

# KOTLIN 프로그래밍 최종 보고서

학번	32161422, 32154908, 32164820, 32152377
이름	두영주, 한남경, 하니, 신동환

## 【제출 요령】 / 제출시 작성요령은 삭제

☞ 제출 파일 리스트

1. 최종 보고서, 2. 소스코드, 3.프로그램 구동 동영상

(초기 바탕화면에서 이클립스 실행 → 프로그램 실행 → 프로그램 수행이 모두 담긴 동영상 제출)

## 1. 프로젝트 요약문(연구목표 및 최종결과물) (두영주 32161422)

### 【작성요령】 / 제출시 작성요령은 삭제

☞ 개발 결과물의 내용을 토대로 요약해서 작성

구분	내용
과제개요	각 팀원들의 타국에서의 학교생활과 경험들을 바탕으로 불편했던 점과 개선되어야 할 점들을 모아 단국대학교에 재학 중인 외국인 학생들이 학교생활을 좀 더 편리하게 할 수 있도록 도와주는 어플리케이션을 연구 개발했다. 실제 사용되는 정보 관련 어플리케이션들을 바탕으로 외국인 학생들의 단국대에서의 생활에 도움이 될 수 있도록 구성을 짜고 디자인 하였다. 외국 친구들에게 각종 놀거리와 볼거리 등을 제공하고자 가는 방법 등을 자세히 알려주는 기능과, 새로운 친구들을 많이 만나 볼 수 있는 기회를 제공하고자 여러 가지 이벤트를 생성하여 캘린더에 표시하고, 다른 학생들과 멘토 멘티 시스템을 통해 관계를 지을 수 있게 게시판 기능을 제공하였다.
최종 목표	연구 개별결과물로 다음과 같은 내용을 산출함 - 다이얼로그를 이용한 로그인 팝업창 기능 - 게시판 기능을 데이터베이스를 이용해 이벤트를 만들거나 멘토멘티 시스템 으로 친구를 만들 수 있는 기능 - 이벤트가 생성됨과 동시에 자동으로 캘린더의 지정된 날짜에 이벤트 내용 저장 - 학교 주변 편리시설, 쇼핑시설, 유통시설 등을 상세하게 알려주는 정보 기능

## 2. Detailed description of the task (Hani 32164820)

### 【Instructions】

- ☞ Organize based on the contents of the first week of the project

On the first week of the class, we were divided into a group of four, consists of 두영주, 신동환, 한남경, and Hani and we were told that every Tuesday we will be having discussion class while on Thursday, we will be learning how to use Kotlin up until we had our mid-term examination. We were given a task of discussing amongst ourselves what comes to mind when we think of Android and draw them in a mind map. After we were done discussing, we were told by the professor to choose a topic each and make individual mind maps by next week's class.

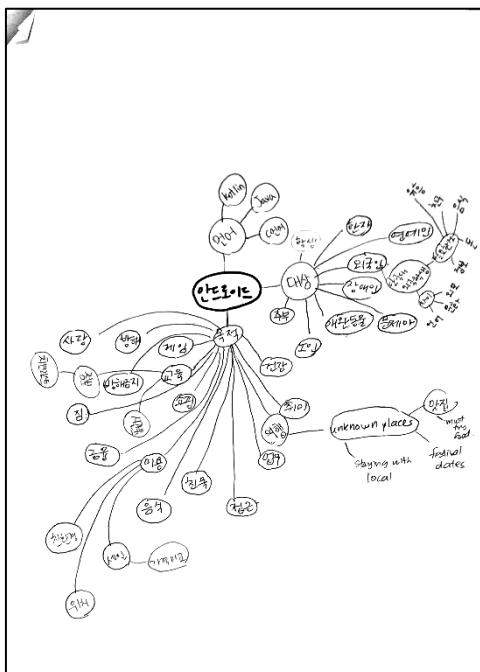
The following week, the four of us brought our mind maps and explain to each of the members what kind of application we wish to make. We decided to create an application to ease foreign students that comes to Dankook University for an exchange, as we had an experience of being in a foreign country before and did not receive enough help. For the next week, we were given a task to individually create pin cards consisting the features we wish to put in our application.

The next week, we brought our pin cards and we were told to write our comments on our members' ideas for the features and after choosing the features we decided to do, we created our final mind map and explain them to our professor. We decided to make 2 features, that is "Community" and "Places". The "Community" part will consist of "Event" tab where students can make appointments for tutoring or party, "Calendar", where the appointments will pop up at, and "Mentoring" tab where students can apply to be a mentor or a mentee while they are in Korea. For the "Places" part, it consists of the details on how to go to places the students desire to go, such as Pub, Club, Amusement Parks, Supermarkets etc. After that, we were given a task of drawing our screen interfaces.

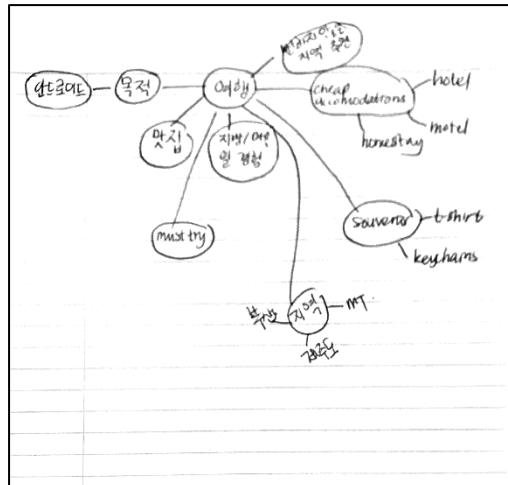
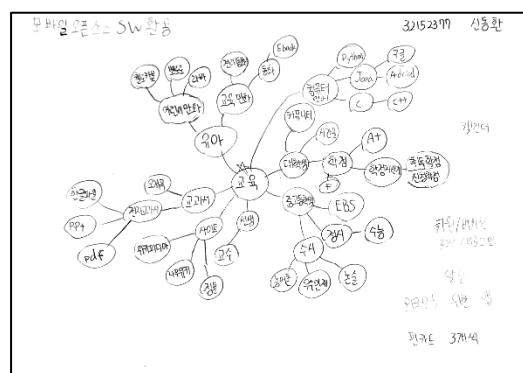
Next, the task we were told to bring was the drawing that was given the previous week. We individually explain our drawings to the team. After we finished explaining to each other, we decide on a common theme we would want our application to look like, so we had to make another drawing with the chosen theme and after that we were given a task of drawing another screen interface consisting the representative screens of each features.

After we completed our mid-term examination, we started using Android Studio in our classes. Our professor taught us how to create layouts for our application. After being taught all the basics, such as using the linear layout, how to insert buttons, images, texts etc to the layout, we were given a task of making the screen interfaces like the ones we had vision for in our drawings for the next 2 weeks. After we completed all layouts, we started working on our kotlin code for our application.

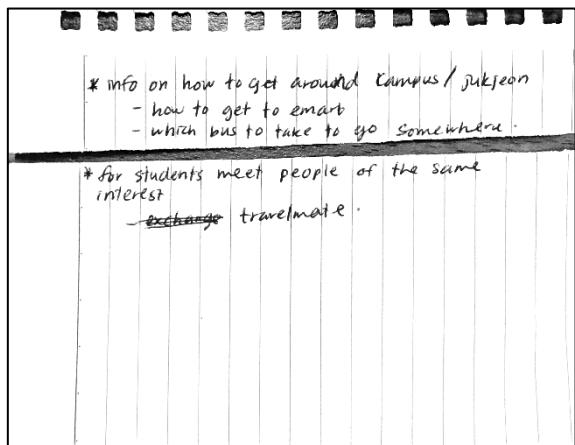
We were given time until the end of the semester to complete our application with progress check every week. Before sending in our project, the professor gave us a final feedback asking us to fix certain things and if it was possible, to add one more feature in the application. Thankfully, we were able to fix everything and did what was asked for us to do in time before our project due date.



Mind map with "Android" as topic

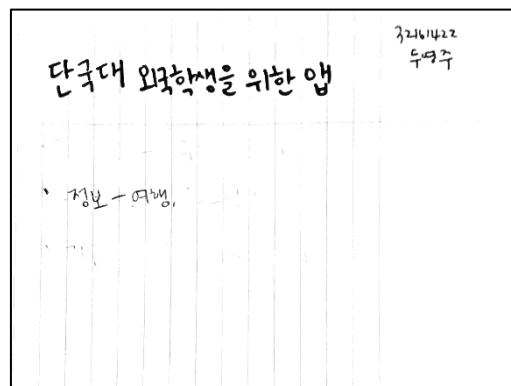


Individual mind maps with desired topic

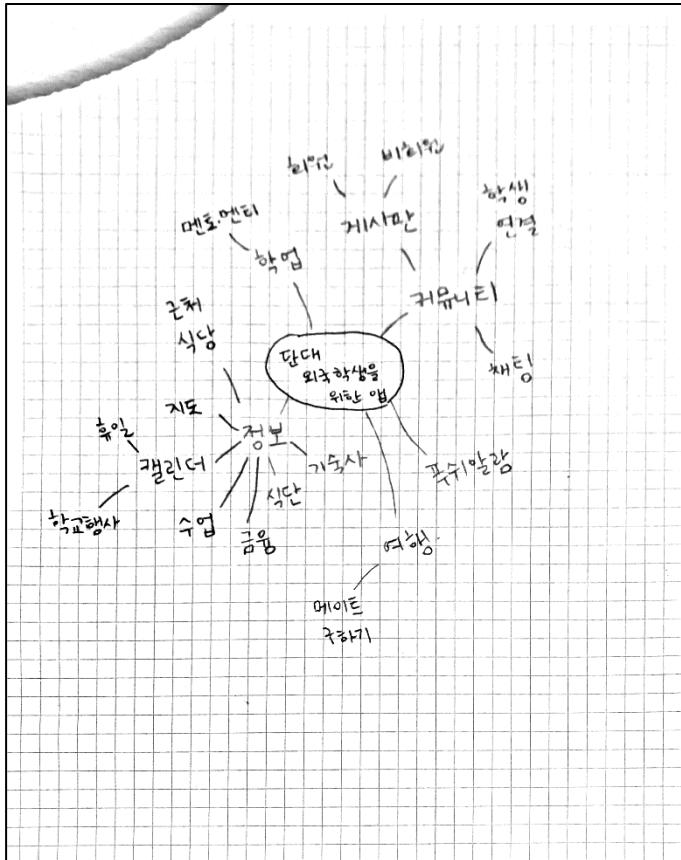


<b>시간표</b> - 강의 알림 (매일 아침에 알려줌) - 몇시 ~ 몇시  <small>신동환 32152371</small>	<b>캠퍼스 주변 카페</b> <b>구글 어스</b> <b>위치 알려줌 (GPS)</b>  <small>신동환 32152371</small>
<b>SNS / 커뮤니티</b> - 이브리 타임: 비슷한 연할 - 교류 - 만남 - 랜덤채팅  <small>신동환 32152371</small>	<b>학점 계산기 / 수강 계획서</b> - 채과리스트 흥식 - 차기년도 학교 (4, 3, 4, 5, 등) <small>신동환 32152371</small>

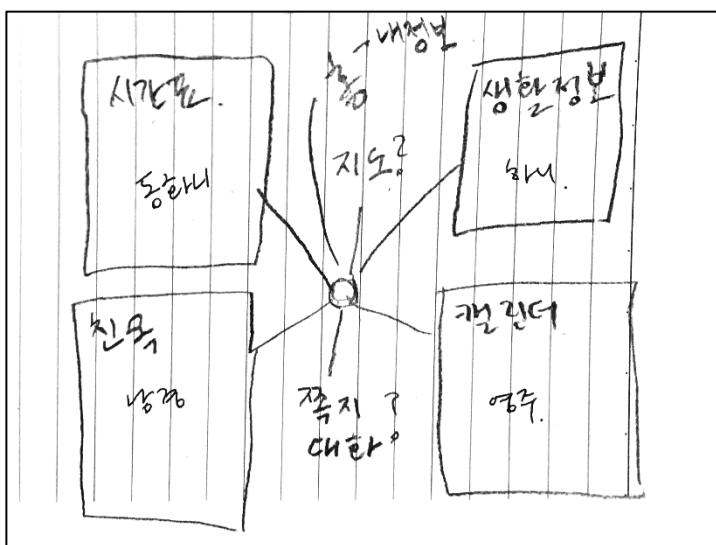
<b>교환학생, 외국학생 관련 정보를 얻을 수 있는 앱.</b> 학교에서 주관하는 행사나 교육적 컨텐츠 등에만 중점적으로 정보를 알려줌. 학교 관련 다양한 정보는 단국대학교에 잘나와있으므로 거기서 살피우는 정보 중심으로 몰린다. 관리자 주관한 공지들도 있지만 서로 알려주고 싶은 정보나 홍보글도 편집 등해 올릴 수 있다. 각 회원들에게 차등적 권한을 부여하고 게시물을 올리고 답글을 볼 수 있다.  <small>3/21/2018 한상근</small>	<b>학생들 간의 커뮤니티가 주축인 앱.</b> 관리자가 학생간 연결을 해주거나, 학생이 직접 멘토, 멘티, 동행, 친구 등을 찾을 수 있다.  <small>3/21/2018 한상근</small>
---	--

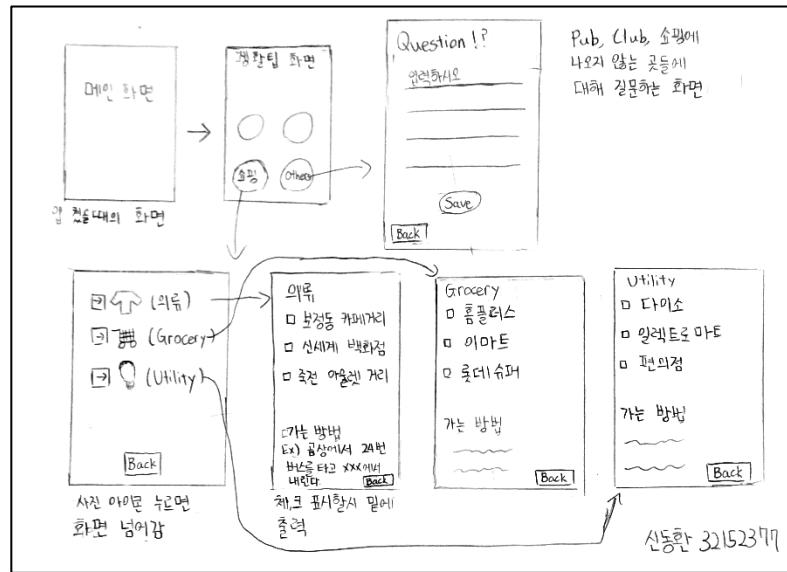
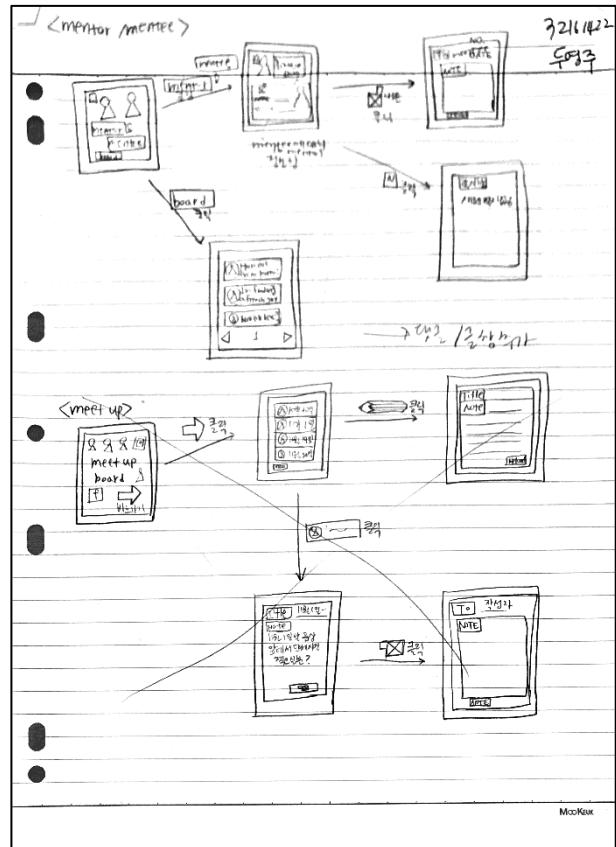
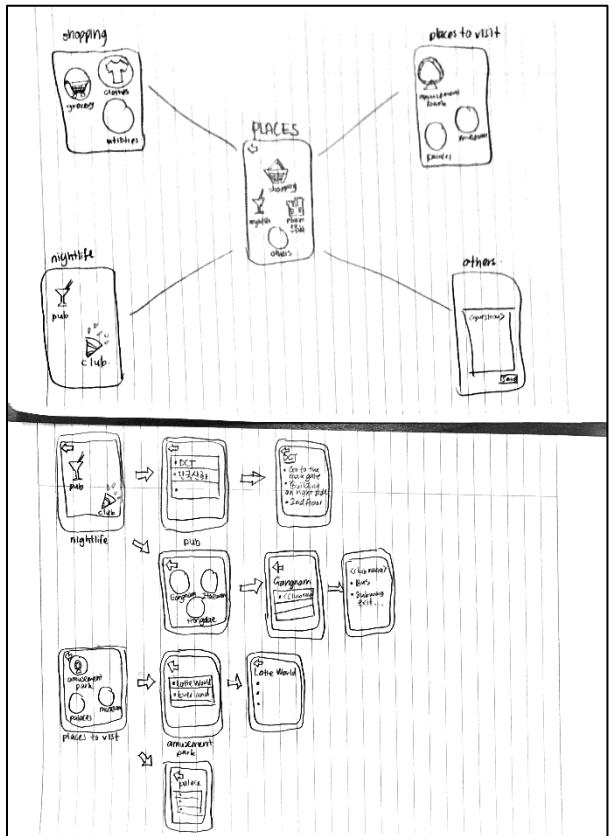


Above are the pincards created by the members consisting the features we want in our application



Final mind map we created with  
the chosen features

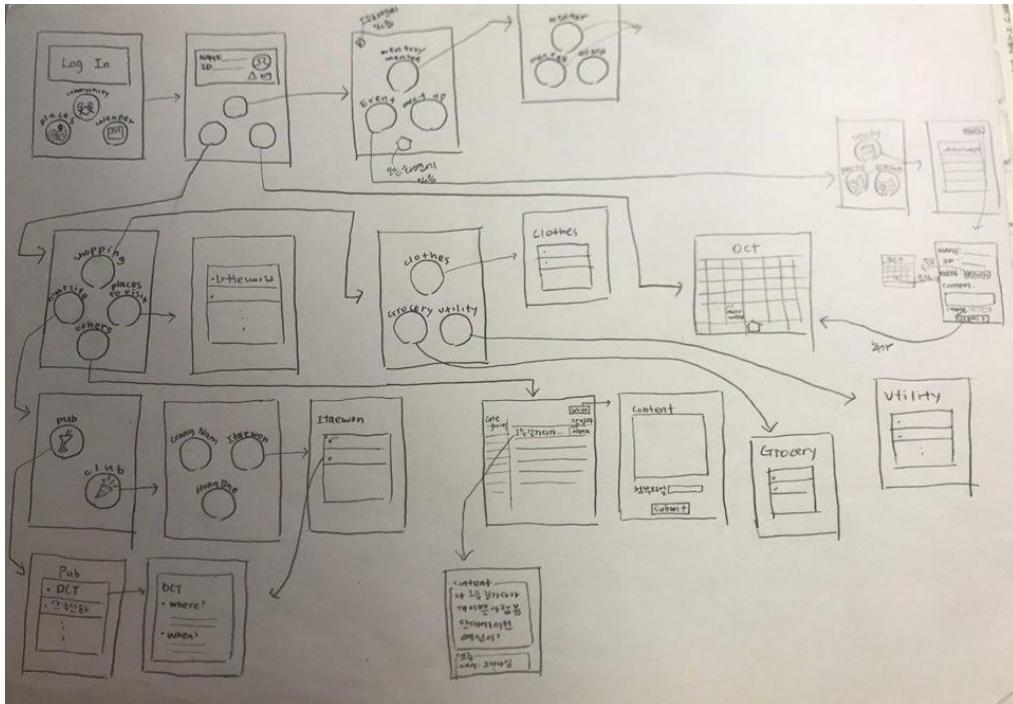




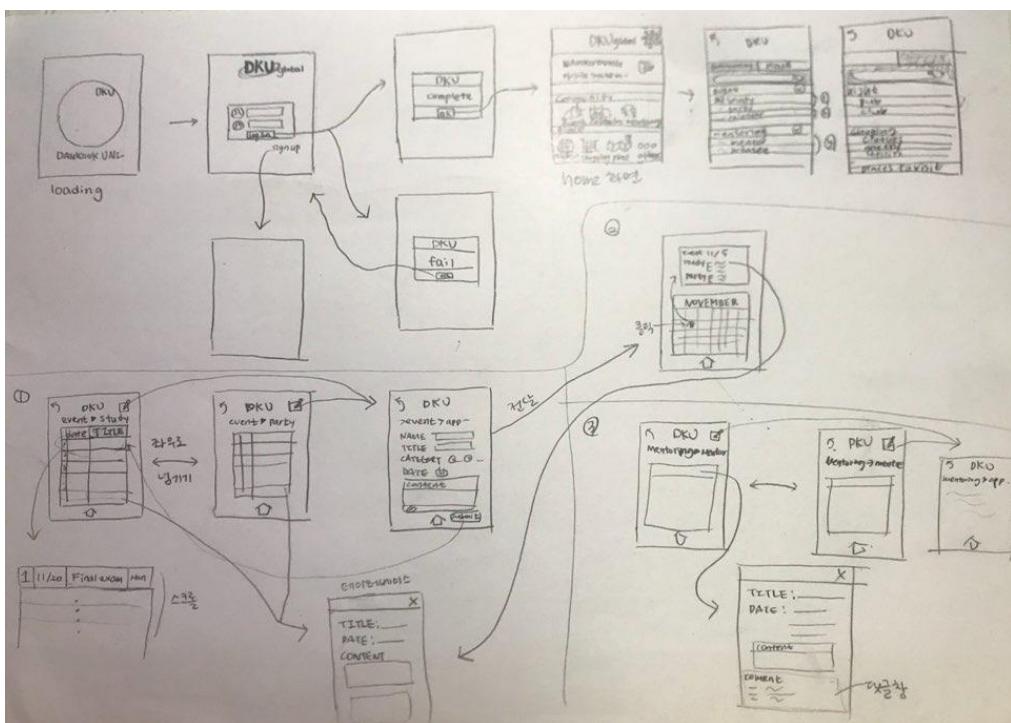
Drawing consisting the screen interface layouts for our application

### 3. 과제 기능별 설명 – (한남경 32154908)

#### 1) 전체 클래스 다이어그램



처음에 짬던 그림



데이터베이스 들어가는 부분 조금 수정해서 다시 그렸던 그림

## 2) 세부 클래스 분류 및 진행상황

해당하는 content 띄우기	진도	계획	진도	시간부족	
community, place에서 스피너 사용					0
아이콘 디자인	계획	진도			100

### 3) 세부 클래스 참조 문서

finish하는 back버튼과 home화면으로 가는 home버튼 등 기능에 중요하지 않은 부분은 생략하겠습니다.

## Splash 클래스

### ▶ 설명

login화면을 띄우기 전에 splash 화면을 먼저 띄운다.

### ▶ 속성

이름	설명
background	thread

### ▶ 메서드

이름	설명
run	thread를 750초 동안 멈추고 그 뒤에 login화면으로 간다

### ▶ example

```
val background = object : Thread(){
override fun run(){
try{
Thread.sleep(750)

val intent = Intent(baseContext, login::class.java)
```

```
startActivity(intent)
} catch (e: Exception){
e.printStackTrace()
}
}
background.start()
```

## login 클래스

### ▶ 설명

login 버튼을 누르면 dialog가 뜨고 ok를 누르면 home 화면으로 넘어간다.

### ▶ 속성

이름	설명
imgbtn_login	setOnClickListener로 버튼 클릭 시ダイ얼로그 켜지게 함
builder	AlertDialog.Builder를 받아서 setView로 dialogview를 보여줌
dialogView	custom_dialog.xml파일을 인플레이트한다
nextIntent	home화면으로 넘어감

### ▶ example

```
builder.setView(dialogView)
.setPositiveButton("Ok") { dialogInterface, i ->
var nextIntent = Intent(this, home::class.java)
startActivity(nextIntent)
}
.show()
```

## home 클래스

### ▶ 설명

버튼을 눌렀을 시 community 또는 place로 넘어간다.

로그아웃 버튼을 누르면 로그아웃 된다.

### ▶ 속성

이름	설명
imgbtn_logout	logout_check화면으로 넘어가고 ok하면 로그인화면 아니면 다시 home
btn_event, btn_mentoring	community화면으로 넘어감
btn_calendar	calendar화면으로 넘어감
btn_nightlife, btn_shopping, btn_placestovisit	place화면으로 넘어감

## community 클래스

### ▶ 설명

event\_study, event\_party, event\_calendar, mentor\_board, mentee\_board 화면으로 넘어감

### ▶ 속성

이름	설명
btn_event_study	event_study 화면으로 넘어감
btn_event_party	event_party 화면으로 넘어감
btn_event_calendar	event_calendar 화면으로 넘어감
btn_mentoringmentor	mentor_board 화면으로 넘어감
btn_mentoringmentee	mentee_board 화면으로 넘어감
btn_communitytoplance	place화면으로 넘어감

## DBHelper\_Party, DBHelper\_Study 클래스

### ▶ 설명

각자 PartyTable과 StudyTable을 생성한다. (idx, dateData, titleData, nameData, contentData를 갖는다) helper가 하는 역할은 앱을 다시 열 때마다 다시 만들어 지는 것을 방지하는 것이다. 이는 처음 한 번만 실행 될 수 있게 도와주는데, 그래서 앱을 삭제하지 않는 이상 데이터는 계속 남아있다.

### ▶ 속성

이름	설명
sql	create문을 받는다.

# ListViewAdapter 클래스

## ▶ 설명

## ▶ 속성

이름	설명
listVO	ListVO클래스 타입 array
convertView	getView에서 parameter로 들어온 convertView를 받고 custom_listview를 inflate함
context	parent.context를 받음
index, numb, date, name	custom_listview에 있던 index, numb, date, name을 findViewById한뒤 텍스트 값에 listVO[position]에 있는 index, numb, date, title, name을 넣어줌

## ▶ 메서드

이름	설명
getCount()	listVO의 size를 리턴
getView(position, convertView, parent)	리스트뷰에 데이터를 넣어줌. 아이템 내 각 위젯에 데이터를 반영해줌
getItem	listVO의 position번째 item을 리턴
getIndex	listVO의 position번째 item의 index를 리턴
getType(position)	listVO의 position번째 item의 type을 리턴
addVO	ListVO타입 item에 요소 하나하나(index, numb, dat, name, title)를 다 넣어주고 listVO배열에 item을 넣어준다
clearVO	listVO배열을 비운다

## ▶ example

getView에서

```
val index = convertView.findViewById<TextView>(R.id.index)
index.text = listViewItem.index
```

addVO에서

```
item.index = index
```

```
listVO.add(item)
```

# ListVO 클래스

## ▶ 설명

리스트뷰에서 한 줄 안에 있는 요소들을 담고있음

## ▶ 속성

이름	설명
index	인덱스를 저장. 하지만 출력은 하지않을 것. 나중에 캘린더에서 study와 party가 섞여서 출력된 리스트뷰를 클릭했을 때 내용으로 이동할 때 원래의 인덱스 위치를 찾기위해 만들어 뒀다. 섞여서 출력 될 때는 numb는 데이터베이스에서의 index와는 다르기 때문
numb	데이터베이스에서 index를 담음
date	데이터베이스에서 dateData를 담음
title	데이터베이스에서 titleData를 담음
name	데이터베이스에서 nameData를 담음

# event\_study 클래스

## ▶ 설명

Cursor를 이용해서 리스트뷰에 데이터베이스 내용을 뿌려주고, 리스트뷰에서 아이템을 클릭했을 때 해당하는 화면으로 넘어감

## ▶ 속성

이름	설명
btn_event_study_write	event_application 화면으로 넘어감
listview	xml파일에 있는 listview를 id로 찾아서 받음
adapter	만들어 놓은 ListViewAdapter 클래스 생성자를 받아서 listview.adapter에 넣어줌
helper	DBHelper_Study를 받음
db	helper.writableDatabase를 받음
sql	StudyTable에서 모든 데이터를 select하는 sql문
c	sql문을 읽고 나서 상응하는 데이터베이스를 가리키는 커서이다. 커서를 차례로 아래로 내려가서 하나하나 리스트뷰에 addVO로 뿌려줄것
helper2, db2, sql2, c2	listview의 item을 클릭했을 때 position값에 해당하는 데이터를 찾기위해 다시 db사용
intent	index값과 테이블 이름을 넘겨주며 event_content 화면으로 이동
numb, date, Title, Name	커서가 가리키고 있는 index, date, title, name 데이터들을 각각 받고 이 변수들을 add해 줄 것임

## ▶ example

```
listview!!.setOnItemClickListener { parent, view, position, id ->
    var helper2 = DBHelper_Study(this)
    var db2 = helper2.writableDatabase

    var sql2 = "select * from StudyTable"
    var c2: Cursor = db2.rawQuery(sql2, null)

    c2.moveToFirst()
    var intent = Intent(this, event_content::class.java)
    intent.putExtra("value", numb)
    intent.putExtra("type", "StudyTable")
    startActivity(intent)

    db2.close()
}
```

```
var helper = DBHelper_Study(this)
var db = helper.writableDatabase

var sql = "select * from StudyTable"

var c: Cursor = db.rawQuery(sql, null)

adapter = ListViewAdapter()

listview = findViewById<ListView>(R.id.Study_List_view)

//어댑터 할당
listview!!.adapter = adapter

while(c.moveToNext()) {
    ----numb,date,title,name 에 값 넣어 줌-----
    adapter!!.addVO(numb, numb, date, Title, Name)
}
db.close()
```

## event\_party 클래스

### ▶ 설명

event\_study와 동일하지만 사용하는 데이터베이스 테이블이 PartyTable임

## event\_application 클래스

### ▶ 설명

event를 원하는 대로 입력해서 submit을 하면 입력된 정보를 데이터베이스에 넣어줌

### ▶ 속성

이름	설명
selected	StudyTable 또는 PartyTable을 받음
rdGroup	setOnCheckedChangeListener를 사용해서 라디오버튼 클릭된 항목에 맞게 selected 값을 바꿔줌
btn_event_date_calendar	클릭하면 애니메이션으로 layout_calendar를 띄워준다. 아래에서 위로 올라오는 애니메이션이다.
application_calendar	캘린더의 날짜를 클릭하면 app_date에 그 날짜 값을 넣어주고 calendar는 없어진다.
app_date	클릭된 날짜값을 기억
btn_event_submit	버튼을 눌렀을 때 입력된 text값들과 선택된 date값을 데이터베이스에 입력해줌. 그리고 event_study 또는 event_party화면으로 다시 넘어간다. 이때 하나라도 입력되지 않은 값이 있다면 toast로 "insert all information"을 띄우고 sql문은 실행하지 않는다.
sql	dateData, titleData, nameData, contentData를 insert하는 sql문. select된 table에 insert 한다.

### ▶ example

```
layout_calendar.startAnimation(Animshow)
layout_calendar.setVisibility(View.VISIBLE)

application_calendar.setOnDateChangeListener(CalendarView.OnDateChangeListener
{ view, year, month, dayOfMonth ->
layout_calendar.startAnimation(Animgone)
layout_calendar.setVisibility(View.GONE)

app_date = (month+ 1).toString() + "/" + dayOfMonth.toString()
})
```

```
if(selected == "StudyTable") {  
    if(app_date == "0" || edt txt_title.getText().toString().equals("") ||  
        edt txt_name.getText().toString().equals("") ||  
        edt txt_event_content.getText().toString().equals("")) {  
        Toast.makeText(applicationContext, "insert all information",  
        Toast.LENGTH_LONG).show()  
        db1.close()  
        db2.close()  
    }  
    else {  
        db1.execSQL(sql, arg1)  
        db1.close()  
        db2.close()  
        var intent = Intent(this, event_study::class.java)  
        startActivity(intent)  
    }  
}
```

# event\_content 클래스

## ▶ 설명

이전의 화면에서 넘어온 값(table이름, index)들을 이용한다. 테이블을 확인한 후 그 테이블 데이터베이스를 사용해서 Cursor가 index번째에 있는 데이터들을 text에 넣어줘서 출력해줌.

## ▶ 속성

이름	설명
intent	이전 화면에서 넘겨준 값을 받음
helper1, db1, helper2, db2	일단 DBHelper_Study와 DBHelper_Party를 모두 사용 허락을 받는다.
type	intent.getStringExtra("type")값을 받는다. type은 StudyTable 또는 PartyTable이다.
sql	"select * from" + type을 해서 해당하는 테이블에서 값을 select하는 sql문
c	sql을 실행해서 커서를 둔다. 차례로 옮겨서 이전 화면에서 넘겨준 값과 인덱스 값이 같을 때 text값을 넣는다.

## ▶ example

```
if(type == "StudyTable") {  
    txt_study_or_party.text = "study"  
    Log.d("yang", "studytable " )  
    var c: Cursor = db1.rawQuery(sql, null)  
    while(c.moveToNext()) {  
  
        ---- numb, title, name, content에 값 넣어줌 ----  
  
        if(numb == intent.getStringExtra("value")) {  
            txt_event_content_title.text = Title  
            txt_event_content_name.text = Name  
            txt_event_content.text = Content  
            break  
        }  
    }  
}
```

# calendar 클래스

## ▶ 설명

StudyTable과 PartyTable에 있는 데이터 베이스를 이용해서 date값에 맞는 event를 보여주고 그 event를 클릭하면 event\_content화면으로 이동해서 내용을 보여줌

## ▶ 속성

이름	설명
isPageOpen	calendar가 이미 열려져있으면 chagedate로 date가 변했는지 확인하고 변하지 않았으면 calendar를 닫아줌
pre_date	이전의 date값을 저장. 새로 입력된 데이터값과 비교해서 chagedate 변수를 바꿔준다
chagedate	date가 바뀌었는지 알려줌
cal_listview	event들을 담고있는 리스트뷰. 선택된 데이터 값에 해당되는 것들을 표시함
adapter	ListViewAdapter()
calendar	캘린더의 날짜를 클릭했을 때 애니메이션으로 event가 띄워진다. 이때 데이터베이스 cursor를 이용한다. date값이 선택된 날짜와 같을 때 리스트 뷰로 해당하는 event의 정보를 띄워준다. 이때, study와 party 데이터베이스를 모두 띄운다.

# DBHelper\_Mentor, DBHelper\_Mentee 클래스

## ▶ 설명

DBHelper\_study, DBHelper\_party와 동일하지만 nationData, subjectData만 추가되었다.

# mentor\_board, mentee\_board 클래스

## ▶ 설명

event\_study, event\_party와 동일하지만 calendar와 관련된 부분만 빠졌다.

# mentoring\_application 클래스

## ▶ 설명

event\_application과 동일하지만 nation, subject부분이 추가되었다.

## mentoring\_content 클래스

### ▶ 설명

event\_content와 동일하지만 nation, subject부분이 추가되었다.

## Place 클래스

### ▶ 설명

community와 동일하게 버튼을 눌렀을 때 그에 맞는 페이지로(pub, club, Cloth, Grocery, Utility, amusementpark, palace, museum)이동  
btn\_placetocommunity를 누르면 community페이지로 넘어간다.

## pub, club, Cloth, Grocery, Utility, amusementpark, palace, museum, hongdae, itaewon, gangnam 클래스

### ▶ 설명

shinhwa, laskal, hitehouse, brown, nb2, cakeshop, soap, octagon, Shinsege, Outlet, Bojeong, Emart, Lottemart, Homeplus, Daiso, Electro, everland, lotteworld, changdeok, changgyeong, deoksu, gyeongbok, nfmok, nmok, smoa, smoh, wmok 화면으로 이동

## MyViewPagerAdapter 클래스

### ▶ 설명

fragment 타입 배열인 fragmentList에 fragment를 추가하거나 리턴하는 클래스

### ▶ 속성

이름	설명
fragmentList	fragment 타입 배열

### ▶ 메서드

이름	설명
getItem(p0)	fragmentList[p0]을 리턴
getCount	fragmentList의 size를 리턴
addFragment(fragment)	fragmentList에 fragment를 추가

▶ example

```
fun addFragment(fragment: Fragment){  
    fragmentList.add(fragment)  
}
```

shinhwa, laskal, hitehouse, brown, nb2, cakeshop, soap,  
octagon, Shinsege, Outlet, Bojeong, Emart, Lottemart,  
Homeplus, Daiso, Electro, everland, lotteworld,  
changdeok, changgyeong, deoksu, gyeongbok, nfmok,  
nmok, smoa, smoh, wmok **클래스**

▶ 설명

상용하는 fragment들을 추가해준다

▶ 속성

이름	설명
adapter	MyViewPagerAdapter를 받음

▶ example

```
val adapter = MyViewPagerAdapter(supportFragmentManager)  
adapter.addFragment(fragshinhwa1())  
adapter.addFragment(fragshinhwa2())  
viewPagershinhwa.adapter = adapter
```

fragbojeong1,2,3, fragbrown1,2,3, fragcakeshop1,2,  
fragchangdeok1,2 fragchanggyeong1,2, fragdaiso1,2,3,  
fragdeoksu1,2, fragelectro1,2,3, fragemart1,2,3, frageverland1,2  
fraggyeongbok1,2, fraghitehouse1,2,  
fraghomeplus1,2,3, fraglaskal1,2, fraglottemart1,2,3,  
fraglotteworld1,2, fragnb21,2, fragnmfok1,2, fragnmok1,2,  
fragoctagon1,2, fragoutlet1,2,3, fragshinhwa1,2,  
fragshinsegae1,2,3, fragsmoa1,2 fragsmoh1,2, fragsoap1,2,  
fragwmok1,2

## 클래스

### ▶ 설명

상응하는 fragment xml파일을 inflate해서 리턴

### ▶ example

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
savedInstanceState: Bundle?): View? {  
    return inflater.inflate(R.layout.frag_shinhwa1, container, false)  
}
```

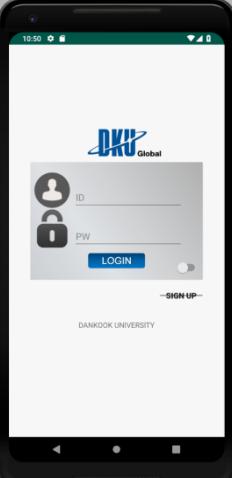
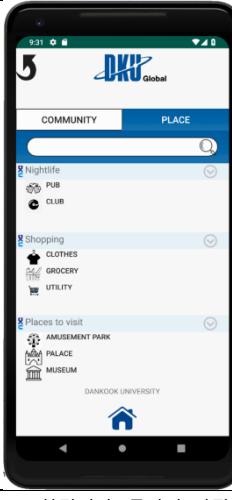
## 4. 프로그램 구성 (신동환 32152377)

### 1) 메뉴 구성도

【작성요령】 / 제출시 작성요령은 삭제

☞ 실행화면 기반 설명(필수)

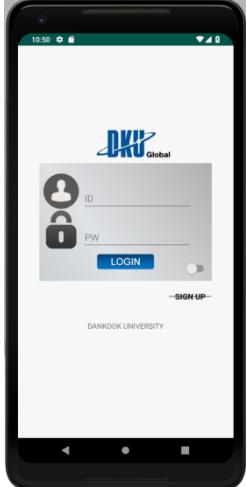
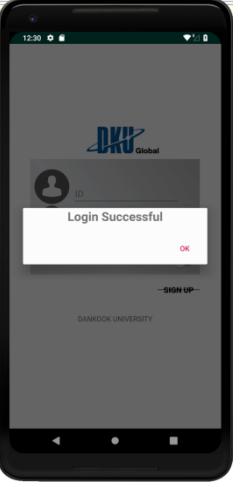
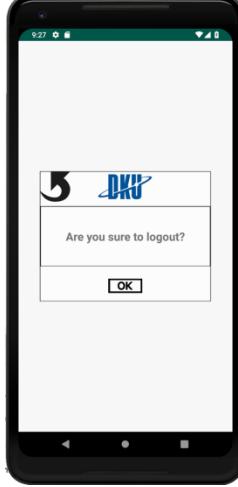
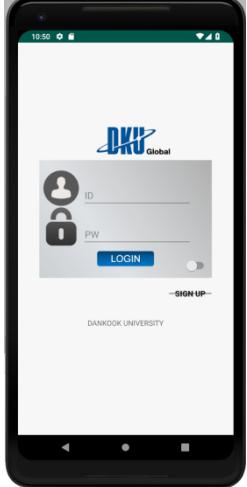
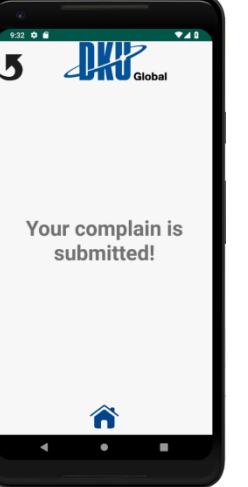
#### 메뉴 구성도 1

1) Splash 화면	2) 로그인 화면	3) 홈 화면	4) 로그아웃 화면
			
시작할 때 순간 나타나며 0.75초 뒤에 사라지는 화면	로그인하는 화면, 아이디+비밀번호는 아무거나 입력을 해도 넘어간다.	홈 화면이며, 화면들마다 아래 파란색 줄 아이콘을 누르면 이 화면으로 오게 된다.	로그아웃을 하는 화면이다.
5) Others(건의사항) 화면	5) Community 화면	5) Place 화면	6) 일정추가 화면
			
건의사항 화면이다.	Community 화면이다. 글자와 사진 중 아무거나 누르면 해당 화면으로 이동한다.	Place 화면이다. 글자와 사진 중 아무거나 누르면 해당 화면으로 이동한다.	Study, Party, Mentor, Meetee 버튼을 누르면 각각 이런 디자인의 화면이 나온다.

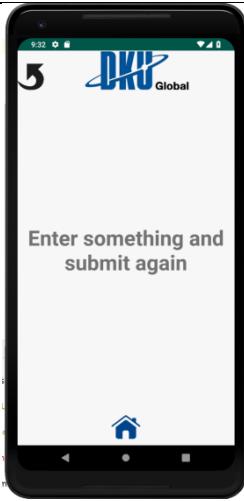
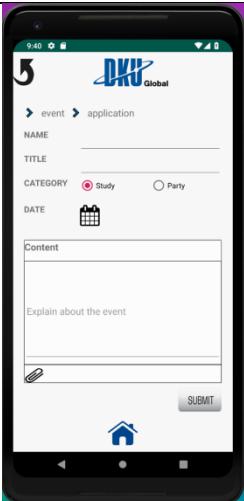
## 메뉴 구성도 2

7) 일정 적는 화면 1	7) 일정 확인하는 화면 1	8) 일정 적는 화면 2	8) 일정 확인하는 화면 2
Study, Party에서 일정을 적는 화면이다.	Study, Party 어플리케이션에서 적은 일정을 확인하는 화면이다.	Mentor, Mentee에서 일정을 적는 화면이다.	Mentor, Mentee 어플리케이션에서 적은 일정을 확인하는 화면이다.
9) 달력 화면	10) Place 정보 버튼화면	11) 정보화면 1	11) 정보화면 2
달력 화면이다.	'5) Place 화면'에서 아무 아이콘을 누르면 해당하는 카테고리에 맞게 이동하는 화면들이다.	'10) Place 정보 버튼화면'에서 넘어오며, 여기서는 해당 장소의 정보를 얻을 수 있다.	옆으로 넘기면서 정보 및 가는 방법을 알 수 있게 하는 화면이다.

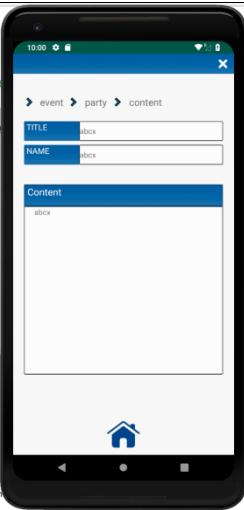
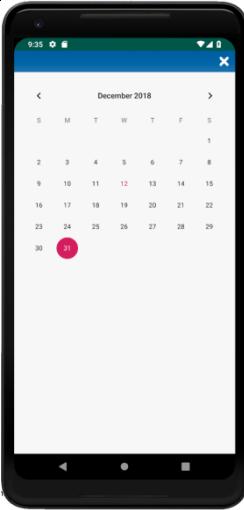
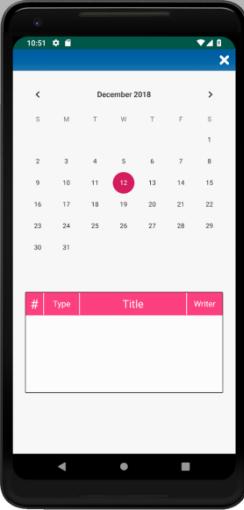
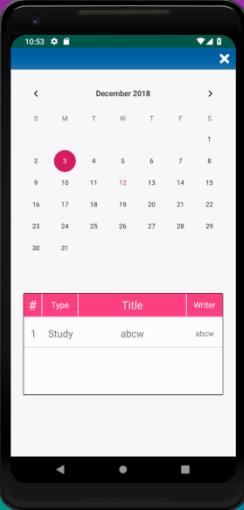
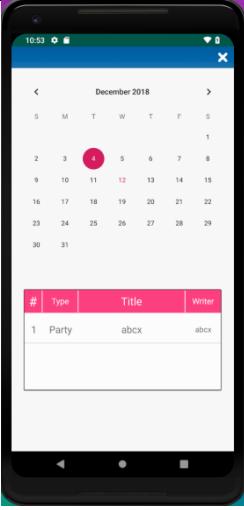
## 시나리오 구성도 1

1) Splash 화면	2-a) 로그인 화면	2-b) 로그인 화면	3) 홈 화면
			
시작할 때 나오는 화면이다. 아무것도 누를 필요가 없으면 0.75초 후에는 로그인 화면으로 넘어간다.	아이디+비밀번호는 아무거나 입력을 하면 '로그인 성공' 문구가 뜬다. Sign Up 위에 나오는 스위치 버튼은 그냥 보여주기 식으로 넣었다 웹사이트 아이디 저장을 표현하였다.	로그인 성공 문구와 함께 ok문구가 뜬다. Ok 버튼을 누르면 몇 초 뒤 홈 화면으로 이동한다.	홈 화면이며, 화면들마다 아래 파란색 집 아이콘을 누르면 이 화면으로 오게 된다.
4-a) 로그아웃 화면	4-b) Logout에서 Ok누른 화면	5-a) Other(건의사항) 화면	5-b) 건의사항 제출성공 화면
			
로그아웃을 하는 화면이다. Ok를 누르면 Login 화면으로 간다.	이렇게 다시 Login 화면으로 되돌아 간다.	건의 사항 화면이다. Content에 내용을 적으면 성공이라는 텍스트가 출력되고, 내용을 하나도 적지 않으면 다시 입력하라는 텍스트가 출력된다.	'5-a 화면'에서 Content에 내용을 적으면 출력되는 화면.

## 시나리오 구성도 2

5-c) 건의사항 제출실패 화면	6) Community 화면	6-a) Study 리스트 화면 (전)	6-b) Study 일정 적는 화면 1
			
'5-a 화면'에서 Content에 내용을 적으면 출력되는 화면.	Community 화면이다. 글자와 사진 중 아무거나 누르면 정보를 제공해 주는 해당 화면으로 이동한다.	Community 화면에서 Study 버튼을 누르면 넘어오는 화면이다. 오른쪽 위에 연필 아이콘을 누르면 일정을 추가 할 수 있다.	일정을 적어서 추가하는 화면이다. 여기서 Study-Party 라디오 버튼을 어느 곳에 설정하느냐에 따라 이 추가된 일정이 Study 리스트로 갈지 Party 리스트로 갈지 결정된다.
6-c) Study 일정 적는 화면 2	6-d) Study 리스트 화면 (후)	6-e) Study 일정 확인 화면	7-a) Party 리스트 화면 (전)
			
날짜를 설정을 안하고, 내용도 입력되지 않으면 '모든 정보를 입력하라'라는 문구가 뜬다.	Submit된 일정을 확인하는 화면이다.	전 화면에서 리스트를 누르면 이런 식으로 모든 내용이 다 나온다.	Community 화면에서 Party 버튼을 누르면 넘어오는 화면이다. 오른쪽 위에 연필 아이콘을 누르면 일정을 추가 할 수 있다.

### 시나리오 구성도 3

7-b) Party 일정 적는 화면 1	7-c) Party 일정 적는 화면 2	7-d) Party 리스트 화면 (후)	7-e) Party 일정 확인 화면
			
일정을 적어서 추가하는 화면이다. 여기서 Study-Party 라디오 버튼을 어느 곳에 설정하느냐에 따라 이 추가된 일정이 Study 리스트로 갈지 Party 리스트로 갈지 결정된다.	날짜를 설정을 안하고, 내용도 입력되지 않으면 '모든 정보를 입력하라'라는 문구가 뜬다.	Submit된 일정을 확인하는 화면이다.	전 화면에서 리스트를 누르면 이런 식으로 모든 내용이 다 나온다.
8-a) 캘린더 화면	8-b) 캘린더 화면	8-c) 캘린더 화면	8-d) 캘린더 화면
			
캘린더 화면이다. Study와 Party 일정의 데이터를 여기로 가지고 온다.	날짜를 누르면 리스트가 뜬다. 만약 Study나 Party에 리스트를 작성하지 않으면 이렇게 빈칸이 출력된다.	Study 리스트를 작성했을 때 나오는 캘린더 화면에서 출력되는 리스트 문구이다.	Party 리스트를 작성했을 때 나오는 캘린더 화면에서 출력되는 리스트 문구이다.

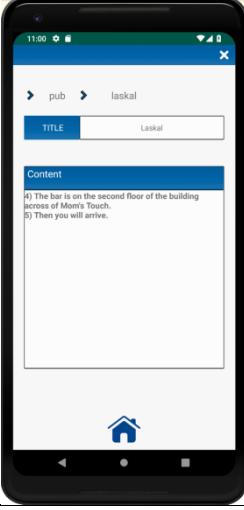
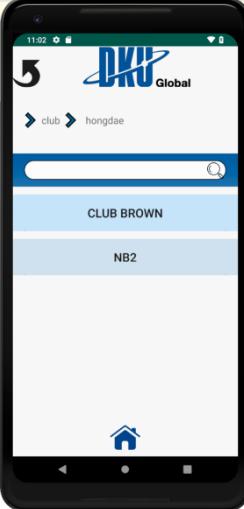
## 시나리오 구성도 4

9-a) Mentor 리스트 화면 (전)	9-b) Mentor 일정 적는 화면 1	9-c) Mentor 일정 적는 화면 2	9-d) Mentor 리스트 화면 (후)
Community 화면에서 Mentor 버튼을 누르면 넘어오는 화면이다. 오른쪽 위에 연필 아이콘을 누르면 일정을 추가 할 수 있다.	일정을 적어서 추가하는 화면이다. 여기서 Mentor-Mentee 라디오버튼을 어느 곳에 설정하느냐에 따라 이 추가된 일정이 Mentor 리스트로 갈지 Mentee 리스트로 갈지 결정된다.	아무 내용도 입력되지 않으면 '모든 정보를 입력하라'라는 문구가 뜬다. Study-Party랑 다르게 캘린더랑 연동이 되지 않는다.	Submit된 일정을 확인하는 화면이다.
9-e) Mentor 일정 확인 화면	10-a) Mentee 리스트 화면 (전)	10-b) Mentee 일정 적는 화면 1	10-c) Mentor 일정 적는 화면 2
전 화면에서 리스트를 누르면 이런 식으로 모든 내용이 다 나온다.	Community 화면에서 Mentee 버튼을 누르면 넘어오는 화면이다. 오른쪽 위에 연필 아이콘을 누르면 일정을 추가 할 수 있다.	일정을 적어서 추가하는 화면이다. 여기서 Mentor-Mentee 라디오버튼을 어느 곳에 설정하느냐에 따라 이 추가된 일정이 Mentor 리스트로 갈지 Mentee 리스트로 갈지 결정된다.	아무 내용도 입력되지 않으면 '모든 정보를 입력하라'라는 문구가 뜬다. Study-Party랑 다르게 캘린더랑 연동이 되지 않는다.

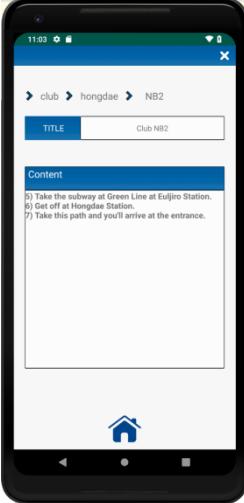
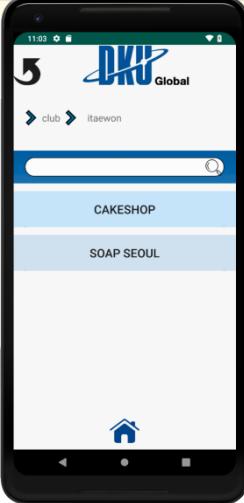
## 시나리오 구성도 5

10-d) Mentor 리스트 화면 (후)	10-e) Mentee 일정 확인 화면	11) 다시 흠 화면	11) Community 화면
Submit된 일정을 확인하는 화면이다.	전 화면에서 리스트를 누르면 이런 식으로 모든 내용이 다 나온다.	Place로 갈 때 흠 버튼이 있는 화면에서 흠 버튼을 흠 화면으로 돌아간다. 그리고 밑에 nightlife, shopping, place to visit를 누르면 place화면으로 이동된다.	또는 Community 화면에서 오른쪽 위 Place를 누르면 Place 화면으로 이동된다. 마찬가지로 Place 화면에서 Community를 클릭하면 Community 화면으로 이동된다.
11) Place 화면	12) Pub 화면	12-a) Pub 정보 화면1	12-a) Pub 정보 화면2
Place 화면이다. 글자와 사진 중 아무거나 누르면 정보를 제공해 주는 해당 화면으로 이동한다.	Place에서 Pub 아이콘을 누르면 넘어오는 화면이다. 여기에 3개의 버튼을 누르면 각 술집 정보 페이지로 이동하게 된다.	'단국신화'라는 술집 정보 페이지이면, Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'단국신화'가는 법을 자세하게 설명하고 있다.

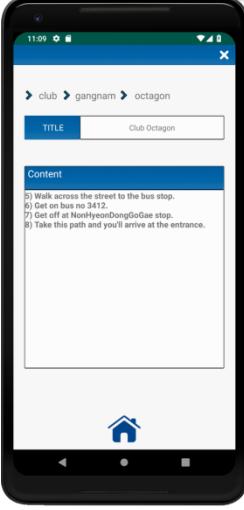
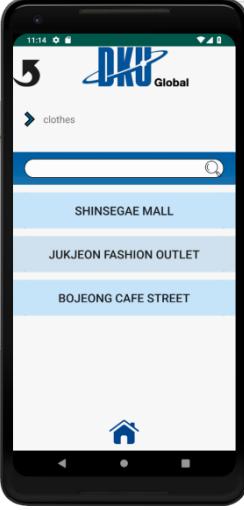
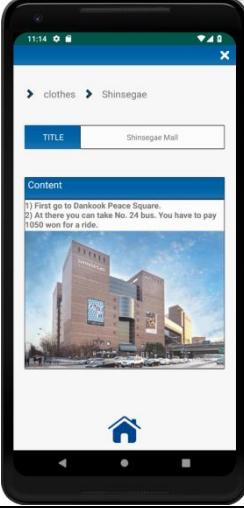
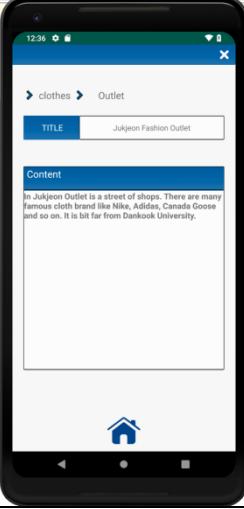
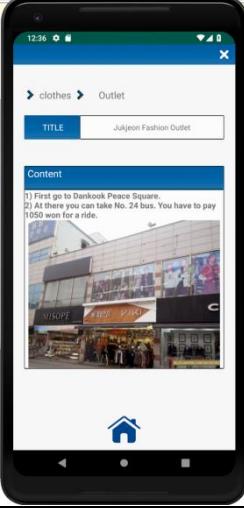
## 시나리오 구성도 6

12-b) Pub 정보 화면1	12-b) Pub 정보 화면2	12-c) Pub 정보 화면1	12-c) Pub 정보 화면2
 <p>'라스칼'이라는 술집 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'라스칼' 가는 법을 자세하게 설명하고 있다.</p>	 <p>'하이트 하우스'라는 술집 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'하이트 하우스' 가는 법을 자세하게 설명하고 있다.</p>
13) Club 화면	13-a) Club 화면	13-aa) Club 정보 화면1	13-aa) Club 정보 화면2
 <p>Place에서 Club 아이콘을 누르면 넘어오는 화면이다. 클럽 페이지만 한번 더 분류되서 클럽들이 소개가 된다.</p>	 <p>홍대에 있는 유명 클럽들의 정보를 얻기 위해 거쳐가는 화면이다. 버튼을 누르면 각 클럽의 정보를 얻는다.</p>	 <p>'브라운'이라는 클럽 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'브라운' 가는 법을 자세하게 설명하고 있다.</p>

## 시나리오 구성도 7

13-ab) Club 정보 화면1	13-ab) Club 정보 화면2	13-b) Club 화면	13-ba) Club 정보 화면1
			
'NB2'라는 클럽 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'NB2'가는 법을 자세하게 설명하고 있다.	이태원에 있는 유명 클럽들의 정보를 얻기 위해 거쳐가는 화면이다. 버튼을 누르면 각 클럽의 정보를 얻는다.	'Cake Shop'이라는 클럽 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.
13-ba) Club 정보 화면2	13-bb) Club 정보 화면1	13-bb) Club 정보 화면2	13-c) Club 화면
			
'Cake Shop'가는 법을 자세하게 설명하고 있다.	'Soup Seoul'이라는 클럽 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'Soup Seoul'가는 법을 자세하게 설명하고 있다.	버튼을 클릭을 하면 강남에 있는 옥타곤 클럽의 정보가 나오는 페이지로 이동한다.

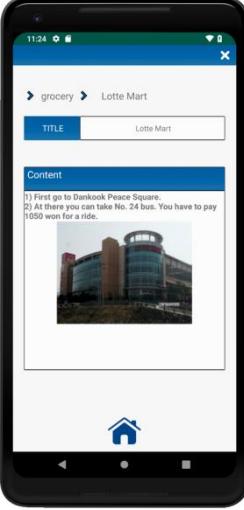
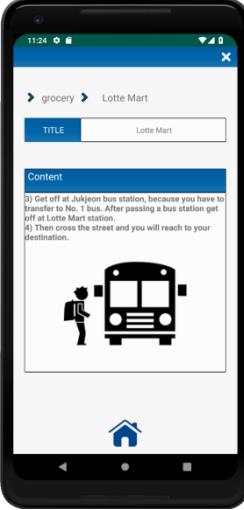
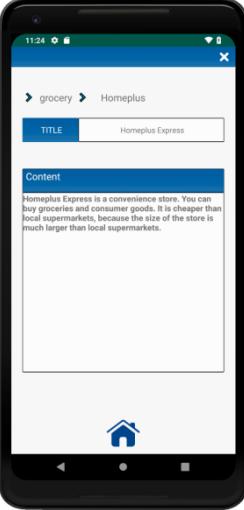
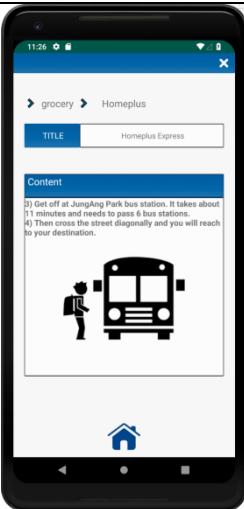
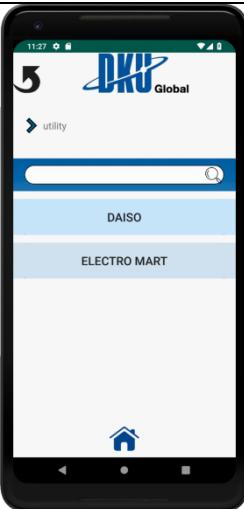
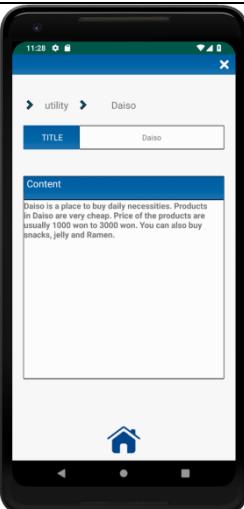
## 시나리오 구성도 8

13-c) Club 정보 화면1	13-c) Club 정보 화면2	14) Clothes 화면	14-a) Clothes 정보 화면1
			
<p>'옥타곤'이라는 클럽 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	<p>'옥타곤' 가는 법을 자세하게 설명하고 있다.</p>	<p>Place에서 Shopping 아이콘을 누르면 넘어오는 화면이다. 여기에 3개의 버튼을 누르면 각 옷을 파는 장소의 정보 페이지로 이동하게 된다.</p>	<p>'신세계 백화점' 정보 페이지이다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>
14-a) Clothes 정보 화면2	14-a) Clothes 정보 화면3	14-b) Clothes 정보 화면1	14-b) Clothes 정보 화면2
			
<p>'신세계 백화점' 가는 법을 자세하게 설명하고 있다.</p>	<p>'신세계 백화점' 가는 법을 자세하게 설명하고 있다.</p>	<p>'죽전 아울렛' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	<p>'죽전 아울렛' 가는 법을 자세하게 설명하고 있다.</p>

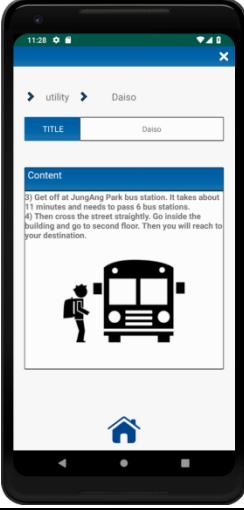
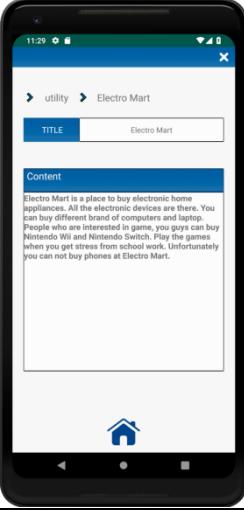
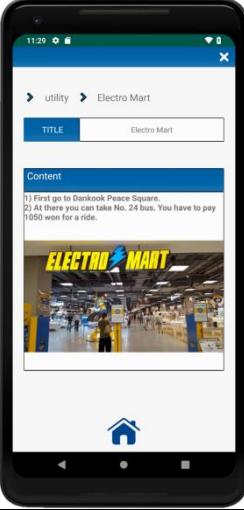
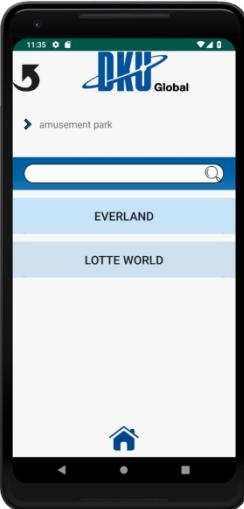
## 시나리오 구성도 9

14-b) Clothes 정보 화면3	14-c) Clothes 정보 화면1	14-c) Clothes 정보 화면2	14-c) Clothes 정보 화면3
'죽전 아울렛' 가는 법을 자세하게 설명하고 있다.	'보정동 카페거리' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'보정동 카페거리' 가는 법을 자세하게 설명하고 있다.	'보정동 카페거리' 가는 법을 자세하게 설명하고 있다.
15) Grocery 화면	15-a) Grocery 정보 화면1	15-a) Grocery 정보 화면2	15-a) Grocery 정보 화면3
Place에서 Grocery 아이콘을 누르면 넘어오는 화면이다. 여기에 3개의 버튼을 누르면 각 대형마트의 정보 페이지로 이동하게 된다.	'이마트 죽전점' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'이마트 죽전점' 가는 법을 자세하게 설명하고 있다.	'이마트 죽전점' 가는 법을 자세하게 설명하고 있다.

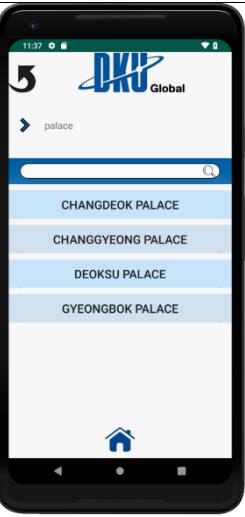
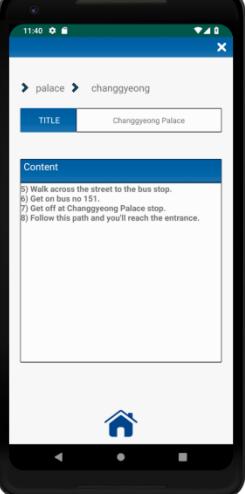
## 시나리오 구성도 10

15-b) Grocery 정보 화면1	15-b) Grocery 정보 화면2	15-b) Grocery 정보 화면3	15-c) Grocery 정보 화면1
 <p>'롯데마트 죽전점' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'롯데마트 죽전점' 가는 법을 자세하게 설명하고 있다.</p>	 <p>'롯데마트 죽전점' 가는 법을 자세하게 설명하고 있다. 볼 수 있다.</p>	 <p>'홈플러스 익스프레스 보정점' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>
15-c) Grocery 정보 화면2	15-c) Grocery 정보 화면3	16) Utility 화면	16-a) Utility 정보 화면1
 <p>'홈플러스 익스프레스 보정점' 가는 법을 자세하게 설명하고 있다.</p>	 <p>'홈플러스 익스프레스 보정점' 가는 법을 자세하게 설명하고 있다.</p>	 <p>Place에서 Utility 아이콘을 누르면 넘어오는 화면이다. 여기에 2개의 버튼을 누르면 각 유ти리티를 파는 장소의 정보 페이지로 이동하게 된다.</p>	 <p>'다이소 보정점' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>

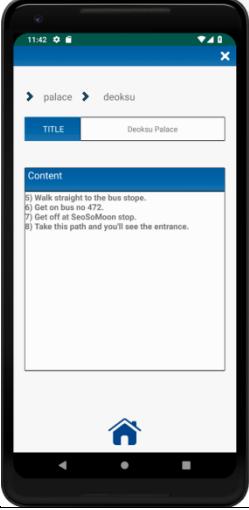
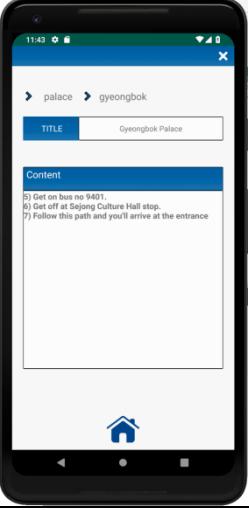
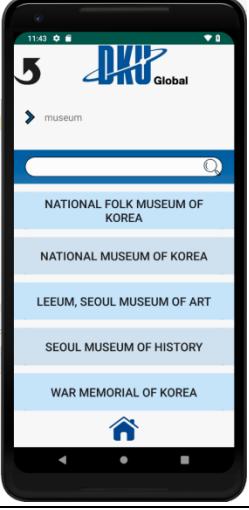
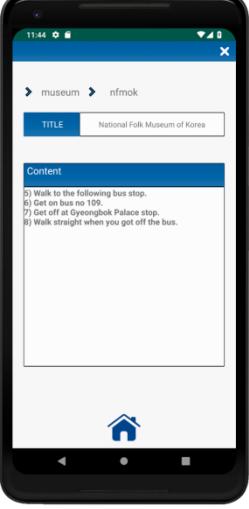
## 시나리오 구성도 11

16-a) Utility 정보 화면2	16-a) Utility 정보 화면3	16-b) Utility 정보 화면1	16-b) Utility 정보 화면2
 <p>'다이소 보정점' 가는 법을 자세하게 설명하고 있다.</p>	 <p>'다이소 보정점' 가는 법을 자세하게 설명하고 있다.</p>	 <p>'일렉트로 마트' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'일렉트로 마트' 가는 법을 자세하게 설명하고 있다.</p>
16-b) Utility 정보 화면3	17) Amusement Park 화면	17-a) 놀이동산 정보 화면1	17-a) 놀이동산 정보 화면2
 <p>'일렉트로 마트' 가는 법을 자세하게 설명하고 있다.</p>	 <p>Place에서 Amusement Park 아이콘을 누르면 넘어오는 화면이다. 여기에 2개의 버튼을 누르면 각 놀이동산의 정보 페이지로 이동하게 된다.</p>	 <p>'에버랜드' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.</p>	 <p>'에버랜드' 가는 법을 자세하게 설명하고 있다.</p>

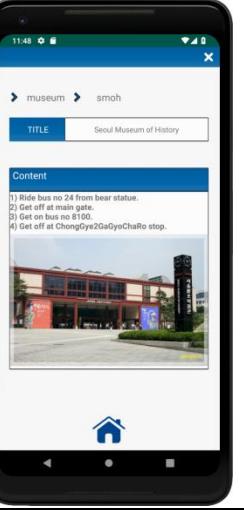
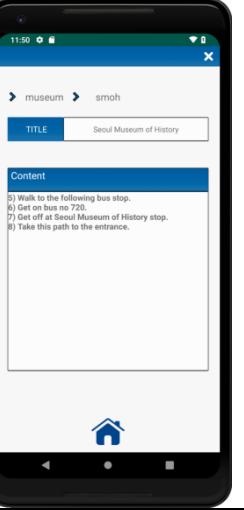
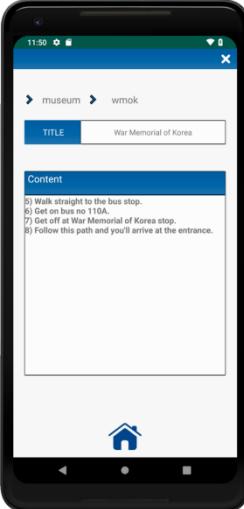
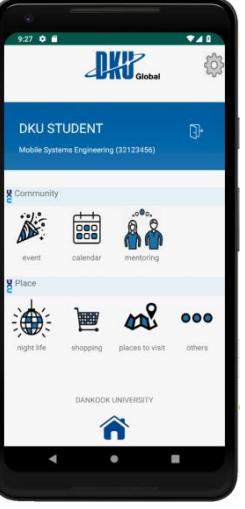
## 시나리오 구성도 12

17-b) 놀이동산 정보 화면1	17-b) 놀이동산 정보 화면2	18) Palace 화면	18-a) Palace 정보 화면1
			
'롯데월드' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'롯데월드' 가는 법을 자세하게 설명하고 있다.	Place에서 Palace 아이콘을 누르면 넘어오는 화면이다. 여기에 4 개의 버튼을 누르면 대한민국 왕들이 살았던 궁의 정보 페이지로 이동하게 된다.	'창덕궁' 가는 법을 자세하게 설명하고 있다.
18-a) Palace 정보 화면2	18-b) Palace 정보 화면1	18-b) Palace 정보 화면2	18-c) Palace 정보 화면1
			
'창덕궁' 가는 법을 자세하게 설명하고 있다.	'창경궁' 정보 페이지다. Content 라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'창경궁' 가는 법을 자세하게 설명하고 있다.	'덕수궁' 정보 페이지다. Content 라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.

## 시나리오 구성도 13

18-c) Palace 정보 화면2	18-d) Palace 정보 화면1	18-d) Palace 정보 화면2	19) Museum 화면
			
'덕수궁' 가는 법을 자세하게 설명하고 있다.	'경복궁' 정보 페이지다. Content 라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'경복궁' 가는 법을 자세하게 설명하고 있다.	Place에서 Museum 아이콘을 누르면 넘어오는 화면이다. 여기에 5개의 버튼을 누르면 대한민국 유명 박물관들의 정보 페이지로 이동하게 된다.
19-a) Museum 정보 화면1	19-a) Museum 정보 화면2	19-b) Museum 정보 화면1	19-b) Museum 정보 화면2
			
'국립 민속 박물관' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'국립 민속 박물관' 가는 법을 자세하게 설명하고 있다.	'국립 박물관' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'국립 민속 박물관' 가는 법을 자세하게 설명하고 있다.

## 시나리오 구성도 14

19-c) Museum 정보 화면1	19-c) Museum 정보 화면2	19-d) Museum 정보 화면1	19-d) Museum 정보 화면2
			
'삼성미술관 리움 박물관' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'삼성미술관 리움 박물관' 가는 법을 자세하게 설명하고 있다.	'서울 역사 박물관' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'서울 역사 박물관' 가는 법을 자세하게 설명하고 있다.
19-e) Museum 정보 화면1	19-e) Museum 정보 화면2	20) 홈 화면(끝)	
			
'전쟁 기념관' 정보 페이지다. Content라 사각형 공간에서 정보를 좌우로 넘기면서 볼 수 있다.	'전쟁 기념관' 가는 법을 자세하게 설명하고 있다.	홈 버튼을 눌러 다시 홈 화면으로 온다. 모든 페이지를 다 둘러 봤다.	

\*특이 사항: 시나리오 5의 6번째 화면부터 시나리오 14 6번째 화면까지는 버튼 눌러서 이동하고 그 페이지에서 Content 칸에서 화면을 좌우로 넘기면서 가는 법을 아는 화면들입니다. 화면들의 기능들이 다 똑같습니다.

## 5. 필수 적용 사항

### 【작성요령】 / 제출시 작성요령은 삭제

☞ 1) 전체 클래스 디어그램을 기반으로 서술형으로 작성

### 1) design

#### ① color

- 앱 기본 색상을 푸른색으로 맞췄다. 기본 파란색이 아니라 xml 파일로 색상을 따로 만들어서 background로 지정해 주었다. (drawable/topbottom.xml) 3가지 색상으로, 위에서 아래로 점점 연한 색으로 그레이션 되며 shape = rectangle이다.

#### ② figure

- 역시 xml 파일로 만들어서 background로 지정했고, 타원은 corner를 조정해서 했다.(ex.<corners android:radius="100dp" />) (drawable/rectangle.xml, drawable/circle\_border.xml 등)

#### ③ image

- 대부분의 이미지는 google에서 다운 받은 이미지를 사용하였고 포토샵 등으로 색상은 우리 앱에 맞게 바꾸어 주었다.

### 2) button color

- places 화면들에서 버튼을 통해 특정 장소로 넘어갈 때 버튼 클릭 시 색깔이 바뀜

```
<ripple  
    android:color="#8dd1ff"  
//버튼 선택시 색상
```

```
<item>  
<shape>  
<solid android:color="#c8e7fc" />  
</shape>  
</item>  
//선택 전 색상
```

### 3) splash

- 처음에 떴다가 자동으로 사라지는 화면

(in values/styles.xml)

```
<style name="SplashTheme" parent="Base.Theme.AppCompat.Light">
</style>
```

(in manifests.xml)

```
<activity
    android:name=".splash"
    android:theme="@style/SplashTheme">
</activity>
```

(in splash.kt)

```
val background = object : Thread{
```

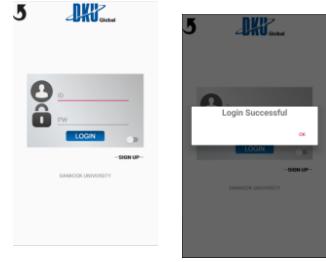
// 750초동안 멈출 thread를 만든다

```
try{
    Thread.sleep(750)
    val intent = Intent(baseContext, login::class.java)
    startActivity(intent)
}
```

//750초간 멈추고 그 후 login화면으로 넘어감

```
background.start()
```

//만들었던 thread를 실행시킨다.



#### 4) dialog

- login 화면에서 login 버튼을 눌렀을 때ダイアログ창 띄우기

(in login.kt)

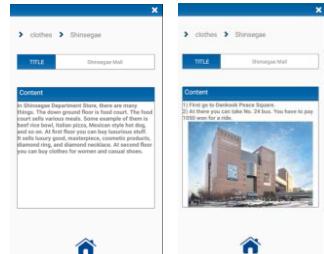
```
val builder = AlertDialog.Builder(this)
// this 액티비티에서 만들어 질 dialog 선언
```

```
val dialogView = layoutInflater.inflate(R.layout.custom_dialog, null)
```

// xml파일을 view로 바꾸어주는 inflater를 사용해서 custom\_dialog.xml 화면 View로 바꾸어서 dialogView라는 변수가 받음

```
builder.setView(dialogView)
.setPositiveButton("Ok") { dialogInterface, i ->
var nextIntent = Intent(this, home::class.java)
startActivity(nextIntent)
}
.show()
```

//선언해둔 dialog 속성 설정. dialogView를 받아서 show해주고 OK를 누르면 다음 화면으로 넘어간다.



#### 5) viewpager

- places의 다양한 장소를 설명하는 화면에서는 content를 슬라이드로 넘기며 내용을 볼 수 있다.

(in shinsegae.kt)

```
class MyViewPagerAdapter(manager: FragmentManager) :
FragmentPagerAdapter(manager){
private val fragmentList : MutableList<Fragment> = ArrayList()
override fun getItem(p0: Int): Fragment {
return fragmentList[p0]
}
override fun getCount(): Int {
return fragmentList.size
}
fun addFragment(fragment: Fragment){
fragmentList.add(fragment)
}}
```

//ViewPager 어댑터 클래스를 먼저 만들어준다. private 변수로 fragment 타입 배열을 만든다. override를 해서 부모가 만들라고 시킨 함수를 필수로 만들어주고 addFragment 함수를 만들어서 fragment를 받아서 추가 하는 함수를 만들었다. addFragment 함수를 쓰면 다음 화면이 하나씩 이어서 생긴다.

```

val adapter = MyViewPagerAdapter(supportFragmentManager)
adapter.addFragment(fragshinsegae1())
adapter.addFragment(fragshinsegae2())
adapter.addFragment(fragshinsegae3())
viewPager.adapter = adapter

```

adapter에 만들어 둔 클래스 생성자를 넘기고, 만들어둔 함수(addFragment)를 사용하여 fragment들을 하나씩 추가한다. addFragment의 매개변수는 fragment타입 클래스이다.

(in fragshinsegae1.kt)

```

class fragshinsegae1 : Fragment(){
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
return inflater.inflate(R.layout.frag_shinsegae1,container,false)
}

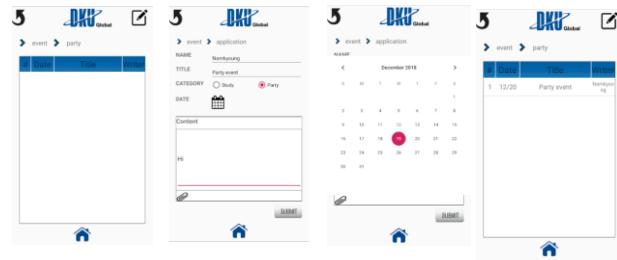
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
super.onViewCreated(view, savedInstanceState)
}
}

```

//add하기위한 fragment들의 클래스는 fragment타입이고 어떠한 layout을 inflate해서 return 한다. 여기서는 frag\_shinsegae1을 View로 바꾸어서 리턴하고 그것을 위 shinsaegae.kt에서 add하는 것이다.

## 6) event (database, listview)

- 외국 학생들이 study 또는 party 이벤트를 만들어서 서로 소통 할 수 있게 만드는 기능. 날짜를 정해서 원하는 이벤트를 작성하면 study, party 페이지에서 확인이 가능하다.



(in DBHelper.kt)

```

class DBHelper_Party(context : Context) : SQLiteOpenHelper(context, "Party.db", null, 1){
override fun onCreate(db: SQLiteDatabase?) {
var sql = "create table PartyTable (" +
"idx integer primary key autoincrement, " +
"dateData text not null, " +
"titleData text not null, " +
"nameData text not null, " +
"contentData text not null " +
")"
db?.execSQL(sql)
}

```

//database를 create한다. Helper가 하는 역할은 앱을 다시 열 때마다 다시 만들어지는 것을 방지한다. 처음 한번만 실행 될 수 있게 해주므로 앱을 삭제하지 않는 한 정보는 남아있다. autoincrement를 해서 자동적으로 순번을 매긴다.

### (in event\_applicaton.kt)

- name, title, category, date, content를 입력해서 submit버튼을 누르면 데이터베이스에 저장되도록 해준다. category는 어떤 헬퍼를 이용해서 어떤 테이블에 저장할지 정해준다.

```
rdGroup.setOnCheckedChangeListener { rdGroup, checkedId ->
    when (checkedId) {
        R.id.rdbtn_study -> selected = "StudyTable"
        R.id.rdbtn_party -> selected = "PartyTable"
    }
}
```

//버튼에 선택된 것에 따라서 selected 변수에 사용할 테이블 이름을 넣어준다

```
var Animshow = AnimationUtils.loadAnimation(this,R.anim.abc_slide_in_bottom)
var Animgone = AnimationUtils.loadAnimation(this, R.anim.abc_slide_out_bottom)
```

//Animshow는 아래에서 들어오게하고, Animgone은 아래로 나가게한다

```
btn_event_date_calendar.setOnClickListener {
    Animshow.setAnimationListener(object : Animation.AnimationListener{
        override fun onAnimationStart(animation: Animation?) {}
        override fun onAnimationEnd(animation: Animation?) {}
        override fun onAnimationRepeat(animation: Animation?) {}
    })
    Animgone.setAnimationListener(object : Animation.AnimationListener{
        override fun onAnimationStart(animation: Animation?) {}
        override fun onAnimationEnd(animation: Animation?) {}
        override fun onAnimationRepeat(animation: Animation?) {
    }})
```

```
layout_calendar.startAnimation(Animshow)
layout_calendar.setVisibility(View.VISIBLE)
```

```
application_calendar.setOnDateChangeListener(CalendarView.OnDateChangeListener
{ view, year, month, dayOfMonth ->
    layout_calendar.startAnimation(Animgone)
    layout_calendar.setVisibility(View.GONE)

    app_date = (month+ 1).toString() + "/" + dayOfMonth.toString()
})
```

//date를 받을 때는 캘린더가 띄워지고 선택한 날짜가 데이터베이스에 들어간다. 이 때 캘린더는 애니메이션(Animshow)으로 띄워지고, app\_date라는 변수에 선택한 날짜가 들어간다. 캘린더에서 선택된 date가 변하면 (date를 선택 하면) 캘린더는 애니메이션(Animgone)으로 사라진다.

```
btn_event_submit.setOnClickListener {
```

```

var helper1 = DBHelper_Study(this)
var db1 = helper1.writableDatabase

var helper2 = DBHelper_Party(this)
var db2 = helper2.writableDatabase

var sql = "insert into "+ selected + " (dateData, titleData, nameData, contentData)" +
" values (?,?,?,?,?)"

var arg1 = arrayOf(app_date, edt txt_title.text, edt txt_name.text, edt txt_event_content.text)

if(selected == "StudyTable") {
if(app_date == "0" || edt txt_title.getText().toString().equals("") || 
edt txt_name.getText().toString().equals("") || 
edt txt_event_content.getText().toString().equals("")) {
Toast.makeText(applicationContext, "insert all information", Toast.LENGTH_LONG).show()
db1.close()
db2.close()
}
else {
db1.execSQL(sql, arg1)
db1.close()
db2.close()
var intent = Intent(this, event_study::class.java)
startActivity(intent)
}
}

else if(selected == "PartyTable"){
if(app_date == "0" || edt txt_title.getText().toString().equals("") || 
edt txt_name.getText().toString().equals("") || 
edt txt_event_content.getText().toString().equals("")) {
Toast.makeText(applicationContext, "insert all information", Toast.LENGTH_LONG).show()
db1.close()
db2.close()
}
else {
db2.execSQL(sql, arg1)
db2.close()
db1.close()
var intent = Intent(this, event_party::class.java)
startActivity(intent)
}
}

```

//submit 버튼을 눌렀을 때 데이터베이스에 입력된 값들이 추가된다. insert문에서 사용 될 테이블은 라디오버튼을 통해 받아온 selected 변수이다. 이 때 4가지 항목 중 하나라도 입력되지 않은 값이 있으면 toast로 "insert all information"을 띄운다.

(in ListVO.kt)

```
class ListVO {  
    var index: String? = null  
    var numb: String? = null  
    var date: String? = null  
    var title: String? = null  
    var name: String? = null  
}
```

리스트뷰에서 한 줄에 표시 될 항목들. index는 실제로 출력은 하지않고 나중에 리스트뷰에서 클릭했을 때 해당하는 항목의 content를 보기 위해 index를 기억하도록 했다. ([calendar에서](#) 뒤섞인 데이터의 실제 위치를 찾기위해 사용)

(in ListViewAdapter.kt)

```
class ListViewAdapter : BaseAdapter() {  
    //ListViewAdapter 클래스  
    private val listVO = ArrayList<ListVO>()  
    //ListVO타입 배열. 한 줄에 ListVO의 항목들이 들어가고 그게 차례대로 이어져 있는 배열.
```

(참고)ListView에서는 화면에 새로운 아이템을 표시하라고 할 때마다 Adapter의 getView()를 호출하게 된다. 여기서 getView 메소드는 각 View를 보여줄 때마다 호출되기 때문에 n개의 View를 보여줄 때 무조건 n번의 호출이 이루어지게 된다. ListView는 자원을 재사용 할 때는 null이 아닌 값이 들어오게 되며, null인 경우에는 레이아웃을 inflate한다. ConvertView가 null이 아닌 경우에는 기존의 View를 재사용하기 때문에 새롭게 View를 inflate 할 필요 없이 데이터만 바꾸는 작업을 진행하면 된다.

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {  
    var convertView = convertView  
    //position = ListView의 위치      /   첫번째면 position = 0  
    val context = parent.context  
  
    if (convertView == null) {  
        val inflater = context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as  
        LayoutInflator  
        convertView = inflater.inflate(R.layout.custom_listview, parent, false)  
    }  
  
    val index = convertView.findViewById<TextView>(R.id.index)  
    val numb = convertView.findViewById<TextView>(R.id.numb)  
    val date = convertView.findViewById<TextView>(R.id.date)  
    val title = convertView.findViewById<TextView>(R.id.title)  
    val name = convertView.findViewById<TextView>(R.id.name)  
  
    val listViewItem = listVO[position]  
    // 아이템내각위젯에 데이터반영
```

```
index.text = listViewItem.index  
numb.text = listViewItem.numb  
date.text = listViewItem.date  
title.text = listViewItem.title  
name.text = listViewItem.name  
  
return convertView  
}
```

//convertView가 null일 때 custom\_listview.xml을 inflate해서 convertView에 넣는다.  
custom\_listview에서 id값을 설정해준 항목 index, numb, date, title, name을 findViewById를 해준다. listViewItem에는 private변수로 선언된 배열 listVO의 position번째 값이 들어간다. ListVO class는 5가지 항목들을 가지고 있었고 ListVO타입인 listVO역시 numb, date, title, name 항목들을 가지고 있다. custom\_listview.xml에서 찾아서 findViewById 해준 텍스트들에 listVO의 항목들을 차례로 넣어주고 convertView를 리턴한다. 즉, getView 함수는 어떤 아이템을 새로 표시하라는 명령이 들어오면, 그에 맞는 numb, date, title, name을 찾아서 text값에 넣어주는 역할이다.

```
fun addVO(index : String, numb : String, date: String, title: String, name: String) {  
    val item = ListVO()  
  
    item.index = index  
    item.numb = numb  
    item.date = date  
    item.name = name  
    item.title = title  
  
    listVO.add(item)  
}
```

//addVO함수는 실제로 ListVO에 표시될 항목들을 추가하는 역할이다. 매개변수로 값을 받아서 listVO배열에 추가해준다.(여기서 추가 된 아이템이 실제로 화면에 표시되는 것, 화면에 뿌려주는 역할이라고 보면된다.)

```
fun getindex(position : Int) : String? {  
    return listVO[position].index  
}
```

//calendar에서 원래 위치를 찾기위해 listview로부터 index를 찾기위해 사용

```
fun gettype(position: Int) : String? {  
    return listVO[position].date  
}
```

//마찬가지로 calendar에서 어떤 테이블인지 찾기위해서 사용. date라고 써져있는 이유는 calendar에서는 date라는 항목에 Study 또는 Party가 들어가기 때문. 캘린더에서는 같은 List 타입을 재활용 하기 위해서 항목이름은 그대로 두고 내용은 다른 것이 들어가게 했다.

```
fun clearVO(){  
    listVO.clear()  
}
```

//listVO배열을 전부 비워주는 역할. calendar에서 한 화면에서 계속 listview의 내용이 바뀌게 되는 일이 생기기 때문에 수시로 비워주고 채워주기 위해 사용.

(in event\_party.kt)

```
var helper = DBHelper_Party(this)
var db = helper.writableDatabase
var sql = "select * from PartyTable"
var c: Cursor = db.rawQuery(sql, null)
adapter = ListViewAdapter()
listview = findViewById<ListView>(R.id.List_view)
listview.adapter = adapter
```

//PartyTable에서 모든 항목들을 select하고 listview의 adapter에 ListViewAdapter를 할당한다.

```
while(c.moveToNext()) {
    var idx_pos = c.getColumnIndex("idx")
    var dateData_pos = c.getColumnIndex("dateData")
    var titleData_pos = c.getColumnIndex("titleData")
    var nameData_pos = c.getColumnIndex("nameData")
    var contentdata_pos = c.getColumnIndex("contentData")

    var numb = c.getString(idx_pos)
    var date = c.getString(dateData_pos)
    var Title = c.getString(titleData_pos)
    var Name = c.getString(nameData_pos)
    var Content = c.getString(contentdata_pos)

    adapter!!.addVO(numb, date, Title, Name)
}
db.close()
```

//데이터 베이스를 커서를 통해 한 줄씩 차례로 읽으며 listview에 추가를 해준다. 즉, 여기서 데이터베이스에서 받은 정보를 리스트뷰를 통해 화면에 뿌려준다.

```
listview.setOnItemClickListener { parent, view, position, id ->
    var helper2 = DBHelper_Party(this)
    var db2 = helper2.writableDatabase

    var sql2 = "select * from PartyTable"
    var c2: Cursor = db2.rawQuery(sql2, null)

    c2.moveToPosition(position)

    var idx_pos = c2.getColumnIndex("idx")

    var numb = c2.getString(idx_pos)

    var intent = Intent(this, event_content::class.java)
    intent.putExtra("value", numb)
```

```
intent.putExtra("type", "PartyTable")
startActivity(intent)
```

```
db2.close()
}
```

//리스트뷰에서 아이템을 클릭했을 때, 어떤 테이블인지, 그리고 데이터 베이스에서 몇 번 째 항목인지를 표시하는 인덱스를 다음 화면에 넘겨주며 event\_content화면으로 넘어간다.

(in event\_content.kt)

```
var type = intent.getStringExtra("type")
var sql = "select * from " + type
```

//이전 화면에서 넘겨받은 table아이템을 type 변수에 저장하고 sql문으로 셀렉트한다.

```
if(type == "StudyTable") {
txt_study_or_party.text = "study"
var c: Cursor = db1.rawQuery(sql, null)
while(c.moveToNext()) {
var idx_pos = c.getColumnIndex("idx")
var titleData_pos = c.getColumnIndex("titleData")
var nameData_pos = c.getColumnIndex("nameData")
var contentdata_pos = c.getColumnIndex("contentData")
```

```
var numb = c.getString(idx_pos)
var Title = c.getString(titleData_pos)
var Name = c.getString(nameData_pos)
var Content = c.getString(contentdata_pos)
```

```
if(numb == intent.getStringExtra("value")) {
txt_event_content_title.text = Title
txt_event_content_name.text = Name
txt_event_content.text = Content
break
}
}
db1.close()
db2.close()
}
```

//type이 StudyTable일 때 이전에 넘겨온 index와 같은 index일 때 text에 데이터베이스에 들어있는 항목들을 맞게 넣어준다. PartyTable일 때도 동일함.

## 7) calendar (calendar, database, listview)



(in calendar.kt)

```
var slidingPanel = findViewById<LinearLayout>(R.id.slidingPanel)
```

//slidingPanel은 해당 date에 있는 study event와 party event들을 보여주기 위한 레이아웃이며 findViewById를 해서 필요에 따라 애니메이션으로 나타났다가 사라질 것.

```
var Animshow = AnimationUtils.loadAnimation(this,R.anim.abc_slide_in_bottom)
```

//밑에서부터 들어오는 애니메이션

```
Animshow.setAnimationListener(object : Animation.AnimationListener{
    override fun onAnimationStart(animation: Animation?) {}
    override fun onAnimationEnd(animation: Animation?) {
        if(isPageOpen){
            if(changedate == false) {
                slidingPanel.setVisibility(View.INVISIBLE)
                isPageOpen = false
            }
            else {
                changedate = false
            }
        }else{
            isPageOpen = true
        }
    }
    override fun onAnimationRepeat(animation: Animation?) {}
})
```

//Animshow가 불려졌을 때 할 작업들. 먼저 페이지가 이미 열려있는지 확인하고 페이지가 열려있다면 date가 바뀌었는지 본다. 선택한 date가 바뀌었으면 그대로 있고(바뀐 date값에 해당하는 event를 보여줌) 똑같은 date를 또 클릭 한 것이라면 slidingPanel을 보이지 않게 한다.

```
Animgone.setAnimationListener(object : Animation.AnimationListener{
    override fun onAnimationStart(animation: Animation?) {}
    override fun onAnimationEnd(animation: Animation?) {
        if(isPageOpen){
```

```

slidingPanel.setVisibility(View.INVISIBLE)
isPageOpen = false
}else{
isPageOpen = true
}
}
override fun onAnimationRepeat(animation: Animation?) {}
//Animgone이 불려 졌을 때 할 작업들. slidingPanel을 invisible로 바꿔준다.(안보이게 된다.)

```

```

calendar.setOnDateChangeListener(CalendarView.OnDateChangeListener { view, year,
month, dayOfMonth ->

if(pre_date == dayOfMonth){
if(isPageOpen == true){
slidingPanel.startAnimation(Animgone)
slidingPanel.setVisibility(View.GONE)
}
else{
slidingPanel.startAnimation(Animshow)
slidingPanel.setVisibility(View.VISIBLE)
}
}else{
changedate = true
slidingPanel.startAnimation(Animshow)
slidingPanel.setVisibility(View.VISIBLE)
}
pre_date = dayOfMonth
}
//캘린더에서 date가 클릭되면 이전 date와 확인후 같을 시 이미 page가 open되어 있다면
Animgone 애니메이션을 부르고 slidingPanel을 없앤다. page가 닫혀있다면 Animshow를 부르고 slidingPanel을 visible하게 바꿔준다. date가 다르다면 changedate 값을 true로 바꾸고
Animshow를 부르고 slidingPanel을 visible하게 바꿔준다.

```

```

var calendar_index = 1
var helper = DBHelper_Party(this)
var db = helper.writableDatabase
var sql = "select * from PartyTable"
var c1: Cursor = db.rawQuery(sql, null)

adapter!!.clearVO()
adapter!!.notifyDataSetChanged()

while(c1.moveToNext()) {
var idx_pos = c1.getColumnIndex("idx")
var dateData_pos = c1.getColumnIndex("dateData")
var titleData_pos = c1.getColumnIndex("titleData")

```

```

var nameData_pos = c1.getColumnIndex("nameData")

var index = c1.getString(idx_pos)
var numb = calendar_index.toString()
var date = c1.getString(dateData_pos)
var type = "Party"
var Title = c1.getString(titleData_pos)
var Name = c1.getString(nameData_pos)

if(date == (month+ 1).toString() + "/" + dayOfMonth.toString()){
    calendar_index+
    adapter!!.addVO(index, numb, type, Title, Name)
}
}

db.close()

```

//(여전히 dateChangeListener 내부에서) 데이터베이스에서 받아온 index와 calendar에서 사용될 index는 다르기 때문에 1부터 지정해서 listview에서 뿐려주는 순서대로 1씩 증가시킨다. slidingPanel안에 있는 리스트뷰에 그 날짜에 해당하는 PartyTable의 event를 리스트뷰에 뿐린다. 이때 커서로 PartyTable의 아이템들을 차례로 확인하면서 date값과 선택된 date값이 같은지 확인한다. (if(date==(~~))에서 확인) 이 때 주의해야 할 점은, 한 화면에서 clickListener에 들어갈 때마다 listview가 갱신되기 때문에 처음에 listview안에 있는 값을 모두 없애 주어야 하고 그 후 이 사실을 adapter에게 꼭 알려야 한다는 것이다.

```

        adapter!!.clearVO()
        adapter!!.notifyDataSetChanged()

```

이 부분이다. 이를 하지 않으면 값이 꼬이거나 출력이 제대로 되지 않는다.

그 후, StudyTable도 똑같이 처리해서 Party와 study event를 모두를 띄워준다.

```

cal_listview!!.setOnItemClickListener { parent, view, position, id ->
    var helper = DBHelper_Study(this)
    var db1 = helper.writableDatabase

    var helper2 = DBHelper_Party(this)
    var db2 = helper2.writableDatabase

    var index = adapter!!.getindex(position)
    var type = adapter!!.gettype(position)

    if(type == "Study") {
        var sql1 = "select * from StudyTable"
        var c: Cursor = db1.rawQuery(sql1, null)

        c.moveToPosition(position)

        var numb = index

        var intent = Intent(this, event_content::class.java)

```

```
intent.putExtra("value", numb)
intent.putExtra("type", "StudyTable")
startActivity(intent)

db1.close()
db2.close()
}
```

//listview를 클릭했을 때 인덱스 값과 table이름을 넘겨주며 event\_content 화면으로 넘어간다. 이 때, adapter를 정의할 때 미리 만들어 놓은 getIndex함수를 사용한다. 왜냐하면 calendar에서 사용된 index는 데이터베이스의 순서와는 다르기 때문이다. 그래서 데이터베이스로부터 index 값을 받아 오는 것이 아니라 listview에서 실제로 출력은 하지 않고 단지 데이터 베이스 내에서의 index값을 기억하는 역할만 했던 변수를 받아와서 위치를 찾는 것이다. type도 마찬가지로 listview에 기억 된 Table이름을 받아와서 어떤 테이블을 선택할지 정한다. 즉, 캘린더에서는 여러 데이터베이스 테이블이 섞여서, 순서가 뒤 섞인 채로 출력 된다. 그래서 원래 데이터베이스의 위치를 찾기위해서 간단히 position을 보는 것이 아닌, 처음에 데이터베이스에서 받아와서 listview에 입력했던 일부 정보를 통해서 역으로 찾는다.