「 공영주차장 알림 앱 」

정보통신 캡스톤디자인

2020-1학기

[2팀]

김연진

김지영

하예진

1. 프로젝트 주제 및 개요

(1) 프로젝트 소개

ㅇ 공영주차장 알림 앱 '하이파킹'은 공공데이터와 Big Data를 활용하여 주차장의 위치와 운영시간, 주차요금 등의 기본정보와 요일 및 시간대별 실시간 주차대수를 제공한다. 또한, 사용자의 주변 편의시설(영화관, 역, 전통시장 등)에서 가장 가까운 주차장을 찾을 수 있는 기능과 즐겨찾기 기능이 있어 편리하며, 피드백 기능을 통해 앱 사용자들의 의견을 수렴하여 주기적으로 업데이트할 수 있다는 점에서 기존 공영주차장 알림서비스 앱의 장점을 확대하고, 단점을 개선한 앱이다.

(2) 추진배경

ㅇ 서울에 자가용을 이용하여 방문할 때 주차난으로 어려움을 겪었고 유료 주차장 사용이 불가피할 때가 많았다.

ㅇ 현재 공영주차장 알림서비스를 제공하는 앱이 존재하지만, 실시간 주차대수 표시는 따로 되어있지 않아 공영주차장을 이용하기 위해 도착했을 때 잔여 주차공간이 없어 사용자들이 어려움을 겪은 경험이 있다.

따라서, 기존의 공영주차장 알림서비스 앱을 이용하며 '현재 실시간 주차대수' 정보가 파악되지 않아 불편했던 경험을 통한 기존 공영주차장 알림서비스 앱의 개선 필요성을 느껴 이 프로젝트를 추진하게 되었다.

(3) 기대효과

ㅇ Big Data를 이용하여 요일과 시간대별 실시간 현재 주차대수 안내 서비스 제공으로, 기존 주차장 앱의 불편함 개선

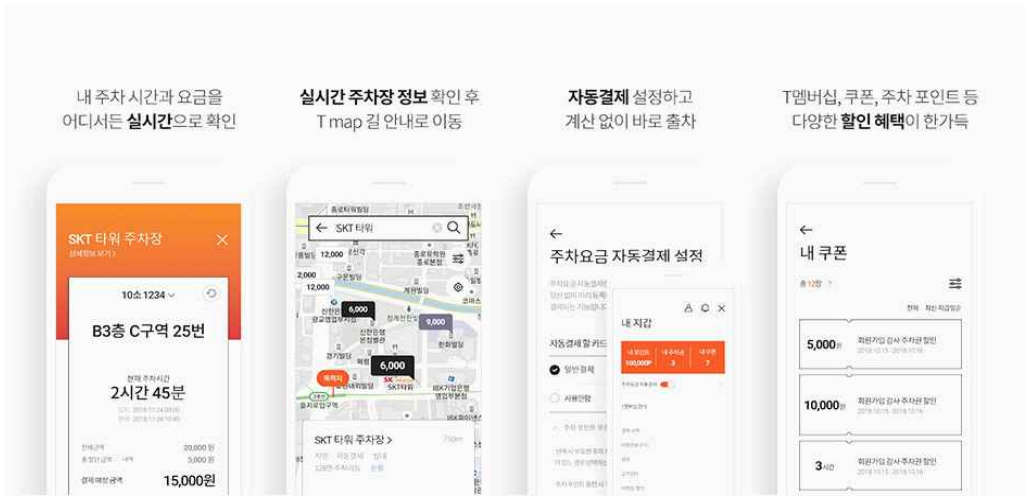
ㅇ 공공데이터, 빅데이터 등 다양한 데이터를 기반으로 여러 상황에서 원활한 주차장 이용을 가능하게 하여 공영주차장 이용을 활성화할 수 있음

ㅇ 전통시장 및 관광지 등에서의 교통 및 주차난 해결에 도움

ㅇ 주차장과 주차장 주변 시설들을 연계하여 전통시장과 관광지 이용의 활성화로 지역사회의 경제 가치 향상에 도움

2. 연구의 필요성

(1) 기존의 주차장 앱과 비교



현재 주차장 사용을 위해 대표적으로 사용되는 애플리케이션에는 '모두의 주차장'과 'T맵'이 있다. 주차장에 대한 정보와 사용요금을 확인할 수 있으며, 결제시스템을 연동해서 자동결제도 가능한 단계로 이루어져 있다. 하지만 실시간 주차대수 정보는 파악하지 못하여 주차장을 방문했을 때 사용이 불가능한 불편함을 겪을 수 있다.
이러한 불편함을 개선하고, 이용 가능한 주차장을 확인하기 위해 실시간 주차대수 정보가 제공될 수 있는 서비스가 필요하다.

3. 전체 시스템 구성 및 설계

(1) 로딩화면

<'하이파킹' 로고>

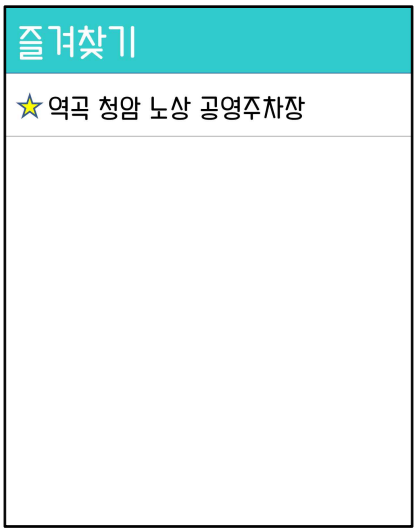
=> 앱을 시작할 때 가장 처음 보이는 화면으로 로고를 띄워 1초 정도 유지하도록 구성하였다.

(2) 홈 화면

<구상한 메인화면(계획)>

=> 앱의 메인화면으로, 네이버 지도를 기반으로 하기 위해 네이버 지도 open API를 연동한다. 원하는 주차장을 클릭하면 주차장 이름 / 잔여 대수 / 주차요금 / 운영시간 등의 정보를 마커를 사용하여 표시하도록 구성하였다. 주차장의 이름, 요금, 운영시간, 장애인 주차 가능 여부는 JSON 형태의 data를 불러오는 형식으로 하고, 잔여 대수는 실시간으로 변화하는 모습을 보여주기 위해 Big Data를 이용한다.

(3) 즐겨찾기 화면



<구상한 즐겨찾기 화면(계획)>

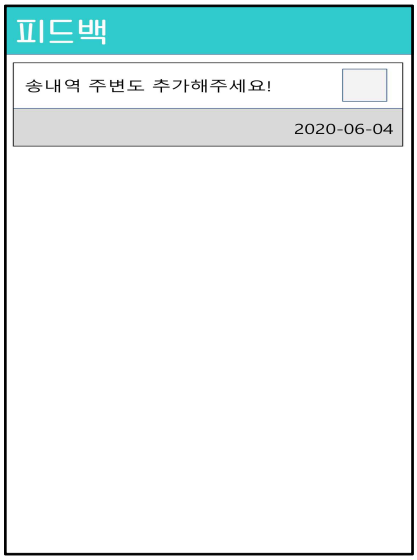
=> 사용자가 자주 가는 주차장을 즐겨찾기에 추가하여 빠른 이용이 가능하도록 설계하였다. 즐겨찾기는 listview 형태로 구성한다. 즐겨찾기 탭에 추가된 주차장을 누르면 해당 주차장으로 이동하게 된다.

(4) 주변정보 화면

<구상한 주변정보 화면(계획)>

=> 사용자가 위치한 지역 근처의 주요 주변 정보를 알려주는 화면이다.
주요 주변 정보도 즐겨 찾기와 마찬가지로 listview 형식으로 구성한다. 주변정보 화면은 역, 영화관, 전통시장을 위주로 하며, 주변정보 list 중 하나를 선택하면 해당 위치에서 가장 가까운 주차장의 정보를 알려주는 서비스를 제공한다.

(5) 피드백 화면



<구상한 피드백 화면(계획)>

=> 주변 정보 추가 요청, 주차대수 수정 요구, 주차장 등록 등 앱 사용자들의 피드백을 받을 수 있는 서비스를 제공한다. 피드백 화면은 메모장 형식으로 구현하며, 사용자들이 작성하는 내용과 작성 날짜가 화면에 계속 누적되어 앱 개발자가 확인할 수 있도록 한다.

4. 제작내용

(1) 사용 기술 및 기능소개

1) 지도화면을 위한 API 연동

지도화면을 구성하기 위하여 네이버 지도 Open API를 안드로이드 스튜디오에 연동하였으며, 사용자의 실시간 위치를 파악할 수 있도록 GPS 기능을 추가하였다. 또한, 주차장의 위치를 마커로 표시하고 주차장에 대한 정보를 표시하기 위해 마커를 클릭하면 정보창을 띄우는 옵션을 추가로 개발하였다.

2) 주차장 정보 data 수집

- 주차장 기본정보 data

주차장 이름, 주차장 위치, 주차요금, 운영시간 정보, 전체 주차면 수를 수집하여 안드로이드 스튜디오에 JSON 형태의 data로 저장하였다.

- 실시간 주차대수 data

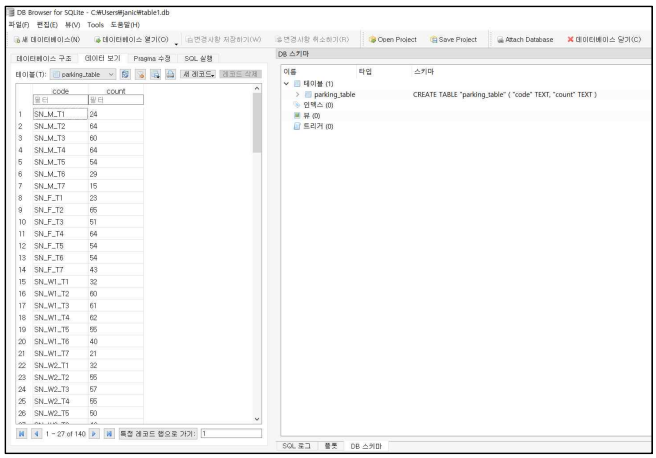
요일과 시간대별로 달라지는 주차대수를 파악하기 위한 data를 조사하고 수집하여 자체적으로 Big Data를 만들어 db 형태의 파일로 저장하였다. 저장한 db 파일을 안드로이드 스튜디오와 연동하여 요일과 시간대가 바뀔 때마다 주차장의 현재 주차대수를 실시간 정보로 받아볼 수 있다.

3) 즐겨찾기, 주변 정보, 피드백 기능

메인화면인 지도화면 이외에 추가로 3가지 메인을 더 구성하여 앱의 완성도를 높였다. 안드로이드 스튜디오 내에서 화면 설계를 통해 개발하였으며 사용자가 자주 사용하는 주차장을 등록하기 위한 즐겨찾기 화면, 주차장 주변 정보를 불러들이기 위한 주변 정보 화면, 앱 개선을 위한 사용자의 피드백을 수렴하는 공간인 피드백 화면을 추가하였다.

(2) App Code 분석

2-1) SQLite DB 편집기



위 그림은 시간대별로 바뀌는 주차대수를 조사하여 입력한 data를 의미한다.
아래 그림을 살펴보면, column 값을 code와 count 2개로 주었고, 각각의 필드명마다 값이 달라지는 것을 확인할 수 있다. code의 A_B_C에서 A는 주차장 이름을 의미하고 B는 요일, C는 시간대를 의미하며 count는 현재 주차대수를 의미한다.

	code	count
	필터	필터
1	SN_M_T1	24
2	SN_M_T2	64
3	SN_M_T3	60
4	SN_M_T4	64
5	SN_M_T5	54
6	SN_M_T6	29
7	SN_M_T7	15
8	SN_F_T1	23
9	SN_F_T2	65
10	SN_F_T3	51
11	SN_F_T4	64
12	SN_F_T5	54
13	SN_F_T6	54
14	SN_F_T7	43
15	SN_W1_T1	32
16	SN_W1_T2	60
17	SN_W1_T3	61
18	SN_W1_T4	62
19	SN_W1_T5	55
20	SN_W1_T6	40
21	SN_W1_T7	21
22	SN_W2_T1	32
23	SN_W2_T2	55

	code	count
	필터	필터
29	BN_M_T1	8
30	BN_M_T2	39
31	BN_M_T3	27
32	BN_M_T4	26
33	BN_M_T5	20
34	BN_M_T6	15
35	BN_M_T7	이용중지
36	BN_F_T1	7
37	BN_F_T2	39
38	BN_F_T3	21
39	BN_F_T4	25
40	BN_F_T5	25
41	BN_F_T6	30
42	BN_F_T7	이용중지
43	BN_W1_T1	5
44	BN_W1_T2	30
45	BN_W1_T3	30
46	BN_W1_T4	42
47	BN_W1_T5	35
48	BN_W1_T6	22
49	BN_W1_T7	이용중지
50	BN_W2_T1	4
51	BN_W2_T2	29
52	BN_W2_T3	30
53	BN_W2_T4	38
54	BN_W2_T5	35

	code	count
	필터	필터
57	YN_M_T1	8
58	YN_M_T2	34
59	YN_M_T3	25
60	YN_M_T4	33
61	YN_M_T5	31
62	YN_M_T6	20
63	YN_M_T7	11
64	YN_F_T1	8
65	YN_F_T2	36
66	YN_F_T3	25
67	YN_F_T4	32
68	YN_F_T5	33
69	YN_F_T6	28
70	YN_F_T7	14
71	YN_W1_T1	9
72	YN_W1_T2	33
73	YN_W1_T3	28
74	YN_W1_T4	25
75	YN_W1_T5	25
76	YN_W1_T6	14
77	YN_W1_T7	6
78	YN_W2_T1	10
79	YN_W2_T2	32
80	YN_W2_T3	26
81	YN_W2_T4	26
82	YN_W2_T5	25

만들어진 파일을 db 형태로 저장하고, 안드로이드 스튜디오에 저장하여 요일과 시간대가 바뀔 때마다 count의 값이 달라지도록 알고리즘을 개발해 실시간으로 주차대수 확인이 가능하도록 하였다.

작성한 db 파일을 안드로이드 스튜디오 assets 폴더 아래에 넣어준 모습이다.

2-2) Android Studio

1) app>java>com.example.parking

▶ **MainActivity.java**

=> App의 메인화면을 구성하는 액티비티이다.

```
public class MainActivity extends FragmentActivity implements OnMapReadyCallback {
private static final int ACCESS_LOCATION_PERMISSION_REQUEST_CODE = 100;
private FusedLocationSource locationSource;

private FragHome fragHome;
private FragStar fragStar;
private FragInfo fragInfo;
private FragFeedback fragFeedback;
private InfoWindow infoWindow;
NaverMap navermap;

ArrayList<String> parking_name = new ArrayList<>(), parking_cnt = new ArrayList<>(),
parking_x = new ArrayList<>(), parking_y = new ArrayList<>(), parking_price = new
ArrayList<>();

ArrayList<InfoWindow> wList = new ArrayList<>();
List<Model> wlist;
double my_x, my_y;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

DataAdapter dataAdapter = new DataAdapter(MainActivity.this);
dataAdapter.createDatabase();
dataAdapter.open();

wlist = dataAdapter.getTableData();
dataAdapter.close();
for (int i = 0; i < wlist.size(); i++) {
Log.e("db", wlist.get(i).getCode() + " / " + wlist.get(i).getCount());
}
add();

MapFragment mapFragment = (MapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);

Intent intent = new Intent(this, LoadingActivity.class);
startActivity(intent);

fragHome = new FragHome();
fragStar = new FragStar();
fragInfo = new FragInfo();
fragFeedback = new FragFeedback();

BottomBar bottomBar = (BottomBar) findViewById(R.id.bottomBar);
bottomBar.setOnTabSelectListener(new OnTabSelectListener() {
@Override
public void onTabSelected(@IdRes int tabId) {
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

if (tabId == R.id.tab_home) {
transaction.replace(R.id.contentContainer, fragHome).commit();
} else if (tabId == R.id.tab_star) {
```

```
transaction.replace(R.id.contentContainer, fragStar).commit();
} else if (tabId == R.id.tab_info) {
transaction.replace(R.id.contentContainer, fragInfo).commit();
} else if (tabId == R.id.tab_feedback) {
transaction.replace(R.id.contentContainer, fragFeedback).commit();
}
}
});
}
```

=> 실시간 주차대수 데이터를 저장해놓은 db 파일을 안드로이드 스튜디오에 불러오기 위해서는 데이터베이스를 불러오도록 도와주는 Adapter가 필요하다. 따라서, DataAdapter와 DataBaseHelper라는 자바 클래스를 별도로 생성하고, 메인액티비티에 모든 클래스들을 연결해주었다.
또한, 앱의 하단 부분에 4가지 메뉴를 구성하기 위해 각각의 메뉴 또한 자바 클래스를 별도로 생성해주었고, 메뉴를 탭 하면 해당하는 메뉴화면으로 넘어가는 것을 구현하도록 코드를 작성해주었다.

```
private void add() {
LocationManager locationManager = (LocationManager)
getSystemService(LOCATION_SERVICE);
Location location = null;
boolean isGPSEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
boolean isNetworkEnabled =
locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
if (!isGPSEnabled && !isNetworkEnabled) {
} else {
if (isNetworkEnabled) {
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

return;
}
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 60000,
10, locationListener);
if (locationManager != null) {
location = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
if (location != null) {
my_x = location.getLatitude();
my_y = location.getLongitude();
}
}
}
if (isGPSEnabled) {
if (location == null) {
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 60000, 10,
locationListener);
if (locationManager != null) {
location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
if (location != null) {
my_x = location.getLatitude();
my_y = location.getLongitude();
}
\
\
}
```

```
}
parking_name.add("역곡청암 노상 공영주차장");
parking_name.add("가톨릭대학교 성심교정 주차장");
parking_name.add("소사종합시장 공영주차장");
parking_name.add("송내북부공영주차장");
parking_name.add("부천남부역공영주차장");

parking_cnt.add("42면");
parking_cnt.add("300면");
parking_cnt.add("100면");
parking_cnt.add("65면");
parking_cnt.add("48면");

parking_x.add("37.488901");
parking_y.add("126.813085");
parking_x.add("37.485927");
parking_y.add("126.801970");
parking_x.add("37.479048");
parking_y.add("126.793487");
parking_x.add("37.488769");
parking_y.add("126.756266");
parking_x.add("37.483428");
parking_y.add("126.782933");

parking_price.add("1시간당3,000원 ₩n운영시간: 00:00~24:00");
parking_price.add("1일당2,000원 ₩n운영시간: 00:00~24:00");
parking_price.add("1시간당1,400원 ₩n운영시간: 00:00~24:00");
parking_price.add("1시간당2,100원 ₩n운영시간: 00:00~24:00");
parking_price.add("1시간당2,100원 ₩n운영시간: 08:00~24:00");
}
```

=> 사용자의 실시간 위치 파악을 위해 GPS를 연동하고, 주차장의 위치를 표시하기 위해 위도와 경도로 나타내고, 주차장의 기본정보들도 입력하여 화면에 나타나도록 하였다.

```
@Override
public void onMapReady(@NonNull NaverMap naverMap) {
locationSource = new FusedLocationSource(this,
ACCESS_LOCATION_PERMISSION_REQUEST_CODE);
naverMap.setLocationSource(locationSource);
UiSettings uiSettings = naverMap.getUiSettings();
uiSettings.setLocationButtonEnabled(true);
navermap = naverMap;
naverMap.addOnLocationChangeListener(new NaverMap.OnLocationChangeListener() {
@Override
public void onLocationChange(@NonNull Location location) {
CameraUpdate cameraUpdate = CameraUpdate.scrollTo(new LatLng(my_x, my_y));
navermap.moveCamera(cameraUpdate);

addMarket();

navermap.removeOnLocationChangeListener(this);
}
});
}

private void addMarket() {
wList.clear();
for (int i = 0; i < parking_x.size(); i++) {
infoWindow = new InfoWindow();
infoWindow.setPosition(new LatLng(Double.parseDouble(parking_x.get(i)),
Double.parseDouble(parking_y.get(i))));
infoWindow.setAdapter(new InfoWindowAdapter(this));
infoWindow.setTag(i);
```

```
infoWindow.setOnClickListener(new Overlay.OnClickListener() {
@Override
public boolean onClick(@NonNull Overlay overlay) {
infoWindow.close();
return false;
}
});
wList.add(infoWindow); //정보창 추가

final Marker marker = new Marker();
marker.setPosition(new LatLng(Double.parseDouble(parking_x.get(i)),
Double.parseDouble(parking_y.get(i))));
marker.setIcon(OverlayImage.fromResource(R.drawable.parking_icon));
marker.setWidth(100);
marker.setHeight(140);
marker.setAnchor(new PointF(1, 1));
marker.setOnClickListener(new Overlay.OnClickListener() {
@Override
public boolean onClick(@NonNull Overlay overlay) {
for (int i = 0; i < wList.size(); i++) {
wList.get(i).close();
}
infoWindow.open(marker);
return true;
}
});
marker.setTag(Integer.toString(i));
marker.setMap(navermap); //마커 추가
}
}
```

=> 지도 위에 나타나는 이벤트를 처리해주기 위한 부분이다. 주차장 위치를 표시하기 위해 marker를 추가해주었고, 마커를 클릭하면 주차장 정보를 말풍선으로 띄워주는데, 클릭이벤트를 위해 setOnClickListener 함수를 사용하여 onClick안에서 열리고 닫히도록 해주었다.

```
@NonNull
@Override
public View getView(@NonNull InfoWindow infoWindow) {
if (rootView == null) {
rootView = View.inflate(context, R.layout.listview_text, null);
text = rootView.findViewById(R.id.title);
}

if (infoWindow.getMarker() != null) {
int xx = Integer.parseInt(infoWindow.getMarker().getTag().toString());

if (parking_name.get(xx).equals("역곡청암 노상 공영주차장")) {
cd = "YN";
} else if (parking_name.get(xx).equals("가톨릭대학교 성심교정 주차장")) {
cd = "CS";
} else if (parking_name.get(xx).equals("소사종합시장 공영주차장")) {
cd = "SJ";
} else if (parking_name.get(xx).equals("송내남부공영주차장")) {
cd = "SN";
} else if (parking_name.get(xx).equals("부천남부역공영주차장")) {
cd = "BN";
}

String date1 = new SimpleDateFormat("EEE", Locale.KOREAN).format(new Date()); //
요일을 구함
int date2 = Integer.parseInt(new SimpleDateFormat("HH", Locale.KOREAN).format(new
```

```
Date()); // 시간을 구함
int date3 = Integer.parseInt(new SimpleDateFormat("mm", Locale.KOREAN).format(new
Date())); // 분을 구함
if (date3 != 0) {
date2++;
}
if (date1.equals("월") || date1.equals("화") || date1.equals("수") || date1.equals("목")) {
cd = cd + "_M";
} else if (date1.equals("금")) {
cd = cd + "_F";
} else if (date1.equals("토")) {
cd = cd + "_W1";
} else if (date1.equals("일")) {
cd = cd + "_W2";
}
if (date2 > 6 && date2 <= 9) { // 6시보다 크고9시와 같거나 큼
cd = cd + "_T1";
} else if (date2 > 9 && date2 <= 12) { // 9시보다 크고12시와 같거나 큼
cd = cd + "_T2";
} else if (date2 == 13) { // 13시
cd = cd + "_T3";
} else if (date2 > 13 && date2 <= 17) { // 13시보다 크고17시와 같거나 큼
cd = cd + "_T4";
} else if (date2 == 18) { // 18시
cd = cd + "_T5";
} else if (date2 > 18 && date2 <= 24 || date2 == 0) { // 18시보다 크고24시와 같거나
큼
cd = cd + "_T6";
} else if (date2 > 0 && date2 <= 6) { // 0시보다 크고6시와 같거나 큼
cd = cd + "_T7";
}
for (int i = 0; i < wlist.size(); i++) { // 클릭한 주차장들의 코드를 계산
if (cd.equals(wlist.get(i).getCode())) { // 디비의 코드값과 같으면
value = wlist.get(i).getCount(); // 주차대수를 확보
}
}
if(value.equals("이용중지")){
text.setText(parking_name.get(xx) + "₩n주차요금: " + parking_price.get(xx) + "₩n총" +
parking_cnt.get(xx) + "₩n현재 주차대수: " + value);
}else {
text.setText(parking_name.get(xx) + "₩n주차요금: " + parking_price.get(xx) + "₩n총" +
parking_cnt.get(xx) + "₩n현재 주차대수: " + value + "대(" + date1 + "요일, " + date2 +
"시(24시 기준))");
}
} else {
text.setText("");
}

return rootView;
}

}
```

=> 작성한 db파일을 불러와 지도 위에 말풍선으로 표시한 부분이다. db파일에서 생성한 필드명을 코드에 나타내주었고, if문을 사용하여 요일과 시간대가 바뀔 때마다 count값이 변하도록 해주었다.

▶ **DataAdapter.java**

```
public class DataAdapter
{
protected static final String TAG = "DataAdapter";

protected static final String TABLE_NAME = "parking_table";

private final Context mContext;
private SQLiteDatabase mDb;
private DataBaseHelper mDbHelper;

public DataAdapter(Context context)
{
this.mContext = context;
mDbHelper = new DataBaseHelper(mContext);
}

public DataAdapter createDatabase() throws SQLException
{
try
{
mDbHelper.createDataBase();
}
catch (IOException mIOException)
{
Log.e(TAG, mIOException.toString() + "  UnableToCreateDatabase");
throw new Error("UnableToCreateDatabase");
}
return this;
}

public DataAdapter open() throws SQLException
{
try
{
mDbHelper.openDataBase();
mDbHelper.close();
mDb = mDbHelper.getReadableDatabase();
}
catch (SQLException mSQLException)
{
Log.e(TAG, "open >>"+ mSQLException.toString());
throw mSQLException;
}
return this;
}

public void close()
{
mDbHelper.close();
}
```

▶ **DataBaseHelper.java**

```
public class DataBaseHelper extends SQLiteOpenHelper {

private static String TAG = "DataBaseHelper";
private static String DB_PATH = "";
private static String DB_NAME = "parking_table2.db";

private SQLiteDatabase mDataBase;
private final Context mContext;

public DataBaseHelper(Context context) {
super(context, DB_NAME, null, 1);// 1은 데이터베이스 version
if (android.os.Build.VERSION.SDK_INT >= 17) {
DB_PATH = context.getApplicationInfo().dataDir + "/databases/";
} else {
DB_PATH = "/data/data/" + context.getPackageName() + "/databases/";
}
this.mContext = context;
}

public void createDataBase() throws IOException {
boolean mDataBaseExist = checkDataBase();
if (!mDataBaseExist) {
this.getReadableDatabase();
this.close();
try {
//assests 폴더에서 데이터베이스 파일 복사해오기
copyDataBase();
Log.e(TAG, "createDatabase database created");
} catch (IOException mIOException) {
throw new Error("ErrorCopyingDataBase");
}
}
}

private boolean checkDataBase() {
File dbFile = new File(DB_PATH + DB_NAME);

return dbFile.exists();
}

private void copyDataBase() throws IOException {
AssetManager manager = mContext.getAssets();
String folderPath = DB_PATH;
String filePath = DB_PATH + DB_NAME;
File folder = new File(folderPath);
File file = new File(filePath);
FileOutputStream fos = null;
BufferedOutputStream bos = null;
try {
InputStream is = manager.open(DB_NAME);
BufferedInputStream bis = new BufferedInputStream(is);
if (folder.exists()) {
} else {
folder.mkdirs();
}
if (file.exists()) {
file.delete();
file.createNewFile();
}
fos = new FileOutputStream(file);
bos = new BufferedOutputStream(fos);
int read = -1;
```

```
byte[] buffer = new byte[1024];
while ((read = bis.read(buffer, 0, 1024)) != -1) {
bos.write(buffer, 0, read);
}
bos.flush();
bos.close();
fos.close();
bis.close();
is.close();
} catch (IOException e) {
Log.e("ErrorMessage : ", e.getMessage());
}

}

//데이터베이스를 열어서 쿼리를 쓸수있게만든다.
public boolean openDataBase() throws SQLException {
String mPath = DB_PATH + DB_NAME;

mDataBase = SQLiteDatabase.openDatabase(mPath, null,
SQLiteDatabase.CREATE_IF_NECESSARY);

return mDataBase != null;
}
```

=> 위의 두가지 클래스는 데이터베이스의 테이블 이름을 명시해주고, 파일을 실제로 불러오는 기능을 수행하는 클래스이다.

▶ **FragHome.java**

```
public class FragHome extends Fragment {
public FragHome(){
}
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState){
return inflater.inflate(R.layout.frag_home, container, false);
}
}
```

=> 홈 메뉴를 구현하기 위해 생성한 클래스이다.

▶ **FragStar.java**

```
ppublic class FragStar extends Fragment {
public FragStar(){
}
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState){
return inflater.inflate(R.layout.frag_star, container, false);
}
}
```

=> 즐겨찾기 메뉴를 구현하기 위해 생성한 클래스이다.

▶ **FragInfo.java**

```
public class FragInfo extends Fragment {
public FragInfo(){
}
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState){
return inflater.inflate(R.layout.frag_info, container, false);
}
}
```

=> 주변정보 메뉴를 구현하기 위해 생성한 클래스이다.

▶ **FragFeedback.java**

```
public class FragFeedback extends Fragment {
SQLiteHelper dbHelper;
RecyclerView recyclerView;
MemoAdapter recyclerAdapter;
Button btnAdd;
List<Memo> memoList;
public FragFeedback(){
}
```

=> 피드백 메뉴를 구현하기 위해 생성한 클래스이다.

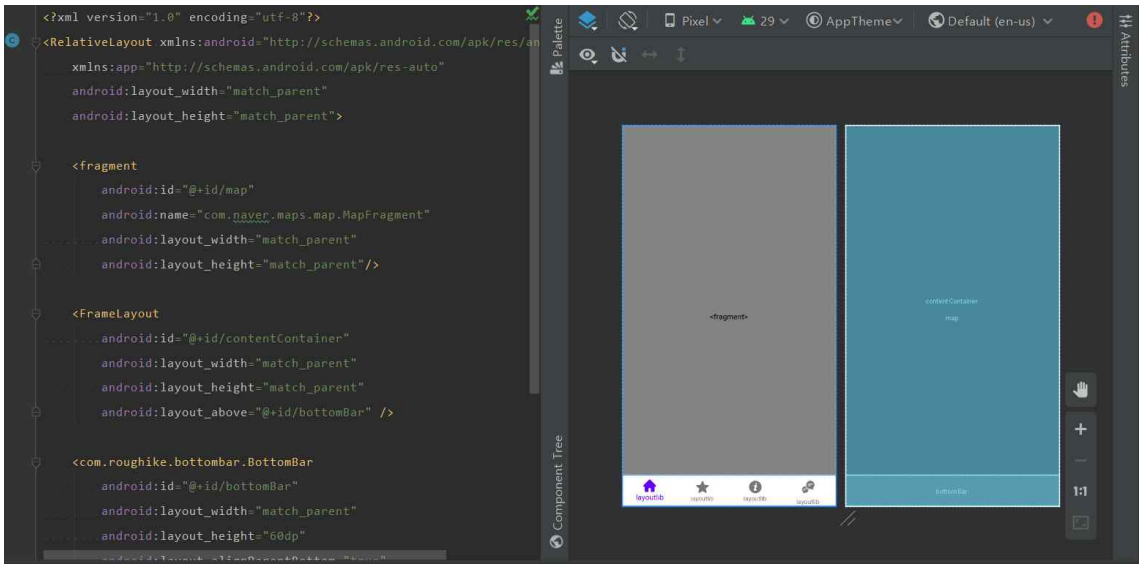
```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState){
return inflater.inflate(R.layout.frag_feedback, container, false);
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
super.onViewCreated(view, savedInstanceState);
memoList = new ArrayList<>();

dbHelper = new SQLiteHelper(getContext());
memoList = dbHelper.selectAll();

recyclerView = view.findViewById(R.id.recyclerview);

LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(getContext());
recyclerView.setLayoutManager(linearLayoutManager);

recyclerAdapter = new MemoAdapter(memoList);
recyclerView.setAdapter(recyclerAdapter);
btnAdd = view.findViewById(R.id.btnAdd);

btnAdd.setOnClickListener(new View.OnClickListener() { // 새로운 메모 작성부분
@Override
public void onClick(View v) {
Intent intent = new Intent(getActivity(), AddActivity.class);
startActivityForResult(intent, 0);
}
});
```

```
dbHelper = new SQLiteHelper(getContext());
memoList = dbHelper.selectAll();
}

@Override
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
super.onActivityResult(requestCode, resultCode, data);
if (requestCode == 0 && data != null) {
String strMain = data.getStringExtra("main");
String strSub = data.getStringExtra("sub");

Memo memo = new Memo(strMain, strSub, 0);
recyclerAdapter.addItem(memo);
recyclerAdapter.notifyDataSetChanged();

dbHelper.insertMemo(memo);
}
}
}
```

=> 사용자들이 피드백을 텍스트로 작성하고 버튼을 누르면, 앱 화면에 메모 형식으로 저장되도록 구현하였다. 버튼 클릭이벤트를 사용하였으며 입력 버튼을 누르면, 또 다른 레이아웃 화면으로 넘어가고 피드백 글이 저장되는 모습을 볼 수 있다.

2) app>res>layout

▶ **activity_main.xml**



=> 메인 화면의 하단 메뉴를 나타내기위해 작성한 코드와 결과화면이다.

▶ **frag_feedback.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
...>

<androidx.recyclerview.widget.RecyclerView
android:id="@+id/recyclerview"
android:layout_width="match_parent"
android:layout_weight="1"
android:layout_height="0dp"/>

<Button
android:id="@+id/btnAdd"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Add"/>

</LinearLayout>
```

▶ **activity_add.xml**

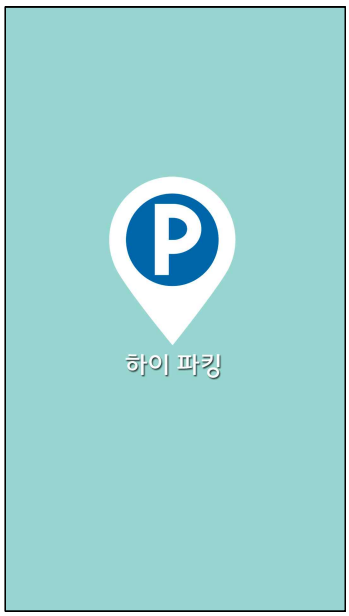
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
...>

<EditText
android:id="@+id/edtMemo"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_centerVertical="true"
android:hint="메모를 입력하세요"/>

<LinearLayout
android:padding="10dp"
android:layout_below="@+id/edtMemo"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="horizontal">

<Button
android:id="@+id/btnNo"
android:layout_marginRight="15dp"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="cancel" />

<Button
android:id="@+id/btnDone"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="done" />

</LinearLayout>
```

▶ **list_item.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
...>

<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
...>

<TextView
android:id="@+id/item_maintext"
android:layout_weight="1"
android:layout_width="0dp"
android:layout_height="match_parent"
.../>

<ImageView
android:id="@+id/item_image"
android:background="@color/colorPrimary"
android:layout_width="30dp"
android:layout_height="30dp"/>

</LinearLayout>

<TextView
android:id="@+id/item_subtext"
android:layout_width="match_parent"
android:layout_height="wrap_content"
... />

</LinearLayout>
```

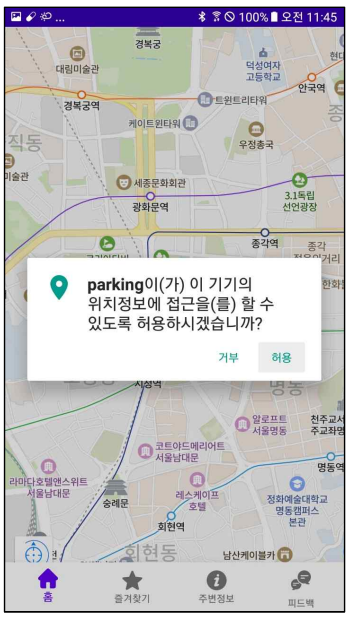
=> 위의 세가지 xml은 피드백 화면을 구현하기 위해 작성한 코드이다.

5. 결과물 분석



< 앱 실행 로딩화면 >

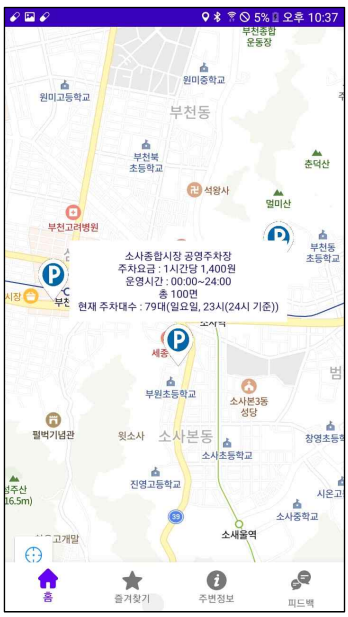
앱이 실행되는 약 1초 동안 나타나는 로딩화면이다. 사용자가 앱 실행시 '하이파킹'을 인지하고 실행창과의 자연스러운 연결을 위해 로딩화면을 설계하여 UI로 추가하였다. '하이파킹'은 높은 편의서비스 제공을 위한 'high'와 많은 시민들이 반갑게 사용할 수 있다는 의미로 'hi'를 축약해서 '하이파킹'이라고 정하게 되었다.



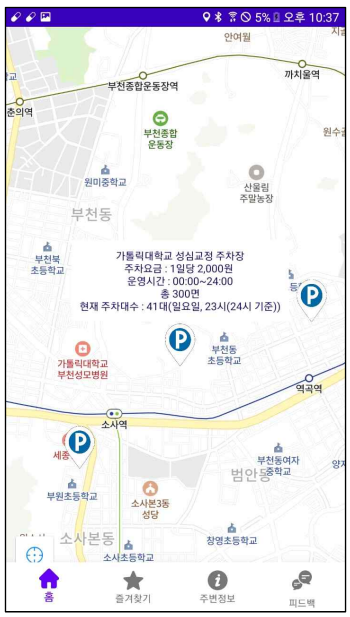
< 앱 실행 초기 화면> < 현재 위치정보 접근 허용 창 >

로딩화면이 끝난 후 화면에 지도를 볼 수 있도록 설정했다. 지도 하단에 보이는 나침반 모양의 버튼을 누르면 기기의 GPS접근 허용 메시지가 뜨고, GPS 허용을 하면 현재 위치한 장

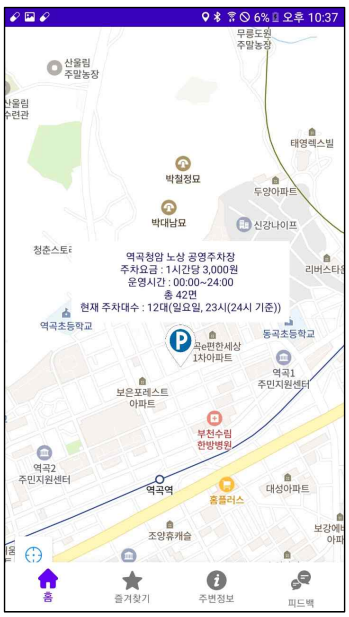
소로 지도가 로딩된다. 주변에 표시된 공영주차장 마커를 확인할 수 있으며, 현재 시범운영 중인 부천시 공영주차장 5곳이 표시됨을 확인할 수 있다. 왼쪽부터 순서대로 '소사종합시장 공영주차장', '가톨릭대학교 성심교정 주차장', '역곡청암 노상 공영주차장', '송내북부 공영주차장', '부천남부역 공영주차장'을 나타낸다. 하단 메뉴바에는 지도 위에 표시된 공영주차장 창이 나타나는 홈, 사용자가 즐겨찾기로 설정한 공영주차장, 편의시설, 전통시장과 가까운 공영주차장을 비교할 수 있는 주변 정보, 사용자가 불편사항이나 피드백을 입력할 수 있는 피드백 버튼이 있다.



< 소사종합시장 공영주차장 정보 >



< 가톨릭대학교 성심교정 주차장 정보 >



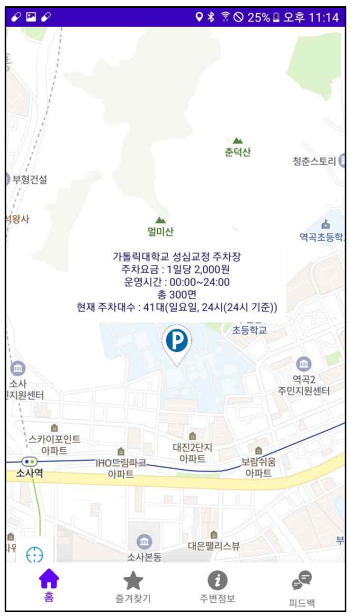
< 역곡청암 노상 공영주차장 정보>

< 송내북부 공영주차장 정보 >

< 부천남부역 공영주차장 정보 >

주차장에 표시된 아이콘을 누르면 주차장 이름과 주차요금, 운영시간, 주차 면수, 현재 주차대수를 확인할 수 있다. 실시간 평균 데이터값을 기반으로 현재 주차대수가 제공됨을 볼 수 있으며, 1시간을 기준으로 평균 주차대수 정보가 제공되고 있다. 프로젝트를 계획할 당시에 잔여 대수를 산정하여 표시하는 방법으로 구상했지만, 공공데이터 포털에서 제공하는 현재 주차대수 값을 얻을 수 있는 오픈 API의 cur_parking 값의 사용이 중지되어 자체적으로 데이터를 구축하는 방법으로 수정하였다. 첨부한 화면은 오후 10시 36-7분에 11시 기준으로 현재 주차대수를 나타내고 있으며, 시간별로 변하는 값을 다음 그림에서 볼 수 있다. 아래에 첨부된 화면은 오후 11시 14-15분에 12시 기준으로 현재 주차대수 데이터를 제공하고 있음을 확인할 수 있다.

< 소사종합시장 공영주차장 정보 >

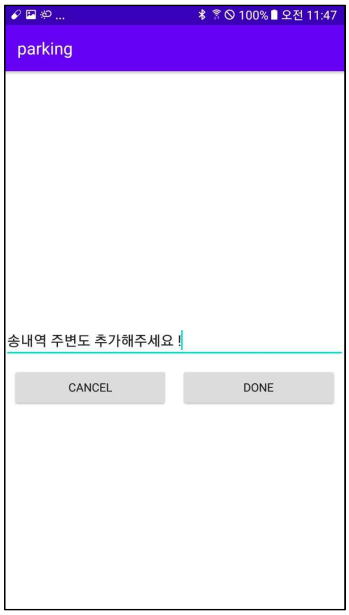


< 가톨릭대학교 성심교정 주차장 정보 >

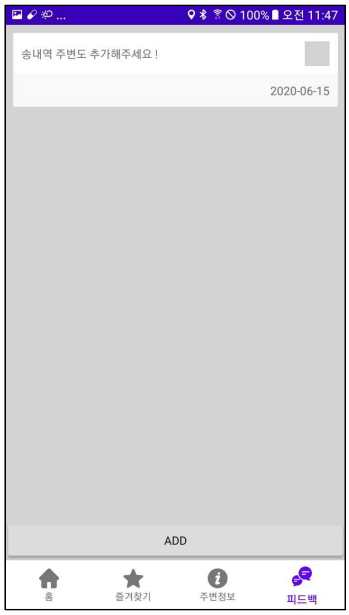
< 역곡청암 노상 공영주차장 정보>

< 송내북부 공영주차장 정보 >

< 부천남부역 공영주차장 정보 >



< 피드백 작성 창 >



< 피드백 작성 완료 화면 >

하단 메뉴 중 피드백 메뉴를 버튼을 클릭하면, 불편사항이나 건의할 부분을 적을 수 있는 textView가 나타난다. 원하는 내용을 적고 'DONE' 버튼을 눌러 완성하면 피드백 정보가 저장되었음을 확인할 수 있다. 추후, 관리하는 서버를 구축하고 연결해서 사용자가 남긴 피드백에 대댓글을 작성하고 응답할 수 있는 시스템으로 확장하고자 한다.

6. 고찰 및 결론

'하이파킹'은 수시로 바뀌는 주차장의 실시간 정보를 빅데이터 화하여 사용자들에게 제공함으로써 주차난을 극복할 수 있고, 사용자에게 편리함을 줄 수 있다. 현재 전국의 주차장은 공영주차장 뿐만 아니라, 민간 주차장 수도 많아 정확한 주차장 정보수집이 어려우며 실시간 주차 대수 등의 정보는 보안과 민원 등의 이유로 열린 공공데이터를 받아볼 수가 없다. 이러한 어려움을 극복하기 위해 우리는 자체적으로 data를 수집하고, 현재 몇 대의 차량이 주차되어 있는지 실시간 정보를 빅데이터로 만들어 앱에 불러오도록 알고리즘을 구현한 것이다. 현재는 부천시의 주요 주차장만 표시하였지만, 피드백 기능을 극대화하여 앱 사용자 및 주차장 관리자의 의견을 적극 수렴하여 더 많은 데이터를 수집하고 지역을 확대하면, 신뢰성 높고 정확한 정보를 제공하는 앱으로 개선이 가능하다. 또한, 주차장과 주변 편의시설(영화관, 역, 전통시장 등)들을 상호 연결하여 사용자의 위치와 가장 가까운 주차장을 알려주는 기능이 있다. 이는 하이파킹 사용자라면, 편의시설을 이용하기 위해 어렵게 주차장을 찾을 필요 없이 앱에서 근처의 주차장을 알려주기 때문에 원활한 주차장 사용을 가능하게 하고, 전통시장 및 관광지 등에서 교통 혼잡을 감소시켜 지역사회의 교통 및 주차난을 극복할 수 있으며 시장 이용의 활성화로 지역사회의 경제 가치를 향상시킬 수 있을 것으로 기대한다.