통합 ICT 설계 제안서

과제명	**스마트 택배 보관함**		
발표자	**김도현**	**제출일자**	2025.11.13
연구참여자 (담당분야)	팀장: 이성진(팀관리, H/W) 팀원: 김도현(제어 S/W, App) 팀원: 유창민(H/W, 보안, 제어 S/W) 팀원: 고동연(H/W, 제어 S/W)		
조명	GoodBox		

목차

1. 설계과제제목 ····· 3P

2. 설계과제 목적 ····· 3P

3. 프로젝트 제안 배경 ····· 3P

3.1 추진 동기

3.2 현황분석

4. 프로젝트 목표와 기준 ····· 4P

4.1 목표와 기준

4.2 현실적 제한 요소

5. 설계 내용 ····· 5P

5.1 설계 배경 및 기술 조사

5.2 프레임워크

5.3 시스템 아키텍쳐

5.4 기능블록도

5.5 동작 방법

5.6 프로젝트 구현

5.7 프로젝트 핵심 기능 구현

6. 추진 체계 ····· 37P

7. 추진 일정 ····· 38P

8. 문제상황 및 해결 방안 ····· 38P

9. 결론 및 기대효과 ····· 40P

10. 회의록 ····· 42P

1. 설계과제 제목

OTP 코드를 활용해 보안을 강화한 스마트 택배 보관함

2. 설계과제 목적

1) 임베디드 시스템 통합 이해
센서, 모터, LCD, 통신 모듈 등 여러 하드웨어를 하나의 제어 시스템으로 통합하는 과정을 학습한다.

2) IoT(사물인터넷) 통신 구조 학습
택배함 상태를 서버나 스마트폰 앱으로 전송하는 구조를 구현하면서 MQTT, HTTP 등 IoT 프로토콜의 개념과 동작 원리를 익힌다.

3) 보안·인증 기초 이해
랜덤으로 생성한 OTP를 통해 비밀번호 인증 절차를 밟으며 기초적인 접근 제어 메커니즘을 설계하고 학습한다.

4) 시스템 설계 프로세스 경험
요구사항 분석 → 하드웨어 설계 → 제어 알고리즘 구현 → 테스트 및 개선의 전체 과정을 경험함으로써 임베디드 시스템 개발의 프로젝트 수행 절차와 협업 프로세스를 익힌다.

3. 프로젝트 제안 배경

3.1 추진 동기

최근 1인 가구와 원룸 거주자 증가로 인해 택배 이용량이 급격히 늘어나면서, 택배 도난 및 분실 문제가 심각한 사회적 문제로 떠오르고 있다.
특히 낮 시간대에 집을 비우는 경우가 많은 1인 가구 특성상, 택배가 문 앞이나 공동 현관에 장시간 방치되는 일이 잦다. 이로 인해 도난·분실·파손 등의 피해 사례가 지속적으로 발생하고 있으며, SNS나 지역 커뮤니티를 통해 관련 제보가 급증하고 있다.
이러한 사회적 배경 속에서, 비대면 시대에 적합하고 보안성이 강화된 '스마트 택배함'의 필요성이 커지고 있다.

3.2 현황분석

최근 LGU+에서도 '우리집 지킴이 도어캠'(현관 앞의 수상한 움직임이나 택배 도착을 자동으로 녹화하고, 실시간으로 스마트폰에 알림을 전송하는 시스템)을 출시하는 등, 가정 보안을 강화하려는 시도가 활발히 이루어지고 있다.
하지만 CCTV와 도어캠이 보급되었다 하더라도, 범죄를 사전에 완전히 차단하기는 어렵다. 특히 도난이 발생한 이후에는 택배사 측의 보상 체계나 대응 기준이 명확하지 않아, 초기 단계에서의 사전 방지 대책이 무엇보다 중요하다.
현재 시중에 보급된 택배함 제품들은 대부분 기계식 열쇠를 이용한 개폐 방식을 사용하고 있다.
이 방식은 설치가 간단하다는 장점이 있으나, 열쇠 분실·복제 등의 보안 취약점이 존재한다.

따라서 저비용이면서도 보안성과 편의성을 동시에 갖춘 스마트 택배함 시스템을 개발하여, 1인 가구나 원룸 거주자도 안전하게 택배를 수령할 수 있는 환경을 구축할 필요가 있다.

4. 프로젝트 목표와 기준

4.1 목표와 기준

정량적 목표

1) 보안성 강화

-OTP 인증 시스템: 90% 이상의 성공적인 인증률을 목표로, 사용자 인증 시 100%의 정확도로 일회용 비밀번호(OTP)가 생성되고 사용되도록 구현.

-무게 센서: 100% 자동 상태 감지 기능 구현. 택배 보관함에 물품이 들어가면 실시간으로 대기 중 → 보관 중으로 상태가 자동으로 전환.

-카메라 모듈: 물품 보관 시점에 100% 사진 촬영을 자동화하여 AWS 클라우드에 사진을 저장. 택배 분실 시 100% 증거자료 활용 가능.

-모바일 애플리케이션: 앱에서 실시간 배송 현황을 5초 이내로 확인 가능하도록 실시간 데이터 동기화 및 푸시 알림 구현.

2) 시스템 안정성

-모터 제어: 모터 동작의 신뢰성 98% 이상. 앱 또는 키패드에서 모터가 정확히 작동하며, 택배함 열기/닫기 동작이 1초 이내에 이루어지도록 개선.

-앱 연동: 앱과 Raspberry Pi 간 실시간 통신 성공률 99% 달성. 버튼 클릭 후 택배함 동작이 3초 이내에 반영되도록 최적화.

3) 유지보수성

시스템 오류 발생률 2% 이하로 유지. 오류 발생 시 3시간 이내 해결이 가능하도록 시스템 점검 및 대응 프로세스 마련.
모바일 앱과 라즈베리파이의 연동성이 문제가 발생할 경우, 주간 점검을 통해 문제를 해결할 수 있는 시스템 관리 절차를 구현.

4) 비용 효율성

총 프로젝트 예산 15만 원 이하로 유지.
주요 부품(무게 센서, 카메라 모듈, Raspberry Pi 등)의 가격을 예산 내에서 최적화하여 시스템 구축.

4.2 현실적 제한 요소

표 1. 본 설계과제의 현실적 제한요소 항목

현실적 제한 요소들	내 용 (Content)
경제	- 예산 제한: 전체 예산 15만 원 이하로, 저비용 부품을 선택해야 하므로 성능과 비용의 균형을 맞추는 것이 중요하다. - 운영 비용: AWS 서버와 클라우드 저장에 드는 지속적인 비용을 관리해야 한다.
편리	- 사용자 인터페이스: 앱과 Raspberry Pi 간의 실시간 통신 지연 문제와 복잡한 설정은 사용자 불편을 초래할 수 있다. - 설치 및 유지보수: 시스템이 간단하게 설치되고 유지 관리가 용이해야 한다.
윤리	- 데이터 보안: OTP, 개인정보 보호 및 암호화가 필수적이며, 사용자 동의를 받아야 한다. - 환경적 영향: 전자기기와 배터리 사용으로 인한 전자폐기물 문제 해결 필요.
사회	- 1인 가구 보안: 택배 도난 문제를 해결하려면 보안성을 강화하고 사용자들이 시스템을 신뢰할 수 있게 해야 한다. - 기술 접근성: 스마트폰 사용이 어려운 사용자들을 위한 대체 솔루션을 고려해야 한다.

5. 설계 내용

본 프로젝트는 택배 도난 및 분실 문제를 해결하기 위한 IoT 기반 스마트 보관함 시스템을 개발한다. OTP 인증, 무게 센서, 카메라 모듈을 활용하여 택배 보관 전 과정을 자동화하고, 모바일 애플리케이션을 통해 사용자에게 실시간 배송 현황을 제공함으로써 안전하고 편리한 택배 수령 환경을 구축한다.

5.1 설계 배경 및 기술 조사

5.1.1 설계 배경

기존 택배함 시스템은 단순한 물리적 잠금장치만을 사용하여 배송 기사와 수령인 간의 보안 인증이 취약하고, 택배 보관 및 수령 과정을 실시간으로 추적할 수 없다는 한계가 있다. 이에 본 프로젝트는 IoT 기술과 모바일 애플리케이션을 결합하여 배송 전 과정을 자동화하고, 실시간 모니터링과 증거 확보가 가능한 스마트 택배 보관함 시스템을 설계하였다.

5.1.2 핵심 기술 요소

1) OTP(One-Time Password) 인증 시스템

배송 기사와 수령인 간의 안전한 접근 제어를 위해 OTP 기반 인증 시스템을 도입하였다. 사용자가 모바일 앱에서 4자리 일회용 비밀번호를 생성하면, 해당 코드는 데이터베이스에 암호화되어 저장되고 유효기간과 사용 횟수가 관리된다. 배송 기사는 사용자로부터 받은 OTP를 키패드에 입력하여 보관함에 접근할 수 있으며, 코드 사용 이력은 모두 기록되어 보안성과 추적성을 확보한다.

2) 무게 센서 기반 자동 상태 관리

HX711 로드셀을 활용한 무게 센서는 택배 보관함 내부의 물품 유무를 실시간으로 감지한다. 배송 기사가 택배를 보관함에 넣으면 무게 센서가 이를 감지하여 자동으로 상태를 대기중'에서 '보관중'으로 변경하고 보관함을 잠근다.

3) 실시간 이미지 촬영 및 저장

Raspberry Pi Camera Module V2를 통해 택배 보관 시점의 사진을 자동으로 촬영한다. 무게 센서가 물품을 감지하고 보관함이 닫히면 LED가 3~4초간 점등되어 촬영 준비를 알린 후, 카메라가 보관함 내부를 촬영한다. 촬영된 이미지는 AWS 클라우드 서버의 MySQL 데이터베이스에 LONGBLOB 형태로 저장됨과 동시에 Raspberry Pi 로컬 저장소에도 백업된다. 이는 택배 분실 또는 손상 발생 시 명확한 증거자료로 활용될 수 있다.

4) 모바일 애플리케이션 연동

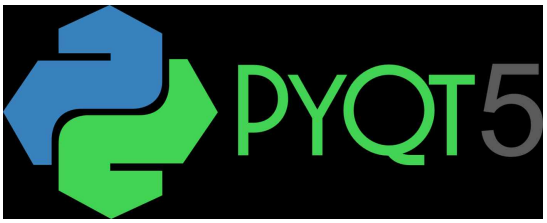
Android 기반 모바일 애플리케이션은 사용자에게 직관적인 인터페이스를 제공한다. 사용자는 앱을 통해 OTP 생성, 택배함 수동 개폐, 배송 이력 조회, 보관 사진 확인 등의 기능을 이용할 수 있다. 앱은 Retrofit 라이브러리를 통해 Node.js 기반 REST API 서버와 통신하며, 실시간으로 배송 상태 정보를 수신하고 푸시 알림을 통해 사용자에게 전달한다.

5) 클라우드 기반 데이터 관리

AWS EC2 인스턴스에 구축된 Ubuntu 서버는 MySQL 데이터베이스와 Node.js API 서버를 운영하며, 모든 배송 데이터를 중앙 집중식으로 관리한다. 사용자 정보, OTP 코드, 배송 상태, 접근 로그, 이미지 데이터 등이 체계적으로 저장되고, RESTful API를 통해 Android 앱과 Raspberry Pi 간의 실시간 데이터 동기화를 지원한다. 이를 통해 확장성과 안정성을 확보하였다.

5.2 프레임 워크

5.2.1 PyQt5



- 개념: PyQt5는 Python에서 GUI 애플리케이션을 개발하기 위한 크로스 플랫폼 프레임워크로, Qt의 강력한 GUI 기능을 Python 언어로 사용할 수 있게 해준다.

- 사용 이유

1) Raspberry Pi에서 4x4 키패드 입력을 시각적으로 표시하기 위한 GUI 구현
2) 비밀번호 입력 현황, 인증 결과, 시스템 상태를 실시간으로 사용자에게 표시
3) 터치스크린 환경에서도 쉽게 조작 가능한 직관적인 인터페이스 제공

- 활용 방식

1) 4x4 매트릭스 키패드 시뮬레이션 GUI 구현 (QGridLayout 활용)
2) 비밀번호 입력 필드, 상태 메시지 라벨 등 위젯 구성
3) pyqtSignal을 활용한 MQTT 메시지와 GUI 간 스레드 안전 통신
4) QTimer를 통한 자동 잠금 카운트다운 기능 구현

5.2.2 Android Studio

- 개념: Android Studio는 Google이 공식으로 제공하는 Android 애플리케이션 개발을 위한 통합 개발 환경(IDE)으로, Java/Kotlin 기반 앱 개발을 지원한다.

- 사용 이유

1) 택배 보관함 제어를 위한 모바일 사용자 인터페이스 개발
2) 배송 상태 조회, OTP 생성, 택배함 원격 제어 등 핵심 기능 구현
3) Material Design을 활용한 직관적이고 현대적인 UI/UX 제공

- 활용 방식

1) Java 언어 기반 Android 앱 개발
2) Material Design Components를 활용한 UI 구성
3) Activity 기반 화면 구성
4) Retrofit2를 통한 REST API 서버와의 HTTP 통신
5) SharedPreferences를 활용한 로컬 데이터 저장 (인증 토큰, 사용자 정보)
6) RecyclerView를 활용한 배송 이력 목록 표시

5.2.3 Node.js

- 개념: Node.js는 JavaScript 기반의 비동기 이벤트 처리 서버 프레임워크로, 빠른 응답성과 높은 확장성을 제공한다.

- 사용 이유

1) 모바일 앱과의 REST API 통신을 효과적으로 처리
2) 데이터 저장, 사용자 인증, 접수 평가 등의 로직 처리에 적합
3) 비동기 처리를 통한 다중 클라이언트 요청의 효율적 관리

- 활용 방식

1) Express.js 프레임워크를 활용한 RESTful API 서버 구축
2) JWT 기반 사용자 인증 및 세션 관리

3) MySQL 데이터베이스 연동을 통한 데이터 CRUD 처리
4) bcrypt를 활용한 비밀번호 암호화
5) MQTT 브로커와 연동하여 Raspberry Pi와 실시간 통신
6) 배송 상태 업데이트, OTP 생성/검증, 접근 로그 기록 등 비즈니스 로직 처리

5.2.4 MySQL

- 개념: MySQL은 오픈소스 관계형 데이터베이스 관리 시스템(RDBMS)으로, 안정적인 데이터 저장과 빠른 쿼리 처리 성능을 제공한다.

- 사용 이유
1) 사용자 정보, 배송 이력, OTP 코드, 접근 로그 등의 구조화된 데이터 관리
2) 트랜잭션 처리를 통한 데이터 무결성 보장
3) 복잡한 관계형 데이터 쿼리 및 조인 연산 지원

- 활용 방식
1) 4개의 핵심 테이블 설계 및 운영
- users: 사용자 계정 정보 및 고유 택배함 비밀번호 관리
- delivery_codes: OTP 코드 생성, 만료, 사용 이력 관리
- deliveries: 배송 상태(대기중/보관중/수령완료), 무게 데이터, 이미지 저장
- box_access_logs: 택배함 접근 이력 및 인증 성공/실패 기록
2) FOREIGN KEY 제약조건을 통한 데이터 참조 무결성 확보
3) INDEX를 활용한 조회 성능 최적화
4) VIEW를 통한 활성 코드 조회 간소화
5) LONGBLOB 타입을 활용한 택배 보관 사진 저장

5.2.5 AWS (Amazon Web Services)

- 개념: AWS는 Amazon이 제공하는 클라우드 컴퓨팅 플랫폼으로, 서버, 데이터베이스, 스토리지 등 다양한 IT 인프라를 온디맨드 방식으로 제공한다.

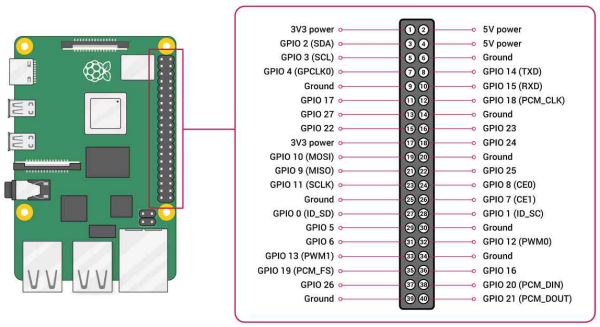
- 사용 이유

1) 안정적이고 확장 가능한 서버 인프라 구축
2) 고정 IP 제공을 통한 안정적인 네트워크 접근성 확보
3) 데이터 백업 및 보안성 강화

- 활용 방식

1) EC2 (Elastic Compute Cloud): Ubuntu 24.04 LTS 기반 가상 서버 운영
   Node.js API 서버 호스팅
   MySQL 데이터베이스 서버 운영
   MQTT 브로커 운영 (Raspberry Pi와 실시간 통신)
2) 탄력적 IP: 고정 공인 IP 주소 할당
   Android 앱에서 안정적인 API 서버 접근
   Raspberry Pi에서 서버로의 일관된 연결 유지
3) 보안 그룹 설정: 포트 3001(API), 3306(MySQL), 1883(MQTT)

5.2.6 RPi.GPIO



- 개념: RPi.GPIO는 Raspberry Pi의 GPIO 핀을 Python에서 제어할 수 있게 해주는 라이브러리로, 하드웨어 제어의 핵심 도구이다.

- 사용 이유

1) 서보모터, 스테핑모터, LED, 부저 등 다양한 하드웨어 제어
2) PWM 신호 생성을 통한 정밀한 모터 제어
3) 키패드 입력 감지 및 처리

- 활용 방식

1) GPIO 핀 모드 설정 (BCM 모드 사용)
2) PWM을 활용한 서보모터 각도 제어 (0도: 잠금, 90도: 해제)
3) 디지털 출력을 통한 스테핑모터, LED, 부저 제어
4) 4x4 매트릭스 키패드 입력 스캔 및 감지

5.2.8 Paho MQTT

- 개념: Paho MQTT는 경량 메시징 프로토콜인 MQTT를 Python에서 사용할 수 있게 해주는 클라이언트 라이브러리로, IoT 디바이스 간 실시간 통신에 최적화되어 있다.

- 사용 이유

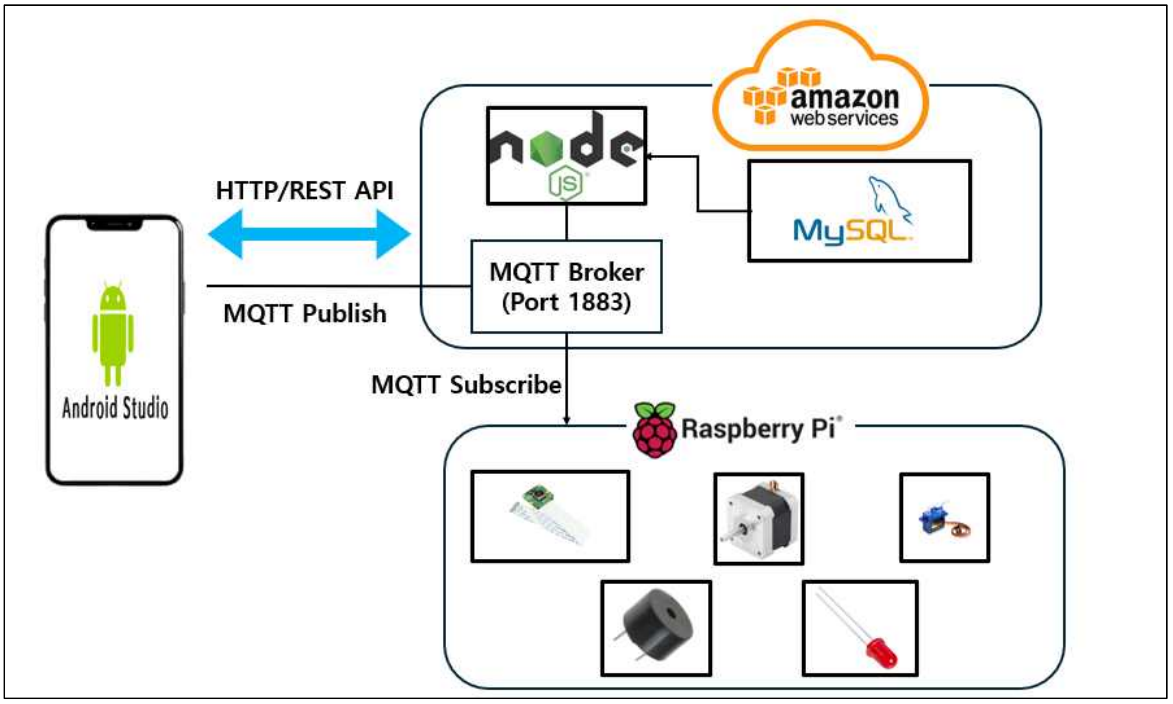
1) Android 앱에서 Raspberry Pi로 실시간 제어 명령 전달 (택배함 열기/닫기)
2) 폴링 방식 대비 효율적이고 즉각적인 양방향 통신
3) 네트워크 대역폭이 제한적인 IoT 환경에 적합

- 활용 방식

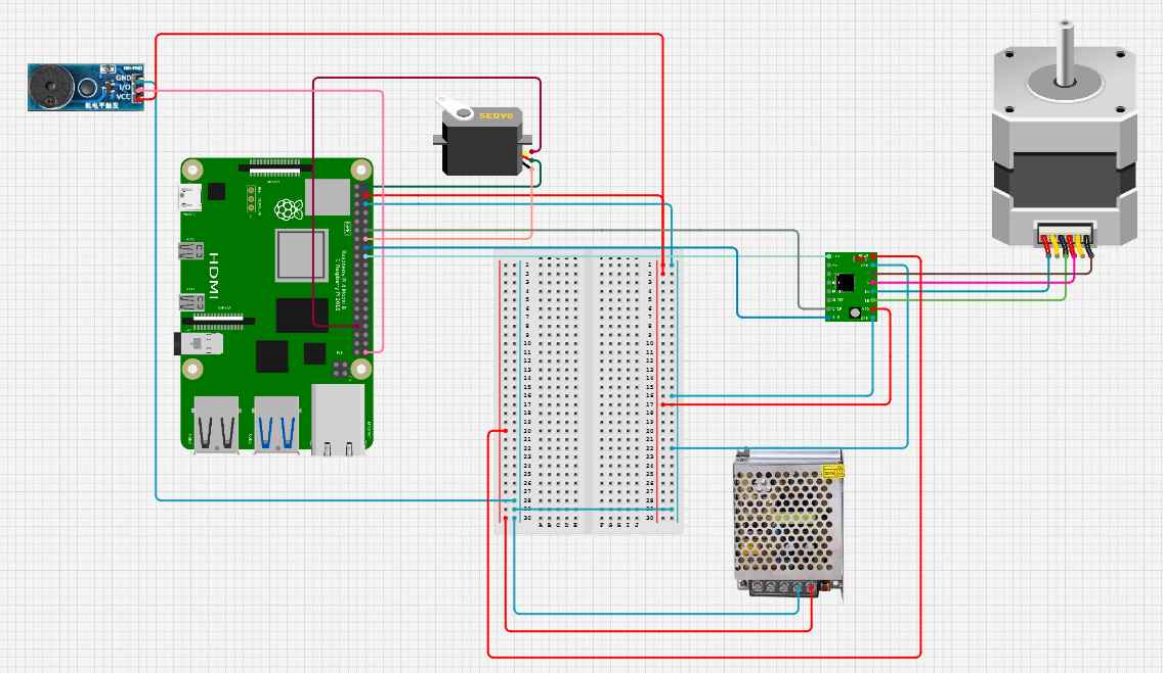
1) MQTT 브로커 연결
2) 토픽 구독: "smartbox/locker/command"
3) 앱에서 publish한 "open"/"close" 명령을 Raspberry Pi에서 수신
4) pyqtSignal을 통해 MQTT 스레드에서 GUI 메인 스레드로 안전하게 신호 전달

5.3 시스템 아키텍쳐

5.3.1 전체 시스템 아키텍처



5.3.2 회로 구성도



1) 라즈베리파이 (중앙 제어 장치)
- 모든 센서와 모터 제어 신호를 처리하는 메인 컨트롤러 역할.
- GPIO 핀을 통해 서보모터, 초음파 센서, 스텝모터 드라이버에 제어 신호를 전달한다.
- 5V 전원은 직접 공급된다.

2) 버저 (소리 출력)
- 라즈베리파이 GPIO에 연결되어 소리 출력을 담당한다.
- 전원은 브레드보드에 연결된 5V로 공급받는다.

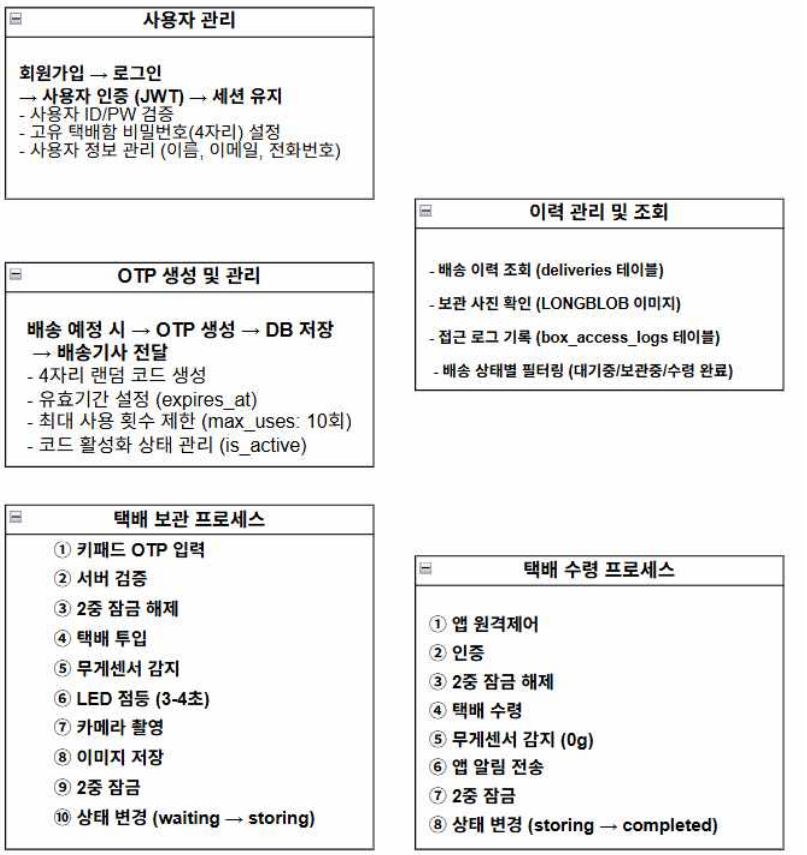
3) 서보모터 (각도 제어 장치)
- 라즈베리파이의 PWM 핀으로 제어된다.
- 제어선(노란색/주황색)은 GPIO로, 전원선은 5V에 연결되어 있다.
- 문 열림·닫힘, 와이퍼 동작 등에 활용 가능.

4) 스텝모터 + 스텝모터 드라이버
- 스텝모터는 직접 라즈베리파이에 연결되지 않고,
- 스텝모터 드라이버(A4988 등) 를 통해 구동된다.
- 드라이버는 라즈베리파이로부터 STEP, DIR 제어 신호를 받는다.
- 스텝모터 전원은 별도의 고전압(예: 12V) 이 필요하므로
- 전원 공급 장치에서 직접 공급받는다.

5) 브레드보드 (배선 분배 및 전원 분기)
- 5V, GND 라인을 여러 센서·모터로 안전하게 분배하는 역할.
- 라즈베리파이–센서–모터 간 연결을 안전하게 정리하는 구조.

5.4 기능블록도

5.4.1 전체 시스템 기능 블록도



1) 사용자 관리
회원가입/로그인: 사용자가 로그인하고 세션을 유지하는 기능 (JWT 사용).
사용자 정보 관리: 이메일, 전화번호 등의 사용자 정보를 관리.

2) OTP 생성 및 관리
배달 예시: 택배 수령을 위한 OTP 생성, DB에 저장 및 유효 기간 관리.
4자리 OTP 생성: 사용자와 배송 기사 간 인증을 위한 일회용 비밀번호 생성.

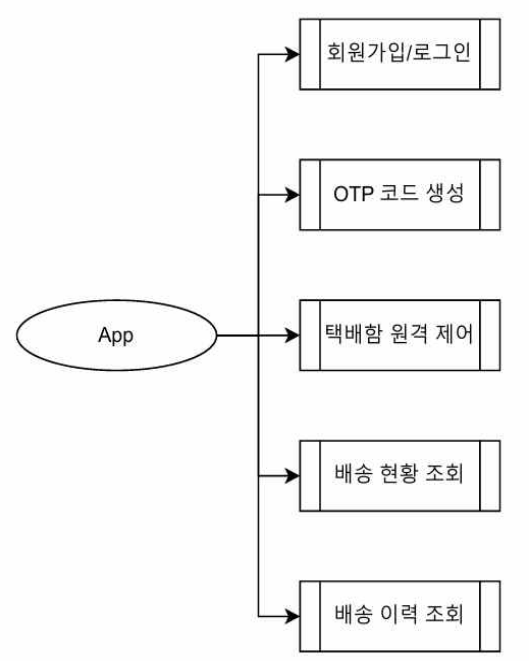
3) 이력 관리 및 조회
배달 이력 조회: 배달 상태 및 사진을 DB에서 확인하고 조회하는 기능.
박스 접근 기록: 사용자의 택배함 접근 기록을 관리하고 조회할 수 있다.

4) 택배 보관 프로세스
기본 OTP 입력: 택배 수령을 위한 OTP 입력 및 인증 과정.
무게 감지: 택배가 들어가면 무게 센서로 상태가 "대기 중"에서 "보관 중"으로 변경.
이미지 촬영: 택배가 보관될 때 자동으로 사진을 찍어 저장.

5) 택배 수령 프로세스
택배 개봉: 사용자 OTP로 택배함을 열고 택배를 수령하는 과정.
배송 상태 변경: 택배 상태를 "완료"로 업데이트하여 수령이 완료된 상태를 기록함.

5.4.2 안드로이드 앱 기능 블록도 및 순서도

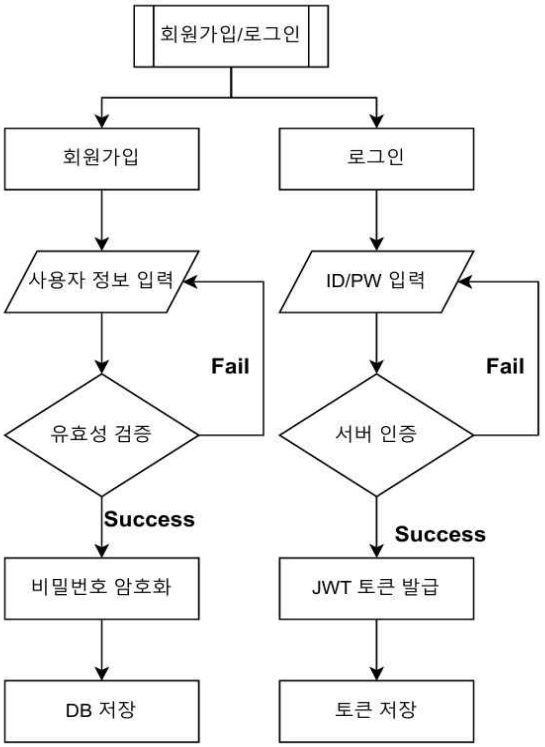
1) 안드로이드 앱 주요 기능 구성



안드로이드 앱은 스마트 택배함 시스템의 사용자 인터페이스를 담당하며, 다음과 같은 5가지 주요 기능으로 구성된다.

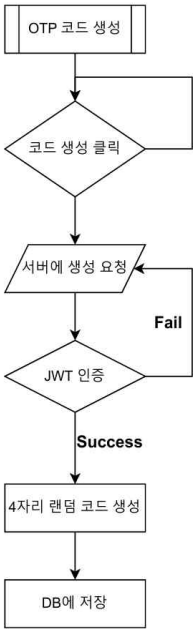
- 회원가입/로그인: 사용자 인증 및 계정 관리를 통해 시스템 접근 권한을 부여한다.
- OTP 코드 생성: 택배함 개폐를 위한 일회용 4자리 인증 코드를 생성한다.
- 택배함 원격 제어: OTP 또는 비밀번호를 이용하여 원격으로 택배함을 열거나 닫는다.
- 배송 현황 조회: 현재 진행 중인 배송의 상태(대기 중, 보관 중, 수령 완료)를 실시간으로 확인한다.
- 배송 이력 조회: 과거 배송 내역을 조회하고 통계(전체/완료/배송 중)를 확인한다.

2) 회원가입/로그인 기능



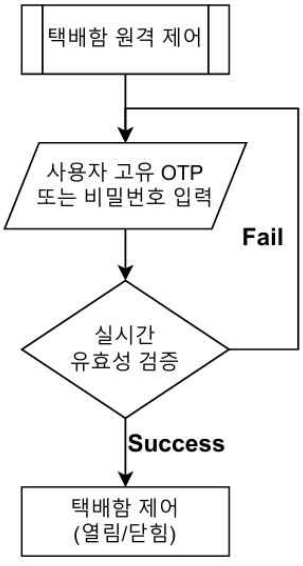
① 사용자가 앱에 접속하면 회원가입 또는 로그인을 선택한다.
② 회원가입을 선택하면 사용자 정보를 입력한다.
③ 입력된 정보의 유효성을 검증한다.
④ 유효성 검증에 성공하면 비밀번호를 암호화하여 DB에 저장한다.
⑤ 로그인을 선택하면 ID/PW를 입력한다.
⑥ 입력된 ID/PW를 서버에서 인증한다.
⑦ 서버 인증에 성공하면 JWT 토큰을 발급하여 로컬에 저장한다.
⑧ 인증 또는 검증에 실패하면 해당 단계로 돌아가 재입력을 요구한다.

3) OTP 코드 생성 기능



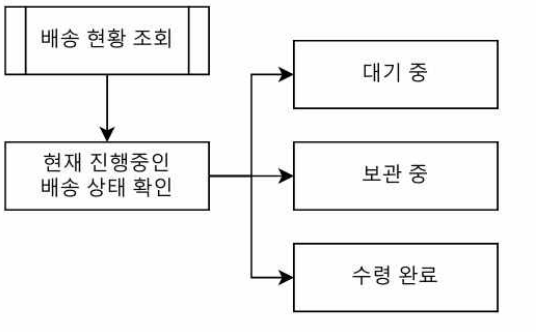
① 사용자가 코드 생성 버튼을 클릭한다.
② 서버에 OTP 생성 요청을 전송한다.
③ JWT 토큰을 이용하여 사용자 인증을 수행한다.
④ 인증에 성공하면 4자리 랜덤 코드를 생성한다.
⑤ 생성된 코드를 DB에 저장한다.
⑥ 인증에 실패하면 코드 생성 단계로 돌아간다.

4) 택배함 원격 제어 기능



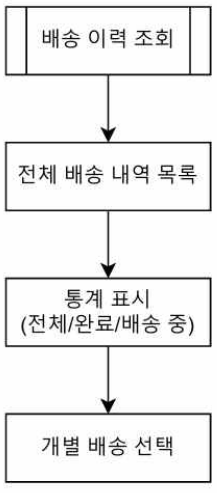
① 사용자가 고유 OTP 또는 비밀번호를 입력한다.
② 입력된 정보의 실시간 유효성 검증을 수행한다.
③ 유효성 검증에 성공하면 서버에서 인증을 진행한다.
④ 서버 인증에 성공하면 택배함 제어 신호를 전송하여 택배함을 열거나 닫는다.
⑤ 검증 또는 인증에 실패하면 입력 단계로 돌아간다.

5) 배송 현황 조회 기능



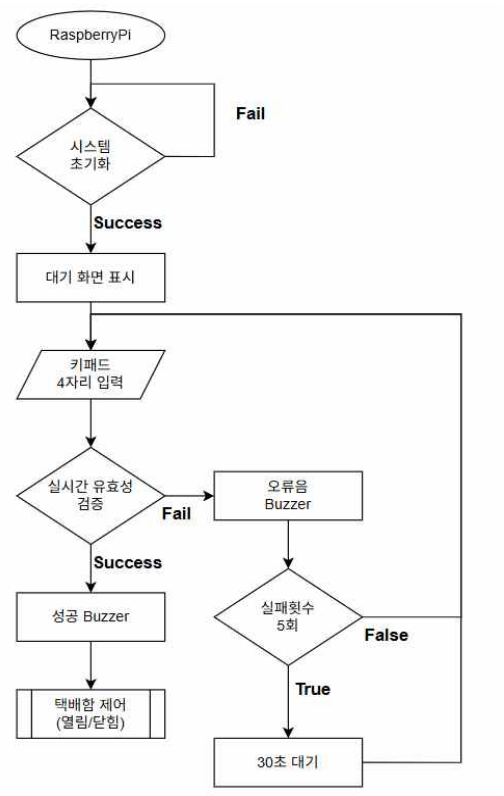
① 사용자가 배송 현황 조회 메뉴를 선택한다.
② 현재 진행 중인 배송 상태를 확인한다.
③ 배송 상태는 대기 중, 보관 중, 수령 완료로 구분되어 표시된다.

6) 배송 이력 조회 기능



① 사용자가 배송 이력 조회 메뉴를 선택한다.
② 전체 배송 내역 목록을 불러온다.
③ 통계 정보(전체/완료/배송 중)를 표시한다.
④ 사용자가 개별 배송을 선택하여 상세 내역을 확인한다.

5.4.3 라즈베리파이 하드웨어 시스템 동작 순서도



1) 시스템 시작 및 초기화

① 라즈베리파이가 부팅되면 시스템 초기화를 시도한다.
② 시스템 초기화에 성공하면 대기 화면을 표시하고, 실패하면 초기화 과정을 반복한다.
③ 초기화가 완료되면 키패드 입력 대기 상태로 전환한다.

2) 키패드 입력 및 인증 처리

① 대기 화면에서 사용자가 키패드를 통해 4자리 코드를 입력한다.
② 입력된 4자리 코드에 대한 실시간 유효성 검증을 수행한다.
③ 유효성 검증에 성공하면 성공 Buzzer 음을 출력하고 택배함 제어 단계로 진행한다.
④ 유효성 검증에 실패하면 오류 Buzzer 음을 출력한다.

3) 실패 횟수 제어 메커니즘

① 검증 실패 시 실패 횟수를 확인한다.
② 실패 횟수가 5회 미만(False)이면 즉시 키패드 입력 대기 화면으로 돌아가 재입력을 허용한다.
③ 실패 횟수가 5회 이상(True)이면 30초 대기 시간을 부여한다.
④ 30초 대기 후 실패 횟수를 초기화하고 키패드 입력 대기 화면으로 돌아간다.

4) 택배함 제어 동작

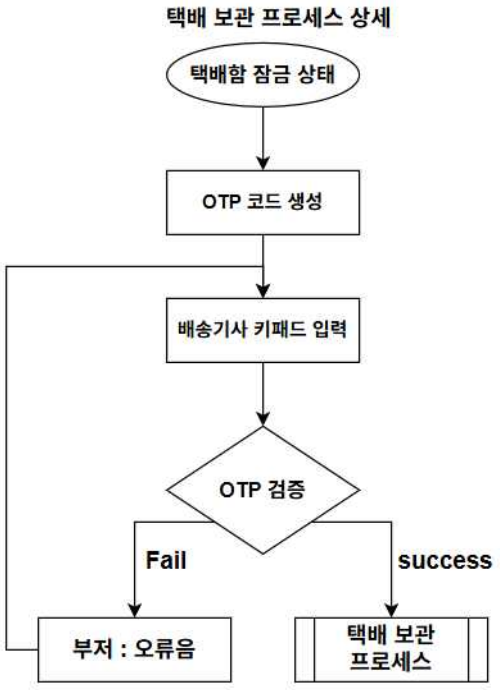
[택배함 열림 프로세스]

① 인증에 성공하면 택배함 제어(열림/닫힘) 명령을 실행한다.
② 택배함 열림 명령이 전달되면 스테핑 모터를 90도 회전시켜 택배함을 연다.
③ 택배함이 열린 후 다시 키패드 입력 대기 상태로 돌아간다.

[택배함 닫힘 프로세스]

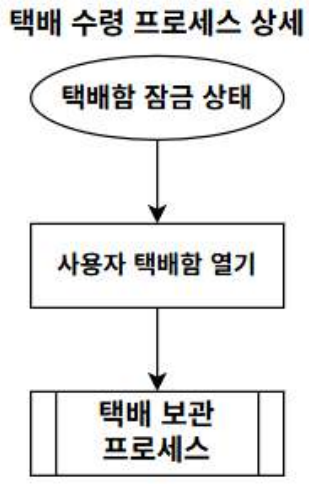
① 택배함 제어(열림/닫힘) 명령에서 닫힘 신호가 전달된다.
② 택배함 닫힘 명령이 실행되면 스테핑 모터를 -90도 회전시켜 택배함을 닫는다.
③ 택배함이 닫힌 후 다시 키패드 입력 대기 상태로 돌아간다.

5.4.4 택배 보관 프로세스 상세 프로세스



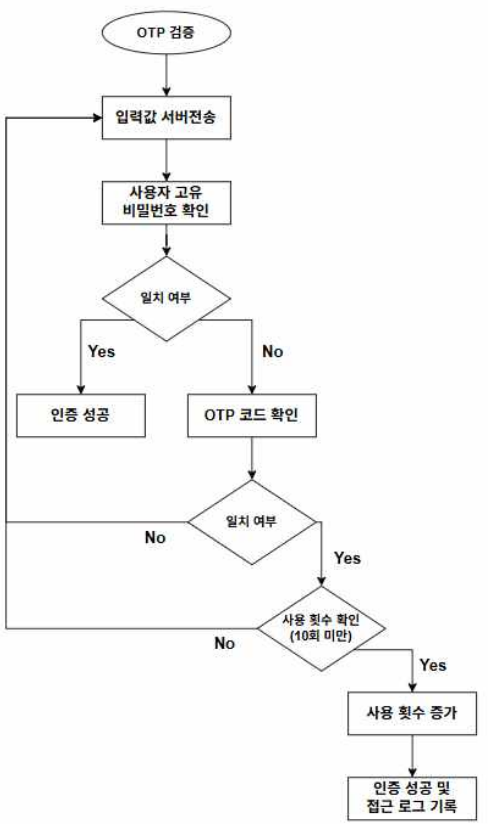
① 택배함 잠금 상태 확인: 택배함이 잠겨 있는지 확인한다.
② OTP 코드 생성: 사용자에게 전달할 OTP 코드를 생성한다.
③ 배송기사 키패드 입력: 배송 기사가 키패드에 OTP 코드를 입력합한다.
④ OTP 검증:
- OTP 검증: 입력된 OTP 코드가 정확한지 확인.
- Fail: OTP 코드가 틀리면 오류 처리.
- Success: OTP 코드가 맞으면 택배 보관 프로세스로 이동하여 택배를 보관.

5.4.5 택배 수령 프로세스 상세 프로세스



① 택배함 잠금 상태 확인: 택배함이 잠겨 있는 상태인지 확인한다.
② 사용자 택배함 열기: 사용자가 택배함을 열어 택배를 수령한다.
③ 택배 보관 프로세스: 택배가 보관되는 과정이 진행된다. 이는 택배 수령 후 택배 보관을 위한 상태로 전환된다.

5.4.6 사용자 인증 프로세스



① OTP 검증: 입력된 OTP 코드가 서버로 전송되고, 그 결과를 통해 인증 여부를 확인한다.
② 사용자 고유 비밀번호 확인: OTP 입력 후, 사용자 비밀번호가 맞는지 확인한다.
③ 입력 여부:
- Yes: 인증 성공, 다음 단계로 진행.
- No: OTP 코드 확인 단계로 돌아가며, 올바른 OTP 코드 입력을 요구.

④ 사용 횟수 확인 (10회 미만): 사용자가 인증 시도 횟수 10회 미만인지 확인.
- Yes: 인증 성공 및 로그인 기록이 추가.
- No: 인증 실패 횟수가 너무 많으면 시스템이 인증 실패 처리를 진행.

5.5 동작 방법

5.5.1 Android 앱 사용자 인터페이스

1) 로그인 화면 (LoginActivity)

사용자 인증을 위한 화면으로, 아이디와 비밀번호를 입력받아 서버에 로그인 요청을 전송한다.

- **화면 구성**
  - 사용자 ID 입력: 영어+숫자 조합 (영어만 가능, 숫자만 불가)
  - 비밀번호 입력: 최소 6자리 이상
  - 로그인 버튼: 입력 검증 후 서버 요청
  - 회원가입 이동 버튼: RegisterActivity로 이동

2) 회원가입 화면 (RegisterActivity)

신규 사용자 계정을 생성하는 화면이다.

- **화면 구성**
  - 사용자 ID: 영어+숫자 조합 (중복 확인)
  - 비밀번호: 최소 6자리
  - 이름: 실명
  - 이메일: 선택사항
  - 전화번호: 선택사항
  - 개인 택배함 비밀번호: 4자리 숫자 (사용자 전용)

3) 메인 화면 (MainActivity)

로그인 후 표시되는 대시보드 화면으로, 모든 주요 기능에 접근할 수 있다.

- **화면 구성**
  - 상단: "안녕하세요, [사용자명]님" 인사말
  - OTP 생성: GenerateCodeActivity로 이동
  - 택배함 원격제어: OpenLockerActivity로 이동
  - 배송 이력: HistoryActivity로 이동
  - 설정 카드: SettingsActivity로 이동

4) OTP 생성 화면 (GenerateCodeActivity)

배송기사에게 전달할 일회용 비밀번호를 생성하는 화면이다.

- **화면 구성**
  - OTP 생성 버튼: 4자리 랜덤 코드 생성
  - 생성된 코드 표시: 큰 글씨로 코드 표시
  - 클립보드 복사 버튼: 코드를 클립보드에 복사
  - 뒤로가기 버튼: 메인 화면으로 복귀

5) 택배함 원격제어 화면 (OpenLockerActivity)

사용자가 앱에서 직접 택배함을 열고 닫을 수 있는 화면이다.

- **화면 구성**
  - 개인 비밀번호 입력: 회원가입 시 설정한 4자리
  - OTP 입력: 앱에서 생성한 임시 코드
  - 택배함 열기 버튼: MQTT로 열기 명령 전송
  - 택배함 닫기 버튼: MQTT로 닫기 명령 전송
  - 현재 상태 표시: 열림/닫힘 상태

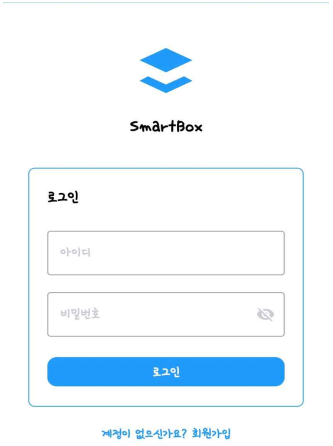
6) 배송 이력 화면 (HistoryActivity)

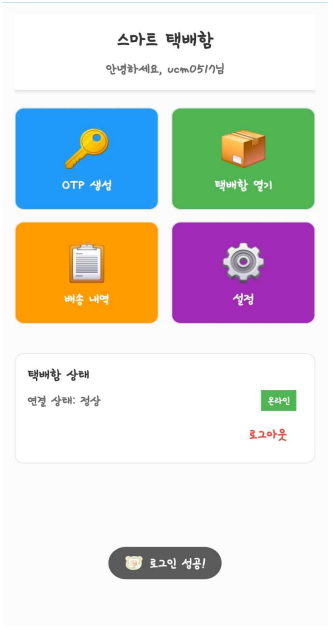
과거 배송 기록을 조회하고 보관 사진을 확인할 수 있는 화면이다.

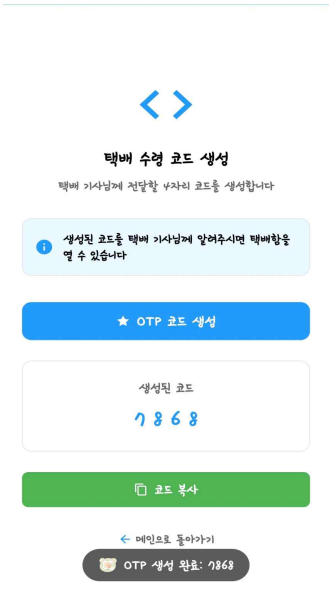
- 화면 구성

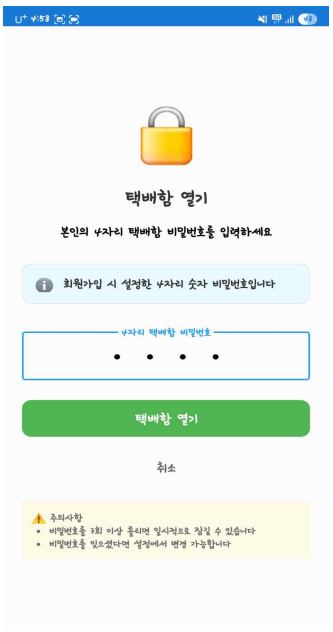
- RecyclerView 리스트: 배송 이력 목록
  - 배송 날짜
  - 상태 (대기중/보관중/수령완료)
  - 택배 무게
  - 보관 사진 썸네일
- 필터 버튼: 상태별 필터링
- 상세보기: 각 항목 클릭 시 상세 화면

- 구현된 앱 인터페이스 이미지





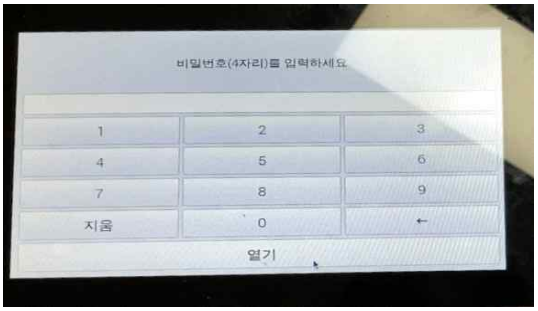




5.5.2 Raspberry Pi 제어 시스템

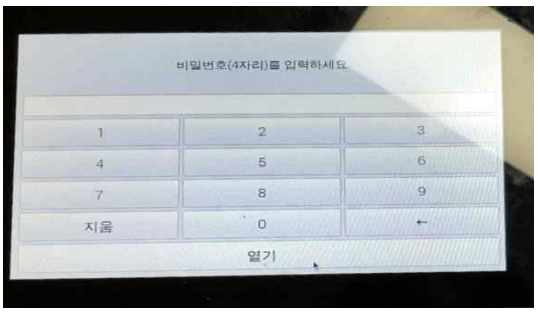
1) 키패드 입력 화면 (PyQt5 GUI)

- 화면 구성

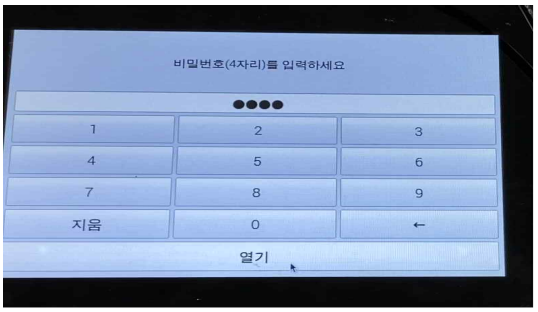


- 동작 상태

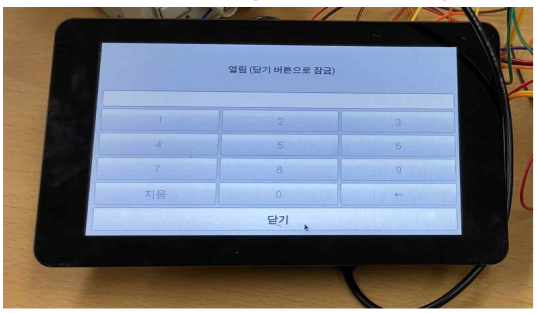
• 상태 1: 초기 대기 상태 (잠금)



• 상태 2: OTP 입력 완료

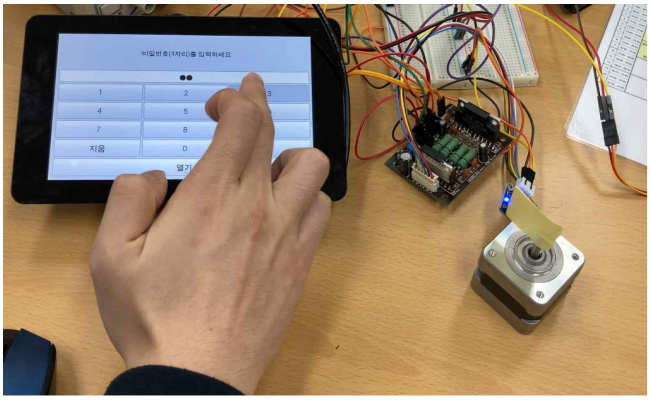


• 상태 3: 문 열림 (택배 투입 대기)

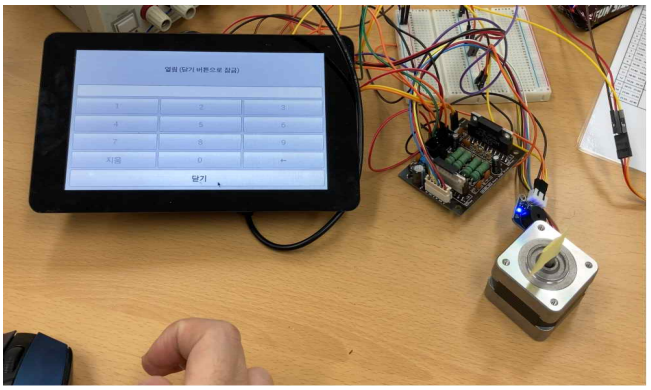


2) 택배 보관 프로세스 (배송기사)

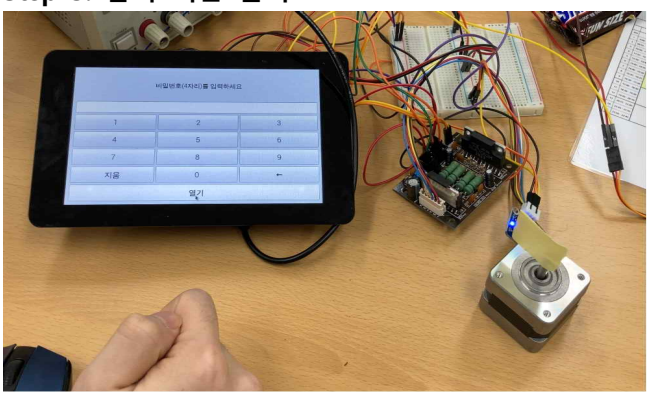
Step 1: 문 열기



Step 2: 택배 투입



Step 3: 닫기 버튼 클릭



Step 4: 보관 중 상태 UI

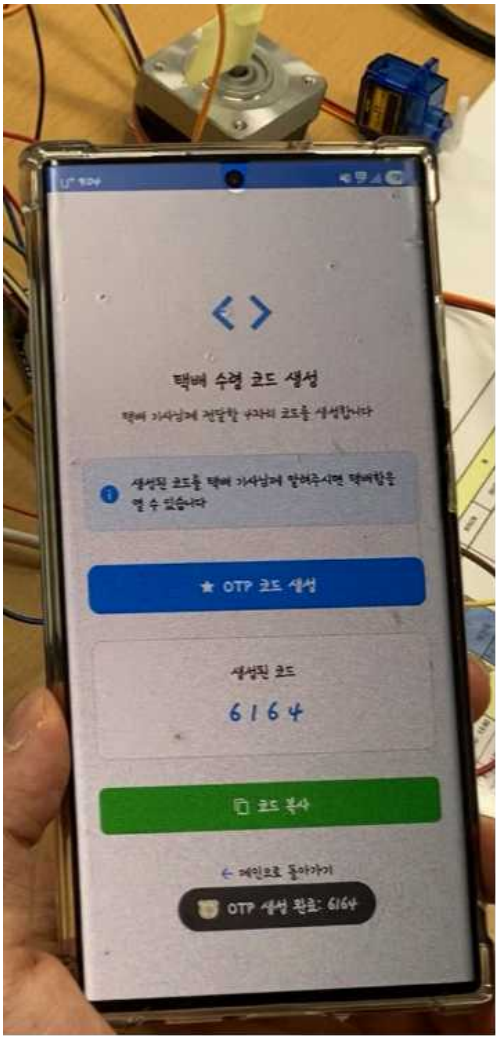
-디스플레이 보관함 속 택배의 유무를 표현하여 빠르게 사용자가 확인할 수 있도록 구현 예정

3) 택배 수령 프로세스 (사용자)

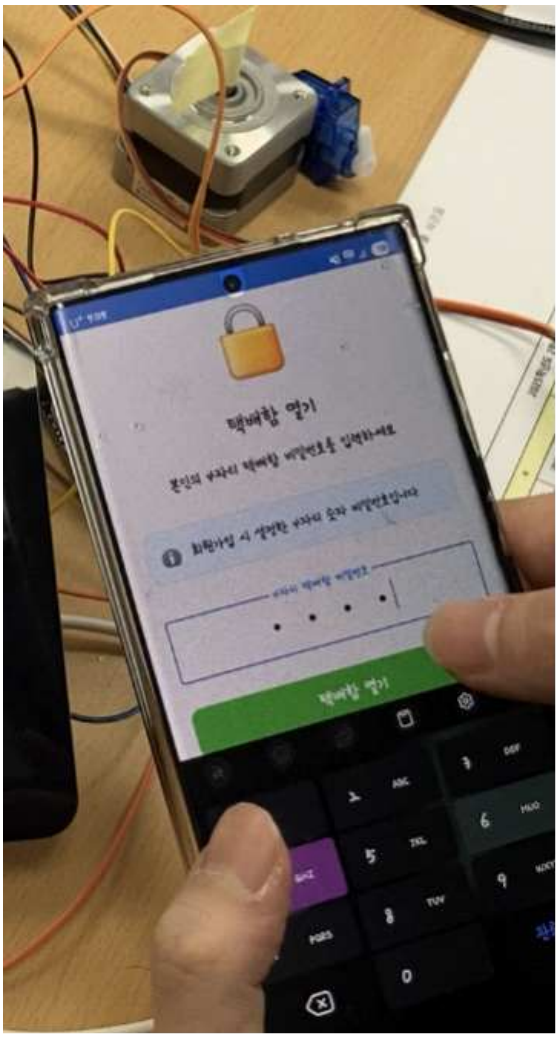
앱 원격 제어 (MQTT)

앱으로 사용자 비밀번호 또는 OTP로 입력받은 앱 화면과 성공 시 모터 회전

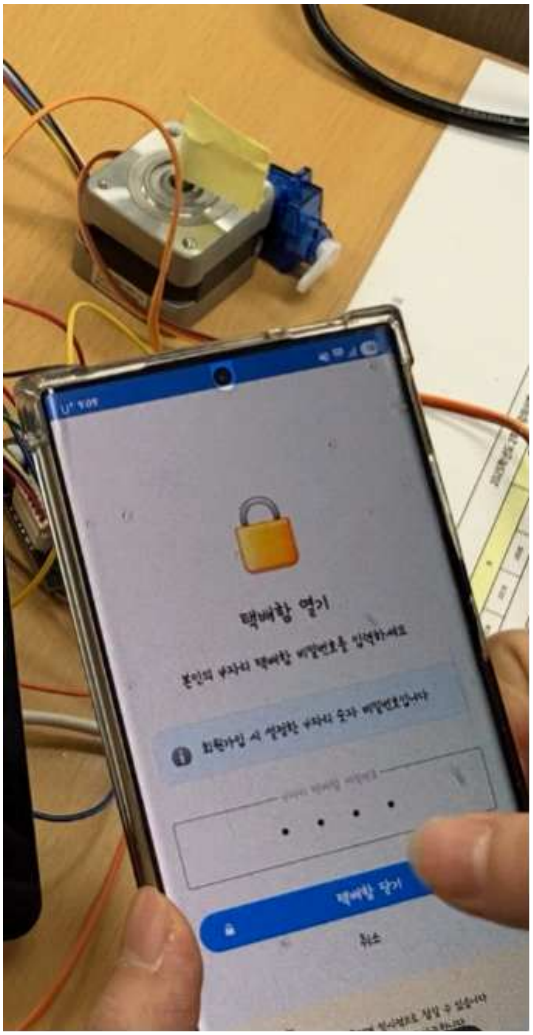
i) OTP 생성



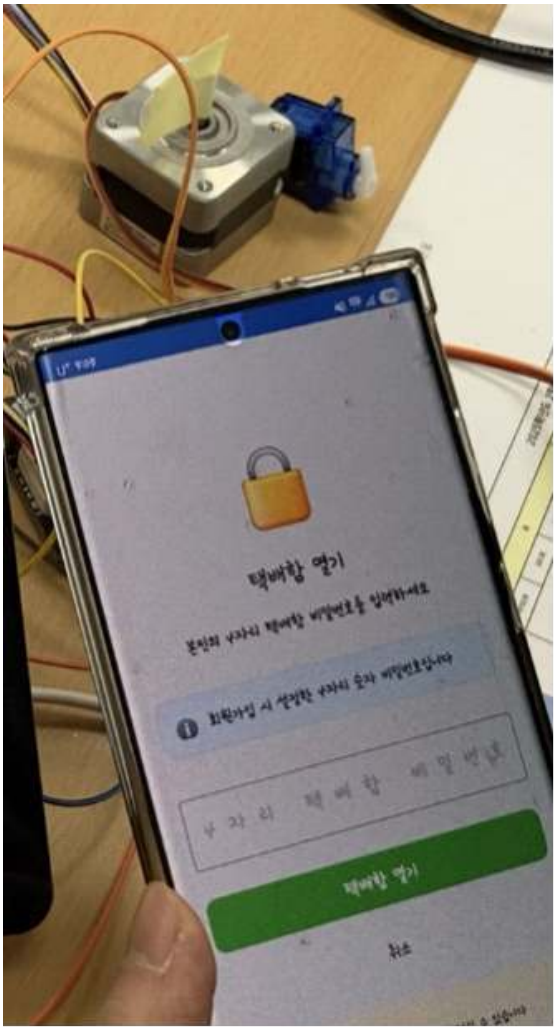
ii) 앱에서 OTP 입력



iii) OTP 일치할 시, 택배함 열림



iiii) 택배함 닫기 버튼으로 택배함 닫기

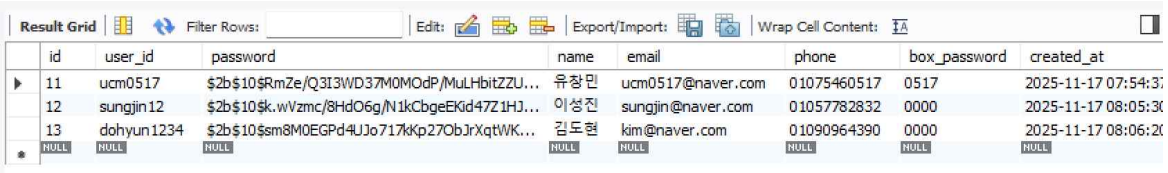


5.6 프로젝트 구현

5.6.1 데이터베이스 설계 및 구축

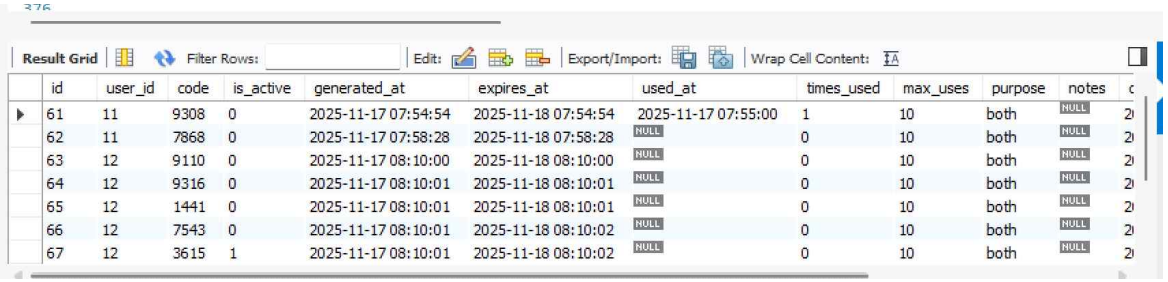
본 시스템은 MySQL 8.0을 사용하여 사용자 정보, 배송 데이터, 접근 로그를 체계적으로 관리한다. AWS EC2 Ubuntu 서버에 MySQL을 설치하고, 다음과 같은 5개의 핵심 테이블을 설계하였다.

(1) users: 사용자 계정 정보 및 개인 택배함 비밀번호 저장



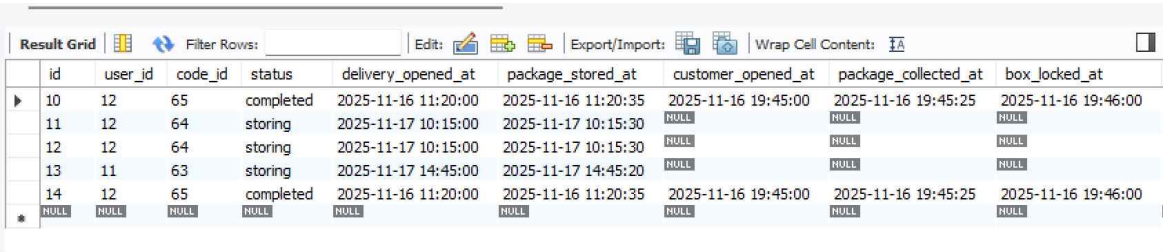
id	user_id	password	name	email	phone	box_password	created_at
11	ucm0517	$2b$10$RmZe/Q3I3WD37M0MOdP/MuLHbitZZU...	유창민	ucm0517@naver.com	01075460517	0517	2025-11-17 07:54:3
12	sungjin12	$2b$10$k.wVzmc/8HdO6g/N1kCbgeEKid47Z1HJ...	이성진	sungjin@naver.com	01057782832	0000	2025-11-17 08:05:3
13	dohyun1234	$2b$10$sm8M0EGPd4UJo717kKp27ObJrXqtWK...	김도현	kim@naver.com	01090964390	0000	2025-11-17 08:06:2

(2) delivery_codes: OTP 코드 생성 및 유효기간 관리

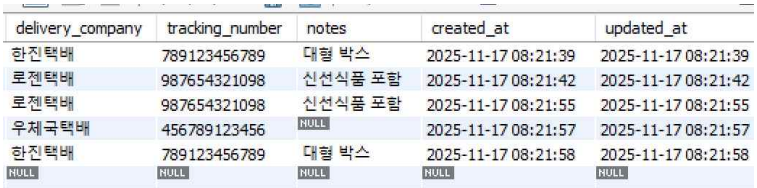


id	user_id	code	is_active	generated_at	expires_at	used_at	times_used	max_uses	purpose	notes
61	11	9308	0	2025-11-17 07:54:54	2025-11-18 07:54:54	2025-11-17 07:55:00	1	10	both	
62	11	7868	0	2025-11-17 07:58:28	2025-11-18 07:58:28		0	10	both	
63	12	9110	0	2025-11-17 08:10:00	2025-11-18 08:10:00		0	10	both	
64	12	9316	0	2025-11-17 08:10:01	2025-11-18 08:10:01		0	10	both	
65	12	1441	0	2025-11-17 08:10:01	2025-11-18 08:10:01		0	10	both	
66	12	7543	0	2025-11-17 08:10:01	2025-11-18 08:10:02		0	10	both	
67	12	3615	1	2025-11-17 08:10:01	2025-11-18 08:10:02		0	10	both	

(3) deliveries: 배송 상태, 무게 데이터, 이미지 저장

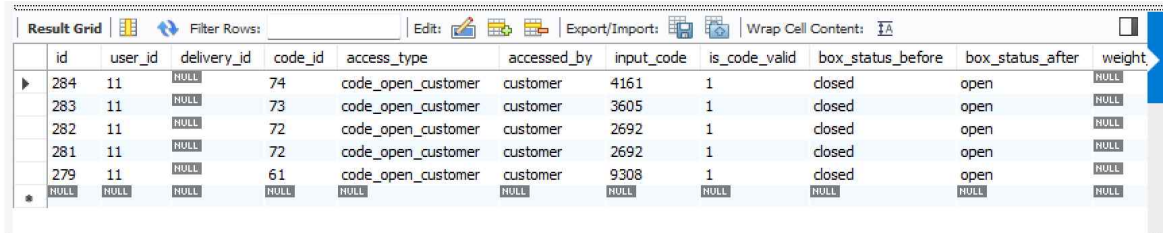


id	user_id	code_id	status	delivery_opened_at	package_stored_at	customer_opened_at	package_collected_at	box_locked_at
10	12	65	completed	2025-11-16 11:20:00	2025-11-16 11:20:35	2025-11-16 19:45:00	2025-11-16 19:45:25	2025-11-16 19:46:00
11	12	64	storing	2025-11-17 10:15:00	2025-11-17 10:15:30			
12	12	64	storing	2025-11-17 10:15:00	2025-11-17 10:15:30			
13	11	63	storing	2025-11-17 14:45:00	2025-11-17 14:45:20			
14	12	65	completed	2025-11-16 11:20:00	2025-11-16 11:20:35	2025-11-16 19:45:00	2025-11-16 19:45:25	2025-11-16 19:46:00



delivery_company	tracking_number	notes	created_at	updated_at
한진택배	789123456789	대형 박스	2025-11-17 08:21:39	2025-11-17 08:21:39
로젠택배	987654321098	신선식품 포함	2025-11-17 08:21:42	2025-11-17 08:21:42
로젠택배	987654321098	신선식품 포함	2025-11-17 08:21:55	2025-11-17 08:21:55
우체국택배	456789123456		2025-11-17 08:21:57	2025-11-17 08:21:57
한진택배	789123456789	대형 박스	2025-11-17 08:21:58	2025-11-17 08:21:58

(4) box_access_logs: 모든 접근 시도 및 성공/실패 기록



id	user_id	delivery_id	code_id	access_type	accessed_by	input_code	is_code_valid	box_status_before	box_status_after	weight
284	11		74	code_open_customer	customer	4161	1	closed	open	
283	11		73	code_open_customer	customer	3605	1	closed	open	
282	11		72	code_open_customer	customer	2692	1	closed	open	
281	11		72	code_open_customer	customer	2692	1	closed	open	
279	11		61	code_open_customer	customer	9308	1	closed	open	

(5) active_delivery_codes (VIEW): 활성 코드 조회 최적화

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	user_id	username	name	code	generated_at	expires_at	times_used	max_uses	hours_remaining	status
74	11	ucm0517	유창민	4161	2025-11-17 08:13:40	2025-11-18 08:13:40	1	10	23	active
69	13	dohyun1234	김도현	7199	2025-11-17 08:10:23	2025-11-18 08:10:24	0	10	23	active
67	12	sungjin12	이성진	3615	2025-11-17 08:10:01	2025-11-18 08:10:02	0	10	23	active

5.6.2 MQTT 기반 실시간 통신 시스템 구축

본 시스템은 Android 앱과 Raspberry Pi 간의 실시간 양방향 통신을 위해 MQTT(Message Queuing Telemetry Transport) 프로토콜을 도입하였다. MQTT는 IoT 환경에 최적화된 경량 메시징 프로토콜로, HTTP 폴링 방식 대비 낮은 대역폭과 즉각적인 응답성을 제공한다.

5.6.2.1 서버 측 MQTT 브로커 구축 (Node.js)

```
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
sudo systemctl start mosquitto
sudo systemctl enable mosquitto

sudo netstat -tuln | grep 1883
```

```
ubuntu@ip-172-31-0-68:~$ sudo netstat -tuln | grep 1883
tcp        0      0 0.0.0.0:1883            0.0.0.0:*               LISTEN
tcp6       0      0 :::1883                 :::*                    LISTEN
```

5.6.2.2 Node.js 서버와 MQTT 통합

```
// MQTT 클라이언트 연결
const MQTT_BROKER = process.env.MQTT_BROKER || 'mqtt://localhost:1883';
const mqttClient = mqtt.connect(MQTT_BROKER);

mqttClient.on('connect', () => {
  console.log('MQTT 브로커 연결 성공');
});

mqttClient.on('error', (err) => {
  console.error('MQTT 연결 오류:', err);
});
```

```
//성공 시 MQTT 발행
if (ok) {
  const mqttMessage = JSON.stringify({
    action: 'open',
    userId: userId,
    message: message,
    timestamp: Date.now()
  });

  mqttClient.publish('smartbox/locker/command', mqttMessage, { qos: 1 }, (err) => {
    if (err) {
      console.error(`[verify-open] MQTT 발행 실패:`, err);
    } else {
      console.log(`[verify-open] MQTT 명령 전송 - userId: ${userId}, action: open`);
    }
  });
}

res.json({ ok, message, user_id: userId, delivery_id: deliveryId, code_id: codeId, device_id: deviceId });
} catch (err) {
  await conn.rollback();
  console.error('[verify-open] error:', err);
  res.status(500).json({ ok: false, message: 'server_error' });
} finally {
  conn.release();
```

5.6.2.3 Raspberry Pi MQTT 클라이언트 구현

(1) Paho MQTT 라이브러리 설치

pip3 install paho-mqtt —break-system-packages

(2) MQTT 구독 및 메시지 처리

```python
# MQTT 시그널 브릿지 - 스레드 간 안전한 통신
class MQTTSignalBridge(QObject):
    open_signal = pyqtSignal()
    close_signal = pyqtSignal()

mqtt_bridge = MQTTSignalBridge()
keypad_window_instance = None

def on_mqtt_connect(client, userdata, flags, rc):
    if rc == 0:
        print(" MQTT 브로커 연결 성공")
        result, mid = client.subscribe(MQTT_TOPIC, qos=1)
        print(f"토픽 구독 시도: {MQTT_TOPIC} (result={result}, mid={mid})")
    else:
        print(f"MQTT 연결 실패: rc={rc}")

def on_mqtt_subscribe(client, userdata, mid, granted_qos):
    print(f" 구독 확인 완료 - mid={mid}, qos={granted_qos}")
```

```python
def on_mqtt_message(client, userdata, msg):
    try:
        payload = msg.payload.decode()
        print(f"\n [MQTT] 메시지 수신!")
        print(f"   토픽: {msg.topic}")
        print(f"   내용: {payload}")

        data = json.loads(payload)
        action = data.get('action')
        user_id = data.get('userId')

        print(f"   → Action: {action}, UserId: {user_id}")

        #  시그널을 통해 메인 스레드로 전달
        if action == 'open':
            print("   open_signal 발생")
            mqtt_bridge.open_signal.emit()
        elif action == 'close':
            print("   close_signal 발생")
            mqtt_bridge.close_signal.emit()

    except json.JSONDecodeError as e:
        print(f"[MQTT] JSON 파싱 오류: {e}")
    except Exception as e:
        print(f"[MQTT] 메시지 처리 오류: {e}")
        traceback.print_exc()
```

```python
# MQTT 클라이언트 초기화
mqtt_client = None
if MQTT_AVAILABLE:
    print(f"\n[MQTT] 클라이언트 초기화 중...")
    mqtt_client = mqtt.Client()
    mqtt_client.on_connect = on_mqtt_connect
    mqtt_client.on_subscribe = on_mqtt_subscribe
    mqtt_client.on_message = on_mqtt_message

    try:
        print(f"[MQTT] 브로커 연결 시도: {MQTT_BROKER}:{MQTT_PORT}")
        mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)
        mqtt_client.loop_start()
        print(f"[MQTT] 루프 시작 완료\n")
    except Exception as e:
        print(f"[MQTT] 연결 오류: {e}")
        traceback.print_exc()
        mqtt_client = None
else:
    print("\n[경고] paho-mqtt 라이브러리 없음\n")
```

5.6.2.4 PyQt5 GUI와 MQTT 시그널 연결

```python
# ====== 키패드 UI ======
class KeypadWindow(QWidget):
    def __init__(self):
        global keypad_window_instance

        # MQTT 시그널 연결 - 메인 스레드에서 실행됨
        mqtt_bridge.open_signal.connect(self._handle_mqtt_open)
        mqtt_bridge.close_signal.connect(self._handle_mqtt_close)
        print(" MQTT 시그널 연결 완료")

        self._build_ui()
        self.showFullScreen()
        QTimer.singleShot(100, self._initial_ping)
```

```python
    # MQTT 시그널 핸들러 - 메인 스레드에서 실행됨
    def _handle_mqtt_open(self):
        """MQTT open 시그널 수신 시 호출 (메인 스레드)"""
        print("[KeypadWindow] _handle_mqtt_open 호출됨(메인 스레드)")

        # 로컬에서 실행 중이면 무시 (중복 실행 방지)
        if self.local_action_flag:
            print("[KeypadWindow] 로컬 실행 중 - MQTT 명령 무시")
            return

        try:
            self._msg("앱에서 열기 요청")
            self.locker.open()
            self.enter_btn.setText("닫기")
            for b in self.buttons:
                b.setEnabled(False)
            self.input.clear()
            self._msg("열림 (앱) - 닫기는 버튼 또는 앱")
            print("[KeypadWindow] 열기 완료")
        except Exception as e:
            print(f"[KeypadWindow] 열기 오류: {e}")
            traceback.print_exc()
```

```python
    def _handle_mqtt_close(self):
        """MQTT close 시그널 수신 시 호출 (메인 스레드)"""
        print("[KeypadWindow] _handle_mqtt_close 호출됨! (메인 스레드)")

        # 로컬에서 실행 중이면 무시 (중복 실행 방지)
        if self.local_action_flag:
            print("[KeypadWindow] 로컬 실행 중 - MQTT 명령 무시")
            return

        try:
            self.locker.close()
            self.enter_btn.setText("열기")
            for b in self.buttons:
                b.setEnabled(True)
            self.input.clear()
            self._msg("비밀번호(4자리)를 입력하세요")
            print("[KeypadWindow] 닫기 완료")
        except Exception as e:
            print(f"[KeypadWindow] 닫기 오류: {e}")
            traceback.print_exc()
```

5.7 프로젝트 핵심 기능 구현

5.7.1 2중 OTP 인증 시스템 구현

본 시스템은 보안성을 극대화하기 위해 2가지 인증 방식을 제공한다.

(1) 앱에서 생성하는 임시 OTP 코드

(2) 회원가입 시 설정한 사용자 고유 비밀번호.

- **2중 인증:** 개인 비밀번호 우선 확인 → OTP 확인
- **유효기간 관리:** 24시간 자동 만료 (expires_at)
- **사용 횟수 제한:** 최대 10회 사용 후 무효화
- **접근 로그 기록:** 모든 시도 box_access_logs에 저장

- **Step 1: OTP 생성 (Android → Node.js)**

```javascript
// OTP 코드 생성 API
// POST /api/delivery/generate-otp
app.post('/api/delivery/generate-otp', authenticateToken, async (req, res) => {
  const userId = req.userId;
  const { purpose = 'both', maxUses = 10 } = req.body;

  const conn = await pool.getConnection();
  try {
    await conn.beginTransaction();

    // 기존 활성 코드 비활성화
    await conn.query(
      'UPDATE delivery_codes SET is_active = FALSE WHERE user_id = ? AND is_active = TRUE',
      [userId]
    );

    // 새 OTP 생성 (4자리 숫자)
    const code = Math.floor(1000 + Math.random() * 9000).toString();
    const expiresAt = new Date(Date.now() + 24 * 60 * 60 * 1000); // 24시간 후

    const [result] = await conn.query(
      `INSERT INTO delivery_codes (user_id, code, is_active, generated_at, expires_at, times_used, max_uses, purpos
       VALUES (?, ?, TRUE, NOW(), ?, 0, ?, ?)`,
      [userId, code, expiresAt, maxUses, purpose]
    );

    await conn.commit();
```

- **Step 2: 인증 검증 (Raspberry Pi → Node.js)**

```
// JWT 없이 OTP 코드 검증
// POST /api/locker/verify-open
app.post('/api/locker/verify-open', async (req, res) => {
  const code = (req.body && req.body.code || '').trim();
  const deviceId = (req.body && req.body.device_id || '').trim();
  const accessType = (req.body && req.body.access_type || 'code_open_customer').trim();

  console.log(`[verify-open] 라즈베리파이 요청 - 코드: ${code}`);

  if (!/^\d{4}$/.test(code)) {
    return res.status(400).json({ ok: false, message: '코드 형식 오류(4자리 숫자)' });
  }

  const conn = await pool.getConnection();
  try {
    await conn.beginTransaction();

    let userId = null;
    let deliveryId = null;
    let codeId = null;
    let ok = false;
    let message = 'invalid';

    // OTP 코드만 확인 (고유 PIN은 앱 전용)
    const [cRows] = await conn.query(
      `SELECT dc.id, dc.user_id, dc.times_used, dc.max_uses, d.id AS delivery_id
         FROM delivery_codes dc
         LEFT JOIN deliveries d ON d.code_id = dc.id AND d.status IN ('waiting','storing')
        WHERE dc.code = ? AND dc.is_active = TRUE AND dc.expires_at > NOW()
        ORDER BY dc.generated_at DESC
        LIMIT 1 FOR UPDATE`,
      [code]
    );
```

```
  //성공 시 MQTT 발행
  if (ok) {
    const mqttMessage = JSON.stringify({
      action: 'open',
      userId: userId,
      message: message,
      timestamp: Date.now()
    });

    mqttClient.publish('smartbox/locker/command', mqttMessage, { qos: 1 }, (err) => {
      if (err) {
        console.error(`[verify-open] MQTT 발행 실패:`, err);
      } else {
        console.log(`[verify-open] MQTT 명령 전송 - userId: ${userId}, action: open`);
      }
    });
  }

  res.json({ ok, message, user_id: userId, delivery_id: deliveryId, code_id: codeId, device_id: deviceId });
} catch (err) {
  await conn.rollback();
  console.error('[verify-open] error:', err);
  res.status(500).json({ ok: false, message: 'server_error' });
} finally {
  conn.release();
}
```

- **Step 3: 실패 처리 및 보안**

```
660     def _enter(self):
661         code = self.input.text()
662         if len(code) != 4:
663             self._msg("4자리를 입력하세요")
664             return
665 
666         ok, msg = (verify_code_rest(code) if VERIFY_MODE == "REST"
667                    else verify_code_local(code))
668 
669         if ok:
670             self._msg("인증 성공! 열립니다.")
671             self.input.clear()
672             self.attempts = 0
673             self._open_lock()
674         else:
675             self.attempts += 1
676             left = max(0, MAX_ATTEMPTS - self.attempts)
677             self._msg(f"실패: {msg} (남은 {left})")
678             self.input.clear()
679             self.locker.play_error_sound()
680             if self.attempts >= MAX_ATTEMPTS:
681                 self._start_cooldown()
```

5.7.2 2중 잠금 메커니즘 구현

택배함의 보안성을 강화하기 위해 서보모터(1차 잠금)와 스텝모터(2차 잠금)를 순차적으로 제어하는 2중 잠금 시스템을 구현하였다.

- **PWM 제어:** 서보모터 정밀 각도 조절 (50Hz)
- **스텝 제어:** 200스텝 모터의 90도 회전 계산
- **순차 제어:** 서보 → 스텝모터 순서로 해제/잠금
- **신호 안정화:** PWM 듀티 사이클 0으로 떨림 방지
- **부저 피드백:** 비동기 스레드로 사운드 재생

- 초기 환경 세팅

```
17  # ====== 설정 ======
18  VERIFY_MODE = "REST"
19  API_BASE    = "http://43.202.10.147:3001"
20  DEVICE_ID   = "locker-01"
21  VERIFY_URL = f"{API_BASE}/api/locker/verify-open"
22  HEALTH_URL = f"{API_BASE}/health"
23
24  # MQTT 설정
25  MQTT_BROKER = "43.202.10.147"
26  MQTT_PORT = 1883
27  MQTT_TOPIC = "smartbox/locker/command"
28
29  # LOCAL 모드용(옵션)
30  BOX_PIN_HASH = ""
31
32  # 동작 설정
33  AUTO_CLOSE_SECONDS = 8
34  MAX_ATTEMPTS       = 5
35  COOLDOWN_SECONDS   = 30
36
37  # GPIO 핀 설정
38  STEPPER_CLK_PIN = 18
39  STEPPER_DIR_PIN = 23
40  STEPPER_EN_PIN  = 24
41  BUZZER_PIN      = 21
42  SERVO_PIN       = 13  # 서보모터 핀 (1차 잠금장치)
43
44  # 스테핑 모터 파라미터
45  STEPS_PER_REV = 200
46  QUARTER_STEPS_PER_REV = STEPS_PER_REV * 2
47  STEP_DELAY = 0.002
```

(1) 서보모터 클래스

```python
# ====== 서보모터 (1차 잠금장치) ======
class ServoMotor:
    def __init__(self, pin):
        self.pin = pin
        self.pwm = None
        self.is_locked = True  # 초기 상태: 잠금

        if GPIO_AVAILABLE:
            try:
                GPIO.setup(self.pin, GPIO.OUT)
                self.pwm = GPIO.PWM(self.pin, 50)  # 50Hz (서보모터 표준)
                self.pwm.start(0)
                # 초기 위치: 잠금 상태 (0도)
                self._set_angle(0)
                time.sleep(0.5)
                self.pwm.ChangeDutyCycle(0) # 신호 끊기
                print(f"서보모터 PIN {self.pin} 초기화 완료 (1차 잠금)")
            except Exception as e:
                print(f"서보모터 초기화 실패: {e}")

    def _set_angle(self, angle):
        """서보모터 각도 설정 (0~180도)"""
        if not GPIO_AVAILABLE or not self.pwm:
            return
        try:
            # 각도를 듀티 사이클로 변환
            # 0도 = 2.5%, 90도 = 7.5%, 180도 = 12.5%
            duty = 2.5 + (angle / 180.0) * 10.0
            self.pwm.ChangeDutyCycle(duty)
            print(f"서보모터: {angle}도로 회전")
        except Exception as e:
            print(f"서보모터 각도 설정 오류: {e}")
```

```python
    def unlock(self):
        """1차 잠금 해제 (90도 회전)"""
        if self.is_locked:
            print("\n=== [1차 잠금] 서보모터 해제 ===")
            if not GPIO_AVAILABLE:
                print("시뮬레이션: 서보 90도 회전")
            else:
                self._set_angle(90)
                time.sleep(0.5)
                self.pwm.ChangeDutyCycle(0)
            self.is_locked = False
            print("=== [1차 잠금] 해제 완료 ===\n")
        else:
            print("[1차 잠금] 이미 해제됨")

    def lock(self):
        """1차 잠금 (0도로 복귀)"""
        if not self.is_locked:
            print("\n=== [1차 잠금] 서보모터 잠금 ===")
            if not GPIO_AVAILABLE:
                print("시뮬레이션: 서보 0도 복귀")
            else:
                self._set_angle(0)
                time.sleep(0.5)
                self.pwm.ChangeDutyCycle(0)
            self.is_locked = True
            print("=== [1차 잠금] 잠금 완료 ===\n")
        else:
            print("[1차 잠금] 이미 잠김")
```

(2) 스텝모터 클래스

```python
class StepperMotor:
    def __init__(self, clk_pin, dir_pin, en_pin):
        self.clk_pin = clk_pin
        self.dir_pin = dir_pin
        self.en_pin = en_pin
        self.is_locked = True

        if GPIO_AVAILABLE:
            try:
                GPIO.setup(self.clk_pin, GPIO.OUT)
                GPIO.setup(self.dir_pin, GPIO.OUT)
                GPIO.setup(self.en_pin, GPIO.OUT)
                GPIO.output(self.clk_pin, GPIO.LOW)
                GPIO.output(self.dir_pin, GPIO.LOW)
                GPIO.output(self.en_pin, GPIO.HIGH)
                print(f"스테핑 모터 초기화 완료")
            except Exception as e:
                print(f"스테핑 모터 초기화 실패: {e}")

    def step(self, steps, clockwise=True, delay=STEP_DELAY):
        if not GPIO_AVAILABLE:
            print(f"시뮬레이션: {'CW' if clockwise else 'CCW'} {steps} 스텝")
            return
        try:
            direction = GPIO.LOW if clockwise else GPIO.HIGH
            GPIO.output(self.dir_pin, direction)
            for i in range(steps):
                GPIO.output(self.clk_pin, GPIO.HIGH)
                time.sleep(delay)
                GPIO.output(self.clk_pin, GPIO.LOW)
                time.sleep(delay)
        except Exception as e:
            print(f"스테핑 모터 오류: {e}")
```

```python
    def open_lock(self):
        if self.is_locked:
            print("\n=== 잠금 해제 ===")
            steps_90 = QUARTER_STEPS_PER_REV // 4
            self.step(steps_90, clockwise=False)
            self.is_locked = False
            print("=== 잠금 해제 완료 ===\n")
        else:
            print("이미 열림")

    def close_lock(self):
        if not self.is_locked:
            print("\n=== 잠금 ===")
            steps_90 = QUARTER_STEPS_PER_REV // 4
            self.step(steps_90, clockwise=True)
            self.is_locked = True
            print("=== 잠금 완료 ===\n")
        else:
            print("이미 잠김")

    def cleanup(self):
        try:
            if GPIO_AVAILABLE:
                GPIO.output(self.en_pin, GPIO.LOW)
        except Exception as e:
            print(f"스테핑 모터 정리 오류: {e}")
```

```python
# ====== 자물쇠 (2중 잠금 시스템) ======
class Locker:
    def __init__(self):
        if GPIO_AVAILABLE:
            GPIO.setmode(GPIO.BCM)
        self.servo = ServoMotor(SERVO_PIN)          # 1차 잠금
        self.stepper = StepperMotor(STEPPER_CLK_PIN, STEPPER_DIR_PIN, STEPPER_EN_PIN)  # 2차 잠금 (메인)
        self.buzzer = Buzzer(BUZZER_PIN)
        print("Locker 초기화 완료 (2중 잠금 시스템)")

    def open(self):
        """
        2중 잠금 해제 순서:
        1. 부저 소리
        2. 서보모터 해제 (1차 잠금)
        3. 스테핑 모터 해제 (2차 잠금)
        """
        try:
            print("\n 2중 잠금 해제 시작")

            # 1단계: 부저 소리 (별도 스레드)
            if self.buzzer:
                threading.Thread(target=self.buzzer.play_success_sound).start()

            # 2단계: 1차 잠금 해제 (서보모터)
            print("1단계: 서보모터 동작")
            self.servo.unlock()
            time.sleep(0.3)  # 서보 동작 완료 대기

            # 3단계: 2차 잠금 해제 (스테핑 모터)
            print("2단계: 메인 잠금 해제 중")
            self.stepper.open_lock()

            print("2중 잠금 해제 완료 \n")
        except Exception as e:
            print(f"열림 오류: {e}")
            traceback.print_exc()
```

6. 추진 체계

이성진 (팀장)
업무: 시스템 설계 지원, 하드웨어 및 소프트웨어 통합, 테스트 환경 구축.
역할: 하드웨어 부품 선정, 회로 설계 및 하드웨어 통합 테스트.

고동연 (팀원)
업무: 모바일 앱 개발 (UI/UX 설계), MQTT 통신 및 REST API 서버 개발.
역할: 앱 제어 기능 구현, 서버와 데이터 통신, API 개발 및 백엔드 관리.

김도현 (팀원)
업무: 하드웨어 설계 (Raspberry Pi, 센서, 모터 연결), 시스템 구성 및 전원 관리.
역할: 하드웨어 연결 및 소프트웨어 통합 작업 지원, 프로젝트 테스트 및 디버깅 참여.

유창민 (팀원)
업무: 하드웨어 설계 (모터 제어, 센서 통합), OTP 인증 시스템 설계, 데이터 암호화 및 클라우드 서버 관리.
역할: 하드웨어 총괄 및 테스트, 보안 시스템 구축, 클라우드 서버와 데이터 관리.

7. 추진 일정

수행 내용		일정					
		1	2	3	4	5	6
목표와 기준 설정	- 주제 선정, 설계목표 설정 - 시장조사, 문헌조사						
	- 소프트웨어 요구사항 파악 - 필요 모듈 구매						
합성	- 기능 블록 구성 - 기능별 구현 방법 정리 - 적용할 이론 및 기술 정리 - UI 설계						
분석	- 세부 기능 블록도 - 목표달성가능성 확인 - 순서도 작성 - 모듈 연결 방법 정리 - 통신 방법 정리						
제작	- 제어 S/W 코딩 - 모듈 연결 및 배치 - UI 구현 - 외형 제작						
시험/평가	- 시험 및 검증 - Trouble shooting - 재설계						
결과	- 결과보고 및 시연						

8. 문제상황 및 해결 방안

8.1 스마트 택배함 앱-라즈베리파이 연동 문제

- **문제상황**

• 앱에서 MQTT를 통해 택배함 열기/닫기 명령 전송 시 라즈베리파이에서 로그는 정상 수신되나 모터가 동작하지 않는 문제 발생.
반면 라즈베리파이 키패드에서 OTP 입력 시에는 모터가 정상 작동함.

MQTT 콜백 함수는 백그라운드 스레드에서 실행되는 반면, GPIO 제어는 PyQt의 메인 스레드에서만 안전하게 동작하는 스레드 충돌 문제로 판단.

• 라즈베리파이 키패드로 인증 후 서버가 MQTT 명령을 재발행하여 부저 소리가 2회 울리는 중복 실행 문제 발생.
앱 사용 후 키패드 사용, 키패드 사용 후 앱 사용 시 local_action_flag가 계속 True 상태로 유지되어 첫 번째 명령이 무시되고 두 번째 명령부터 동작하는 문제 발생.

- 해결 방안

• PyQt의 시그널-슬롯 메커니즘을 활용한 MQTTSignalBridge 클래스를 구현하여 MQTT 백그라운드 스레드에서 수신한 명령을 pyqtSignal을 통해 메인 스레드로 안전하게 전달하도록 수정.
라즈베리파이 키패드에서 로컬 동작 시 local_action_flag를 True로 설정하여 서버에서 재발행되는 MQTT 명령을 무시하도록 처리하여 부저 중복 재생 방지.

• QTimer.singleShot을 활용하여 로컬 동작 후 1초 뒤 local_action_flag를 자동으로 False로 초기화하여 키패드와 앱 간 교차 사용 시에도 정상 동작 보장. 서보모터(1차 잠금)와 스테핑 모터(2차 잠금)의 순차적 동작을 Locker 클래스 내 open() 및 close() 메서드에서 time.sleep()을 통해 동작 완료 대기 시간을 추가하여 2중 잠금 시스템의 안정성 확보.
결과적으로 앱과 키패드 모든 경로에서 모터 제어가 정상 동작하며 부저 소리도 1회만 재생되고 교차 사용 시에도 즉시 반응하도록 개선 완료.

```python
#  MQTT 시그널 핸들러 - 메인 스레드에서 실행됨
def _handle_mqtt_open(self):
    """MQTT open 시그널 수신 시 호출 (메인 스레드)"""
    print("[KeypadWindow] _handle_mqtt_open 호출됨(메인 스레드)")

    # 로컬에서 실행 중이면 무시 (중복 실행 방지)
    if self.local_action_flag:
        print("[KeypadWindow] 로컬 실행 중 - MQTT 명령 무시")
        return

    try:
        self._msg("앱에서 열기 요청")
        self.locker.open()
        self.enter_btn.setText("닫기")
        for b in self.buttons:
            b.setEnabled(False)
        self.input.clear()
        self._msg("열림 (앱) - 닫기는 버튼 또는 앱")
        print("[KeypadWindow] 열기 완료")
    except Exception as e:
        print(f"[KeypadWindow] 열기 오류: {e}")
        traceback.print_exc()
```

```python
def _open_lock(self):
    try:
        self.local_action_flag = True  # 로컬 실행 플래그 설정
        # 1초 후 자동 초기화
        QTimer.singleShot(1000, lambda: setattr(self, 'local_action_flag', False))

        self.locker.open()
        self.enter_btn.setText("닫기")
        for b in self.buttons:
            b.setEnabled(False)
        self.input.clear()
        self._msg("열림 (닫기 버튼으로 잠금)")
    except Exception:
        traceback.print_exc()
```

8.2 앱 원격 제어 시 다른 사용자 비밀번호로 인증 가능한 보안 취약점

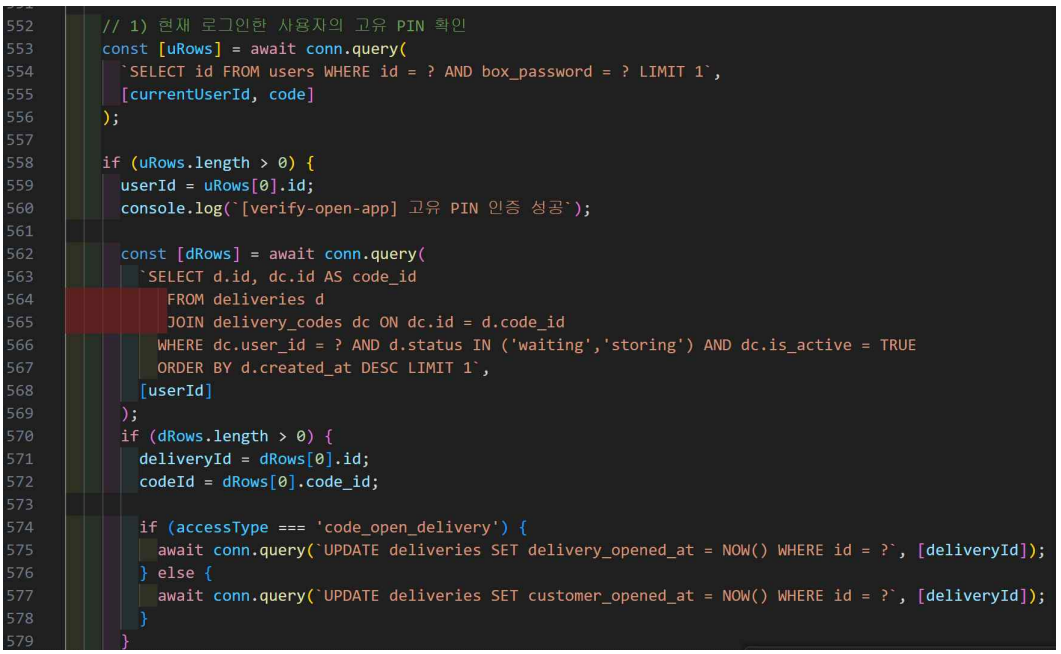
- **문제상황**

• 초기 시스템 설계에서 앱을 통한 원격 제어 기능은 사용자가 입력한 4자리 개인 비밀번호를 검증할 때, 단순히 데이터베이스의 모든 사용자 중에서 해당 비밀번호와 일치하는 계정을 찾아 인증을 성공 처리하는 구조였다.

- **해결방안**

• JWT 토큰 기반 사용자 식별을 강화하여 비밀번호 인증 체계를 악용하여 발생할 수 있는 문제를 근본적으로 차단하는 방향으로 서버 로직을 개선하였다. 먼저 authenticateToken 미들웨어를 통해 요청을 보낸 사용자의 ID를 JWT에서 추출하여 인증 단계에서 사용자 고유 ID와 비밀번호를 동시에 검증하는 이중 인증 체계로 변경하였다.

• 기존처럼 단순히 동일한 비밀번호만 일치하면 다른 사용자의 고유 사물함도 열릴 수 있으므로, 수정된 코드에서는 WHERE id = ?, AND box_password = ? 조건을 사용하여 로그인한 사용자의 본인 OTP만 보관함이 개방되도록 했다. 또한 키패드와 앱의 인증 방식을 분리하여, 키패드는 OTP 기반 /verify-open, 앱은 JWT 기반 본인 인증 /verify-open-app으로 명확히 구분함으로써 사용자 인증 체계의 안정성과 책임성을 확보하였다.



```
552   // 1) 현재 로그인한 사용자의 고유 PIN 확인
553   const [uRows] = await conn.query(
554     `SELECT id FROM users WHERE id = ? AND box_password = ? LIMIT 1`,
555     [currentUserId, code]
556   );
557
558   if (uRows.length > 0) {
559     userId = uRows[0].id;
560     console.log(`[verify-open-app] 고유 PIN 인증 성공`);
561
562     const [dRows] = await conn.query(
563       `SELECT d.id, dc.id AS code_id
564          FROM deliveries d
565          JOIN delivery_codes dc ON dc.id = d.code_id
566         WHERE dc.user_id = ? AND d.status IN ('waiting','storing') AND dc.is_active = TRUE
567         ORDER BY d.created_at DESC LIMIT 1`,
568       [userId]
569     );
570     if (dRows.length > 0) {
571       deliveryId = dRows[0].id;
572       codeId = dRows[0].code_id;
573
574       if (accessType === 'code_open_delivery') {
575         await conn.query(`UPDATE deliveries SET delivery_opened_at = NOW() WHERE id = ?`, [deliveryId]);
576       } else {
577         await conn.query(`UPDATE deliveries SET customer_opened_at = NOW() WHERE id = ?`, [deliveryId]);
578       }
579     }
```

9. 결론 및 기대효과

9.1 결론

본 프로젝트는 단순한 하드웨어 제작을 넘어, IoT 기술의 실생활 적용 가능성을 입증하고 사회적 문제를 기술로 해결하는 의미 있는 활동이었다. OTP 기반 2중 인증 방식, MQTT 실시간 통신, 클라우드 기반 데이터 관리 등 최신 기술을 융합하여 안전하고 편리한 택배 수령 환경을 구현하였다.현재 프로토타입 단계에서 앞으로 센서 기반 택배 감지 기능의 정확도를 더욱 높이기 위해 각 종 센서의 오차값 보정 알고리즘과 노이즈 필터링 기법을 적용할것이고, 향후 상용화 및 확장 가능성이 충분하다고 판단된다.

본 프로젝트를 통해 팀원 모두는 임베디드 시스템 설계, IoT 통신 프로토콜, 클라우드 인프라 구축, 모바일 앱 개발 등 전방위적인 기술 역량을 배양하였으며, 관리자용 대시보드를 제작하여 다양한 조건(날짜별, 번호별, 사용자 ID별 등)으로 데이터 분석이 가능해진다면 이는 향후 관련 분야의 진로 개척에 중요한 밑거름이 될 것이다.

끝으로, 본 프로젝트가 택배 도난 문제로 불편을 겪는 1인 가구와 원룸 거주자들에게 안전하게 물건을 보관할 수 있는 시스템을 제공하기 위해 기획하였다. 누구나 쉽게 사용할 수 있는 스마트 보관함을 구축함으로써, 사용자는 집을 비운 시간에도 택배를 안심하고 맡길 수 있고, 관리자는 물품 보관 현황을 한눈에 확인하며 체계적으로 관리할 수 있다. 또한 카메라 인증, 센서 감지, 원격 제어 같은 기능을 통해 기존 보관함보다 보안성과 편의성이 높아져 실제 생활 속 문제를 직접적으로 해결하는 데 도움이 될 것이다.

9.2 기대효과

1) 원룸 가구 및 아파트 단지 내 택배 도난 문제 해결

스마트 보관 시스템을 구축하여 편리하지만 기존 배송 방식보다는 보안적으로 강화된 시스템을 사용자에게 제공함으로써 초기에 도난 문제를 해결할 수 있다.

2) 개인정보 탈취 및 악용 방지

도난 방지 외에도 기존 택배 배송 방식은 단순히 현관문 앞에 배송되어 택배 운송자 정보를 타인에게 노출될 수 있어 도난 방지를 넘어서 신변 보호에 위협을 가할 수 있는 위험이 있다.그래서 스마트 무인 보관함을 통해 비대면 배송 및 수령을 통해 보다 안전하고 편리한 서비스를 제공하고자 한다.

3) 체계화 된 보안 시스템 구축

앱을 통해 암호환 된 OTP를 생성해서 사용자에게 제공하거나, 직접적인 잠금 장치 제어를 통해 보관함에 택배 도착 시 카메라 및 초음파센서, 하중센서를 통해 감지하고 사용자에게 실시간으로 전송하게 하여 여러 시스템을 연계해서, 보다 복잡한 보안 시스템을 구축하였다. 또한 택배기사님과 시스템 사용자 간의 여러 가지 수령 시나리오를 고려하여 상황에 맞는 보안 시스템을 구축했다.

4) 스마트 홈 기술 적용 사례 확장

최근 도난 방지 사례를 찾아보면 40대 이상이많이 적발되는 사례가 있는 것 같다. 물론 20~30대는 생활고나 단순 호기심으로돌발적인 범죄가 일어나지만 40~50대 이상은 상습 절도 전과자이거나 지속적으로 도난을 하는 경우가 많은 것 같다. 그래서 스마트 시스템을 구축함으로써 보다 노인층이 디지털 기술에 약하다는 점을 이용해 일부 연령층의 도난 방지를 구축하는 것도 방법이라고 생각했다.

5) 실제 설치 및 적용 가능한 제품 구축

원룸 및 오피스텔, 아파트 단지 내에서 실질적으로 설치 가능한 현실성 높은 제품을 설치할 수 있도록 한다.

6) 스마트 홈, IoT기술 이해 및 확장성 확보

라즈베리파이, 센서, 카메라, 네트워크 연동을 통한 IoT 시스템 구축 경험을 바탕으로, 향후 스마트 보관함, 분실물 보관함, 공유 우편함 등 다양한 응용 서비스로 확장 가능하다

10. 회의록

<table>
<tr><th colspan="4">회 의 록 1</th></tr>
<tr><td>회의명</td><td colspan="3">브레인스토밍을 통해 프로젝트 주제 도출 후 최종 주제 선정</td></tr>
<tr><td>일 시</td><td>2025.10.01</td><td>장 소</td><td>성결관 207호</td></tr>
<tr><td>의 제</td><td colspan="3">주제 후보들의 예상 결론 내용과 장단점 파악</td></tr>
<tr><td>토의
내용</td><td colspan="3">1. IoT 세탁물 관리 시스템
-세탁기 또는 세탁 바구니에 부착된 무게센서·습도센서를 통해 세탁물 양·건조상태를 자동 측정
-세탁 완료 시점, 세탁물 건조 알림을 스마트폰 앱으로 전송
-세탁 진행 상태(남은 시간, 건조 여부)를 실시간 대시보드 UI로 확인

장점:사용자가 세탁 완료/건조 여부를 수시로 확인할 필요가 없고, 다량의 세탁물이 필요할 때(기숙사·가정), 효율적인 관리 가능하다. IoT 센서 기반으로 유지비가 낮고 확장성이 높다

단점:센서 정확도가 낮으면 잘못된 알림이 가능하고, 세탁기 모델마다 연동 방식이 달라 표준화 어렵다. 습기 많은 환경에서 센서 내구성 문제 발생 가능하다

2. 심박수·산소포화도 실시간 측정 웨어러블
-손목 밴드 또는 클립형 장치에 PPG 센서, SpO2 센서내장
-측정된 심박수·산소포화도를 Bluetooth → 앱으로 실시간 전송
-사용자 건강 데이터(맥박 패턴, 위험 알림)를 그래프 형태 시각화

장점: 건강 이상(저산소증, 부정맥 등) 조기 감지 도움되고, 실시간 모니터링으로 운동·헬스케어 활용도 높다. 착용형이므로 휴대가 편하고 사용자 경험 우수하다
단점
센서 품질에 따라 정확도 편차가 크고, 지속 착용 시 배터리 관리 필요하다.의료기기 수준의 정확도 검증이 어렵다.

3. 택배 도난 방지 스마트 보관함
-스마트 락(솔레노이드 락)과 RFID·QR·앱 인증시스템이 결합된 보관함
-택배 기사나 사용자만 인증 후 문 열림
-보관함 열림/침입 시도 발생 시 카메라 촬영 + 앱 알림 · 서버 전송

장점: 택배 도난 문제를 근본적으로 해결할 수 있고, 다양한 인증 방식을 적용 가능하다. 관리자가 서버에서 보관 현황 및 사용 로그 확인 가능하다.

단점: 제작 비용이 다른 프로젝트보다 높고, 네트워크 장애 시 인증,개폐 기능에 지연이 발생한다. 실외 설치 시 방수 및 보안강화가 필요하다.</td></tr>
<tr><td>결의
사항</td><td colspan="3">제안된 세 가지 프로젝트인 IoT 세탁물 관리 시스템, 심박수·산소포화도 실시간 측정 웨어러블, 택배 도난 방지 스마트 보관함에 대해 제작 난이도, 기술성, 실현 가능성, 활용성 측면에서 각각의 장단점을 종합적으로 검토하였다.이번 회의 이후 논의와 평가를 거쳐 최종 프로젝트 주제를 '택배 도난 방지 스마트 보관함'으로 선정하기로 결의하였다.</td></tr>
<tr><td>이견
사항</td><td colspan="3">-</td></tr>
<tr><td>작성자</td><td>김도현</td><td>작성일</td><td>2025.10.01</td></tr>
</table>

회 의 록 2

회의명	최종적으로 선정 된 주제 관련해서 구체화 논의		
일 시	2025.10.06	장 소	Discord 온라인 회의
의 제	동작 절차 및 결과를 구체적으로 설계		
토의 내용	**1. 프로젝트 초기 목표 설정** -택배 도난 방지 스마트 보관함 주제를 확정하고 동작 절차 및 결과를 구체적으로 설계한다 . - 1 인 가구 및 원룸에서 발생하는 택배 도난 사례 및 기대효과를 조사하여 해당 결과물에 필요성과 타당성을 검토한다 . **2.하드웨어 및 소프트웨어 설계 및 동작 절차** - 라즈베리파이를 중심으로 IR , 초음파 , 무게 센서가 택배 도착을 감지하면 카메라가 자동으로 촬영하여 서버와 앱으로 전송된다 . 사용자는 앱에서 생성한 OTP나 개인 비밀번호를 입력해 인증하게 되면 , 성공 시 서보 모토가 작동해 잠금이 해제된다 . **3. 소프트웨어 구현** - 사용자용 앱 : 지문 인식 , 비밀번호 설정 , 원격 잠금 제어 , 도착 알림 및 택배 실물 사진 확인 기능 구현 예정 - 중앙 서버 : 하드웨어와 앱 간의 데이터 통신 및 관리 . 수령과정과 상태 변화 또한 DB에 기록되며 , gps 모듈을 통해 보관함 위치와 상태를 관리자 또한 확인할 수 있다 . **4.기대효과 및 최종 결과물 예상** • **기대 효과** - 택배 도난 문제 해결 : 사용자가 직접 제어하는 강화된 보안시스템을 통해 택배 도난을 사전에 방지 - 개인정보 보호 : 택배 운송장에 노출될 수 있는 개인정보 탈취 위험 차단 - 체계적인 보안 시스템 구축 : 앱을 통한 원격 제어 , 실시간 알림 등 다중 보안 시스템을 연계하여 보안성 강화 - 스마트홈 기술 확정 : IOT 기술을 실생활 문제 해결에 적용한 사례를 구축하고 , 기술에 취약한 연령층의 도난문제까지 해결 - 제품 상용화 가능성 : 실제 원룸 , 오피스텔 , 아파트 단지 등에 설치 가능한 현실성 높은 제품 구축 • **최종 결과물** - 스마트 택배 보관함 시제품 : 라즈베리파이와 각종 센서 , 잠금장치가 결합된 하드웨어 - 사용자용 모바일 앱 : 보관함 상태 확인 및 잠금 제어가 가능한 앱 제작 - 프로젝트 결과 보고서 및 시연 영상 : 프로젝트의 전 과정과 결과물을 정리한 문서와 실제 작동 모습을 담은 영상 제출		
결의 사항	다음 회의때는 전체적인 프로젝트 추진 일정과 시나리오 별 예상 문제점 및 동작 방식을 구상한다.		
이견 사항	-		
작성자	이성진	작성일	2025.09.27

회 의 록 3

회의명	동작 시나리오 별 예상 문제점 및 인증 체계 구상		
일 시	2025.10.10	장 소	성결관 207호
의 제	추진 일정을 수립하고, 시나리오 별 예상 문제점, 동작을 구상해본다		
토의 내용	**1. 추진 일정** - 1주차 : 주제 선정 및 시장 조사 (택배 도난 사례 , 유사 제품 분석) - 2주차 : 기능 및 시스템 구조 설계 및 앱 개발 시작 - 3 ~ 4주차 : 필요 부품 선정 및 하드웨어 연결 테스트 - 5 ~ 9주차 : 센서 감지 로직 구현 및 각 센서 모듈화 개발 - 10 ~ 12주차 : 사용자 앱 개발 완료 - 13 ~ 14주차 : 전체 시스템 연동 및 최종 제품 시연 테스트 - 15주차 : 최종 발표 및 피드백 **2. 시나리오 별 동작 방식(보관함 속 택배의 유무에 따른 절차)** **- 시나리오1 (첫번째 택배 도착 후)** 물품 감지 후 '자동 잠금' 방식 : 기사님이 물품을 넣은 후 센서가 감지한다 . 카메라 촬영 및 이미지가 전송되고 보관함은 자동으로 잠금된다 . 물품 감지 후 '사용자 원격 제어' 방식 : 사용자 원격 제어 방식으로써 사용자가 전용 앱을 통해 잠금 장치를 실시간으로 직접 제어할 수 있다 . **- 시나리오2 (두번째 이후 택배 도착)** 택배 기사님 직접 인증 방식 : 기사님이 도착 후 고유 코드나 인증 장치를 태그하면 보관함이 열리면 물품을 넣고 다시 닫으면 다시 잠기는 방식이다 . 사용자 원격 제어 방식 : 동일하게 기사님이 보관함의 호출 버튼을 누르면 사용자에게 알림이 전송된다 . 그 후 사용자가 앱을 통해 원격으로 문을 열어주고 닫는것까지 제어할 수 있다 .		
결의 사항	다음 회의때는 전체적인 프로젝트 추진 일정과 시나리오 별 예상 문제점 및 동작 방식을 구상한다.		
이견 사항	-		
작성자	고동연	작성일	2025.10.10

회 의 록 4

회의명	택배 도난 방지 스마트 보관함의 구체적인 설계 및 동작 절차 논의		
일 시	2025.10.15	장 소	성결관 207호
의 제	세부적인 동작 절처와 설계 및 기획 시작		
토의 내용	**1. 동작 시나리오 확정** 비밀번호를 라즈베리파이가 직접 관리하는 구조가 아니라, 서버(MySQL + REST API)가 생성·저장·검증까지 모두 담당하는 구조로 확정되었다. 사용자는 앱에서 일회용 택배 비밀번호를 생성하면 서버가 난수 생성 후 이를 MySQL DB에 저장하고, 사용자는 해당 코드를 택배기사에게 전달한다. 택배기사가 보관함에 도착해 PyQt 키패드에 비밀번호를 입력하면 라즈베리파이는 입력값을 서버로 보내 검증을 요청하며, 서버는 DB에서 비밀번호·유효시간·사용 여부를 확인한 뒤 결과를 다시 라즈베리파이에 전달한다. 검증에 성공하면 잠금장치를 제어하여 보관함을 열고, 초음파 및 무게센서로 택배 투입 여부를 확인한 뒤 서버에 "배송 완료" 상태를 업데이트하며, 사용자는 앱에서 알림과 카메라 이미지를 받아볼 수 있다. **2. 데이터베이스 테이블 설계(MySql) :** - **user** 테이블에는 사용자 로그인 정보와 개인정보 , 사용자 고유의 4 자리 보관함 비밀번호 (box_password)를 저장하여 고유 사용자가 자신의 보관함을 안전하게 관리하도록 한다 . - **delivery_codes** 테이블은 앱에서 생성된 OTP코드를 저장하며 , 코드의 만료시간과 사용 횟수를 제한하여 임시 인증 수단으로 활용할 수 있게 하였다 . - **deliveries** 은 배송 상태를 확인 및 관리하고 , 택배의 무게 데이터와 촬영된 이미지를 동시에 저장하여 배송 과정을 사용자에게 한눈에 파악할 수 있게한다 .		
결의 사항	**진행 결과** 라즈베리파이 환경에서 PyQt를 활용한 키패드 GUI를 구성하여 사용자가 4 자리 비밀번호를 입력할 수 있도록 구현하였다 . 키패드 입력값은 서버의 DB에 저장된 비밀번호와 비교 , 검증되며 , 일치할 경우 라즈베리파이가 서브모터를 제어해 보관함의 잠금이 해제되도록 하였다 .		
이견 사항	-		
작성자	유창민	작성일	2025.10.15

회 의 록 5

회의명	앱 - 서버 통신 구조 연구 및 어플리케이션 제작 회의		
일 시	2025.10.22	장 소	성결관 207호
의 제	사용자가 보관함을 제어할 수 있는 앱 개발 후 서버와의 통신 구조에 활용		
토의 내용	**1. 앱-서버 통신 연구** 앱과 서버 간 통신은 ApiClient.java와 SmartBoxApiService.java가 핵심적으로 담당한다. 먼저 ApiClient.java는 Retrofit과 OkHttp 기반의 통신 클라이언트를 싱글턴 형태로 구성하여, 모든 요청에 Content-Type: application/json헤더를 자동으로 포함시키고, 필요 시 인증을 위한 Authorization: Bearer <token>헤더를 추가하도록 설계되어 있다. 반면 SmartBoxApiService.java는 로그인, 보관함 정보 조회, 배송 내역 조회 등 실제 기능별 API 엔드포인트를 정의하는 역할을 수행한다. 즉,ApiClient.java가 통신환경을 설정하는 기반 구성이면, SmartBoxApiService.java는 이를 이용해 실제 서버 요청을 실행하는 구조라고 할 수 있다. 다만 서버가 동작 중이 아니거나 Constants.BASE_URL 값이 잘못되어 있을 경우, 앱은 서버에 연결할수없어 ConnectException, TimeoutException, UnknownHostException과 같은 네트워크 오류가 발생할 수 있다는 점을 고려해야 한다 **2. 앱 디자인 컨셉 및 공통 UI 요소 개발** 안드로이드 Studio 환경에서 Material Design에서 가이드라인들을 참고하여 앱의 기본 컬러 , 디자인 요소 , 레이아웃 구조를 정의하였다 . 공통 UI 요소는 Custom View와 JetpackCompose Composable 함수를 이용했다 . 앱의 '빌드 설계도' 같은 역할을 하는 설정 파일 (Gradle)을 복잡한 암호문을 대체하여 , Kotlin DSL 이라는 명확하고 오류가 적은 언어로 선택하여 작성하였다.		
결의 사항	앱 개발은 주기적으로 계속 진행하고 다음 회의때는 어떤 센서들을 사용할건지 의논 후 각 센서 별 동작 테스트를 진행할 예정이다.		
이견 사항	-		
작성자	김도현	작성일	2025.10.22

<table>
<tr><th colspan="5">회 의 록 6</th></tr>
<tr><td>회의명</td><td colspan="4">라즈베리파이 모듈 연동 및 센서 동작 점검</td></tr>
<tr><td>일 시</td><td colspan="2">2025.10.29</td><td>장 소</td><td>성결관 207호</td></tr>
<tr><td>의 제</td><td colspan="4">모듈 제어 코드 작성 및 동작 테스트 결과 검토</td></tr>
<tr><td>토의
내용</td><td colspan="4">-라즈베리파이 모듈 연결 및 센서 동작 테스트
Raspberry Pi 4 경에서 Python을 기반으로 각 하드웨어 모듈의 제어 코드를 작성하였다. 서보모터는 RPi.GPIO 라이브러리의 PWM기능을 활용해 회전 각도를 제어하였고, 스피커는 pygame.mixer 라이브러리를 통해 이벤트 트리거 기반으로 단순 알림음을 재생하였다. 초음파 센서(HC-SR04)는 Trig/Echo 핀 신호를 측정하여 time-of-flight 계산으로 거리를 산출하였으며, 무게 센서(HX711 모듈)는hx711-python 라이브러리를 사용해 ADC 값을 읽고 보정 계수를 적용하여 무게를 측정하였다.</td></tr>
<tr><td>결의
사항</td><td colspan="4">Raspberry Pi에서 서보모터, 스피커, 초음파 센서, 무게 센서를 각각 제어하는 데 성공하였다. 서보모터는 원하는 각도 제어가 가능했고, 스피커는 알림음 출력이 정상적으로 동작했으며, 초음파 센서는 거리 측정에서 오차 범위가 작게 나타났다. 무게 센서 또한 hx711 모듈을 통해 보정된무게 값을 안정적으로 출력하였다. 이를 통해 개별 하드웨어 모듈이 정상 작동함을 확인하였다.</td></tr>
<tr><td>이견
사항</td><td colspan="4">문제점: 라즈베리파이에서 서보모터를 연결하여 전원 공급 후 ->동작하지 않았고 물리적인 힘을 강제로 가해도 움직이지 않음
해결점: 서보모터 문제는 우선 전압 및 전류 공급량이 충분한지 확인하고, 전원부를 보조 배터리나 별도의외부 전원(5V/5A이상)을 통해 안정적으로 공급하도록 개선하였다. 또한 PWM 신호 주파수 및 듀티비를 조정하여 모터가 정확한 제어 신호를 받을 수 있도록 수정하였다</td></tr>
<tr><td>작성자</td><td>이성진</td><td>작성일</td><td colspan="2">2025.10.29</td></tr>
</table>

<table>
<tr><th colspan="5">회 의 록 7</th></tr>
<tr><td>회의명</td><td colspan="4">라즈베리파이 기반 모터·센서 제어를 통한 문 여닫이 방식 논의</td></tr>
<tr><td>일 시</td><td colspan="2">2025.11.05</td><td>장 소</td><td>성결관 207호</td></tr>
<tr><td>의 제</td><td colspan="4">스텝모터·서보모터 테스트 결과 검토 후 문 열리는 방식 논의</td></tr>
<tr><td>토의
내용</td><td colspan="4">• 택배 보관함 개방 방식 논의
택배함 문 개방 방식을 구현하기 위해 스텝 모터와 서보 모터를 각각 테스트하고, 라즈베리파이와의 연동을 통해 실제 동작을 확인할 것 이다 또한 HTTP 통신 기반으로 센서만 독립적으로 동작시키는 구조를 실험하여 전체 시스템 구조의 확장성을 확인할 예정이다.

• 진행 내용
- 스텝 모터 제어 확인: 라즈베리파이 GPIO 핀을 통해 드라이버 모듈과 연결하여 스텝 모터 회전 각도 및 속도를 제어함.
- 서보 모터 제어 확인: PWM 신호를 이용해 서보모터의 회전 각도를 조정하고 문 잠금 해제 구조를실험함.
- 스텝모터 vs 서보모터 비교 테스트: 문을 열고 닫는 속도, 제어 정확도, 전력 소비량, 등을 비교 분석.
- HTTP 통신 기반 제어: F서버를 라즈베리파이에 구성하고, REST API를 통해 /sensor엔드포인트를 호출하면 초음파 센서만 독립적으로 작동하도록 구현했다.
- 두 모터 동시 제어 실험: Python 스레드를 사용하여 서보모터 두개를 동시에 제어했으나, 명령 신호 간 약 0.5~1초의 지연이 발생함을 확인했다.</td></tr>
<tr><td>결의
사항</td><td colspan="4">• 회의 진행 결과
- 스텝 모터는 정확한 회전 제어와 높은 토크로 견고한 문을 여닫기에 적합하나, 구동 시 전류 소모가 크고 소음이 있었다.
- 서보 모터는 응답속도가 빠르고 제어 회로가 간단하지만, 회전 각도 제한(약 180°)으로 문 구조에 따라 추가 기구적 보완이 필요했다.
- 라즈베리파이 HTTP 서버에서 초음파 센서만 동작시키는 방식으로, 앱에서 실시간 상태 확인('택배도착 감지' 기능)을 가능하게 했다.
- 두 모터를 동시에 제어할 경우, GPIO 명령 큐 처리 지연 및 Python 스레드 간 타이밍 충돌로 인해제어 타이밍이 미세하게 어긋나는 현상이 발생했다.</td></tr>
<tr><td>이견
사항</td><td colspan="4">-</td></tr>
<tr><td>작성자</td><td>이성진</td><td>작성일</td><td colspan="2">2025.11.05</td></tr>
</table>

회 의 록 8			
회의명	라즈베리파이 기반 모터·센서 제어를 통한 문 여닫이 방식 논의		
일 시	2025.11.07	장 소	성결관 207호
의 제	스텝모터·서보모터 테스트 결과 검토 후 문 열리는 방식 논의		
토의 내용	• **가상 키패드 UI 및 사용자 입력 처리 확인** PyQt로 구현된 가상 키패드 UI가 라즈베리파이 디스플레이에 정상적으로 출력되는 것을 확인했습니다. 사용자 입력(키패드 터치)이 PyQt 이벤트 루프 내에서처리되어 시스템 내부 로직(비밀번호 입력 버퍼)으로 전달되는 것을 확인했습니다 • **하드웨어 회로 구성 이론적 분석 완료** 스마트 사물함 제작에 필요한 라즈베리파이와 아두이노중심의 하드웨어 회로 구성을 이론적으로 분석하고 이해했습니다. 특히, 스텝모터 드라이버의 제어원리(ENA/DIR/PUL 신호), 센서 및 카메라의 데이터 처리 구조를 파악하여 향후 하드웨어 통합을 위한 기반을 마련했습니다		
결의 사항	다음 회의때는 라즈베리파이를 이용하여 모터 제어의 기본 원리와 동작을 복습하고, 드라이버_스텝모터_서보모터 등을 직접 연결하여 회로를 구성해보기로 했다. 사전에 회로를 미리 스케치 해와서 회의할때 바로 설계 단계로 들어갈수있게 준비 요청했다.		
이견 사항	-		
작성자	고동연	작성일	2025.11.07

회 의 록 9			
회의명	모터 제어의 원리와 동작 방식 복습		
일 시	2025.11.09	장 소	성결관 207호
의 제	스텝모터, 드라이버, 스텝모터 등을 연결하여 회로 구성		
토의 내용	라즈베리파이를 이용하여 모터 제어의 기본 원리와 동작 방식을 학습하고, 스텝모터드라이버·스텝모터·서보모터 등을 라즈베리파이에 직접 연결하여 회로를 구성하였다.또한 제어 프로그램을 작성하여 각 모터의 동작을 테스트하고, 하드웨어 및 소프트웨어 간의 연동 과정을 이해하였다		
결의 사항	라즈베리파이와 각 모듈 간의 연결을 완료하고, 서보모터·스텝모터·부저·LCD 등 주요부품의 동작을 성공적으로 확인하였다.제어 프로그램을 통해 각 모터의 회전 방향 및 속도 제어, 서보모터의 각도 제어,부저 음 발생 등 기능 테스트를 수행하였으며, 하드웨어 신호가 정상적으로 동작함을 확인하였다. WiringPi 라이브러리를 이용하여 GPIO 핀을 제어하고, 소프트웨어 PWM을 적용하여 모터의 세기 조절이 가능함을 검증하였다.이를 통해 모터 제어의 기본 원리와 신호 흐름을 이해하였으며, 하드웨어와 소프트웨어의 연동 과정에서 발생할 수 있는 타이밍 문제와 전원 공급 안정성의 중요성을체험적으로 학습하였다.		
이견 사항	-		
작성자	김도현	작성일	2025.11.09

<table>
<tr><th colspan="4">회 의 록 10</th></tr>
<tr><td>회의명</td><td colspan="3">스마트 보관함 인증·제어 기능 구현 회의</td></tr>
<tr><td>일 시</td><td>2025.11.09</td><td>장 소</td><td>성결관 207호</td></tr>
<tr><td>의 제</td><td colspan="3">앱–서버–키패드–모터 간 연동 구조를 기반으로 OTP 인증 후 모터·버저가 동작하는 전체 흐름 점검</td></tr>
<tr><td>토의
내용</td><td colspan="3">1.앱–서버 연동 방식 검토
-사용자 앱에서 서버로 요청을 보내 OTP를 생성하는 구조 확정.
-OTP 생성 시 만료 시간, 1회성 사용 여부를 서버에서 관리하도록 협의.
2.터치스크린 키패드 입력 처리
-라즈베리파이 디스플레이 기반 터치 키패드에서 사용자가 OTP를 입력하면 서버에 검증 요청을 전송.
-키패드는 Python + OpenCV/GUI 기반으로 구성하고, 입력된 데이터는 HTTP 또는 Socket 기반으로 서버에 전달하기로 결정.
3.OTP 인증 후 모터 동작 시나리오
-서버에서 OTP가 일치하면 라즈베리파이에 제어 신호 전달.
-신호 수신 시 스텝모터 또는 서보모터가 동작하여 보관함 문을 개방하도록 구조 확정.
-모터 동작 성공 여부는 라즈베리파이가 서버에 다시 반환하도록 결정.
4.버저 경고음 동작
-인증 성공 시 짧은 알림음, 인증 실패 시 연속 짧은 음으로 처리하도록 합의.
-버저 제어는 GPIO 기반으로 모터 동작과 함께 실행되도록 동기화.</td></tr>
<tr><td>결의
사항</td><td colspan="3">• 결의 사항
-OTP 인증 절차를 서버 중심으로 확정하고 앱과 키패드에서 이를 연동하기로 했다
-키패드 입력 값은 서버로 전달하여 인증하는 구조로 통일했다.
-인증 성공 시 모터 구동 및 버저 알림 방식을 최종 확정했다.
-일부 흐름(앱–서버–키패드–모터–버저)을 실제 장비로 테스트 동작을 확인했다
-다음 활동때는 만든 시스템을 외형과 결합하여 완전한 프로젝트 실물을 제작할 예정이다</td></tr>
<tr><td>이견
사항</td><td colspan="3">• 문제 상황 및 해결방안
문제점: 앱–서버–라즈베리파이 센서 간 통신 과정에서 로그는 정상적으로 수신되지만 모터가 동작하지 않는 문제가 발생

해결방안: 분석 결과, MQTT 콜백 스레드가 직접 GPIO 모터 제어를 수행하려고 하면서 권한 및 스레드 충돌 문제가 발생한 것이 원인이었다.
라즈베리파이 GPIO는 메인 스레드에서의 제어를 권장하기 때문에, 서브 스레드(MQTT 스레드)에서 제어 신호를 직접 실행하려고 하면 동작이 차단될 수 있다. 이를 해결하기 위해 MQTT 스레드가 모터 제어 요청만 전달하고, 실제 모터 구동은 메인 스레드에서 처리하도록 구조를 변경하였다. 수정 이후 제어 흐름이 안정적으로 동작하며, MQTT 메시지 수신 시 메인 스레드가 안전하게 모터와 버저를 제어할 수 있게 되어 문제를 해결하였다</td></tr>
<tr><td>작성자</td><td>김도현</td><td>작성일</td><td>2025.11.09</td></tr>
</table>