

종합설계 프로젝트 수행 보고서

프로젝트명	스마트 코인 노래방 시스템
팀번호	S3-1
문서제목	수행계획서() 2차발표 중간보고서() 3차발표 중간보고서() 최종결과보고서(O)

2021.06.16

팀원 : 2016150017 박민국(팀장)
2016150026 윤혁진(팀원)
2016150027 이건웅(팀원)
2016150041 최원규(팀원)

지도교수 : 정의훈 교수
지도교수 : 방영철 교수

문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	
2020.12.18	박민국(팀장)	1.0	수행계획서	최초작성
2020.03.04	박민국(팀장)	2.0	2차발표자료	설계서추가
2020.05.06	박민국(팀장)	3.0	3차발표자료	시험결과추가
2020.06.16	박민국(팀장)	4.0	최종결과보고서	시험결과 수정

문서 구성

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

이 문서는 한국산업기술대학교 컴퓨터공학부의
 “종합설계” 교과목에서 프로젝트 “스마트 코인노래방
 시스템”을 수행하는 (S3-1, 박민국, 윤혁진, 이건웅, 최원규)들이
 작성한 것으로 사용하기 위해서는 팀원들의 허락이 필요합니다.

목 차

I. 서론

1. 작품선정 배경 및 필요성
2. 기존 연구/기술동향 분석
3. 개발 목표
4. 팀 역할 분담
5. 개발 일정
6. 개발 환경

II. 본론

1. 개발 내용
2. 문제 및 해결방안
3. 시험시나리오
4. 상세 설계
5. Prototype 구현
6. 시험/ 테스트 결과
7. Coding & DEMO

III. 결론

1. 연구 결과
2. 작품제작 소요재료 목록

참고자료

I. 서론

1.1 작품 선정 배경 및 필요성

우리나라에 노래방이 처음 생긴 시기는 약 1991년 일본과 인접한 부산의 한 오락실에서 개설되었으며, 이후 폭발적인 인기를 얻으며 전국적으로 빠르게 확대되기 시작했다. 그러나 노래방을 즐길 수 있는 조건 일정 이상의 비용, 기본 시간이라는 것이 존재하면서 노래방을 찾는 고객들의 기준이 한정적일 수 밖에 없었다. 그러나 이를 깬 코인 노래방이 등장하였다. 최소 5분에서부터 원하는 시간만큼 곡을 부를 수 있었고 가격도 300원부터 자신이 원하는 만큼 즉 기준선이 노래방에 비해 많이 낮아졌다.

또한 사회, 경제적 트렌드가 공동체 생활보다는 혼자만의 문화생활을 중요시하는 방향으로 흘러가자, 혼자서 놀 수 있는 즉 1인 문화 공간이 유명세를 타기 시작하였다. 이러한 점에서 코인 노래방은 좋은 공간이었다. 1000~1500원만 투자하면 노래 4곡을 부르며 스트레스를 풀 수 있는 시간이 생기고, 이를 가성비 좋은 취미로 여기는 사람들이 많이 생겨났다.

이러한 배경들로 현재 우리나라에는 약 740만 개의 코인 노래방들이 생겼을 정도로 코인 노래방은 대중적인 시설이 되었고, 많은 사람들이 즐기는 문화 공간이 되었다.

이렇게 많은 사람이 이용하는 만큼 공통적으로 등장하는 불편한 점들이 생겨났다. 바로 대부분의 코인 노래방 서비스는 현금 결제만 지원된다는 점과 자신이 어떤 곡을 부를지 고민하는 시간이 길어진다는 점이다. 이를 해결하기 위하여 핸드폰 결제 시스템, 노래 리스트 불러오는 서비스를 제공해주는 스마트 코인 노래방 어플을 제작하였다.

1.2 기존 연구/기술 동향 분석

현재 노래방의 IOT기기와 어플의 연동 기술이 존재하는지 어플 다운로드 플랫폼을 먼저 살펴 보았다.

우선 코노GO, 코인 노래방과 관련하여 구글 플레이스토어에 검색되는 유일한 애플리케이션이다. 이 어플은 주변 코인 노래방의 위치를 검색하여 지도에 표현해주고 따로 결제기능은 존재하지 않고 스마트폰에서 노래 검색, 음향 조절하는 기능정도만 갖추고 있다.

TJ 노래방(가정용), TJ 가정용 노래방 반주기를 원격 제어할 수 있고 반주기를 원격으로 제어하는 기능을 가지고 있다. 이 어플은 가정용 모델 전용이므로 앱 내에서 결제 시스템을 구축하지는 않았다.

블루투스 비컨 반주기, 반주기 내에 블루투스 비컨을 부착하여 서버-반주기, 서버-단말기, 단말기-반주기 사이의 상호 통신을 할 수 있는 제품이다. 서버에게 반주기 위치 정보와 반주기의 사용 상태를 확인할 수 있고 단말기를 통해 사용자의 정보 확인이 가능하다.

이러한 기술 동향 분석을 통해 현재까지 어플을 통해 결제하는 시스템과 스마트폰의 노래 목록을 가져오는 기능의 보완이 필요하다는 것을 알 수 있었다.

1.3 개발 목표

- 현재 코인노래방의 현금 결제 시스템을 보완하여 어플을 통하여 결제될 수 있도록 한다.
- 사용자가 곡을 좀 더 빠르고 고민 없이 선택하기 위해 자신의 음악 재생 목록을 불러와, 선호하는 곡의 목록을 불러온다.
- 기존 노래방 전용 리모컨의 불편함을 개선하여 더욱 향상된 제어 환경을 제공한다.
- 노래방 기기의 신곡 업데이트를 자동화 하여 서버에서 계속 노래가 추가될 때 마다, 노래를 다운로드 한다.
- 위 기능들을 통하여 궁극적으로 통하여 현금 부족 문제, 곡을 선택하는 시간 낭비 등의 문제점을 보완한다.

1.4 팀 역할 분담

	박민국	윤혁진	이건웅	최원규
자료수집	<ul style="list-style-type: none"> BLE 설정 HTTP 통신 방법 appyom 라이브러리 	<ul style="list-style-type: none"> QT 라즈베리파이 내 데이터 관리 서버-앱-디바이스 간 통신 	<ul style="list-style-type: none"> HTTP 통신 방법 Node.js 서버 User 세션 관리 	<ul style="list-style-type: none"> BLE 연결 및 데이터 전송 zxing API HTTP 통신 방법
설 계	<ul style="list-style-type: none"> 앱-디바이스 간 데이터 전송 서버-디바이스 간 데이터 전송 라즈베리파이 주변기기 제어 	<ul style="list-style-type: none"> 라즈베리파이 GUI 라즈베리파이 내부 데이터 관리 및 출력 	<ul style="list-style-type: none"> 서버 및 DB 결제 시스템 DB를 통한 전체 데이터 흐름 	<ul style="list-style-type: none"> 앱-디바이스 간 데이터 전송 QR 인식 앱 UI 및 주요 기능
구 현	<ul style="list-style-type: none"> 라즈베리파이 BLE 설정 주요 반주기 기능 연결된 마이크, 스피커 제어 	<ul style="list-style-type: none"> 라즈베리파이 GUI 내부 데이터 처리 구현 수신값을 통한 기기제어 	<ul style="list-style-type: none"> 웹 서버 구현 HTTP를 통한 데이터 송수신 세션 및 사용자 인증 	<ul style="list-style-type: none"> 앱 UI BLE, HTTP를 이용한 앱 기능 사용자 인증
테스트	<ul style="list-style-type: none"> 전체 동작 / 기능 테스트 유지보수 			

1.5 개발 일정

	11월	12월	1월	2월	3월	4월	5월	6월	7월	8월
주제선정 및 자료조사										
요구사항 정의 및 분석										
시스템 설계 및 상세 설계										
구 현										
테스트										
문서화 및 발표										
종합설계 최종제품 데모										

1.6 개발 환경

구분		내용
S/W	OS	Windows 10, Ubuntu 18.04 LTS, Raspbian
	개발 환경	Android Studio 4.1
	개발 언어	Kotlin, Python, JavaScript
	프레임워크	Node.js Android(API 23)
	DB	MySQL
	기타사항	aupyom, zxing, BLE, QT
H/W	마이크로컨트롤러	라즈베리파이 3B
	주변기기	USB 라즈베리 스피커 브리츠 마이크 MC-1500B1K 라즈베리파이 7인치 터치스크린
	모바일 디바이스	LG G7

II. 본론

2.1 개발 내용

Raspberry

- Raspberry와 블루투스 통신 문자열을 받아서 노래를 검색
- 노래 예약/우선 예약, 음정 조절, 마이크 볼륨조절, 노래 속도 조절의 노래방 기본기능
- 사용자 잔여 곡을 알거나 신곡 업데이트를 위한 서버 연동

App

- 사용자 로그인, 회원가입/탈퇴 기능
- Raspberry와 MAC주소를 이용한 블루투스 페어링 기능
- 포인트를 이용한 결제 기능
- 회원관리를 위한 서버 연동
- 기기 제어를 위한 리모컨 기능

Server

- 사용자와 해당하는 결제 정보를 불러오기 위한 APP과 통신
- 신곡 업데이트와 사용자를 확인하기 위한 Raspberry와 통신
- 회원정보 기기구분 데이터 등을 저장하기 위한 데이터베이스 연동

2.2 문제점 및 해결 방안

여러 기기가 있는 환경에서 자신이 사용할 기기를 블루투스로 찾는 문제

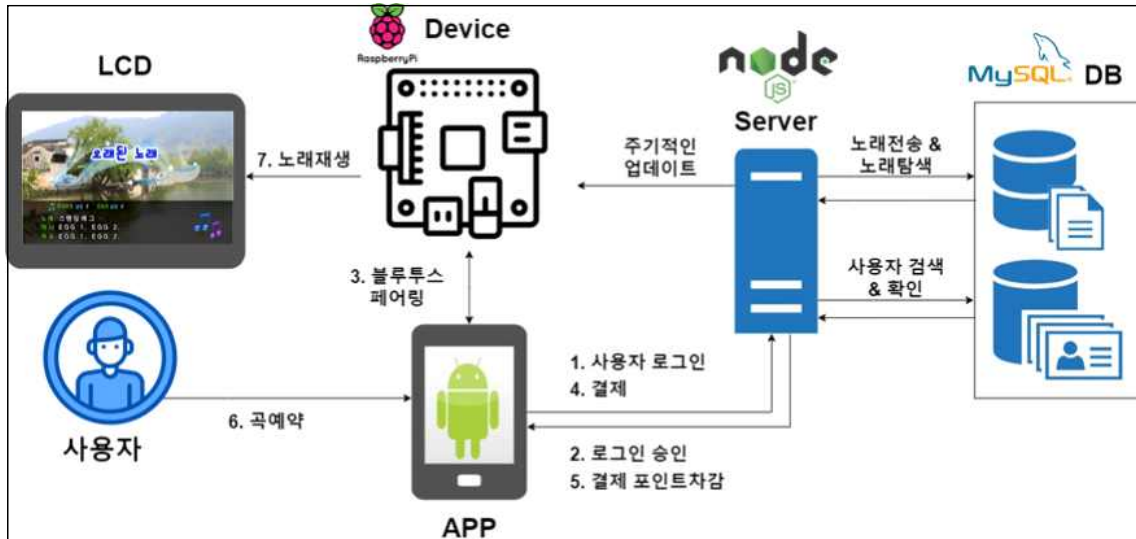
- QR코드 스캔으로 사용자가 사용할 기기와 연동한다.

서버의 다중 회원들의 접속을 구현하는 문제

- 세션 유지가 쉬운 카카오 로그인 API를 사용하였다.

2.3 시험시나리오

* 전체적인 시나리오 구성도

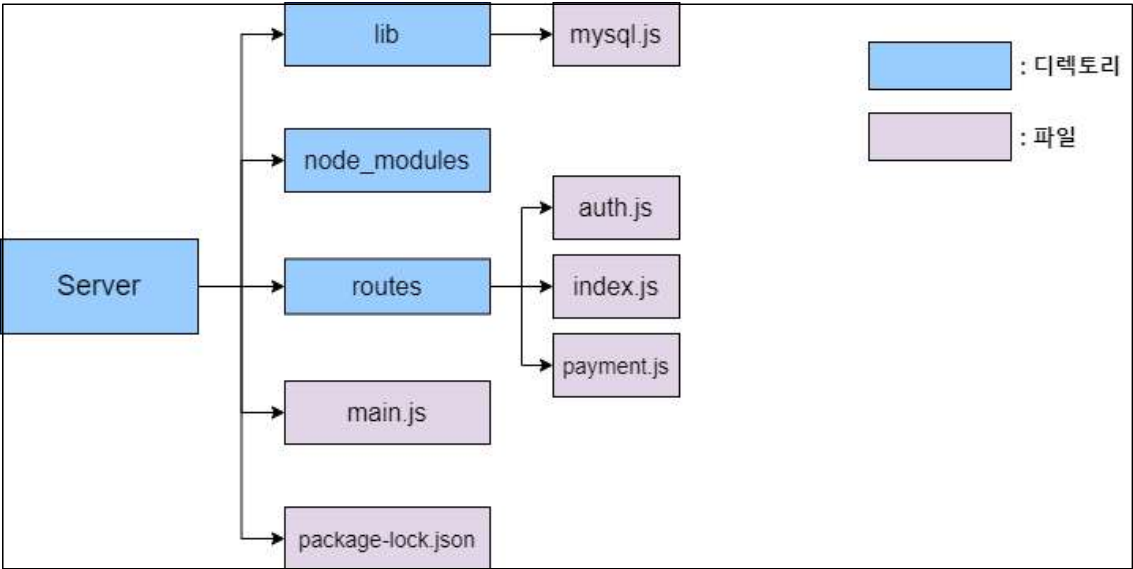


- 사용자는 노래방에 방문하여 기존에 설치한 APP을 실행한다.
- APP의 로그인 기능을 사용하여 로그인 정보를 서버로 전달하여 주고 서버는 받은 로그인 정보를 이용해 확인 후 로그인 승인을 하여준다.
- APP의 QR코드 인식을 활용하여 노래방 디바이스 장치와 블루투스 페어링을 한다.
- 사용자에게 제공되는 버튼UI 화면을 통해 노래방 디바이스 장치에게 하고자 하는 동작을 전달한다.
- 노래방 디바이스는 사용자의 명령에 따라 노래재생/예약/검색과 같은 기능을 수행한다.



2.4 상세 모듈

<server>



위 그림은 스마트 코인 노래방 어플리케이션의 서버 상세 설계표이다. 우선 서버에는 웹 통신과 메인 컨트롤러 역할을 하는 main.js 파일이 있고 서버 실행에 필요한 모듈을 포함하는 node_modules가 있다. 또한 DB와의 연결을 위한 파일이 들어있는 lib 디렉터리가 존재하고, routes 디렉터리에는 각각 로그인, 데이터 전송, 결제를 맡고있는 model들이 파일들이 존재한다.

Database 모듈 상세 설계	
테이블 명	컬럼
userInfo	id(int), email(varchar), passwd(varchar), username(varchar), point(int), salt(varchar)
video	id(int), name(varchar), dir(varchar), Date(date)

위 표는 데이터베이스에서 활용되는 테이블의 상세 설계이다. 테이블은 총 2개로 각각의 테이블은 어플리케이션에서 정보를 받아오거나, 라즈베리 연동을 통해 데이터가 관리된다.

userInfo 테이블은 회원가입된 사용자들의 정보가 들어있는 테이블이다. 튜플은 id, email, 비밀번호, 이름, 적립 포인트, 난수(비밀번호 암호화)로 이루어져 있다.

video는 노래를 부를 때 재생되는 동영상의 디렉터리 위치를 담고 있는 테이블이다. 튜플은 id, 이름, 디렉터리 위치, 날짜로 이루어져 있다.

웹 서버 상세 설계	
파일 명	함수 이름
index.js	router.get('/',function(req,res)() router.get('/download',function(req,res)() res.download(dir,function(err) db.query('', function({})

위 표는 웹 서버의 상세 설계이다. 우선 main.js의 메소드는 다음과 같다. app.listen(80,function());은 포트 번호 80을 통해 express를 통해 서버를 실행시키는 함수이다. app.use('/',indexRouter);는 '/'를 상위 경로로 가지는 모든 uri를 indexRouter이라는 모듈 안에 있는 메소드들로 실행시키겠다는 함수이다. app.use('/auth',authRouter);는 '/auth'를 상위 경로로 가지는 모든 uri를 authRouter이라는 모듈 안에 있는 메소드들로 실행시키겠다는 함수이다. app.use('/payment',paymentRouter);는 '/payment'를 상위 경로로 가지는 모든 uri를 paymentRouter이라는 모듈 안에 있는 메소드들로 실행시키겠다는 함수이다. mysql.js의 메소드인 mysql.createConnection()은 db와 연동 할 수 있게 하는 커넥션을 생성하는 것이고 db.connect()는 커넥션을 통해 db와 연결하는 메소드이다. index.js이 메소드는 다음과 같다. router.get('/',function(req,res)()은 '/' 경로를 get 방식으로 접근하는 클라이언트에게 callback함수인 function을 실행시키는 메소드이다.

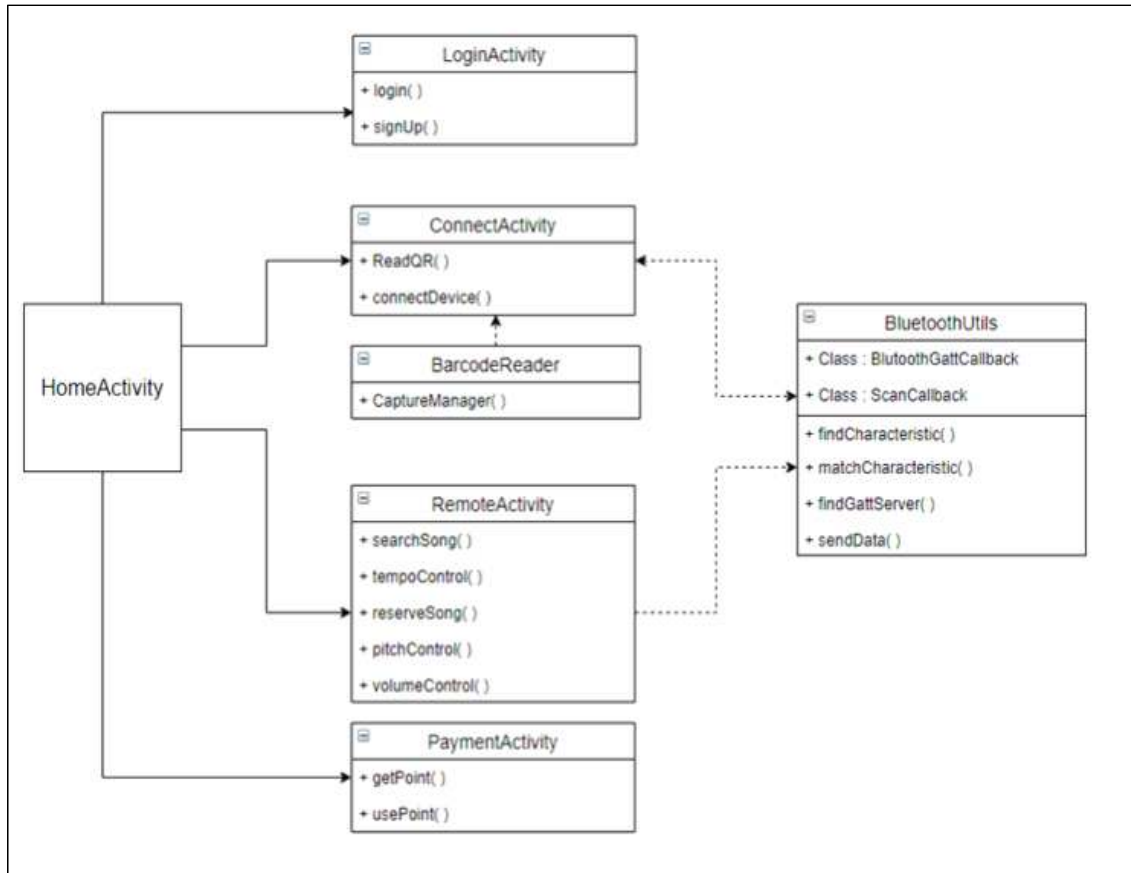
router.get('/download',function(req,res)()은 '/download/ 경로를 get 방식으로 접근하는 클라이언트에게 res.download(dir,function(err) 메소드를 통해 매개변수인 dir 위치에 있는 동영상 파일을 전송할 수 있게 한다. db.query('', function({})는 db와 연동되어 쿼리문을 사용할 수 있게하는 함수이다.

auth.js는 로그인과 관련된 메소드들이 모여있는 파일이다. 메소드들을 살펴보면 router.post('/login',function(req,res)는 '/login' 경로에 post방식으로 접근되어 클라이언트가 로그인을 수행할 수 있게하는 메소드이다. router.post('/logout',function(req,res)는 '/logout' 경로에 post방식으로 접근되어 클라이언트가 로그아웃을 수행할 수 있게 하는 메소드이다. router.post('/expire',function(req,res)는 클라이언트가 회원 탈퇴를 진행할 때 db에서 회원 정보를 삭제하는 기능을 수행하는 메소드이다.

router.post('/register',function(req,res)는 클라이언트가 회원가입을 할 때 db에 회원 정보를 삽입하는 메소드이다.

payment.js는 결제와 관련된 메소드가 모여있는 파일이다. 메소드를 살펴보면 router.get('/pay',function(req,res)는 클라이언트가 결제를 진행할 때 db에 회원 정보에 포인트를 추가해주고 결제 완료 메시지를 전송해주는 기능을 하는 메소드이고, db.query('', function({})는 db와 연결되어 쿼리문을 수행할 수 있게하는 메소드이다.

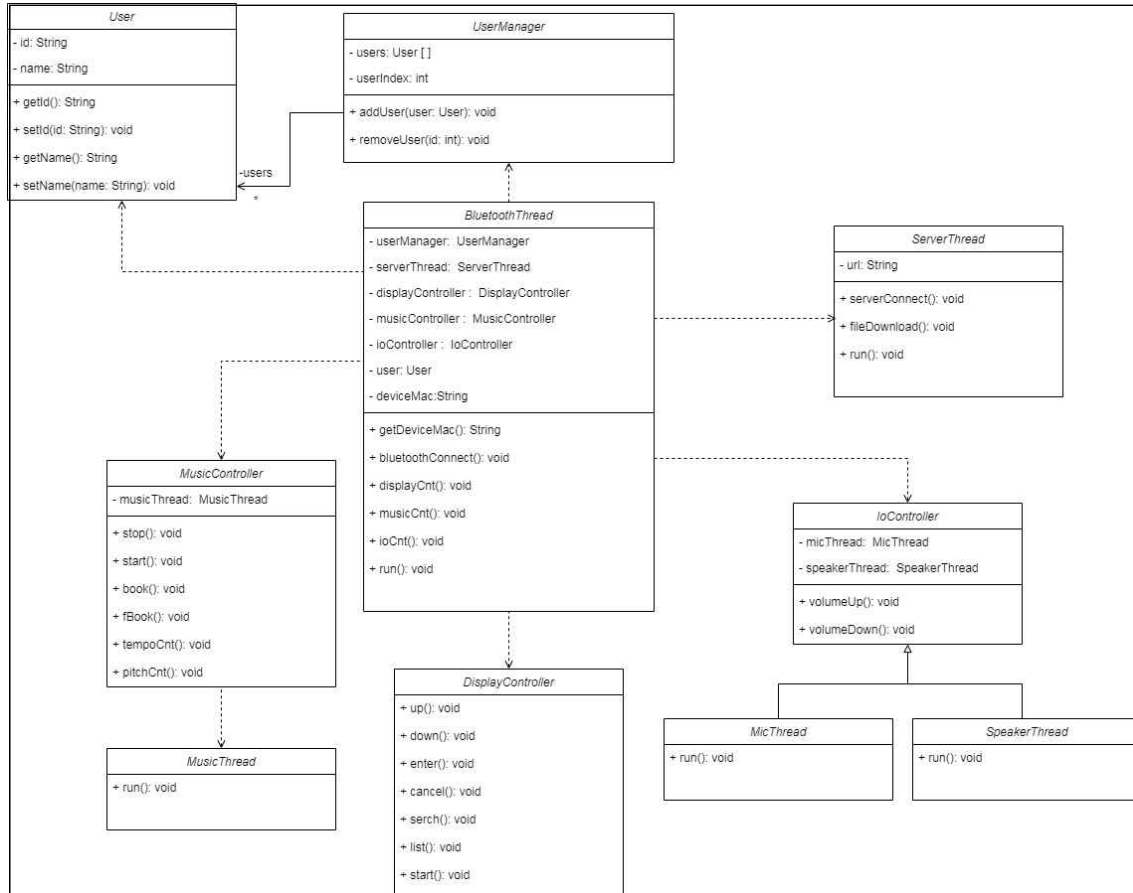
<application>



위 표는 애플리케이션의 상세 설계이다. 우선 LoginActivity의 메소드는 다음과 같다. login()메소드는 사용자의 아이디, 비밀번호를 확인하여 로그인하는 메소드이고 signUp() 메소드는 사용자의 회원가입을 수행하고 카카오 API를 활용한 이메일을 통해 회원가입을 수행하는 메소드이다. ConnectActivity와 BluetoothUtils의 메소드는 다음과 같다. ReadQR()은 QR 코드를 스캔하고 BarcodeReader 클래스로 튜닝하는 기능을 맡고 있다. connectDevice()는 특정 MAC 주소를 가지는 디바이스의 스캔과 연결을 수행하는 메소드이고 BluetoothUtils: BLE 통신에 필요한 각종 메소드를 정의하며 BLE 연결에 필요한 각종 메소드를 제공하는 BluetoothGattCallback 클래스와 ScanCallback 클래스를 내부 클래스로 사용되는 메소드이다.

RemoteActivity와 BluetoothUtils의 메소드는 다음과 같다. searchSong() 반주기가 제공하는 곡 검색, reserveSong() 반주기가 제공하는 특정 곡을 예약, tempoControl() 재생중인 곡의 템포 조절, volumeControl() 재생 중인 곡의 볼륨 조절, pitchControl() 재생중인 곡의 음정 조절하는 기능을 맡고 있다. sendData() 메소드는 BluetoothUtils에 정의된 메소드로 예약이나 검색하려는 곡명이나 제어 신호를 String 타입으로 BLE 통신을 이용해 반주기에 전송한다. PaymentActivity의 메소드는 getPoint()는 사용자 포인트 구매 메소드이고 usePoint()는 사용자가 반주기를 사용하며 곡 예약 시 발생하는 포인트 사용을 제어하는 메소드이다.

<RaspberryPI>



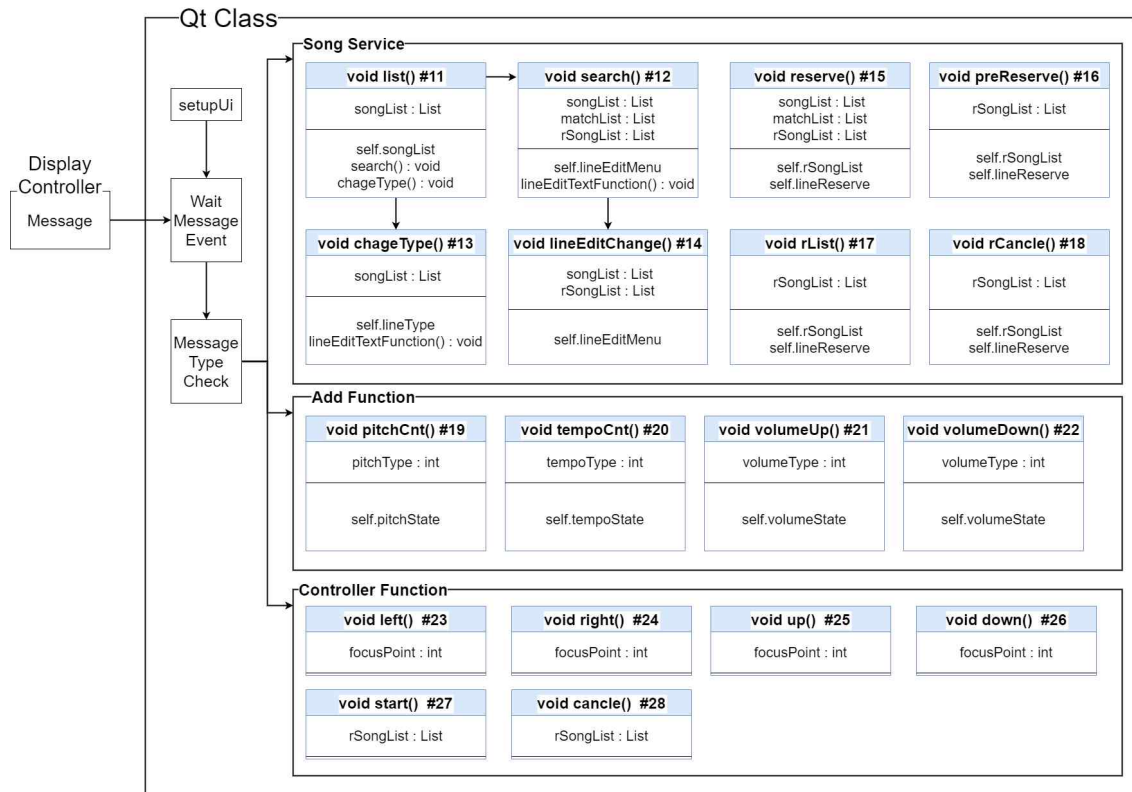
위 표는 라즈베리 파이의 상세 모듈 설계이다. User 클래스는 블루투스에 연결한 유저들의 정보(User class)를 관리하는 클래스이고 이를 addUser, removeUser 함수로 저장하고 삭제를 수행한다. IOController는 raspberry에 연결된 마이크와 스피커의 볼륨을 제어를 맡는 클래스이고 MicThread 함수를 통해 raspberry에 연결된 마이크를 실행하고 SpeakerThread 함수는 raspberry에 연결된 스피커를 실행한다.

ServerThread는 서버와의 연동을 맡는 클래스이고 serverConnect()는 서버와 연동, 일정 시간마다 서버에 파일이 업데이트 되는지 확인하는 함수이다. fileDownload() 업데이트된 파일을 다운로드하는 함수이다.

MusicController는 재생되는 노래 제어 기능을 맡는 클래스이다. 메소드는 노래를 정지하는 Stop() 함수, 노래 우선 예약을 수행하는 fBook() 함수, 노래 시작 기능의 start(), 템포 제어 기능의 tempoCnt(), 노래 예약의 book(), 노래 음정을 제어하는 pitchCnt() 함수가 있다.

BluetoothThread 클래스는 블루투스 연결을 대기하고 들어오는 데이터를 적절한 컨트롤러에 전달하는 기능을 맡는 클래스이다. 메소드는 블루투스 연결을 대기하는

bluetoothConnect(), 들어오는 데이터로 디스플레이를 제어하는 displayCnt(), 들어오는 데이터로 노래를 제어하는 musicCnt(), 들어오는 데이터로 마이크/스피커를 제어하는 ioCnt()가 있다.



위는 라즈베리파이 Qt를 활용한 UI 동작별 함수를 정의해 놓은 것이다. 크게 노래관련 서비스와 부가기능, 조작기능으로 나누어 구분하였고 외부로부터 message(int type)를 전달받아 해당 메시지와 매칭되는 번호에 대한 시그널을 발생시켜 함수를 동작합니다.

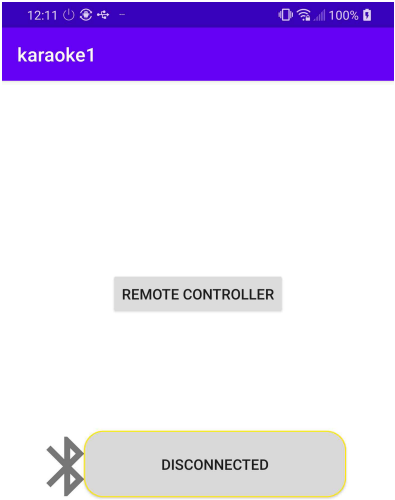
노래관련 서비스의 경우, 검색을 위한 메뉴를 list()함수를 통해 제공하며 메뉴에 있는 텍스트 입력 가능 부분에 검색 단어를 받아 search()함수를 이용하여 노래를 검색합니다. 메시지의 변화에 따른 리스트에 즉각적인 변화를 일으키기 위해 lineEditChange()와 연결하여 실시간 변화를 감지합니다. 이 외의 예약과 관련하여 예약하기[reserve()], 우선예약[preReserve()], 예약목록[rList()], 예약취소[rCandle()] 기능이 있습니다.



부가기능의 경우 기존 노래방시스템에서 사용하는 음정조절[pitchCnt()]과 템포조절[(tempoCnt)], 볼륨조절[volumeUP/Down()] 의 변화를 사용자에게 보여주는 기능을 제공한다.

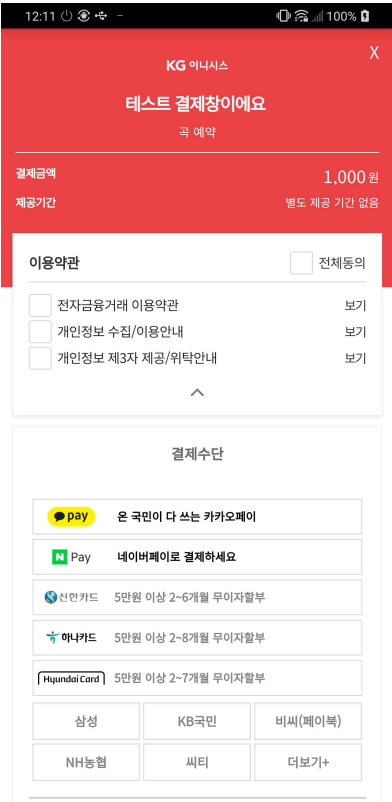
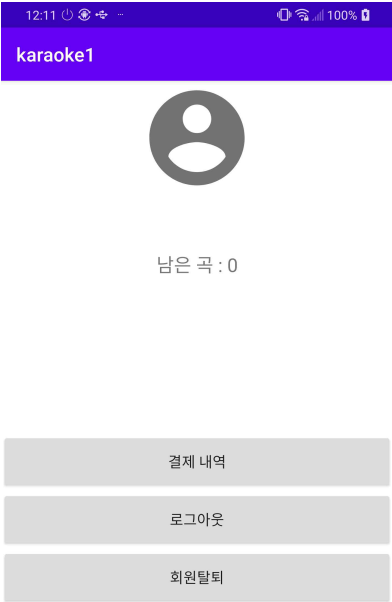
조작기능의 경우 UI와 관련되서 노래시작[strat()], 취소[candle()], 방향키 ‘←’, ‘→’, ‘↑’, ‘↓’ 역할을 할 수 있는 기능을 제공한다.

2.5 프로토타입

2.5.1 APP

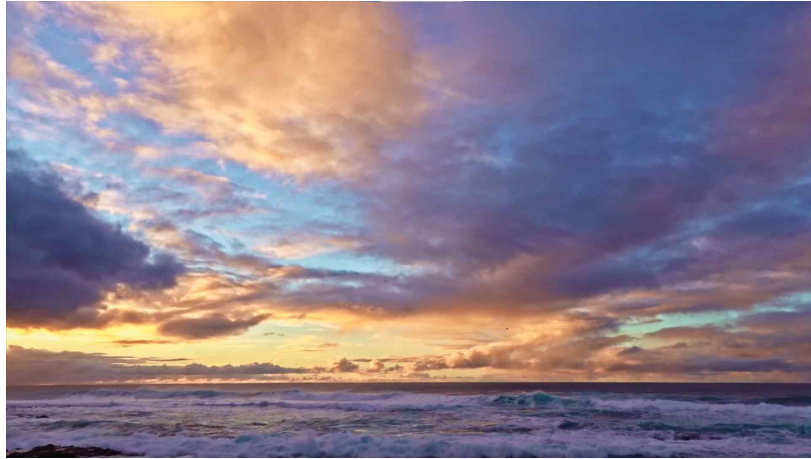
순서	화면	내용
메인 화면		<ul style="list-style-type: none">▪ Kakao ID를 이용한 로그인▪ 반주기 리모컨으로 이동▪ 블루투스 연결을 위한 버튼

<p>블루투스 연결화면</p>		<ul style="list-style-type: none"> ▪ QR 코드 스캐너 호출 ▪ 디바이스 QR 코드 스캔 시 QR 코드의 MAC 주소에 해당하는 디바이스 자동 스캔 및 연결
<p>리모컨 화면</p>		<ul style="list-style-type: none"> ▪ 제목, 가수 별 검색 기능과 검색 결과를 통한 곡 예약 ▪ 예약 내역 확인과 예약 취소 기능 ▪ 템포, 볼륨, 음정 조절 기능 ▪ 반주기 화면 제어 ▪ 마이 페이지, 결제 화면으로 이동

결제화면		<ul style="list-style-type: none">▪ Bootpay API를 이용한 PG결제▪ 결제 내역에 따른 예약 가능 횟수 자동 갱신▪ PG신청을 하지 않았으므로 테스트 결제로 진행(관리자 페이지에서 취소 가능 혹은 24시간 이내 자동 취소)
마이 페이지		<ul style="list-style-type: none">▪ 마이페이지에서 남은 곡 수 확인▪ 결제 내역 확인▪ 로그아웃, 회원탈퇴 기능

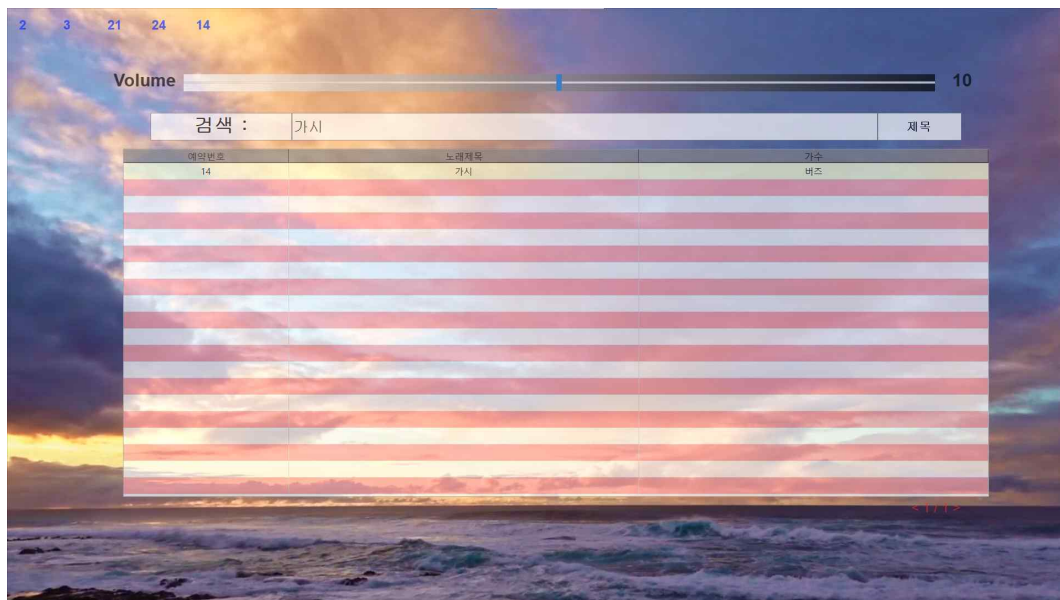
2.5.2 Raspberry

메인 화면



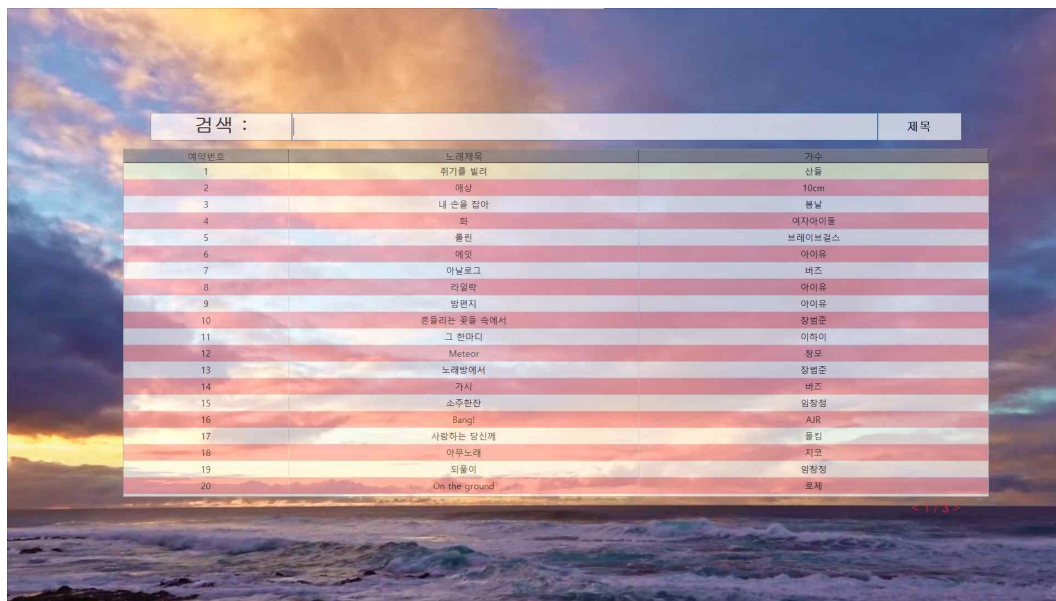
- 반주기 메인 화면 UI

메뉴 리스트 화면 (예약목록 & 검색)



- 노래 예약 및 예약목록 상단 바 표시
- 노래 리스트 검색 기능
- 소리(Volume) & 음정(peach) 조절 확인

메뉴 리스트 화면 / DB 테이블



id	name	dir	Date	singer
1	취기를 불러	C:\Users\82107\Desktop\fileEx	2021-05-01	산들
2	애상	C:\Users\82107\Desktop\fileEx	2021-05-01	10cm
3	내 손을 잡아	C:\Users\82107\Desktop\fileEx	2021-05-01	불날
4	화	C:\Users\82107\Desktop\fileEx	2021-05-01	여자아이들
5	롤린	C:\Users\82107\Desktop\fileEx	2021-05-01	브레이브걸스
6	에잇	C:\Users\82107\Desktop\fileEx	2021-05-01	아이유
7	아날로그	C:\Users\82107\Desktop\fileEx	2021-05-01	버즈
8	라일락	C:\Users\82107\Desktop\fileEx	2021-05-01	아이유
9	밤편지	C:\Users\82107\Desktop\fileEx	2021-05-01	아이유
10	흔들리는 꽃들 속에서	C:\Users\82107\Desktop\fileEx	2021-05-01	장범준
11	그 한마디	C:\Users\82107\Desktop\fileEx	2021-05-01	이하이
12	Meteor	C:\Users\82107\Desktop\fileEx	2021-05-01	창모
13	노래방에서	C:\Users\82107\Desktop\fileEx	2021-05-01	장범준
14	가시	C:\Users\82107\Desktop\fileEx	2021-05-01	버즈

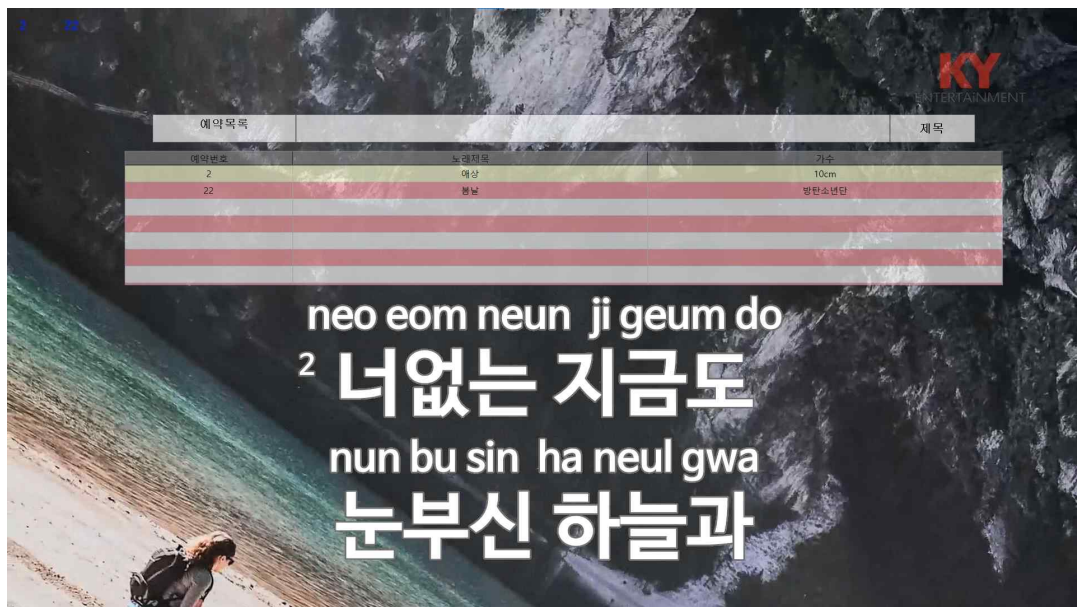
- 노래방 메뉴 리스트 출력
- 노래 리스트 행 선택 기능
- 노래 리스트 페이지 이동 기능

노래 재생 화면



- 노래 재생, 취소
- 소리(Volume) & 음정(peach) 조절 확인
- 노래 예약리스트 확인

노래 재생 화면 (예약목록 확인)



- 노래 예약목록 확인

※추후 개선 사항

- Application
 - UI 개선
 - 리모컨 화면에서의 사용자 편의성 개선
 - 기능 구현에 따른 버튼 추가 혹은 제거
- Raspberry
 - UI 개선 (가시성 & 편리성)

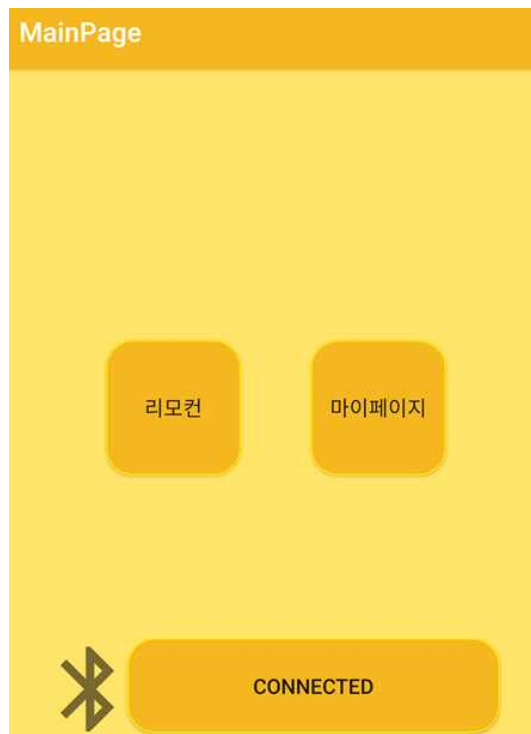
2.6 시험/테스트 결과

<application>

1. QR코드 스캔 후 Bluetooth 기능 정상 동작 확인



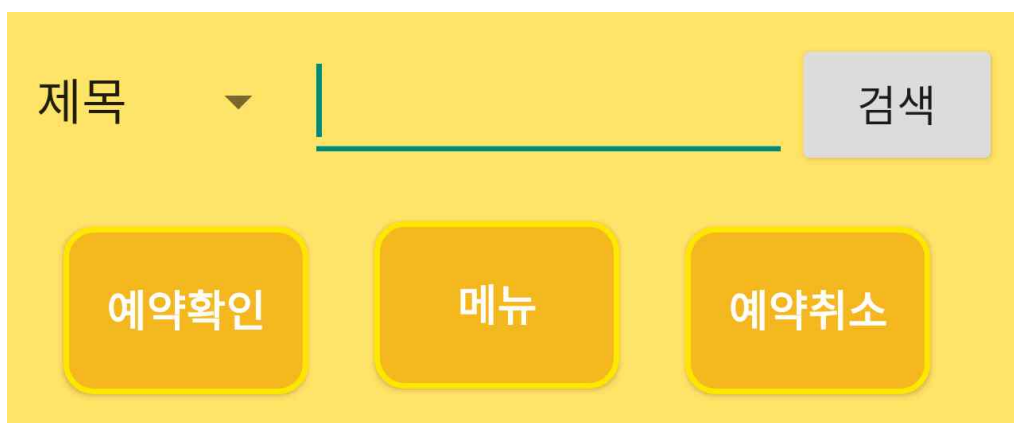
연결을 시도하기 위해 Disconnected 상태인 버튼을 클릭 시 QR 스캐너가 실행되고 MAC 주소 정보를 이용해 디바이스를 스캔하고 스캔 성공 시 Bluetooth 연결을 시도한다.




```
Shell x
GATT application registered
GetAll
returning props
Advertisement registered
value:19
value:19
value:23,1
value:21,1
value:22,1
```

연결이 정상적으로 완료되면 버튼과 토스트 메시지를 통해 연결됨을 확인하고 리모컨의 버튼을 클릭하여 디바이스 콘솔 화면을 통해 의도된 텍스트가 정상적으로 수신됨을 확인하여 블루투스 기능이 정상적으로 동작함을 테스트한다.

2. Application-서버 간 통신 테스트



곡 검색 기능을 통해 서버로부터 검색 키워드에 대한 결과값이 정상적으로 수신되는지 확인한다.

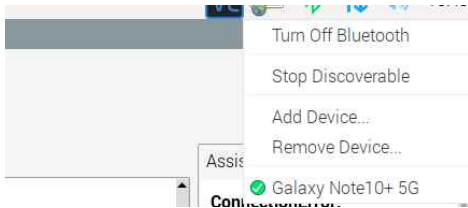
제목	가수
취기를 빌려	산들
애상	10cm
내 손을 잡아	봄날
화	여자아이들
롤린	브레이브걸스
에잇	아이유
아날로그	버즈
라일락	아이유
밤편지	아이유
흔들리는 꽃들 속에서	장범준
그 한마디	이하이

검색 키워드가 아무것도 없을 시 서버에 저장된 모든 곡 정보를 받아온다. 이를 통해 서버와 애플리케이션 간 Http 통신이 정상적으로 동작함을 테스트한다.

<RaspberryPI>

```
>>> %Run dkd_service.py
GetManagedObjects
GATT application registered
GetAll
returning props
Advertisement registered
```

블루투스 기기 connect와 데이터를 기다리는 모습



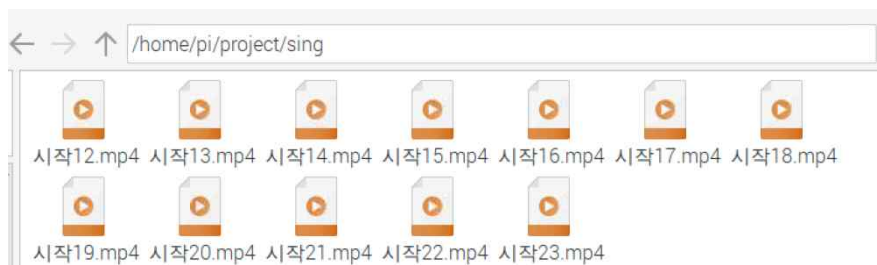
블루투스 연결 성공

```
>>> %Run dkd_service.py
GetManagedObjects
GATT application registered
GetAll
returning props
Advertisement registered
value:test
value:test
value:test
```

연결된 기기로 정해진 데이터를 받아온 모습

```
{"mac": "2", "saveNumber": 35, "changeValue": 0}
0
bool 0
make zip
down load
```

다운로드 대기 중인 상태에서 일정 시간이 되자마자 다운로드를 진행했다



정해진 시간에 다운로드가 잘 되었고 압축파일도 압축해제가 잘 됐다. 또한 남아있는 압축 파일도 삭제되었다.

<결과-1>

예약번호	노래제목	가수
1	취기를 빌려	산들
2	애상	10cm
3	내 손을 잡아	봄날
4	화	여자아이들
5	롤린	브레이브걸스
6	옛것	아이유
7	아날로그	버즈
8	라일락	아이유
9	밤편지	아이유
10	흔들리는 꽃들 속에서	장범준
11	그 한마디	이하이
12	Meteor	창모
13	노래방에서	장범준
14	가시	버즈
15	소주한잔	임창정
16	Bang!	AJR
17	사랑하는 당신께	폴킴
18	아무노래	지코
19	되돌이	임창정
20	On the ground	루제

<결과-2>

검색 :	아	제목
예약번호	노래제목	가수
3	내 손을 잡아	봄날
7	아날로그	버즈
18	아무노래	지코

<결과-1> 서버 요청 데이터

```

제목
[
  {
    RowDataPacket {
      id: 1,
      name: '취기를 빌려',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '산들'
    },
    RowDataPacket {
      id: 2,
      name: '애상',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '10cm'
    },
    RowDataPacket {
      id: 3,
      name: '내 손을 잡아',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '봄날'
    },
    RowDataPacket {
      id: 4,
      name: '화',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '여자아이들'
    },
    RowDataPacket {
      id: 5,
      name: '롤린',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '브레이브걸스'
    },
    RowDataPacket {
      id: 6,
      name: '옛것',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '아이유'
    },
    RowDataPacket {
      id: 7,
      name: '아날로그',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '버즈'
    },
    RowDataPacket {
      id: 8,
      name: '라일락',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '아이유'
    },
    RowDataPacket {
      id: 9,
      name: '밤편지',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '아이유'
    },
    RowDataPacket {
      id: 10,
      name: '흔들리는 꽃들 속에서',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '장범준'
    },
    RowDataPacket {
      id: 11,
      name: '그 한마디',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '이하이'
    },
    RowDataPacket {
      id: 12,
      name: 'Meteor',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '창모'
    },
    RowDataPacket {
      id: 13,
      name: '노래방에서',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '장범준'
    },
    RowDataPacket {
      id: 14,
      name: '가시',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '버즈'
    },
    RowDataPacket {
      id: 15,
      name: '소주한잔',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '임창정'
    },
    RowDataPacket {
      id: 16,
      name: 'Bang!',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: 'AJR'
    },
    RowDataPacket {
      id: 17,
      name: '사랑하는 당신께',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '폴킴'
    },
    RowDataPacket {
      id: 18,
      name: '아무노래',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '지코'
    },
    RowDataPacket {
      id: 19,
      name: '되돌이',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '임창정'
    },
    RowDataPacket {
      id: 20,
      name: 'On the ground',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '루제'
    }
  ]

```

<결과-2> 서버 요청 데이터

```

제목 아
[
  {
    RowDataPacket {
      id: 3,
      name: '내 손을 잡아',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '봄날'
    },
    RowDataPacket {
      id: 7,
      name: '아날로그',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '버즈'
    },
    RowDataPacket {
      id: 18,
      name: '아무노래',
      dir: 'C:\Users\82107\Desktop\fileEx',
      Date: 2021-04-30T15:00:00.000Z,
      singer: '지코'
    }
  ]

```

서버를 통해 노래 리스트 목록을 받아오는 부분이다. <결과-1>의 경우 전체 노래 리스트를 불러오는 경우로서, JSON 형식으로 전송된 노래 리스트를 받아 테이블에 필요한 id, name, singer를 추출하여 채워 넣어준다. 특정 단어를 포함한 검색 요청 시, 해당 단어가 포함된 노래 리스트를 찾아 보내주게 된다. <결과-2>의 경우 '아'라는 글씨를 요청하였고 서버는 제목에 '아'라는 단어가 포함된 노래들을 찾아 이를 JSON 형태로 보내주게 된다.

2.7 Coding & DEMO

<server>

```
var express = require('express');
var router = express.Router();
const db = require('../lib/mysql');

router.get('/',function(req,res){
  res.send('auth');
});
router.post('/register',function(req,res){
  console.log(['레지스터'])
  req.on('data',function(data){
    inputData = JSON.parse(data);
    req.on('end',function(){
      name=inputData.name; // 변수 확인 후 수정 필요
      email = inputData.email; // 수정 필요
      db.query('insert into userInfo (email,username,point) values(?,?,?)',[email,name,0],function(err,result){
        if(err){
          if(err.code === "ER_DUP_ENTRY"){ //id 중복
            res.send("이미 가입되어있는 email입니다.");
          }else{
            console.log(err);
            res.send('회원가입 도중 오류가 발생했습니다. 다시 시도해주세요');
          }
        }else{
          res.send('회원가입이 완료되었습니다.');
```

```
        }
      })
    })
  })
});
module.exports = router;
```

- 어플로부터 회원 정보를 받아와서 만약 DB에 유저 정보가 없을 시에 회원 가입을 진행하여 DB에 값을 추가한다. 탈퇴를 어플에서 시도하면 서버에서는 유저의 이메일을 받아와서 DB에 그에 맞는 정보를 삭제하여 서버에 완료되었다는 메시지를 전달한다.

```

router.get('/d',function(req,res){
  console.log('down')]
  res.download('videos.zip',async function(err){
    await fs.unlinkSync('videos.zip');
    console.log('삭제 완료');
  });
});
})
router.post('/invest',function(req,res){ //mac을 통해 업데이트가 필요한지 확인
  var post = req.body;
  var mac = post.mac;
  console.log(mac);
  db.query('select * from device where mac = ?', [mac],function(err,result){
    if(err){
      console.log(err);
    }else{
      console.log(result[0]);
      res.send(result[0]);
    }
  });
});
})

```

- 라즈베리 파이에서 맥 주소를 전송하고 DB에 값을 통해 곡을 업데이트 상태를 받아와 라즈베리 파이에 전송해준다. 만약 업데이트가 필요하면 '/d' url을 통해 라즈베리파이에서 업데이트 곡 정보인 zip 파일을 다운받는다.

```

router.post('/download', function(req,res){ //라즈베리한테 추가된 곡 개수를 보내는 url
  var post = req.body;
  var mac = post.mac;
  console.log(mac)
  pool.query('select * from device where mac = ?', [mac]).then(devices =>{
    pool.query('select * from video order by id desc limit 1').then(videos =>{
      try{
        len = videos[0][0].id;
        sub = len-devices[0][0].saveNumber;
        console.log('보내야 할 곡 개수:', sub);
        pool.query('select * from video where id >?', len-sub).then(sendV => {
          for(var i =0; i<sub; i++){
            var dir = `C:\\Users\\82107\\Desktop\\fileEx\\${sendV[0][i].name}.mp4`;
            zip.file(`${sendV[0][i].name}.mp4`, fs.readFileSync(dir));
            console.log(dir);
          }
          var data = zip.generate({type:"uint8array"});
          console.log("zip 생성")
          fs.writeFile('videos.zip',data,'binary',async function(err){
            if(err){
              console.log("에러", err);
              res.send(err);
            }
          })
          pool.query('update device set saveNumber =?,changeValue=? where mac =?', [len,1,mac]).then(result =>{
            try{
              console.log("수정 완료");
            }catch(e){
              console.log(e)
            }
          })
          res.send('create success');
        })
      }catch(e){
        console.log(e);
      }
    })
  })
})

```

- 라즈베리파이에서 '/invest' url을 통해 업데이트 상태를 받아와 업데이트할 곡의 목록이 있으면 서버의 하드디스크에서 곡을 받아와 zip 파일로 압축시켜 저장하고 url에 접속한 mac 주소에 대해 DB에 업데이트된 총 곡 수를 업데이트시킨다.

```

var express = require('express');
const db = require('../lib/mysql');
var router = express.Router();
router.post('/search', function(req,res){
  console.log("search")
  req.on('data',function(data){
    inputData = JSON.parse(data);
    req.on('end',function(){
      type = inputData.type; // 수정 필요
      text = inputData.text;
      console.log(type,text);
      if (type === "제목"){
        db.query('select * from video where UPPER(name) like ?', ['%${text}%'],function(err,result){
          if(err){
            console.log(err);
          }else{
            sendResult = [];
            temp=[];
            var index = 0
            while(index<=result.length){
              if(index != 0 & (index % 20==0 || index == result.length)){
                sendResult.push(temp);
                temp=[];
              }
              temp.push(result[index]);
              index++;
            }
            console.log(sendResult)
            res.send(sendResult);
          }
        })
      }
    })
  })
})

```

```

      }else if (type === "가수"){
        db.query('select * from video where singer like ?', ['%${text}%'],function(err,result){
          if(err){
            console.log(err);
          }else{
            sendResult = [];
            temp=[];
            var index = 0
            while(index<=result.length){
              if(index != 0 & (index % 20==0 || index == result.length)){
                sendResult.push(temp);
                temp=[];
              }
              temp.push(result[index]);
              index++;
            }
            res.send(sendResult);
          }
        })
      }
    })
  })
  }else{
    res.send('형식이 잘못 되었습니다.')
  }
}
})
});
module.exports = router;

```

- 어플에서 곡 정보를 가수, 제목을 통해 검색할 때 이에 대한 정보를 서버에 전송하고 서버는 이를 받아 DB에서 해당되는 정보를 리스트 형태로 어플에 전송한다.

```

var express = require('express');
const db = require('../lib/mysql');
var router = express.Router();
router.post('/invest',function(req,res){
  req.on('data',function(data){
    inputData = JSON.parse(data);
    req.on('end',function(){
      email = inputData.email; // 수정 필요
      db.query('select * from userInfo where email=?',[email],function(err,result){
        if(err){
          console.log(err);
        }else{
          res.send(result[0].point);
        }
      })
    })
  })
});

```

- 어플에서 해당 유저에 대한 포인트 정보를 받아오려고 서버에 이메일 정보를 전송한다. 서버는 이에대한 유저 정보를 DB에서 받아와 포인트 값만 어플에 전송해준다.

```

router.post('/decrease',function(req,res){
  console.log('decrease')
  req.on('data',function(data){
    inputData = JSON.parse(data);
    req.on('end',function(){
      email = inputData.email; // 수정 필요
      console.log(email);
      db.query('select * from userInfo where email=?',[email],function(err,result){
        if(err){
          console.log(err);
        }else{
          console.log(result[0].point)
          if(result[0].point==0){
            res.send("No point");
          }else{
            db.query('update userInfo set point =? where email=?',[result[0].point-1,email],function(err,result2){
              if(err){
                console.log(err);
              }else{
                res.send("차감 완료");
              }
            })
          }
        }
      })
    })
  })
});

```

- 어플에서 곡의 예약을 진행하면 현재 포인트에서 1씩 차감되어야함으로 어플은 '/payment/decrease' url로 접속하여 이메일 정보를 전송하고 서버는 DB에서 포인트 개수를 만약 0이면 차감하지 않고 에러를 전송하고 0이 아닐시 현재 가지고 있는 포인트 수에서 1을 차감하여 DB를 수정한다.

```

router.post('/increase',function(req,res){
  req.on('data',function(data){
    inputData = JSON.parse(data);
    req.on('end',function(){
      email = inputData.email; // 수정 필요
      addPoint = parseInt(inputData.point);
      console.log(email,addPoint)
      db.query('select * from userInfo where email=?',[email],function(err,result){
        if(err){
          console.log(err);
        }else{
          db.query('update userInfo set point =? where email=?',[result[0].point+addPoint,email],function(err,result2){
            if(err){
              console.log(err);
            }else{
              res.send("구매 완료");
            }
          })
        }
      })
    })
  })
});

```

- 어플에서 포인트를 구매하려고 할 때 따로 구매한 포인트의 개수와 유저의 이메일 정보를 전송하여 이메일 정보에 따른 포인트 정보를 구매한 곡의 개수를 더하여 수정한다.

```

const fs = require('fs');
const db = require('./mysql');

const timeId = setInterval(function(){
  var dir = 'C:\\Users\\82107\\Desktop\\fileEx';// 음원 하드
  fs.readdir(dir,function(err,list){
    if(err){
      console.log(err);
    }else{
      var len = list.length;
      db.query('select * from device',function(err1,result){
        console.log(result);
        if(err1){
          console.log(err1);
        }else{
          for(var i =0; i<result.length;i++){
            if(len !== result[i].saveNumber){
              console.log(result[i].mac)
              db.query('update device set changeValue = ? where mac =?',[false,result[i].mac],function(err2,updateResult){
                if(err2){
                  console.log(err2);
                }else{
                  console.log(updateResult,'변경완료')
                }
              })
            }
          }
        }
      })
    }
  });
},600000);

module.exports = timeId;

```

- 1시간을 주기로 현재 하드디스크에 있는 노래 곡 수를 받아와 DB에 저장되어있는 라즈베리 파이들의 업데이트 곡 수와 일치하지 않으면 DB의 업데이트 상태를 False 값으로 바꾼다.

```
const mysql = require('mysql');
✓ const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'passwd',
  database: 'serverdb'
})
db.connect();
module.exports = db;
```

- 접근할 DB의 정보를 저장하고 커넥션을 생성한다.

<application>

1. 로그인 세션

```
private class SessionCallback : ISessionCallback {
    override fun onSessionOpened() {
        requestMe()
    }
    private fun requestMe() {
        UserManager.getInstance()
            .me(object : MeV2ResponseCallback() {
                override fun onSessionClosed(errorResult: ErrorResult) {
                    Toast.makeText(MyApplication.applicationContext(), "세션이 만료했습니다. 다시 시도해 주세요: " + errorResult.errorMessage, Toast.LENGTH_LONG).show()
                }
                override fun onFailure(errorResult: ErrorResult) {
                    if (errorResult.errorCode == ApiErrorCode.CLIENT_ERROR_CODE) {
                        Toast.makeText(MyApplication.applicationContext(), "네트워크 연결이 불안정합니다. 다시 시도해 주세요.", Toast.LENGTH_LONG).show()
                    } else {
                        Toast.makeText(MyApplication.applicationContext(), "로그인 도중 오류가 발생했습니다: " + errorResult.errorMessage, Toast.LENGTH_LONG).show()
                    }
                }
            })
    }
    @Suppress("DEPRECATION")
    override fun onSuccess(result: MeV2Response) {
        //JsonExecutes().execute("signin", "http://172.18.28.138/auth/register", result.nickname, result.kakaoAccount.email) //서버에 정보 저장
        JsonExecutes().execute("signin", "http://192.168.122.228/auth/register", result.nickname, result.kakaoAccount.email)
        Toast.makeText(MyApplication.applicationContext(), "Login success", Toast.LENGTH_LONG).show()
        MyApplication.userEmail = result.kakaoAccount.email
        val intent = Intent(MyApplication.applicationContext(), HomeActivity::class.java)
        Intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        startActivity(MyApplication.applicationContext(), intent, null)
    }
}
override fun onSessionOpenFailed(exception: KakaoException?) {
    Toast.makeText(MyApplication.applicationContext(), "로그인 도중 오류가 발생했습니다. 인터넷 연결을 확인해주세요: " + exception.toString(), Toast.LENGTH_SHORT).show()
}
```

- Kakao API를 이용해 Kakao ID를 기반으로 한 회원가입 및 로그인 기능을 지원한다. 이를 이용해 사용자 세션 관리까지 수행할 수 있으며 서버 DB에 사용자 정보가 있는지 확인하여 로그인하고 없을 경우 자동으로 Kakao ID 입력을 통해 회원가입 창으로 이동하고 회원가입이 완료되면 가입 정보가 서버 DB에 저장된다.

2. Bluetooth 스캔

```
@RequiresApi(Build.VERSION_CODES.M)
private fun startScan() {
    if (ble_adapter == null || !ble_adapter.isEnabled) {
        requestEnableBLE()
        //txtState.text = "Scanning Failed: ble not enabled"
        Log.d(TAG, "Scanning Failed: ble not enabled")
        return;
    }
    val filters: MutableList<ScanFilter> = ArrayList()
    val scan_filter = ScanFilter.Builder()
        .setDeviceAddress(MAC_ADDR)
        .build()
    filters.add(scan_filter)
    val settings = ScanSettings.Builder()
        .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER)
        .build()
    //scan_results = HashMap()

    ble_scanner = ble_adapter!!.bluetoothLeScanner
    ble_scanner.startScan(filters, settings, scan_cb)
    is_scanning = true
    Handler(Looper.getMainLooper()).postDelayed({
        //txtState.setText("add Scanned Device : ${scanDevice}")
        Log.d(TAG, "Stop Scanning")
        stopScan()
    }, 7000L)
}
```


- QR코드에 입력된 MAC 주소를 스캔하여 해당 MAC 주소를 가지고 있는 디바이스를 스캔한다. BLE Bluetooth를 이용하며 7초 동안 스캔한 뒤 스캔을 종료한다.

3. Bluetooth 스캔 종류 및 연결

```
@RequiresApi(Build.VERSION_CODES.M)
private fun stopScan(){
    Log.d("Stop Scan","stopScan")
    ble_adapter?.bluetoothLeScanner?.stopScan(scan_cb)
    is_scanning=false
    scan_results= HashMap()
    //txt.text="${scanDevice}"
    if(scanDevice!=null) {
        connectDevice(scanDevice)
        Log.d("mGatt", "${mGatt}")
    }
    else {
        Log.e(TAG, "Scan Failed. Retry, please")
        Toast.makeText(applicationContext, "Scan Failed. Retry, please", Toast.LENGTH_LONG).show()
        //startScan()
    }
    scanDevice=null
    MAC_ADDR=""
}
```

- Bluetooth 스캔이 종료되면 스캔이 정상적으로 진행되었는지 확인한다. 스캔한 디바이스가 있을 경우 해당 디바이스와 연결을 요청하고 없을 경우에는 스캔이 되지 않았음을 출력한다. 연결까지 정상적으로 되었을 경우 Disconnect에서 Connected로 블루투스 연결 버튼의 텍스트가 변경된다.

4. 블루투스 스캔을 위한 콜백 인터페이스 구현

```
val scan_cb = object : ScanCallback() {
    private var cb_scan_result: MutableMap<String, BluetoothDevice>? = null

    init {
        cb_scan_result = scan_results as HashMap<String, BluetoothDevice>
        Log.d(TAG, "scan_cb")
    }

    override fun onScanFailed(errorCode: Int) {
        super.onScanFailed(errorCode)
        Log.e(TAG, "BLE Scan Failed / $errorCode")
        Toast.makeText(applicationContext, "BLE Scan Failed : $errorCode", Toast.LENGTH_LONG).show()
    }

    override fun onScanResult(callbackType: Int, result: ScanResult?) {
        if (result != null) {
            Log.d(TAG, "onScanResult")
            addScanResult(result)
        }
    }

    override fun onBatchScanResults(results: MutableList<ScanResult>?) {
        if (results != null) {
            for (result in results) {
                Log.d(TAG, "onBatchScanResults")
                addScanResult(result)
            }
        }
    }

    private fun addScanResult(result: ScanResult) {
        Log.d(TAG, "add Scan Result")
        val device: BluetoothDevice = result.device
        val device_address: String = device.address
        scan_results?.put(device_address, device)
        scanDevice=device
        Log.d(TAG, "$device $device_address ${scan_results?.get(device)} ${scanDevice.toString()}")
    }
}
```


5. 블루투스 Gatt 서버를 위한 콜백 인터페이스 구현

```
private val gattClientCallback: BluetoothGattCallback = object: BluetoothGattCallback() {
    override fun onConnectionStateChange(gatt: BluetoothGatt?, status: Int, newState: Int) {
        super.onConnectionStateChange(gatt, status, newState)
        if (status == BluetoothGatt.GATT_FAILURE) {
            Log.d(TAG, "BluetoothGatt.GATT_FAILURE, $is_connected")
            disconnectGattServer()
            return
        } else if (status != BluetoothGatt.GATT_SUCCESS) {
            Log.d(TAG, "BluetoothGatt.GATT_FAILURE / $status")
            disconnectGattServer()
            return
        }
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            //setConnected(true)
            Log.d(TAG, "Connected to the GATT server")
            gatt!!.discoverServices()
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            Log.d(TAG, "BluetoothProfile.STATE_DISCONNECTED")
            disconnectGattServer()
        }
    }

    override fun onServicesDiscovered(gatt: BluetoothGatt?, status: Int) {
        super.onServicesDiscovered(gatt, status)
        val services: List<BluetoothGattService>? = gatt!!.services
        lateinit var characteristics: List<BluetoothGattCharacteristic>
        lateinit var descriptors: List<BluetoothGattDescriptor>

        /*discover failure*/
        if (status != BluetoothGatt.GATT_SUCCESS) {
            Log.e(TAG, "Device service discovery failed, status: $status")
            return
        }
        Log.d(TAG, "Services discovery is successful $status")

        // command characteristic를 GATT 서버로부터 찾을
        if (services != null) {
            for (service in services) {
                characteristics = service.characteristics
                for (characteristic in characteristics) {
                    descriptors = characteristic.descriptors
                    for (descriptor in descriptors) {
                        //Log.d(TAG, "${descriptor.uuid.toString()}")
                        if (descriptor.uuid.toString().equals(Constants.CLIENT_CHARACTERISTIC_CONFIG)) {
                            mGatt!!.setCharacteristicNotification(characteristic, true)
                            descriptor.value = BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
                            gatt.writeDescriptor(descriptor)
                        }
                    }
                }
            }
        }
    }
}
```

```
Log.d(TAG, "Services get $mGatt")
is_connected = true
btnconnect2.text = "CONNECTED"
}

override fun onCharacteristicWrite(gatt: BluetoothGatt?, characteristic: BluetoothGattCharacteristic?, status: Int) {
    super.onCharacteristicWrite(gatt, characteristic, status)
    if (status == BluetoothGatt.GATT_SUCCESS) {
        Log.d(TAG, "Characteristic written successfully")
    } else {
        Log.e(TAG, "Characteristic write unsuccessful, status: $status")
        disconnectGattServer()
    }
}

override fun onCharacteristicRead(gatt: BluetoothGatt?, characteristic: BluetoothGattCharacteristic, status: Int) {
    super.onCharacteristicRead(gatt, characteristic, status)
    if (status == BluetoothGatt.GATT_SUCCESS) {
        Log.d(TAG, "Characteristic read successfully")
        readCharacteristic(characteristic)
    } else {
        Log.e(TAG, "Characteristic read unsuccessful, status: $status")
    }
}

override fun onCharacteristicChanged(gatt: BluetoothGatt?, characteristic: BluetoothGattCharacteristic) {
    super.onCharacteristicChanged(gatt, characteristic)
    readCharacteristic(characteristic)
}

private fun readCharacteristic(characteristic: BluetoothGattCharacteristic) {
    val msg = characteristic.getStringValue(0)
    //txtRead += msg
    //txtRead.set(_txtRead)
}
}
```

6. QR코드 스캔

```
val integrator = IntentIntegrator(this)
integrator.setBeepEnabled(true)
integrator.captureActivity = MyBarcodeReaderActivity::class.java
integrator.initiateScan()
}
```

```
class MyBarcodeReaderActivity : AppCompatActivity() {
    private lateinit var capture: CaptureManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_my_barcode_reader)
        capture = CaptureManager(this, barcodeScanner)
        capture.initializeFromIntent(intent, savedInstanceState)
        capture.decode()
    }
}
```

- QR 코드 스캐너를 실행한다. QR 코드 스캐너의 UI를 지정하는 액티비티를 호출하고 해당 액티비티에서 커스텀된 스캐너 UI를 실행하고 스캔이 완료되면 intent를 통해 이전 액티비티로 결과값을 반환한다.

7. PG 결제

```
fun BootpayRequest(price: Int, count: Int) {
    val bootUser = BootUser()
    val bootExtra = BootExtra().setQuotas(intArrayOf(0, 2, 3))
    Bootpay.init(this)
        .setApplicationId("6048ae135b29480027521bb3")
        .setContext(this)
        .setBootUser(bootUser)
        .setBootExtra(bootExtra)
        .setUX(UX_PG_DIALOG)
        .setMethod(Method.CARD)
        .setPG(PG.INICIS)
        .setName("국 예약")
        .setOrderId("${MyApplication.userEmail}")
        .setPrice(price)
        .onConfirm { message ->
            Bootpay.confirm(message)
            Log.d("Bootpay", message)
        }
        .onDone { message ->
            Log.d("Bootpay", "onDone")
            Toast.makeText(MyApplication.getGlobalApplicationContext(), "결제가 정상적으로 완료되었습니다.", Toast.LENGTH_LONG)
            JsonExecute().execute("payment", "http://192.168.122.228/payment/increase", MyApplication.userEmail, count.toString())
        }
        .onReady { message -> Log.d("Bootpay", "onReady") }
        .onCancel { message ->
            Log.d("Bootpay", "onCancel")
            Toast.makeText(MyApplication.getGlobalApplicationContext(), "결제가 취소되었습니다.", Toast.LENGTH_LONG)
        }
        .onError { message ->
            Log.d("Bootpay", "onError")
            Toast.makeText(MyApplication.getGlobalApplicationContext(), "결제 Error", Toast.LENGTH_LONG)
        }
        .onClose { message -> Log.d("close", "close") }
        .request()
}
```

- Bootpay의 PG 결제 모듈을 통해 결제를 수행한다. 해당 기능은 테스트 모드로 동작하며 Bootpay 관리자 화면을 통해 결제 취소와 결제 내역을 확인할 수 있다. 결제가 정상적으로 완료되면 곡 결제에 따른 사용자의 예약 가능 곡 개수가 올라가며 해당 내용은 서버 DB에 갱신된다.

8. Background Http 통신

```
public class JsonExecute: AsyncTask<String, String, String>() {
    override fun doInBackground(vararg strings: String?): String? {
        val arr = arrayOfNulls<String>(strings.size)
        for (i in strings.indices) {
            arr[i] = strings[i]
        }
        val ob = JSONObject()
        var con: HttpURLConnection? = null
        var reader: BufferedReader? = null
        var commend:String= arr[0].toString()
        lateinit var url:URL
        try {
            when(commend){
                "signin"->{
                    url= URL(arr[1].toString())
                    ob.accumulate("name", arr[2])
                    ob.accumulate("email", arr[3])
                }
                "search"->{
                    url= URL(arr[1].toString())
                    ob.accumulate("type", arr[2])
                    ob.accumulate("text", arr[3])
                }
                "countdown"->{
                    url= URL(arr[1].toString())
                    ob.accumulate("email", arr[2])
                }
                "payment"->{
                    url= URL(arr[1].toString())
                    ob.accumulate("email", arr[2])
                    ob.accumulate("point", arr[3])
                }
                "count"->{
                    url= URL(arr[1].toString())
                    ob.accumulate("email", arr[2])
                }
            }
        }
        try {
            //val url = URL("http://175.118.28.138/payment/increase ") // url 수정
            //URL url = new URL(urls[0]);
            con = url.openConnection() as HttpURLConnection
            con.requestMethod = "POST" //post 방식
            con.setRequestProperty("Cache-Control", "no-cache") //캐시 설정
            con.setRequestProperty("Content-Type", "application/json") // json형태로 전송
            con.setRequestProperty("Accept", "text/html") //서버에 response 데이터를 html로 받음
            con.doOutput = true //OutputStream으로 post데이터를 넘겨주겠다는 의미
            con.doInput = true //InputStream으로 서버의 응답을 받겠다는 의미
            con.connect()
        }
```

- Http 통신은 스레드를 통해 백그라운드에서 동작한다. Http 통신 요청 시 요청 내용을 키로 받아 해당 내용에 맞는 URL에 접속하여 데이터 처리를 동작시킨다.

<RaspberryPI>

```
36 while True:
37     try:
38         while dkdk_service.rcv != '':
39             message = dkdk_service.rcv
40             break
41     except:
42         message = 'none'
43
44     if message == '10': # 10 : 제목/가수별
45         self.signal10.emit()
46     elif message == '11' and check == 0: # 11 : 메뉴
47         check = 1
48         self.signal11.emit()
49     elif message == '12': # 12 : 취소
50         if check == 1 or reserve_check == 1: # 취소(메뉴화면)
51             check = 0
52             reserve_check = 0
53             self.signal12.emit('menu')
54         else: # 취소(노래재생화면)
55             self.signal12.emit('video')
56     elif message == '13': # 13 : 아래 화살표
57         self.signal13.emit()
58     elif message == '14': # 14 : 위 화살표
59         self.signal14.emit()
60     elif message == '15' and check == 1: # 15 : 오른쪽 화살표
61         self.signal15.emit()
62     elif message == '16' and check == 1: # 16 : 왼쪽 화살표
63         self.signal16.emit()
64     elif re.search('^17$', str(message)) is not None and check == 1: # 17 : 노래 문자열 입력 (17,name) 형식
65         self.signal17.emit(message)
66     elif message == '18' and check == 1: # 18 : 예약하기 (선택된 행)
67         self.signal18.emit()
68     elif message == '19': # 19 : 시작하기 (선택된 행 or 예약목록)
69         check = 0
70         self.signal19.emit()
71     elif message == '20': # 20 : 예약목록 불러오기
72         reserve_check = 1
73         self.signal20.emit()
74     elif message == '21,1': # 21,1 : 볼륨조절(up)
75         self.signal21.emit('up')
76     elif message == '21,0': # 21,0 : 볼륨조절(down)
77         self.signal21.emit('down')
78     elif message == '22,1': # 22,1 : 피치조절(up)
79         self.signal22.emit('up')
80     elif message == '22,0': # 22,0 : 피치조절(down)
81         self.signal22.emit('down')
82     elif message == '23,1': # 23,1 : 템포조절(up)
83         self.signal23.emit('up')
84     elif message == '23,0': # 23,0 : 템포조절(down)
85         self.signal23.emit('down')
86     elif message == '24' and reserve_check == 1: # 24 : 예약취소
87         self.signal24.emit()
88     elif message == '25' and check == 1: # 25 : 우선 예약
89         self.signal25.emit()
90
91     message = ''
92     dkdk_service.rcv = ''
93     time.sleep(0.1) # 0.1초 기다림
```

애플리케이션에서 블루투스 연결을 통해 오는 명령 메시지에 대해 해당하는 동작을 매칭하여 실행 시켜 주는 부분이다. dkdk_service.rcv를 통해 null 외의 명령 메시지를 받을 경우 동작하며 해당 명령 메시지 타입에 따라 그에 맞는 signal을 발생시켜 동작을 실행한다.


```

20 url = 'http://175.118.28.138/music/search' # 노래 검색 서버 주소
55 # 서버로 부터 파일 가져와 노래 리스트 저장
56 file_data = OrderedDict()
57 file_data["type"] = "재목"
58 file_data["text"] = ""
59 res = requests.post(url, data=json.dumps(file_data))
60 for file in res.json():
61     for val in file:
62         total_song.append([val['id'], val['name'], val['singer']])

```

```

120 self.songList.setSelectionBehavior(QAbstractItemView.SelectRows) # 행단위 선택
121 self.searchEdit.textChanged.connect(self.line_editTextFunction) # 검색창 연결
122 self.typeEdit.textChanged.connect(self.line_editTextFunction) # 재목/가수별 연결
123 column_headers = ['예약번호', '노래제목', '가수']
124 self.songList.setHorizontalHeaderLabels(column_headers)
125 stylesheet = "QHeaderView::section{background-color:rgb(126,126,126)}" # 헤더 스타일 시트
126 self.songList.horizontalHeader().setStyleSheet(stylesheet)
127 self.songList.horizontalHeader().setSectionResizeMode(1, QHeaderView.Stretch)
128 self.songList.horizontalHeader().setSectionResizeMode(2, QHeaderView.Stretch)
129
130 # 노래 리스트 출력
131 for row, val in enumerate(song_list):
132     col = 0
133     for a in val:
134         item = QTableWidgetItem(str(a)) # 아이템 설정
135         item.setTextAlignment(Qt.AlignVCenter | Qt.AlignCenter) # 아이템 가운데 정렬
136         self.songList.setItem(row, col, item)
137         col = col + 1
138     if row == 19:
139         break
140

```

```

159 # 노래검색
160 def line_editTextFunction(self):
161     global current_row
162     global current_page
163     temp_title = ['id', 'name', 'singer'] # json 추출 목록
164
165     search = self.searchEdit.text()
166     division = self.typeEdit.text()
167
168     # 노래 리스트 출력
169     file_data = OrderedDict()
170     file_data["type"] = division
171     file_data["text"] = search
172     res = requests.post(url, data=json.dumps(file_data)) # 서버로 부터 리스트 불러오기
173     # 검색결과 0일시
174     self.songList.clearContents() # 테이블 초기화
175     if len(res.json()) == 0:
176         return 0
177
178     for row, dicts in enumerate((res.json())[0]):
179         col = 0
180         for key, val in dicts.items():
181             if temp_title[col] == key:
182                 item = QTableWidgetItem(str(val))
183                 item.setTextAlignment(Qt.AlignVCenter | Qt.AlignCenter)
184                 self.songList.setItem(row, col, item)
185                 col = col + 1
186             if row == 19:
187                 break

```

노래 목록 리스트를 위한 실행 시켜 주는 부분이다. 미리 정의된 서버 URL을 통해 노래 리스트의 타입과 이름을 요청해 이를 JSON 형식으로 받아온다. JSON 형식의 파일을 노래 리스트에 포맷에 맞추어 등록하기 위해 해당 컬럼 값으로 구성된 리스트를 만들게 된다. 해당 리스트를 Qt UI파일의 songList이름의 테이블에 값을 넣어 메뉴 화면을 구성한다. line_editTextFunction 함수를 노래검색창과 연결시켜 변화를 감지 할 수 있게 하였으며 검색내용이 바뀔 경우, 바뀐 내용에 대한 결과를 songList 테이블로 제공한다.

```

142 def mainScreen(self): # 메인화면 재생 함수
143     self.mediaPlayer.setMedia(QMediaContent(QUrl.fromLocalFile("/home/pi/project/NBackVideo.mp4")))
144     self.mediaPlayer.play()
151 def mediaStateChanged(self): # 노래 종료시
152     # 노래 정지시 & 타이머 9초이상 경과 & 중복실행 불가
153     global test
154     if self.mediaPlayer.state() == QMediaPlayer.StoppedState and self.time > 9 and self.check == 0:
155         self.check = 1
156         if 0 <= self._index < self._clips - 1: # 남은 노래 있을시
157             self._index = self._index + 1
158             self.mediaPlayer.setMedia(
159                 QMediaContent(QUrl.fromLocalFile("/home/pi/project/" + self._songList[0] + ".mp4")))
160             test = pygame.mixer.Sound(self._songList[0] + ".wav")
161             test.play()
162             self.mediaPlayer.play()
163
164             del self._songList[0] # 재생노래 삭제
165             mywindow.reserveEdit.clear()
166             for val in self._songList: # 상단 bar 재설정
167                 mywindow.reserveEdit.setText(mywindow.reserveEdit.text() + val + "\t")
168             self.time = 0
169             self.check = 0
170             self.timerVar.start()
171         else: # 남은 노래 없을시
172             self.time = 0
173             self.check = 0
174             self.mainScreen()
175
176 @pyqtSlot(list) # 전체 재생
177 def video_signal10_emitted(self, arg):
178     global test
179     self._songList = arg
180     self._clips = len(arg)
181     self._index = 0
182     self.check = 0
183     self.mediaPlayer.setMedia(QMediaContent(QUrl.fromLocalFile("/home/pi/project/" + self._songList[0] + ".mp4")))
184     test = pygame.mixer.Sound(self._songList[0] + ".wav")
185     test.play()
186     self.mediaPlayer.play()
187
188     del self._songList[0] # 재생노래 삭제
189     print(self._songList)
190     mywindow.reserveEdit.clear()
191     for val in self._songList: # 상단 bar 재설정
192         mywindow.reserveEdit.setText(mywindow.reserveEdit.text() + val + "\t")
193
194     self.timerVar.setInterval(500)
195     self.timerVar.start()

```

메인 배경화면과 예약된 노래번호에 해당하는 영상 재생시키는 부분이다. 동영상 재생을 위한 QMediaPlayer 객체를 생성시켜 해당되는 노래이름을 찾아 영상을 재생시킨다. 영상이 종료되고 재생해야할 다음 영상이 존재하면 다음영상을 재생, 없으면 기본 배경영상을 재생한다.

```

# Get ServiceManager and AdvertisingManager
service_manager = get_service_manager(bus)
ad_manager = get_ad_manager(bus)

# Create gatt services
motor = setup_motor()
app = MotorApplication(bus, motor)

# Create advertisement
dkdk_advertisement = MotorAdvertisement(bus, 0)

mainloop = gobject_MainLoop()

```

블루투스 서비스 매니저와 해당 블루투스를 advertising할 객체를 저장시킨다. 동시에 GATT 서비스를 만든다.

```

# Register gatt services
service_manager.RegisterApplication(app.get_path(), {},
                                   reply_handler=register_app_cb,
                                   error_handler=register_app_error_cb)

# Register advertisement
ad_manager.RegisterAdvertisement(dkdk_advertisement.get_path(), {},
                               reply_handler=register_ad_cb,
                               error_handler=register_ad_error_cb)

try:
    mainloop.run()
except KeyboardInterrupt:
    print("Finished")

```

만들어진 서비스들을 기반으로 서비스 들을 등록시키고 블루투스 연결까지 코드를 대기시킨다

```

CMD_UUID = 'c6a89af5-0385-4d4a-8cb4-c856fcfb1321'

def __init__(self, bus, index, service, motor):
    Characteristic.__init__(
        self, bus, index,
        self.CMD_UUID,
        ['read', 'write'],
        service)
    #self.value = [0x32]
    self.value = [ 0x00 for i in range(1024) ]
    self.motor = motor

```

코드들이 모두 정상수행 됐을 때, 위의 사진의 UUID로 블루투스를 advertise한다. GATT 서비스는 라즈베리 파이에서 데이터를 보내는 read와 다른 블루투스 연결 장치의 데이터를 받아오는 write서비스를 포함시킨다. 받아오는 데이터는 value에 저장한다.

```

def WriteValue(self, value, options):
    global rcv
    rcv = ''.join([str(v) for v in value])
    print("value:%s" % rcv)

```

위는 데이터를 받아왔을 때 마지막으로 수행되는 코드로 value에 저장된 데이터를 rcv에 데이터를 가공한 후 문자열로 저장한다.

```

t = threading.Thread(target=dkdk_service.main)
t.start()
# 화면출력

```

qt 메인에 있는 스레드 함수이며, 블루투스를 한 스레드로 수행시킨다.

```

def run(self):
    message = ''
    check = 0
    video_check = 0
    reserve_check = 0
    while True:
        try:
            while dkdk_service.rcv != '':
                message = dkdk_service.rcv
                break
        except:
            message = 'none'

        if message == '10': # 10 : 제목/가수별
            self.signal10.emit()

```

스레드로 수행된 함수에서 rcv에 데이터가 들어왔을 때, 메시지에 그 데이터가 저장된다. 이
에 들어온 메시지에 따라 화살표 기능 노래 재생기능 등을 수행하게 되고 마지막으로
message와 rcv를 초기화 시켜주면서 다시 rvc의 데이터를 기다리게 된다.

```

def download(self, url, file_name):
    with open(file_name, "wb") as file: # open in binary mode
        response = requests.get(url) # get request
        file.write(response.content)

def invest_download(self):
    res = requests.post(self.investURL, data = {'mac':2})
    print(res.text)
    dict = json.loads(res.text)
    print(dict['changeValue'])

    return dict['changeValue']

```

download는 다운로드 될 파일의 경로와 이름, 그리고 다운로드 할 URL로 파일을 다운로드
한다. invest_download는 이 장치가 업데이트할 파일이 있는지 없는지 확인한다.


```

def mkZip(self):
    requests.post(self.zipURL,data = {'mac':2})
    print("make zip")

def download_file(self):
    self.download(self.downloadURL, self.zipName)
    print("down load")

def zipExtract(self):
    fantasy_zip = zipfile.ZipFile(self.zipName)
    fantasy_zip.extractall(self.singLocation)
    fantasy_zip.close()

def removeFile(self):
    if os.path.isfile(self.zipName):
        os.remove(self.zipName)

```

업데이트할 파일이 있다고 판단되면 mkZip으로 서버에 압축 파일을 생성해 줄 것을 요청한다. download_file로 download함수를 호출하여 파일을 다운로드 한 뒤, zipExtract로 압축을 해제한다. 남아있는 압축파일은 removeFile을 이용하여 삭제한다.

```

def scheduleDownload():
    schedule.every().day.at("05:00").do(downloadSchedul)
    while True:
        schedule.run_pending()
        time.sleep(1)

```

파일 다운로드 기능은 매일 오전 5시에 수행되도록 스케줄 화 되어있다

```

scheduleTread = threading.Thread(target=serverConnection.scheduleDownload)
scheduleTread.start()

```

또한 이는 하나의 스레드로 동작한다.