

2018 산학협력프로젝트2 일반반 (2037) 최종 설계 문서

: (2조) 영화 상영관 예매 시뮬레이션 프로그램 [ToPM]



2 조

201611186 김나경

201611227 임현정

201514517 하자뢰

201611245 홍혜진

제출일

2018년 12월 4일

[목차]

1. 프로젝트 팀 구성

2. 동작 흐름

- 2.1 초기화면
- 2.2 회원가입
- 2.3 관리자 view
- 2.4 사용자 view

3. 클래스 설계

4. Flow Chart

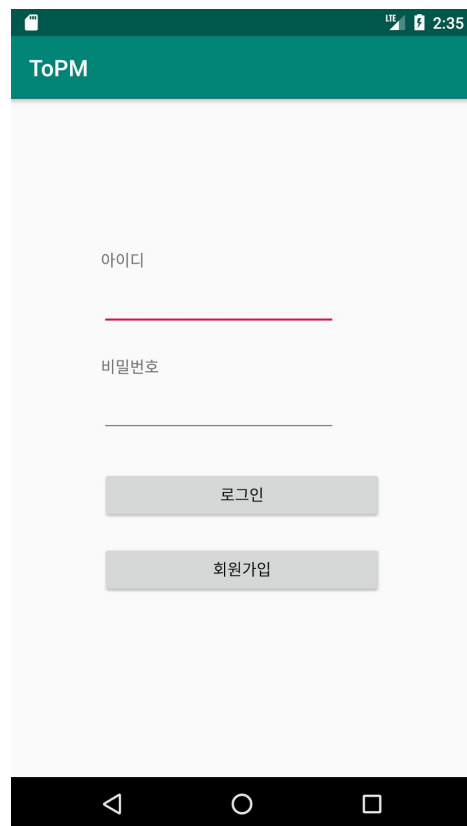
1. 프로젝트 팀 구성

2조	
김나경	201611186
임현정	201611227
하자뢰	201514517
홍혜진	201611245

2. 동작 흐름

2.1. 초기 화면

- 로그인
 - LoginActivity.java
 - activity_login.xml
 - EditText Widget 2개 - login_id, login_pw
 - Button 2개 - loginBtn, joinBtn



- **loginBtn 클릭리스너 - void loginClick(View view)**

사용자에게서 입력받은 ID와 PW를 Firebase DB에 저장된 회원 정보와 대조 한다.

분기 1) 입력한 ID가 FirebaseDB에 존재하는지 판단한다.

```
for(DataSnapshot data : dataSnapshot.getChildren()){
    //아이디가 같은지 판단
    if(data.getKey().equals(login_id.getText().toString())) {
        //아이디가 같다면 유저 인스턴스를 하나 생성해서 데이터를 받아온 다음
        user = data.getValue(User.class);
        //아이디가 user 데이터베이스에 존재함을 표시한다.
        idExist = true;
        break;
    }
}
```

- 입력한 ID가 FirebaseDB에 존재한다면 boolean flag(변수명 idExist, 초기값 false)를 true 값으로 설정하고, DB로부터 User정보를 가져와 인스턴스(변수명 user)에 저장한다.

분기 2) idExist의 값을 확인한다.

- idExist가 true라면 user의 getPw()함수로 pw를 가져와 입력한 PW와 일치하는지 비교한다.
- idExist가 false라면 해당 ID가 존재하지 않음을 표시하고, ID,PW입력란을 둘다 초기화 한다.
- idExist가 true이지만, pw가 일치하지 않았다면,비밀번호가 틀렸음을 알리고, PW 입력 란만 초기화된다.

분기 3) 분기 2에서 true이고, pw까지 일치한다면 로그인에 성공한 회원이 관리자인지, 일반 사용자인지 구별한다. (user의 isAdmin()함수)

```
if (user.isAdmin()) //관리자 액티비티로
    intent = new Intent(getApplicationContext(), AdminMainActivity.class);
//유저 인스턴스가 일반 사용자라면
else //일반 사용자 액티비티로
    intent = new Intent(getApplicationContext(), UserMainActivity.class);
```

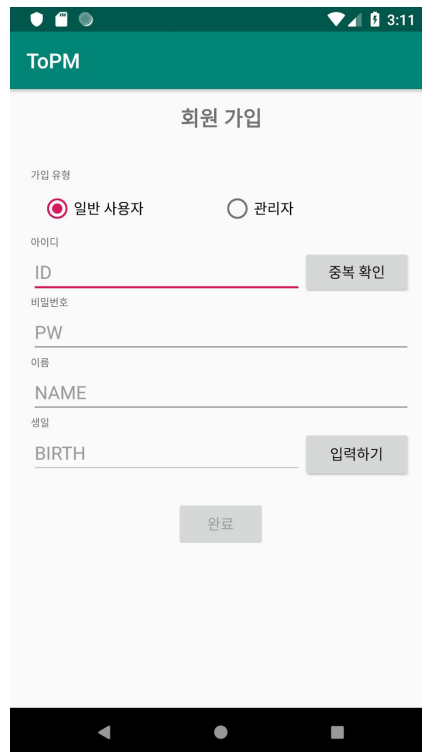
관리자 회원일 시 관리자 액티비티로, 일반 사용자일시 일반 사용자 액티비티로 전환한다.

- **joinBtn의 클릭 이벤트 - void joinClick(View view)**

회원가입을 할 수 있는 JoinActivity로 전환한다.

2.2. 회원가입

- 회원가입
 - JoinActivity.java
 - activity_join.xml
 - RadioGroup 1개 - type_join, (RadioButton 2개 - normalRB, adminRB)
 - EditText Widget 3개 - id_join, pw_join, name_join
 - Button Widget 3개 - idCheckBtn, joinBtn, birthBtn



- **void setIdInput(boolean isIdChecked)**

ID의 중복체크버튼(idCheckBtn)로 입력한 ID가 유효한 ID(중복되지 않는 ID)인지 아닌지 판단해 isIdChecked로 setIdInput함수에 넘겨준다. 해당 함수는 ID가 체크 통과되기 전에는 완료버튼을 누를 수 없고, 체크 통과된다면 완료버튼을 누를 수 있고, 입력했던 아이디를 재입력버튼을 눌러야만 수정할 수 있도록 하기 위함이다.

 - isIdChecked가 true이면 중복확인 Btn을 “재입력하기”로 setText한다. 회원가입 완료Btn을 활성화하고, ID 입력창를 비활성화한다.
 - isIdChecked가 false이면 중복확인Btn을 “중복 확인”으로 setText한다. 회원가입 완료Btn을 비활성화하고, ID 입력창를 활성화한다.

- idCheckBtn의 클릭 이벤트 - void idCheckClick(View view)

분기 1) ID 입력이 공백인지, 공백으로 시작하는지 검사한다.

```
//아이디에 공백은 입력 불가, 혹은 스페이스가 처음으로 들어가게 입력불가
if(input_id.equals("")||input_id.charAt(0)==' '){
    Toast.makeText(this,"공백으로 시작하는 아이디는 입력
    불가",Toast.LENGTH_SHORT).show();
}
```

- 공백이거나 공백으로 시작하는게 아니라면 다음 분기로 넘어간다.

분기 2) ID가 이미 Firebase DB 서버에 존재하는지 검사한다.

```
for(DataSnapshot data : dataSnapshot.getChildren()){ //현재
    //파이어베이스에 저장된 데이터의 스냅샷을 차례로 가져와 data에 저장한다.
    if(data.getKey().equals(input_id)) { //data의 키 =
        //user_id로 설정 돼 있음. 즉, data의 키가 input_id와 같은지 검사한다.
        isValidID = false; //같은 정보가 있다면
        //아이디 중복이므로 isValidID = false
        break; //for문 탈출
    }
```

- 입력한 ID가 DB에 이미 있다면 boolean flag(변수명 isValidID, 초기값 true)를 false 값으로 설정하고, 사용할 수 없는 아이디임을 AlertMessage로 알린다.
- DB에 없다면 isValidID를 초기 값(true)으로 유지한다.

분기 3) isValidID의 값을 확인한다.

- isValidID가 true라면 사용할 수 있는 아이디임을 AlerMessage로 알린다. UI를 고치기 위해 setIdInput(true)를 호출한다.
- isValidID가 false라면 사용할 수 없는 아이디임을 AlerMessage로 알린다. UI를 고치기 위해 setIdInput(false)를 호출한다.

- birthBtn 클릭 이벤트 - void birthClick(View view)



- 생일입력은 이 버튼을 눌러야 입력이 가능하다.
- 직접 입력이 아니고, 사용자가 선택한 날짜를 해당 포맷으로 바꿔서 프로그램 상에서 직접 문자열을 만든다.

//데이트픽커 다이얼로그

```
DatePickerDialog datePickerDialog = new DatePickerDialog(JoinActivity.this, new
DatePickerDialog.OnDateSetListener() {
```

```
@Override
```

```
public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
```

```
//입력 받은 데이터를 Date 객체에 저장
```

```
Date birthData = new Date(year,month,dayOfMonth);
```

```
//파이어베이스에 저장하는 포맷은 yyMMdd
```

```
SimpleDateFormat birthFormat = new SimpleDateFormat("yyMMdd");
```

```
//포맷대로 생일데이터를 저장
```

```
birthOutput = birthFormat.format(birthData);
```

```
//month는 출력 상으로 -1 돼서 나오므로 +1 처리
```

```
month = month+1;
```

```
//생일 에디트텍스트에 입력 받은 값 표시해줌.
```

```
birthEditText.setText(year+"년 "+ month+"월 "+dayOfMonth+"일");
```

```
}
```

```
//Dialog가 띄워졌을 때, 현재 시간으로 기본 설정
```

```
},calendar.get(Calendar.YEAR),calendar.get(Calendar.MONTH)+1,calendar.get(Ca
lendar.DATE));
```

```
//현재 날짜 이후에는 선택할 수 없게 맥스를 오늘로 설정
```

```
datePickerDialog.getDatePicker().setMaxDate(new Date().getTime());
```

생일

BIRTH

입력하기

`new DatePickerDialog(this, mDateSetListner, mYear, mMonth, mDay);`
로 `DatePickerDialog`를 생성한다.

`DatePicker`로 생일 날짜를 입력받고, `yyMMdd`포맷으로 `string`에 저장해 생일 입력 칸에 `setText`한다. 생일 텍스트뷰는 사용자가 직접 입력할 수 없게 `setEnabled = false`로 되어있다.

- **joinCompleteBtn 클릭 이벤트 - void joinCompleteClick(View view)**

분기1) 입력 창들 중 하나라도 공백인 창이 있는지 확인한다.

(아이디는 무조건 중복검사를 해야 회원가입 완료버튼을 누를 수 있으므로 공백이 아님이 보장되므로 검사하지 않는다.)

```
if(input_pw.length() <= 0 || input_name.length() <= 0 || birthOutput.length() <= 0)
```

- 공백이 있다면 모든 입력해야한다는 토스트메세지를 띄운다.
- 공백이 없다면 `User` 클래스의 생성자에 입력받은 값들을 파라미터로 전달해 인스턴스를 하나 생성한다. `FirebaseDB` user 노드에 생성한 `User` 인스턴스를 추가한다. `finish()`로 `LoginActivity`로 돌아간다.

2.3. 관리자 View

- 관리자의 관리자 모드로 회원가입을 할 수 있다.
- 관리자의 Main View
 - AdminMainActivity.java
 - activity_admin_main.xml
 - ListView 1개, 편집 Button Widget 3개
- 날짜별 영화 상영 현황 - dayScheduleList
- 영화 등록 View - movieEditBtn로 MovieManageActivity 연결
- 상영관 등록/수정 View - screenEditBtn로 ScreenManageActivity 연결
- 상영 등록 View - scheduleEditBtn 로 ScheduleManageActivity 연결



관리자 메인 화면에서는 현재 날짜와 이후 날짜를 포함해 총 4일간의 영화 상영 스케줄을 열람할 수 있다. (오늘이 1일이면 1,2,3,4일)

2.3.1. 영화 등록/제거 액티비티

- MovieManageActivity.java
- activity_movie_manage.xml
- ListView 1개 - 현재 등록 영화 목록 - movieList
- List의 row개수마다 Button widget - deleteBtn
- Button Widget - completeBtn
- EditText Widget 3개 - movie_title, movie_director, movie_time
- RadioGroup 1개, RadioButton 3개 - radioGroup, RBall, RB12, RB15, RB19

KT 74% 오후 2:28

ToPM

현재 등록된 영화

신비한 동물들과 그린델왈드의 범죄 삭제

신비한 동물사전 삭제

아가씨 삭제

쥬라기 월드 삭제

상영 등급: ☒ 전체 ☐ 12세 ☐ 15세 ☐ 청소년

제목: title

감독: director

러닝타임: running time (분 단위)

완료

제목, 감독, 러닝타임 정보를 모두 입력하고 완료 버튼을 누르면 데이터베이스의 movie 노드 아래 객체의 정보가 저장된다.

- **public void init() 함수 속 영화등급 RadioGroupListener**

```
rateGroup.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        switch (checkedId){
            // 전체 관람가
            case R.id.RBall:{
                rate = -1;
                break;
            }
            // 12세 이상
            case R.id.RB12:{
                rate = 12;
                break;
            }
            // 15세 이상
            case R.id.RB15: {
```

```

        rate = 15;
        break;
    }
    // 19세 이상
    case R.id.RB19:{
        rate = 19;
        break;
    }
}
});
...후략..

```

전체 관람가일 경우 rate값은 -1, 그 외 나이제한은 12세 미만이면 12, 15세 미만이면 15, 19세 미만이면 19로 저장된다.

● completeBtn 클릭 이벤트

분기 1) 모든 입력 칸이 정상적으로 채워졌는지 검사한다.

- 모든 칸을 채우지 않았으면 입력을 확인하라는 메시지가 뜨고 작업을 완료할 수 없다.
- 모든 칸이 채워졌다면 Movie 클래스의 생성자로 인스턴스를 하나 생성한다.

Movie(int runningTime, String title, String director, int rating)
데이터베이스의 movie 노드 아래에 객체를 업로드하고,
movieList에 영화 정보가 추가된 후 AdminMainActivity로
전환된다.

이때 '모든 칸이 정상적으로 채워졌다'는 것은 이름, 감독,
러닝타임을 받는 EditText가 getText()를 했을 때, null을
반환하지 않을 때이고, 특히 러닝타임 같은 경우는 1이상의
양의 정수일때를 말한다. 나머지의 경우에는 (ex.. 음수, 실수)는
숫자 키보드만 뜨게 하여 아예 입력할 수 없도록 제재한다. 또한
라디오버튼은 기본 값으로 항상 전체관람가의 값을 가지고
있으므로 따로 검사하지 않는다.

@Override

● public void onMovieDeleteBtnClick(final int position)

영화리스트뷰의 row마다 존재하는 삭제 버튼의
클릭리스너이다.

영화를 삭제하려고 시도할 때, 해당 영화를 삭제할 수 있는지
검사한다.

firebaseDB 중 movie, bookingInfo노드 총 두 개를
고려해야한다.

분기 1) 삭제하려는 영화가 bookingInfo에 존재하는가?

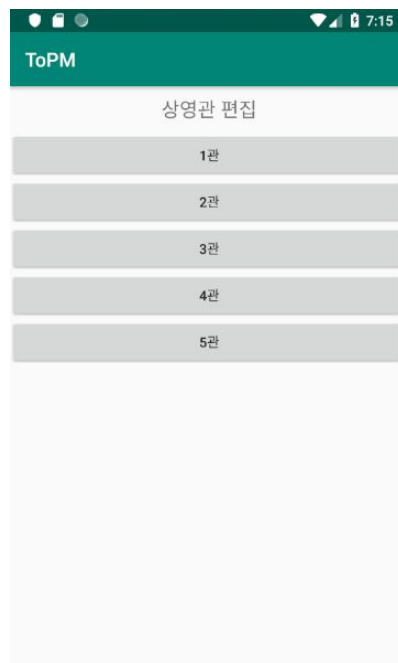
```
for(DataSnapshot data : dataSnapshot.getChildren()){
    BookingInfo bi = data.getValue(BookingInfo.class);
    // 예매정보 중에 삭제하려고 선택한 영화와 같은 제목의 영화가 있다면
    if(bi.getTitle().equals(movieTitle)){
        // 예매정보가 존재한다. true
        bookingExist = true;
        Toast.makeText(getApplicationContext(),"해당 영화는 고객의
예매내역이 존재하는 영화입니다.",Toast.LENGTH_SHORT).show();
        break;
    }
    // 같은 제목의 영화가 없다.
    else{
        bookingExist = false;
    }
}
```

- 존재한다면, 영화를 지울 수 없음을 알린다.
- 존재하지 않는다면, 영화를 DB에서 삭제한다.
-

2.3.2. 상영관 좌석 수정 액티비티

(1) ScreenListActivity.java

- activity_screen_list.xml
- Button Widget 5개
- 5개의 버튼 이벤트 -> 해당 상영관의 좌석 정보를 가진 ScreenShowActivity로 연결



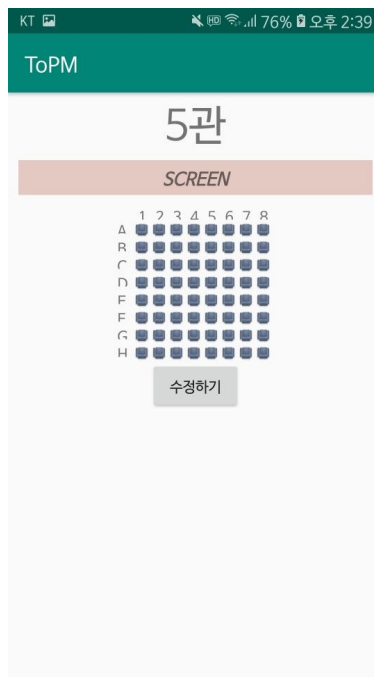
```
final int SCREEN_COUNT = 5;
```

상수를 정의해 상영관 개수를 지정한다. 상수에 따라 버튼을 동적으로 할당한다.

```
// 상영관의 개수가 달라져도 SCREEN_COUNT만 바꾸면 버튼이  
// 늘어나도록 버튼을 직접 자바코드로 작성  
for(int i=0; i<SCREEN_COUNT; i++) {  
    screenEdits[i] = new Button(this);  
    screenEdits[i].setId(i+1); // id 추가함  
    screenEdits[i].setText(String.valueOf(i+1)+"관");  
    screenEdits[i].setWidth(ViewGroup.LayoutParams.WRAP_CONTENT);  
  
    screenEdits[i].setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);  
    buttonLayout.addView(screenEdits[i]);  
  
    screenEdits[i].setOnClickListener(this);  
}
```

(2) ScreenShowActivity.java

- activity_screen_show.xml
- Button Widget 1개
- 해당 상영관의 현재 좌석 정보를 보여준다.



이전 액티비티에서 받아온 상영관 번호 정보를 토대로 데이터 베이스에 저장되어 있는 상영관의 현재 상태를 출력한다. 좌석 버튼 클릭은 불가능하다.

- **editBtn 클릭 이벤트**

수정하기 버튼을 클릭하면 상영관 좌석의 열과 행을 지정할 수 있는 ScreenEdit1Activity로 전환된다.

(3) ScreenEditActivity1.java

- activity1_screen_edit.xml
- EditText Widget 2개 - row, col
- Button Widget 1개 - nextBtn



/ 상수 */*

final int ROW_MAX = 20;

final int COL_MAX = 20;

final int ROW_MIN = 5;

final int COL_MIN = 5;

상수를 정의해 입력 최소, 최대 값을 제한한다.

- **nextBtn 클릭 이벤트**

분기 1) row값과 col값이 공백인지 검사한다.

- 공백이 아니면 분기 2를 검사한다.
- 공백이라면 다시 받는다.

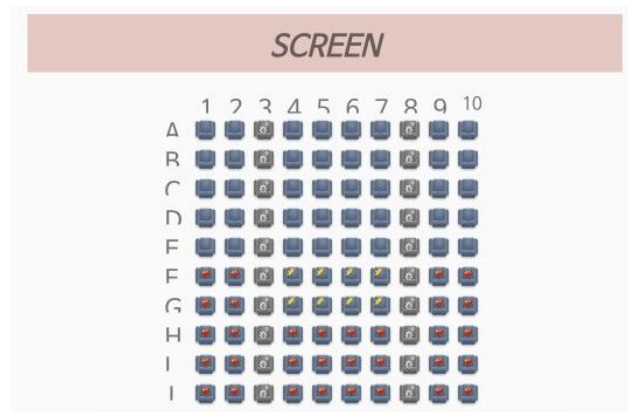
분기 2) row 값이 ROW_MAX이하, ROW_MIN이상

col 값이 COL_MAX이하, COL_MIN이상 인지 검사한다.

- 모든 검사가 통과하면 row, col 값을 ScreenEditActivity2 전송하고 화면을 전환한다.
- 조건을 만족하지 못할 시, InputException을 발생시키고, “입력을 확인하세요.”라는 Toast 메시지를 띄우며 현재 뷰에 머무른다.

(4) ScreenEditActivity2.java

- activity2_screen_edit.xml
- MyButton Widget - col*row개
- Button Widget - 4개 (수정 모드 선택 3개, 수정 완료 1개)



HashMap<String, Boolean> **abled**; // 좌석인지 아닌지 여부
저장

HashMap<String, Boolean> **special**; // 우등석인지 아닌지 여부
저장

HashMap<String, Boolean> **couple**; // 커플석인지 아닌지 여부
저장

좌석인지 아닌지, 우등석인지 아닌지, 커플석인지 아닌지 여부를 저장하는 해쉬맵 3개를 사용한다. Key 값으로는 각각 좌석의 Button ID값을, Value 값으로는 MyButton.UNABLED와 같은 True, False 값을 저장한다.

- 모드 변경 버튼 클릭 이벤트
각 버튼을 클릭하면, 해당 유형의 좌석을 지정할 수 있는 모드로 변경된다. 유형 정보는 상수에 정의해두고, 현재 모드 상태를 변수에 저장한다. 처음 모드는 **MODE_NORMAL** 로 초기화되어있다.

```
int mode;    // 좌석 수정 모드 (좌석/우등석/커플석)
final static int MODE_NORMAL = 11;    // 좌석/비좌석 설정
// 모드 상수
final static int MODE_SPECIAL = 22;    // 우등석 설정 모드
// 상수
final static int MODE_COUPLE = 33;    // 커플석 설정 모드
// 상수
```

좌석 버튼 클릭 이벤트에서 현재 설정된 모드 값에 따라 클릭시 액션을 다르게 정의한다.

```
@Override
public void onClick(View v) {
    // 좌석 버튼 클릭 이벤트
    switch(mode) {
        case MODE_NORMAL: // 좌석 수정 모드
            ... 종락
            break;

        case MODE_SPECIAL: // 우등석 수정 모드
            ... 종락
            break;

        case MODE_COUPLE: // 커플석 수정 모드
            ... 종락
            break;
    }
}
```

- 1) mode = **MODE_NORMAL**
현재 클릭한 자리의 HashMap **abled**의 Value가 MyButton.**ABLED**인 경우 Value를 MyButton.**UNABLED**로 변경한다.

현재 클릭한 자리의 HashMap **abled**의 Value가 MyButton.**UNABLED**인 경우 Value를 MyButton.**ABLED**로 변경한다.
- 2) mode = **MODE_SPECIAL**
현재 클릭한 자리의 HashMap **special**의 Value가 MyButton.**SPECIAL**인 경우 MyButton.**UNSPECIAL**로 변경한다.

현재 클릭한 자리의 HashMap **special**의 Value가
MyButton.**UNSPECIAL**인 경우
Value를 MyButton.**SPECIAL**로 변경한다.

3) mode = **MODE_COUPLE**

현재 클릭한 자리의 HashMap **couple**의 Value가
MyButton.**UNCUPLE**인 경우

현재 클릭한 좌석의 오른쪽 자리를 함께 커플석으로
등록한다. Value를 MyButton.**COUPLE**로 변경한다.
오른쪽 자리가 커플석 등록이 불가능 한 경우(좌석이
아니거나, 클릭한 좌석이 해당 열의 오른쪽 끝 좌석
등)에는 오류 메시지를 띄우며 커플석으로 지정할 수
없다.

커플석 지정이 완료되면, 두 좌석의 ID값을 더한 값을
Key로, Value를 MyButton.COUPLE로 설정해 해쉬맵에
put한다. 이후 어떤 좌석끼리 세트인지 알기 위함이다.

```
// ★ 어떤 좌석끼리 세트인지 알아내느냐.. ★  
// 1) 세트인 두 좌석의 ID를 더해서 해쉬맵의 Key로 사용합니다.  
예) 3021+3022 = 6043  
// 2) Key를 2로 나눈 몫과, 그 몫에 1을 더한 값으로 세트인 두  
좌석의 ID임을 알 수 있습니다.  
// 예) 6043 / 2 = 3021 이므로 3021과 3021+1=3022 가  
세트입니다.  
String coupleKey = String.valueOf(index + index + 1);  
couple.put(coupleKey, MyButton.COUPLE);
```

현재 클릭한 자리의 HashMap **couple**의 Value가
MyButton.COUPLE인 경우

버튼 ID값을 이용해 현재 클릭한 좌석이 왼쪽 좌석과
오른쪽 좌석 중 어느 것과 세트인지 알아낸다.

```
// 왼쪽 자리랑 세트인지 오른쪽 자리랑 세트인지 판별해야 한다.  
int leftSet = index + index - 1; // 클릭한 자리와 왼쪽 자리의 ID  
합  
int rightSet = index + index + 1; // 클릭한 자리와 오른쪽 자리의  
ID 합  
  
if(couple.get(String.valueOf(leftSet)) != null &&  
couple.get(String.valueOf(index - 1)) != null  
&& couple.get(String.valueOf(index -  
1)).equals(MyButton.COUPLE)) {  
    // 왼쪽 자리와 세트  
    couple.put(String.valueOf(index), MyButton.UNCUPLE);  
    couple.put(String.valueOf(index - 1), MyButton.UNCUPLE);  
    // 비커플로 상태 변경  
  
    .. 중략  
}  
  
else if(couple.get(String.valueOf(rightSet)) != null &&  
couple.get(String.valueOf(index + 1)) != null  
&& couple.get(String.valueOf(index +
```





```

1)).equals(MyButton.COUPLE)) {
    // 오른쪽 자리와 세트
    couple.put(String.valueOf(index), MyButton.UNCUPLE);
    couple.put(String.valueOf(index + 1), MyButton.UNCUPLE);
    // 비선택으로 상태 변경
    .. 중략
}

```

세트인 좌석을 알아내면, 해당 좌석과 클릭한 좌석의 Value를 MyButton.UNCUPLE로 변경해 커플석 지정을 취소하고, HashMap에 저장된 ID의 합도 삭제한다.

- 좌석 상태 출력

- 1) 좌석인 경우 
- 2) 좌석이 아닌 경우 
- 3) 우등석인 경우 
- 4) 커플석인 경우 

- row, col만큼 직사각형의 모양으로 MyButton들이 동적할당되고, 각 버튼은 n001부터 시작해 1씩 증가해 고유의 ID를 갖는다. (n은 현재 상영관의 번호이다. ex. 2관일 경우 2)
- HashMap **abled**는 모든 버튼의 ID를 Key 값으로 가져 각 버튼과 매핑되며, 모든 Value가 MyButton.ABLED로 초기화된다.
- HashMap **special**는 모든 버튼의 ID를 Key 값으로 가져 각 버튼과 매핑되며, 모든 Value가 MyButton.UNSPECIAL로 초기화된다.
- HashMap **couple**는 모든 버튼의 ID를 Key 값으로 가져 각 버튼과 매핑되며, 모든 Value가 MyButton.UNCUPLE로 초기화된다.

- nextBtn 클릭 이벤트

Screen 객체의 멤버 변수인 HashMap<String, Boolean> **abled**와 HashMap<String, Boolean> **special**에 현재 수정한 결과가 반영된 해쉬 맵으로 갱신하고, saveToDataBase() 함수에서 데이터베이스에 변경된 Screen 정보를 업로드한다. 결과가 저장된 후, AdminMainActivity.java로 전환한다.

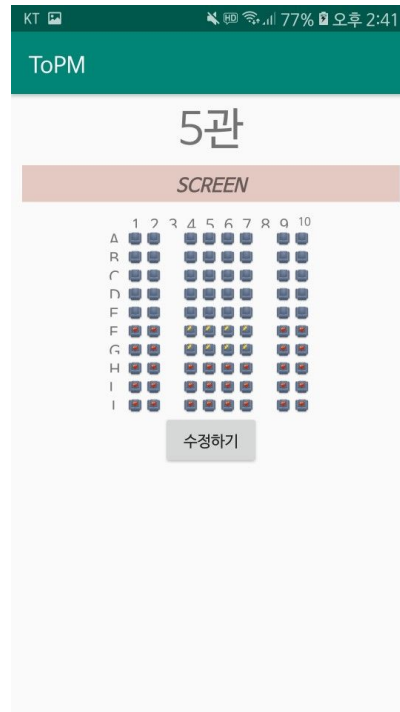
```

Screen newScreen = new Screen(row, col, screenNum/*, IDs*); // 객체 생성
newScreen.setAbleMap(abled); // 해쉬 맵 갱신
newScreen.setSpecialMap(special); // 해쉬 맵 갱신
newScreen.setCoupleMap(couple);

```

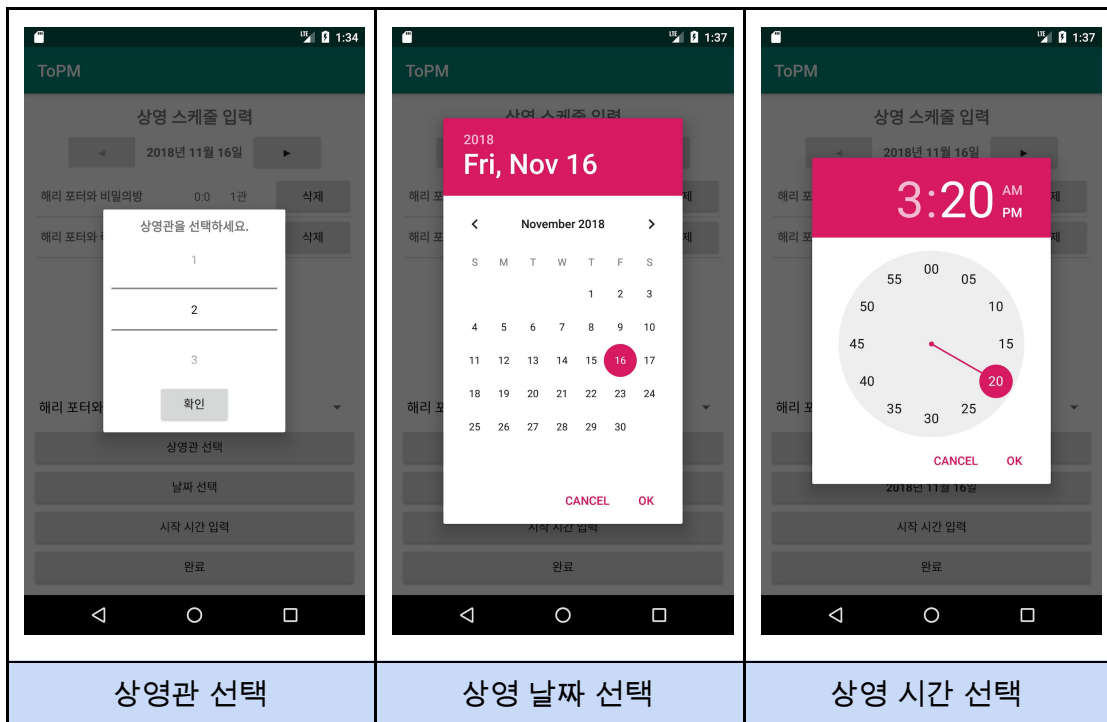
```
screenReference.child(screenKey).setValue(newScreen); // 저장
```

- 방금 수정한 내용을 다시 보려면 상영관편집>해당상영관 클릭을 하면 볼 수 있다. - 비활성화좌석은 없는 좌석이므로 여기서는 출력되지 않는다.



2.3.3. 상영 등록/제거 액티비티

- ScheduleManageActivity.java
- activity_schedule_manage.xml
- ListView 1개, ComboBox 1개, NumberPicker 1개, DatePicker 1개, Button 6개



- 1) 날짜 양 옆의 화살표 버튼을 누르면 현재 날짜를 포함한 4일간의 스케줄을 이동하며 열람할 수 있다.

- 2) Spinner를 클릭하면 현재 저장된 영화 중에서 상영 등록을 할 영화를 고를 수 있다.
- 3) Spinner 아래 버튼들을 누르면 각각 선택하는 정보에 맞는 Dialog가 뜨며, 해당 Dialog에서 정보를 입력할 수 있다.

```
// 입력 값에 따라 다른 다이얼로그 생성
@Override
protected Dialog onCreateDialog(int id) {
    switch(id) {
        // 날짜 다이얼로그
        case DATE_DIALOG:
            Calendar calendar = new
            GregorianCalendar(Locale.KOREA);
            ... 종료
            return dateDialog;

        // 시간 다이얼로그
        case TIME_DIALOG:
            TimePickerDialog timeDialog =
            new TimePickerDialog(this,
            new TimePickerDialog.OnTimeSetListener() {
                ..종료
            }
            return timeDialog;
        }
        return super.onCreateDialog(id);
    }
}
```

@Override

- **public void onScheduleDeleteBtnClick(int position, String strDateKey, int dateCount)**

상영회차(스케줄) 리스트뷰의 row마다 존재하는 삭제버튼의 클릭리스너이다.

상영회차를 삭제하려고 시도할 때, 해당 회차를 삭제할 수 있는지 검사한다.

firebaseDB 중 schedule, bookingInfo노드 총 두 개를 고려해야한다.

분기 1) 삭제하려는 영화가 bookingInfo에 존재하는가?
: bookingInfo에 존재한다는 뜻은 누군가 예매를 했다는 것이다.

- 존재한다면, 삭제할 수 없음을 알린다.
- 존재하지 않는다면, 스케줄 DB에서 삭제한다.

- **completeBtn 클릭 이벤트**

분기 1) 모두 입력했는지 검사한다.

모든 정보를 입력하지 않았다면, InputException을 발생시키고 “입력을 확인하세요.” 라는 Toast 메시지가 뜨고 상영 회차 추가 작업을 완료할 수 없다.

모든 정보를 입력했다면 MovieSchedule 클래스의 생성자로 인스턴스를 하나 생성한다.

MovieSchedule(String movieTitle, String screenNum, Date screeningDate)

해당 객체의 멤버 변수인 HashMap<String, Boolean>

bookedMap; 의 Key를 해당 상영관의 좌석 버튼 ID로, Value를 MyButton.**UNBOOKED**로 초기화한다.

데이터베이스의 schedule 노드 아래, 날짜 노드 아래에 “상영관 + 스케줄 Key”를 기본키로 하여 객체를 업로드한다. schedule 노드 아래, 날짜 노드에 리스너를 달아서 추가하려는 Key가 이미 존재하는지 검사한다. 존재한다면 키가 중복되므로 추가할 수 없다. 검사하는 변수로 isAddable을 둔다.

```
scheduleReference.child(strDate).addListenerForSingleValueEvent(new
 ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        for(DataSnapshot data : dataSnapshot.getChildren()){
            MovieSchedule ms=data.getValue(MovieSchedule.class);
            String temp = ms.getScreenNum()+ms.getScreeningDate();
            //
            Toast.makeText(getApplicationContext(),data.getKey(),Toast.LENGTH_S
            HORT).show();
            if(temp.equals(scheduleKey)){
                isAddable=false;
                break;
            }else{
                isAddable=true;
            }
        }
        if(isAddable){
            // 데이터베이스에 해당 순서대로 스케줄객체 삽입.
            scheduleReference.child(strDate).child(scheduleKey).setValue(movieSc
            hedule);
            finish();
        }else {
            Toast.makeText(getApplicationContext(),"해당 스케줄은 상영관의
            중복입니다.",Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

분기 2) isAddable의 값이 true인가 false인가

isAddable은 같은 날짜에 추가하려는 스케줄과 같은 상영관을 쓰는 스케줄 정보들을 가져와서 시작시간과 끝시간이 겹치지 않는지 검사하는 변수이다.

true인 경우 dayScheduleList에 상영회차 정보가 추가된 후 AdminMainActivity로 전환된다.

false인 경우 해당시간과 상영관이 중복인 것을 알리고 현재 뷰에 머무른다.

```
// DB의 스케줄 노드 아래, 추가하려는 날짜 노드 아래에서 검사함
scheduleReference.child(strDate).addListenerForSingleValueEvent(new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        // 최종적으로 해당 스케줄을 추가할 수 있는지를 판단하는 변수
        isAddable = true;

        // 같은 날짜에 같은 상영관에서 상영하는 영화스케줄을 모은 객체ArrayList
        ArrayList<MovieSchedule> sameScreenMovies = new ArrayList<>();

        // for문으로 해당 날짜의 모든 스케줄을 검사함.
        for(DataSnapshot data : dataSnapshot.getChildren()){
            MovieSchedule ms=data.getValue(MovieSchedule.class);
            // 영화의 상영관 번호가 추가하려는 스케줄의 상영관 번호와 같다면 객체ArrayList에
            추가
            if(ms.getScreenNum().equals(addScreenNum))
                sameScreenMovies.add(ms);
        }
        // sameScreenMovies를 차례로 검사하면서 지금 추가하려는 영화의 상영시간과
        비교하여 겹치는지 검사함
        // 만약 겹친다면 바로 for문을 탈출해 추가할 수 없다고 알림.
        for(int i=0;i<sameScreenMovies.size();i++){
            isAddable = true;
            MovieSchedule ms = sameScreenMovies.get(i);
            Date cmpDate = ms.getScreeningDate(); // sameScreenMovies의 영화
            시작시간, 끝시간을 계산함.
            int cmpStartHour = cmpDate.getHours(); // 시작 시각
            int cmpStartMin = cmpDate.getMinutes(); // 시작 분
            int cmpEndMin = ms.getEndMin(); // 종료 시각
            int cmpEndHour = ms.getEndHour(); // 종료 분

            // 상영시간이 겹치는지 겹치지 않는지 검사함.
            if(startHour<cmpEndHour ||
            ((startHour==cmpEndHour)&&(startMin<=cmpEndMin))){
                if(endHour>cmpStartHour ||
                ((endHour==cmpStartHour)&&(endMin>=cmpStartMin))) {
                    isAddable = false;
                    break;
                }
            }
        }

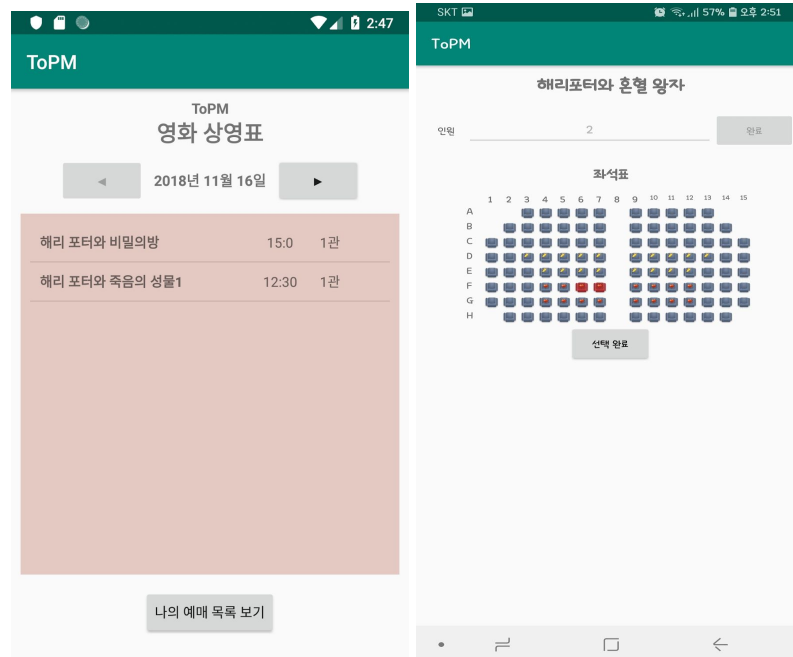
        if(isAddable){
            // 데이터베이스에 해당 순서대로 스케줄객체 삽입.
            scheduleReference.child(strDate).child(scheduleKey).setValue(movieSchedule);
            finish();
        }else {
            Toast.makeText(getApplicationContext(),"해당 스케줄은 상영관의
            중복입니다.",Toast.LENGTH_SHORT).show();
        }
    }
    .. 후략 ..
}
```

2.4. 사용자 View

2.4.1. 예매 액티비티

- 사용자가 영화를 예매할 수 있는 액티비티
 - UserMainActicity.java
 - activity_user_main.xml
 - BookMovieActivity.java
 - activity_book_movie.xml
 - ShowMovieInfoActivity.java
 - activity_show_movie.xml
- 멤버 변수
 - final int THIS_YEAR // 올해 연도

```
// 올해 저장 - 나이제한 대비  
final int THIS_YEAR =  
Calendar.getInstance().get(Calendar.YEAR);
```



UserMainActivity→ BookMovieActivity

- UserMainActivity의 스케줄리스트 클릭리스너

사용자의 나이 정보를 THIS_YEAR과 사용자 생년정보를 이용해서 구한다. 선택한 스케줄의 영화 상영등급을 가져와서 비교한 후 전체관람가이거나 사용자 나이가 상영등급보다 많다면 다음 BookMovieActivity로 넘어간다.

```
@Override  
public void onItemClick(AdapterView<?> parent, View view,  
int position, long id) {
```


.. 중략...

```
// 현재 클릭한 위치의 영화스케줄 정보를 가져온다
MovieSchedule ms = (MovieSchedule)
parent.getItemAtPosition(position);
// 선택한 영화의 제목 정보 저장
final String selectedMovie = ms.getMovieTitle();
// 전역변수로 선언해둔 THIS_YEAR과 사용자의 생년정보를
이용해 사용자의 나이를 계산한다.
final int user_age = THIS_YEAR - user.getBirth().getYear();

// 사용자가 예매를 할 수 있는 나이인지 검사하는 변수
isEnoughAge = false;
// 영화DB에 리스너를 달아 영화의 등급정보를 가져온다.
movieReference.addListenerForSingleValueEvent(new
 ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
        for(DataSnapshot data : dataSnapshot.getChildren()){
            Movie movie = data.getValue(Movie.class);
            if(isEnoughAge)
                break;
            // 선택한 영화의 제목과 같은 정보를 찾아 등급정보를
            저장한다.
            if(movie.getTitle().equals(selectedMovie)){
                movieRate = movie.getRating();
                // 일치하는 그 영화에 대해 사용자의 나이가 등급보다
                많거나 전체상영가의 영화라면
                // isEnoughAge를 갱신한다.
                if(user_age>movieRate || movieRate==1){
                    isEnoughAge = true;
                }
            }
        }
        if(isEnoughAge){
            // 다음 액티비티로 정보를 전송한다.
            Intent intent = new Intent(getApplicationContext(),
BookMovieActivity.class);
            intent.putExtra("user", user); // 현재 로그인 유저
            intent.putExtra("key", key); // 데이터베이스 접근 키
            intent.putExtra("date", date);
            startActivity(intent);
        }else{
            AlertDialog.Builder alertBuilder = new
AlertDialog.Builder(UserMainActivity.this); //context는 직접
명시해야 함.
            alertBuilder.setTitle("예매 불가!");
```

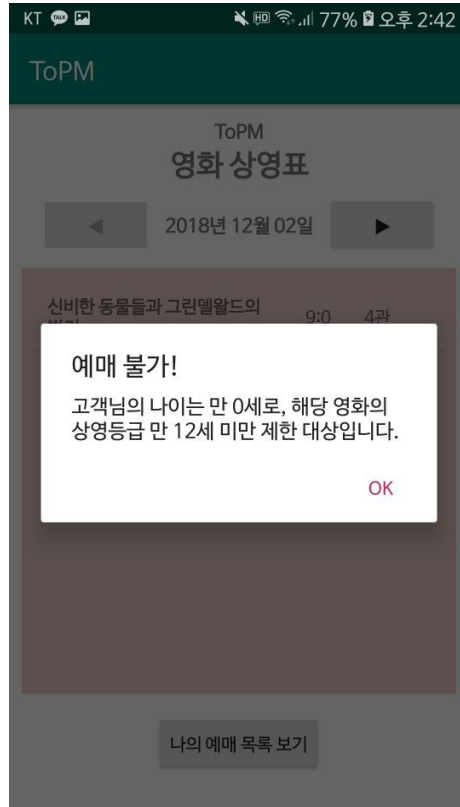
```

        alertBuilder.setMessage("고객님의 나이는 만
        "+user_age+"세로, 해당 영화의 상영등급 만 "+movieRate+"세
        미만 제한 대상입니다.");
        alertBuilder.setPositiveButton("ok", new
        DialogInterface.OnClickListener() {

... 후락
    }








```

그 외 관람할 수 없는 경우 alertMessage를 띄워 알려준다.



- BookMovieActivity

- 좌석 상태 출력

- 1) 좌석인 경우 
- 2) 좌석이 아닌 경우 이미지를 출력하지 않는다.
- 3) 우등석인 경우 
- 4) 커플석인 경우 
- 5) 이미 예매 된 자리인 경우 
- 6) 현재 사용자가 선택한 자리인 경우   

- 현재 날짜의 상영 시간표를 Firebase 서버에서 받아와 ListView로 출력한다. 상영 횟수만큼 ListView의 row가 동적할당 된다.
- **showMyBookingBtn 클릭 이벤트**
myBookingListActivity (예매 확인 액티비티)로 전환된다.
- 날짜를 변경할 경우 변경한 날짜의 상영 시간표를 Firebase DB에서 받아와 영화별로 ListView를 출력한다. 이 때, 변경할 수 있는 날짜의 범위는 **final int FUTURE_DATE = 4;** 현재 날짜를 제외해 3일로 상수로 선언해둔다. 영화이름 출력 후 ListView순서로 출력한다.
- **completeBtn 클릭 이벤트**
선택한 예매 정보를 확인해 예매를 확정한다.

분기 1) 인원을 입력하는 EditText에 아무것도 입력되지 않았는지를 확인한다. 아무것도 입력되지 않았다면 “인원수를 입력하세요”라는 토스트 메시지를 띄운다.

분기 2) 인원을 입력하는 EditText의 값이 1미만, 혹은 남은 좌석수 초과일 때 “인원수를 확인해주세요”라는 토스트 메시지를 띄운다.

분기 3) 입력한 인원 수 만큼 좌석이 선택되었다면 예매를 확정한다. 좌석 정보를 Firebase DB 서버에 갱신한다.

분기 4) 입력 된 인원 수보다 적은 좌석이 선택되었다면 “좌석을 모두 선택하세요.” 라는 토스트 메시지를 띄운다. 입력한 인원 수보다 많은 좌석을 선택할 경우, 더 이상 좌석이 선택되지 않고 토스트 메시지로 경고 메시지를 띄운다.

모든 분기를 통과하면 BookingInfo 인스턴스를 생성해 데이터베이스의 bookingInfo노트 아래 bookingInfoKey (사용자 ID + 해당 상영 회차 key)에 객체로 저장한다. 또한, 예매한 유저가 가지고 있는 bookedSchedules 배열에 bookingInfoKey를 저장하고 모든 이 모든 정보를 데이터베이스에 업로드 한다.

```
public void saveToDataBase() {
    /* MovieSchedule 갱신 */
    // movieSchedule.setBookedMap(booked);
    bookedMap을 갱신한다.
    movieSchedule.setBookedMap(tempBooked);
    bookedMap을 갱신한다.

    scheduleReference.child(strDate).child(scheduleKey).setValue(movieS
    chedule); // 객체를 데이터베이스에 업로드한다.
```

```

Toast.makeText(this, "영화 예매가 완료되었습니다.",
Toast.LENGTH_SHORT).show();

// 유저의 예매 리스트에 추가하기!!!
// public BookingInfo(String userID, String scheduleKey, int personnel,
ArrayList<String> bookedSeats, Date screeningDate, String title)
BookingInfo bookingInfo = new BookingInfo(user.getId(), scheduleKey,
personnel, bookedSeats, movieSchedule.getScreeningDate(),
movieSchedule.getMovieTitle()); // bookingInfo 객체 생성
String bookingInfoKey = user.getId() + " " + scheduleKey; // 예매
정보 키: 유저 아이디 + 스케줄 키

bookingInfoReference.child(bookingInfoKey).setValue(bookingInfo);
user.bookedSchedules.add(bookingInfoKey); //
사용자의 예매 목록에 이번 예매의 키를 추가한다.
userReference.child(user.getId()).setValue(user); //
데이터베이스에 유저 정보 갱신
... 후략
}

```

- 인원 입력 form (editPersonnel) : 숫자만 입력 가능한 EditText 위젯이다.
- 좌석 선택 form : 안드로이드 기본 위젯 Button을 상속받아 만든 커스텀 클래스를 배열로 사용해 현재 좌석 상태를 표현한다. 이때, 좌석 상태는 boolean 변수로 표현되고 상수로 선언된다.
- 좌석을 선택하기 전, 반드시 인원을 먼저 입력해야 한다. 인원을 입력하기 전에는 좌석 선택 창 클릭이 활성화되지 않는다.
- 저장이 완료되면 나의 예매 목록을 열람할 수 있는 MyBookingListActivity로 전환된다.

● myButton (좌석 버튼) 클릭 이벤트

현재 액티비티에 저장되어 있는 HashMap<String, Boolean> **booked**;에는 데이터베이스에서 받아온 현재 상형 회차의 좌석 별 예매 여부가 저장되어 있다. 이 HashMap을 HashMap<String, Boolean> **tempBooked**;에 복사한다. tempBooked는 이미 예매 된 좌석과, 현재 사용자가 선택한 좌석의 Value가 MyButton.**BOOKED** 로, 나머지 좌석들은 MyButton.**UNBOOKED**로 저장되어 있다. 이 HashMap을 참조해 좌석 클릭 시 분기한다.

분기 1) 예매 인원 입력이 완료 되었는가?

예매 인원이 입력되지 않았다면 좌석 선택이 불가능하다.

분기 1-1) 현재 선택된 좌석이 입력 인원보다 적은가?

분기 1-1-1) 현재 선택하지 않은 좌석인 경우

분기 1-1-1-1) 클릭한 좌석이 커플석인 경우

버튼 ID값을 통해 클릭한 좌석과 세트인 좌석이 왼쪽인지 오른쪽인지 알아낸 후, 두 좌석의 tempBooked의 Value를 MyButton.**BOOKED**로 변경한다. 좌석 선택 수를 저장하는

personnelCount의 값을 2 증가한다. 현재 선택한 좌석의 ID값을 저장하는 ArrayList인 bookedSeats에 두 좌석 ID를 add한다.

커플석이 아닐 경우 (일반석이거나 우등석일 경우), 좌석을 선택하면, **tempBooked**의 해당 좌석의 Value를 MyButton.**BOOKED**로 변경하고, 좌석 선택 수를 저장하는 personnelCount의 값을 1 증가한다. 현재 선택한 좌석의 ID값을 저장하는 ArrayList인 bookedSeats에 좌석 ID를 add한다.

```
tempBooked.put(String.valueOf(index), MyButton.BOOKED); // 선택으로  
상태 변경  
v.setBackgroundResource(R.drawable.movie_seat_select); // 좌석  
이미지 변경  
personnelCount++; // 인원 수 증가  
bookedSeats.add(String.valueOf(index)); // 예매 목록에 추가
```

분기 1-1-2) 클릭한 좌석이 이미 예매되지 않고, 현재 선택한 좌석인 경우

분기 1-1-2-1) 그리고 커플석인 경우

버튼 ID값을 통해 클릭한 좌석과 세트인 좌석이 왼쪽인지 오른쪽인지 알아낸 후, 두 좌석의 tempBooked의 Value를 MyButton.**UNBOOKED**로 변경한다. 좌석 선택 수를 저장하는 personnelCount의 값을 2 감소한다. 현재 선택한 좌석의 ID값을 저장하는 ArrayList인 bookedSeats에서 두 좌석 ID를 제거한다.

커플석이 아닌 경우, 선택한 좌석을 취소하는 경우이므로, **tempBooked**의 해당 좌석의 Value를 MyButton.**UNBOOKED**로 변경하고, 좌석 선택 수를 저장하는 personnelCount의 값을 1 감소한다. 현재 선택한 좌석의 ID값을 저장하는 ArrayList인 bookedSeats에서 좌석 ID를 제거한다.

```
// 현재 선택했고, 예매되지는 않은 자리  
// 선택을 취소한다.  
tempBooked.put(String.valueOf(index), MyButton.UNBOOKED);  
// 비선택으로 상태 변경  
v.setBackgroundResource(R.drawable.movie_seat_ok);  
// 좌석 이미지 변경  
personnelCount--; // 인원 수 감소  
bookedSeats.remove(String.valueOf(index));  
// 예매 목록에서 제거
```

분기 1-2) 현재 선택된 좌석이 입력 인원과 같은가?

현재 선택한 좌석만 취소할 수 있다. 분기 1-1-2에서의 작업을 동일하게 실행한다.

2.4.2. 예매 확인 액티비티

- 사용자가 지금까지 예매한 내역을 확인할 수 있는 액티비티
 - myBookingListActivity.java
 - activity_my_booking_list.xml
 - ListView widget - 1개 (myBookingList)

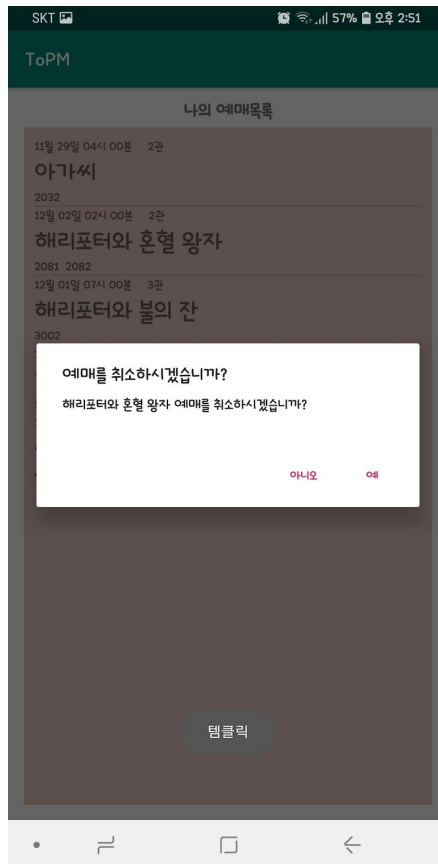


- 현재 로그인 한 회원의 예매 내역 (bookedSchedules)을 Firebase DB 서버에서 받아와 myBookingList에 출력한다.

```
bookingInfoReference.addListenerForSingleValueEvent(new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        for(DataSnapshot data : dataSnapshot.getChildren()) {
            BookingInfo b = data.getValue(BookingInfo.class);

            if(b.getUserID().equals(user.getId())) {
                // 현재 사용자가 예매한 정보만
                bookingData.add(b); // 예매 정보 저장
                adapter.notifyDataSetChanged();
            }
            else continue;
        }
    }
} .. 후락
```

- myBookingList 아이템 클릭 리스너



리스트뷰의 아이템을 클릭하면 예매를 취소할 것이냐고 묻는 다이얼로그가 뜬다. 예를 클릭할 경우, bookedData에서 해당 예매 정보를 지우고, Firebase DataBase에서도 bookingInfo 노드 아래 정보를 지우며, 해당 schedule의 BookedMap의 예매되었던 좌석을 MyButton.UNBOOKED 상태로 변경해 갱신한다.

```

scheduleReference.child(screenDate).addListenerForSingleValueEvent(
new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        // DB에 저장된 스케줄 검사
        for(DataSnapshot data : dataSnapshot.getChildren()) {

            if(data.getKey().equals(bookingInfo.getScheduleKey())) {
                // 예매한 바로 그 스케줄이다... 그렇다면!!!
                MovieSchedule movieSchedule =
                data.getValue(MovieSchedule.class);

                for(int i=0; i<bookingInfo.getBookedSeats().size(); i++) {
                    // 예매한 좌석 수만큼 반복합니다
                    String seatNum = bookingInfo.getBookedSeats().get(i); // 좌석
번호를 획득
                    movieSchedule.getBookedMap().put(seatNum,
                    MyButton.UNBOOKED); // 예매되지 않음으로 상태 변경
                }

                scheduleReference.child(screenDate).child(bookingInfo.getScheduleKey
                ()).setValue(movieSchedule); // 객체를 데이터베이스에 업로드한다.
            }
        }
    }
}

```

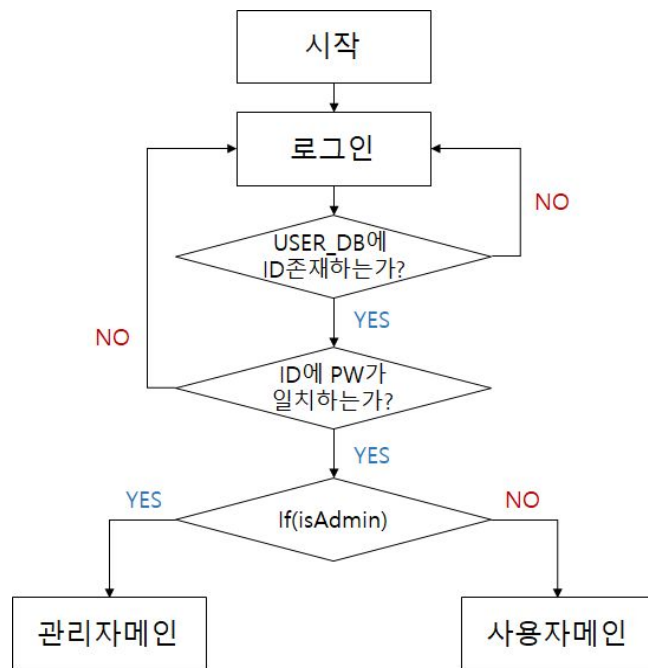

	Boolean> bookedMap; public Date screeningDate; // 상영 날짜 int bookedSeats; // 예약된 좌석 숫자 int restSeats; // 남은 좌석 숫자 int endHour; int endMin; int runningTime;		endHour, endMin, runningTime
Screen(상영관)	int row; // 행 int col; // 열 int totalSeats; // 총 좌석 개수 String screenNum; // 상영관 번호 HashMap<String, Boolean> abledMap; // 좌석인지 아닌지 여부 저장 HashMap<String, Boolean> specialMap; // 우등석인지 아닌지 여부 저장	public int getTotalSeats() // 전체 좌석 수 반환하는 함수	
MyButton(좌석)	없음	/* 상수 */ public final static boolean ABLED = true; // 좌석입니다. public final static boolean UNABLED = false; // 좌석이 아닙니다. public final static boolean BOOKED = true; // 예약되었습니다. public final static boolean UNBOOKED = false; // 예약 안 되었습니다. public final static boolean SPECIAL = true; // 우등 좌석입니다. public final static boolean UNSPECIAL = false; // 일반 좌석입니다.	
BookingInfo (예매정보)	String userID; // 예매자 아이디 String scheduleKey; // 예매한 스케줄 key int personnel; // 인원 ArrayList<String>		Movie 참조를 위한 title

	<pre> bookedSeats; // 예매한 자리 번호 저장 (버튼 ID) Date screeningDate; // 상영 시간 String title; // 영화 제목 </pre>		
--	---	--	--

4. Flow Chart

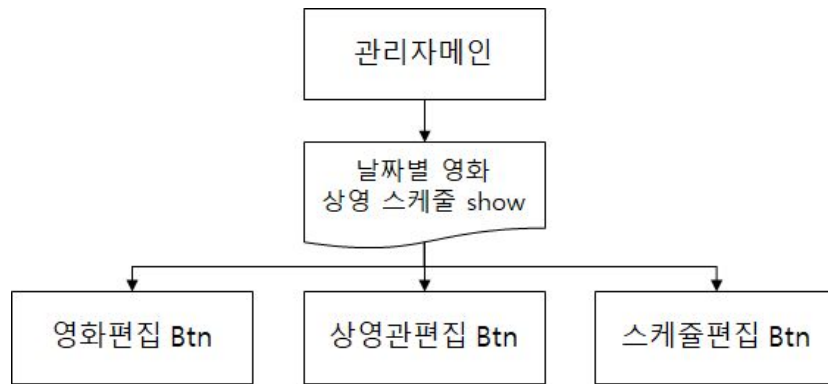
4.1. 동작 흐름

4.1.1. 로그인 순서도

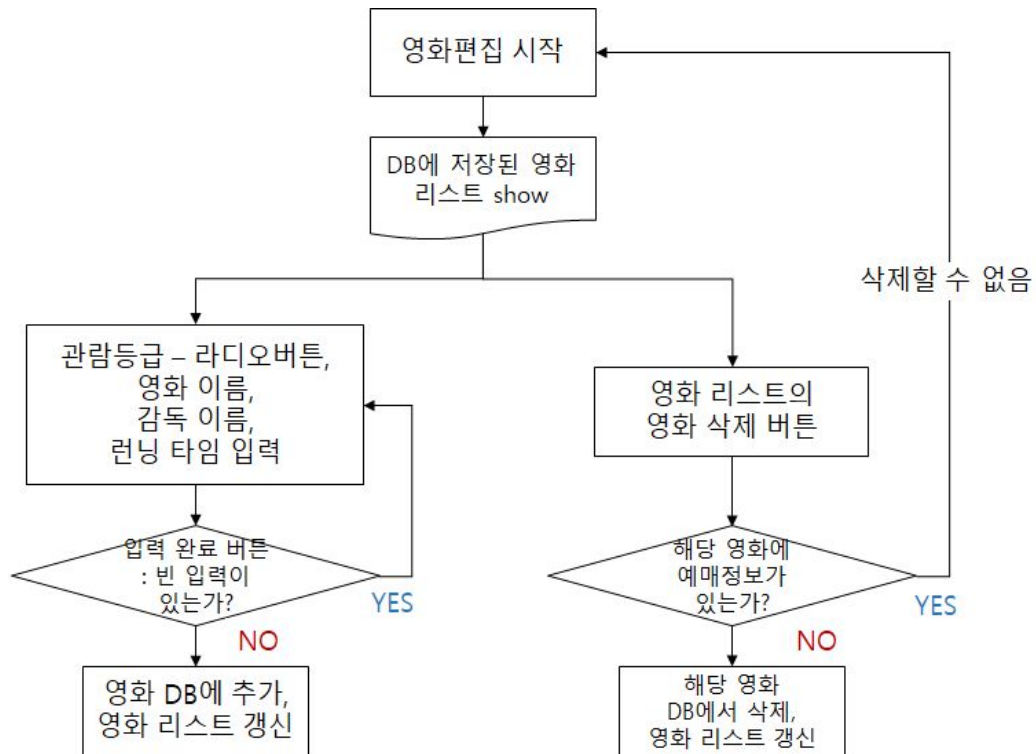


로그인 순서도

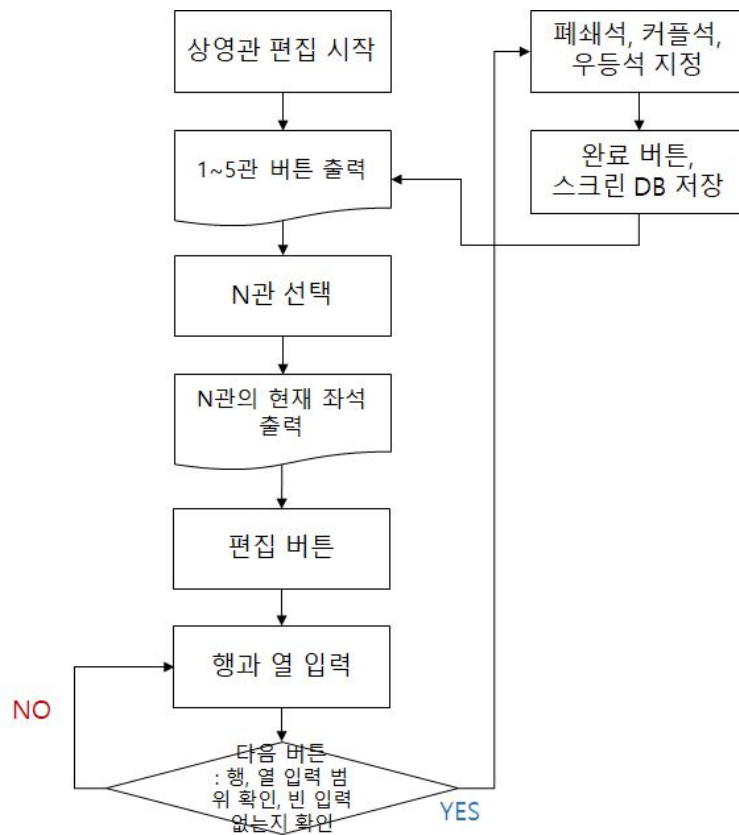
4.1.2. 관리자 동작흐름 - 등록 순서도



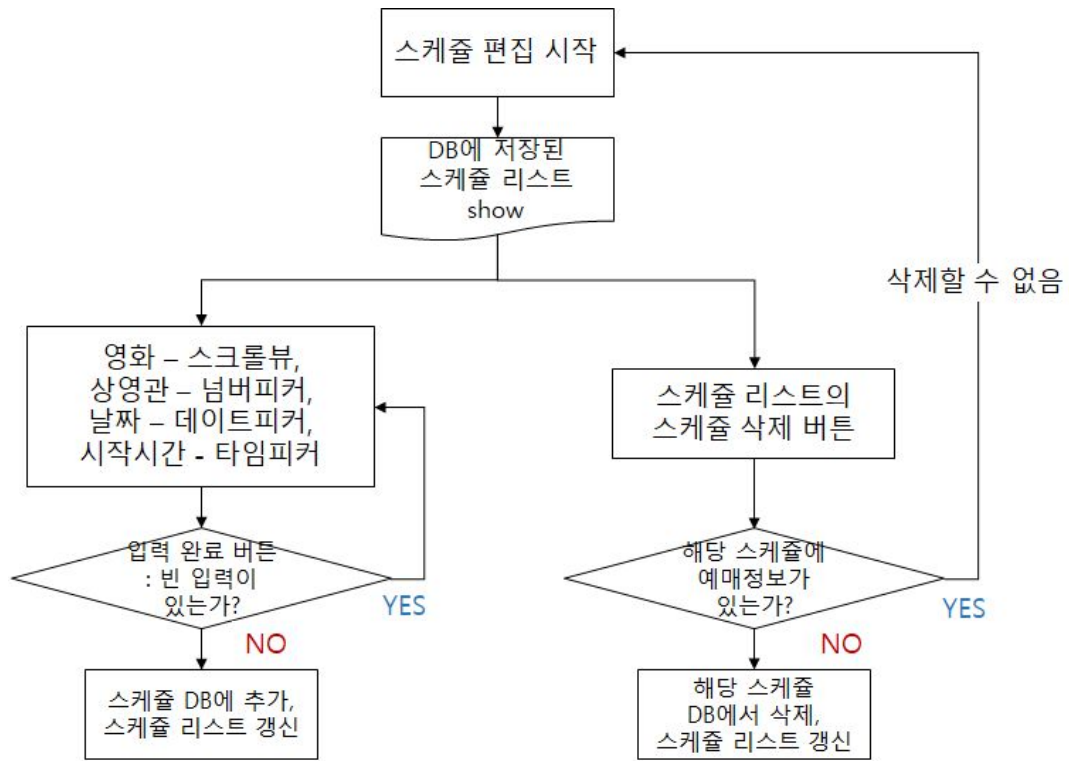
관리자 메인 순서도



영화편집 순서도



상영관 편집 순서도



스케줄편집 순서도