종합설계 프로젝트 수행 보고서

프로젝트명	스마트온실
팀번호	S2-8
문서제목	수행계획서( ) 2차발표 중간보고서( ) 3차발표 중간보고서( ) 최종결과보고서(O)

2020.12.04

팀원 : 노 정 민 (팀장)
강 재 연
김 경 연
박 유 진
지도교수 : 공 기 석 (인)

문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	
2020.01.22	노정민	1.0	수행계획서	최초작성
2020.02.26	노정민	2.0	2차발표자료	설계서추가
2020.05.02	노정민	3.0	3차발표자료	시험결과추가
2020.12.04	노정민	4.0	최종결과보고서	시험결과 수정

문서 구성

진행단계	프로젝트 계획서 발표	중간발표1 (2월)	중간발표2 (4월)	학기말발표 (6월)	최종발표 (10월)
기본양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식
포함되는 내용	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~3) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~4) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~5) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~7) 참고자료	Ⅰ Ⅱ Ⅲ

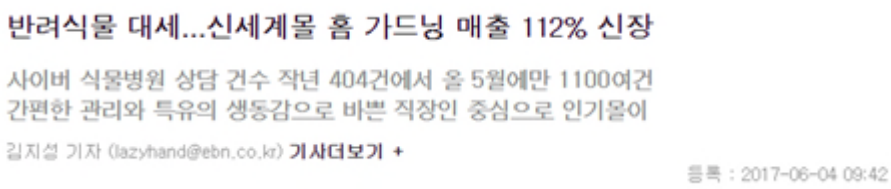
이 문서는 한국산업기술대학교 컴퓨터공학부의
“종합설계”교과목에서 프로젝트“스마트온실”을 수행하는
(S2-8, 노정민, 강재연, 김경연, 박유진)이 작성한 것으로 사용하기
위해서는 팀원들의 허락이 필요합니다.

목 차

Ⅰ. 서론

1. 작품선정 배경 및 필요성

손이 많이 가지 않으면서도 식물 자체가 가지고 있는 싱그러움을 보며 교감할 수 있는 장점으로 '반려식물'이라는 새로운 트렌드가 등장하였다. < 그림 1 >에 따르면 홈 가드닝 시장 규모가 점점 커지고 있다는 것을 알 수 있다. 하지만 이 시장 자체가 1인 가구에 가장 각광받고 있기 때문에 집을 비우거나 외부 활동을 하는 시기에는 제대로 관리가 되지 않을 수 있다. 또한 < 그림 2 >에서와 같이 제대로 된 관리에 어려움을 겪는 사람들이 많다는 것을 알 수 있다. 이러한 상황에 맞춰 반려 식물을 쉽지만 잘 키울 수 있도록 식물별로 적합한 성장을 위한 정보를 제공한다. 제공받은 정보를 바탕으로 어플리케이션을 통해 원격으로 온실을 관리할 수 있도록 하며 온실에 부착된 센서들로 데이터를 수집하고 사용자가 확인할 수 있도록 한다. 온실에 부착된 카메라를 통해 언제 어디서나 반려 식물들의 상태를 확인하고 관리할 수 있도록 한다.



반려식물 대세...신세계몰 홈 가드닝 매출 112% 신장

사이버 식물병원 상담 건수 작년 404건에서 올 5월에만 1100여건
간편한 관리와 특유의 생동감으로 바쁜 직장인 중심으로 인기몰이

김지성 기자 (lazyhand@ebn.co.kr) 기사더보기 +

등록 : 2017-06-04 09:42

< 그림 1 >

< 그림 2 >

2. 기존 연구/기술동향 분석

기존 제품으로는 대규모의 스마트 팜이나 소규모의 스마트 화분이 있으며, 현재 판매되고 있는 대표적인 스마트 화분으로는 < 그림 3 >과 같은 로팟, 블룸엔진, 플라워파워가 있다.

로팟은 식물의 컨디션을 LED를 통해 확인 가능하고 블루투스를 통해 전용 앱에서 토양의 정보를 확인할 수 있다. 하지만 현재 한국어 버전을 지원하지 않고 있으며, 품종에 대한 정보도 영문으로 제공한다. 블룸 엔진은 자동 급수 기능과 LED 조정 등의 기능이 있다. 블루투스를 통해 스마트폰 앱과 통신하며, 제품에서 키울 수 있는 식물이 정해져 있다. 플라워파워는 화분에 꽂아서 사용하는 방식으로 빛, 온도, 비료, 습도를 측정한다. 블루투스를 통해 전용 앱에 데이터를 전송하고 스마트폰에서 알림을 받을 수 있는 방식이다.

제품별 기술 분석은 < 표 1 >에 정리되어 있다. 집에서 기를 수 있는 소규모 제품에는 전체적으로 스마트폰 애플리케이션은 모두 가지고 있고, 자동 급수 기능도 존재한다. 하지만, 와이파이가 아닌 블루투스 통신을 사용하므로 원격 제어가 불가능하다. 또한, 영상 스트리밍 기능도 존재하지 않는다. 스마트 화분이기 때문에 식물 개수도 하나로 한정되어 있으며, 대규모의 스마트 팜이 아닌 실내에서 쉽게 기를 수 있는 소규모의 스마트 온실은 아직 개발되지 않았다.

< 그림 3 > 기존 제품(로팟, 블룸엔진, 플라워파워 순)

< 표 1 > 제품별 기술 분석

제품명	와이파이	스마트폰 앱	영상 스트리밍	자동 급수	식물 개수
로팟	X	O	X	X	1
플라워파워	X	O	X	X	1
블룸엔진	X	O	X	O	1
스마트온실	O	O	O	O	3

3. 개발 목표

스마트 온실의 개발 목표는 크게 두 가지다. 앱을 통한 온실 모니터링 및 환경 제어와 작물 별 최적의 생장 환경에 따른 원격 제어이다.

앱을 통한 온실 환경 제어 기능으로는 온/습도 상태를 체크하여 쿨링팬을 제어하는 기능과 물탱크에 물이 부족할 경우 사용자에게 알림을 전달하는 기능이 있다. 또한 온실에 부착된 카메라를 통해 실시간으로 온실의 상태를 확인하고 촬영을 할 수 있다.

작물 별 생장 환경에 따른 원격제어로는 다양한 식물 품종의 생장환경에 대한 정보를

제공하고, 그 정보를 토대로 자동 급수, 환풍, 광량 조절 등의 기능을 수행한다.

4. 팀 역할 분담

팀 역할 분담은 아래 < 표 2 > 팀원별 역할 분담과 같다.

< 표 2 > 팀원별 역할 분담

	노정민	강재연	김경연	박유진
자료수집	· 사례 조사 · 아두이노-라즈베리파이 연결 방법 · Pi 카메라 스트리밍 방법 · 온실 제작 방법	· 사례 조사 · 라즈베리파이-서버 통신 방법 · Pi 카메라 스트리밍 방법 · 자동 급수기 제작 방법	· 사례 조사 · 라즈베리파이-서버 통신 방법 · 아두이노-라즈베리파이 전력 문제 · 자동 급수기 제작 방법	· 사례 조사 · DB-App 데이터 연동 방법 · 센서 데이터 DB 저장 방법 · 온실 제작 방법
설계	· Web 서버 구성 및 DB 설계 · 온실 설계	· Web 페이지 설계 및 디자인 · 데이터 통신 설계	· Web 서버 구성 및 DB 설계 · 온실 설계	· App 설계 및 디자인 · 데이터 통신 설계
구현	· Web 서버 구축 · DB 구현 · 아두이노 설계 · 온실 제작	· App 구현 · Web 페이지 구현 · 라즈베리파이 UV4L 영상 스트리밍 구현	· Web 서버 구축 · DB 구현 · 아두이노 설계 · 온실 제작	· App 구현 · Web 페이지 구현 · 라즈베리파이 UV4L 영상 스트리밍 구현
테스트	통합 테스트 및 유지보수			

5. 개발 일정

< 표 3 > 개발 일정을 기반으로 12월부터 1월까지 사전조사 및 요구사항 분석을 한다. 조사한 내용을 토대로 1월부터 4월까지 App 설계 및 제작, 서버 구축 및 DB 설계, 스마트 온실 설계 및 제작을 한다. 제작을 완료한 후 5월부터 데모 및 유지보수를 한다. 5월까지 모듈 간 통신 통합을 끝낸 후, 6월 달에 문서화 및 발표를 진행한다. 6월부터 9월까지 최종 검토 및 발표를 준비하고 최종 발표를 한다.

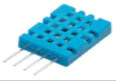
< 표 3 > 개발 일정

	개발 사항	12월	1월	2월	3월	4월	5월	6월	7-9월
개발 일정	사전조사 및 요구사항 분석								
	App 설계 및 제작								
	서버 구축 및 DB 설계								
	스마트 온실 설계 및 제작								
	모듈 간 통신 통합								
	문서화 및 발표								
	데모 및 유지보수								
	최종 검토 및 발표								

6. 개발 환경

H/W 개발 환경으로는 < 표 4 >의 내용을 바탕으로 아두이노 우노와 라즈베리파이 3B, 그리고 센서 데이터 값을 측정할 수 있는 수위 센서, 온습도 센서, 조도 센서, 토양 수분 센서를 사용한다. 또한, 스마트 온실 설계 시 물펌프, 물탱크, 식물 생장용 LED, DC 쿨링팬, 그리고 식물의 생장 과정을 확인할 수 있는 라즈베리파이용 카메라를 사용하여 개발한다.

< 표 4 > 하드웨어 개발 환경

HW 부품		기능
	Arduino UNO (아두이노 보드)	아두이노 보드
	Raspberry Pi 3 B (라즈베리 파이 보드)	라즈베리파이 보드
	HS-WATER-SENSOR (수위 센서)	물탱크의 수위를 측정하기 위한 수위 센서
	DHT11Lib (온습도 센서)	온실 내의 온도와 습도를 측정하기 위한 온습도 센서
	GL5537 (조도 센서)	빛의 세기 측정을 위한 조도 센서
	DM456 (토양 수분 센서)	토양의 수분을 측정하기 위한 토양 수분 센서
	DWP-370N (물펌프)	자동 급수를 위한 물펌프
	(물탱크)	식물에 공급할 물을 저장하고 있는 물탱크
	8MP CAMERA (파이카메라)	온실의 실시간 영상 확인을 위한 카메라 보드
	HB122 (식물 생장용 LED)	식물의 적합한 생장 환경을 위한 식물 생장용 LED
	DC 5V (DC 쿨링팬)	온실의 환기를 위한 DC 쿨링팬

S/W 개발 환경으로는 < 표 5 > 내용을 바탕으로 AWS EC2를 이용하여 가상 서버를 생성하고, AWS RDS를 이용하여 MySQL DB 인스턴스를 생성, 웹 서버와 연동 가능한 가능하도록 한다. 라즈베리파이에는 Raspbian OS를 설치하고, 아두이노 IDE를 사용해 아두이노 코드를 작성하여 업로드 한다. Python을 사용하여 라즈베리파이와 EC2 가상 서버의 통신 및 DB에 데이터 업로드를 가능하도록 한다. DB 생성은 Mysql workbench, 웹서버 구축은 Apache Tomcat, 어플리케이션 개발은 Android Studio, 웹 개발은 Eclipse를 사용한다.

< 표 5 > 소프트웨어 개발 환경

종류	소프트웨어명	버전
Cloud Server	AWS EC2	
Raspberry Pi	Raspbian OS	ver 4.14
Externel Programming Language	Python	ver 2.7.13
DBMS	MySQL	ver 5.7.24
Android Development Tool	Android Studio	ver 3.4.2
Arduino Development Tool	Arduino IDE	ver 1.8.10
Web Server	Apache Tomcat	ver 9.0.26
Web	Eclipse	ver 4.11

Ⅱ. 본론

1. 개발 내용

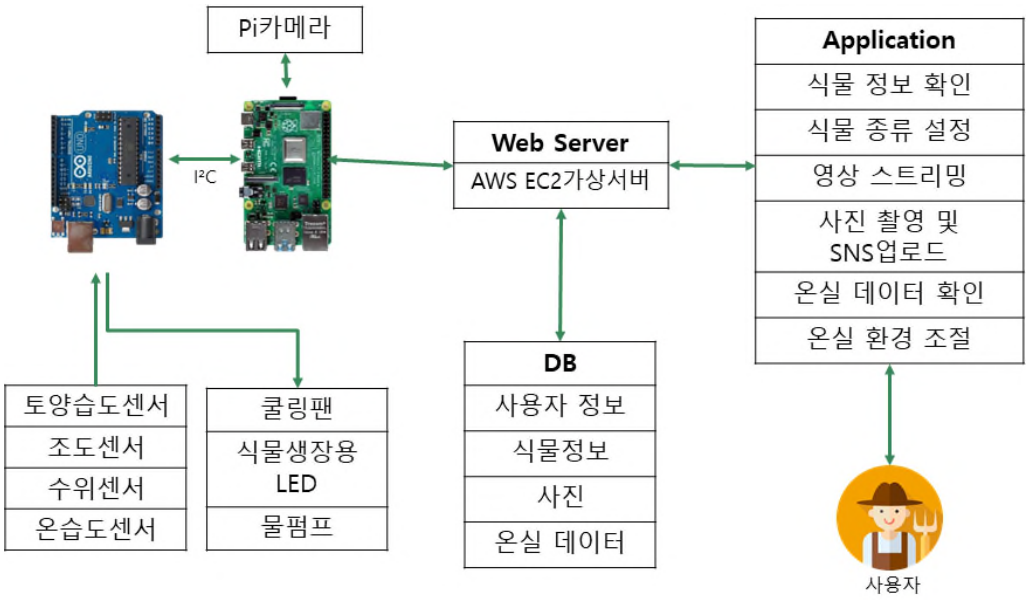
< 그림 4 >와 같이 라즈베리파이는 아두이노로부터 json 형식 메시지를 10초 간격으로 받아온 후 웹 서버에 전달한다. 또한, 사용자가 app에서 설정한 값을 받아와 아두이노의 작동을 제어한다. UV4L을 이용하여 영상 스트리밍 서버를 생성하고 APP과 WEB에서 Pi Camera 영상을 확인하도록 한다.

아두이노는 센서의 데이터 값을 측정하고 장치를 제어하는 등의 기능을 수행한다. 아두이노와 라즈베리파이는 시리얼 통신을 하여 데이터를 주고받는다. 센서를 통해 받아온 측정값을 JSON 형식으로 변환하여 라즈베리파이에 전달한다. 또한 라즈베리파이로부터 받아온 값을 이용해 워터펌프, 쿨링팬 등의 장치 동작을 제어한다.

웹 서버는 아마존 웹 서비스(AWS)의 EC2 가상 서버를 생성하고 아파치 톰캣을 이용하여 구축한다. 소켓을 이용한 통신으로 라즈베리파이로부터 데이터를 받아오고 아마존 RDS DB와 연동하여 받아온 데이터를 DB에 업로드한다. AWS RDS를 사용해 MySQL DB 인스턴스를 생성하고 웹 서버와 연동한다. DB에서 다양한 식물 정보와 센서 측정값, 식물의 사진, APP에서 입력받은 사용자 정보 등의 데이터를 저장하고 관리한다.

APP에서는 회원가입, 로그인, 아이디 또는 비밀번호 찾기 등의 기능을 제공하여 자신의 온실을 관리할 수 있도록 한다. DB에 저장된 정보로 그래프를 생성하여 사용자에게 시각화된 정보를 제공한다. 현재 온실의 상태를 영상으로 확인할 수 있고 사진을 촬영해 기록할 수 있다. 이때 촬영한 사진을 SNS로 컨텐츠 공유가 가능하다. 농업 전문 포털 사이트 '농사로'의 오픈API를 이용하여 온실에서 기르기 적합한 식물들의 정보를 확인하고 자신이 기르는 식물을 선택할 수 있다.

온실은 폴리카보네이트 소재를 이용하여 육면체로 제작한다. 각종 센서들은 벽면에 부착한다. 자동급수기는 아두이노에 워터펌프를 연결하여 구동한다. 워터펌프를 사용하여 물을 끌어올리는 방식으로, 워터 펌프에 튜브를 연결하여 구성한다.



< 그림 4 > 시스템 구성도

2. 문제 및 해결방안

센서로 들어온 데이터에 대한 가공을 통하여 부가적인 정보를 생성할 수 있도록 식물별 맞춤형 설정이 필요하며, 식물종류와 정보의 연관성 없이 one-way 통신만 있는 문제가 있다. 이를 해결하기 위해 DB에 저장된 데이터를 가공하여 사용자에게 시각화된 정보를 제공한다. 시각화된 정보는 안드로이드 스튜디오에서 그래프 라이브러리를 사용하여 개발한다. 또한, 식물의 성장 기간에 따라 다른 값을 적용해서 성장 시기에 맞는 적합한 환경을 구성할 수 있도록 한다. 양방향 통신에 관한 문제로는 Tensorflow를 통한 머신러닝 기능 이용을 고려하는 중이다.

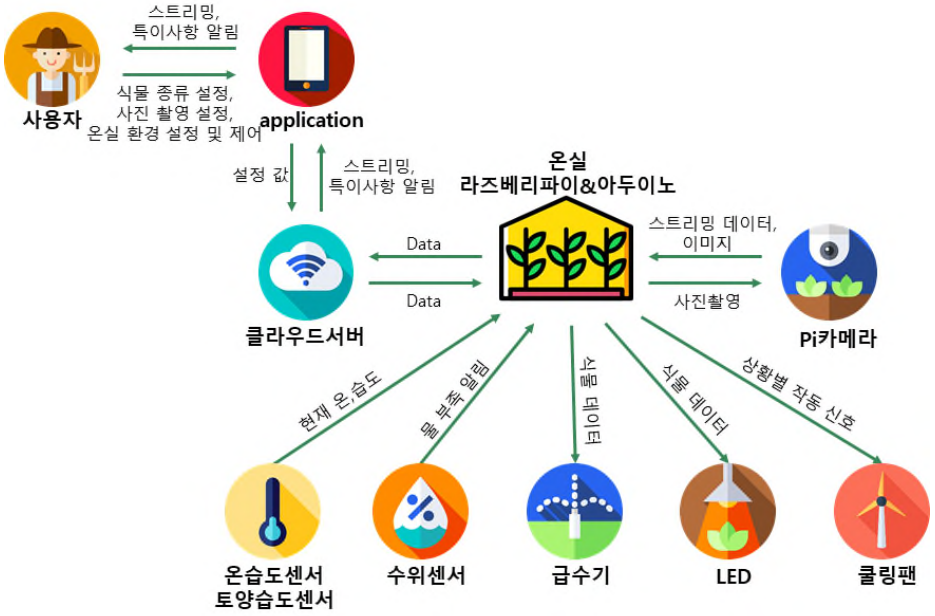
AWS IoT , MQTT 프로토콜을 사용하려 하였으나 진행과정에서 두 가지를 사용하지 않기로 결정하였다. MQTT 메시지 브로커에 장치를 등록하고 구독, 발행 등을 시도하는 과정에서 크고 작은 문제가 발생하였는데 그것을 해결할 만한 데이터가 부족하여 사소한 문제를 해결하는 데에도 많은 시간이 필요했다. 이런 상황에서 위 두 가지를 계속 사용하는 것은 앞으로 개발에 많은 영향을 끼칠 것이라 생각했다. 해결방안으로 MQTT 프로토콜을 사용하지 않는 대신 소켓통신을 사용하기로 하였다. 아두이노에서 센서 값을 라즈베리파이에 전송하고, 라즈베리파이에서는 아두이노에서 받아온 데이터를 소켓통신을 통해 서버에 전송한다. 서버에서 아두이노로 설정 값을 전송할 때에도 마찬가지로 소켓통신을 이용함으로써 AWS IoT와 MQTT를 대체하도록 한다.

3. 시험시나리오

< 그림 5 >는 시스템 수행 시나리오를 바탕으로 사용자 설정을 하는 시나리오와 온실 환경 설정 및 제어에 관한 시나리오이다.

사용자 설정 시나리오는 다음과 같다. 먼저 사용자가 온실에 식물을 심고 APP에서 제공하는 식물 목록 중에서 자신이 키우는 식물을 설정한다. 사용자가 APP에서 설정한 값은 서버에 전송되고 DB에 저장된다. 이후 사용자에게는 선택한 식물에 대한 식물의 정보를 제공하고, DB에 기록된 센서 값들을 시각적으로 가공하여 현재 상태, 온실 관리 기록 등에 대한 정보를 제공하도록 한다. 사용자가 APP을 통해 온실의 설정을 바꿀 경우에도 서버와 라즈베리파이를 거쳐 아두이노로 전송되고 급수, LED, 쿨링팬 등의 상태를 변경하게 된다. 스트리밍을 하는 경우 카메라를 통해 온실의 상황을 확인하고 원할 경우 사진 촬영을 할 수 있다.

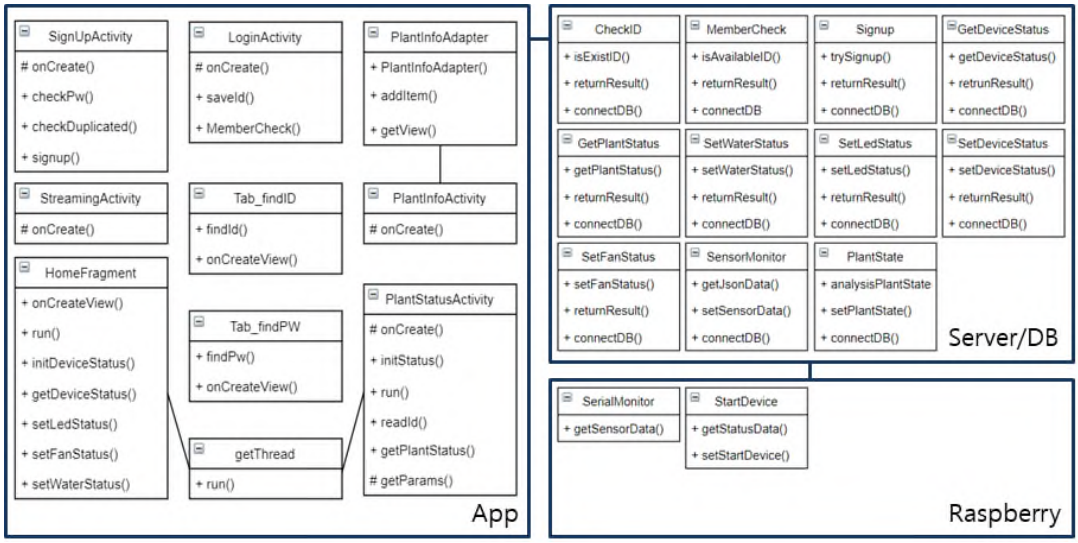
자동으로 온실 환경을 설정 및 제어하는 시나리오는 다음과 같다. 먼저 DB에 저장된 값들을 통해 현재 온실의 상태를 확인한다. 사용자가 설정한 식물에 대한 정보를 확인하고, 현재 온실의 상태와 사용자가 설정한 식물에 대한 정보를 비교하여 장치의 작동 여부를 결정한다. 장치 작동이 필요한 경우, 아두이노에 신호를 보내서 장치를 작동 시키고 작동 기록을 저장한다.



< 그림 5 > 시스템 수행 시나리오

4. 상세 설계

< 그림 6 >은 App, Server/DB, Raspberry의 클래스 다이어그램이다.



< 그림 6 > 클래스 다이어그램

< 표 6 >의 User는 회원가입을 할 경우 사용자들의 정보가 저장되는 테이블이고, Sensor는 온실의 센서 데이터들을 저장하는 테이블이다. Device 테이블은 현재 장치들의 동작상태를 저장한 테이블이고, UserPlant는 사용자가 기르기로 선택한 식물에 대한 정보를 저장하는 테이블이다.

< 표 6 > DB 테이블

User	
id	유저아이디
password	비밀번호
name	이름
phone	전화번호
regdate	가입날짜

Sensor	
id	유저아이디
temp	온도
humi	습도
soil1	토양습도1
soil2	토양습도2
soil3	토양습도3
Cds	조도
level	수위
Date	측정시간

Device	
number	번호
water1	워터펌프1
water2	워터펌프2
water3	워터펌프3
led	led on/off
Fan	fan on/off

UserPlant	
plantId	식물번호
Data	식물 데이터

4.1 아두이노에서의 기능

<그림 7 >은 아두이노에서 측정한 온도, 습도, 토양습도, 조도, 수위센서의 값을 Json 형식으로 변환하여 시리얼 모니터에 출력하는 makeJson 함수이다. < 그림 8 >은 라즈베리파이에서 전송한 문자열을 아두이노에서 시리얼을 통해 읽은 후 문자열이 'StartMotor1'일 경우에는 operateMotor 함수가 동작되도록 하여 워터펌프를 작동시키고 식물에 물을 공급한다. 문자열이 'StartLed'일 경우 operateLED 함수를 작동시켜 LED가 작동하도록 하고, 문자열이 'StartFan'일 경우 operateFan 함수를 작동시켜 쿨링팬을 작동시키도록 한다.

makeJson()	
형식	void makeJson(float temp, float humi, int soil1, int soil2, int soil3, int cds, int level)
리턴 값	Void
설명	아두이노에서 측정한 온도, 습도, 토양습도, 조도, 수위센서의 값을 JSON 형식으로 변환하여 시리얼 모니터에 출력
예시	makeJson(temp, humi, soil1, soil2, soil3, cds, level)

< 그림 7 > makeJson 함수

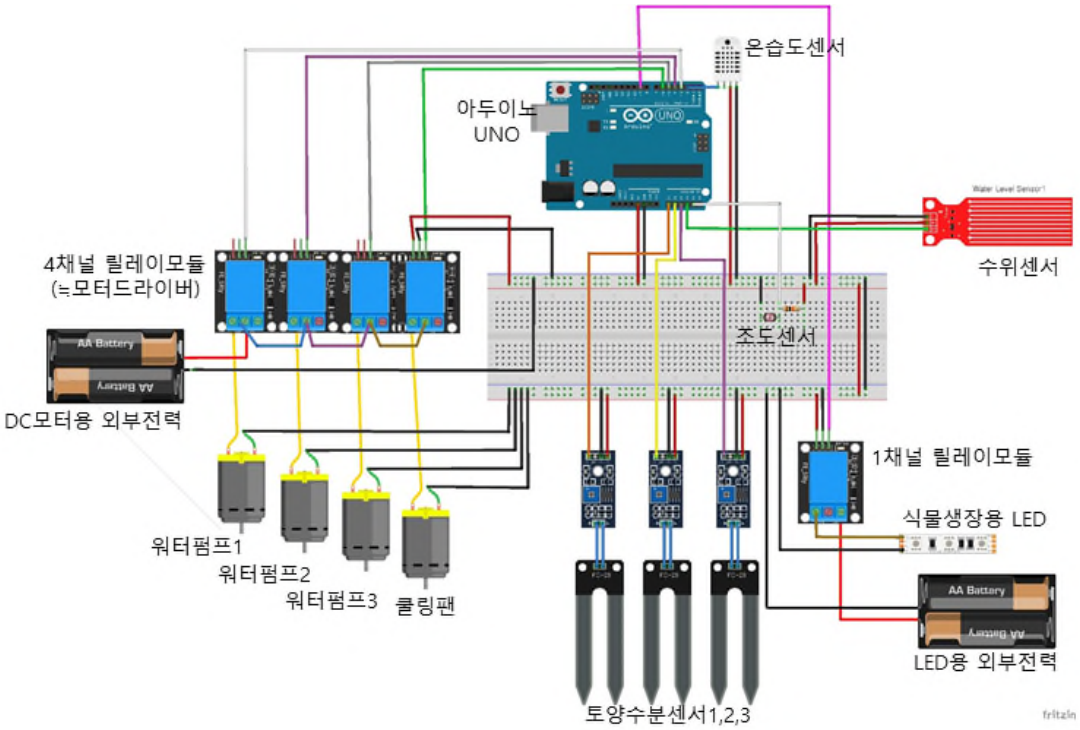
operateMotor()	
형식	void operateMotor(char motor)
리턴값	Void
설명	라즈베리파이에서 전송된 문자열에 따라 위치 별 워터펌프가 동작하여 식물에 물 공급
예시	if(Data == "StartWater1") { operateMotor('a'); }

operateLED()	
형식	void operateLED(int operatetime)
리턴값	Void
설명	라즈베리파이에서 전송된 문자열에 포함된 작동시간에 따라 식물 생장용 LED가 작동
예시	if(Data == "StartLed") { operateLed(3000); }

operateFan()	
형식	void operateFan(int operatetime)
리턴값	Void
설명	라즈베리파이에서 전송된 문자열에 포함된 작동시간에 따라 쿨링팬이 작동
예시	if(Data == "StartFan") { operateFan(3000); }

< 그림 8 > operateMotor, LED, Fan 함수

<그림 9>는 온습도센서, 토양수분센서 3개, 수위센서, 조도센서, 10KΩ 저항, 쿨링팬, 워터펌프 3개, 4채널 릴레이 모듈, 외부전력 2개, 식물 생장용 LED, 1채널 릴레이 모듈을 사용하여 구성한 아두이노 회로도이다. 온도와 습도를 측정하기 위한 온습도 센서는 디지털 2번 핀에 연결하고, 토양 수분을 측정하기 위한 토양수분센서 3개는 아날로그 A0, A1, A2번 핀에 연결한다. 물통의 수위를 측정하기 위한 수위센서는 아날로그 A4번 핀에 연결한다. 조도센서는 10KΩ 저항을 사용하여 측정 가능하게 하고 아날로그 A4번 핀에 연결한다. 쿨링팬과 워터펌프는 디지털 3, 4, 5, 6번 핀에 각각 연결하여 작동을 관리하는데, 전력소모가 크기 때문에 외부전력을 이용한다. 또한 모터의 작동 시에만 외부전력을 이용하기 위해 4채널 릴레이 모듈과 연결한다. 마찬가지로, 식물 생장용 LED는 디지털 9번 핀에 연결하고 외부전력과 1채널 릴레이모듈을 이용하여 작동시킨다.



< 그림 9 > 아두이노 회로도

4.2 라즈베리파이에서의 기능

라즈베리파이에서는 센서 값을 전송하는 모듈인 < 그림 10 >의 SerialMonitor 함수를 사용한다. JSON 형식으로 된 아두이노의 시리얼 데이터를 읽어들여 SensorValue 파일에 센서 데이터를 갱신하며 저장한다. 그리고 SensorValue 파일에서 읽어온 JSON 형식의 데이터를 10초에 한 번씩 서버로 전송한다.

또한, 장치 작동 모듈인 < 그림 11 >의 StartDevice 함수에서는 서버로부터 라즈베리파이에 전송된 문자열을 분석한다. water1, water2, water2, led, fan 데이터에 들어있는 문자열의 끝이 bad로 끝날 경우, 상황에 맞는 문자열을 아두이노로 전송하여 각 디바이스를 작동시킨다.

SerialMonitor
기능
1. JSON 형식으로 된 아두이노의 시리얼 데이터를 읽어들여 SensorValue 파일에 데이터를 갱신하며 저장 2. SensorValue 파일의 데이터를 서버로 전송
다루는 정보
1. Json_data : SensorValue 파일에서 읽어온 JSON 데이터의 저장 변수
반영되는 시점
10초에 한번씩 서버에 측정값을 전송

< 그림 10 > SerialMonitor 모듈

StartDevice
기능
1. 서버로부터 전송된 문자열을 분석한다. 2. 상황에 맞는 문자열을 아두이노로 전송하여 각 디바이스를 작동하게 한다.
다루는 정보
1. Data : 서버에서 전송된 문자열의 배열 2. water1, water2, water3, led, fan : 배열에 들은 각 문자열
반영되는 시점
water, led, fan 데이터에 들어있는 문자열의 끝이 bad로 끝날 경우 아두이노에 메시지를 전송하여 장치를 작동시킨다.

< 그림 11 > StartDevice 모듈

4.3 서버에서의 기능

서버에서는 센서 값을 저장하는 모듈인 < 그림 12 >의 SensorMonitor 함수를 사용한다. 라즈베리파이에서 10초에 한 번씩 전송된 JSON 형식의 센서 값을 파싱한다. 서버에서 DB와 연결을 하고 SQL 구문을 수행하여 파싱한 데이터를 DB에 저장되어 있는 temp, humi, soil1, soil2, soil3, cds, water에 각각 저장한다.

SensorMonitor
기능
1. 라즈베리파이에서 전송된 JSON 형식의 센서 값을 파싱하여 DB에 저장
다루는 정보
1. Json_data : JSON 데이터의 저장 변수 2. Temp, humi, soil1, soil2, soil3, cds, water : json_data를 파싱한 데이터의 저장 변수
반영되는 시점
10초에 한번씩 서버에 전송되는 JSON 데이터를 파싱하고, DB와의 연결 및 SQL문을 수행

< 그림 12 > SensorMonitor 모듈

또한, 식물 상태 분석 모듈인 <그림 13>의 PlantState 함수에서는 사용자가 키우고 있는 식물에 대한 기본 정보를 저장하고 있는 식물 DB와 센서 값 DB를 30분에 한 번씩 읽어 비교 및 분석한다. < 그림 14 >의 CurrentSeason 함수에서는 현재 달에 맞는 계절 코드를 확인한다. WaterCycle 함수에서는 현재 계절 코드와 토양수분센서 값을 비교분석하고, Humi 함수는 설정된 식물의 습도코드와 현재 습도를 비교하며, WaterLevel 함수에서는 현재 물통의 수위를 확인한다. 이 함수들을 통해 분석한 후, 식물의 현재 상태를 문자열로 리턴한다. 분석 결과인 리턴한 문자열을 DB에 업로드 하고, 라즈베리파이에 전송하여 값에 따라 아두이노 장치를 작동시킬 수 있다. 또한, 물통의 수위 값에 따라 기준을 초과할 시 App에서 사용자에게 알림을 주도록 한다.

PlantState
기능
1. 사용자가 키우고 있는 식물에 대한 데이터를 분석하여 현재 상태를 라즈베리파이로 전송한다. 2. 분석한 결과를 DB에 업로드 한다.
다루는 정보
1. Name, hdCode, watercycleSpringCode, watercycleSummerCode, watercycleAutumnCode, watercycleWinterCode : 사용자가 키우는 식물 이름, 습도 코드, 봄, 여름, 가을, 겨울 별 물주기 코드 2. Month, humi, soil1, soil2, soil3, level: 현재 날짜의 달, 온실 습도, 토양습도1, 2, 3, 물통의 물 수위
반영되는 시점
30분에 한번씩 식물 DB와 센서 값 DB의 값을 읽어 비교 분석한다. 분석결과를 DB에 저장하고, 라즈베리파이로 전송하여 값에 따라 아두이노 장치를 작동시킬 수 있다. 또한 물통의 수위 값에 따라 App에서 사용자에게 알림을 주게 할 수 있다.

< 그림 13 > PlantState 모듈

CurrentSeason()	
형식	def CurrentSeason(month)
리턴값	String
설명	현재 달에 맞는 계절 코드 확인
예시	If date == '02' or date == '01' or date == '03': return watercycleSpringCode

WaterCycle()	
형식	def WaterCycle(watercycleCode, soil)
리턴값	String
설명	현재 계절 코드와 토양수분센서 값을 비교 분석하여 문자열 리턴
예시	If watercycleCode == "053001" : if soil > 1000 : return "soilgood" else : return "soilbad"

Humi()	
형식	def Humi(hdCode, humi)
리턴값	String
설명	설정된 식물의 습도코드와 현재 습도를 비교한다.
예시	If hdCode == "083001" : #습도 40%이하로 유지 if humi < 40 : return "humigood" else : return "humibad"

WaterLevel()	
형식	def WaterLevel(level)
리턴값	String
설명	현재 물통의 수위를 확인하여 값을 리턴한다.
예시	If level < 100 : return "waterlevelbad" else : return "waterlevelgood"

< 그림 14 > CurrentSeason, WaterCycle, Humi, WaterLevel 함수

4.4 앱에서의 기능

4.4.1 로그인 기능

로그인을 처리하는 모듈인 < 그림 15 >의 Login 모듈을 통해 아이디와 비밀번호를 입력 후, DB에 저장된 값과 일치한다면 로그인이 가능하다. 자동로그인 체크박스 활성화 이후 어플리케이션을 재실행 하면 로그인 절차 없이 자동으로 로그인이 가능하다. EditText로 입력된 아이디와 비밀번호를 확인하고 CheckBox로 로그인 유지 여부를 확인한다. 사용자가 DB에 저장된 아이디와 패스워드를 입력했거나 자동로그인 체크박스 적용 시 반영된다.

Login 모듈
기능
1. 자동로그인 체크박스 활성화 이후 어플리케이션 재실행 시, 로그인 여부 확인 절차 없이 자동 로그인 가능 2. 아이디와 비밀번호 확인 후, DB에 저장된 값과 일치한다면 로그인
다루는 정보
1. EditText userId, userPassword : 입력된 아이디와 비밀번호 2. CheckBox autoLogin : 로그인 유지 여부 체크박스 3. TextView loginMessage : 아이디 패스워드 확인 텍스트뷰
1. 반영되는 시점
1. 사용자가 DB에 저장된 아이디와 패스워드를 입력하였을 경우 2. 자동로그인 체크박스 적용 시

< 그림 15 > Login 모듈

< 그림 16 >의 MemberCheck() 함수로 사용자가 입력한 아이디와 패스워드를 확인하고, DB에 저장된 값과 일치한 경우 saveId() 함수로 사용자가 입력한 아이디를 txt파일에 저장한다.

saveId()		MemberCheck()	
형식	void saveId(String inputId)	형식	Void MemberCheck(final String id, final String password)
리턴값	Void	리턴값	Void
설명	사용자가 입력한 아이디를 txt파일에 저장	설명	사용자가 입력한 아이디와 패스워드를 확인
예시	saveId(id)	예시	MemberCheck(txtId, txtPassword)

< 그림 16 > saveId, MemberCheck 함수

4.4.2 회원가입 기능

회원가입을 처리하는 모듈인 그림 < 그림 17 >의 SignUp 모듈을 통해 EditText로 각각의 가입 항목을 사용자로부터 입력받는다. 사용자가 입력한 아이디가 중복되는지 DB에 저장된 값들과 비교해 중복 여부를 확인한다. 가입 항목이 입력 양식에 맞는지, 비밀번호와 비밀번호 확인이 일치하는지에 대한 알림을 TextView에 출력한다. 사용자가 가입항목을 모두 양식에 맞게 작성하여 회원가입이 완료되면 사용자가 입력한 정보들을 DB에 저장한다.

SignUp 모듈
기능
1. 사용자가 입력한 아이디가 중복되는지 DB에 탐색 2. 사용자가 입력한 회원가입 정보를 DB에 저장 3. 중복 확인 결과 및 회원 가입 가능 여부를 사용자에게 알림
다루는 정보
1. EditText inputId, inputPassword, checkPassword, inputName, inputPhone : 입력된 가입 항목 (아이디, 비밀번호, 비밀번호 확인, 이름, 전화번호) 2. TextView idMsg, passwordMsg, checkPasswordMsg, nameMsg, phoneMsg : 입력된 가입 항목에 대한 알림 텍스트뷰 3. boolean isCheckID, isCheckPw : 입력한 아이디 확인, 입력한 비밀번호 확인 4. boolean result : 아이디와 비밀번호 확인 결과, 가입 항목 DB 저장 결과
반영되는 시점
사용자가 가입 항목을 양식에 맞게 작성하였을 경우 INSERT 쿼리문을 수행해 user 테이블에 데이터를 저장

< 그림 17 > SignUp 모듈

< 그림 18 >의 checkDuplicated() 함수로 사용자가 입력한 아이디가 기존 DB에 존재하는지의 여부를 체크한다. 회원가입 완료시, < 그림 19 >의 signup() 함수를 통해 사용자가 입력한 가입 정보를 DB에 저장한다. signup() 함수의 데이터구조는 String 타입의 id, password, name, phone 이다. 이는 사용자가 입력한 아이디, 비밀번호, 이름, 전화번호를 뜻한다.

checkDuplicated()	
형식	void checkDuplicated(final String id)
리턴값	Void
설명	사용자가 입력한 아이디가 기존 DB에 존재하는지 여부 체크
예시	checkDuplicated(txtId)

< 그림 18 > checkDuplicated 함수

signup()	
형식	void signup(final String id, final String password, final String name, final String phone)
리턴값	Void
설명	사용자가 입력한 가입 정보를 DB에 저장
예시	signup(txtId, txtPw, txtName, txtPhone)

signup() – 데이터 구조		
데이터	자료형	설명
id	String	사용자가 입력한 아이디
password	String	사용자가 입력한 비밀번호
name	String	사용자가 입력한 이름
phone	String	사용자가 입력한 전화번호

< 그림 19 > signup 함수와 데이터구조

4.4.3 온실상태확인 기능

현재 온실의 상태(온도, 습도, 토양습도, 조도, 물탱크, 수위 센서 값)를 서버에서 수신받아 확인하는 기능을 하는 < 그림 20 >의 온실 상태 확인 모듈을 통해 TextView로 각

센서의 값을 출력한다. 이를 위해 Volley API를 사용하여 웹서버와 HTTP 통신이 가능하여야 한다.

온실 상태 확인 모듈
기능
1. 현재 온실의 상태(온도, 습도, 토양습도, 조도, 물탱크 수위 센서값)를 서버에서 수신받은 온실 상태를 화면에 출력
다루는 정보
1. TextView textTemperature, textHumidity, textSoil1, textSoil2, textSoil3, textWater,textCds,textDate : 각 센서의 값을 출력하는 텍스트뷰
반영되는 시점
Volley API를 사용하여 웹서버와 HTTP 통신이 가능한 경우

< 그림 20 > 온실 상태 확인 모듈

< 그림 21 >의 readId() 함수로 id.txt 파일에서 id 값을 읽어오고 읽어온 id값을 이용해서 getPlantStatus() 함수로 아두이노 센서의 온도, 습도, 토양습도, 조도, 물탱크, 수위 센서 값을 받아온다.

getPlantStatus()	
형식	void getPlantStatus(final String id)
리턴 값	void
설명	아두이노 센서의 온도, 습도, 토양습도, 조도, 물탱크 수위 센서값을 받아오는 함수
예시	getPlantStatus(readId)

readId()	
형식	String readId()
리턴 값	void
설명	Id.txt파일에서 id 값을 읽어오는 함수
예시	readId()

< 그림 21 > getPlantStatus, readId 함수

4.4.4 장치조절 기능

현재 장치 상태 확인과 장치 조절을 처리하는 < 그림 22 >의 장치 조절 모듈을 통해 실시간으로 쿨링팬, 물펌프, LED의 현재 장치 상태를 DB로부터 가져와 ToggleButton의 ON, OFF 형태로 화면에 출력한다. 버튼을 통해 사용자가 장치를 직접 ON, OFF할 수 있다. 버튼의 상태 변화에 따른 ON, OFF 문자열 데이터가 서버로 전달되고, 장치 상태 DB에 저장되어 갱신된다. Volley API를 사용하여 웹서버와 HTTP 통신이 가능하여야 한다.

장치 조절 모듈
기능
1. 현재 장치(쿨링팬, 물펌프, LED)의 상태를 서버로부터 수신 받아 화면에 토글버튼으로 출력 2. 토글버튼의 상태를 서버에 전송하여 장치를 켜거나 끔
다루는 정보
1. ToggleButton toggleButton1, toggleButton2, toggleButton3 : 각 장치들의 상태를 ON, OFF로 출력하는 버튼
반영되는 시점
Volley API를 사용하여 웹서버와 HTTP 통신이 가능한 경우

< 그림 22 > 장치 조절 모듈

< 그림 23 >의 InitDeviceStatus() 함수로 각 장치들의 상태를 DB로부터 전송받아 초기 ToggleButton 상태를 설정한다. getDeviceStatus() 함수를 통해 각 장치들의 상태를 DB로부터 수신한다. SetLedStatus(), SetFanStatus(), SetWaterStatus() 함수를 통해 사용자가 변경한 각각의 LED, 쿨링팬, 워터펌프의 상태 값을 DB로 전송한다.

InitDeviceStatus()	
형식	void initDeviceStatus(final String id)
리턴값	Void
설명	각 장치들의 상태를 DB로부터 전송받아 초기 토글버튼에 상태 설정
예시	initDeviceStatus(finallD)

getDeviceStatus()	
형식	void getDeviceStatus(final String id)
리턴값	Void
설명	각 장치들의 상태를 DB로부터 수신
예시	getDeviceStatus(finallD)

SetLedStatus()	
형식	void setLedStatus(final String id, final String led)
리턴값	Void
설명	사용자가 변경한 LED 상태 값을 DB로 전송
예시	setLedStatus(finallD, “on”)

SetFanStatus()	
형식	void setFanStatus(final String id, final String led)
리턴값	Void
설명	사용자가 변경한 쿨링팬 상태 값을 DB로 전송
예시	setFanStatus(finallD, “on”)

SetWaterStatus()	
형식	void setWaterStatus(final String id, final String led)
리턴값	Void
설명	사용자가 변경한 워터펌프 상태 값을 DB로 전송
예시	setWaterStatus(finallD, “on”)

< 그림 23 > InitDeviceStatus, getDeviceStatus, SetLedStatus, SetFanStatus, SetWaterStatus 함수

4.4.5 식물 리스트 조회 기능

농사로 오픈 API를 이용해 다양한 식물 목록과 그에 따른 세부 정보를 조회하는 기능을 하는 < 그림 24 >의 식물 리스트 조회 모듈을 통해 ArrayList<PlantInfoItem>로 식물 정보를 받아와 배열에 저장한다. 받아온 식물 정보를 PlantInfoAdapter를 이용해서 리스트 뷰에 출력한다. ListView로 식물 정보를 출력하고 TextView로 식물의 상세정보를 출력한다. 이를 위해 API가 제공하는 XML 문서를 파싱 가능해야 한다.

식물 리스트 조회 모듈
기능
1. 오픈 API를 이용해 다양한 식물 목록과 그에 따른 세부 정보를 조회
다루는 정보
1. ArrayList<PlantInfoItem> arrayList : 식물 정보를 받아와 저장하는 배열 2. PlantInfoAdapter adapter : 받아온 식물 정보를 리스트뷰에 출력해주는 어댑터 3. ListView listView : 식물 정보를 출력할 리스트뷰 4. TextView txtName, txtCode, txtDetail; : 식물의 이름, 코드, 상세정보를 출력하는 텍스트뷰
반영되는 시점
API가 제공하는 XML문서를 파싱 가능한 경우

< 그림 24 > 식물 리스트 조회 모듈

4.4.6 스트리밍 기능

온실 영상을 출력해 사용자가 실시간으로 온실 식물의 상태를 확인할 수 있게 하는 < 그림 25 >의 Streaming 모듈을 통해 WebView에서 uv4l 스트리밍 서버와 통신하고 온실에 부착된 라즈베리파이 파이카메라에서 촬영되는 영상을 출력한다. webSettings를 통해 WebView의 상태를 설정할 수 있다.

Streaming 모듈
기능
1. 파이카메라에서 웹으로 출력하는 영상을 웹뷰로 출력
다루는 정보
1. webView webview : StreamingActivity의 웹뷰 객체 2. webSettings webSettings : 웹뷰 설정
반영되는 시점
웹뷰에서 스트리밍 서버와 통신이 가능할 경우

< 그림 25 > Streaming 모듈

5. Prototype 구현

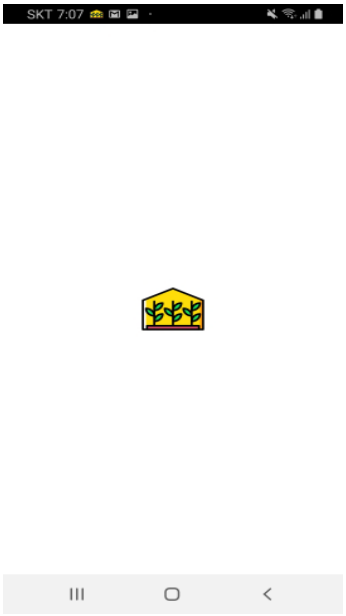
5.1 구현 현황

구현 범위	구현 현황	구현한 부분	문제점 및 해결방안
전체	80%		
아두이노	80%	1. 센서 회로 연결 및 값 측정 2. 센서 값을 JSON 형태로 라즈베리파이에 전송 3. 라즈베리파이로부터 받은 문자열에 따라 작동 4. 온실 제작 5. 하드웨어 부착	1.식물 생장용 led 작동 오류 -> 라즈베리파이의 문제인지 아두이노의 문제인지 분석 후, 코드 수정
라즈베리파이	90%	1. UV4L 스트리밍 서버 구현 2. 센서 값 서버로 전송 3. 서버에서부터 받은 데이터 아두이노로 전송	
서버/DB	80%	1. 센서 데이터 파싱 및 DB 업로드 2. JSP를 이용한 앱과 DB 연동 3. 현재 장치 상태 확인 및 제어 4. 장치 예약 기능 활성화 5. 식물 데이터 분석 후 장치 작동	1. DB 용량 부족 문제 -> DB에 쌓이는 데이터 중 시간이 지나 불필요해진 데이터 제거할 예정
APP	80%	1. DB연동을 통해 회원 가입, 로그인 기능 구현 2. 온실 영상 스트리밍 3. DB에 저장된 센서 값 불러와 표시 4. 키우는 식물 설정 및 Open api로부터 식물 데이터 파싱하여 정보 제공 5. 각 식물 별 장치(쿨링팬, 워터펌프, LED) 작동 설정 6. 센서별 주간, 월간 그래프 출력 7. 사용자가 장치 작동 시간 설정하여 매일 해당 시간에 작동	1. 스트리밍 영상 캡쳐 후 저장 -> 화면의 부분만 캡쳐하는 방식 고려 2. 홈화면 동기화 문제 -> 스레드를 종료시키는 방법 고려

5.2 Application 스토리보드

5.2.1 인트로

<그림 26 >은 Application 이름이 나오고 서비스를 잘 표현한 인트로 이미지가 표시된다. 3초 후에는 자동으로 로그인 화면으로 이동한다.



< 그림 26 >

5.2.2 로그인

< 그림 27 >은 로그인 화면이다. Application의 이름이 표시되고, 회원인 경우 아이디와 비밀번호를 입력 후 로그인 버튼을 클릭하면 홈화면으로 이동한다. 로그인 상태 유지를 체크하면 로그인 상태를 유지시킬 수 있다.

< 그림 27 >

올바르지 않은 아이디와 비밀번호를 입력 시 < 그림 28 >과 같이 가입되지 않은 계정이라는 경고 메시지가 출력된다.

< 그림 28 >

5.2.3 회원 가입 화면 구성

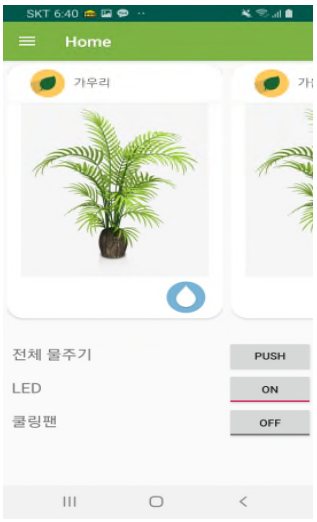
< 그림 28 >은 회원 가입 화면이다. 회원 가입 시 아이디 중복 검사 후 사용 가능한 아이디일 경우 가입이 가능하다. 비밀번호는 4~10자 이내 영문자, 숫자 조합만 사용할 수 있다. 형식에 맞게 정보들을 입력하고 회원가입 버튼을 클릭하면 회원 가입에 성공한다. 잘못된 형식이 하나라도 있을 경우 회원가입을 할 수 없다.

< 그림 29 >

5.2.4 홈화면 구성

< 그림 30 >은 로그인에 성공하면 처음으로 뜨는 홈화면 이다. 저장된 사용자의 식물을 리스트로 출력하고, 온실 장치들의 상태를 토글버튼으로 출력한다. 전체 물주기 버튼을 클릭하면 등록된 식물 전체에 한 번에 물을 줄 수 있으며, LED와 쿨링팬 버튼을 이용하여 장치를 켜거나 끌 수 있다. 식물별로 다르게 물을 주고 싶다면 식물 리스트에 있는 물방울 이미지 버튼을 클릭하면 줄 수 있다. 식물 Item을 클릭하면 해당 식물의 자세한 정보를 출력한다. 식물 Item을 길게 눌러 Swipe하면 식물의 위치를 이동할 수 있다. 다른 화면으로 이동할 때는 상단에 네비게이션 바를 클릭하여 여러 기능으로 이동할 수 있는 목록을 출력한다.

라즈베리파이로부터 문자열을 읽어 들인 후, 문자열에 따라 해당하는 장치를 작동시킨다. 장치를 작동시키면서 디바이스의 작동 상태를 문자열로 표현하여 라즈베리파이에 전송한다.

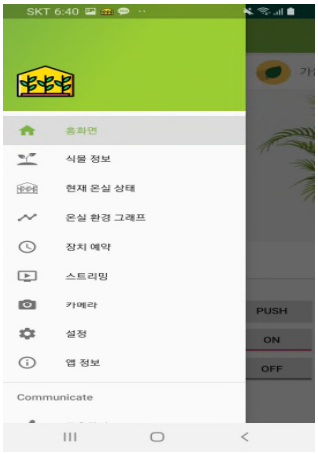


< 그림 30 >

< 그림 31 >

5.2.5 네비게이션 바 구성

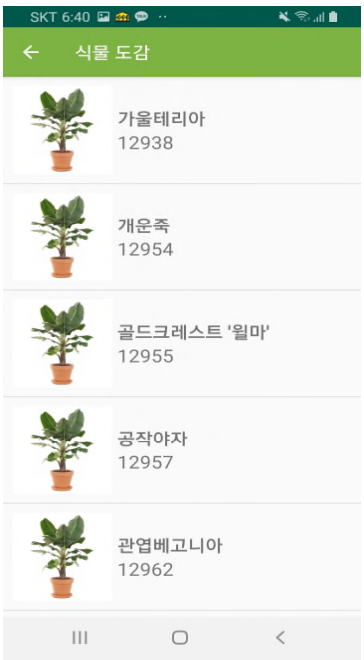
< 그림 32 >은 네비게이션 바 화면이다. 각 메뉴를 클릭하면 홈화면, 식물 정보, 현재 온실 상태, 온실 환경 그래프, 장치 예약, 스트리밍, 카메라, 설정, 메일보내기 화면으로 이동한다.



< 그림 32 >

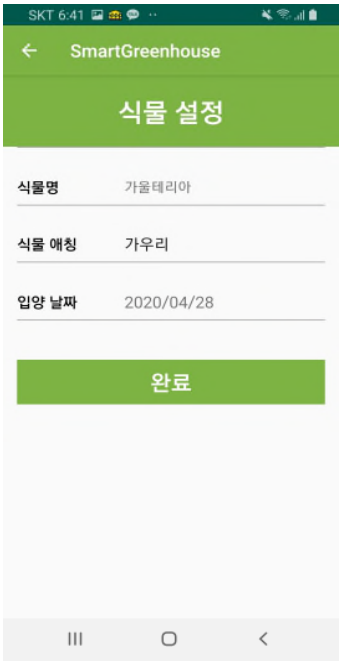
5.2.6 식물 정보 화면 구성

< 그림 33 >는 식물 정보 화면을 구성하는 페이지이다. Open API를 이용하여 식물을 불러온다. 식물 리스트를 클릭하면 해당 식물의 정보 화면으로 이동한다. < 그림 34 > 식물 정보 화면에서 선택 버튼을 클릭하면 해당 식물을 설정할 수 있는 식물 설정 화면으로 이동한다. < 그림 35 > 식물 설정 화면에서 사용자 식물을 설정하여 저장한다.



< 그림 33 >

< 그림 34 >



< 그림 35 >

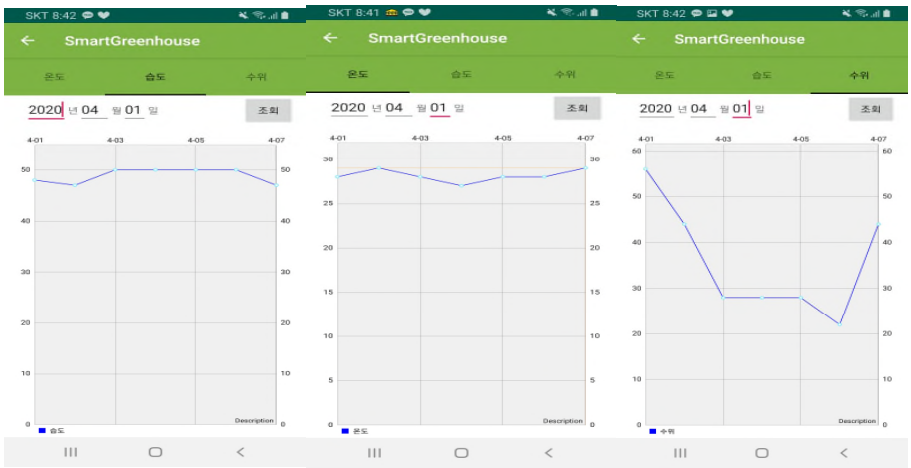
5.2.7 현재 온실 상태 화면 구성

< 그림 36 >는 센서 데이터를 불러와 현재 온실 상태를 확인하는 페이지이다. 현재 온실의 온도, 습도, 토양습도 1, 2, 3 물탱크, 물 양, 조도 데이터를 출력한다.

< 그림 36 >

5.2.8 온실 환경 그래프 화면 구성

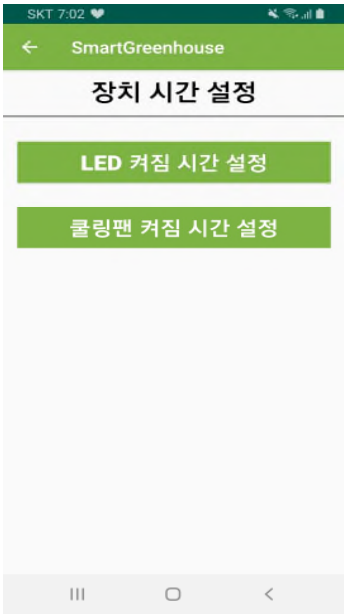
< 그림 37 >은 온실 환경 그래프를 출력하는 화면이다. 사용자가 출력하고자하는 센서를 선택하고 조회할 날짜를 입력하면 해당 센서의 일간 그래프가 출력된다.



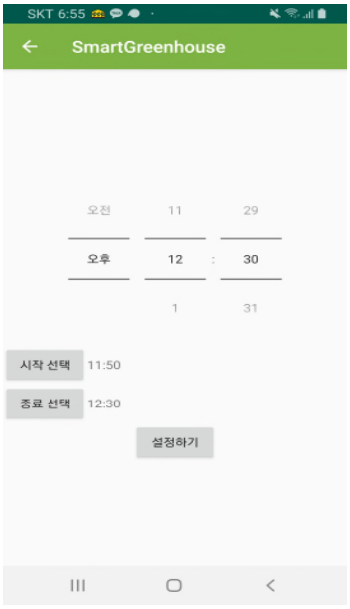
< 그림 37 >

5.2.9 장치 예약 화면 구성

< 그림 38 >은 시간 설정 장치를 선택하는 화면이다. 시간 설정을 하고자하는 장치를 선택하면 < 그림 39 > 시간 설정 페이지로 이동한다. 시작 시간과 종료 시간을 선택하고 설정하기를 누르면 해당 시간에 장치가 작동하도록 설정된다.



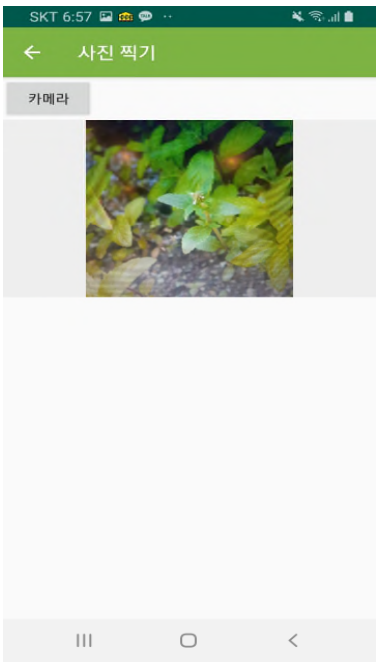
< 그림 38 >



< 그림 39 >

5.2.10 스트리밍 화면 구성

< 그림 40 >은 스트리밍 화면이다. 온실에 부착된 카메라를 통해 온실의 실시간 영상을 스트리밍할 수 있다.



< 그림 40 >

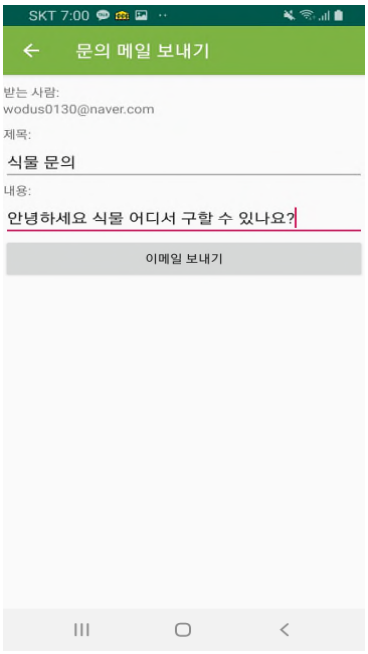
5.2.11 설정 화면 구성

< 그림 41 >은 설정 화면이다. 로그아웃을 클릭하면 접속되어 있던 계정이 로그아웃 되고, 로그인 페이지로 이동한다.

< 그림 41 >

5.2.12 메일 보내기 화면 구성

< 그림 42 >은 메일 보내기 화면이다. 받는 사람으로 관리자의 이메일 주소가 표시되고, 사용자는 보낼 메일의 제목과 내용을 입력할 수 있다. 입력 후 이메일 보내기 버튼을 클릭하면 사용자가 메일을 보낼 Application을 선택할 수 있다. 선택한 Application을 통해 메일이 관리자에게 전송된다.

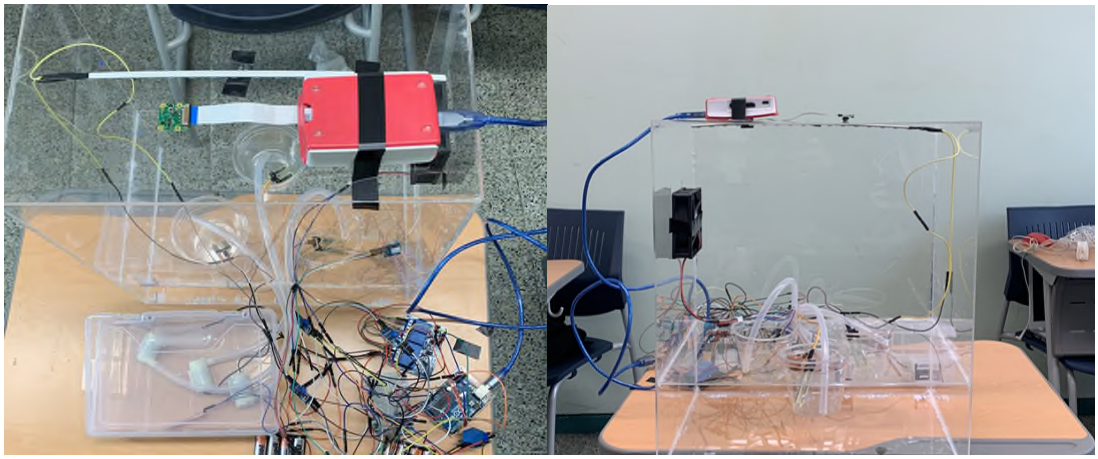


< 그림 42 >

5.3 하드웨어

5.3.1 하드웨어

< 그림 43 >은 하드웨어 및 스마트 온실 사진이다. 아크릴판을 이용하여 35*25*40cm 사이즈로 제작한 온실 뒷면에 7*3cm의 작은 구멍을 뚫어서 깔끔한 온실 상태를 위해 워터펌프와 연결된 호스 3개, 온습도 센서, 조도 센서, 토양 수분 센서를 온실 내부에 넣고 온실 뒷쪽에 아두이노, 물탱크, 물탱크 안에 워터펌프와 수위센서를 놔둔다. 아두이노와 연결된 라즈베리파이는 pi 카메라를 부착하고 있으므로 온실 위쪽에 고정시켜 카메마를 통해 촬영할 수 있게 한다. 그리고 온실 내부의 천장 부분에 led를 부착한다. 온실 내부 왼쪽 벽면에는 8*8cm의 정사각형 구멍을 뚫고 쿨링팬을 부착하여 온실 외부와 환기가 가능하게 한다. 또한, 온실 내부 바닥에서부터 10cm 위에 3개의 원형 구멍이 뚫린 아크릴판을 부착하고, 각각의 원형 구멍에 화분을 놓는다. 온실 바닥에는 물받이를 두어 화분에서 나오는 물을 받을 수 있게 한다.

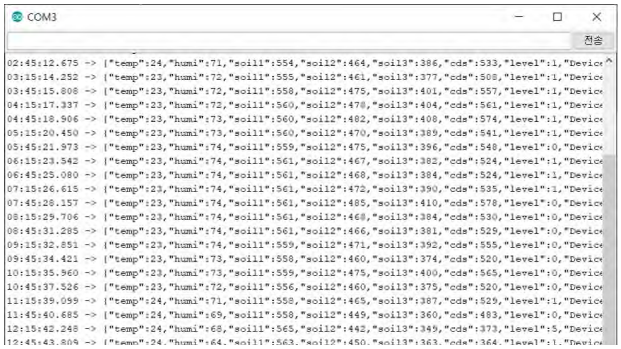


< 그림 43 >

5.3.2 아두이노

라즈베리파이로부터 문자열을 읽어 들이고 분석한 후, 작동을 멈추라는 신호를 받기 전까지 문자열에 따라 해당하는 함수를 실행해서 장치를 작동시킨다. 장치를 작동시키면서 디바이스의 작동 상태를 문자열로 표현하여 라즈베리파이에 전송한다.

또한, 아두이노와 연결된 센서들의 값을 각각 읽어 들인 후, 해당 값을 디바이스의 작동 상태 문자열과 함께 라즈베리파이에 전송한다. < 그림 44 >는 라즈베리파이에 전송할 센서 값과 디바이스 작동 상태를 시리얼 모니터로 출력한 화면이다.



< 그림 44 >

5.4 서버

서버에서는 5.2에서 설명한 Application의 기능들 중 홈화면, 온실상태화면 구성, 온실 환경 그래프 및 LED와 쿨링팬의 장치 작동 시간 예약 기능이 수행될 수 있도록 한다. 하드웨어 장치들로부터 얻은 센서 데이터들을 일정한 간격으로 DB에 날짜 데이터와 함께 저장하도록 하여 온실상태화면 구성을 위한 데이터를 제공하고, 그래프를 작성할 수 있도록 한다. 또한 현재 장치들의 ON, OFF 상황을 기록하여 홈화면에서 사용자가 확인하고 작동 상태를 변경할 수 있도록 하고 LED와 쿨링팬의 예약 상태를 확인하여 사용자가 지정한 시간대에 LED와 쿨링팬을 작동시킨다. 이외에도 사용자가 선택한 식물 데이터를 이용해 현재 상태와 비교분석 후 워터펌프, 쿨링팬 등의 장치들을 작동시켜 적정한 상태를 유지할 수 있도록 한다.

5.5 정리 및 분석

회원가입, 로그인 등의 기본 기능들은 구현을 완료하였다. 스마트 온실의 핵심 기능인 온실 환경을 원격으로 제어하고 사용자가 직접 장치들을 설정하여 온실 환경을 지속적으로 원하는 방향으로 맞추는 기능, 스트리밍 기능들을 구현하였다. 현재 온실의 상태를 센서 값을 통해 사용자가 확인할 수 있도록 하였으며, DB에 저장된 센서 값을 바탕으로 주간, 월간 그래프를 출력하는 기능을 추가하였다. 또한, OpenAPI에서 여러 식물 정보를 받아오고 이것을 출력하고 사용자가 선택한 식물은 사용자 식물 DB에 저장되도록 하였으나, OpenAPI에서 사진까지 가져와서 저장하는 것에는 문제가 있어 보완이 필요하다.

서버에서는 실시간으로 사용자 설정과 식물 상태 분석을 바탕으로 하드웨어를 작동시키고 상태를 업로드하기까지 구현하였다. 하드웨어는 설계대로 모든 장치의 연결을 진행했다. 장치를 각각 작동시킬 때는 문제가 없지만 한 번에 여러 개가 작동할 경우 불안정함이 남아있어 보완이 필요하다. 각 센서 값이 어느 정도의 상태를 가리키는지에 대해 실험을 통해 알아보았고, 더 정확한 데이터를 만들기 위해서는 몇 차례의 추가 실험이 필요하다.

5.6 향후 계획

Application에서 OpenAPI에서 사진을 받아오는 기능이 구현되지 않은 문제를 해결할 계획이다. 또한, 전체적인 디자인을 다듬고 기능적인 측면에서는 게시판 기능, 식물을 구매할 수 있는 곳을 지도로 보여주는 기능 등을 추가할 예정이다.

DB에 계속해서 쌓이는 센서 값들 중에서 일정 시간이 지남에 따라 불필요해진 데이터를 삭제하여 DB의 용량 부족 문제를 해결할 계획이다. 또한, 식물 생장용 led가 작동하지 않는 오류를 코드 수정을 통해 해결할 예정이다. 현재 센서 데이터가 저장되어 있는 DB에 불필요한 값이 계속 들어가고 있어서 원인을 찾고 수정할 것이다.

6. 시험/ 테스트 결과

장치가 사용자의 설정과 온실 상태에 따라 정확하게 작동하는지 확인하고자 앱에서 자동 급수 동작까지 걸린 시간과 급수량을 측정하였고, 온실 자동 관리 상태에서 식물의 종류 별로 정해진 동작을 수행하는지에 대한 실험을 진행하였다.

< 표 8 >에서는 앱에서 '물주기' 버튼을 클릭했을 때 자동 급수 장치가 동작할 때까지 걸린 시간과 한 번에 공급되는 물의 양을 측정한 실험 결과를 나타낸 표이다. 실험을 8회 실시하였을 때 앱에서 온실의 자동 급수까지 평균 2.4초가 걸렸고, 평균 급수량은 100ml이고 오차범위는 ±10ml를 넘지 않았다.

< 표 8 > 급수 시간과 급수량

Number	Time (second)	Water Suply (ml)
1	2	98
2	2.6	107
3	2.2	98
4	2.4	95
5	2.9	103
6	2.3	100
7	2.2	93
8	2.6	100

온실의 상태와 식물의 종류에 따라 자동 급수가 진행되는지에 대한 실험은 온도와 습도가 일정하게 유지되는 공간에서 진행되었다. 실험에 사용된 식물은 칼랑코에와 히포에스테스이고 육묘에 있어서 적정 환경은 농사로의 OpenAPI에서 제공하는 내용으로 각각 < 표 9 >와 < 표 10 >으로 정리하였다. 적절한 토양 수분량은 문장으로 제공되어 개발 단계에서 기준을 정하여 시스템을 설계했다. 토양 표면이 말랐을 때는 토양 수분도가 40% 이하가 되었을 경우 관수하고, 토양을 촉촉하게 유지하는 상황에서는 토양 수분도 70% 이하일 경우 관수하도록 하였다.

< 표 9 > 칼랑코에 생장 환경

Name	Conditions
temperature	23~26 ℃
humidity	70 %
water supply	Sufficient irrigation when the soil surface is dry

< 표 10 > 히포에스테스 생장 환경

Name	Conditions
temperature	21~25 ℃
humidity	70 %
water supply	Keep the soil moist. Be careful not to submerge in water.

온실의 온도와 습도가 칼랑코에와 히포에스테스에 적정한 상태로 유지된 상태에서 실험을 진행하였다. 칼랑코에는 9월 5일 48%의 수분량으로 시작하여 9월 14일에 51%의 수분량을 기록하였고 히포에스테스는 9월 5일 76%의 수분량으로 시작하여 9월 14일에 75%의 수분량을 기록하였다. 약 10일간 식물별 토양 수분도를 기록한 결과 두 식물 모두 적절한 토양 수분도를 유지되고 있는 것을 통해 자동 온실 관리 모드가 사용자가 설정한 식물 데이터를 토대로 제대로 작동하는 것을 확인하였다. < 표 11 >은 측정된 온도, 습도, 토양 습도를 나타내는 표이다.

< 표 11 > 날짜별 측정 온도, 습도, 토양 습도

Date	Tempera-ture (℃)	Humidity (%)	Kalanchoe Soil Moisture (%)	Hypoestes Soil Moisture (%)
9/5	24	78	48	76
9/6	24	75	41	70
9/7	25	78	39	81
9/8	24	75	52	75
9/9	24	72	46	71
9/10	23	76	42	83
9/11	24	76	37	77
9/12	23	73	53	73
9/13	24	75	50	83
9/14	24	72	51	75

7. Coding & DEMO

- 서버 소스코드

```
import pymysql
import json
import datetime
from datetime import timedelta
import time
import schedule
from apscheduler.jobstores.base import JobLookupError
from apscheduler.schedulers.background import BackgroundScheduler
from socket import error as SocketError
import errno
import sys

HOST = ''
PORT = 10001
BUFSIZE = 1024
ADDR = (HOST, PORT)
s1='null'
s2='null'
s3='null'
l='null'
h='null'
conn2 = pymysql.connect(host='database-1.cudf3zzu3npf.us-east-2.rds.amazonaws.com', user='jeongmin', password='97shwjdals!', db='mydb',charset='utf8')

# 소켓 생성
serverSocket = socket(AF_INET, SOCK_STREAM)
# 소켓 주소 정보 할당
serverSocket.bind(ADDR)
print('bind')
# 연결 수신 대기 상태
serverSocket.listen(100)
print('listen')
# 연결 수락
clientSocekt, addr_info = serverSocket.accept()
print('accept')
print('--client information--')
print(clientSocekt)

def RecvData():
    try:
        data = clientSocekt.recv(BUFSIZE)
        Ard_data = data.decode()
        json_data = json.loads(Ard_data)
        DeviceState = json_data['DeviceState']
        userid = json_data['id']
        print('디바이스 상태', DeviceState,userid)
        if DeviceState != "null":
            DeviceStateUpload(DeviceState,userid)
        return json_data
    except:
        pass

def UserSettingCheck(): #사용자 설정 확인 후 장치 작동
    try:
        #사용자 장치 설정 값
        curs = conn.cursor()
        sql = "select * from device where number = 1"
        curs.execute(sql)
        row = curs.fetchone()
        user_s1 = row[1]
        user_s2 = row[2]
        user_s3 = row[3]
        user_l = row[4]
        user_h = row[5]
        print("사용자 설정 값:",user_s1,user_s2,user_s3,user_l,user_h)
        #현재 장치 상태 값
        sql = "select * from device where number = 2"
        curs.execute(sql)
        row = curs.fetchone()
        cur_s1 = row[1]
        cur_s2 = row[2]
        cur_s3 = row[3]
        cur_l = row[4]
        cur_h = row[5]
        print("현재 상태 값:",cur_s1,cur_s2,cur_s3,cur_l,cur_h)
        #LED, FAN 예약 확인, 사용자 예약이 없으면 기본값
        sql = "select * from led order by number desc limit 1"
        curs.execute(sql)
        row = curs.fetchone()
        Start_led = row[1]
        End_led = row[2]

        sql = "select * from fan order by number desc limit 1"
        curs.execute(sql)
        row = curs.fetchone()
        Start_fan = row[1]
        End_fan = row[2]
        Current = datetime.datetime.now()+timedelta(hours=9) #서버가 오하이오주라서 9시간
        CurrentTime = Current.strftime('%H:%M') #str형식으로 변환
        print(Start_led, End_led, Start_fan, End_fan, CurrentTime)
```

```
    global s1,s2,s3,l,h
    if (user_s1 == 'on'):
        s1 = "StartMotor1"
        sql = "update device set water1 = 'user' where number=3"
    if (user_s2 == 'on'):
        s2 = "StartMotor2"
        sql = "update device set water2 = 'user' where number=3"
        curs.execute(sql)
    if (user_s3 == 'on'):
        s3 = "StartMotor3"
        sql = "update device set water3 = 'user' where number=3"
        curs.execute(sql)
    if (user_l == 'on'):
        l = "StartLed"
        sql = "update device set led = 'user' where number=3"
        curs.execute(sql)
    if (user_l == 'off'):
        l = "StopLed"
        sql = "update device set led = 'user' where number=3"
        curs.execute(sql)
    if (user_h == 'on'):
        h = "StartFan"
        sql = "update device set fan = 'user' where number=3"
        curs.execute(sql)
    if (user_h == 'off'):
        h = "StopFan"
        sql = "update device set fan = 'user' where number=3"
        curs.execute(sql)
    conn.commit()

    #사용자 설정과 현재 작동상황이 같으면 명령x
    if (user_h == 'on' and cur_h =='on'):
        h = 'null'
    if (user_l == 'on' and cur_l == 'on'):
        l = 'null'
    if (user_h == 'off' and cur_h =='off'):
        h = 'null'
    if (user_l == 'off' and cur_l == 'off'):
        l = 'null'
    if (Start_led == CurrentTime):
        l = "StartLed"
        sql = "update device set led = 'on' where number=1"
        curs.execute(sql)
        sql = "update device set led = 'reservation' where number=3"
        curs.execute(sql)
    if (End_led == CurrentTime):
        l = "StopLed"
        sql = "update device set led = 'off' where number=1"
        curs.execute(sql)
        sql = "update device set led = 'reservation' where number=3"
        curs.execute(sql)
    if (Start_fan == CurrentTime):
        h = "StartFan"
        sql = "update device set fan = 'on' where number=1"
        curs.execute(sql)
        sql = "update device set fan = 'reservation' where number=3"
        curs.execute(sql)
    if (End_fan == CurrentTime):
        h = "StopFan"
        sql = "update device set fan = 'off' where number=1"
        curs.execute(sql)
        sql = "update device set fan = 'reservation' where number=3"
        curs.execute(sql)
    conn.commit()
    print(s1,s2,s3,l,h)

except:
    pass
```

```
def SendMessage(s1, s2, s3, l, h):
    clientSocekt.send(bytes('%s,%s,%s,%s,%s'%(s2,s3,l,h,s1),"UTF-8")) #순서 이대로 보내야 Rpi에서 s1부터 받아짐

def WaterCycle(watercycleCode, soil):
    if watercycleCode == "053001":
        if soil < 550 :
            return "soilgood"
        else:
            return "StartMotor"
    elif watercycleCode == "053002":
        if soil < 750 :
            return "soilgood"
        else:
            return "StartMotor"
    elif watercycleCode == "053003":
        if soil < 900 :
            return "soilgood"
        else:
            return "StartMotor"
    elif watercycleCode == "053004":
        if soil < 1000 :
            return "soilgood"
        else:
            return "StartMotor"

def Humi(humi):
    if humi <=50 :
        return "humigood"
    else:
        return "StartFan"

##def WaterLevel(level):
##    if level <= 2 :
##        return "waterlevelbad"
##    else:
##        return "waterlevelgood"

def DeviceStateUpload(DeviceState,userid):
    global s1, s2, s3, l, h
    time = datetime.datetime.now()+timedelta(hours=9) #서버가 오하이오주라서 +9시간
    curs = conn.cursor()
    if DeviceState == "water1Finish":
        sql = "UPDATE device SET water1 = %s LIMIT 2"
        curs.execute(sql, ('off'))
        s1 = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT water1 FROM device WHERE number=3),%s)"
        curs.execute(sql, ("water1", "finish", time, userid))
    if DeviceState == "water2Finish":
        sql = "UPDATE device SET water2 = %s LIMIT 2"
        curs.execute(sql, ('off'))
        s2 = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT water2 FROM device WHERE number=3),%s)"
        curs.execute(sql, ("water2", "finish", time, userid))
    if DeviceState == "water3Finish":
        sql = "UPDATE device SET water3 = %s LIMIT 2"
        curs.execute(sql, ('off'))
        s3 = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT water3 FROM device WHERE number=3),%s)"
        curs.execute(sql, ("water3", "finish", time, userid))
    if DeviceState == "ledFinish":
        sql = "UPDATE device SET led = %s WHERE number = 2"
        curs.execute(sql, ('off'))
        l = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT led FROM device WHERE number=3),%s)"
        curs.execute(sql, ("led", "finish", time, userid))
    if DeviceState == "ledOperate":
        sql = "UPDATE device SET led = %s WHERE number = 2"
        curs.execute(sql, ('on'))
        l = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT led FROM device WHERE number=3),%s)"
        curs.execute(sql, ("led", "operate", time, userid))
    if DeviceState == "fanFinish":
        sql = "UPDATE device SET fan = %s WHERE number = 2"
        curs.execute(sql, ('off'))
        h = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT fan FROM device WHERE number=3),%s)"
        curs.execute(sql, ("fan", "finish", time, userid))
    if DeviceState == "fanOperate":
        sql = "UPDATE device SET fan = %s WHERE number = 2"
        curs.execute(sql, ('on'))
        h = 'null'
        sql = "INSERT INTO DeviceRecord (device, state, time, mode,id) VALUES(%s,%s,%s,(SELECT fan FROM device WHERE number=3),%s)"
        curs.execute(sql, ("fan", "operate", time, userid))
    conn.commit()

def SensorUpload(table):
    curs = conn2.cursor()
    data = clientSocekt.recv(BUFSIZE)
    Ard_data = data.decode()
    json_data = json.loads(Ard_data)
    temp = json_data['temp']
    humi = json_data['humi']
    soil1 = json_data['soil1']
    soil2 = json_data['soil2']
    soil3 = json_data['soil3']
    cds = json_data['cds']
    level = json_data['level']
    currentdate = datetime.datetime.now()+timedelta(hours=9)
    date = currentdate.strftime('%Y-%m-%d %H:%M:%S')
    sql = "INSERT INTO " + table + " (temp, humi, soil1, soil2, soil3, cds, level, date) VALUES (%s,%s,%s,%s,%s,%s,%s,%s)"
    curs.execute(sql, (temp, humi, soil1, soil2, soil3, cds, level, date))
    conn2.commit()
    print(temp, humi, soil1, soil2, soil3, cds, level, date)
```

```
def PlantInfo(json_data):
    currentdate = datetime.datetime.now()+timedelta(hours=9)
    month = currentdate.strftime('%m')
    temp = json_data['temp']
    humi = json_data['humi']
    soil1 = json_data['soil1']
    soil2 = json_data['soil2']
    soil3 = json_data['soil3']
    cds = json_data['cds']
    level = json_data['level']
    userid = json_data['id']
    curs = conn.cursor()
    sql = "select * from userplant where id = %s"
    curs.execute(sql,userid)
    rows = curs.fetchall()
    curs.close()
    h = Humi()
    if h == 'StartFan':
        sql = "update device set fan ='auto' where number=3"
        curs.execute(sql)

    if rows[0] != None:
        row = rows[0]
        plantdata = json.loads(row[2])
        hdCode= plantdata["hdCode"]
        if month == '02' or month == '01' or month =='12':
            watercycleCode = plantdata["watercycleWinterCode"]
        elif month == '03' or month == '04' or month == '05':
            watercycleCode = plantdata["watercycleSprngCode"]
        elif month == '06' or month == '07' or month == '08':
            watercycleCode = plantdata["watercycleSummerCode"]
        elif month == '09' or month == '10' or month == '11':
            watercycleCode = plantdata["watercycleAutumnCode"]
        s1 = WaterCycle(watercycleCode, soil1) + '1'
        if s1 == 'StartMotor1':
            sql = "update device set water1='auto' where number=3"
            curs.execute(sql)

    if rows[1] == None:
        pass
    else:
        row = rows[1]
        plantdata = json.loads(row[2])
        hdCode= plantdata["hdCode"]
        if month == '02' or month == '01' or month =='12':
            watercycleCode = plantdata["watercycleWinterCode"]
        elif month == '03' or month == '04' or month == '05':
            watercycleCode = plantdata["watercycleSprngCode"]
        elif month == '06' or month == '07' or month == '08':
            watercycleCode = plantdata["watercycleSummerCode"]
        elif month == '09' or month == '10' or month == '11':
            watercycleCode = plantdata["watercycleAutumnCode"]
        s2 = WaterCycle(watercycleCode, soil2) + '2'
        if s2 == 'StartMotor2':
            sql = "update device set water2='auto' where number=3"
            curs.execute(sql)

    if rows[2] == None:
        pass
    else:
        row = rows[0]
        plantdata = json.loads(row[2])
        hdCode= plantdata["hdCode"]
        if month == '02' or month == '01' or month =='12':
            watercycleCode = plantdata["watercycleWinterCode"]
        elif month == '03' or month == '04' or month == '05':
            watercycleCode = plantdata["watercycleSprngCode"]
        elif month == '06' or month == '07' or month == '08':
            watercycleCode = plantdata["watercycleSummerCode"]
        elif month == '09' or month == '10' or month == '11':
            watercycleCode = plantdata["watercycleAutumnCode"]
        s3 = WaterCycle(watercycleCode, soil3) + '3'
        if s3 == 'StartMotor3':
            sql = "update device set water3='auto' where number=3"
            curs.execute(sql)
    conn.commit()
    SendMessage(h,s1,s2,s3,1)

########################################################################
sched = BackgroundScheduler()
sched.start()

sched.add_job(PlantInfo,'cron', second='1', id="test_1", args=[RecvData()]) #1분에 한번
sched.add_job(SensorUpload,'cron', second='31', id="test_2", args=['Sensor']) #1분에 한번
#sched.add_job(SensorUpload,'cron', minute="00", second='00', id="test_3", args=['Graph']) #1시간에 한번
########################################################################
while True:
    try:
        conn = pymysql.connect(host='database-1.cudf3zzu3npf.us-east-2.rds.amazonaws.com', user='jeongmin', password='97shwjdals!', db='mydb',charset='utf8')
        RecvData()
        UserSettingCheck()
        SendMessage(h,s1,s2,s3,1)
        time.sleep(3)

    except SocketError as e:
        print('에러 발생', e)
        conn.close()
        clientSocekt.close()
        serverSocket.close()
        print('close')

    except KeyboardInterrupt:
        print ("Shutdown requested...exiting")
        sys.exit()

    finally:
        conn.commit()
        conn.close()
```

- 라즈베리파이 소스코드

```
from socket import *
from select import *
import sys
from time import ctime
import json
from collections import OrderedDict
import serial
import time
import os
from socket import error as SocketError
import errno
import schedule
from apscheduler.jobstores.base import JobLookupError
from apscheduler.schedulers.background import BackgroundScheduler
HOST = '18.218.71.35' # EC2 서버 주소
PORT = 10001
BUFSIZE = 1024
ADDR = (HOST,PORT)

if os.path.exists("/dev/ttyACM0") :
    tty = "/dev/ttyACM0"
elif os.path.exists("/dev/ttyACM1") :
    tty = "/dev/ttyACM1"
ArduinoSerial = serial.Serial(tty, 9600);

s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST,PORT))
print("success!")

def sendSensorData():
    f = open("SensorValue", 'r')
    json_data = f.readline()
    if not json_data:
        print("empty sensor data")
    # 서버에 데이터 전송
    s.send(json_data.encode())
    f.close()

sched = BackgroundScheduler()
sched.start()

sched.add_job(sendSensorData,'interval', seconds=5, id="test_1", args=[])
```

```
while True:
    try:
        buffer = s.recv(BUFSIZE)
        data = buffer.decode("UTF-8")
        DeviceValue = data.split(',')
        water1 = DeviceValue[0]
        water2 = DeviceValue[1]
        water3 = DeviceValue[2]
        led = DeviceValue[3]
        fan = DeviceValue[4]

        print(water1, water2, water3, led, fan)
        ArduinoSerial.flushInput();

        if (water1 == "StartMotor1"):
            ArduinoSerial.write(b'StartWater1')
            print("StartWater1")
        elif (water2 == "StartMotor2"):
            ArduinoSerial.write(b'StartWater2')
            print("StartWater2")
        elif (water3 == "StartMotor3"):
            ArduinoSerial.write(b'StartWater3')
            print("StartWater3")
        elif (led == "StartLed"):
            ArduinoSerial.write(b'StartLed')
            print("StartLed")
        elif (led == "StopLed"):
            ArduinoSerial.write(b'StopLed')
            print("StopLed")
        elif (fan == "StartFan"):
            ArduinoSerial.write(b'StartFan')
            print("StartFan")
        elif (fan == "StopFan"):
            ArduinoSerial.write(b'StopFan')
            print("StopFan")
        else:
            ArduinoSerial.write(b'null')
            print("Nonoperating")

        ArduinoSerial.flushOutput();
        time.sleep(4)
    except SocketError as e:
        print('에러 발생', e)
        s.close()
    except:
```

```
import serial
import os

if os.path.exists("/dev/ttyACM0") :
    tty = "/dev/ttyACM0"
elif os.path.exists("/dev/ttyACM1") :
    tty = "/dev/ttyACM1"

ArduinoSerial = serial.Serial(tty, 9600);
ArduinoSerial.flushInput();

while 1:
    try:
        input = ArduinoSerial.readline();
        input = str(input)
        with open('SensorValue','w') as f:
            f.write(input);
    except:
        pass
```

- 아두이노 주요 소스코드

```
  if(Serial.available())
  {
    String Data = Serial.readString();
    if(Data == "StartFan") //쿨링팬 작동
    {
      operateFan();
      DeviceState = "fanOperate";
    }
    if(Data == "StopFan") //쿨링팬 작동
    {
      stopFan();
      DeviceState = "fanFinish";
    }
    if(Data == "StartLed") //식물생장용 LED 작동
    {
      operateLed();
      DeviceState = "ledOperate";
    }
    if(Data == "StopLed") //쿨링팬 작동
    {
      stopLed();
      DeviceState = "ledFinish";
    }
    if(Data == "StartWater1") //워터펌프1 작동
    {
      operateMotor('a');
      DeviceState = "water1Finish";
    }
```

```
void operateFan(){ //쿨링팬
  digitalWrite(FAN, LOW);
}
void stopFan(){
  digitalWrite(FAN, HIGH);
}
```

```
void operateLed() {  //LED
  digitalWrite(LED, HIGH);
}
void stopLed() {
  digitalWrite(LED, LOW);
}
```

```
void operateMotor(char motor){   //워터펌프 1,2,3
  if (motor == 'a'){ //워터펌프1
    digitalWrite(motor1, LOW);
    delay(3000);
    digitalWrite(motor1, HIGH);
  }
  if (motor == 'b'){ //워터펌프2
    digitalWrite(motor2, LOW);
    delay(3000);
    digitalWrite(motor2, HIGH);
  }
  if (motor == 'c'){ //워터펌프3
    digitalWrite(motor3, LOW);
    delay(3000);
    digitalWrite(motor3, HIGH);
  }
}
```

```
//함수로 받아온 데이터값을 Json으로 변환
void makeJson(String ID, float temp, float humi,
int soil1, int soil2, int soil3,
int cds, int level, String DeviceState)
{
  StaticJsonDocument<256> doc;
  JsonObject root = doc.to<JsonObject>();
  root["id"] = ID;
  root["temp"] = temp;
  root["humi"] = humi;
  root["soil1"] = soil1;
  root["soil2"] = soil2;
  root["soil3"] = soil3;
  root["cds"] = cds;
  root["level"] = level;
  root["DeviceState"] = DeviceState;
  serializeJson(root, Serial);
  Serial.println();
}

void loop() {
  float temp = dht.readTemperature();
  float humi = dht.readHumidity();
  int soil1 = analogRead(SOIL1);
  int soil2 = analogRead(SOIL2);
  int soil3 = analogRead(SOIL3);
  int cds = analogRead(CDS);
  int level = analogRead(LEVEL);
```

- 안드로이드 주요 소스코드

```
public void setLedStatus(final String id, final String led)
{
    RequestQueue queue = Volley.newRequestQueue(getContext());
    String setLedStatusUrl = "http://18.218.71.35/SetLedStatus.jsp"; //온습도를 받아오는 url
    //Toast.makeText(getContext(), led + "상태", Toast.LENGTH_SHORT).show();
    StringRequest stringRequest = new StringRequest(Request.Method.POST, setLedStatusUrl, new Response.Listener<String>()
    {
        @Override
        public void onResponse(String response)
        {
            // Response
            try
            {
                JSONArray jarray = new JSONObject(response).getJSONArray( name: "List");
                JSONObject jObject = jarray.getJSONObject( index: 0);
                String result = jObject.optString( name: "RESULT");
                if(result.equals("1"))
                {
                    Toast.makeText(getContext(),  text: led + "led 상태 변경 완료", Toast.LENGTH_SHORT).show();
                }
                else
                {
                    Toast.makeText(getContext(),  text: led + "led 상태 변경 실패", Toast.LENGTH_SHORT).show();
                }
            }
            catch(org.json.JSONException e)
            {
            }
        }
    }, new Response.ErrorListener()
    {
        @Override
        public void onErrorResponse(VolleyError error)
        {
            // Error Handling
            Toast.makeText(getContext(),  text: error.getMessage() + "시스템 오류", Toast.LENGTH_SHORT).show();
        }
    })

    {
        @Override
        protected Map<String, String> getParams() throws AuthFailureError
        {
            Map<String, String> params = new HashMap<>();
            params.put("ID", id);
            params.put("LED", led);
            return params;
        }
```

```
try {
    URL url = new URL( spec: "http://api.nongsaro.go.kr/service/garden/gardenList?"    //'농사로' OpenAPI
            + "numOfRows="
            + "217"
            + "&apiKey="
            + "20200114KXGDZSXXBNORHYRUSLWYQ"
            + "&cntntsNo=&pageNo=1&word=&lightChkVal=055003&grwhstleChkVal=054001%2C054002%2C054004&le
    ); //검색 URL부분

    XmlPullParserFactory parserCreator = XmlPullParserFactory.newInstance();
    XmlPullParser parser = parserCreator.newPullParser();
    parser.setInput(url.openStream(), inputEncoding: null);
    int parserEvent = parser.getEventType();
    System.out.println("파싱시작합니다.");

    while (parserEvent != XmlPullParser.END_DOCUMENT) {
        switch (parserEvent) {
            case XmlPullParser.START_TAG://parser가 시작 태그를 만나면 실행
                if (parser.getName().equals("cntntsNo")) { //컨텐츠 번호 만나면 내용을 받을수 있게 하자
                    inNum = true;
                }
                if (parser.getName().equals("cntntsSj")) { //식물 이름 만나면 내용을 받을수 있게 하자
                    inName = true;
```

Ⅲ. 결론

1. 연구 결과

온실 관리를 위한 기존 국내 외 기술을 분석한 결과, 현재 시중에 판매되고 있는 온실 관리 제품들에 비해 와이파이 이용이 가능하고 그를 통해 온실 원격 조정 및 온실 스트리밍이 가능하다. 또한 식물에 맞는 생장 환경을 조절해주는 장점을 가지고 있다. 이러한 점들이 Greenery 시스템의 효용성을 나타낸다.

이에 본 논문에서 다루는 원격 온실 관리 시스템은 기존 제품들의 단점을 보완하여 블루투스가 아닌 와이파이를 이용하여 식물을 관리하고, 영상 스트리밍을 통해 식물의 생장 과정을 확인 가능하며, 여러 종류의 식물을 선택할 수 있는 기능을 추가하였다. 또한, 관리자 페이지를 제작하여 서버관리자가 온실의 장치 상태 확인 및 회원을 관리할 수 있도록 하였다.

현재 식물에 관한 데이터는 농사로의 OpenAPI에 의존하고 있으나 그 정보는 실험 및 결과에서 언급한 것처럼 일부 데이터를 정확한 수치가 아닌 문장으로 설명하는 부분이 존재한다는 단점이 있다. 또한 식물은 주변 온도와 습도, 광량 등 생육 환경마다 다른 점이 분명 존재한다. 이에 앞으로의 연구는 지능형 사물인터넷을 이용한 식물 생장 환경 예측과 같은 방향으로 기존 데이터를 기초로 하되 센서들을 통해 수집한 데이터를 분석하여 동적으로 온실의 환경을 조절할 수 있도록 알고리즘을 설계하는 방향으로 나아가고자 한다.

2. 작품제작 소요재료 목록

- Arduino UNO
- Raspberry Pi 3 B
- HS-WATER-SENSOR : 수조 수위 측정용 수위 센서
- DHT11 : 온실 온도 및 습도 측정용 센서
- GL5537 : 조도 측정 센서
- DM456 : 토양 수분 측정을 위한 토양 수분 센서
- SZH-GNP155 : 자동 급수 펌프
- 물탱크 : 급수용 물 저장 수조
- 8MP CAMERA : 실시간 온실 이미지 확인을 위한 카메라
- 식물생장용 LED
- 쿨링팬
- 온실용 아크릴판
- 실험용 식물

참고자료

[1] 이강오, 『즐거운 농업의 시작, 스마트팜 이야기』, 넥센미디어(2019)
[2] 정수영, “토닥토닥 선인장이 위로해주네…'반려식물' 인기”,NEWS1제주, 2017.11.26, http://jeju.news1.kr/news/articleView.html?idxno=22338
[3] AWS,AWS EC2, https://aws.amazon.com/ko/ec2/
[4] 농사로,OpenAPI(식물정보제공), http://www.nongsaro.go.kr/portal/ps/psz/psza/contentMain.ps?menuId=PS03954