

2018 산학협력프로젝트2 일반반 (2037) 설계 문서

: (2조) 영화 상영관 예매 시뮬레이션 프로그램



2 조

201611186 김나경

201611227 임현정

201514517 하자뢰

201611245 홍혜진

제출일

2018년 10월 2일

[목차]

1. 프로젝트 팀 구성

2. 동작 흐름

2.1 초기화면

2.2 관리자 view

2.3 사용자 view

3. 클래스 설계

4. Flow Chart

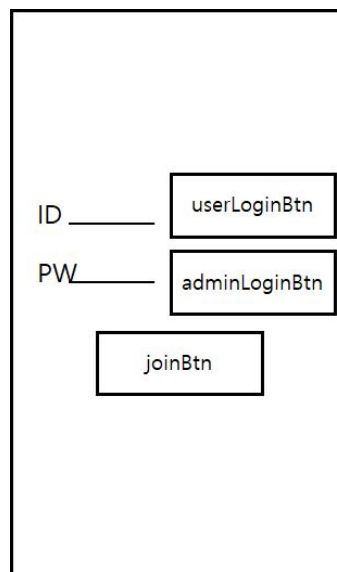
1. 프로젝트 팀 구성

2조	
김나경	201611186
임현정	201611227
하자뢰	201514517
홍혜진	201611245

2. 동작 흐름

2.1. 초기 화면

- 로그인
 - LoginActivity.java
 - activity_login.xml
 - TextEdit Widget 2개 - login_id, login_pw
 - Button 3개 - userLoginBtn, adminLoginBtn, joinBtn



- void loginCheck()**

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

분기 1)

ID와 PW가 저장된 회원 정보와 일치한다면 로그인에 성공해 UserMainActivity로 연결된다. ID와 PW가 저장된 회원 정보와 불일치한다면 로그인에 실패해 ID, PW 입력 란이 초기화된다.

- void adminLogInCheck()**

입력한 ID와 PW가 관리자 계정인지 확인한다.

분기 1)

ID, PW가 모두 문자열 “admin”일 시 관리자 로그인에 성공해 AdminMainActivity로 전환된다.

- **joinBtn의 클릭 이벤트**

회원가입을 할 수 있는 JoinActivity로 전환한다.

2.2. 회원가입

- **회원가입**

- JoinActivity.java
- activity_join.xml
- EditText Widget 3개 - id_join, pw_join, name_join
- Button Widget 3개 - idCheckBtn, joinBtn, birthBtn

The image shows a vertical layout for a registration form. It includes labels and input fields for 'ID', 'PW', 'Name', and 'Birth'. The 'PW' field is masked with four dots. There are three buttons: 'idCheckBtn' next to the ID field, 'birthBtn' next to the Birth field, and a larger 'joinBtn' at the bottom center.

- **idCheckBtn 클릭 이벤트**

사용자가 입력한 ID가 이미 Firebase DB 서버에 존재하는지 검사한다.

분기 1)

ID가 이미 Firebase DB 서버에 존재할 경우, “이미 존재하는 아이디입니다.” 라는 토스트 메시지를 띄운다.

존재하지 않을 경우, ID를 “사용 가능한 아이디입니다.”라는 토스트 메시지를 띄우고, id_join을 받는 EditText 위젯을 비활성화 한다.

- **birthBtn 클릭 이벤트**

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

DatePicker로 생일날짜를 입력받으며, 받은 날짜는 String형태의 지역변수에 임시저장한다.

- **joinBtn 클릭 이벤트**

User 클래스의 생성자로 인스턴스를 하나 생성한다.

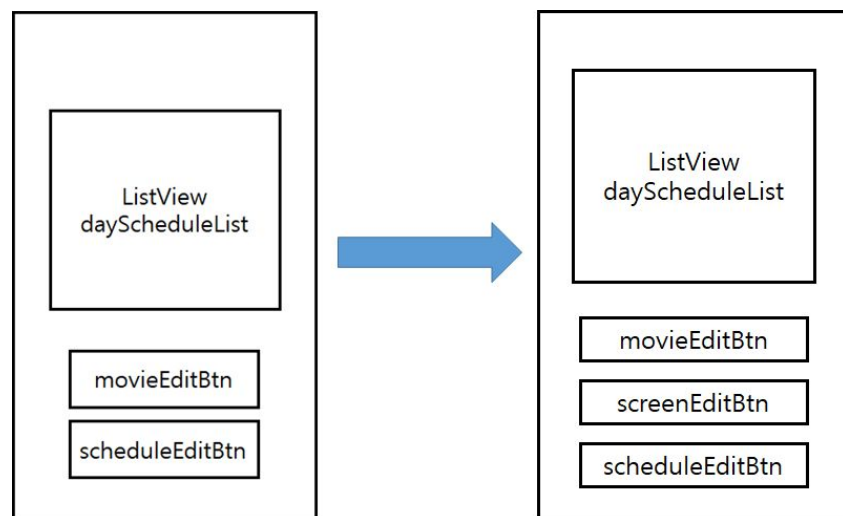
User(String id, String pw, String birth, String name)

Firebase의 *DatabaseReference* 인스턴스를 생성하여 생성한 User 인스턴스를 추가한다.

분기1) 정상적으로 추가 됐을 때, 회원가입을 축하하는 토스트 메시지를 띄우고, LoginActivity로 이동한다. 정상적으로 추가되지 않았다면 현재 View에 머무른다.

2.3. 관리자 View

- 관리자의 ID,PW는 둘다 “admin” - 상수로 선언해둔다.
- 관리자의 Main View
 - AdminMainActivity.java
 - activity_admin_main.xml
 - ListView 1개, 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.



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_name, movie_director, movie_time

- **boolean deleteCheck()**

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

분기 1) 해당 영화가 스케줄에 등록돼 있고(1), 해당 스케줄을 단 1명이라도 예매한 고객이 있다면(2) false를, 예매한 고객이 없다면 true를 반환한다.

(1)만 만족한다면 - 스케줄에 등록만 돼있고, 아무도 예매를 안한상태

(2)는 (1)을 만족해야 가능한 조건. 스케줄이 등록돼있지 않다면 예매 불가이므로

분기는 (1)과 (2)를 모두 만족해야한다.

deleteBtn 클릭 이벤트에서 이 함수의 리턴값을 받아 영화를 삭제할 수 있는지 여부를 검사해 삭제 작업을 진행한다.

- **completeBtn 클릭 이벤트**

모든 입력 칸이 정상적으로 채워졌는지 검사한다.

분기 1) 모든 칸을 채우지 않았으면 영화 추가 작업을 완료할 수 없고, 현재 View에 머무른다. 모든 칸이 정상적으로 채워진 경우 Firebase DB에 영화 정보가 업로드되고, movieList에 영화 정보가 추가된 후 AdminMainActivity로 전환된다.

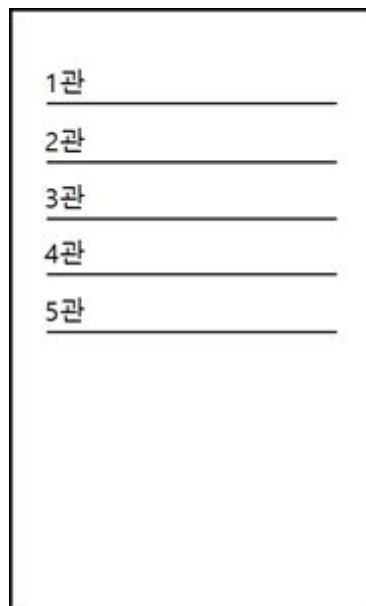
이때 '모든 칸이 정상적으로 채워졌다'는 것은 이름, 감독, 러닝타임을 받는 EditText가 getText를 했을 때, null을 반환하지 않을 때이고, 특히 러닝타임 같은 경우는 1이상의 양의 정수일때를 말한다. 나머지의 경우에는 (ex.. 음수, 실수)는 다이얼로그를 띄워서 제재 후 현재 View에 머무른다.

- **deleteBtn 클릭 이벤트**

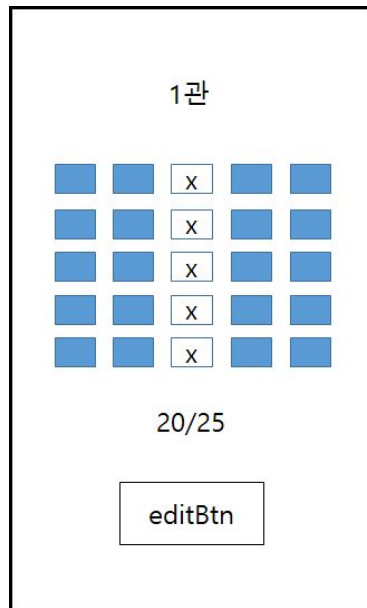
deleteCheck() 함수를 실행해 해당 행의 영화가 삭제 가능한 영화인지 검사한다. deleteCheck() 함수의 반환값이 true일 경우 해당 행의 영화 정보가 Firebase DB에서 삭제되며, 리스트에서도 삭제된다. false일 경우에는 “영화를 삭제할 수 없습니다.”라는 토스트 메시지를 띄우고, 영화가 삭제되지 않으며 현재 View에 머무른다.

2.3.2. 상영관 좌석 수정 액티비티

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



- ScreenShowActivity.java
- activity_screen_show.xml
- Button Widget 1개
- 해당 상영관의 좌석 정보를 알려줌.



- **editBtn 클릭 이벤트**

editCheck()함수가 return true를 할 시에만 ScreenEditActivity로 연결함. return false일시에는 토스트메세지로 수정할 수 없음을 알리고 해당 뷰에 머무른다.

- **bool editCheck()**

분기 1) 해당 상영관이 스케줄에 등록돼 있지 않다면 → return true

분기 2) 해당 상영관이 스케줄에 등록돼 있다면 → 해당 스케줄을 단 1명이라도 예매한 고객이 있다면 return false, 예매한 고객이 없다면 return true한다.

- ScreenEditActivity1.java
- activity1_screen_edit.xml
- EditText Widget 2개 - row, col

1관

가로

세로

nextBtn

- **nextBtn 클릭 이벤트**

row값과 col값이 공백인지 검사한다.

row 값이 ROW_MAX이하, ROW_MIN이상

col 값이 COL_MAX이하, COL_MIN이상 인지 검사한다.

모든 검사가 통과하면 row, col 값을 가지고 다음 액티비티로 넘어간다. ScreenEditActivity2

- ScreenEditActivity2.java
- activity2_screen_edit.xml
- Button Widget - col*row개
- TextView 1개

1관

X		X		
		X		
		X		
		X		
		X		

19/25

saveBtn

- row,col만큼 MyButton클래스가 동적할당된다.
- MyButton의 mode는 UNBOOKED(0)으로 초기화된다.
- 버튼 배열을 누르면 해당 위치의 좌석(MyButton)의 mode가 UNABLED(-1)로 저장된다. 즉, 비활성화, 없는 좌석이 된다.

- 버튼 배열 아래에 있는 TextView는 (활성화된 좌석)/(row*col)을 버튼 상태가 바뀔때마다 갱신해서 보여준다.
- **nextBtn 클릭 이벤트**
saveBtn을 누를 시 결과가 저장되고, AdminMainActivity.java로 넘어간다.

2.3.3. 상영 등록/제거 액티비티

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

- **boolean deleteCheck()**
상영회차를 삭제하려고 시도할 때, 해당 회차를 삭제할 수 있는지 검사한다.

분기 1) 해당 회차를 예매한 고객이 1명도 존재하지 않는다면 true를 반환한다. 예매한 고객이 존재한다면 false를 반환한다.

deleteBtn 클릭 이벤트에서 이 함수의 리턴값을 받아 회차를 삭제할 수 있는지 여부를 검사해 삭제 작업을 진행한다.
- **completeBtn 클릭 이벤트**
모든 입력을 정상적으로 받았는지 검사한다.

분기 1) 모든 입력을 채우지 않았으면 회차 추가 작업을 완료할 수 없다. 모든 입력이 정상적으로 채워진 경우 Firebase DB에

회차 정보가 업로드되고, dateScheduleList에 영화 정보가 추가된 후 AdminMainActivity로 전환된다.

위의 ‘입력을 채운다’는 것은 다음과 같다.

ComboBox에서는 MovieManageActivity에서 미리 추가해 둔 영화를 보여주며, 그 영화 중에서만 선택가능하다.

NumberPicker에서는 상영관의 번호를 받으며, 1관부터 5관까지 선택가능하도록 한다.

TimePicker에서는 영화의 시작시간을 받으며, 선택된 영화의 러닝타임 정보로 자동으로 종료시간을 계산한다.

위의 ‘모든 입력을 정상적으로 받았다’는 것은 이 세 항목에 입력 값이 있고, isEmpty()함수의 리턴값이 null이 아닐 때를 말한다.

- **MovieSchedule isEmpty()**

상영 회차를 추가하려는 상영관이 해당 시간에 비어있는지 확인한다.

completeBtn 클릭 이벤트가 발생하면 실행되며, 입력받은 정보를 사용해 MovieSchedule 객체를 생성한다.

분기 1) 이미 상영 리스트에 등록되어 있는 MovieSchedule 배열의 요소들과 현재 새로 생성된 객체의 상영관 번호와 상영 시간을 비교한다. 상영관 번호와 상영 시간이 중복된다면 null을 반환한다. 중복되지 않는다면 생성해둔 MovieSchedule객체를 반환한다.

이때 상영 시간의 중복이란, 영화의 시작시간부터 러닝타임 후 종료시간까지 비교하는 두 영화의 시간이 1분이라도 겹치는지 확인하는 것을 말한다.

- **deleteBtn 클릭 이벤트**

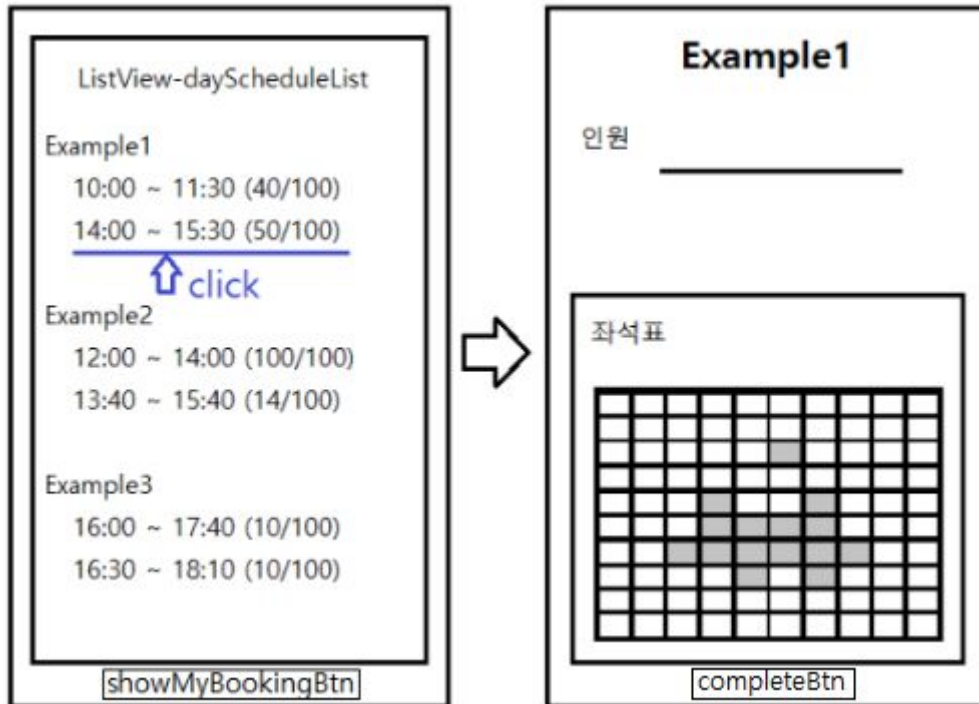
deleteCheck() 함수를 실행해 해당 행의 회차가 삭제 가능한 영화인지 검사한다. deleteCheck() 함수의 반환값이 true일 경우 해당 행의 회차 정보가 Firebase DB에서 삭제되며, 리스트에서도 삭제된다. false일 경우에는 “회차를 삭제할 수 없습니다.”라는 토스트 메시지를 띄우고, 회차가 삭제되지 않는다.

2.4. 사용자 View

2.4.1. 예매 액티비티

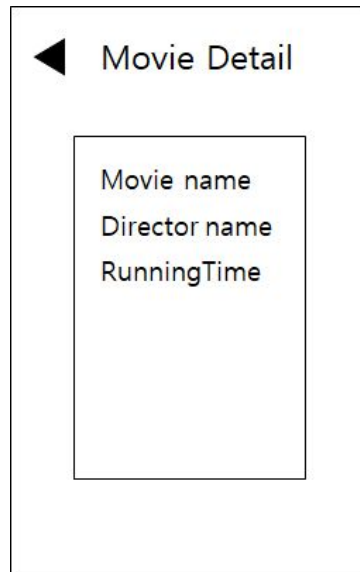
- 사용자가 영화를 예매할 수 있는 액티비티
 - ShowScheduleActivity.java
 - activity_show_schedule.xml
 - bookMovieActivity.java

- activity_book_movie.xml
- ShowMovieInfoActivity.java
- activity_show_movie.xml
- 멤버 변수
 - final int THIS_YEAR // 올해 연도



showScheduleActivity → bookMovieActivity

- 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 현재날짜를 제외해 3일로 상수로 선언해둔다. 영화이름 출력 후 ListView순서로 출력한다.
- **dayScheduleList 위 영화이름 클릭 이벤트**
ShowMovieInfoActivity (영화 상세 정보 확인 액티비티)로 전환된다. ◀ 버튼을 누를시 이전 액티비티로 전환된다.



- **dayScheduleList row 클릭 이벤트**

분기 1) 예매 가능한 좌석이 남아있다면 bookMovieActivity로 전환되고, 모든 좌석이 이미 예매 완료된 회차라면 클릭해도 bookMovieActivity로 전환되지 않고, “이미 매진된 회차입니다.”라는 토스트 메시지가 뜬다.

- **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 추가한다.

- 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.

- **myButton (좌석 버튼) 클릭 이벤트**

버튼을 누르면 멤버 함수인 bookingComplete() 를 호출해 멤버 변수 mode의 값을 바꾼다.

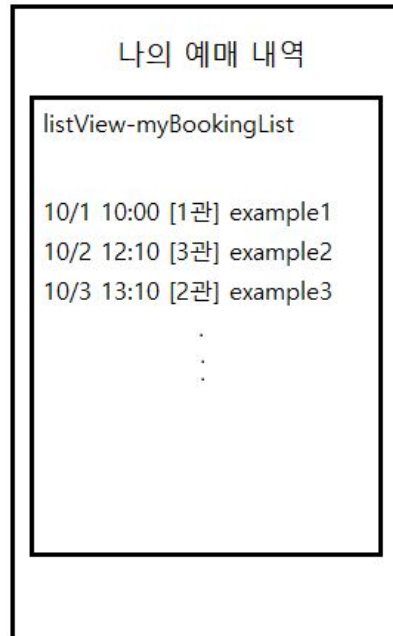
분기 1) 현재 좌석의 mode 값이 UNBOOKED (false)일 경우 BOOKED (true)로, BOOKED (true)일 경우 UNBOOKED (false)로 바꾼다.

- **boolean checkRating() 나이 계산 및 등급 확인 함수**

고객이 영화를 예매하기 위해 영화 제목을 클릭하면 가장 먼저 실행된다. 현재 연도를 사용해 고객의 나이를 계산한다.

2.4.2. 예매 확인 액티비티

- 사용자가 지금까지 예매한 내역을 확인할 수 있는 액티비티
 - myBookingListActivity.java
 - activity_my_booking_list.xml

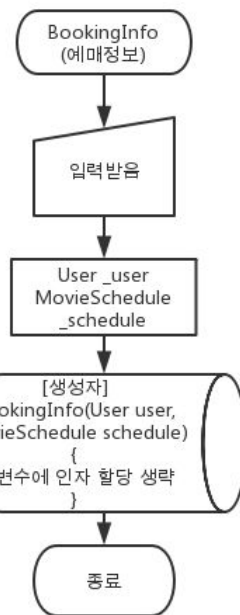
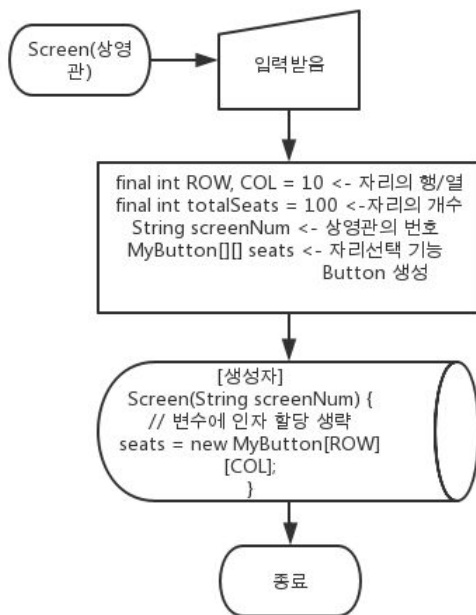
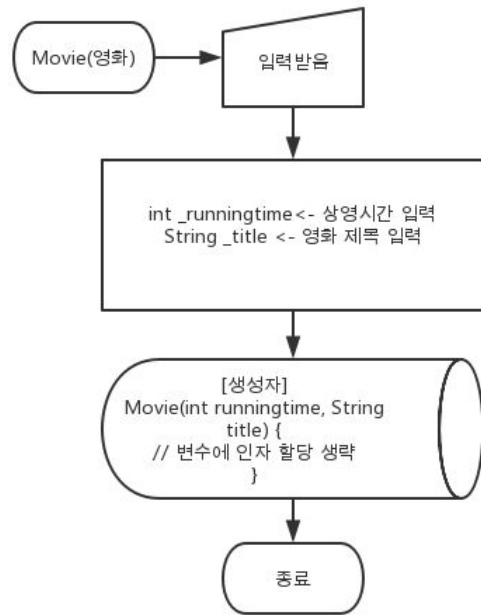
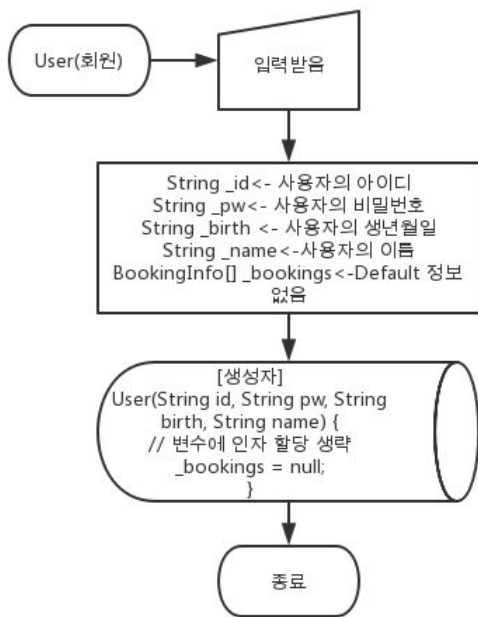


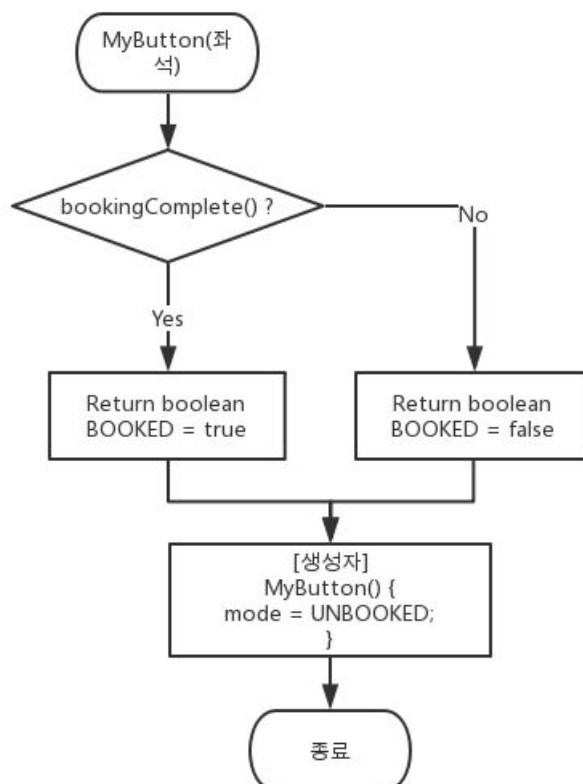
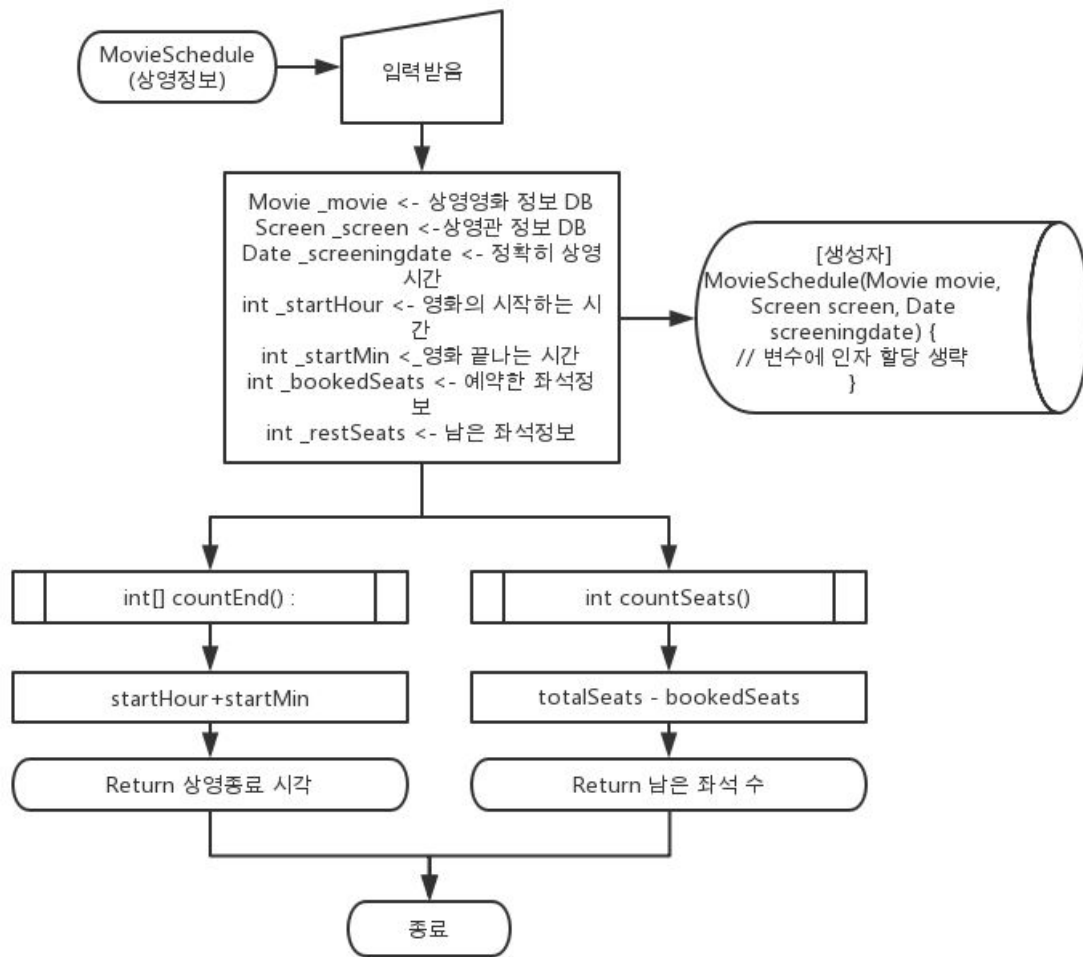
- bookMovieActivity (예매 액티비티)에서 completeBtn (예매 완료) 버튼을 클릭하면 myBookingListActivity (예매 확인 액티비티)로 전환된다.
- 현재 로그인 한 회원의 예매 내역을 Firebase DB 서버에서 받아와 출력한다.

3. 클래스 설계

클래스	멤버변수	멤버함수	비고
User(회원)	String _id String _pw String _birth String _name BookingInfo[] _bookings	[생성자] User(String id, String pw, String birth, String name) { // 변수에 인자 할당 생략 _bookings = null; }	
Movie(영화)	int _runningtime String _title String _director int rating // 상영등급	[생성자] Movie(int runningtime, String title, String director) { // 변수에 인자 할당 생략 }	
MovieSchedule (상영정보)	Movie _movie Screen _screen Date _screeningdate int _startHour int _startMin int _bookedSeats int _restSeats	[생성자] MovieSchedule(Movie movie, Screen screen, Date screeningdate) { // 변수에 인자 할당 생략 } int[] countEnd() : startHour와 startMin에 runningtime(분)을 더하여 상영종료 시각을 구해 리턴하는 함수 int countSeats() : totalSeats 에서 bookedSeats 을 빼서 남은 좌석 수를 리턴하는 함수	Movie 클래스 포함 Screen 클래스 포함
Screen(상영관)	int row, col; int totalSeats; String screenNum final int ROW_MAX = 20; final int COL_MAX= 20; final int ROW_MIN = 5; final int COL_MIN =5; MyButton[][] seats	[생성자] Screen(int row, int col, String screenNum) { // 변수에 인자 할당 생략 seats = new MyButton[row][col]; totalSeats = row * col; }	좌석의 행과 열은 관리자가 직접 입력할 수 있게 함. 행과 열의 최대값은 상수로 정의해둠.
MyButton(좌석)	int mode; final int BOOKED = 1; final int UNBOOKED = 0; final int UNABLED = -1;	[생성자] MyButton() { mode = UNBOOKED; }	좌석의 상태 mode 변수는 상수로 관리할 것

		void bookingComplete() : 좌석이 예매되면 해당 좌석의 mode 정보를 변경하는 함수, mode 값에 따라 버튼의 색도 바꾼다.	
BookingInfo (예매정보)	User _user MovieSchedule _schedule	[생성자] BookingInfo(User user, MovieSchedule schedule) { // 변수에 인자 할당 생략 }	MovieSchedule 클래스 포함 User 클래스 포함

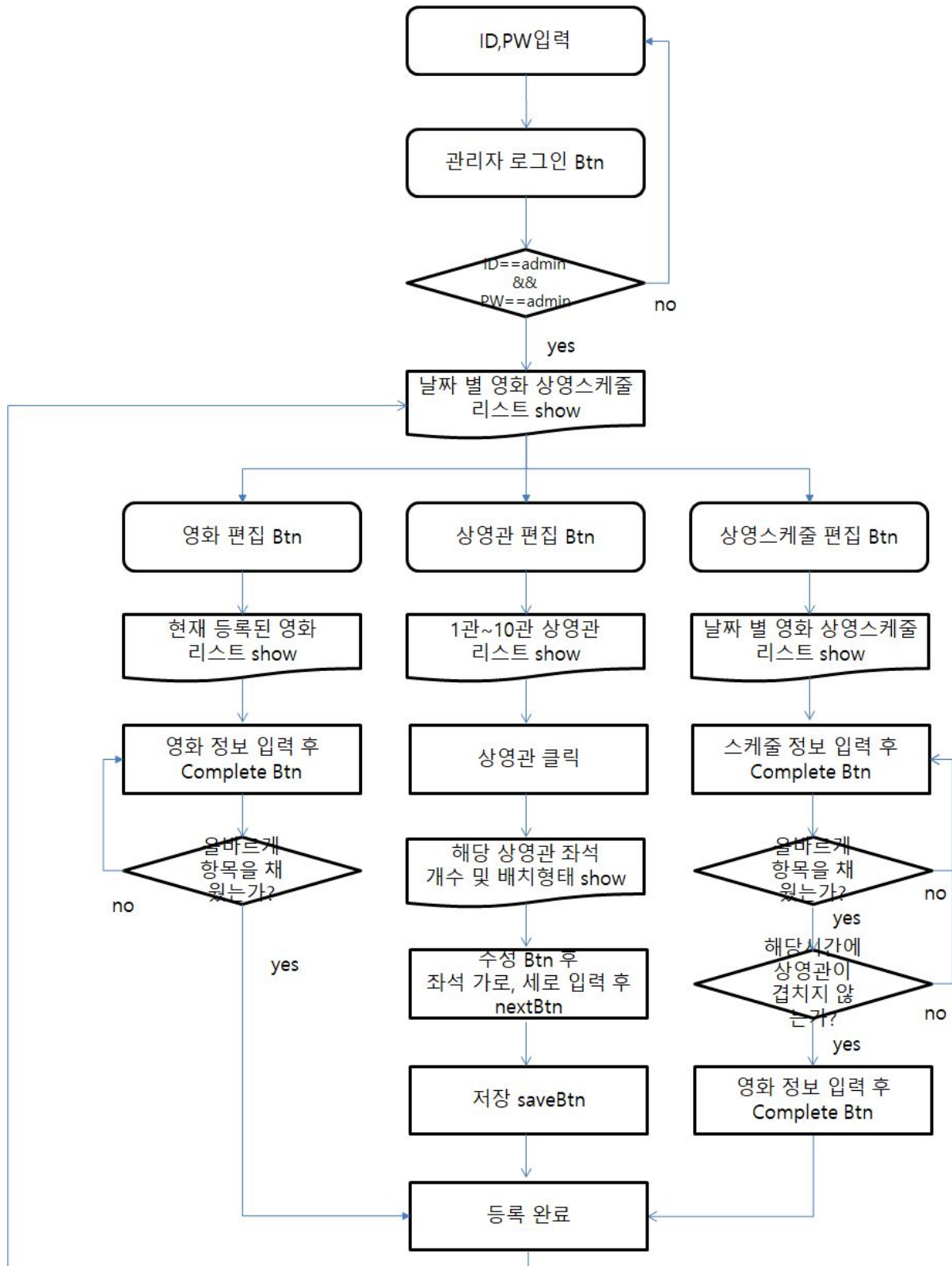




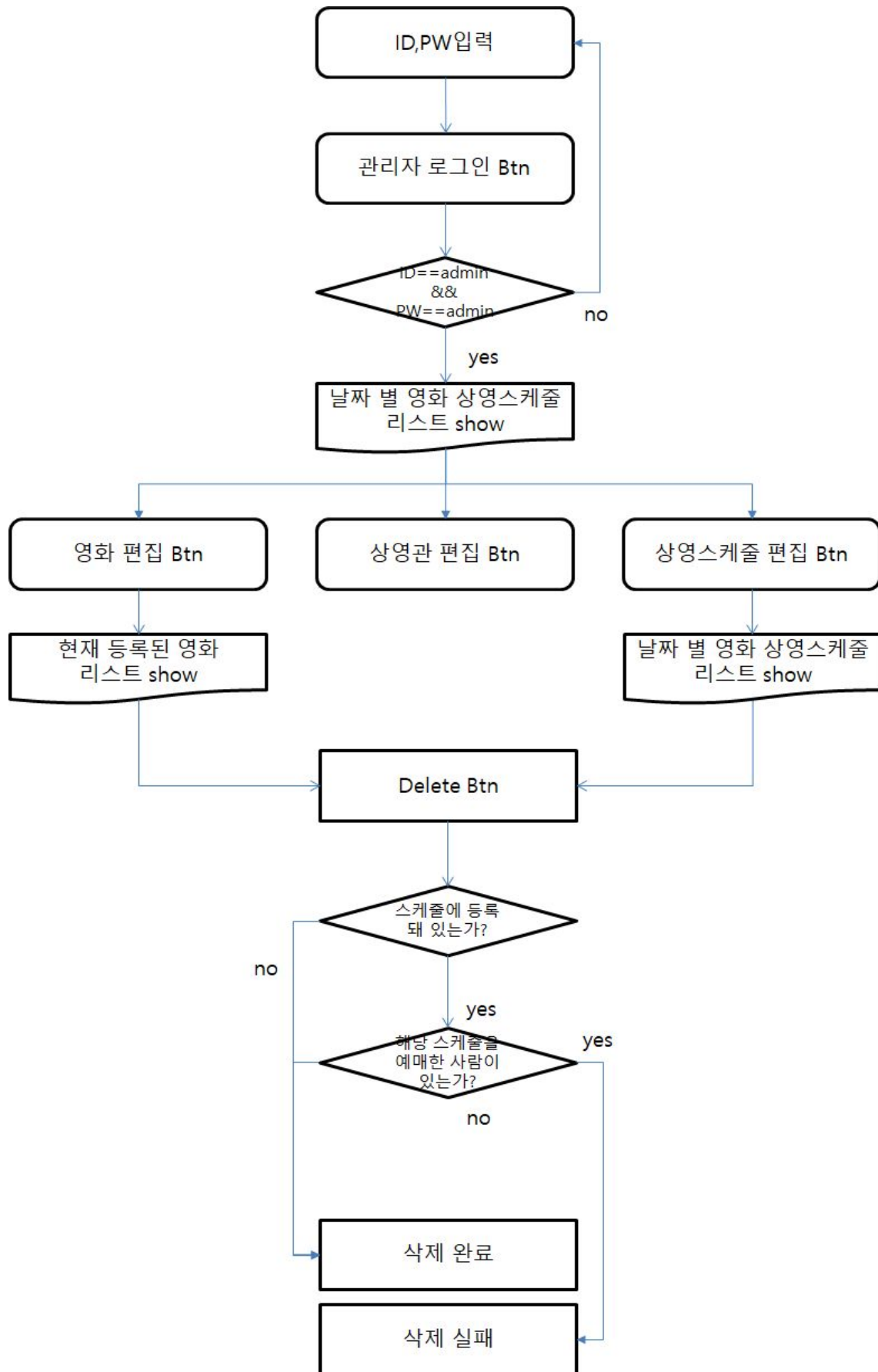
4. Flow Chart

4.1. 동작 흐름

4.1.1. 관리자 동작 흐름 - 등록 순서도



4.1.2. 관리자 동작흐름 - 삭제 순서도



4.1.3. 사용자 동작흐름

