드등 시스템의 전체적인 아키텍처 구성에 대해서 요약하여 서술한다. 시스템 구성은 크게 3가지로 무드등 디바이스, 모바일 어플리케이션, 클라우드 서버로 이루어져 있다. 무드등 디바이스는 초음파센서, 인체감지센서, LED와 LED 드라이버, Wemos D1 mini 모듈로 구성되어있으며 HTTP(HyperText Transfer Protocol) 프로토콜을 이용하여 낙상사고 감지 이벤트를 클라우드 서버로 전송한다. 이때 클라우드 서버는 해당 낙상사고 이벤트 정보를 수신하여 사전에 등록된 사용자 휴대폰 번호로 낙상사고 감지 정보가 포함된 SMS를 전송한다. 모바일 어플리케이션은 MQTT(Message Queuing Telemetry Transport) 프로토콜을 이용하여 무드등 디바이스의 LED램프 점등 시간과 LED램프 밝기값, 초음파센서의 감지 거리를 제어할 수 있다. 또한 어플리케이션은 클라우드 서버의 MySQL 데이터 베이스와 마이그레이션된 REST API를 통해 사용자와 디바이스 정보 등록, 조회, 삭제, 수정 기능을 수행한다.

다음은 전체 시스템 개요도에 대해서 설명하도록 한다. 우측 하단의 사용자는 모바일 어플리케이션을 통해서 좌측 하단의 스마트 무드등의 밝기, 점등시간, 감지거리를 제어한다. 또한 사용자 정보와 디바이스 정보를 상단의 Ubuntu 클라우드 서버로부터 주고받을 수 있다. 스마트 무드등은 감지된 보행자의 움직임 정보를 기반으로 LED 점멸 기능 및 낙상사고 발생 정보전달 기능을 수행한다.



[그림1. 시스템 아키텍처 다이어그램]

제 2 장 디바이스 설계

이제, 본 연구의 핵심인 '디바이스 설계'에 대한 논의를 시작한다. 이 장에서는 스마 트 무드등의 핵심 구성 요소인 하드웨어 설계에 대해 다루며, 무드등 디바이스가 어떻게 작동하고, 사용자와 클라우드 서버와 어떻게 상호작용하는지를 자세히 살펴볼 것이다.

제 1 절 하드웨어 구성

스마트 무드등의 하드웨어는 기본적으로 모바일 어플리케이션과 클라우드 서버와 무선통신을 통해 데이터를 주고받을 수 있어야 하기 때문에 Wi-Fi를 지원하며 최소한의 아날로그 핀과 디지털 핀을 제공함으로써 소형화되어 상용화된 WeMos D1 mini 보드를 기반으로 제작되었다. WeMos D1 Mini는 ESP8266 칩셋을 기반으로 한 개발 보드로, IoT(Internet of Things) 프로젝트 및 원격 제어 응용 프로그램에 적합한 장점이 있다. 앞서 말했듯이 기본적으로 Wi-Fi연결을 지원함으로써 모바일 어플리케이션과 클라우드 서버와 TCP(Transmission Control Protocol) 통신을 가능하게 하며 아두이노 IDE를 통한 다양한 라이브러리를 활용할 수 있는 개발의 용이함을 이유로 사용하였다. 특히 작은 크기와 저렴한 가격을 통해 추후 시장성 측면에서 이점을 가질 수 있다. 본 연구에서는 WeMos D1 mini를 IEEE 802.11 프로토콜을 이용한 인터넷 통신을 위해 사용하였다. 그 결과 모바일 어플리케이션, 클라우드 서버 간의 HTTP, MQTT 통신을 구현할 수 있었다.

스마트 램프 디바이스는 보행자의 움직임을 감지하기 위해서 인체 감지 센서, 초음파 센서를 사용하였다. 인체 감지 센서와 초음파 센서는 각각 고유한 장단점을 가지고 있으며, 두 센서를 함께 사용함으로써 보다 정밀한 보행자 움직임을 감지할 수 있다. 인체 감지 센서는 넓은 감지 각도를 지원하지만 감지된 사물까지의 정확한 거리 보정를 알 수 없다, 이와는 반대로 초음파 센서는 좁은 감지 각도를 지원하지만 사물까지의 정확한 거리 정보를 알 수 있다는 특징이 있다. 이와 같이 두가지 센서를 함께 사용함으로써 각각 단일 센서를 사용했을때의 한계점을 극복할 수 있다.

인체 감지 센서(Passive infrared sensor, PIR센서)는 시장에 상용화된 SEN050135 센서를 활용하였다. SEN050135 센서는 적외선(IR, infrared ray) 복사선을 활용하여 동작하기 때문에 보행자 신체에서 검출되는 적외선을 감지하기 위해서 사용되었다. SEN050135 센서는 보행자의 움직임을 감지하면 설정된 밝기와 점등시간만큼 LED를 점등하는 트리거 역할을 수행한다. SEN050135 센서는 보행자의 움직임이 감지되었을 때 High, 움직임이 감지되지 않을 때 Low 신호를 출력한다.

초음파 센서는 HC-SR04 모듈을 활용하였다. 초음파 센서는 공기중에서 초음파가 물체에 부

딪치고 돌아오는 시간을 계산하여 센서로부터 물체가 떨어진 거리를 도출할 수 있다. 음속 C는 1기압, 15℃의 대기 중에서 340m/s이라 가정한다.

$$D = (C \times T)/2$$

(**D**: 물체로부터 거리(m), **C**: 음속(m/s), **T**: 물체까지의 왕복시간(s))

인체 감지 센서에 보행자의 움직임이 감지된 후 5분 동안 초음파 센서는 보행자까지의 거리를 계산하며 10초간 거리 정보가 변화하지 않는다면, 보행자에게 낙상사고가 발생한것으로 추정하고 클라우드 서버에 HTTP Request를 수행하여 등록된 사용자 휴대폰으로 낙상사고 발생 의심 장소 정보를 SMS를 통해 전달한다. 사용자는 최대 보행자 감지 거리를 모바일 어플리케이션 상에서 설정 가능하다.

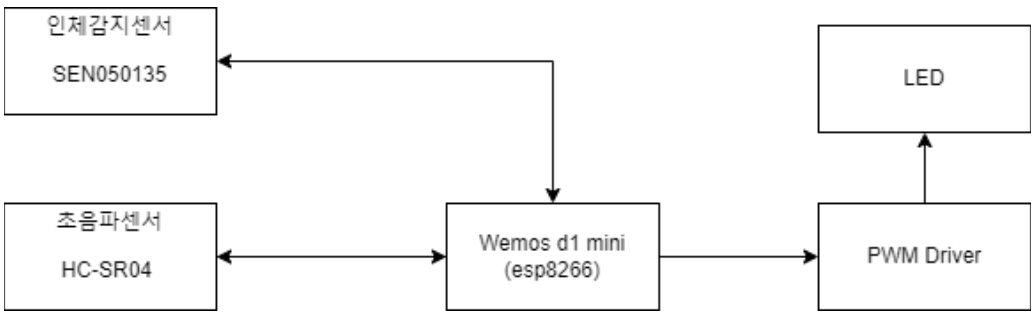
램프의 광원은 3W LED를 활용하였다. LED와 함께 정전류 모듈(LED Driver)을 사용함으로써 PWM(Pulse Width Modulation, 펄스 폭 변조) 방식의 LED의 밝기 조절(Dimming)을 수행하였다. PWM는 신호의 펄스 폭을 변경하여 전압이나 전류를 제어하는 방법으로, LED를 켜고 끄는 주기를 조절하여 밝기를 조절한다. PWM Dimming을위해 LED Driver의 출력 전류는 PWM 신호의 듀티 사이클과 특정 관계를 가진다. 다음과 같은 계산식을 통해 드라이버의 출력 전류가 결정된다. 사용자는 모바일 어플리케이션 상에서 10단계의 밝기 단계를 조절함으로써 듀티 사이클을 변화하여 LED밝기를 제어할 수 있다.

$$I_{set} = \frac{High\ Time}{Period} \times I_{norm} = \frac{Duty\ Cycle(\%)}{100} \times I_{norm}$$

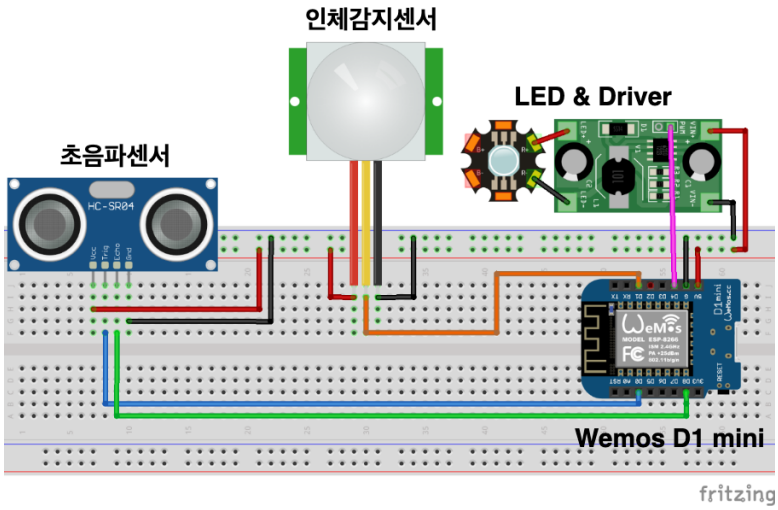
(**I_set**: 전류 출력값(mA), **I_norm**: 최대 전류 출력값(mA), **High Time**: PWM 신호가 High 상태일 때의 시간, **Period**: PWM 신호의 주기, **Duty Cycle**: PWM 신호가 High(켜짐) 상태에 있을 때의 시간 비율)

[표1. 밝기 단계 별 듀티 사이클 변화량]

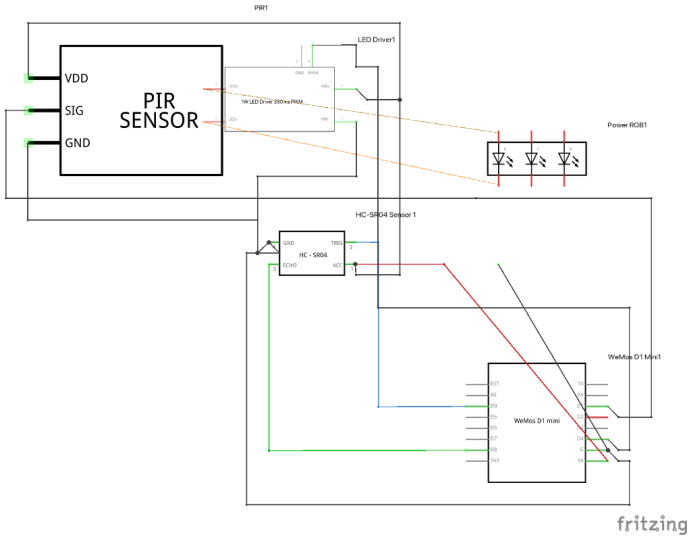
밝기 단계	0	1	2	3	4	5	6	7	8	9	10
듀티 사이클	0	10	20	30	40	50	60	70	80	90	100



[그림2. 하드웨어 구성도]

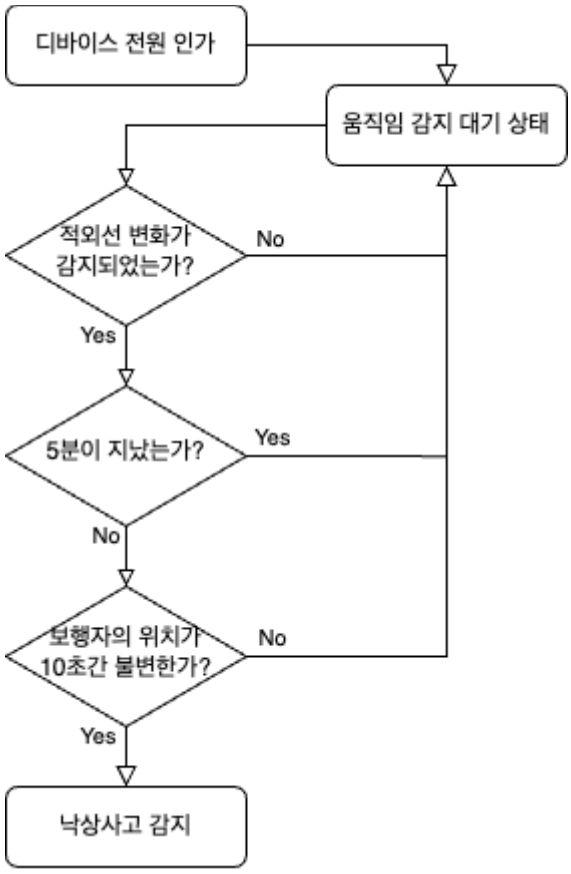


[그림3. 브레드보드 구성도]



[그림4. 스케메틱 구성도]

다음은 무드등 디바이스의 보행자 낙상 감지 로직에 대해서 설명한다. 디바이스에 전원 인가 시, 디바이스는 보행자의 움직임을 감지할 때 까지 대기 상태로 존재한다. 이후 인체감지센서로부터 보행자의 적외선 변화가 검출될 시 해당 시점을 기준으로 이후 5분간 적외선 변화가 지속된다면, 초음파센서로부터 보행자와 디바이스 간의 거리를 측정한다. 10초간 보행자와 디바이스 간의 거리가 변화하지 않는다면 낙상사고 발생을 추정한다.



[그림5. 낙상사고 감지 플로우차트]

제 2 절 데이터 통신

본 절에서는 스마트 무드등 디바이스의 관점에서 사용자 어플리케이션, 클라우드 서버 간의 데이터 통신을 위한 프로토콜에 대해 논의한다.

MQTT(MQ Telemetry Transport) 프로토콜은 경량 메시징 프로토콜로, 사물인터넷(IoT) 및 기계 간 통신(M2M) 분야에서 널리 사용된다. 이 프로토콜은 주로 발행자(Publisher)와 구독자(Subscriber) 간의 메시지 전송을 담당하며, 발행-구독 패턴을 따른다. MQTT는 경량 프로토콜로, 헤더 크기가 작고 네트워크 대역폭을 효율적으로 활용하여 실시간 데이터 전송에 매우 적합하다. 또한, 토픽(Topic) 기반 메시징을 지원하며, 클라이언트는 관심 있는 토픽을 구독하여 해당 주제의 메시지를 수신한다. MQTT는 QoS(Quality of Service) 수준을 통해 메시지 전송 신뢰성을 조절할 수 있어, 다양한 응용 분야에서 데이터 통신의 핵심 역할을 하며, 중개자 또는 브로커(Broker)를 통해 메시지를 중개하므로 효율적인 메시지 라우팅과 클라이언트 간 통신을 관리한다. 이러한 특징으로 MQTT는 IoT 및 M2M 응용 분야에서 광범위하게 활용되며, 경량성과 높은 신속성을 제공하여 데이터 통신에 효과적으로 사용된다. 이러한 이유로 무드등 디바이스는 MQTT 프로토콜을 이용하여 사용자 모바일 어플리케이션으로부터 발행되는 제어 메세지

를 제어 토픽을 구독함으로써 전달받을 수 있다. 사용자 모바일 어플리케이션으로부터 받을 수 있는 제어 메세지는 4종류로 LED ON/OFF, LED 점등 지속 시간, LED 밝기 단계, 초음파 센서 감지 거리가 있으며, 각 제어 메세지가 도착하면 정해진 해당 기능을 수행한다.

HTTP는 프로토콜은 클라이언트와 서버 간에 웹 데이터를 교환하는 프로토콜로, 클라이언트가 서버에게 요청을 보내면 서버가 해당 요청에 대한 응답을 보내는 방식으로 동작한다. 이는 무상태(Stateless)이며, 각각의 요청은 이전 요청과 독립적인 특성이 있다. HTTP는 주로 웹 브라우저와 웹 서버 간에 사용되며, 텍스트 기반의 메시지를 주고받아 웹 페이지를 로드하고 상호작용하는 데 사용된다. 앞서 설명한 바와 같이 보행자가 한 위치에서 일정시간동안 이동하지 않는다면 디바이스의 HC_SR04 초음파 센서에 일정 시간 이상 사물이 감지되어 보행자에게 낙상사고가 발생한것으로 추정하고 클라우드 서버에 unitCode 정보를 body에 포함한 HTTP Request Post Method를 수행하여 등록된 사용자 휴대폰으로 낙상사고 발생 의심 장소 정보를 SMS를 통해 전달한다.

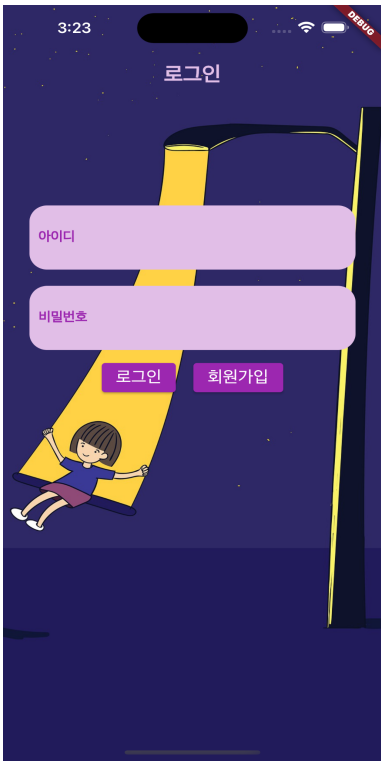
제 3 장 사용자 모바일 어플리케이션

[표2. 모바일 어플리케이션 개발환경]

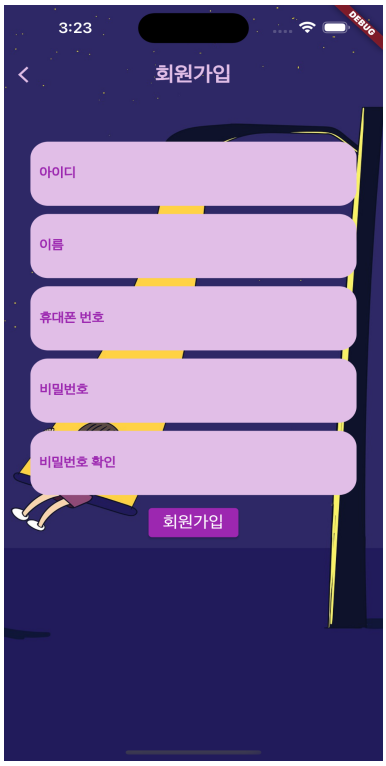
Target	Version
Development OS	macOS Sonoma 14.0 ver
Framework	Flutter 3.7.8 ver
Physical Test Device	iphone 8 plus
Virtual Test Device	iphone 15 pro

제 1 절 Flutter 프레임워크 어플리케이션 설계

본 절에서는 Flutter 프레임워크를 사용하여 모바일 어플리케이션을 개발하는 과정과 주요 기능에 대해 상세히 설명한다. 어플리케이션 최초 실행 시 사용자는 [로그인 화면]을 볼 수 있다. 사용자 계정이 존재한다면 아이디와 비밀번호를 입력 후 '로그인' 버튼을 눌러 [메인 화면]으로 이동할 수 있다. 사용자 계정이 존재하지 않는다면 '회원가입' 버튼을 눌러 [회원가입 화면]으로 이동하여 회원가입을 진행할 수 있다. 사용자는 [회원가입 화면]에서 사용자 아이디, 이름, 휴대폰 번호, 비밀번호를 등록함으로써 회원가입을 진행할 수 있도록 설계하였다. 그 후 가입이 완료된 아이디와 비밀번호를 이용하여 [로그인 화면]에서 생성한 회원 계정으로 로그인할 수 있도록 설계하였다.



[그림6. 로그인 화면]

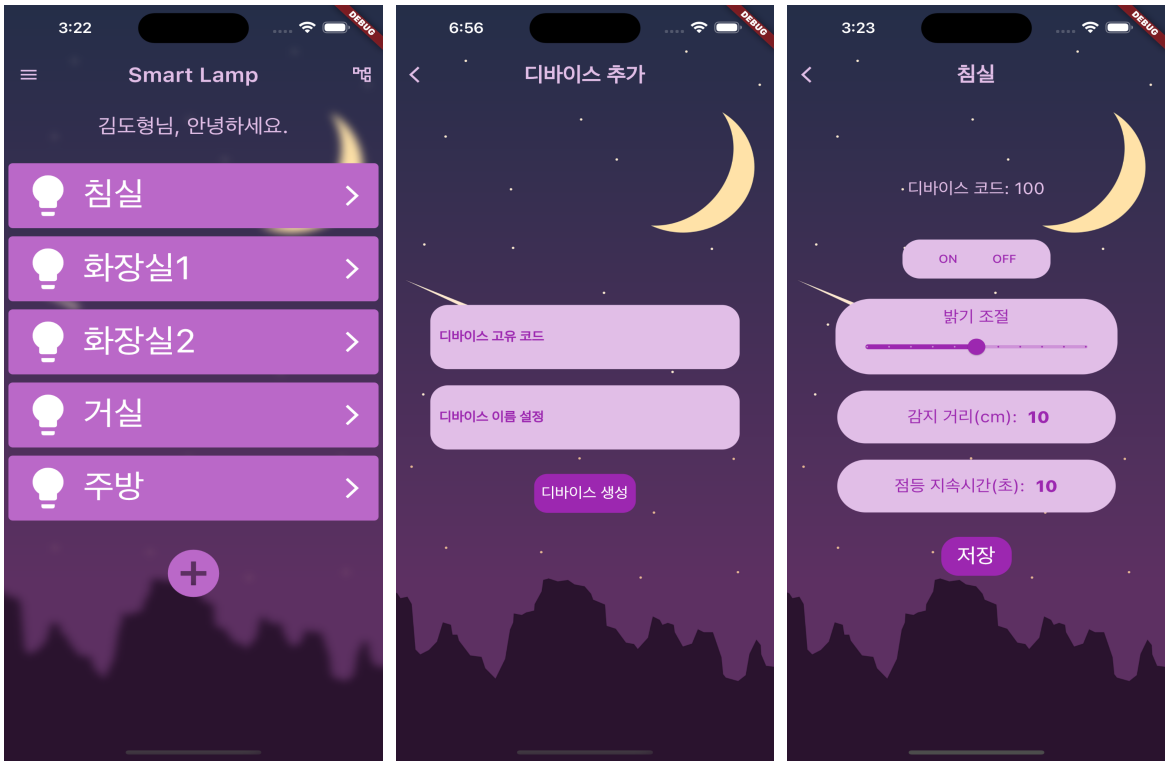


[그림7. 회원가입 화면]

사용자가 로그인 후 처음 마주하는 화면은 [메인 화면]이다. 사용자가 사전에 등록한 디바이스가 있다면 사용자의 다바이스 목록 리스트를 보여주며, 등록한 디바이스가 없다면 아래 '+' 버튼을 눌러 [디바이스 추가 화면]으로 이동할 수 있다. 사용자는 디바이스 추가 화면에서 보유한 디바이스의 고유 코드를 입력 후 디바이스를 구분할 이름을 입력함으로써 사용자 계정에 디바이스 정보를 추가할 수 있다. 반면, 사용자가 등록한 디바이스가 있다면 메인 화면의 디바이스 리스트 중 하나를 선택하여 [디바이스 세부 설정 화면]으로 이동할 수 있다.

사용자가 디바이스 세부 설정 화면으로 이동했다면, 해당 디바이스 코드를 확인할 수 있으며 LED 램프의 ON/OFF 동작을 수행하도록 버튼을 눌러 제어할 수 있다. 그리고 밝기 조절 슬라이더를 이용하여 해당 디바이스의 밝기를 육안으로 실시간으로 확인하면서 1~10단계로 빛의 세기를 조절할 수 있다.

또한 감지 거리 버튼과 점등 지속시간 버튼을 클릭하여 디바이스의 초음파 센서가 감지하는 보행자 감지 거리와 LED 램프의 점등 지속시간을 스크롤 리스트로 선택할 수 있다. 이 과정에서 설정한 디바이스 설정 값들은 실시간으로 디바이스에 업데이트 되지만, 클라우드 서버의 데이터에 업데이트 하기 위해서는 '저장' 버튼을 눌러야한다.



[그림8. 메인 화면] [그림9. 디바이스 추가 화면] [그림10. 디바이스 세부 설정 화면]

제 2 절 데이터 통신

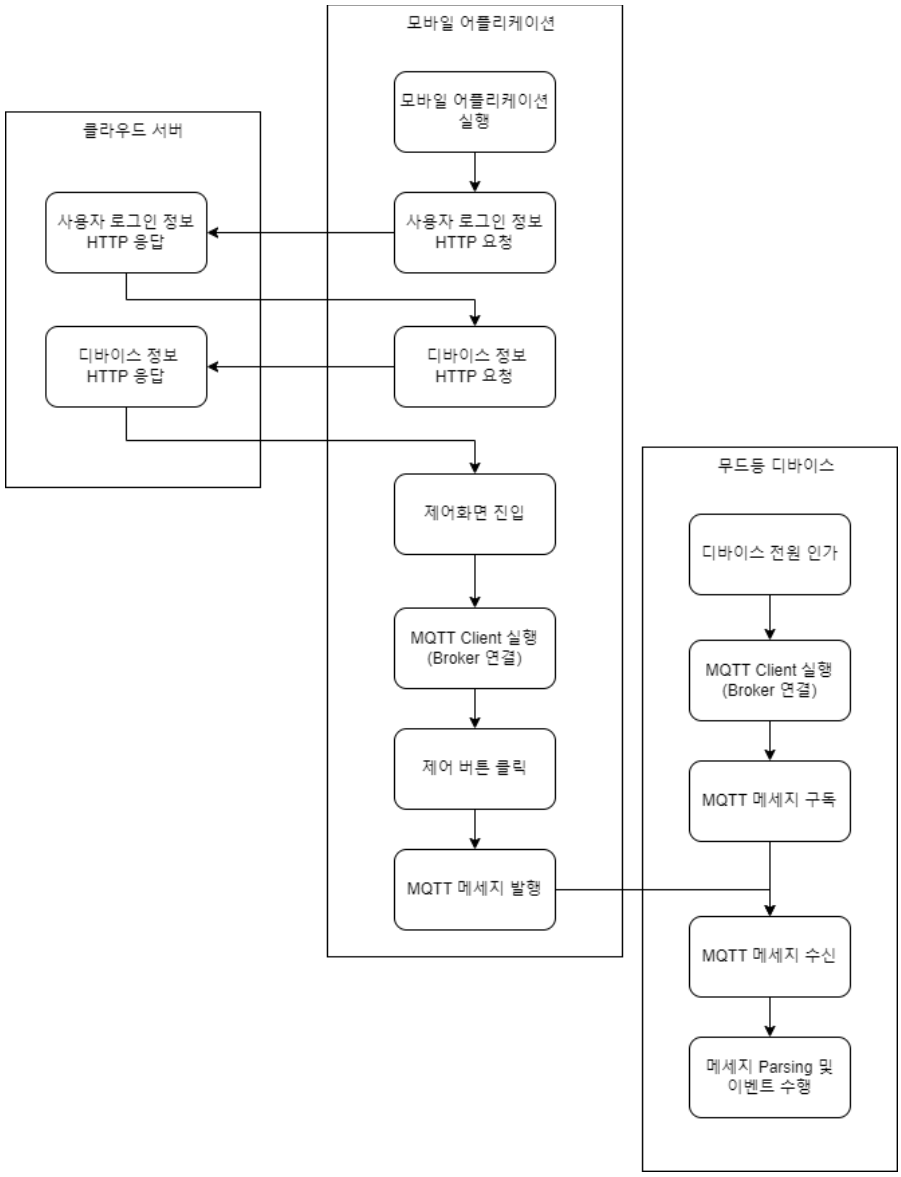
본 절에서는 사용자 모바일 어플리케이션의 관점에서 무드등 디바이스와 클라우드 서버 간의 데이터 통신을 위한 프로토콜 설계에 대해 논의한다. 또한 MQTT 프로토콜과 HTTP 프로토콜을 모바일 어플리케이션에서 어떻게 활용하는지에 대한 내용을 다룬다.

mqtt_client 라이브러리는 Dart언어에서 MQTT 클라이언트를 구현한 라이브러리로, MQTT 브로커와의 통신 설정, 메시지 발행 및 구독을 처리한다. 이를 통해 Flutter 어플리케이션은 MQTT 프로토콜을 활용하여 데이터를 교환하고, IoT 기기 및 다른 애플리케이션과의 효과적인 통신을 가능하게 한다. 다음은 [그림 11. 데이터 통신 플로우차트]의 동작을 설명한다.

먼저, 어플리케이션의 mqtt client를 클라우드 서버에 구성한 MQTT 브로커에 연결을 요청한다. 그 후 앞서 제 2 장 무드등 디바이스의 제 2 절에서 설명한 데이터 통신에서 설명한 바와 같이 [디바이스 세부 설정 화면]에서 LED ON/OFF, LED 점등 지속시간 설정, LED 밝기 조절, 보행자 감지 거리 설정 등 특정 토픽으로 제어 명령을 발행하는 동작을 통해 무드등 디바이스가 해당 토픽으로 제어 명령을 받을 수 있다.

다음으로 모바일 어플리케이션은 HTTP 프로토콜을 사용하여 클라우드 서버와 양방향 통신의 상호작용으로 사용자 및 장치 정보를 관리한다. 즉, 클라이언트 애플리케이션이 사용자 관리와 장치 정보 관리를 위한 API 요청을 수행한다.

먼저, “joinUser“ 함수는 사용자 등록을 처리한다. 사용자의 아이디, 비밀번호, 이름, 전화번호를 입력으로 받아 서버에 등록 요청을 보내고, 등록이 성공적으로 이루어지면 “join success“ 메시지를 출력한다. 다음으로, “loginUser“ 함수는 사용자 로그인을 처리한다. 사용자 아이디와 비밀번호를 입력으로 받아 서버에 로그인 요청을 보내고, 로그인이 성공하면 “login success“ 메시지를 출력하고 서버로부터의 응답을 반환한다. “logoutUser“ 함수는 사용자 로그아웃을 담당하며, 현재 로그인된 사용자 정보를 서버로 보내 로그아웃을 요청한다. 성공적으로 로그아웃되면 “logout success“ 메시지를 출력한다. “getUnitModelList“ 함수는 사용자의 장치 목록을 서버에서 가져오는 역할을 한다. 현재 로그인된 사용자 정보를 서버에 보내서 해당 사용자의 장치 목록을 가져오고, 이 정보를 UnitModel 객체로 파싱하여 반환한다. “postUnitInfo“, “patchUnitInfo“, 그리고 “deleteUnitInfo“ 함수는 장치 정보를 서버에 생성, 업데이트, 삭제하는 역할을 한다. 각각 새로운 장치 정보를 생성하거나, 기존 정보를 업데이트하거나, 삭제하는 요청을 서버에 보내고 그 결과에 따라 메시지를 출력한다.



[그림 11. 데이터 통신 플로우차트]

제 4 장 클라우드 서버 설계

[표3. 클라우드 서버 개발 환경]

Target	Version
Deployment OS	Ubuntu 14.04.6 LTS (groomide)
Development OS	macOS Sonoma 14.0 ver
Java Development Kit	openjdk 11 ver
Framework	Spring Boot 2.7.10
MySQL	mysql Ver 14.14
MQTT Broker	mosquitto version 1.6.3

제 1 절 MQTT 브로커

MQTT Broker(이하 “브로커“)는 MQTT 프로토콜을 기반으로 클라우드 서버에서 중요한 역할을 수행한다. 브로커는 메시징 시스템의 핵심 구성 요소로서, 다양한 장점과 단점을 가지고 있다. 먼저, MQTT 브로커의 역할을 간단히 설명하면, 브로커는 클라이언트 간의 효율적인 메시지 전달을 중개하고 관리하는 역할을 한다. 이는 다음과 같은 장점을 가진다.

브로커는 메시지의 발행(Publish)과 구독(Subscribe)을 관리하므로 클라이언트는 서로 직접 통신할 필요 없이 브로커를 통해 메시지를 교환할 수 있다. 이러한 중개 역할은 네트워크 트래픽을 최소화하고, 메시지 전달의 신뢰성과 성능을 향상시킨다. 또한, MQTT 브로커는 다양한 클라이언트 간에 표준화된 프로토콜을 제공하여 다양한 플랫폼과 언어에서 MQTT를 사용할 수 있도록 한다. 이로써 다양한 종류의 디바이스 및 시스템 간에 상호 작용이 용이하다.

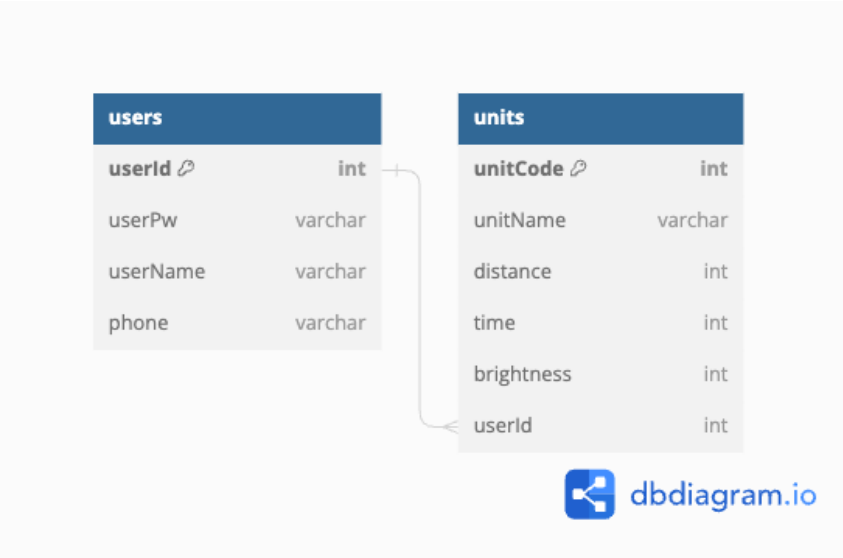
그러나 MQTT 브로커도 일부 단점을 가지고 있다. 첫째, 중요한 역할을 하는 만큼, 브로커의 가용성과 성능은 항상 유지되어야 한다. 브로커의 장애는 메시지 전달의 문제를 야기할 수 있으므로 이에 대한 관리와 백업 시스템이 필요하다. 둘째, MQTT 브로커를 사용하려면 추가적인 인프라스트럭처와 리소스가 필요하며, 운영 및 유지보수 비용이 소요될 수 있다.

따라서 MQTT 브로커는 메시징 시스템에서 중요한 역할을 수행하여 메시지 교환의 효율성과 신뢰성을 향상시키지만, 가용성 및 유지보수와 같은 측면에서도 주의해야 할 중요한 구성 요소이다.

제 2 절 MySQL 데이터베이스

본 연구에서 사용된 데이터베이스 스키마는 사용자(User)와 디바이스 유닛(Unit) 간의 관계를 시각적으로 나타내는 ERD를 포함하고 있다. 이 ERD는 데이터베이스 내의 두 테이블, 즉 'users'와 'units'를 중심으로 구성되었으며, 이 두 테이블 간에는 'user : unit = 1 : N' 관계가 형성되어 있다. 첫 번째 테이블인 'users'는 사용자 정보를 담고 있으며 'userId'를 주요 키(primary key)로 사용한다. 이 테이블은 사용자의 고유 아이디('userId'), 비밀번호('userPw'), 사용자 이름('userName'), 연락처(전화번호, 'phone')와 같은 속성들을 포함하고 있다. 두 번째 테이블인 'units'는 디바이스 유닛 정보를 저장하며 'unitCode'를 주요 키로 사용한다. 'units' 테이블은 디바이스의 고유 코드('unitCode'), 디바이스 이름('unitName'), 탐지 거리('distance'), 점등 시간('time'), 밝기('brightness')와 같은 디바이스 유닛에 관한 정보를 담고 있다. 이 테이블은 'userId'라는 외래 키(foreign key)를 가지며, 'users' 테이블과의 관계를 형성하고 있다.

이러한 ERD를 통해 'users' 테이블과 'units' 테이블 사이의 관계가 명확하게 표현되었으며, 하나의 사용자가 여러 디바이스 유닛을 등록할 수 있는 '1 : N' 관계임을 시각적으로 확인할 수 있다.



[그림12. ERD 다이어그램]

제 3 절 Spring 프레임워크 REST API 설계

본 연구에서는 Java Persistence API(JPA)를 사용하여 데이터베이스와 Java 언어 객체 간의 매핑을 정의하고 있으며, 객체-관계 매핑(ORM)을 구현하기 위한 핵심 코드를 포함하고 있다. JPA는 Java 애플리케이션과 관계형 데이터베이스 간의 상호 작용을 단순화하고 추상화하기 위한 기술로, 객체 지향 애플리케이션과 관계형 데이터베이스 간의 연동을 수행한다. JPA는 객체 지향 프로그래밍 언어인 Java와 관계형 데이터베이스 사이의 패러다임 불일치를 해결하기 위한 기술로, Java 객체와 데이터베이스 레코드 간의 매핑 및 상호 작용을 단순화하고 표준화한다.

'User'와 'Unit' 클래스는 각각 'user'와 'unit' 테이블과 매핑되며, 클래스의 멤버 변수와 테이블의 컬럼 사이의 관계를 어노테이션을 통해 정의한다. '@Entity' 어노테이션은 클래스가 JPA 엔터티임을 나타내며, '@Id' 어노테이션은 주요 키(primary key)를 정의한다. '@ManyToOne'과 '@OneToMany' 어노테이션은 엔터티 간의 관계를 정의하고, 'mappedBy' 속성은 양방향 관계를 설정한다. 이러한 JPA 코드를 통해 객체와 데이터베이스 간의 매핑 및 관계를 손쉽게 설정할 수 있으며, 이는 애플리케이션 개발을 효율적으로 수행할 수 있도록 도와준다. 또한, JPA는 데이터베이스 스키마와의 일관성을 유지하고 데이터베이스 조작을 추상화함으로써 개발 생산성을 향상시키는 데 도움이 된다.

UnitController는 스마트 램프 어플리케이션의 디바이스 정보를 관리하는 역할을 한다. RESTful API를 통해 디바이스 정보의 생성, 반환, 수정, 삭제 작업을 수행한다. 디바이스 정보 등록 API에서는 클라이언트로부터 디바이스 정보를 받아와 데이터베이스에 등록한다. 디바이스 정보 반

환 API는 사용자 정보를 받아 해당 사용자와 연관된 디바이스 정보를 반환한다. 디바이스 정보 수정 API는 수정할 디바이스 정보를 받아 데이터베이스의 디바이스 정보를 수정한다. 디바이스 삭제 API에서는 삭제할 디바이스 정보를 받아 데이터베이스에서 해당 디바이스 정보를 삭제한다.

UserController는 사용자 정보를 관리하고 사용자의 인증과 로그아웃을 처리한다. RESTful API를 통해 사용자 정보의 등록, 인증, 로그아웃 작업을 수행한다. 사용자 정보 등록 API에서는 클라이언트로부터 사용자 정보를 받아와 데이터베이스에 등록한다. 사용자 인증 API는 클라이언트로부터 사용자 정보를 받아 사용자 인증을 수행하고, 유효한 사용자인 경우 사용자 정보를 반환한다. 사용자 로그아웃 API는 클라이언트로부터 사용자 정보를 받아 사용자 로그아웃을 처리한다.

이 두 컨트롤러는 스마트 램프 어플리케이션의 핵심 로직을 노출하는 API 엔드포인트를 제공하며, 클라이언트와 어플리케이션 간의 효율적인 통신을 가능하게 한다.

sendSms 클래스는 스마트 램프 시스템의 백엔드 엔드포인트로 사용되는 Spring Framework 기반의 RESTful API를 구현한 Java 클래스로, 메시지 관련 기능을 처리하는 데 사용된다. 이 API는 스마트 램프 시스템의 핵심 기능 중 하나로, 사용자에게 SMS 알림을 보내고 SMS 서비스의 잔액을 확인하는 데 중점을 둔다.

의존성 주입과 환경 설정은 이 클래스의 생성자에서 이루어진다. 생성자는 UserRepository, UnitRepository, 및 Environment를 의존성 주입으로 받아온다. Environment는 설정 파일에서 API 키 및 비밀 키를 가져오는 데 사용되며, 이러한 키는 SMS 서비스와의 통신을 설정하는 데 필요하다.

다음으로, 두 개의 POST 엔드포인트인 “/send-test“와 “/send-one“가 정의되어 있다. “/send-test“ 엔드포인트는 특정 스마트 램프 유닛에 테스트 SMS 메시지를 보내기 위해 사용된다. 스마트 램프 디바이스에서 낙상사고를 감지하여 해당 이벤트를 전달하기 위해 위 엔드포인트로 HTTP Request를 요청하면, 엔드포인트는 요청 본문에서 UnitInfoDto객체를 받아 해당 유닛을 검색하고, 해당 유닛의 사용자에게 SMS 메시지를 보낸다. 이 메시지는 사용자와 유닛 정보를 포함하며 Message객체로 반환됩니다. 반면에 “/send-one“ 엔드포인트는 /send-test와 유사하지만, 실제 SMS를 보내지 않고 SMS 메시지를 전송 서비스에 보내는 데 사용된다. SingleMessageSendingRequest를 통해 메시지를 전송하고 SingleMessageSentResponse를 반환한다.

마지막으로, GET 엔드포인트 “/get-balance“는 SMS 서비스의 현재 잔액을 조회한다. 이 엔드포인트는 SMS 서비스의 잔액 정보를 포함한 Balance객체를 반환한다.

3. 연구 결과

제 1 장 Wi-Fi Latency, MQTT Latency 측정

(1) 측정 방법

본 연구결과에서는 스마트 무드등 디바이스의 실제 무선 통신 성능을 측정하여 실제 활용할 수 있는 수준의 Latency를 지원할 수 있는지 확인하기 위하여 WiFi Latency와 MQTT Latency를 측정한다. 연구를 진행한 디바이스의 Wi-Fi Latency와 MQTT Latency를 측정하기 위해 C언어를 이용하여 각 클라이언트의 접속 소요시간을 각 10회씩 측정하였다.

(2) 측정 결과

그 결과 다음과 같은 결과를 얻을 수 있었다. Wi-Fi Latency는 평균 505ms, 표준편차 0ms, 분산 $0ms^2$ 으로 측정되었으며 MQTT Latency의 경우 평균 363.8ms, 표준편차 70.4ms, 분산 $4956.56ms^2$ 으로 측정되었다. 이는 MQTT Latency의 측정값이 상당한 변동을 가지고 있다는 것을 나타낸다. 이는 MQTT 브로커와 클라이언트의 버퍼 및 스레드 조절을 통해 통신 최적화로 개선이 필요하다.

[표4, Wi-Fi Latency, MQTT Latency 10회 측정 결과]

회차	Wi-Fi Latency	MQTT Latency
1	505ms	313ms
2	505ms	469ms
3	505ms	313ms
4	505ms	324ms
5	505ms	311ms
6	505ms	473ms
7	505ms	471ms
8	505ms	328ms
9	505ms	324ms
10	505ms	312ms

제 2 장 LED램프 밝기 제어 성능

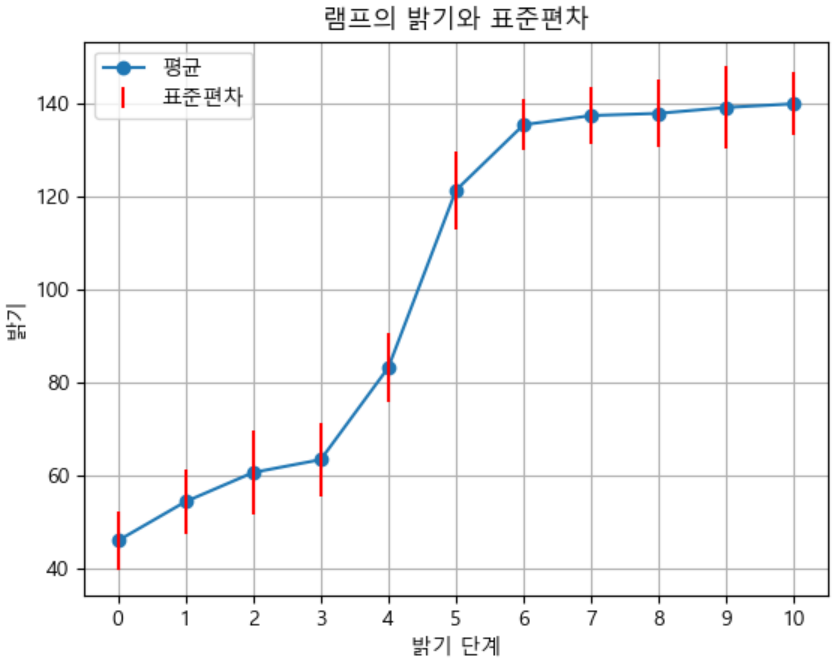
(1) 측정 방법

본 연구결과에서는 스마트 무드등과 모바일 어플리케이션의 실사용 시 사용자의 의도에 맞게 밝기 조절이 가능한지 확인하기 위하여 밝기 단계 별 실제 밝기를 측정한다. 밝기 단계란 모바일 어플리케이션의 0~10단계의 제어 기능상의 밝기 단계이며 이와 함께 변화하는 LED Driver의 펄스 폭 변조 듀티 사이클에 의존한다. 하드웨어 LED 밝기 성능을 측정하기 위해 카메라로 이미지 사진을 촬영하고 Python의 OpenCV 라이브러리를 사용하여 밝기 성능을 측정하였다. 실험에서 사용된 이미지는 아이폰 14 Pro의 광학 카메라로 촬영되었으며, 이로 인해 낮은 단계에서의 밝기 측정이 어렵다는 문제가 있었다. 이에 대한 해결책으로 LED단자 주변이 촬영된 중앙 부분을 추출하여 밝기를 계산하는 방법을 채택하였다. 실험 횟수는 각 단계별 10회씩 밝기를 측정 후, 각 단계 포인트에서 평균을 해당 단계의 밝기 대푯값으로 가정한 후 편차를 제시한다.

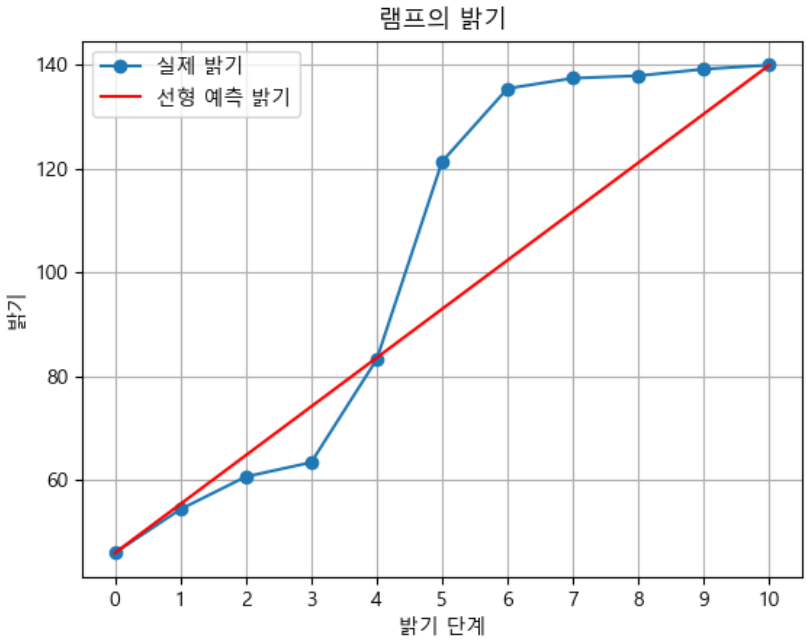
(2) 측정 결과

실험 결과, 각 단계별 램프의 밝기는 다음과 같이 측정되었다. 램프의 밝기 그래프의 경우 시그모이드 함수의 곡선 모양과 닮은 형태로 측정되었다. 밝기 값을 이상적인 선형그래프와 오차를 비교해본 결과 각 밝기 단계에서의 평균 밝기를 대푯값으로 가정 시 평균제곱오차(Mean Squared Error, MSE)는 277.28이다.

연구 결과에서 낮은 단계에서의 밝기 측정 데이터의 포화가 나타난 이유는 아이폰 14 Pro의 광학 카메라가 촬영 시 주변의 밝기와 LED 램프의 낮은 밝기 차이에 대한 민감도 한계에 도달하여 더 이상의 변화를 정확하게 감지하지 못하기 때문이다. 또한 밝기 6단계 이상의 LED 램프의 밝기 변화량을 육안으로 확인 시 뚜렷한 밝기 변화량을 확인 할 수 있었지만, 광학 카메라로 측정 시 높은 단계에서 밝기의 포화 문제가 발생하였다. 이는 높은 단계의 LED 램프의 밝기가 수광소자의 인식 범위를 벗어난 문제로 인해 발생한 것으로 나타난다. 이러한 문제를 보완하기 위해 수광소자의 인식 범위를 벗어나지 않도록 적절한 실험 환경과 조명을 설정하는 것이 필요하다. 또한, 포화 문제를 최소화하기 위해 카메라의 민감도 및 측정 범위를 고려하여 실험을 설계하는 것이 필요하다.



[그림13. 단계별 측정 포인트에서 평균 및 편차 그래프]



[그림14. 단계별 실제 램프의 밝기와 선형그래프와의 밝기 비교 결과]

[표5. 단계별 실제 램프의 밝기와 선형그래프와의 밝기 오차 결과]

단계	밝기 대표값	밝기 오차
0	46.0	0.0
1	45.4	1.0
2	60.7	4.1
3	63.4	10.8
4	83.3	0.3
5	121.4	28.4
6	135.5	33.1
7	137.5	25.7
8	137.9	16.7
9	139.2	8.6
10	140.0	0.0

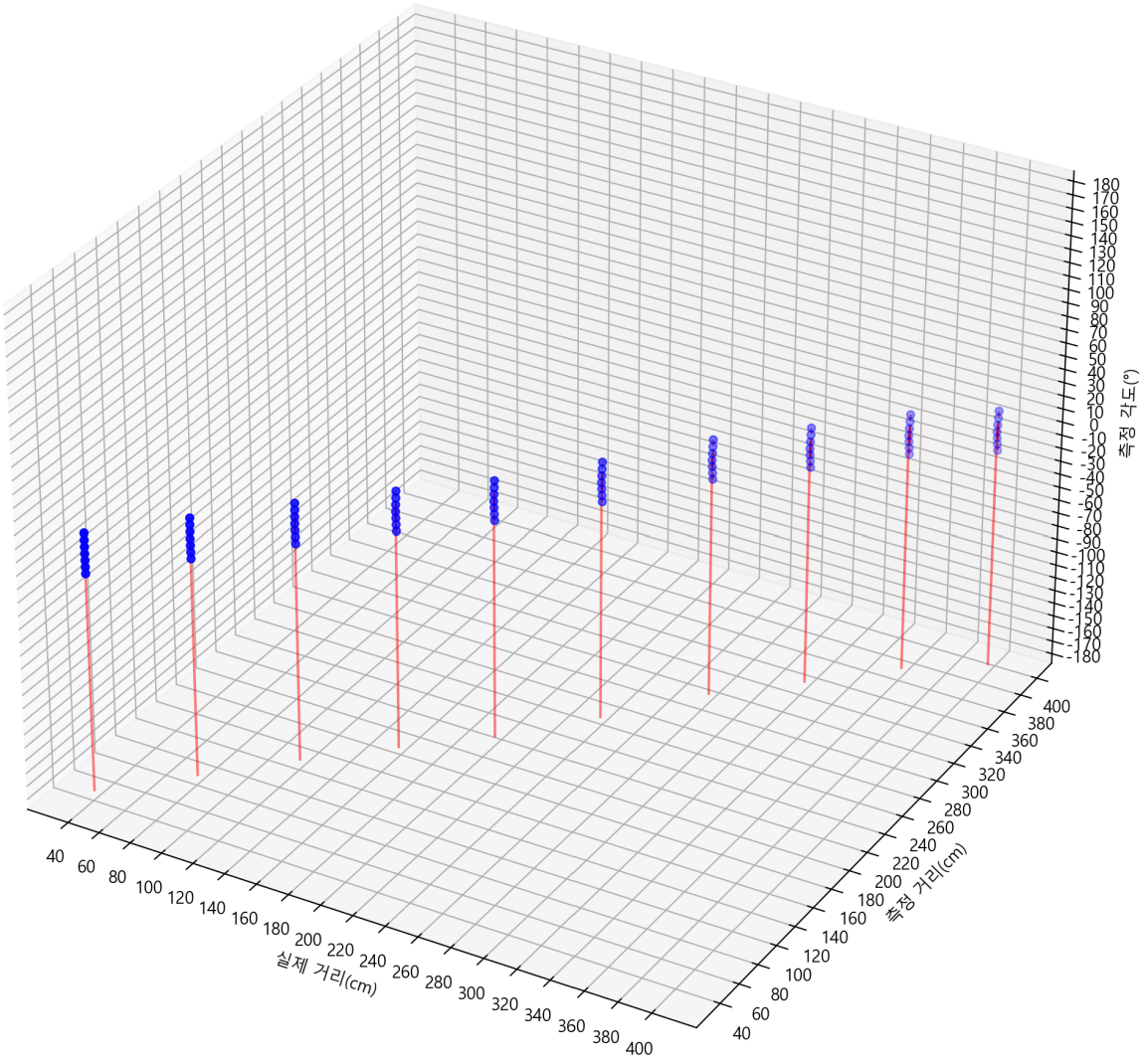
제 3 장 초음파 센서 감지 거리 및 방사패턴 측정

(1) 측정 방법

본 연구결과에서는 보행자 거리 위치 계산을 위한 초음파 센서 도입을 고려하기 위하여 해당 초음파 센서의 실제 거리 별 측정 거리의 정확도와 초음파 방사 패턴을 측정하고자 한다. 해당 센서의 데이터 시트의 스펙 상 감지 범위인 2cm ~ 400cm 내에서 40cm 간격으로 특정 거리 당 10회 물체와의 거리를 측정하여 실제 물체와의 거리와 계산된 물체와의 거리의 평균을 대푯값으로 측정한다. 또한 각도별 측정 거리 -180도 ~ 180도 범위에서 5도 간격의 각도별 측정 거리를 수집함으로써 초음파 센서의 방사 패턴을 확인한다.

(2) 측정 결과

측정 결과 실제 거리 400cm 범위 내에서 물체의 거리를 측정이 가능한 것으로 나타났다. 또한 초음파 센서의 방사 패턴을 확인한 결과, 초음파 센서의 중앙으로부터 -15도 ~ 15도까지 원활한 거리 측정이 가능한 것으로 나타났다.



[그림15. 초음파 센서 각도별 실제 거리와 측정 거리]

제 4 장 인체감지센서 최대 보행자 감지 거리 측정

(1) 측정 방법

본 연구결과에서 보행자 움직임 감지를 위한 인체감지센서 도입을 고려하기 위하여 해당 인체감지센서의 최대 보행자 감지 거리를 측정하고자 한다. 인체감지센서는 스펙 상 산란 각도의 정보는 있지만 최대 감지거리의 정보가 존재하지 않기 때문에 실제 최대 보행 감지 거리를 10회 측정하여 해당 결과값의 통계량을 계산한다. 1회 측정 방식의 경우 센서 위치를 기준으로 300cm 지점부터 신체활동이 감지되지 않을 때까지 10cm씩 이동하며 측정하며 마지막으로 감지되지 않은 직전의 위치를 최대 감지 거리로 가정한다.

(2) 측정 결과

측정 결과 인체감지센서의 최대 감지 거리는 평균 396cm, 표준편차 4.899cm, 분산 24.0cm으로 측정되었다. 이는 초음파 센서의 최대 감지 거리와 유사하기 때문에 본 연구의 디바이스의 적용 범위는 보행자로부터 약 4m이내에서 기능을 수행할 수 있다.

[표6. 인체감지센서의 최대 감지 거리 측정 결과]

회차	측정결과
1	390cm
2	400cm
3	400cm
4	400cm
5	390cm
6	400cm
7	390cm
8	400cm
9	400cm
10	390cm

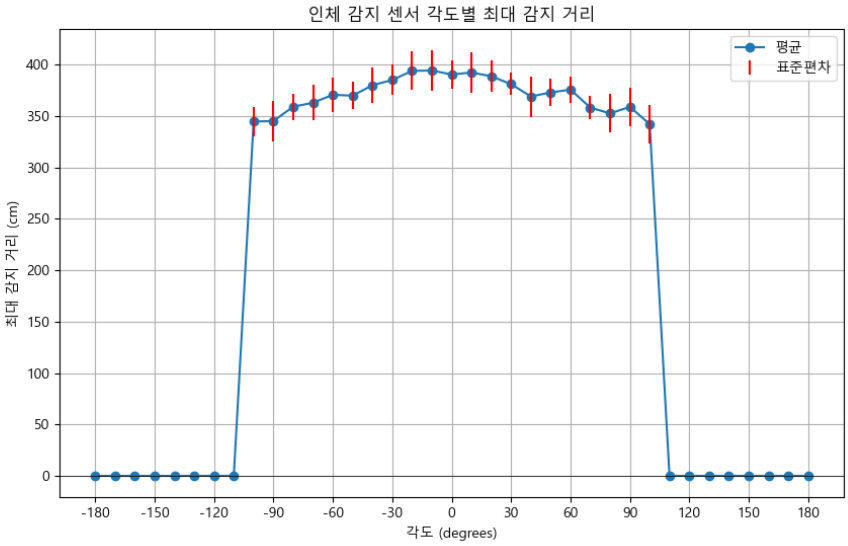
제 5 장 인체 감지 센서 각도별 최대 감지 거리 측정

(1) 측정 방법

본 연구결과에서 보행자 움직임 감지를 위한 인체감지센서 도입을 고려하기 위하여 해당 인체 감지 센서의 각도별 최대 감지 거리를 측정하고자 한다. 인체 감지 센서의 상단 수직방향을 0° 의 기준점이라고 가정한다. 이후 인체 감지 센서의 옆면에서 본 단면상에서 -180° ~ 180° 까지의 각도 범위 내에서 10° 씩 이동하며 최대 감지 거리를 각 각도 포인트 별 10회씩 측정한다. 이후 각 포인트 별 최대 감지 거리의 편차를 계산하며, 평균값을 대푯값으로 가정 한다.

(2) 측정 결과

측정 결과 인체 감지 센서의 최대 감지 각도는 +100도 ~ -100도 까지 지원하며, 해당 감지 각도 내에서 340cm ~ 400cm 이하의 감지 거리를 지원하는 것으로 나타났다(자세한 측정 결과는 다음 그림 참고).



[그림16. 인체 감지 센서의 각도 별 최대 감지 거리 측정 결과]

4. 결론

본 연구는 낙상사고 예방을 위한 스마트 무드등 시스템을 개발하는 것을 목표로 하고있다. 이 시스템은 야간 보행 시 낙상사고에 취약한 환자들을 위한 안전장치로써 연구되었으며, 스마트 무드등 시스템의 구성 요소에는 크게 3가지로 무드등 디바이스, 모바일 어플리케이션, 클라우드 서버로 구성되어 있다.

무드등 디바이스는 인체감지센서를 통해 보행자의 움직임을 감지하고, LED 램프를 점등한다. 모바일 어플리케이션은 사용자가 무드등 디바이스를 제어할 수 있도록 한다. 클라우드 서버는 MQTT 및 HTTP 프로토콜을 이용하여 무드등 디바이스와 모바일 어플리케이션 간의 통신을 담당한다. 이러한 구성 요소들이 함께 작동하여, 스마트 무드등 시스템은 안전한 보행 환경을 제공한다.

전체적인 시스템의 실험 결과, 첫 번째로, Wi-Fi Latency와 MQTT Latency를 측정하여 디바이스의 무선 통신 성능을 확인했다. 실험 결과, Wi-Fi Latency는 평균 505ms로 안정적이었지만, MQTT Latency는 평균 363.8ms로 변동이 크게 나타났다. 이는 MQTT 브로커와 클라이언트 간의 통신 최적화가 필요하다는 것을 시사한다. 두 번째로, LED 램프의 밝기 제어 성능을 측정했다. 밝기 단계별로 측정을 진행하고, LED 램프의 밝기는 시그모이드 함수 모양의 곡선을 보였다. 그러나 낮은 단계에서의 측정에는 광학 카메라의 민감도 한계로 어려움이 있었고, 고단계에서의 포화 문제도 확인되었다. 세 번째로, 초음파 센서의 감지 거리와 방사 패턴을 확인했다. 측정 결과, 초음파 센서는 400cm까지의 물체를 감지할 수 있었으며, 중앙을 기준으로 -15도에서 15도까지의 각도 범위에서 원활한 거리 측정이 가능했다. 마지막으로, 인체감지센서의 최대 보행자 감지 거리를 평가했다. 측정 결과, 최대 감지 거리는 평균 396cm로 나타났으며, 이는 초음파 센서의 최대 감지 거리와 유사한 수준이었다. 종합적으로, 이 실험 결과를 토대로 디바이스의 성능을 평가하고 향후 개선 방향을 도출할 수 있었다.

본 연구에서 도출된 실험 결과를 토대로 스마트 무드등 디바이스의 개선 방향을 제안한다. 첫째로, MQTT Latency의 변동성이 큰 것으로 확인되었다. 따라서 MQTT 통신의 안정성을 향상시키기 위해 브로커와 클라이언트 간의 통신 프로토콜 및 버퍼 및 스레드 조절을 최적화하는 작업이 필요하다. 둘째로, LED 램프의 낮은 단계에서의 측정 어려움과 높은 단계에서의 포화 문제에 대한 개선이 필요하다. 광학 카메라의 민감도 조절 및 LED 램프의 특성을 고려하여 더 정확한 밝기 조절이 가능하도록 하고, 포화 문제를 최소화하기 위해 더 높은 단계에서의 밝기 측정을 개선해야 한다. 셋째로, 초음파 센서의 측정 결과는 안정적이었지만, 향후 활용을 고려할 때 센서의 감지 범위를 조절하는 등의 최적화가 필요하다. 또한, 초음파 센서의 방사 패턴을 더 다양한 각도와 거리에서 측정하여 정확한 거리 측정을 확보하는 작업이 필요하다. 마지막으로, 인체감지센서의 최대 감지 거리는 어느 정도 수준으로 나타났으나, 센서의 감지 각도 및 정확도를 고려하여 실제 환경에서의 보행자 감지 효율을 향상시킬 필요가 있다. 이러한 개선을 통해 디바이스의 성능을 높일 수 있으며, 사용자 경험을 향상시키는 방향으로 나아갈 수 있다.

참고문헌

[1] 통계청, 장래 인구추계자료 (2021.12)

[2] Kim, M. J. (2004). Associated factors caused by falls of older people in community-dwelling. Unpublished master's thesis, Ewha Woman's University, Seoul. 근관절건강학회지, 18(1), 50-62.

[3] No, J. H. (2006). A comparative study fall risk assesment among nursing homes and geriatric hospitals in a urban city. Unpublished master's thesis, Ewha Woman's University, Seoul.

[4] Rubenstein, L. Z., & Josephson, K. R. (2006). Falls and their prevention in elderly people: what does the evidence show?. The Medical Clinics of North America, 90, 807-824.

부록

```
import 'dart:async';
import 'dart:io';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';

final brokerAddress = dotenv.env['MQTT_BROKER_URL'].toString();
final brokerPort = int.parse(dotenv.env['MQTT_BROKER_PORT'].toString());
class MyMqttClient {
  final MqttClient client;
  MyMqttClient()
      : client = MqttServerClient.withPort(brokerAddress, '', brokerPort);
  Future<void> connect() async {
    client.logging(on: false);
    client.setProtocolV311();
    client.onDisconnected = onDisconnected;
    client.onConnected = onConnected;
    client.onSubscribed = onSubscribed;
    client.pongCallback = pong;
    client.connectionMessage = MqttConnectMessage()
        .withClientIdentifier('mobileClient')
        .withWillTopic('connected')
        .withWillMessage('mobile client connected')
        .startClean()
        .withWillQos(MqttQos.atLeastOnce);
    try {
      await client.connect();
    } on NoConnectionException catch (e) {
      print('client exception - $e');
      client.disconnect();
    } on SocketException catch (e) {
      print('socket exception - $e');
```

```
      client.disconnect();
    }
    if (client.connectionStatus!.state == MqttConnectionState.connected) {
      print('Mosquitto client connected');
    } else {
      print(
          'ERROR Mosquitto client connection failed - disconnecting, status is
${client.connectionStatus}');
      client.disconnect();
      exit(-1);
    }
  }
  Future<void> publishMessage(
      {required String topic, required String msg}) async {
    if (client.connectionStatus!.state == MqttConnectionState.connected) {
      var pubTopic = topic;
      final builder = MqttClientPayloadBuilder();
      builder.addString(msg);
      client.publishMessage(pubTopic, MqttQos.exactlyOnce, builder.payload!);
      print("Publish $topic $msg ${DateTime.now()}");
      await MqttUtilities.asyncSleep(10);
    } else {
      print('ERROR Mosquitto client connection failed - message not published');
    }
  }
  void onSubscribed(String topic) {}
  void onDisconnected() {}
  void onConnected() {}
  void pong() {}
}
```

[소스코드1. MQTT Client 구성]

```
import 'dart:convert';
import 'package:client/models/unit_model.dart';
import 'package:client/service/prefs_service.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:http/http.dart' as http;

Future<void> joinUser(
    String userId, String userPw, String userName, String phone) async {
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.post(
    Uri.parse('$baseUrl/api/user'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: jsonEncode({
      'userId': userId,
      'userPw': userPw,
      'userName': userName,
      'phone': phone
    }),
  );
  if (response.statusCode == 201) {
    print("join success ${response.statusCode}");
```

```
  } else {
    print("Failed to join ${response.statusCode}");
    throw Exception('Failed to join user ${response.statusCode}');
  }
}
Future<String> loginUser(String userId, String userPw) async {
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.post(
    Uri.parse('$baseUrl/api/user/authenticate'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: jsonEncode({
      'userId': userId,
      'userPw': userPw,
    }),
  );
  if (response.statusCode == 200) {
    print("login success ${response.statusCode}");
    return utf8.decode(response.bodyBytes);
  } else {
    throw Exception('Failed to login user ${response.statusCode}');
  }
}
Future<void> logoutUser() async {
  final userInfo = await getUserPrefs();
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.patch(
    Uri.parse('$baseUrl/api/user/authenticate'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: jsonEncode(userInfo),
  );
  if (response.statusCode == 200) {
    print("logout success ${response.statusCode}");
  } else {
    throw Exception('Failed to logout user ${response.statusCode}');
  }
}
Future<List<UnitModel>> getUnitModelList() async {
  final userInfo = await getUserPrefs();
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.post(
    Uri.parse('$baseUrl/api/unit/unitList'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: jsonEncode(userInfo),
  );
  if (response.statusCode == 200) {
    List responseUserInfo = jsonDecode(utf8.decode(response.bodyBytes));
    List<UnitModel> unitModelList = [];
    for (var responseUnitInfo in responseUserInfo) {
      UnitModel unitModel = UnitModel.fromJsonMap(responseUnitInfo);
      unitModelList.add(unitModel);
```

```
      }
      return unitModelList;
    } else {
      print("Failed to get UnitModelList ${response.statusCode}");
      throw Exception('Failed to get UnitModelList ${response.statusCode}');
    }
}
Future<void> postUnitInfo(UnitModel unit) async {
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.post(
    Uri.parse('$baseUrl/api/unit'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: unit.toJson(),
  );
  if (response.statusCode == 201) {
    print("Succeeded to post unit ${response.statusCode}");
  } else {
    print("Failed to post unit ${response.statusCode}");
    throw Exception('Failed to post unit ${response.statusCode}');
  }
}
Future<void> patchUnitInfo(UnitModel unit) async {
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.patch(Uri.parse('$baseUrl/api/unit'),
      headers: {
        'Content-Type': 'application/json',
      },
      body: unit.toJson());
  if (response.statusCode == 201) {
    print("Succeeded to put unit ${response.statusCode}");
  } else {
    print("Failed to post unit ${response.statusCode}");
    throw Exception('Failed to put unit ${response.statusCode}');
  }
}
Future<void> deleteUnitInfo(UnitModel unit) async {
  final baseUrl = dotenv.env['BASE_URL'];
  final response = await http.delete(
    Uri.parse('$baseUrl/api/unit'),
    headers: {
      'Content-Type': 'application/json',
    },
    body: unit.toJson(),
  );
  if (response.statusCode == 204) {
    print("Succeeded to delete unit ${response.statusCode}");
  } else {
    print("Failed to delete unit ${response.statusCode}");
    throw Exception('Failed to delete unit ${response.statusCode}');
  }
}
```

[소스코드2. HTTP Request 동작 구성]

```java
@Entity(name="user")
public class User {
    @Id
    @Column(name = "user_id")
    private String userId;
    @Column(name = "user_pw")
    private String userPw;
    @Column(name = "user_name")
    private String userName;
    @Column
    private String phone;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Unit> unitList;
    private boolean authenticated;
    public String getUserId() {
        return userId;
    }
    public void setUserId(String userId) {
        this.userId = userId;
    }
    public String getUserPw() {
        return userPw;
    }
    public void setUserPw(String userPw) {
        this.userPw = userPw;
    }
    public boolean isAuthenticated() {
        return authenticated;
    }
    public void setAuthenticated(boolean authenticated) {
        this.authenticated = authenticated;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public List<Unit> getUnitList() {
        return unitList;
    }
    public void setUnitList(ArrayList<Unit> unitList) {
        this.unitList = unitList;
    }
    public void addUnit(Unit unit){
        if(!this.unitList.contains(unit)) this.unitList.add(unit);
        if(unit.getUser() != this){
            unit.setUser(this);
        }
```

```
    }
}
```

[소스코드3, User 객체 클래스]

```
@Entity(name="unit")
public class Unit {
    @Id
    @Column(name = "unit_code")
    private String unitCode;
    @Column(name = "unit_name")
    private String unitName;
    private Integer distance;
    private Integer time;
    private Integer brightness;
    @ManyToOne
    @JoinColumn(name = "user_id")
    @JsonBackReference
    private User user;
    public String getUnitCode() {
        return unitCode;
    }
    public void setUnitCode(String unitCode) {
        this.unitCode = unitCode;
    }
    public String getUnitName() {
        return unitName;
    }
    public void setUnitName(String unitName) {
        this.unitName = unitName;
    }
    public Integer getDistance() {
        return distance;
    }
    public void setDistance(Integer distance) {
        this.distance = distance;
    }
    public Integer getTime() {
        return time;
    }
    public void setTime(Integer time) {
        this.time = time;
    }
    public Integer getBrightness() {
        return brightness;
    }
    public void setBrightness(Integer brightness) {
        this.brightness = brightness;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
        if(!user.getUnitList().contains(this)){
            user.addUnit(this);
```

```
        }
    }
}
```

[소스코드4, Unit 객체 클래스]

```
@RestController
@RequestMapping("/api/unit")
public class UnitController {
    private final UnitService unitService;
    @Autowired
    public UnitController(UnitService unitService) {
        this.unitService = unitService;
    }
    // 디바이스 정보 등록 API
    @PostMapping
    public ResponseEntity<Void> createUnit(@RequestBody UnitInfoDto unitInfoDto) {
        try {
            unitService.create(unitInfoDto);
            return ResponseEntity.status(HttpStatus.CREATED).build();
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.CONFLICT).build();
        }
    }
    // 디바이스 정보 반환 API
    @PostMapping("/unitList")
    public ResponseEntity<List<Unit>> getUnitList(@RequestBody UserInfoDto userInfoDto) {
        return ResponseEntity.ok(unitService.getUnitList(userInfoDto));
    }
    // 디바이스 정보 수정
    @PatchMapping
    public ResponseEntity<Void> updateUnit(@RequestBody UnitInfoDto unitInfoDto) {
        try {
            unitService.update(unitInfoDto);
            return ResponseEntity.status(HttpStatus.CREATED).build();
        } catch (NoSuchElementException e) {
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        }
    }
    // 디바이스 삭제 API
    @DeleteMapping
    public ResponseEntity<Void> deleteUnit(@RequestBody UnitInfoDto unitInfoDto) {
        try {
            unitService.delete(unitInfoDto.getUnitCode());
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        } catch (NoSuchElementException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
    }
}
```

[소스코드5. Unit 관련 API Controller]

```
@RestController
@RequestMapping("/api/user")
public class UserController {
```

```
    private final UserService userService;
    private UserController(UserService userService) {
        this.userService = userService;
    }
    // 사용자 정보 등록 API
    @PostMapping
    public ResponseEntity<Void> createUser(@RequestBody UserInfoDto userInfoDto) {
        try {
            userService.register(userInfoDto);
            return ResponseEntity.status(HttpStatus.CREATED).build();
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.CONFLICT).build();
        }
    }
    // 사용자 인증 API
    @PostMapping("/authenticate")
    public ResponseEntity<UserInfoDto> loginUser(@RequestBody UserInfoDto userInfoDto) {
        try {
            UserInfoDto userInfo = userService.login(userInfoDto);
            return ResponseEntity.ok(userInfo);
        } catch (AuthenticationException | NoSuchElementException e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }
    // 사용자 로그아웃 API
    @PatchMapping("/authenticate")
    public ResponseEntity<Void> logoutUser(@RequestBody UserInfoDto userInfoDto) {
        try {
            userService.logout(userInfoDto);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }
}
```

[소스코드6. User 관련 API Controller]

```
@RestController
@PropertySource("classpath:properties/env.properties")
public class sendSms {
    private final Environment environment;
    final DefaultMessageService messageService;
    private final UserRepository userRepository;
    private final UnitRepository unitRepository;

    public sendSms(UserRepository userRepository, UnitRepository unitRepository, Environment
environment) {
        this.userRepository = userRepository;
        this.unitRepository = unitRepository;
        this.environment = environment;
        String apiKey = environment.getProperty("apiKey");
        String apiSecretKey = environment.getProperty("apiSecretKey");
        this.messageService = NurigoApp.INSTANCE.initialize(apiKey, apiSecretKey,
"https://api.coolsms.co.kr");
```

```
    }
    @PostMapping("/send-test")
    public Message send(@RequestBody UnitInfoDto unitInfoDto) {
        try {
            Unit storedUnit = unitRepository.findByCode(unitInfoDto.getUnitCode())
                    .orElseThrow(() -> new NoSuchElementException("No unit Found"));
            User user = storedUnit.getUser();
            Message message = new Message();
            message.setFrom(environment.getProperty("sendFrom"));
            message.setTo(user.getPhone());
            message.setText("[SmartLampService]\n\n" + '`' + storedUnit.getUnitName() + '`' +
"에서 낙상사고가 예상됩니다.");
            return message;
        } catch (NoSuchElementException e) {
            throw new RuntimeException(e);
        }
    }
    @PostMapping("/send-one")
    public SingleMessageSentResponse sendOne(@RequestBody UnitInfoDto unitInfoDto) {
        try {
            Unit storedUnit = unitRepository.findByCode(unitInfoDto.getUnitCode())
                    .orElseThrow(() -> new NoSuchElementException("No unit Found"));
            User user = storedUnit.getUser();
            Message message = new Message();
            message.setFrom(environment.getProperty("sendFrom"));
            message.setTo(user.getPhone());
            message.setText("[SmartLampService]\n\n" + '`' + storedUnit.getUnitName() + '`' +
"에서 낙상사고가 예상됩니다.");
            SingleMessageSentResponse response = this.messageService.sendOne(new
SingleMessageSendingRequest(message));
            return response;
        } catch (NoSuchElementException e) {
            throw new RuntimeException(e);
        }
    }
    @GetMapping("/get-balance")
    public Balance getBalance() {
        Balance balance = this.messageService.getBalance();
        return balance;
    }
}
```

[소스코드7. 낙상사고 SMS알림 API]