

소프트웨어 실습 3

최종 보고서

목차

1. 과제개요

- 1) 프로젝트 배경
- 2) 개발 필요성 / 창의성
- 3) 개발 목표

2. 과제 수행 내용

- 1) 전체 구현 계획
- 2) 구현 상세
 - DB
 - ARTIK
 - Calendar
 - 문제풀이

3. 과제 수행 결과

- 1) 최종 개발 결과
- 2) 아쉬운 점 및 발전 가능성
- 3) 기대효과

1. 과제 개요

1) 프로젝트 배경

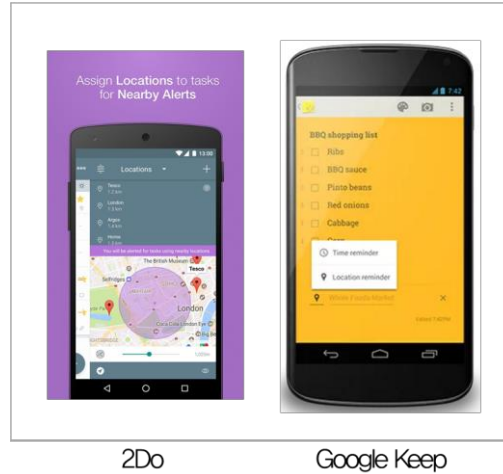
매년 사람들은 새해 다짐을 합니다. 올해는 꼭 본인이 가지고 있던 나쁜 버릇을 고치고, 보다 생산적인 삶을 살고자 스스로와 약속합니다. 이 때문에 새로운 한 해가 시작될 때마다 ‘금연보조제품’, ‘요가매트’, ‘다이어리’ 등의 상품 판매량도 순간적으로 증가합니다. 언론과 마케팅 시장에선 이것을 사람들의 다짐을 도와주는 상품이라 하여 ‘결심상품’이라 부릅니다. 우리가 주제로 꼽은 ‘스케줄러’ 역시 대표적인 결심상품 중 하나입니다. 자신의 모든 일정을 머릿속에 숙지하는 사람은 매우 드물기에, 사람들은 스케줄을 기록하고, 상기하며 자신의 루틴을 관리합니다. 이렇듯, ‘계획적인 삶을 살고 싶은 욕구’라는 의미에서 스케줄러는 소비자들의 니즈가 확실하다고 할 수 있습니다.

하지만 이미 기존 종이 재질, 그리고 PC 및 스마트 폰 단말기와 같은 플랫폼에서 스케줄러는 다양하게 존재합니다. 그럼에도 불구하고 우리가 전자 스케줄러를 개발 주제로 삼은 이유는 이러한 시제품들이 스케줄러의 ‘본질적 기능’을 개선할 여지가 있다고 판단했기 때문입니다. 본 프로젝트에서 정의한 스케줄러의 본질적 기능이란 ‘사용자가 자신의 일정을 상기하고, 나아가 전반적인 일정을 머릿속으로 기억할 수 있게 도와준다.’입니다. 기존 시제품들은 다양한 레이아웃, 나아가 알림 기능을 제공하지만, 근본적으로 사용자의 장기적인 일정의 상기를 유도하지 못하는 한계점이 있습니다.



출처: 다중기억이론 모형(이영애, 2012)

사람이 어떤 정보를 장기적으로 기억하기 위해선 순간적으로 집중하여 감각기억을 단기기억으로 옮기고 반복적인 암송을 통해 장기기억으로 옮기는 작업이 필요합니다. 하지만, 기존 스케줄러들의 알림 기능은 그저 배너 메시지 형태로 주어져 사용자가 집중해 살피지 않거나 스팸 메시지처럼 여겨질 확률이 너무 높습니다.



전자 스케줄러 중에선 알림을 받을 시간과 장소까지 정할 수 있는 기능을 제공하는 제품도 있었지만, 이것은 사용자에게 일정 등록 외에 다른 작업을 요구하는 일이 됩니다. 일정 알림을 언제 어디서 받아야할지 일일이 생각하는 것은 번거로운 일이고, '일정 상기를 위한 새로운 일정을 만든다.'는 다소 모순적 발상입니다.



따라서, 본 프로젝트에선 이러한 점을 개선하여 IoT 디바이스를 활용해 적절한 알림 상황을 감지하고, 일정을 단순한 배너 알림이 아닌 퀴즈 형태로 변환해 제공하는 '모더라' 어플리케이션을 개발하기로 하였습니다.

2) 개발 필요성 / 창의성



앞서 말했듯, 시중에 존재하는 전자 스케줄러들의 일정 알림 기능은 한계가 있습니다. 기본적으로 배너 알림으로 제공되기 때문에 사용자가 이를 스팸 문자처럼 무시하기 쉽고, 몇몇 어플리케이션은 특정 시간, 공간에서 알림을 받을 수 있는 기능을 제공하지만, 사용자가 일일이 설정을 추가해야하는 번거로움이 있습니다. 결과적으로, 일정 상기를 위한 집중력을 끌어내지 못하는 것입니다. 이에 대해 모데라 어플리케이션이 지향하는 차별성은 일정을 단순한 알림이 아닌, 중요한 일정들을 퀴즈 형태로 변환해 제공하고, 문제를 풀 적절한 타이밍을 IoT 디바이스를 활용해 감지하는 것입니다.



본 프로젝트에선 삼성 smart things를 활용해 사용자가 아침에 침대에서 일어나거나, 자신의 책상 앞에 앉는 등, 집이나 개인공간에서 잠깐 집중할 수 있는 때를 잡아냅니다. 하루를 시작하거나 자기 전, 혹은 책상에 앉아있는 순간이 사용자가 스마트폰을 킬 가능성이 높고, 잠깐 집중해서 퀴즈를 풀 수 있는 시간일 가능성이 높다고 판단했기 때문입니다. 따라서 모션센서를 침대 밑 혹은 협탁 밑에 달거나, 책상 밑에 설치해 위 행동들을 감지하여 일정 퀴즈 알림을 주도록 하였습니다.

4지선다문제

- ☐ 다음 중 오늘로부터 3주 뒤 있는 일정은 무엇인가요?
- ☐ 소프트웨어실습3 2차 발표를 한다
- ☐ 김윤성, 최준석과 강남에서 밥을 먹는다
- ☐ 김은빈, 윤응구와 사당에서 술을 마신다
- ☐ 아이캠퍼스에서 우주론 출책을 한다

OX문제

- ☐ 나는 2주 뒤에 소프트웨어실습3 조모임이 있다.
- ☐ O
- ☐ X

단순한 일정 알리는 사용자가 그저 읽고 지나칠 수 있기 때문에 일정을 기억하는데 큰 도움이 되지 못합니다. 따라서 모더라 어플리케이션은 모션센서에서 유효한 신호를 받으면 전체 스케줄 중에서 중요한 일정들을 퀴즈 형태로 변환해 사용자가 제한시간 내에 풀도록 합니다. 이를 통해 사용자는 1분 안에 5지선다와 OX 문제로 구성된 퀴즈를 풀며 마치 자신의 일정으로 만든 쪽지 시험을 보는 듯한 경험을 하게 됩니다. 이는 순간적인 집중을 요구하고 퀴즈를 푸는 과정에서 전반적인 일정을 상기하고 기억하도록 도와줍니다. 즉, 위 2가지 개선점을 통해 사용자는 적절한 타이밍에 짧은 시간동안 반복적으로 퀴즈를 푸는 행동을 통해 자신의 스케줄을 학습하게 됩니다.

3) 개발 목표

Modeola의 방향성이 정해지면서 개발 목표도 명확해 졌습니다. 스케줄러 앱인 만큼 기존의 스케줄러가 지원하는 기본적인 기능들을 제공하고, 추가적으로 Artik 연결 및 문제출제 기능이 필요했습니다. 구체적으로 살펴보면 다음과 같습니다.

A) 스케줄러

- 일정 관리

일정 관리는 여러 기능을 묶어서 통칭하는 말인데, 구체적으로는 일정을 추가, 수정 및 삭제할 수 있는 기능 등이 모두 포함됩니다. 일정을 저장하고 관리하는 데에는 데이터베이스가 사용됩니다. 사용자가 일일이 명령어를 사용해 데이터베이스를 통제할 수는 없으므로, 적절한 UI와의 연결이 필요합니다.

한편, Modeola는 문제출제를 위해 ‘누구와’ 어떤 일정을 하는지의 데이터가 필요합니다. 사용자의 편의성을 위해 이 부분은 전화번호부에 저장되어 있는 인물 목록을 연동할 수 있

도록 해야 합니다.

- 달력 UI 구현

달력 UI 역시 여러 가지 기능을 한데 묶어서 지칭했다고 봐야 합니다. 세분화하면 달력 화면 표시, 양옆 드래그 혹은 날짜 지정으로 달 전환, 날짜 일정이 있는 날은 달력에 표시, 날짜를 선택하면 그 날의 일정 표시/일정 수정 등이 있습니다. 사실상 일정 관리와 밀접하게 이어지는 부분으로써, 제대로 구현되지 않으면 원활한 앱 사용이 어렵다고 봐도 무방합니다.

B) ARTIK

- 로그인을 통해 Artik에 연결

Artik은 기본적으로 계정 로그인을 통해 클라우드 서비스에 연결하도록 하고 있습니다. 따라서 Artik을 활용하기 위해서는 사용자가 Artik에 로그인할 수 있도록 적절한 UI를 제공해야 합니다.

- Artik motion sensor에서 신호 받기

Modeola는 모션 센서에서 신호가 오면 문제풀이 화면을 띄우도록 했습니다. 따라서 Artik과 연결한 이후, 센서에서 정상적으로 신호를 받아 작동하는지를 확인해야 합니다.

C) 문제출제

- 문제 출제

Modeola에서 저희는 크게 두 가지 문제 유형을 기획했는데, OX퀴즈와 5지선다 문제입니다. 문제 출제 방식은 데이터베이스에 저장된 일정을 읽어 온 이후, 여러 가지 항목을 섞는 것입니다. 항목이란 일시, 일정을 함께하는 사람, 장소, 일정 내용 등 유저가 입력한 내용들을 말합니다.

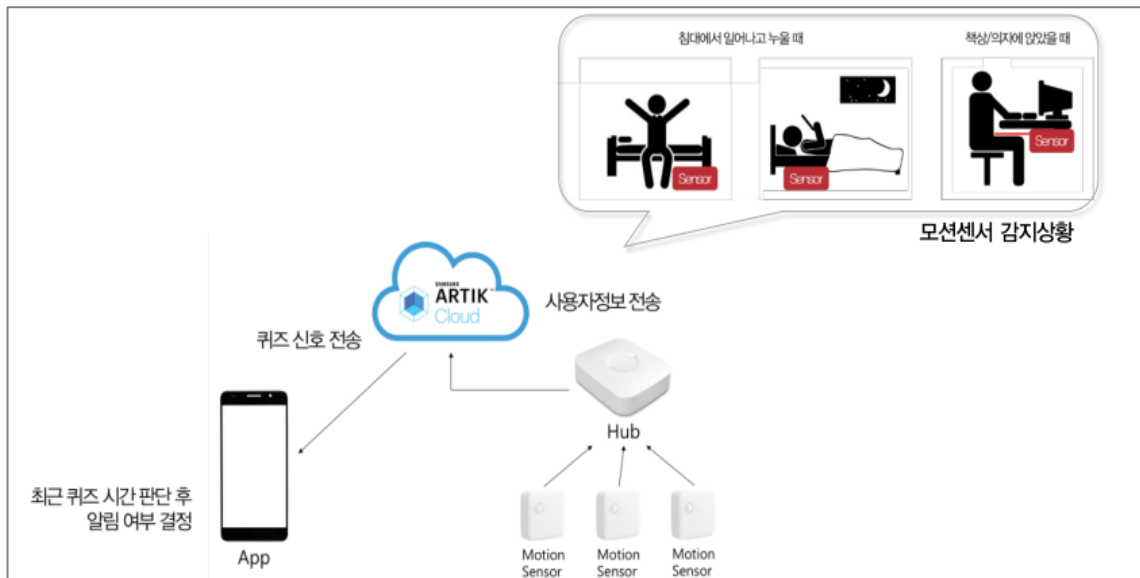
- 피드백

문제를 풀었다면 결과를 봐야 합니다. 따라서 각 문제가 틀렸는지 맞았는지는 결과 창 역시 따로 제작해야 합니다.

2. 과제 수행 내용

1) 전체 구현 계획

전체적인 데이터의 흐름은 다음과 같습니다.



앱 내부적으로 있는 기능은 크게 4가지 부문으로 나눌 수 있는데, 바로 데이터베이스, Artik, 달력, 문제풀이 등입니다. 따라서 각 기능의 기초적인 부분들을 따로 개발한 후에 하나의 앱에 합치는 방식으로 구현했습니다.

단, 4가지 기능 중 달력과 DB는 상호의존적이고, 문제풀이는 DB에 의존적입니다. 달력은 DB에서 데이터를 받아 표시해야 하며, DB는 모든 데이터 입력을 달력을 통해 받습니다. 마찬가지로 문제 출제 역시 DB에서 정보를 받아와야 제대로 작동할 수 있습니다. 따라서 기본적인 데이터베이스 구축이 가장 우선순위에 있는 과제였습니다. 이후 달력과 DB를 가장 먼저 합쳐서 함께 개발을 진행했고, 추후 Artik과 문제출제 등이 병합되었습니다.

스케줄러 앱인 만큼 달력 화면을 사용자는 가장 많이, 자주, 기본적으로 보게 됩니다. 다만, 앱을 처음 실행했을 때 Artik에 연결되어 있지 않다면 'Connect to Artik'이 먼저 뜨도록 했습니다. 메뉴 간 이동은 좌측의 navigation view를 통하도록 하였습니다. 자세한 내용은 이후의 기능별 구현 상세에서 설명하도록 하겠습니다.

2) 구현 상세

2-1) DB

안드로이드 스튜디오에서 사용하는 DB 시스템은 SQLite입니다. 안드로이드 스튜디오에서 SQLite를 사용하기 위한 핵심이 되는 요소는 SQLiteDatabase 클래스입니다. SQLiteDatabase 클래스는 데이터베이스 내에 데이터의 추가, 삭제, 수정, 조회의 역할을 해 줍니다.

'모더라' 앱은 일정의 전체 날짜(년, 월, 일), 시간(시), 장소, 만날 사람, 해야 할 일, 가중치를 중요 속성으로 가집니다. 이를 바탕으로 다음과 같이 ScheduleTBL 테이블의 속성을 정의하였습니다.

id	년	월	일	시간	장소	만날 사람	할일	가중치

〈모더라 앱의 ScheduleTBL 테이블 속성〉

id는 primary key로 가진 중복될 수 없는 속성입니다. 그리고, 날짜의 경우, SQLite는 Date 자료형이 없어서 전체 날짜를 한번에 저장할 수 없었기 때문에, 위와 같이 년, 월, 일로 속성을 쪼개서 테이블에 적용했습니다.

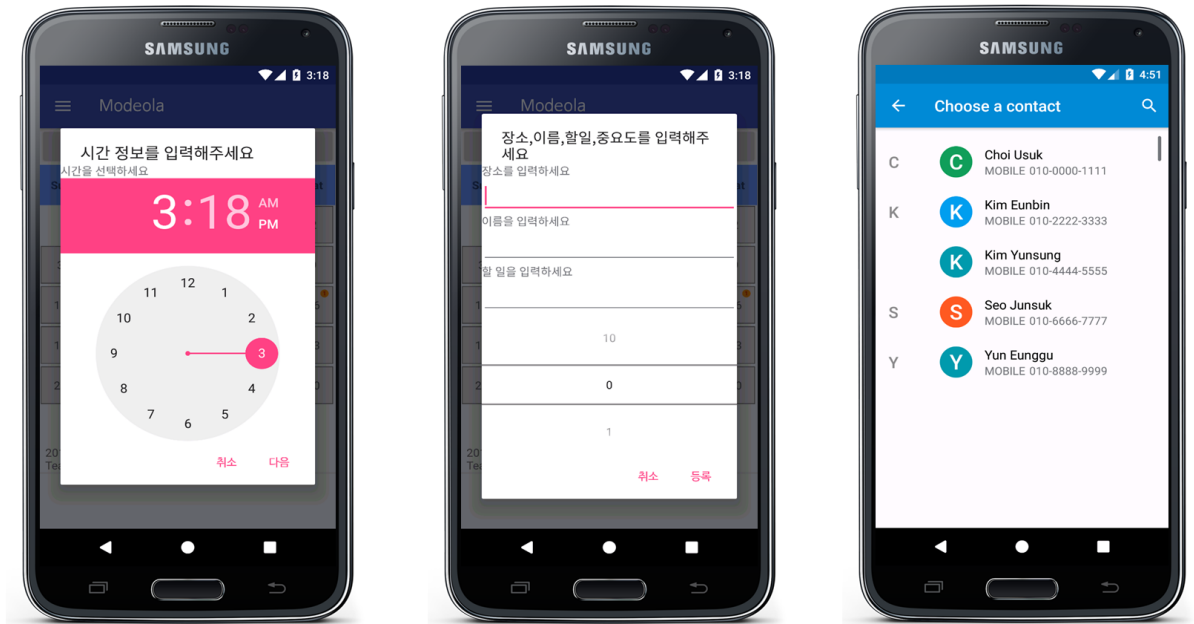
첫 달력 UI에서 날짜를 선택하면, Select 문을 사용하여 그 날 있는 일정을 한 문장으로 리스트로 나열합니다. Select로 일정 리스트를 가져오면, Dialog 위젯을 사용해 출력합니다.



```
SELECT COUNT (*) FROM scheduleTBL where
Year = "+year+" and Month = "+(month+1)+" and Day = "+day+";
```

〈Select 문을 활용한 일정 리스트 출력〉

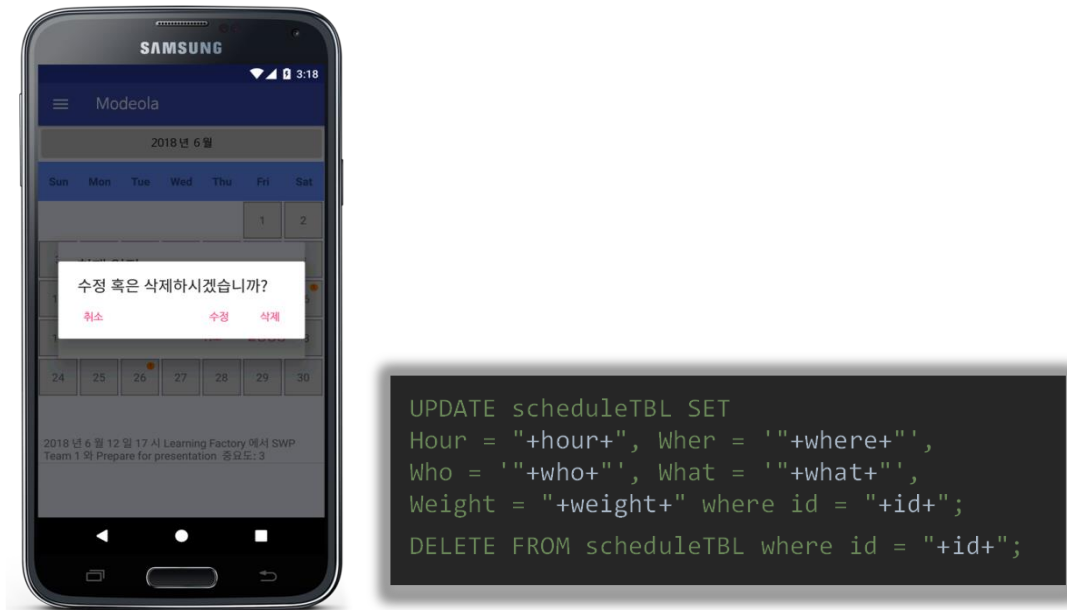
위 사진에서 하단의 '일정생성' 버튼을 누르면 일정을 생성할 수 있습니다. 날짜 데이터는 미리 Select문에서 선택한 날짜를 받고, TimePicker 위젯을 통해 일정 시간을 입력받습니다. 그리고, 장소, 이름, 할 일을 입력 받고, 중요도를 1에서 10까지 선택합니다. 또한, 이름 란을 2번 터치하면, 연락처 데이터를 가져와 만날 사람을 간단히 선택할 수 있습니다. 이렇게 입력한 데이터는 Insert문을 사용해 테이블에 데이터를 저장합니다.



```
INSERT INTO scheduleTBL (Year,Month,Day,Hour,Where,Who,What,Weight) VALUES(?,?,?, ?, ?, ?, ?, ?);
```

<Insert문을 활용한 일정생성 및 등록>

일정 리스트에서 일정을 하나 터치하면, 그 일정을 수정할지, 삭제할지 선택할 수 있습니다. 수정 버튼을 클릭하면, '일정생성' 버튼을 눌렀을 때와 같이 Dialog 화면이 나오고, 처음부터 다시 설정 가능합니다. 최종적으로 수정을 완료하면, Update 문을 사용해 그 데이터를 새로 설정한 일정으로 덮어씹습니다.



〈Update문을 활용한 일정 수정〉

삭제 버튼을 누르면, Delete 문을 사용하여 간단하게 일정을 지울 수 있습니다.

```
DELETE FROM scheduleTBL where id = "+id+";
```

〈Delete문을 활용한 일정 삭제〉


2-2) ARTIK

저희 프로젝트에서 Artik은 모션 센서의 데이터를 앱이 사용할 수 있게 하여, 앱이 사용자가 접근했는지 판단할 수 있게 해주는 역할을 하고 있습니다. 이를 기능을 구현하기 위해서는 모션 센서에 반응이 있을 때마다 Artik에서 앱에 푸시 알림을 주는 방법이 제일 효율적이라고 생각하였습니다. 하지만 푸시 알림을 위해서는 FCM(Firebase Cloud Messaging)과 같은 서비스를 이용하는 등의 복잡한 과정이 추가되기 때문에, 다소 비효율적이지만 앱에서 백그라운드 서비스를 돌려서 20초에 한번씩 Artik에 연결해 모션 센서가 반응했는지 여부를 확인하게 했습니다.

Artik 기능의 빠른 구현을 위해 Artik 개발자 사이트에서 제공하는 샘플 코드에서 앱이 필요로 하는 부분을 추출하여 활용하였습니다.

가. Artik에 앱 등록

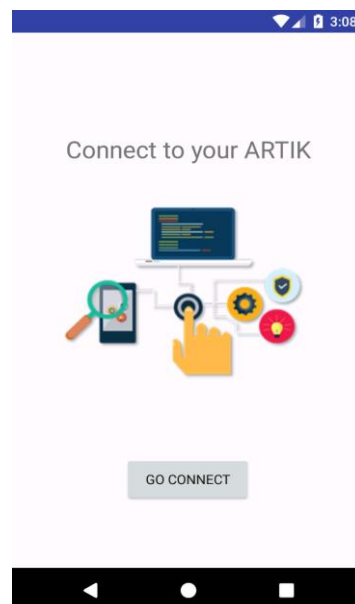
앱이 Artik에 연결하기 위해서 먼저 Artik 개발자 계정으로 앱을 등록하였습니다. 앱은 모션 센서 하나를 사용하는 권한을 갖도록 했습니다.

 Modeola Test Todo Scheduler EDIT APP 3 ERRORS		
MESSAGES/HR 0.012	API CALLS/HR 2.07	ACTIONS/HR 2.05
NOTIFICATIONS/HR 0		ERRORS/HR 0.004

〈Artik 에 등록된 앱 정보〉

나. Artik연결 화면

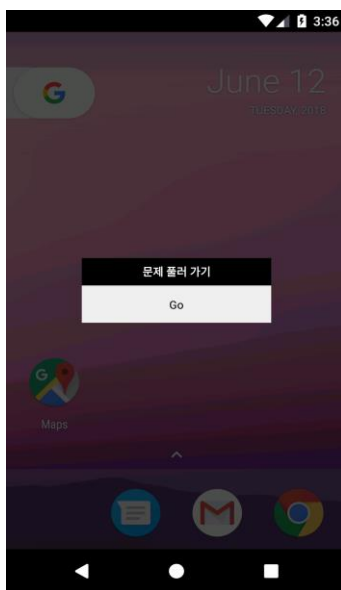
앱이 처음 실행되면 사용자가 앱을 Artik에 연결했는지 확인하여, 연결 되어있지 않다면 Artik에 연결하라는 메시지를 띄웁니다. 사용자가 이 과정을 한번만 수행하면, 이후에는 Artik에서 받아온 액세스 토큰을 이용해서 자동으로 Artik에 연결하게 됩니다.



〈Artik 연결 화면〉

다. 백그라운드 서비스

앱이 Artik에 연결되고 나면, 앱은 백그라운드 서비스를 시작하여 Artik에 새로운 센서 정보가 있는지 확인합니다. 모션 센서가 반응을 했다면, 팝업 메시지를 띄워서 사용자가 문제를 풀게 합니다. 이때 모션 센서의 민감도와 앱이 사용자에게 문제를 풀게 할 상황을 감안하여, 센서가 반응하고 난 이후 특정 시간 간격(2시간)이 지나기 전에는 센서의 신호를 무시하도록 했습니다. 사용자가 임의로 팝업창을 종료시키면 안되기 때문에, 뒤로 가기 등 이외의 버튼은 누를 수 없도록 설정하였습니다.



〈문제 풀기 팝업〉

```
@Override
public void run() {
    while (true) {
        getLatestMsg();

        Log.v(TAG, msg: ":: Prev Msg Time = " + prevMsgTime);
        Log.v(TAG, msg: ":: Latest Msg Time = " + latestMsgTime);
        Log.v(TAG, msg: ":: Msg Time Interval = " + (latestMsgTime - prevMsgTime) / 1000 + "second");

        if (latestMsgTime == 0) {
            Log.w(TAG, msg: "Auth Failed");
        } else if (prevMsgTime == 0) {
            prevMsgTime = latestMsgTime;
        } else if (latestMsgTime - prevMsgTime > NotificationInterval) {
            prevMsgTime = latestMsgTime;
            Intent popup = new Intent(getApplicationContext(), PopupActivity.class);
            popup.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
            startActivity(popup);
        }

        try {
            Thread.sleep(refreshInterval);
        } catch (InterruptedException exc) {
            exc.printStackTrace();
        }
    }
}
```

〈백그라운드 서비스에서 주기적으로 Artik 을 확인하는 코드〉

원래는 제공받은 Samsung Smart Things의 모션 센서를 실제로 활용하여 앱을 구동하려고 하였지만, Smart Things 허브가 Artik에 모션 센서의 데이터를 제공하지 못하는 문제가 발생하였습니다. 이를 해결하기 위해 고객센터에 문의해 보기도 했지만 결국 문제의 원인과 해결방안을 찾지 못했고, 어쩔 수 없이 Artik에서 제공하는 가상센서를 이용하여 앱을 테스트하게 되었습니다.

2-3) Calendar

개발 초기 단계에서는 달력이 보조적인 인터페이스에 불과하다고 생각되었으나, 직접 개발 해본 결과 DB의 정보 입출력을 모두 담당하는 사용자 UI라는 측면에서 중심적인 역할을 담당하게 되었습니다.

달력에서 필요한 기능은 다음과 같습니다.

1. 사용자가 일정을 관리하기 쉽도록 달력을 표시해 준다.
2. 각 날짜를 클릭했을 때 날짜에 맞는 일정이 표시된다.
3. 일정을 추가/수정/삭제할 수 있도록 적절한 창을 제공한다.
4. 일정이 있는 날을 쉽게 알 수 있게 점, 선 등을 각 날짜 칸에 표시한다.

사실 모두 기존의 캘린더 앱에서 제공하는 기능입니다. 따라서 저희는 Flexible Calendar라는 오픈 소스 소프트웨어를 참조해 사용했으며, 저희 프로젝트에 맞게 적절히 변형해서 사용하였습니다. 참고로 Flexible Calendar의 라이선스는 MIT입니다.

가. 달력 표시

기존 Flexible Calendar에서는 몇 가지의 달력 인터페이스를 제공합니다. 원래는 검은 배경에 숫자만 표시되는 달력을 사용할 계획이었으나, 모더라의 전체적인 UI 컨셉의 통일을 위해 흰색 바탕의 파란색 UI로 재구성하였습니다.



<기존 달력 UI>



<수정된 달력 UI>

나. 날짜 클릭 시 일정 표시

Modeola에서 날짜를 클릭하면 화면 중앙의 dialog와 하단의 listview에서 동시에 해당 날짜의 일정을 표시해 주게 됩니다. 이와 관련된 내용은 database 파트에서 자세히 다뤘으므로 따로 설명하지 않겠습니다.

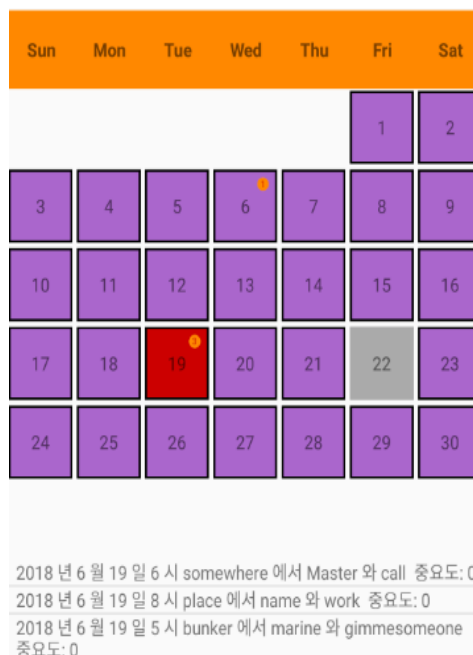
다. 일정 관리

마찬가지로 일정 추가/수정/삭제에 대한 부분은 database와 연동되는 부분이기 때문에 다시 설명하지 않겠습니다.

라. 달력에 일정 표시

달력에 일정을 표시하는 코드는 두 가지 객체를 사용해서 구현됩니다. 바로 CustomEvent 리스트와 eventMap입니다. 일정이 여러 개 있으면 List<CustomEvent>에 Event를 추가하고, 이 리스트를 다시 eventMap 으로 달력상의 적절한 위치에 매핑해주는 형식입니다. 예를 들어 27일에 두 개의 일정이 있다면 리스트에 CustomEvent를 두 개 추가하고, eventMap의 put함수로 27일에 리스트를 연결하는 것입니다.

달력에 일정을 채워야 할 때는 처음 앱을 실행했을 때, 다른 달로 달력을 넘겼을 때, 일정의 수정 사항이 생겼을 때 등입니다. 따라서 그때마다 fillEvents() 함수를 호출되게 구현했습니다. 이벤트 리스트는 ArrayList[31]로 총 서른한 개를 선언하며, 함수가 호출될 때마다 들어있는 내용을 모두 clear 합니다. 이후, 반복문을 통해 데이터베이스에서 각 연, 월, 일에 해당하는 일정을 조회합니다. 각 날짜마다 일정의 수를 받아온 후, 그 수만큼 CustomEvent 추가, 이후 eventMap에 연결하는 것이 이 기능의 핵심이라 볼 수 있습니다. 추가된 일정의 수는 각 날마다 표시되는 점의 숫자에서 확인할 수 있습니다.



<개발 과정에 있었던 앱 화면. 일정이 3개일 때 숫자 3이 점에 표시된다.>

2-4) 문제풀이

문제 출제 알고리즘

사용자가 문제 풀기 팝업 버튼을 클릭하면 앱은 그때부터 Database에서 스케줄을 받아서 문제를 만들기 시작합니다. 문제는 OX 문제 3개와 5지선다 문제 2개로 이루어져 있습니다.

가. 스케줄 추출

문제를 출제하는 `getProblems()` 함수는 먼저 `getScheduleForProblem()` 함수를 이용하여 문제 출제에 사용할 스케줄을 가져옵니다. `getScheduleForProblem()` 함수는 오늘 이후에 있는 스케줄들을 Database에서 가져온 후 스케줄의 우선순위를 기준으로 정렬을 합니다. 좋은 문제 출제를 위해서는 적절한 우선순위 계산식이 있어야겠지만, 프로젝트 시간 관계상 사용자가 직접 설정한 스케줄의 `weight`를 기준으로 우선순위를 계산했습니다. 스케줄들을 정렬한 후에는 우선순위가 높은 순서대로 문제 출제에 필요한 스케줄의 개수 (OX : 2개, 5지선다 : 5개, 총 16개)만큼 스케줄들을 반환합니다. 프로젝트 시간 관계상 새로운 스케줄을 자동으로 만드는 기능을 구현하지 못하여 Database에 저장된 스케줄이 부족한 경우에는 작동하지 못하는 한계점을 가지고 있습니다.

```
static Comparator<Schedule> priorityComparator = new Comparator<Schedule>(){
    public int compare(Schedule schedule1, Schedule schedule2){
        // Descending order
        return calculatePriority(schedule2) - calculatePriority(schedule1);
    }
    public int calculatePriority(Schedule schedule) {
        // Priority principle
        // Priority == Weight of schedule
        return schedule.getWeight();
    }
};
```

〈스케줄의 우선순위 계산〉

나. 문제 출제

`getProblems()` 함수는 이후 받아온 스케줄들을 바탕으로 문제를 출제합니다. 먼저 `shuffle()` 함수를 이용하여 받아온 스케줄들을 섞습니다. 그 후 각 문제에 필요한 스케줄의 개수만큼 차례대로 출제에 사용됩니다.

OX 문제를 출제할 때는 스케줄 두 개를 사용합니다. 첫 번째 스케줄을 기준으로, 이 스케줄에 대한 답이 O가 되어야 하는지 X가 되어야 하는지를 결정합니다. O라면 그대로 문

제 내용으로 사용하고, X라면 스케줄의 네 가지 속성(언제, 어디서, 누구와, 무엇을) 중에서 하나를 선택하여 두번째 스케줄의 내용과 서로 바꿔서 문제 내용으로 사용합니다.

5지선다 문제를 출제할 때는 스케줄 다섯 개를 사용합니다. 우선 스케줄의 네 가지 속성(언제, 어디서, 누구와, 무엇을) 중에서 서로 섞을 속성 하나를 선택합니다. 그 후 다섯 번째 스케줄은 그대로 두고 나머지 네 개 스케줄들을 선택한 속성에 대해 서로 섞습니다. 다음으로 정답이 될 번호를 선택하여 다섯 번째 스케줄과 해당 번호의 스케줄을 교환하여 위치를 맞춥니다.

```
if(answers.get(problemCur) == 1) {
    int property = random.nextInt( bound: 4);
    if(property == 0) { // When
        schedules.get(scheduleBaseCur).setYear(schedules.get(scheduleBaseCur+1).getYear());
        schedules.get(scheduleBaseCur).setMonth(schedules.get(scheduleBaseCur+1).getMonth());
        schedules.get(scheduleBaseCur).setDay(schedules.get(scheduleBaseCur+1).getDay());
        schedules.get(scheduleBaseCur).setHour(schedules.get(scheduleBaseCur+1).getHour());
    } else if(property == 1) { // Where
        schedules.get(scheduleBaseCur).setWhere(schedules.get(scheduleBaseCur+1).getWhere());
    } else if(property == 2) { // Who
        schedules.get(scheduleBaseCur).setWho(schedules.get(scheduleBaseCur+1).getWho());
    } else if(property == 3) { // What
        schedules.get(scheduleBaseCur).setWhat(schedules.get(scheduleBaseCur+1).getWhat());
    }
}
```

<OX 문제 출제>

```
if(property == 0) { // When
    int[][] time = new int[4][4];
    for(int j = 0; j < 4; j++) {
        time[j][0] = schedules.get(scheduleBaseCur+j).getYear();
        time[j][1] = schedules.get(scheduleBaseCur+j).getMonth();
        time[j][2] = schedules.get(scheduleBaseCur+j).getDay();
        time[j][3] = schedules.get(scheduleBaseCur+j).getHour();
    }
    for(int j = 0; j < 4; j++) {
        schedules.get(scheduleBaseCur+j).setYear(time[contentShuffle.get(j)][0]);
        schedules.get(scheduleBaseCur+j).setMonth(time[contentShuffle.get(j)][1]);
        schedules.get(scheduleBaseCur+j).setDay(time[contentShuffle.get(j)][2]);
        schedules.get(scheduleBaseCur+j).setHour(time[contentShuffle.get(j)][3]);
    }
} else if(property == 1) { // Where
    String[] where = new String[4];
    for(int j = 0; j < 4; j++)
        where[j] = schedules.get(scheduleBaseCur+j).getWhere();
    for(int j = 0; j < 4; j++)
        schedules.get(scheduleBaseCur+j).setWhere(where[contentShuffle.get(j)]);
} else if(property == 2) { // Who
    String[] who = new String[4];
    for(int j = 0; j < 4; j++)
        who[j] = schedules.get(scheduleBaseCur+j).getWho();
    for(int j = 0; j < 4; j++)
        schedules.get(scheduleBaseCur+j).setWho(who[contentShuffle.get(j)]);
} else if(property == 3) { // What
    String[] what = new String[4];
    for(int j = 0; j < 4; j++)
        what[j] = schedules.get(scheduleBaseCur+j).getWhat();
    for(int j = 0; j < 4; j++)
        schedules.get(scheduleBaseCur+j).setWhat(what[contentShuffle.get(j)]);
}
```

<5 지선다 문제 출제>

다. 문제 저장

getProblems() 함수에서 문제 출제가 끝나면, 이 함수는 문제들에 대한 정보가 담긴 Problems 객체를 반환합니다. Problems 클래스는 Problem 객체 5개를 포함하고 있으며 Problem 서브클래스는 문제로 낼 스케줄에 대한 정보, 문제로 낸 스케줄의 원래 정보, 문제의 답, 문제의 종류를 저장합니다.

```
public class Problems {
    private Problem[] problems;

    Problems(List<String> contents,
             problems = new Problem[5];
```

<Problems 클래스>

```
public class Problem {
    private String[] content;
    private String[] contentOriginal;
    private int answer;
    private int type;
```

<Problem 서브클래스>

라. 문제 풀이 답안 제공

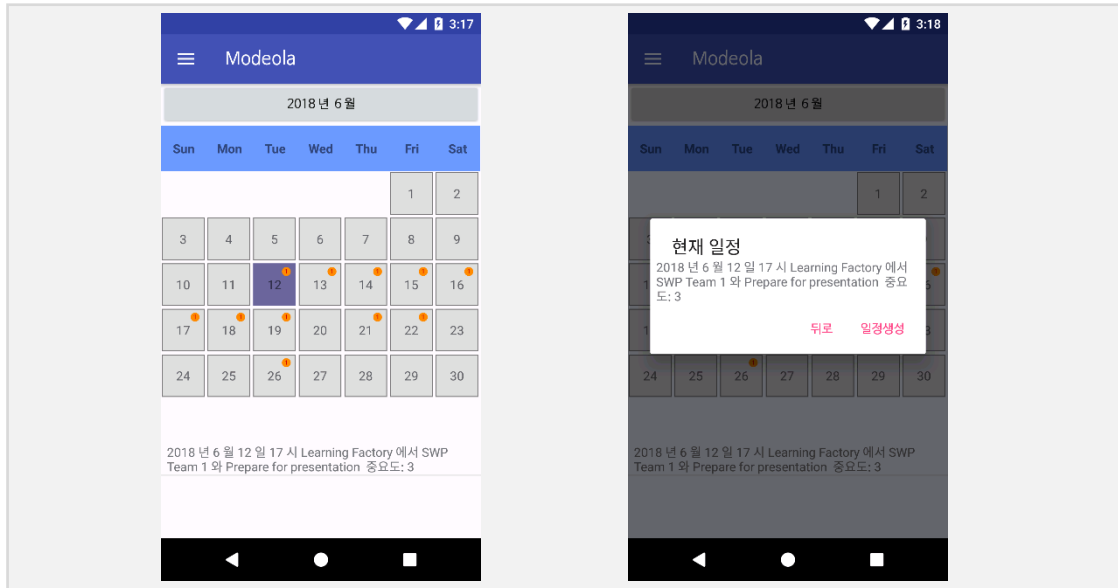
사용자가 답안을 다 작성해 제출하거나, 제한시간이 다 되면 시스템은 채점을 하고 결과를 나타내는 액티비티를 띄웁니다. 사용자가 총 몇 문제를 맞혔는지, 그리고 해당 문제의 답안을 알려주어 틀린 일정에 대해 다시 상기할 수 있도록 피드백을 제공합니다. 사용자는 이를 통해 자신이 어떤 일정을 까먹고 있었고, 혹은 전체 스케줄을 얼마나 숙지하고 있는지 되새겨 볼 수 있습니다.

3. 과제 수행 결과

1) 최종 개발 결과

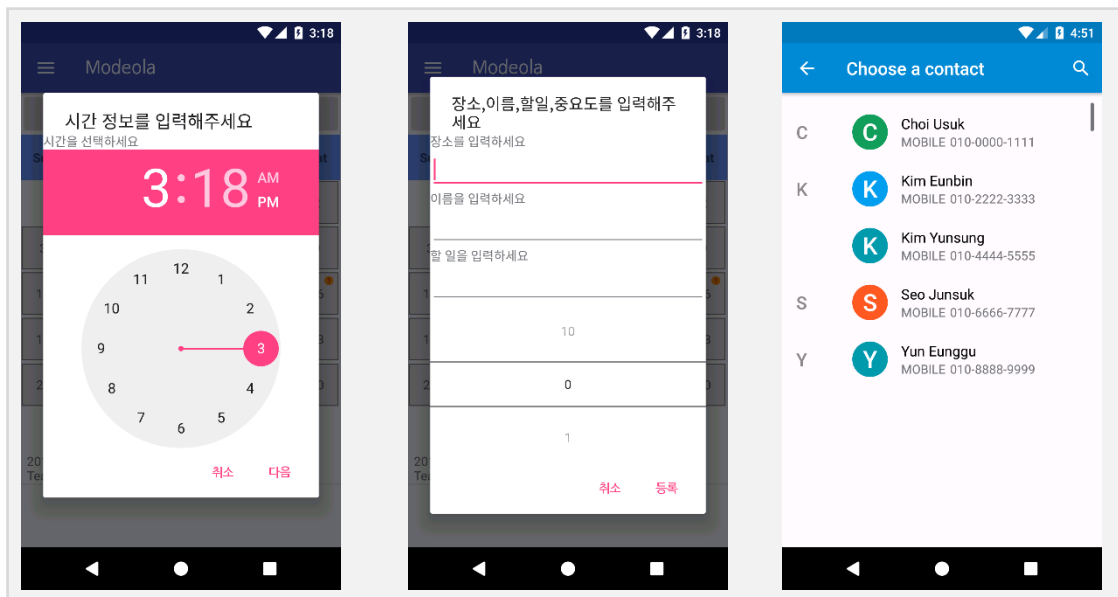
1-1) 실행 첫 화면

어플리케이션을 처음 실행했을 때의 화면은 스케줄러라는 특성에 알맞게 달력화면으로 구성하였습니다. 사용자는 날짜를 클릭하여 등록된 스케줄을 확인할 수 있고, '일정생성' 버튼을 통해 새로운 일정을 등록하는 것도 가능합니다.



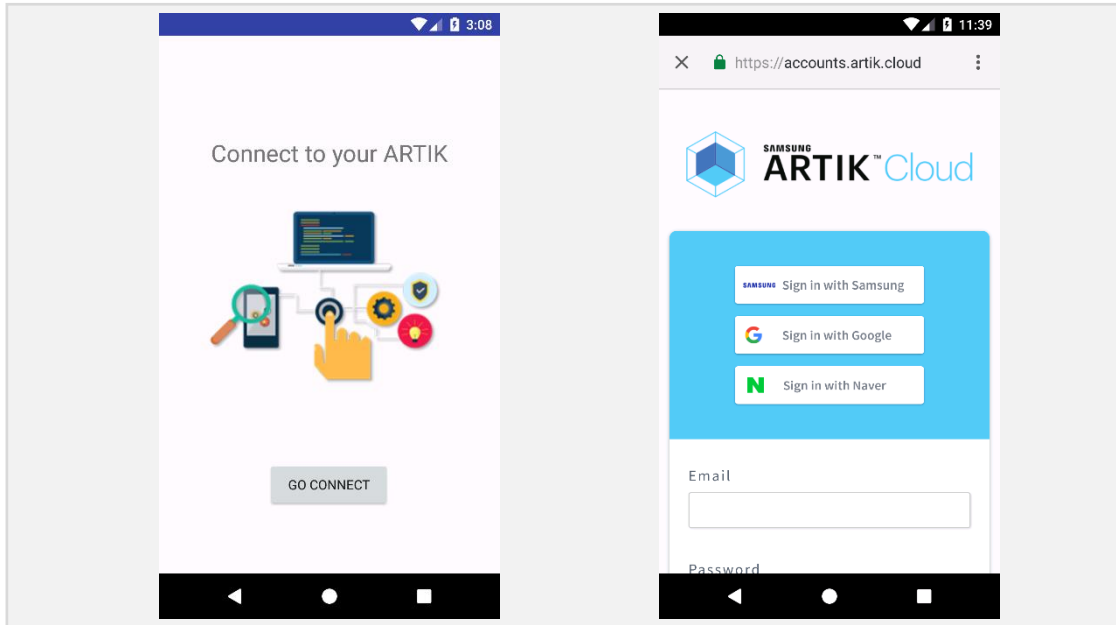
1-2) 일정 생성

일정을 등록할 때에는 먼저 세부 시간을 위젯을 사용하여 간편하게 설정합니다. 그 후, 만날 장소, 만나는 사람의 이름, 할 일, 그리고 가중치를 입력합니다. 만날 장소와 할 일은 키보드로, 가중치는 스크롤, 만나는 사람은 연락처를 이용해 설정 가능합니다.



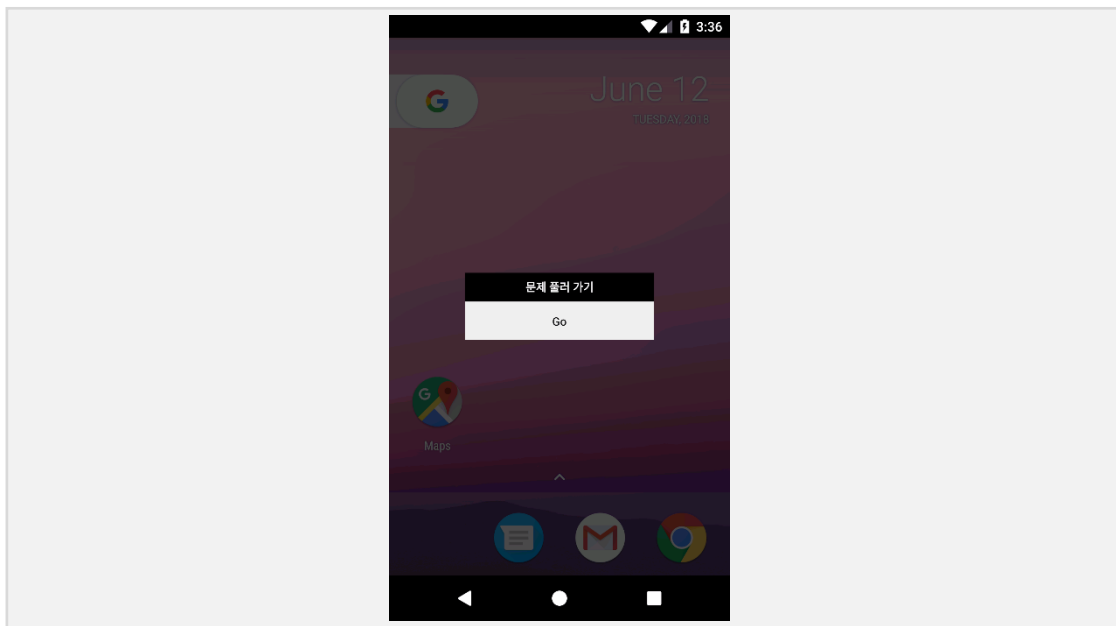
1-3) ARTIK 연결

어플은 모션 센서를 활용해 동작하기 때문에 ARTIK 클라우드에 연결되어 있어야 합니다. 이에 따라 어플 처음 실행 시, 디바이스가 클라우드에 연결됐는지 확인하고 되어 있지 않다면 클라우드 연결 화면을 띄우게 됩니다.



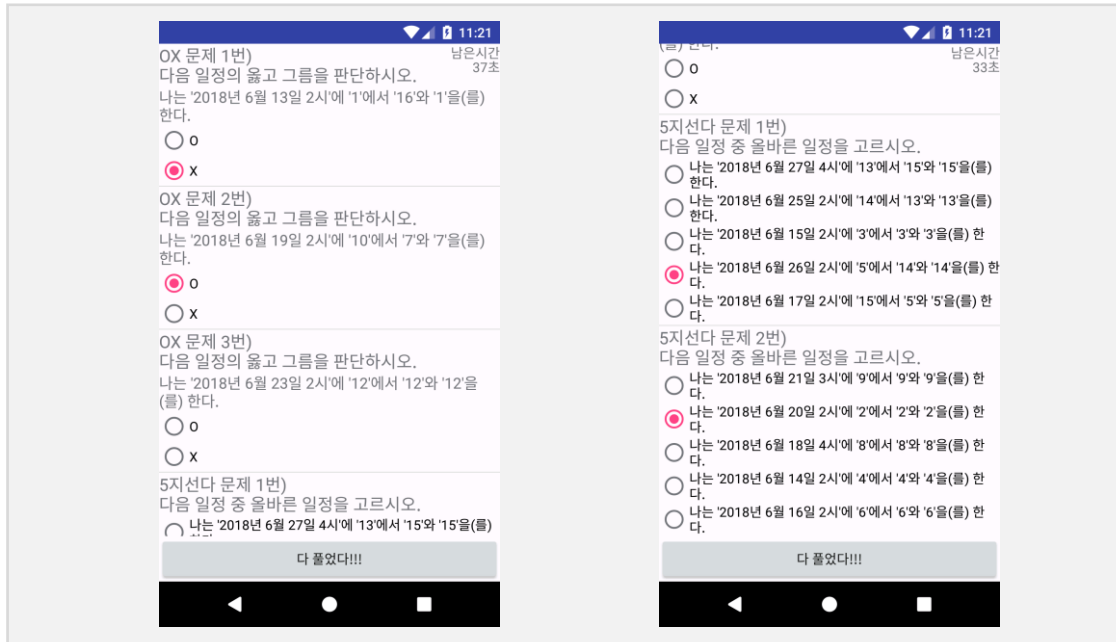
1-4) 모션 센서 인식에 따른 문제 출제

모션 센서가 반응했다는 정보를 받으면, 그 전 센서가 반응했던 시간과 비교해 충분히 오래 되었다고 판단 시, 문제 풀이 알림 창을 띄웁니다.



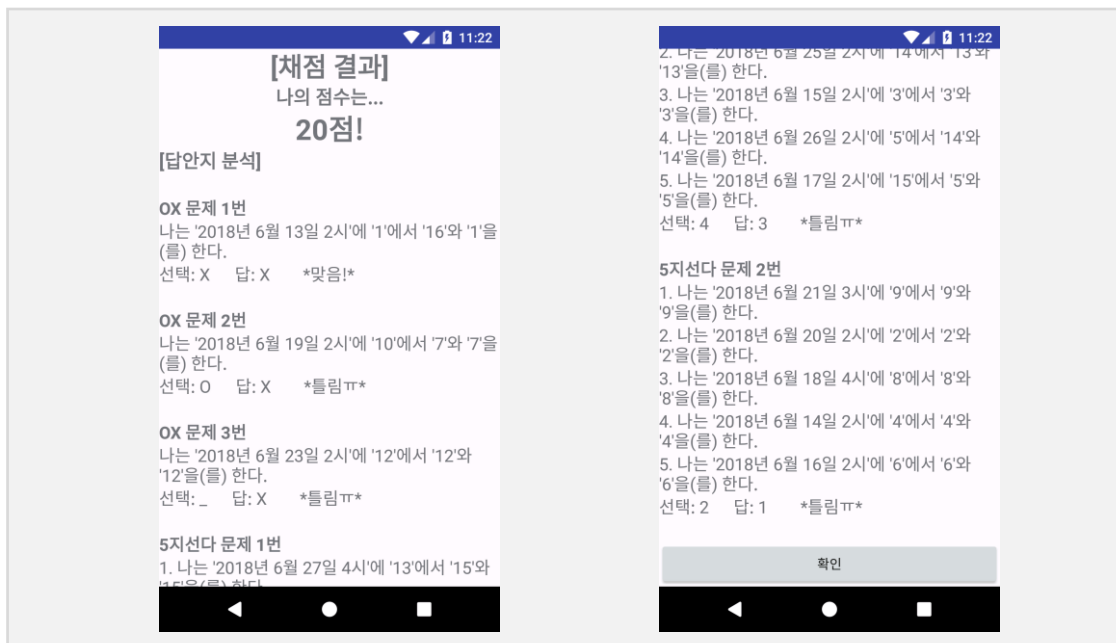
1-5) 문제풀이

Go 버튼을 누르면 사용자는 OX 문제, 5지선다 문제로 이루어진 다섯 문제를 1분안에 풀어 제출해야 합니다



1-6) 채점 결과 및 피드백

사용자가 답안을 제출하거나, 제한시간이 다 되면 시스템은 채점을 한 뒤 결과 창을 띄워 줍니다. 사용자에게 답안지 피드백을 제공함으로써 자신이 일정을 얼마나 숙지하고 있는지 되새겨 볼 수 있습니다.



2) 아쉬운 점 및 발전 가능성

2-1) 아쉬운 점

첫 번째는 Smart Things의 모션 센서를 실제로 사용해보지 못한 것입니다. 처음 기획 단계에서 강조했던 ‘모더라’의 장점 중 한가지는 IoT 디바이스를 이용해 사용자가 집중할 수 있는 시간, 동시에 일정의 상기가 필요한 시간을 인식하여 알림을 준다는 것이 포함되어 있었습니다. 만약 실제 모션 센서로 시연을 해보았다면 ‘모더라’ 어플이 줄 수 있는 장점을 더욱 쉽게 인식할 수 있었을 것이라고 생각합니다. 하지만 Smart Things 허브가 ARTIK 클라우드에 모션 센서 정보를 제대로 넘겨주지 못하는 문제가 발생함으로 인해, 실제 모션 센서로 실험해보지 못하였고 가상센서를 만들어 개발할 수 밖에 없는 상황이었다는 점이 안타까웠습니다.

두 번째는 문자 전송 패널티 부분을 구현하지 못한 점입니다. 사용자가 일정을 잘못 기억하고 있었거나 잊어버리고 있었을 시 제공되기로 했었던 문자 전송 패널티는 ‘모더라’의 특별한 아이디어 중 하나였습니다. 하지만 시연 및 구현에 있어 문자 전송을 위해서는 연결 가능한 USIM이 필요하거나, 두 개의 에뮬레이터가 필요함 등의 한계점이 있었기 때문에 구현하지 못하였습니다. 실제 해당 부분이 개발되었다면, 사용자에게 보다 높은 집중력을 유도하고, 어플 사용에 있어 게임과 같은 재미를 부여할 수 있었을 것이라 판단됩니다.

2-2) 발전 가능성

패널티 부여의 보완책으로 답안 피드백 제공 기능을 만들었습니다. 이번 프로젝트에서 개발에서는 각각의 문제 풀이에 대한 답안 피드백을 제공하였지만, 만약 각 결과들을 누적해서 사용자의 총 문제 풀이 능력을 측정하고, 이를 관리할 수 있는 서브 시스템을 구현한다면, 전체적인 일정에 대한 숙지도를 더욱 객관적으로 파악 가능할 것이라고 생각합니다. 당장 지금 푼 문제는 상황에 따라 결과가 좌지우지될 수 있다는 한계를 가지나, 누적 결과를 제공한다면 좀 더 신뢰 가능한 일정 숙지 정도를 판단 가능할 것으로 예상됩니다.

3) 기대 효과

기존 스케줄러 어플과는 차별화된 시스템으로 ‘모더라’는 사용자의 장기적 일정 상기를 도와줄 수 있습니다. 단순히 일정을 기입하고, 직접 확인하는 시스템에서, 모션 센서를 통해 알맞은 때를 확인하고, 사용자의 일정을 좀 더 재미있는, 그리고 집중시킬 수 있는 문제풀이의 방법으로 상기시켜줍니다.

또한 사용자에게 편리함을 제공할 수 있습니다. 항상 스케줄러는 일정을 기입하고 확인할 수 있는 플랫폼에 지나지 않았습니다. 하지만 스케줄러 스스로 사용자의 일정 상기를 도와준다는 점에서 사용자가 굳이 일정을 확인해보지 않아도 된다는 편리함을 제공합니다.

마지막으로 소소한 일상의 재미를 제공할 것입니다. 가끔 일정은 부담으로만 느껴지기도 합니다. 일정이 쌓여 있는 스케줄러를 확인하면 갑갑할 때가 있습니다. 그러나 ‘모더라’는 이를 퀴즈 형식으로 풀어내며 사용자에게 좀 더 편하게 다가갈 수 있습니다.