목 차

1. 요구 조건 및 제약 사항 분석에 대한 수정사항

1-1. 기존 요구 조건

1-1-1. 실내 위치 추정

RSS(무선 신호 세기) 값을 사용한 실내 위치 추정 기술을 개선하고, 이를 이용해 사용자에게 다양한 서비스를 제공하는 어플리케이션을 개발한다.

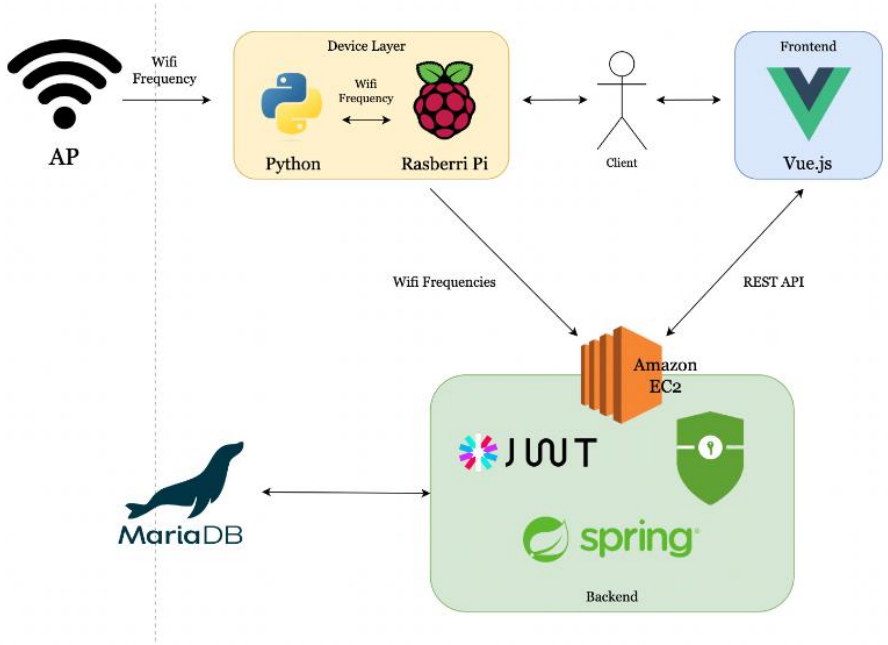
- 실내 위치 추정 기술 개선
- 개선된 실내 위치 추정 기술 기반 실내 길 안내 서비스 개발
- 개선된 실내 위치 추정 기술 기반 사용자 주변의 자원 제어 서비스 개발

1-1-2. 사용자 인증

사용자 인증 보안을 활용하여 사용자 인증 기술을 개발하고, 개인정보가 보다 안전하게 유지될 수 있는 서비스를 개발한다.

- Email 인증: 부산대 웹메일로 인증을 진행하여 사용자에게 "학생" 권한 부여
- OTP: 모든 사용자는 OTP 인증을 통해 로그인 함으로써 시 보안 강화
- QR 인증: 사용자의 권한이 "교수"인 경우 QR 인증을 통해 강화된 인증 절차 진행
- 기기 인증: Rasberry pi에 미리 기입된 고유 코드를 통해 해당 기기의 유효성 검증

1-1-3. 시스템 구조도



1-2. 요구 조건 수정사항

1-2-1. 수집 데이터 변경 사항

- 데이터 수집
  - → 수정 전: 정확한 측정을 위해 10 회 측정 후 threshold(-70) 이상의 값이 5 회 이상 나온 MAC 주소만 radio map, 즉 위치 별 RSS 값이 기록된 지도에 기입한다.
  - → 수정 후: 값이 5 회 이상 나온 MAC 주소만 사용하기에는 정확도가 떨어진다고 판단하여, 한 번이라도 등장한 MAC 주소들은 모두 radio map 에 기입한다.
- 데이터 전처리
  - → 수정 전: 수집 단계에서 반복 측정한 MAC 주소 별 RSS값의 median(중간값)을 사용하도록 한다
  - → Median(중간값)보다 Mean(평균값)이 더 정확하다고 판단하여 mean(평균값)을 사용하도록 한다.

1-3. 제약 사항 분석에 대한 수정사항

1-3-1. 추가 제약사항 및 대책

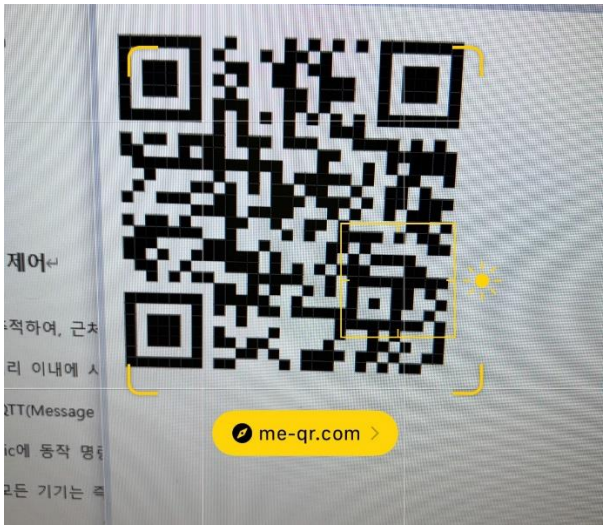
- 실내 위치 추정 기능의 정확도 문제
  - ✓ 위에서 언급했듯 5 회 이상 나온 MAC 주소가 아닌, 한 번이라도 threshold 이상의 값이 나온 MAC 주소들은 모두 Radio map 에 기입하고, 평균값 계산해서 서버로 업로드 하도록 로직 변경
  - ✓ 사용자의 진행 방향에 대하여 너무 튀는 값, 혹은 길이 아닌 곳으로 추정되는 값은 제외한다.
- 같은 리소스를 제어하려는 사용자 간의 충돌 가능성
  - ✓ 먼저 요청한 사용자에게 권한을 우선 부여하도록 규칙을 설정한다.
- 인터넷 환경에 따른 전송 속도 저하 및 연결 불안정
  - ✓ 안정적인 연결을 보장할 수 없는 공용 와이파이가 아닌, 모바일 핫스팟을 사용한다.
  - ✓ 핫스팟은 휴대폰 배터리가 충분하고 근처에서 소지하고만 있으면 안정적인 네트워크 연결이 보장된다는 장점이 있다.
- Rasberry pi 에서 실시간으로 전송하는 데에 걸리는 시간이 매번 다름
- Rasberry pi 측정 중 서버 과부화로 인한 측정 실패 현상 발생

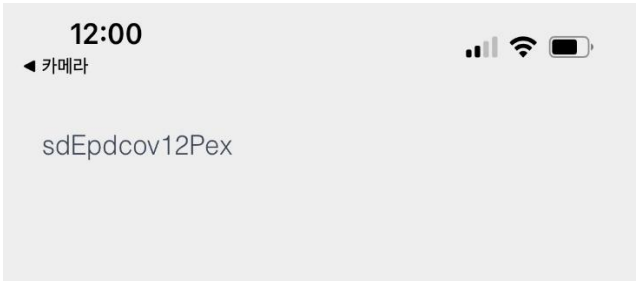
2. 설계 상세화 및 변경 내역

2-1. 다양한 인증 방식

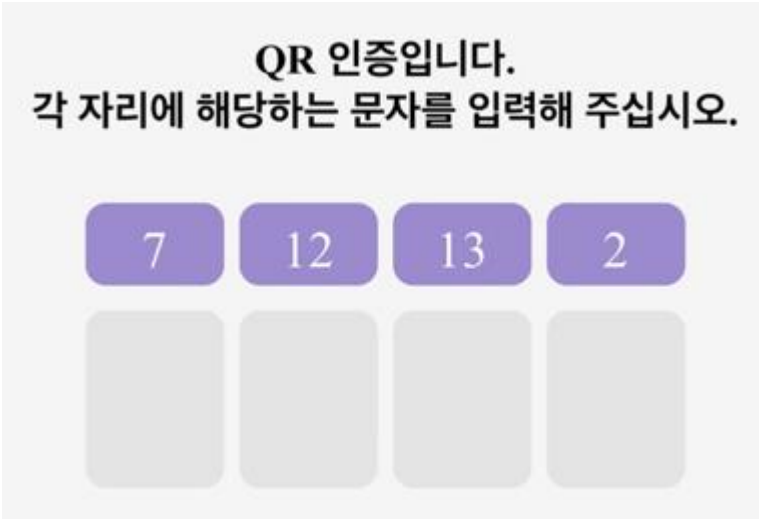
2-1-1. QR 인증

- 교수 권한을 가진 사용자에게는 임의의 QR 코드가 관리자로부터 지급된다.
  - ✓ 해당 QR 코드를 촬영하면 우측과 같이 텍스트를 획득할 수 있다.



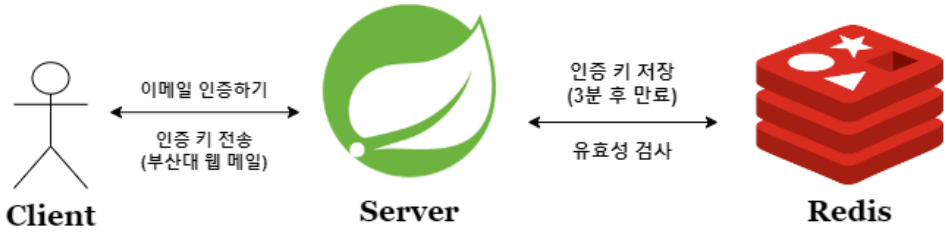


- 해당 QR 코드는 사용자의 어플리케이션에서 아래와 같이 인증 시 사용된다.
  - ✓ 상단에 출력되는 자리 수에 맞는 문자를 입력해 서버에 전송한다.
  - ✓ 서버에서는 DB에 저장된 QR코드 문자열과 비교하여, 유효한 QR 코드인지 인증하도록 한다.



2-1-2. 이메일 기반 인증(SMTP)

- 부산대 웹 메일을 통하여 인증한 사용자에 한하여 "학생" 권한을 부여한다.
  - ✓ 일반 권한의 사용자는 지도 기능만 사용 가능하다.
- Springboot 서버에서 SMTP(Simple Message Transfer Protocol)를 통해 인증 키를 전송한다.
- Redis에 인증 키를 저장하며, 사용자의 요청에 따라 유효한 인증 키 인지 검사한다.
  - ✓ Redis는 유효 기간을 설정할 수 있어 유용한 in-memory DB이다.

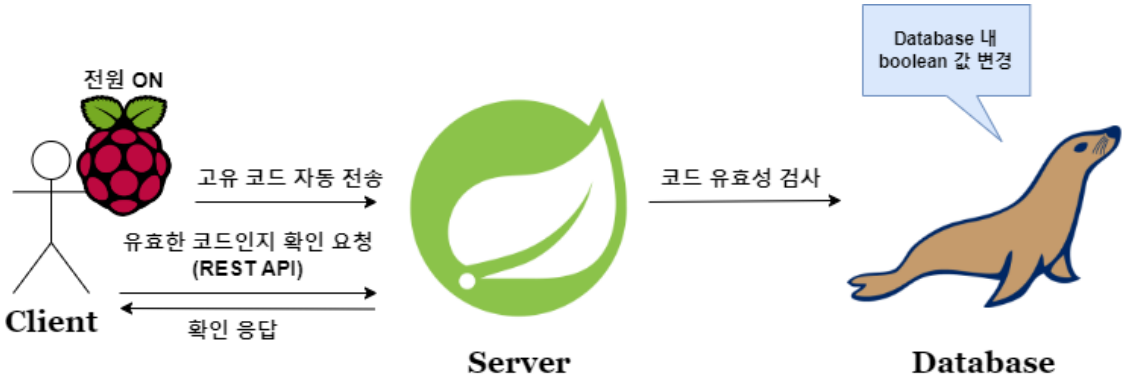


2-1-3. OTP 인증

- 학생 또는 교수 권한을 가진 사용자는 모두 OTP 인증을 받도록 한다.
- 2-1-2의 SMTP를 이용하여, 서버에서 랜덤하게 생성한 4자리 OTP를 메일로 전송한다.
  - ✓ 서버에서 매 요청마다 랜덤하게 코드를 생성하므로, One Time Password 방식이다.
  - ✓ Random 함수를 통해 15자리의 난수 생성 후, 해당 난수 내에서 한 번 더 Random 함수를 통해 자리를 골라 4자리 코드를 생성한다. (ex) 927354019275214 → 7495)
- OTP 또한 유효기간이 필요하므로, 이메일 인증과 동일하게 Redis를 사용한다.
  - ✓ Redis에 저장된 코드와 비교하여, 유효한 인증 코드인지 검사한다.

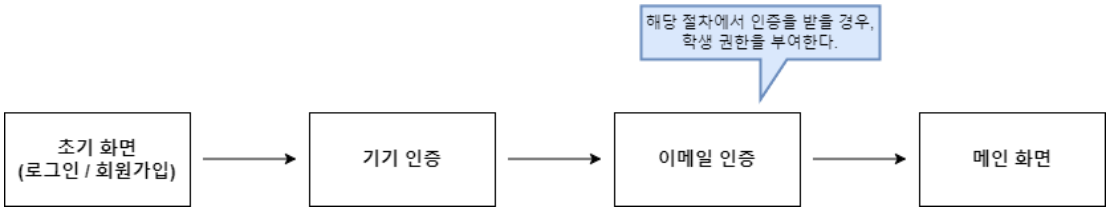
2-1-4. 기기 기반 인증

- 사용자가 휴대중인 Raspberry pi가 유효한 기기인지 검증한다.
  - ✓ 관리자들은 초기에 각 기기에 고유 코드를 미리 DB와 기기에 기록하여 배부한다.
  - ✓ 해당 고유 코드는 Raspberry pi의 전원이 켜질 때, 자동으로 서버에 전송된다.
- 해당 고유 코드가 DB 내에 존재하는 유효한 코드일 경우 유저를 완전히 로그인 시킨다.
  - ✓ 이전 단계까지는 가로그인 상태이며, 해당 단계를 통해 인증 단계가 마무리된다.
- 사용자는 REST API 요청을 보내 자신이 유효한 코드를 전송했는지 한 번 더 확인한다.
  - ✓ DB 내에 해당 사용자가 로그인 상태로 기록되어 있으면, 서비스를 이용하도록 인가한다.



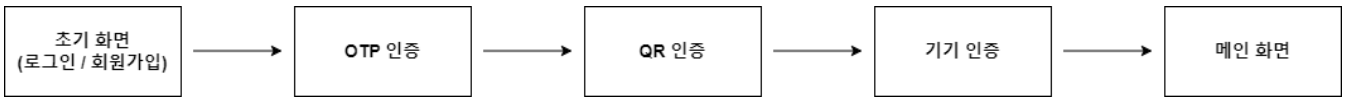
2-1-5. 권한 별 인증 절차

- 일반 권한: 이메일 인증을 거치지 않은 초기 사용자



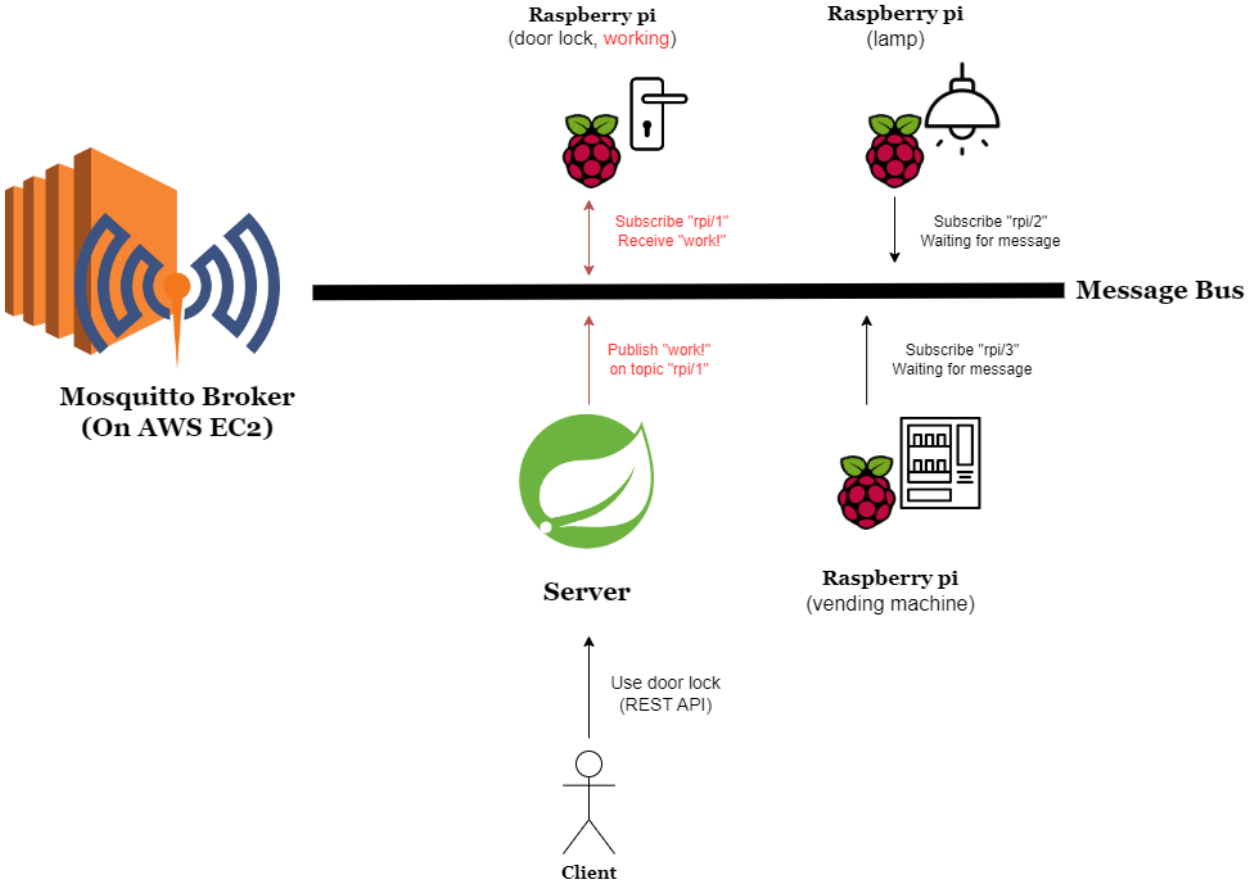
- 학생 권한: 이메일 인증을 진행한 사용자

- 교수 권한: 관리자에게 교수임을 증명한 뒤 권한을 획득한 사용자



2-2. 위치 기반 주변 리소스 제어

- 사용자의 위치를 지속 추적하여, 근처에 있는 기기를 제어할 수 있도록 한다.
  - ✓ 기기로부터 일정 거리 이내에 사용자가 들어오면 푸시 알림을 전송한다.
- 아래와 같은 구조의 MQTT(Message Queue Telemetry Transport) 프로토콜을 이용해 구현한다.
  - ✓ 사용자가 푸시 알림을 통해 기기를 사용하겠다고 요청을 보낸다.
  - ✓ 서버에서 특정 topic에 동작 명령을 내리고, 해당 topic을 구독한 기기는 message를 받는다.
  - ✓ Message를 받은 모든 기기는 즉시 동작이 가능하다.
- 사용자가 기기 근처에서 벗어나면, 자동으로 해당 기기는 종료된다.
  - ✓ 사용자의 위치를 지속 추적하기 때문에, 해당 기능 또한 MQTT로 구현 가능하다.
  - ✓ 사용자가 기기에서 일정 거리 이상 벗어나면 서버에서 동작 중지 message를 보낸다.
  - ✓ Message를 받은 모든 기기는 즉시 동작을 중지한다.
- 위의 모든 통신에는 Broker가 반드시 필요하며, 해당 과제에서는 **Mosquitto**를 사용하였다.



2-3. 위치 기반 경로 탐색

2-3-1. 실내 위치 추정

- Fingerprint 기법을 사용하여 실내 위치 추정을 진행한다.
  - ✓ Fingerprint 기법은 각 위치에서 사전에 기록해 놓은 RSS(무선 신호 세기) 값을 바탕으로 현재 위치에서 수신되는 RSS 값과의 유사도를 측정해 위치를 추정하는 방식이다.
  - ✓ 모든 사용자에 대해 동일한 radio map(RSS 값 지도)을 사용할 수 있어야 하므로 서버에 static(전역 변수)하게 구현하여 유지한다.
  - ✓ MAC 주소와 해당 주소에 대한 RSS mean(평균)값을 서버에 아래와 같이 JSON 형태로 전달한다.

```
2023-04-30 16:58:01.396  INFO 42121 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        :
2023-04-30 16:58:01.436  WARN 42121 --- [nio-8080-exec-1] .w.s.m.s.DefaultHandlerExceptionResolver :
test = {AA:AA:AA:AA:AA:AA=-38, BB:BB:BB:BB:BB:BB=-80, CC:CC:CC:CC:CC:CC=-37, CD:CD:CD:CD:CD:CD=-75}
test = {AA:AA:AA:AA:AA:AA=-38, BB:BB:BB:BB:BB:BB=-80, CC:CC:CC:CC:CC:CC=-37, CD:CD:CD:CD:CD:CD=-75}
```

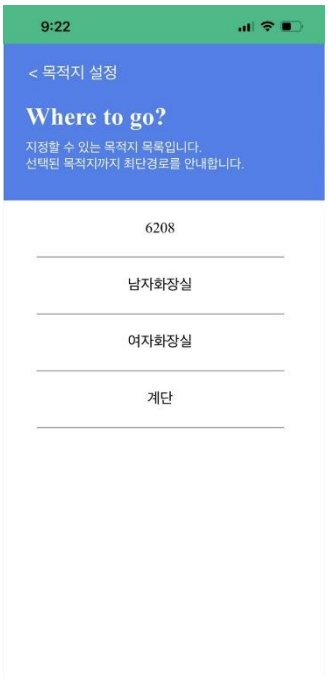
```
2023-05-02T09:42:01.264Z  INFO 63304 --- [           main] com.example.test.TestApplication         : Starting TestApplication v0.0.1-SNAPSHOT using Java 17.0.6 with PID 63304 (/home/ec2-use
r/test-0.0.1-SNAPSHOT.jar started by ec2-user in /home/ec2-user)
2023-05-02T09:42:01.279Z  INFO 63304 --- [           main] com.example.test.TestApplication         : No active profile set, falling back to 1 default profile: "default"
2023-05-02T09:42:03.458Z  INFO 63304 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-05-02T09:42:03.479Z  INFO 63304 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-05-02T09:42:03.480Z  INFO 63304 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-05-02T09:42:03.734Z  INFO 63304 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-05-02T09:42:03.750Z  INFO 63304 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2347 ms
2023-05-02T09:42:05.004Z  INFO 63304 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-05-02T09:42:05.046Z  INFO 63304 --- [           main] com.example.test.TestApplication         : Started TestApplication in 4.717 seconds (process running for 5.692)

2023-05-02T09:43:20.838Z  INFO 63304 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-05-02T09:43:20.838Z  INFO 63304 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2023-05-02T09:43:20.840Z  INFO 63304 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
rss = }
rss = {"08:40:F3:19:2F:81" : "-30", "CC:2D:21:9A:84:81" : "-40", "58:86:94:76:CA:6E" : "-54", "08:40:F3:1A:25:B1" : "-48", "88:36:6C:68:E4:B6" : "-63", "50:33:F0:67:84:ED" : "-72", "50:33:F0
:67:78:51" : "-62", "08:40:F3:1A:25:85" : "-71"}
rss = {"08:40:F3:19:2F:81" : "-34", "CC:2D:21:9A:84:81" : "-38", "58:86:94:76:CA:6E" : "-54", "08:40:F3:1A:25:B1" : "-55", "88:36:6C:68:E4:B6" : "-65", "50:33:F0:67:84:ED" : "-72", "50:33:F0
:67:78:51" : "-59", "08:40:F3:1A:25:85" : "-76"}
rss = {"08:40:F3:19:2F:81" : "-27", "CC:2D:21:9A:84:81" : "-37", "58:86:94:76:CA:6E" : "-59", "08:40:F3:1A:25:B1" : "-61", "88:36:6C:68:E4:B6" : "-65", "50:33:F0:67:84:ED" : "-72", "50:33:F0
:67:78:51" : "-49", "08:40:F3:1A:25:85" : "-75"}
rss = {"08:40:F3:19:2F:81" : "-30", "CC:2D:21:9A:84:81" : "-54", "58:86:94:76:CA:6E" : "-59", "08:40:F3:1A:25:B1" : "-61", "88:36:6C:68:E4:B6" : "-65", "50:33:F0:67:84:ED" : "-72", "50:33:F0
:67:78:51" : "-58", "08:40:F3:1A:25:85" : "-75"}

rss = {"08:40:F3:19:2F:81" : "-28", "CC:2D:21:9A:84:81" : "-49", "58:86:94:76:CA:6E" : "-70", "08:40:F3:1A:25:B1" : "-65", "50:33:F0:67:84:ED" : "-72", "50:33:F0:67:78:51" : "-55", "08:40:F3
:1A:25:B5" : "-75"}
rss = {"08:40:F3:19:2F:81" : "-65", "CC:2D:21:9A:84:81" : "-49", "58:86:94:76:CA:6E" : "-70", "08:40:F3:1A:25:B1" : "-65", "50:33:F0:67:78:51" : "-55"}
rss = {"08:40:F3:19:2F:81" : "-54", "CC:2D:21:9A:84:81" : "-81", "58:86:94:76:CA:6E" : "-70", "08:40:F3:1A:25:B1" : "-65", "50:33:F0:67:78:51" : "-55", "08:40:F3:18:C7:A1" : "-82", "08:40:F3
:1A:25:B5" : "-77"}
rss = {"08:40:F3:19:2F:81" : "-44", "CC:2D:21:9A:84:81" : "-81", "58:86:94:76:CA:6E" : "-70", "08:40:F3:1A:25:B1" : "-65", "50:33:F0:67:78:51" : "-84", "08:40:F3:18:C7:A1" : "-82", "08:40:F3
:1A:25:B5" : "-77"}
rss = {"08:40:F3:19:2F:81" : "-30", "CC:2D:21:9A:84:81" : "-53", "58:86:94:76:CA:6E" : "-68", "08:40:F3:1A:25:B1" : "-69", "50:33:F0:67:78:51" : "-62", "08:40:F3:18:C7:A1" : "-82", "08:40:F3
:1A:25:B5" : "-77"}
```

- 전송된 MAC 주소와 값을 서버에서 받아, 기존 radio map 에 기록된 MAC 주소와 값을 비교하여 MSE(Mean Square Error) 값을 계산해 현재 위치를 추정할 수 있다.

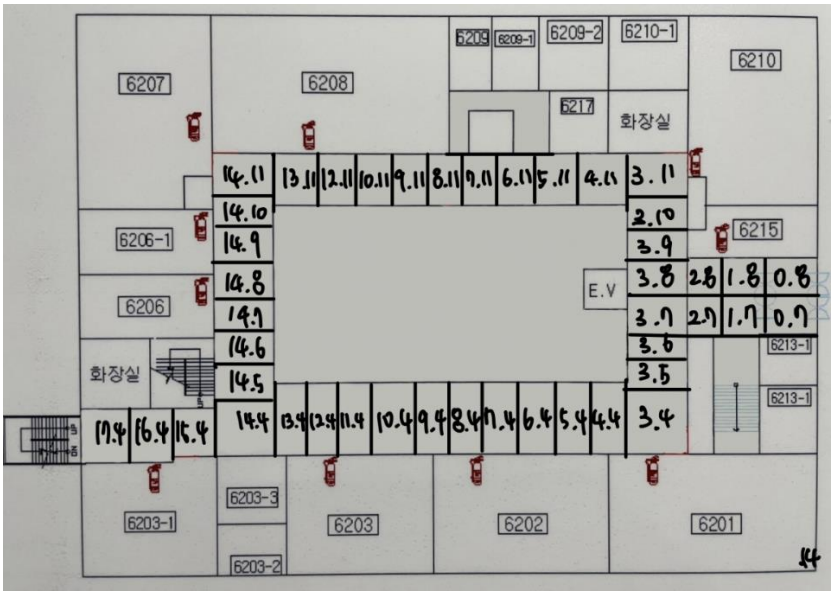
```
@Service
public class TestService {
    1 usage
    public void fingerprint(HashMap<String , Integer> rss){
        for (int i = 0; i < Rss.radioMap.length; i++) {
            for (int j = 0; j < Rss.radioMap[0].length; j++) {
                double mse = 0;
                for (Map.Entry<String ,Integer> entry : rss.entrySet()) {
                    if(rss.get(entry.getKey()) <= -70)
                        continue;
                    double diff = Rss.radioMap[i][j][Rss.indexMap.get(entry.getKey())] - rss.get(entry.getKey());
                    mse += Math.pow(diff, 2);

                    System.out.println("Key = " + entry.getKey());
                    System.out.println("pre : " + Rss.radioMap[i][j][Rss.indexMap.get(entry.getKey())]);
                    System.out.println("real : " + rss.get(entry.getKey()));


                }

                System.out.println(i + ", " + j + "'s mse = " + mse);
            }
```

2-3-2. 최단 경로 탐색

- BFS 기반의 Dijkstra 알고리즘을 사용하여 목적지까지 최단 경로를 찾아 안내한다.
  - ✓ 아래와 같이 사용자들에게 목적지를 선택할 수 있도록 목적지 리스트를 제공한다.



- 아래와 같이 측정 장소(컴퓨터 공학관)를 배열의 형태로 나누어 인접 행렬과 같은 형태로 간주한다.
  - ✓ 기존 Dijkstra 알고리즘과 동일하게 인접 행렬을 BFS 로 탐색하여 최단 경로를 안내한다.
  - ✓ Dijkstra 알고리즘은 최단 경로를 추적할 수 있으므로, 최단 경로를 따라가는 좌표 배열을 서버에서 생성하여 사용자에게 시각화하여 제공한다.



3. 갱신된 과제 추진 계획

단계	활동	5 월					6 월				
		1 주	2 주	3 주	4 주	5 주	1 주	2 주	3 주	4 주	5 주
계획	착수 보고										
	자재 준비										
	인프라 요구사항 정의										
설계	서버 구조 설계										
	디바이스 - 서버 연결 구조 설계										
	DB 구조 설계										
분석	신호 세기 측정										
구현	서버(백엔드) 구현										
	위치 추정 알고리즘 구현										
	디바이스 - 서버 연결										

단계	활동	7 월				8 월					9 월			
		1 주	2 주	3 주	4 주	1 주	2 주	3 주	4 주	5 주	1 주	2 주	3 주	4 주
계획	서비스 요구사항 정의													
설계	서비스 설계													
	중간 보고													
	백엔드 구현													
구현	서비스 개발 (PWA 포함)													
	최종 발표													

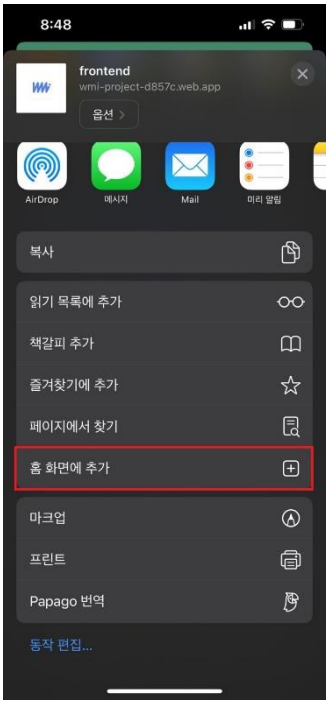
4. 구성원 별 진척도

담당자	업무
이준희	- 회원 및 리소스 정보 저장용 Maria DB 및 서버(AWS) 구성 완료 - 데이터 전처리를 통한 지도(Radio map) 구성 완료 - 서비스 제공을 위한 로직(최단 경로, 자원 제어) 구현 진행 중
심진섭	- Raspberry Pi 내에 RSS 측정 로직 구현 완료 - 데이터 전처리를 통한 지도(Radio map) 구성 완료 - Raspberry Pi – Java 간 MQTT 통신 구현 중 - Vue.js 기반 PWA 구현 중
이민경	- Raspberry pi 를 통한 RSS 데이터 수집 완료 - 데이터 전처리를 통한 지도(Radio map) 구성 완료 - Vue.js 기반 PWA 구현 중

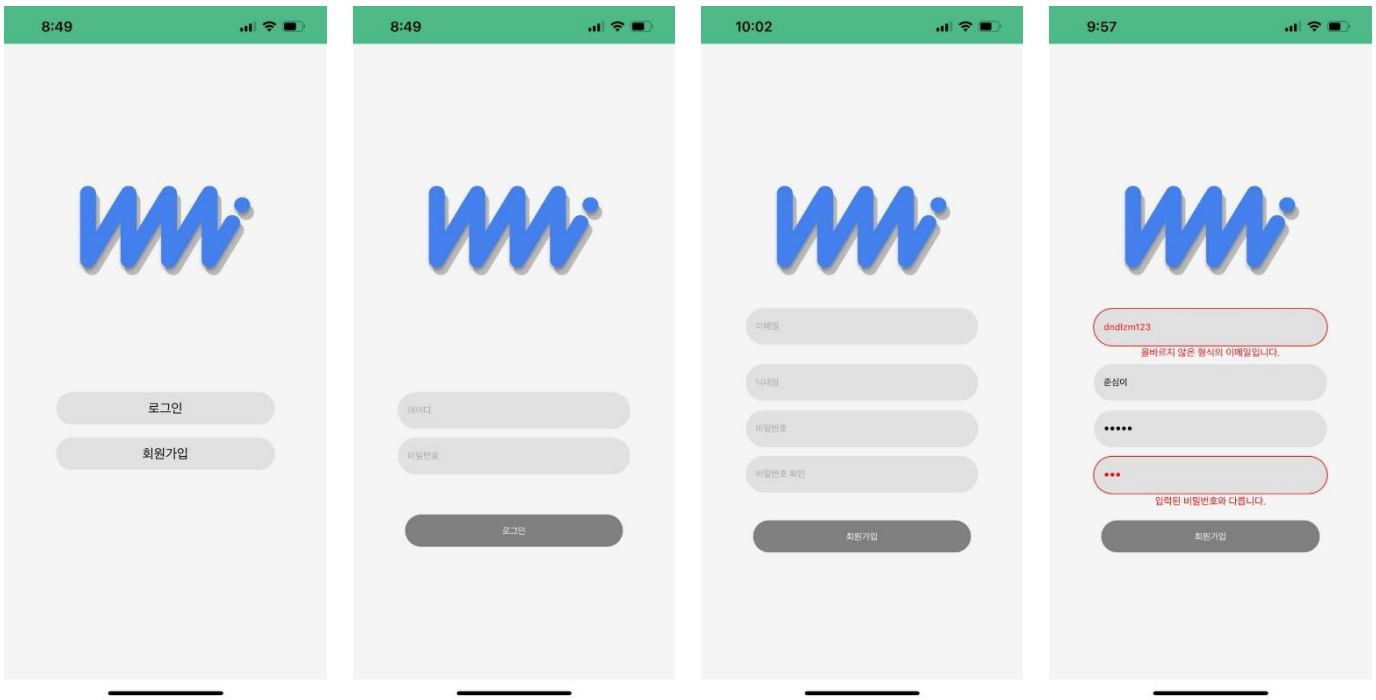
5. 보고 시점까지의 과제 수행 내용 및 중간 결과

5-1. Vue.js 기반 PWA

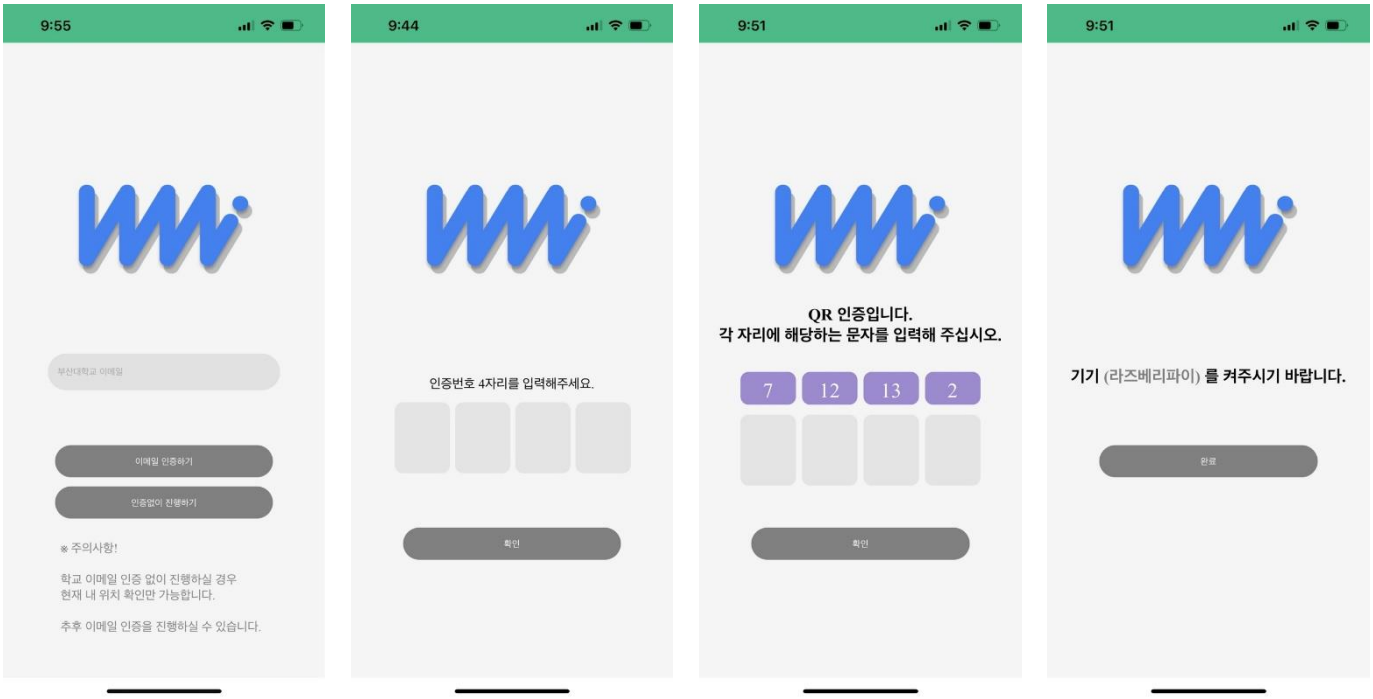
- PWA(Progressive Web App)이기 때문에, App 과 같이 홈 화면에 추가가 가능하다.



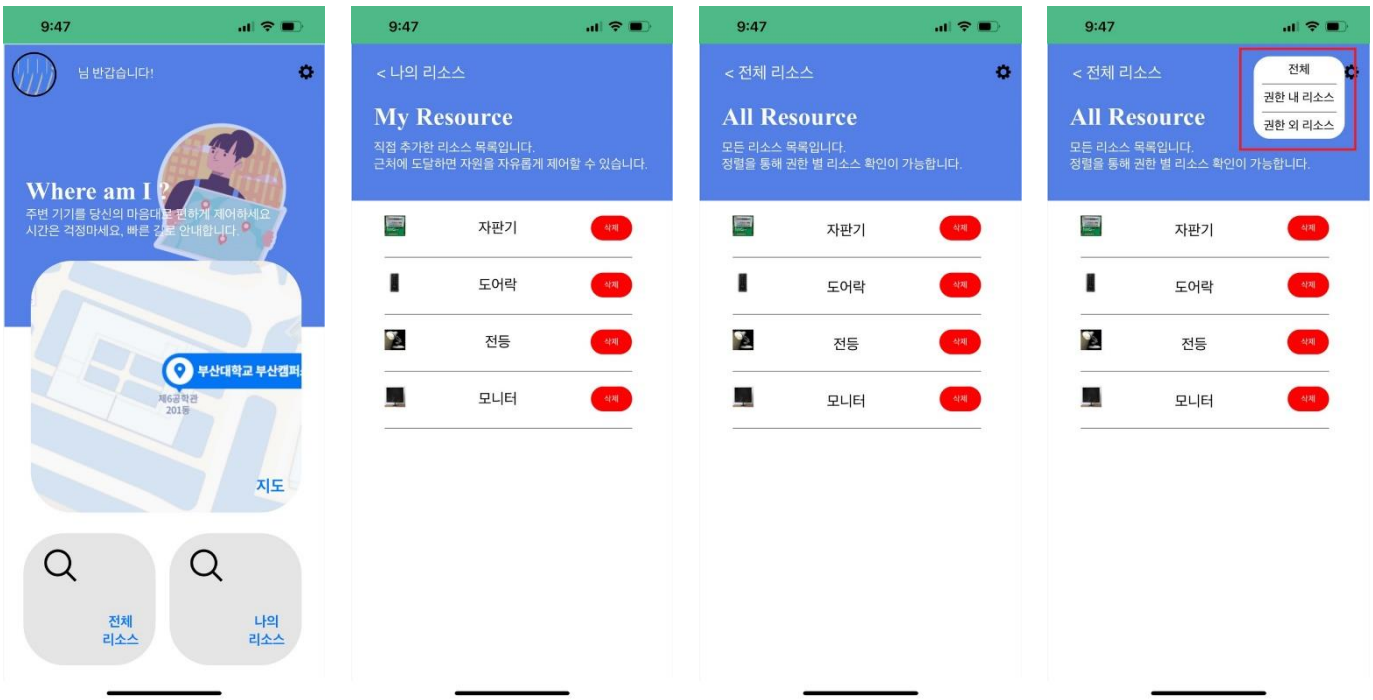
- 초기화면 및 로그인, 회원가입 화면 레이아웃



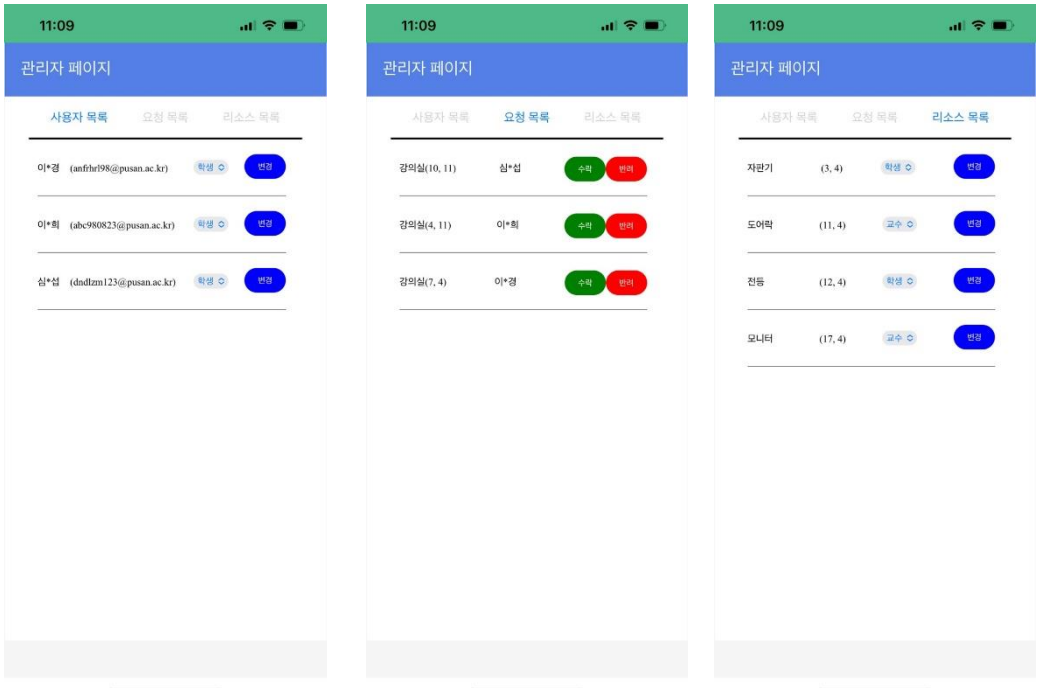
- 이메일, OTP, QR, 기기 인증 화면 레이아웃



- 메인 화면 및 리소스 목록 레이아웃

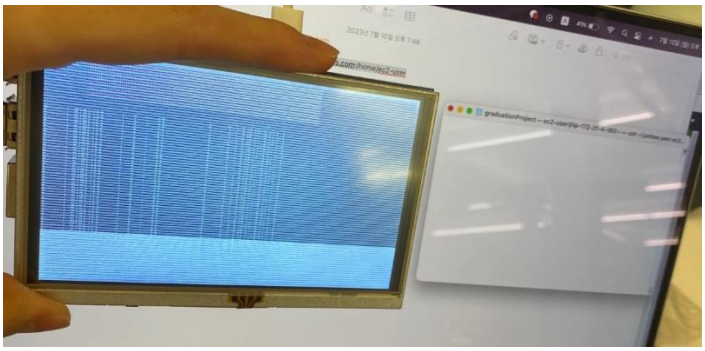


- 관리자 페이지 레이아웃



5-2. Raspberry pi 기반 인증 방식 구현

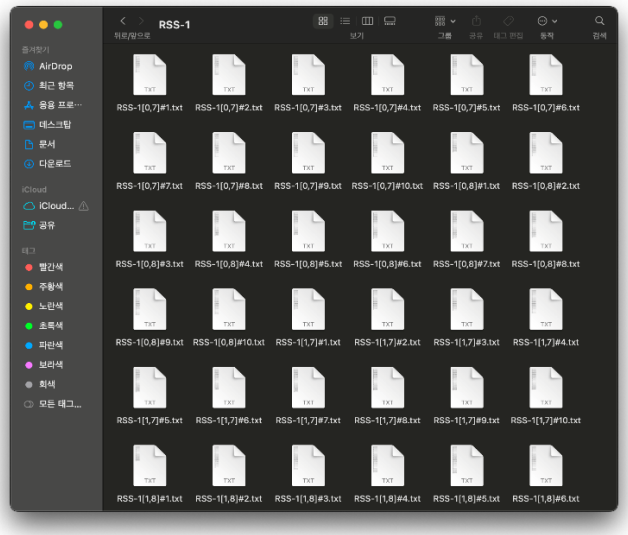
- Raspberry pi 전원을 켰을 때 자동으로 python 파일을 실행해, 서버에 코드 전송이 가능함을 확인
  - ✓ Raspberry pi 에서 .bashrc 파일을 수정하여 전원이 켜지자마자 코드 실행이 가능하다.

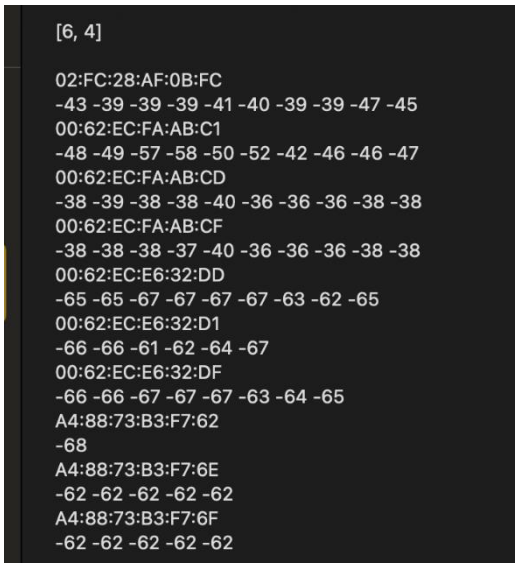


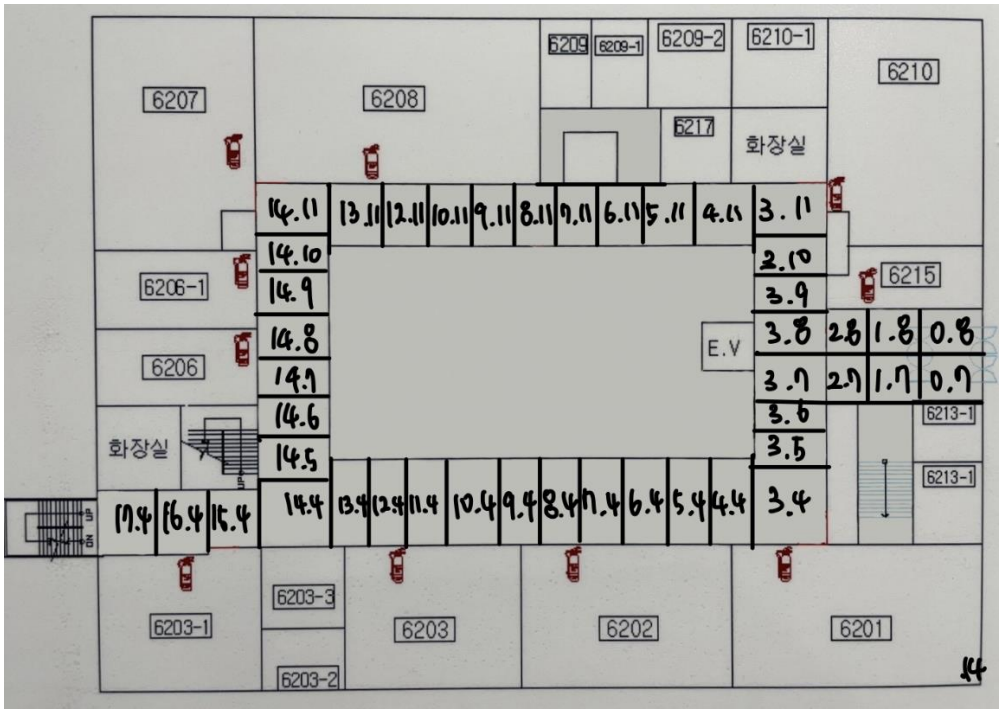
- 기기 인증 화면에서 서버로 Http 요청을 보내, 유효한 기기가 켜져 있는지 인증할 수 있다.
  - ✓ 유효한 기기를 켰을 경우, 서버의 DB 에 해당 사실이 기록된다.
  - ✓ 사용자가 다시 요청을 보내면 유효한 기기가 켜졌는지 검증 후 서비스를 사용할 수 있다.

5-3. Raspberry pi 기반 실내 위치 추정

- 지도 제작을 위해 Raspberry pi 를 통해 RSS 데이터 수집
  - ✓ 정해진 장소(컴퓨터공학관 2 층)를 아래와 같이 좌표로 분할하여 좌표마다 RSS 값을 측정했다.
  - ✓ 이후 좌표마다 측정한 RSS 값을 모두 모아 평균값을 계산했다.

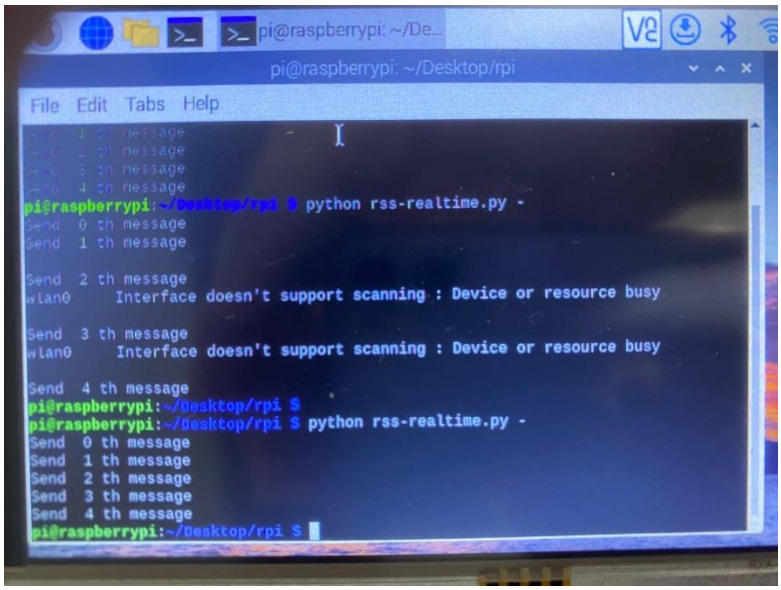






- Raspberry pi 를 통한 실시간 위치 측정 (컴퓨터공학관 2 층)

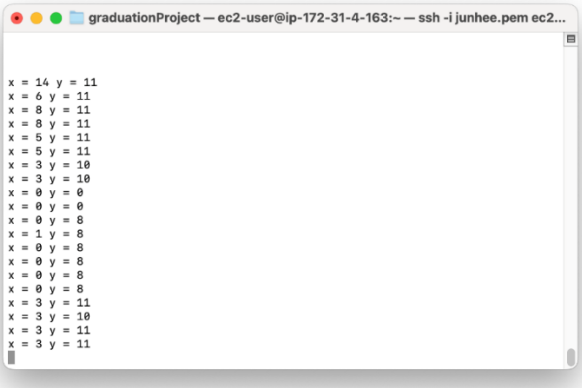
  ✓ 실제로 Raspberry pi 를 휴대하고 다니며, 실시간으로 서버에 RSS 값을 전송한다.

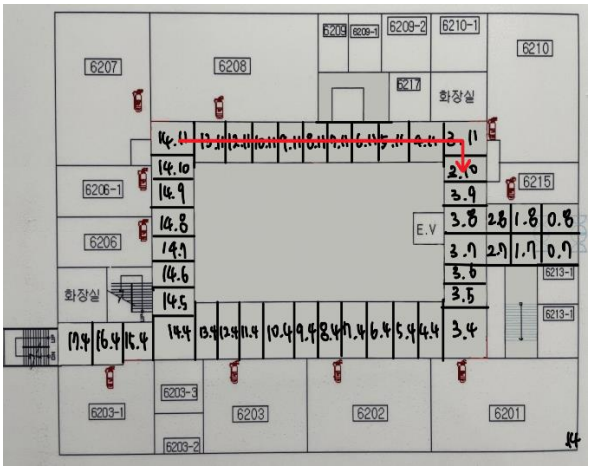


- 서버에 출력된 실제 위치 추정 결과

  ✓ 서버에서 받은 RSS 값과 Radio map 을 비교하여 사용자의 현재 위치를 출력한다.

  ✓ 사용자가 아래 지도의 화살표 방향대로 이동하고 있음을 확인할 수 있다.





5-4. MQTT 기반 서버-기기 간 통신

- Raspberry pi – 서버 간 MQTT 통신 가능 여부 확인 완료
  - ✓ AWS EC2 에서 **Mosquitto** 를 설치하여 **Broker** 로 사용하였다.
  - ✓ 서버에서 **"Hello I'm mosquitto"**를 **"rpi/1" topic** 에 **publish** 하였다.
  - ✓ **Broker(AWS EC2)**가 해당 메시지를 message bus 에 흘려보낸다.
  - ✓ **"rpi/1" topic** 을 **subscribe** 하는 Raspberry pi 에서 해당 message 를 출력됨을 확인했다.
  - ✓ 이는 곧 서버에서 각 기기로 동작 명령이 가능함을 의미한다.