

PROJECT 최종 보고서

Be Attention

- 집중해조 -

2016104095 강세희

2016104109 김성수

2018102694 서예진

2017101614 이진아

2018102243 최용욱

<https://github.com/KHU-Rainbow>

김재홍 교수님

Table of Contents

1. Abstract
 2. Introduction
 3. Background Study
 - A. 관련 접근방법/기술 장단점 분석
 - B. 프로젝트 개발환경
 4. Goal/Problem & Requirements
 5. Approach
 6. Project Architecture
 - A. Architecture Diagram
 - B. Architecture Description
 7. Implementation Spec
 - A. Input/Output Interface
 - B. Inter Module Communication Interface
 - C. Modules
 8. Solution
 - A. Implementations Details
 - B. Implementations Issues
 9. Result
 - A. Experiments
 - B. Result Analysis and Discussion
 10. Division & Assignment of Work
 11. Conclusion
- ◆ [Appendix] User Manual

1. Abstract

'RAINBOW 서비스'는 책상 앞에서 시간을 투자하며 자신의 꿈을 조금씩 키워나가는 모든 이들을 위한 서비스이다. 사용자는 ①실제 공부시간 측정 ②목표 공부시간 관리 ③핸드폰 사용 여부 모니터링 기능을 제공하는 RAINBOW 서비스를 통해 공부시간을 보다 효율적으로 관리할 수 있다.

2. Introduction

현재 Google PlayStore에 배포된 기존의 공부 관리 어플리케이션인 <Forest>를 살펴보면 사용자가 집중하고자 하는 시간에 Application을 실행해 공부를 시작하고, 끝날 때 다시 Application을 실행하여 공부를 끝내는 번거로운 입력 방식이다. 이번 프로젝트를 통해 개발하려고 하는 RAINBOW 서비스는 사용자가 집중 시간을 직접 입력하지 않고, IoT Device에서 자체적으로 측정함으로써 사용자에게 공부 이외의 부담을 최소화하는 서비스를 제공하는 것을 목표로 한다.

RAINBOW 서비스는 사용자가 '책상에 앉아 있을 때' 공부를 하고 있는 것으로 인지하고 이 시점을 기준으로 시간을 측정한다. 자세 확인은 '초음파 센서'를 활용하여 사용자가 임계 거리 이상으로 가까워지면 앉은 것으로, 멀어지면 일어난 것으로 판단한다. 이때 사용자는 Buzzer 모듈을 통해 시간이 측정되고 있는지 여부를 확인할 수 있다.

RAINBOW 서비스는 핸드폰 사용이 감지되면 알림을 통해 사용자가 다시 집중할 수 있도록 도와주는 기능도 제공한다. 일반적으로 책상에 앉아 공부할 때 집중이 흐트러지는 가장 큰 원인이 핸드폰 사용이기 때문이다. 이를 구현하기 위한 방법은 다음과 같다. 먼저 라즈베리파이에 네트워크 어댑터를 USB 연결하여 패킷 모니터링 모드로 전환한다. 이때 핸드폰에서 웹서핑을 시도하거나 게임을 시도하는 패킷이 잡히는 경우, Buzzer에서 소리가 나 사용자가 공부에 다시 집중할 수 있도록 돋는 방식이다.

RAINBOW 서비스는 Android Application을 통해 사용자와 직접적으로 상호작용한다. 특히 UI를 통해 자신의 공부 현황을 한 눈에 확인할 수 있고, 매일 목표 공부시간을 채우면 받을 수 있는 보상을 통해 성취감을 고취시킨다. 사용자는 아침에 하루 목표 공부시간을 Application에 등록하고, 무지개의 일곱 색깔과 일주일의 7일을 대응해서 당일 목표를 달성한 경우 배지를 받는다. ①당일 측정된 실제 공부시간 ②핸드폰 사용 횟수는 다음날 00:00분에 AWS Cloud를 이용한 Server에서, 사용자의 Application에 Push 메시지 형식으로 전달된다. 만약 일주일 모두 목표를 달성할 경우에는 종합 평가에서 무지개를, 4~6일을 달성할 경우엔 반 무지개, 3일 이하를 달성할 경우 구름 이미지가 등록된다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

① 필요 목적 외 핸드폰 사용 방지

공부나 일을 할 때 가장 방해가 되는 것은 핸드폰이다. 하지만 핸드폰을 사용해서 전화를 하거나 급한 연락을 받을 수 있다. 그렇기 때문에 의미 없는 핸드폰 사용만을 감지할 수 있는 방법에는 무엇이 있는지 알아보았다. 관련 자료를 찾아보면서 라즈베리파이에 네트워크 어댑터를 연결하여 Monitor Mode로 네트워크 패킷을 캡처할 수 있는 기능을 알게 되었다. 라즈베리파이를 이용한다면 네트워크 접속 패킷, 게임을 할 때 발생하는 패킷을 감지할 수 있다. 패킷이 감지된다면 사용자에게 ‘Buzzer 센서’를 이용해 알림을 주고, 이러한 기능을 통해 필요 목적 외 핸드폰 사용을 줄일 수 있을 것이다.

② 사용자 착석여부 판단

공부를 할 때 책상에 자료들을 짹 펼쳐 놓고, 의자에 바른 자세로 앉는 것은 “나는 공부를 할 것이다!”라는 마음가짐을 다잡는 필수적인 자세이다. RAINBOW 서비스를 찾는 사람은 공부를 목적으로 하는 사람들이 대다수일 것이며, 「책상에 앉아있는 시간 = 공부를 하는 시간」이라 가정해도 좋을 것이다. 만약 RAINBOW 서비스를 시장에서 유통한다면, 가격을 낮추면서도 어렵지 않게 착석여부를 확인하기 위한 방법이 무엇이 있을지 고민해봤다. 먼저 ‘초음파 센서’를 책상 아래쪽에 비치한 후 무릎과의 거리를 측정한다. 이때 앉은 자세를 구별할 수 있는 임계값을 잘 설정해준다면, 착석여부를 어렵지 않게 판단할 수 있을 것이다.

③ 다양한 알림 방법

2020년을 살아가는 현대인들에게는 핸드폰을 확인하는 것이 ‘습관’에 가까울 정도로 빈번하게 발생하는 일이다. 따라서 사용자가 자기도 모르게 하는 행동에 대한 알림을 주는 것이 좋다고 판단했다. 라즈베리파이 디바이스를 이용해서 핸드폰 사용에 대한 알림을 주는 방법을 고민한 결과 직관적이고 빠르게 알림을 줄 수 있는 ‘Buzzer 센서’를 이용하기로 했다. 핸드폰 사용이 라즈베리파이에서 감지된다면 Buzzer 센서에서 소리가 나게 된다. 외에도 시간 측정이 이루어지고 / 이루어지지 않고 있다는 것을 알려주기 위한 ‘Buzzer 센서’, 현재 시점까지 공부 한 시간을 알려주는 ‘스탑워치’와 비슷한 기능을 ‘FND 센서’를 이용해서 구현하기로 했다.

④ 클라우드 서버 활용

2011년 하버드 비즈니스 리뷰 분석 서비스 “클라우드가 선택되는 이유”에 따르면, 비즈니스 와 기술 관련 전문가들은 비즈니스 민첩성, 유연성, 신기술 도입속도, 고정비용 감소, IT 시스템 개발 및 비용 감소, IT 업데이트 없이도 신규 버전의 애플리케이션 사용 가능 등 6개 영역

에 대해 클라우드가 상당한 가치가 있다고 답변했다. RAINBOW 서비스에서도 매일 라즈베리파이와, Application에서 발생하는 데이터를 저장해두는 서버 역할을 해주는 공간이 필요하다. 특히 Application의 내장 Storage는 크기가 제한되어있기 때문에 Application 사용 날짜가 길어진다면, 그만큼 사용자의 핸드폰에 데이터가 많이 쌓이게 될 것이다. AWS의 S3 스토리지 등 데이터 저장 서비스를 이용한다면, 적절한 비용으로 오래된 데이터 보관을 위한 기능을 제공받을 수 있을 것이다.

⑤ Application 활용

데이터를 이용해서 통계를 낸다면, 이러한 결과물을 사용자가 편하게 볼 수 있어야 한다. 이때 어디서나 쉽게 확인할 수 있는 방법에는 무엇이 있을지 생각해봤고, 핸드폰 Application 을 이용하면 좋을 것이라고 생각했다. 특히 Application UI를 통해 자신의 공부 현황을 한 눈에 확인할 수 있고, 매일 목표 공부시간을 채우면 받을 수 있는 보상을 통해 사용자에게 성취감을 주는 것으로 스스로 동기부여를 느낄 수 있도록 한다.

B. 프로젝트 개발환경

① IoT Device

IoT Device는 라즈베리파이를 중심으로 앞에서 소개한 초음파 센서, Buzzer 센서, FND 센서, 네트워크 어댑터 등 여러 센서가 연결된 구조이다. 라즈베리파이는 센서에서 수집되는 데이터에 대한 Pre-Processing과 VCC, Ground, GPIO Pin을 이용해 센서에게 구동 가능할 수 있도록 하는 리소스를 제공한다. 라즈베리파이에는 많은 센서가 연결되기 때문에 VCC, Ground 기능을 연장하기 위한 빵판을 따로 준비했다. 이렇게 다양한 센서에서 출력되는 데이터를 수집, 분석하기 위해서는 라즈베리파이 내부에서 작동하는 프로그램이 필요한데, Python과 C 를 이용해서 소프트웨어를 구현했다.

② AWS Cloud

기본적인 서비스 구조인 서버-클라이언트 환경에서 서버의 역할은 크게 두 가지가 있다. 먼저 클라이언트의 내장 Storage만으로는 쌓이는 데이터가 너무나도 많기 때문에, 서버에는 이를 커버할 수 있는 충분한 저장 공간이 필요하다. 두 번째는 IoT Device와 Application의 네트워크는 달라질 수 있는데, 이들 사이에서 공유해야하는 자원을 제공해주는 중간자 역할이다. AWS Cloud를 이용한다면 서버의 두 가지 기능을 사용자에게 제공할 수 있을 것이다. 이외에도 Google에서 제공하는 Firebase Cloud Messaging 기능을 이용해, 매일 자정 Application 에 Push 메시지를 송신하는 역할을 할 것이다.

③ Application

Android Application을 통해서 사용자는 서비스의 대부분 기능을 활용할 수 있다. 특히 Application을 통해 자신의 공부 현황을 한 눈에 확인할 수 있고, 매일 목표 공부시간을 채우면 받을 수 있는 아이콘(월, 화, 수, 목, 금, 토, 일 / 빨, 주, 노, 초, 파, 남, 보)을 통해 스스로 동기부여를 줄 수 있다. Application의 자세한 컨셉은 다음과 같다. 사용자는 하루 목표 공부시간을 Application에 등록한 후, 목표를 달성한 경우 당일에 아이콘 배지를 받는다. 이때 측정된 공부시간과, 핸드폰 사용 횟수는 다음날 00:00분에 사용자에게 Push 메시지 형식으로 전달된다. 만약 일주일 모두 목표를 달성할 경우에는 종합 평가에서 무지개를, 4~6일을 달성할 경우엔 반 무지개, 3일 이하를 달성할 경우 구름 이미지가 등록된다. Application 개발을 위한 IDE는 Android Studio에서 진행했다.

4. Goal/Problem & Requirements

RAINBOW 서비스의 본질적인 목표는 사용자가 목표한 공부시간만큼 온전히 공부에 집중할 수 있도록 도와주는 것이다. 사용자가 스스로 정한 공부시간 동안 핸드폰으로 다른 행동을 하거나 자리를 비우는 등 공부에 방해가 되는 행위를 알려줌으로써, 사용자 스스로 개선해야 할 점을 발견하고 이를 개선하여 공부에 더욱 집중할 수 있도록 도와준다. 종합해보면 사용자는 ①실제 공부시간 측정 ②목표 공부시간 관리 ③핸드폰 사용 여부 모니터링 기능을 제공하는 RAINBOW 서비스를 통해 공부시간을 효율적으로 관리할 수 있을 것이다.

5. Approach

① 핸드폰의 인터넷 접속여부 확인

라즈베리파이에 네트워크 어댑터를 연결하여 Monitor Mode로 패킷을 캡처한다. 사용자의 핸드폰에서 인터넷 접속을 시도하는 패킷이 인지되거나, 게임을 하는 도중에 발생하는 패킷이 인지되면, Buzzer에서 소리가 나오게 하여 사용자에게 경고한다.

② ‘앉은 상태’가 인식되는지 확인

초음파 센서를 여러 각도에서 이용해 사용자의 무릎까지의 거리를 측정해 바른 자세로 앉아 있는지 인지한다. 옳은 자세로 앉아있으면 Buzzer 센서에서 ‘삐빅’ 소리가, 자세가 틀어지면 Buzzer 센서에서 ‘삐비빅’ 소리가 발생하면서, 사용자는 자세가 틀어졌음을 인지하고 자세를 바르게 함과 동시에, 잘 측정되고 있는지를 알 수 있다.

③ 현재까지 공부한 누적 시간 확인

초음파 센서의 임계값을 이용해서 바른 자세로 앉아있는지를 확인한다. 임계값보다 작다면, 책상-의자 사이에 무릎만큼의 길이가 추가되었다는 뜻으로 바른 자세로 인지된다. 이때 Buzzer 센서에서 ‘삐빅’ 소리가 발생함과 동시에 FND 센서의 공부 시간이 누적된다. 반대로 임계값보다 크다면, 책상-의자 사이에 아무런 방해요소가 없다는 뜻으로 바른 자세가 아닌 것으로 인지된다. 이때는 Buzzer 센서에서 ‘삐비빅’ 소리가 나면서 FND 센서에서는 스톱워치처럼 공부시간 누적이 멈춘다.

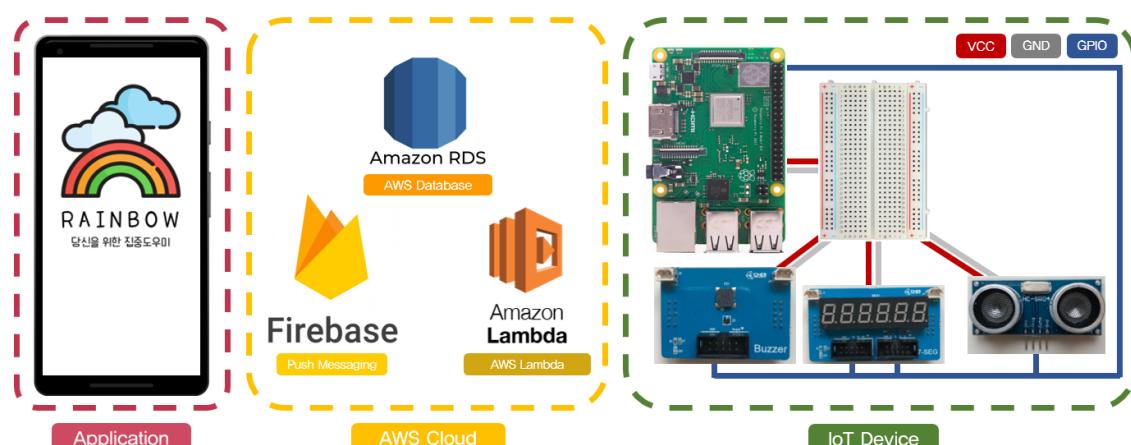
④ 하루 동안의 공부 시간, 핸드폰을 이용한 횟수 알림

누적된 데이터를 바탕으로 하루 동안의 공부 시간, 핸드폰 이용횟수를 보여준다. AWS Cloud에서는 Apache에서 제공하는 웹 서버 컨테이너인 Tomcat과, Google에서 제공하는 FCM(Firebase Cloud Messaging) 기능이 탑재되어있고, 이들을 이용해 당일(00:00~23:59)까지의 공부시간과 핸드폰 사용 횟수에 대한 푸시 알람을 핸드폰으로 전송한다. 이때 핸드폰의 내장 DB에서는 최대 35일의 공부 시간, 핸드폰 사용 횟수를 데이터베이스화해서 보여준다. 이를 초과하는 데이터는 AWS의 S3 등 Storage에다 백업해둠으로써, 사용자가 저장 공간을 효율적으로 사용할 수 있도록 한다.

6. Project Architecture

A. Architecture Diagram + Architecture Description

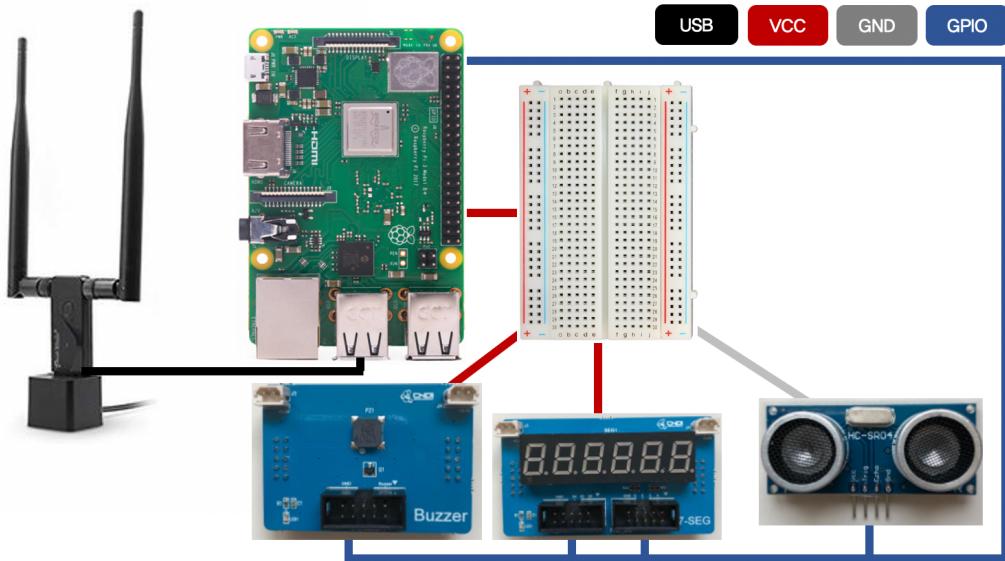
◆ Overall Architecture



- ①IoT Device ②AWS Cloud ③Application 구조

- 각 섹터별 상세 설명은 다음 장에서 이어짐

◆ IoT Device



① 네트워크 어댑터 + Buzzer 센서

- 와이파이에 접속 시도하는 상황 → 해당 패킷 캡쳐
- 와이파이에 접속된 상황 → 인터넷에서 통신하는 패킷 캡쳐
- Buzzer에서 알림 → 사용자가 핸드폰 사용할 때
- 핸드폰 사용 횟수 누적 → 몇 번 사용했는지 Application에서 알려주기

② 초음파 센서

- 책상 밑에 설치해, 의자에 앉아있는지 인식
- 바른자세로 제대로 앉아야만 앉은 것으로 확인
- 거리측정해서 무릎까지의 거리 체크

③ Buzzer 센서

- 앉은 자세가 인식됐는지 여부 알림

④ FND 센서

- 현재까지 공부한 시간 누적 카운트
- 스탑워치와 비슷한 개념

⑤ Bread Board

- 여러 센서의 연결을 위해 부족한 VCC, Ground Pin을 연장하기 위해 사용

◆ AWS Cloud



① Firebase

- Firebase: Firebase Cloud Messaging 기능을 활용해, 사용자에게 푸시 알람 전송
- 당일(00:00~23:59) 공부량이 얼마나 되는지 핸드폰으로 푸시 알람
- 당일(00:00~23:59) 공부 핸드폰 사용량이 몇 번인지 체크 (RPi와 같은 네트워크)

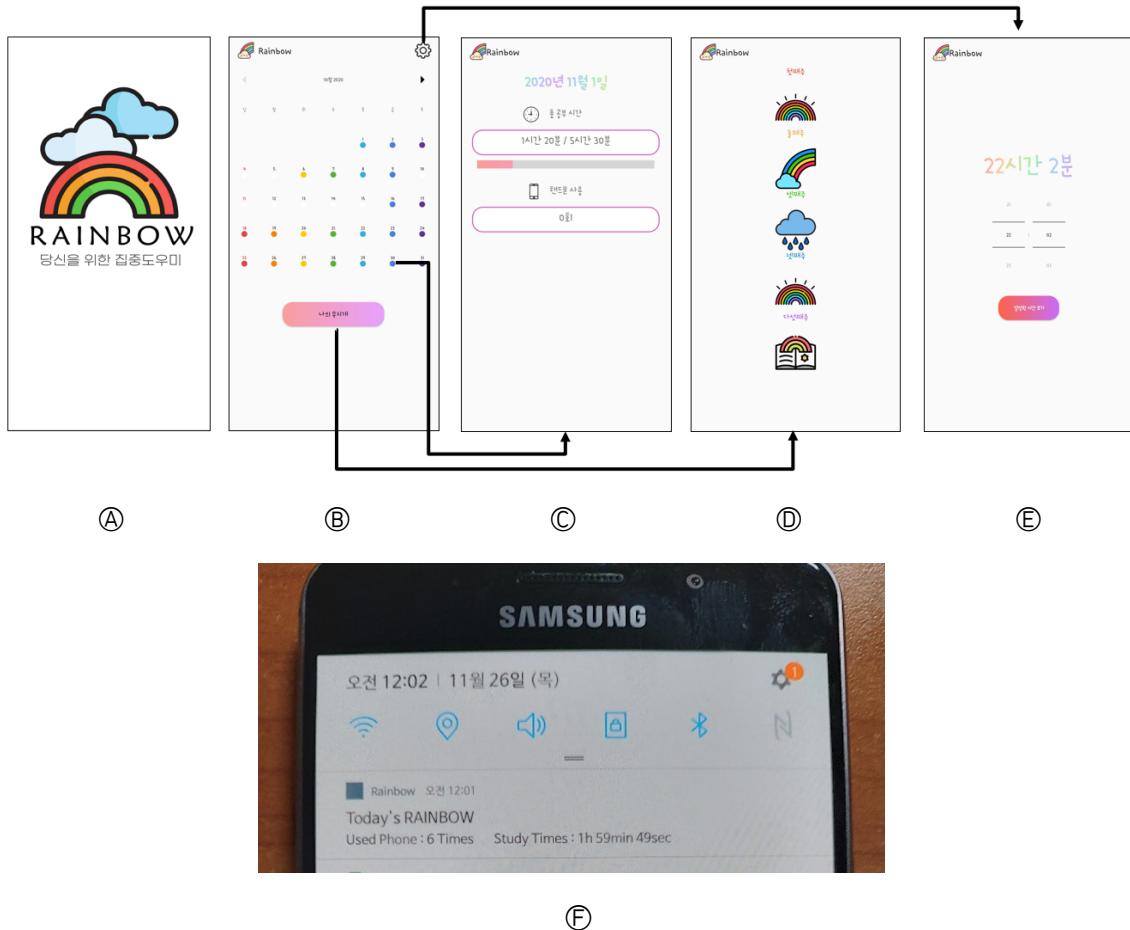
② AWS RDS

- Data Storage 역할
- '당일 누적 공부 시간', '당일 핸드폰 사용 횟수' DB화

③ AWS Lambda + API Gateway

- 라즈베리파이, 앱간 통신을 위한 API Gateway 정의
- 라즈베리파이, 앱에서 Data 요청 시 제공할 수 있도록 RDS를 접근하는 람다 함수 정의

◆ Application



① Concept

- ④: 하루 목표 공부시간 설정 화면
- ⑤: 실제 공부시간 측정 → 목표한 공부시간 채웠는지 여부 체크
- ⑤: 당일 공부시간, 특정 행동(와이파이 연결, 게임 접속 등)을 한 시각과 그 횟수 체크

② UI/UX

- ①: 처음 로딩 화면
- ②: 성취감을 자극하는 Rainbow 아이디어
- ③: 목표 달성을 한 눈에 볼 수 있는 달력형 UI
- ③: 목표 공부시간 달성 → 동그라미 무지개색 배지 제공 (요일, 색 1대1 대응)
- ④: 달성을 따라 주간평가에 서로 다른 이미지 부여 (무지개, 반무지개, 먹구름)
- ⑤: 사용자가 가장 보고 싶어하는 정보인 총 공부시간 상단에 표현
- ⑥: 공부한 시각은 언제인지 한 눈에 볼 수 있는 Bar 형태 UI

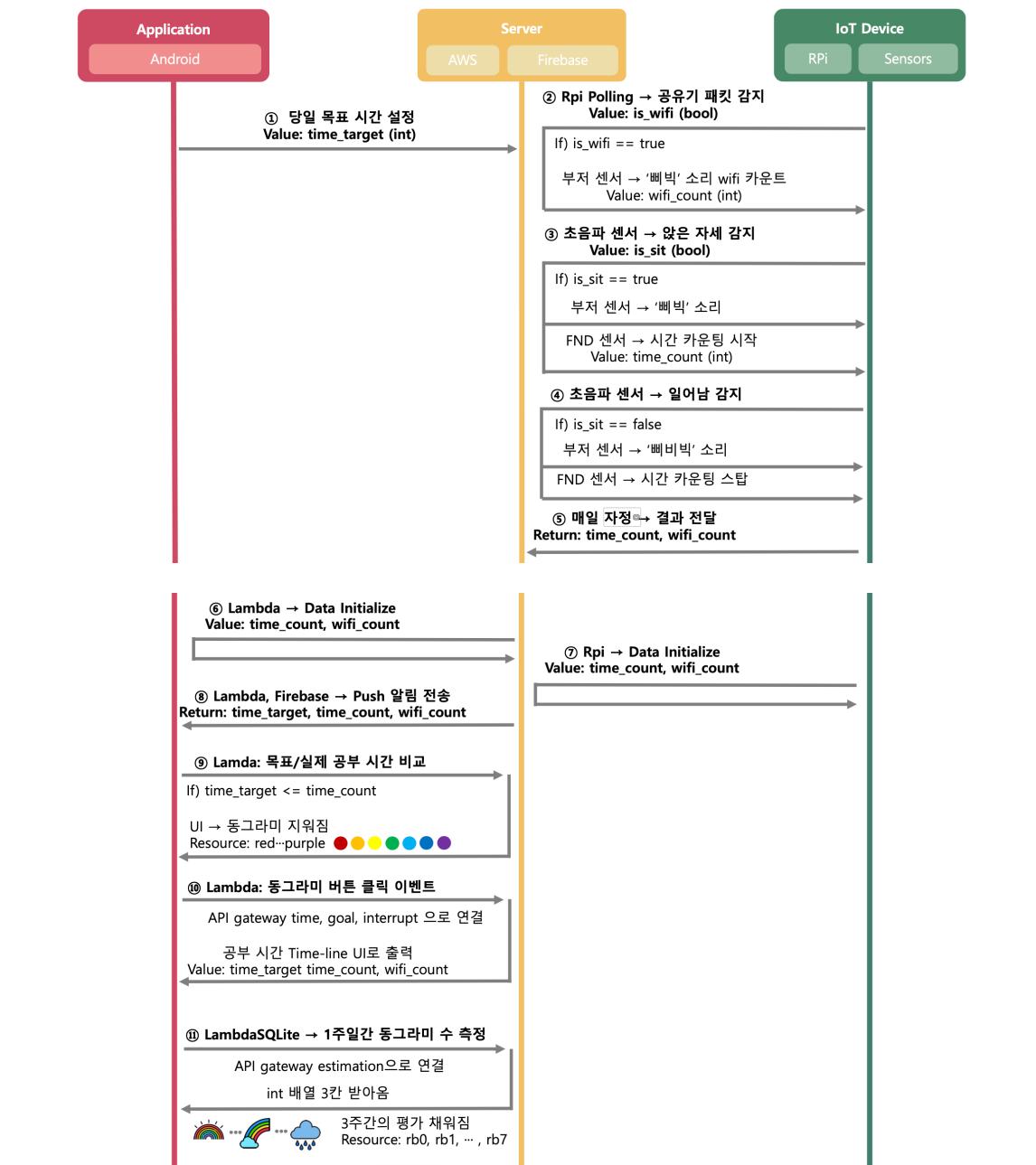
③ 푸시알람

- ⑫: 매일 자정 → 핸드폰 사용횟수, 실제 공부 시간에 대한 푸시 알림 전송

7. Implementation Spec

A. IO Interface B. IPC Interface C. Modules → Ladder Diagram으로 표현

◆ Ladder Diagram



8. Solution

A. Implementations Details

◆ RPi에서 핸드폰에서 나오는 패킷을 캡쳐

```
device_mac : 14:bd:61:ef:d1:0c, rssi : -50, timestamp : 1605679543  
42 Probe Request=====  
device mac : 50:50:a4:0e:16:90  
== This is from your phone ==  
BuzzerFlag is 1  
DETECT_DATE: 2020-11-18  
DETECT_TIME: 21969  
{"message": "Forbidden"}rssi : -48, timestamp : 1605679570
```

<라즈베리파이에서 패킷 캡쳐 화면>

Monitor Mode로 전환된 네트워크 어댑터는, 핸드폰에서 주변 Access Point에 보내는 Probe Request 패킷을 캡쳐할 수 있다. 이때, 패킷 캡쳐를 하려고 하는 핸드폰의 MAC Address를 프로그램에 입력하면, Probe Request 패킷을 파싱해서 MAC Address가 일치하는 패킷들만 잡아낼 수 있다. 패킷을 파싱해서 얻은 정보를 이용해서, 패킷이 발생한 시각(DETECT_TIME)과 날짜(DETECT_DATE) 즉, 사용자가 핸드폰을 사용하기 시작한 시각과 날짜를 서버에 전송하고, Buzzer Flag값을 1로 설정하여 Buzzer에서 알림이 울려서 사용자가 핸드폰을 내려놓을 수 있도록 경고음을 준다.

```
//Adapter model name : AWUS036NH(ALFA), PAU09(PANDA)  
//length : 18  
struct radiotap_header {  
    uint8_t it_version; /* set to  
    uint8_t it_pad;  
    uint16_t it_length; /* entire  
    uint32_t it_present_flags; /* fields  
    uint8_t it_flags;  
    uint8_t it_data_Rate;  
    uint16_t it_channel_frequency;  
    uint16_t it_channel_flags;  
    uint8_t it_antenna_signal;  
    uint8_t it_antenna;  
    uint16_t it_RX_flags;  
};  
  
struct ieee80211_header {  
    uint8_t type_subtype;  
    uint8_t flags;  
    uint16_t duration;  
/*  
Beacon Frame, Probe Request, Probe Response, Authentication, Deauthentication.  
    add1 = Receiver, Destination  
    add2 = Transmitter, Source  
    add3 = BSSID  
Data  
    add1 = Rec, Des, STA  
    add2 = Trans, BSSID  
    add3 = Source  
    Qos Null function, Qos Data, Null function  
    add1 = Receiver, BSSID  
    add2 = Transmitter, Source, STA  
    add3 = Destination  
*/  
    uint8_t add1[6];  
    uint8_t add2[6];  
    uint8_t add3[6];  
    uint16_t fragment_sequence;  
};
```

<패킷 파싱을 위한 구조체 정의>

Probe Request 패킷을 파싱하기 위해서 선언한 구조체이다. Radiotap_Header의 경우 네트워크 어댑터 모델마다 길이가 다르며, 해당 구조체는 본 프로젝트에서 사용한 판다 네트워크

어댑터 (PAU09) 에 맞추어 선언하였다. ieee802.11_Header의 경우, type_subtype을 통해서 Probe Request 패킷인지 확인하는데, 이는 상단에 "#define PROBE_REQUEST 0x40" 라고 정의해둔 값을 활용한다. MAC Address는 uint8_t add[6]의 형태로 정의한다. Probe Request 패킷에서는 ieee802.11에서 Destination, Source, BSSID 순으로 MAC Address 정보를 얻어낼 수 있다.

```

void parsing(const u_char* packet, char* node_mac, char* my_device_mac){
    //packet header setting
    struct radiotap_header *rh = (struct radiotap_header *)packet;
    struct ieee80211_header *ih = (struct ieee80211_header *)(packet + rh->it_length);
    //uint8_t *wlh = (uint8_t *)ih + IEEE_LEN;           //wireless LAN header

    //my_device_mac change to hex
    uint8_t my_mac[6];
    char t[4];

    for(int i = 0; i < 6; i++){
        memcpy(t, (my_device_mac + i*3), 3);
        t[3] = '\0';
        *(my_mac + i) = (uint8_t)strtoul(t, NULL, 16);
    }
    //printf("my_mac : %02x:%02x:%02x:%02x:%02x:%02x", my_mac[0], my_mac[1], my_mac[2], my_mac[3], my_mac[4], my_mac[5]);

    //Catch Probe_request & parsing
    if( ih->type_subtype == PROBE_REQUEST && rh->it_antenna_signal > 186){
        static int cnt = 0;
        cnt++;
        printf("%3d Probe Request=====\n", cnt);
        ledControl();

        char device_mac[17]; // enough size
        //%02X : 2 hex code
        sprintf(device_mac, "%02x:%02x:%02x:%02x:%02x:%02x", ih->add2[0], ih->add2[1], ih->add2[2], ih->add2[3], ih->add2[4], ih->add2[5]);
        printf("device_mac : %s, ", device_mac);

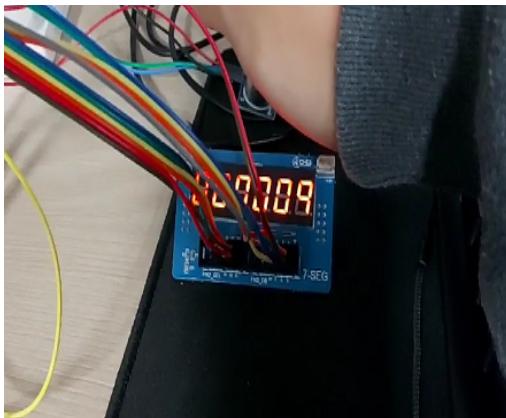
        if((my_mac[0] == ih->add2[0])&&(my_mac[1] == ih->add2[1])&&(my_mac[2] == ih->add2[2])&&(my_mac[3] == ih->add2[3])&&(my_mac[4] == ih->add2[4])&&(my_mac[5] == ih->add2[5])){
            printf("\n==== This is from your phone ===\n");
        }
    }
}

```

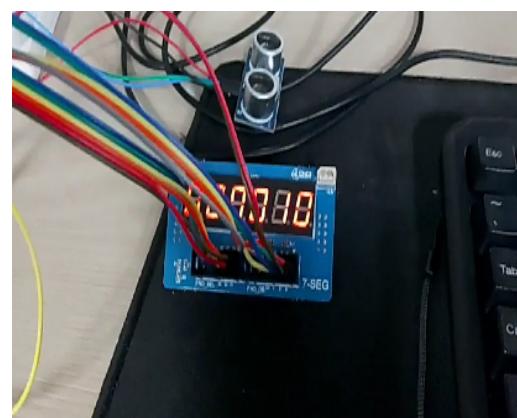
<패킷 파싱 메인 함수>

해당 코드는 패킷을 파싱하는 메인 함수의 일부분이다. 패킷을 앞서 선언한 Radiotap_Header, ieee802.11_Header로 받고, 해당 부분들에 대해서 값의 비교를 진행한다. type_subtype의 값이 Probe Request인지 확인하고, antenna_signal(신호세기) 값이 186보다 큰 값인 경우에만 의미있는 Probe Request 패킷이라고 판단하게 된다. 사용자가 원하는 기기의 MAC 주소와 비교해야 하는데, 이는 memcpy함수와 strtoul함수를 이용해 적절하게 형 변환을 하고, 기기 맥주소인 char device_mac[17]과 비교한다. MAC Address가 일치하면, 이는 사용자의 핸드폰에서 주변 AP에 접속 및 연결을 시도하는 것으로 판단하고 정보를 저장한다.

◆ RPi에 초음파 센서, FND 센서, Buzzer 센서 연동



손을 가져다 댔을 때,
Buzzer 및 FND 센서 작동하는 모습



손을 뺐을 때, Buzzer 센서가 작동하고,
FND 센서가 멈추는 모습

```
// 초음파 센서 감지, 버튼 행동дин
else if (is_sit == 1){
    if (f1 > 30.0 && f2 > 30.0 && f3 > 30.0 & f4 > 30.0 && f5 > 30.0) {
        is_sit = 0;
        softToneWrite(BP, melody[4]);
        delay(250);
        softToneWrite(BP, melody[2]);
        delay(250);
        softToneWrite(BP, melody[0]);
        delay(250);
        softToneWrite(BP, 0);

        // 객체에 키를 추가하고 공부날짜 저장
        json_object_set_string(rootObject, "Study_date", today);
        // 객체에 키를 추가하고 공부시간 저장
        json_object_set_number(rootObject, "Study_time", s_data);

        //Send Data to Cloud Service
        printf("\nDistance: %.4f, %.4f, %.4f, %.4f, %.4f", f1, f2, f3, f4, f5);
        printf("\nSTUDY_DATE: %s", today);
        printf("\nSTUDY_TIME: %d\n", s_data);

        // JSON_Value를 사람이 읽기 쉬운 문자열(pretty)로 만든 뒤 파일에 저장
        json_serialize_to_file_pretty(rootValue, "today.json");

        // 객체에 키를 추가하고 공부날짜 저장
        json_object_set_string(rootObject, "Study_date", tomorrow);
        // 객체에 키를 추가하고 공부시간 저장
        json_object_set_number(rootObject, "Study_time", 0);
        json_object_set_number(rootObject, "Study_goal", 0);
        json_object_set_number(rootObject, "Study_achieved", 0);

        // JSON_Value를 사람이 읽기 쉬운 문자열(pretty)로 만든 뒤 파일에 저장
        json_serialize_to_file_pretty(rootValue, "tomorrow.json");
    }
}
```

```

// 초음파 센서를 이용해 Distance를 받아오는 함수
double getDistance() {
    double fDistance = 0.0;
    int nStartTime, nEndTime;
    nStartTime = nEndTime = 0;

    digitalWrite(TP, LOW);

    delayMicroseconds(10);
    digitalWrite(TP, HIGH);

    delayMicroseconds(10);
    digitalWrite(TP, LOW);

    while (digitalRead(EP) == LOW);
    nStartTime = micros();

    while (digitalRead(EP) == HIGH);
    nEndTime = micros();

    fDistance = (nEndTime - nStartTime) / 29. / 2.;

    return fDistance;
}

// FND를 선택하는 함수, S0 ~ S5 중 파라미터(position)에 해당하는 FND 선택
void FndSelect(int position) {
    int i;
    for (i = 0; i < 6; i++) {
        if (i == position) {
            digitalWrite(FndSelectPin[i], LOW); // 선택된 FND의 Select 핀 ON
        }
        else {
            digitalWrite(FndSelectPin[i], HIGH); // 선택되지 않은 FND의 Select 핀 OFF
        }
    }
}

```

초음파 센서는 사용자 다리와 떨어져있는 거리를 측정하는 역할을 담당한다. 초음파 센서를 책상 밑에 사용자의 무릎 높이에 설치한다. 현재 RPi.c에 구현된 내용으로는 임계값=25cm라고 설정되어 있지만, 나중에 사용자마다 적합하게 길이를 조정해주면 된다. 초음파 센서는 임계값을 이용해 사용자가 앉은 상태를 인식한다. FND 센서는 누적된 공부 시간을 시각화하는 기능을 제공한다. FND 센서는 사용자가 확인할 수 있도록 책상 위에 설치하고, 자동으로 누적 공부 시간이 측정되는 ‘스탑워치’ 기능과 같다고 생각하면 된다. “측정거리 < 임계값”이면 FND 센서가 시작되고, “측정거리 >= 임계값”이면 FND 센서가 멈추고, 누적된 시간을 데이터로 보관하고 있다. Buzzer 센서는 시간의 누적 여부(인식 여부)를 청각적으로 알려주는 기능을 제공한다. 만약 “측정거리 < 임계값”이면 Buzzer에서는 도미솔♪을 출력하고 “측정거리 >= 임계값”이면 솔미도♪를 출력한다.

◆ AWS RDS

| Study | | |
|-------|--|------------------------------|
| PK | study_date study_time study_goal study_achieved | VARCHAR INT INT INT |

| detect | | |
|--------|----------------------------|----------------|
| PK | detect_date detect_time | VARCHAR INT |

왼쪽은 구축한 데이터베이스의 ER 다이어그램이다. 테이블은 Study와 Detect로 구성된다. Study 테이블은 공부한 날짜를 나타내는 STUDY_DATE, 공부한 시간을 나타내는 STUDY_TIME, 목표 공부 시간을 나타내는 STUDY_GOAL, 목표 달성을 여부를 나타내는 STUDY_ACHIEVED 앤트리뷰트로 구성된다. 또한, Detect 테이블은 인터넷을 한 날짜를 나타내는 DETECT_DATE, 인터넷을 한 시각을 나타내는 DETECT_TIME으로 구성된다. STUDY_DATE 앤트리뷰트는 일자별로 1개의 튜플을 가져야 하기 때문에 12시에 의무적으로 생성되며, DETECT_DATE 앤트리뷰트는 사용자가 인터넷을 했을 때에만 튜플이 생성된다.

오른쪽은 AWS Cloud에 RDS 인스턴스를 구축한 모습으로, RDS를 구축하기 위해 EC2 인스턴스를 만들었다.

◆ AWS Lambda

AWS에서 서비스를 대표하는 Lambda로 데이터베이스에 접근하고 쿼리를 실행한다. 다음은 공부의 성취를 나타내는 주간평가를 구현한 람다 함수 중 일부이고, 아래는 핸드폰 사용 횟수 감지하는 람다함수이다.

```
sql = "select STUDY_ACHIEVED from Study where((STUDY_DATE between %s and %s) and (STUDY_ACHIEVED is not null))"

cursor = conn.cursor()
cursor.execute(sql, (startdate1, enddate1))
result1 = cursor.fetchall()
print("sql1 : ", sql)
res1 = []
for item in result1:
    res1.append(item[0])

estimate = 0
for i in res1:
    if i == 1:
        estimate += 1
if estimate == 7:
    final.append(3)
elif 4 <= estimate <= 6:
    final.append(2)
else:
    final.append(1)

cursor.execute(sql, (startdate2, enddate2))
result2 = cursor.fetchall()

res2 = []
for item in result2:
    res2.append(item[0])

estimate = 0
for i in res2:
    if i == 1:
        estimate += 1
if estimate == 7:
    final.append(3)
elif 4 <= estimate <= 6:
    final.append(2)
else:
    final.append(1)

cursor.execute(sql, (startdate3, enddate3))
result3 = cursor.fetchall()

res3 = []
for item in result3:
    res3.append(item[0])
```

<주간 평가를 나타내는 람다 함수 중 일부>

```
def lambda_handler(event, context):
    target = event['queryStringParameters']['target']
    cursor = conn.cursor()
    sql = "SELECT DETECT_TIME FROM Detect where DETECT_DATE = %s"
    cursor.execute(sql,target)
    result = cursor.fetchall()
    res = []
    count = 0
    for item in result:
        res.append(item)
        count+=1
    d = dict()
    d['target'] = count
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps(d)
    }
```

<핸드폰 사용 횟수를 얻어오는 람다 함수>

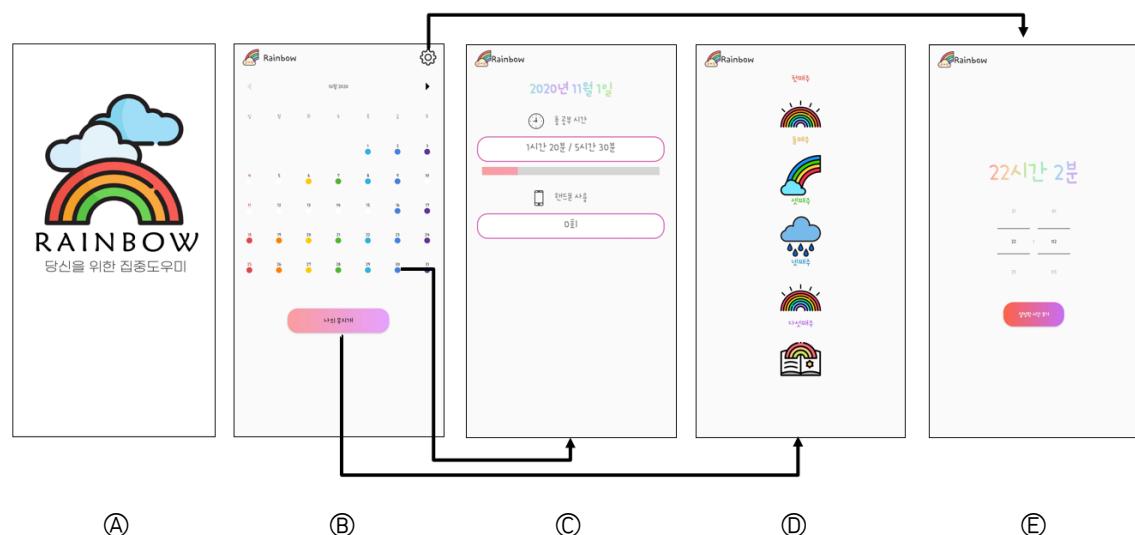
◆ AWS RESTful API

The screenshot shows the AWS API Gateway console for a RESTful API named 'rainbow-API'. The top navigation bar includes the API name, creation details ('Created by AWS Lambda r89kbtj8x9'), and deployment information ('Regional 2020-11-12'). The main interface displays the API's resources under the '작업' (Operations) tab. The '리소스' (Resources) section lists several endpoints:

- /achievement (GET)
- /calendar (GET)
- /detect (POST)
- /estimation (GET)
- /goal (GET, POST)
- /interrupt (GET)

DB와 라즈베리파이, 어플리케이션 간에 데이터를 주고받을 수 있는 AWS API Gateway를 개발했다. 본 프로젝트에서는 API를 읽는 것만으로도, 매뉴얼 없이 어떤 역할을 하는지 명확하게 나타내기 위해 RESTful API를 사용했다. 위 사진과 같이, 평가하는 람다 함수와의 연결을 estimation의 GET, 목표 공부시간에 관한 람다 함수와의 연결을 goal 리소스의 GET 등 url만 보고도 함수의 동작을 쉽게 유추할 수 있다.

◆ Application



본 프로젝트를 통해 구현할 애플리케이션의 흐름을 그리는 과정에서 Application의 디자인과 레이아웃 흐름을 실제로 구현하였다. 1주일인 7일간 목표 달성을 도와준다는 의미로 무지개의 일곱 빛깔로 UI를 제공하겠다는 컨셉을 잡았으며, 지금은 위에 소개된 Layout의 기본 골조에 따라 Android Studio를 이용해 Application 개발을 진행하고 있는 상태다.

Ⓐ 화면은 애플리케이션이 처음 실행됐을 때 나타나는 로딩 화면이다. Ⓑ 화면은 애플리케이션의 메인 화면으로, 사용자가 목표한 시간만큼 공부를 했을 때 동그란 배지가 각 요일마다 등록된다. 정보를 보다 직관적으로 표현하기 위하여 하단에 [나의 무지개] 버튼을 만들어 클릭 시 공부 성취량을 나타내는 Ⓒ 화면으로 전환된다. 화면 구성은 간소화하여 깔끔하게 필요한 정보만을 표시했다. Ⓓ 화면은 Ⓑ 화면에서 일자를 클릭하면 전환된다. 공부 시간에 대하여 실제 공부 시간과 목표 시간을 함께 나타내어, 사용자의 동기를 부여할 수 있도록 도움을 주었고, 상단의 프로그래스 바 UI를 통해 자신의 공부 성취량을 직관적으로 파악할 수 있도록 화면을 구성했다. 하단에는 당일 핸드폰을 사용한 횟수를 표시하여 한 눈에 핸드폰 사용량을 파악할 수 있도록 구성했다. Ⓔ 화면은 Ⓑ 화면에서 나의 무지개를 클릭하면 나타나는 화면이다. 무지개, 반무지개, 먹구름 등 달성을 따라 주간평가에 서로 다른 이미지로 배지를 부여한다. 누적된 배지를 통해 사용자에게 공부량을 채우기 위한 동기부여 기능을 제공한다. Ⓕ 화면은 Ⓑ 화면에서 상단의 톱니바퀴 설정을 클릭하면 나타나는 화면이다. 해당 화면에서 사용자는 목표 공부 시간을 설정할 수 있다.

◆ Application API 연동

```
public interface RainbowAPI {  
  
    @GET("goal")  
    Call<PostItem> getGoalTime(@Query("target") String target);  
  
    @GET("time")  
    Call<PostItem> getStudyTime(@Query("target") String target);  
  
    @GET("interrupt")  
    Call<PostItem> getInterruptTime(@Query("target") String target);  
  
    @GET("achievement")  
    Call<PostItemStringList> getAchievementList(@Query("achieved") int target0, @Query("start") String target1, @Query("end") String target2);  
  
    // @FormUrlEncoded  
    @POST("goal")  
    // Call<PostItemGoal> postGoal(@Field("date") String date, @Field("time") int time);  
    Call<PostItemGoal> postGoal(@Body PostGoal newgoal);  
  
    @GET("estimation")  
    Call<PostItemEstimation> getInterruptTime(@Query("start") String target1, @Query("end") String target2);  
}
```

안드로이드에서 API를 연동하는데 레트로핏이라는 오픈소스 라이브러리를 사용하였다. 이것은 Post, GET 등의 HTTP Method를 사용하는 REST API를 안드로이드 상에서 간단히 다룰 수 있게 해준다. 위는 Rest API에 대한 인터페이스를 정의한 모습이다. 먼저 HTTP Method

를 명시한 뒤, AWS API Gateway에서 정의한 리소스 명 또는 URL을 적어줌으로써 정의한다.

```
package com.example.rainbow;

import com.google.gson.annotations.SerializedName;

public class PostGoal {

    @SerializedName("date") private String date;
    @SerializedName("time") private int time;

    public String getDate() { return date; }
    public int getTime() { return time; }

    public void setDate(String date) { this.date = date; }
    public void setTime(int time) { this.time = time; }

}
```

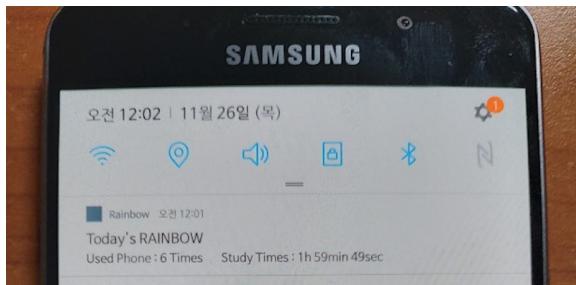
서버로 받은 응답에 대한 body는 json 객체로, 키와 타입에 대한 클래스를 선언한다. 레트로핏에서는 json의 body에 해당하는 부분을 클래스의 변수 타입과 변수명에 맞게 데이터를 매치한다.

```
Retrofit mRetrofit = new Retrofit.Builder()
    .baseUrl("https://r89kbtj8x9.execute-api.us-east-1.amazonaws.com/last/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
RainbowAPI mRetrofitAPI = mRetrofit.create(RainbowAPI.class);

Call<PostItemListStringList> mCallMovieList = mRetrofitAPI.getAchievementList(0,param[0],param[1]);
mCallMovieList.enqueue(new Callback<PostItemListStringList>() {
    @Override
    public void onResponse(Call<PostItemListStringList> call, Response<PostItemListStringList> response) {
        PostItemListStringList result = response.body();
        new ApiSimulator(result.getTarget()).executeOnExecutor(Executors.newSingleThreadExecutor());
    }
    @Override
    public void onFailure(Call<PostItemListStringList> call, Throwable t) {
        t.printStackTrace();
    }
});
```

레트로핏으로 url, 보낼 데이터와 정의한 인터페이스를 사용하여 비동기 형식으로 데이터를 전송한다. 그 후, 정의한 클래스를 이용하여 데이터를 받는다. 받아온 데이터의 사용은 클래스의 변수를 사용하는 것과 같이 쉽게 가능하다.

◆ Application 푸시알림



<핸드폰에 푸시알림이 도착한 화면>

| 알림 | 상태 | 몇몇 | 시작/전송 | 종료 | 전송 | 열림 |
|----------------------------------|-------|-----|-------------------------|----|-------|----|
| Used Phone : 5 Times Study Ti... | ✓ 완료됨 | --- | 2020.11.29. 오전 12:00 | - | 1천 미만 | 0% |
| Used Phone : 4 Times Study Ti... | ✓ 완료됨 | --- | 2020.11.28. 오전 12:00 | - | 1천 미만 | 0% |
| Used Phone : 6 Times Study Ti... | ✓ 완료됨 | --- | 2020.11.27. 오전 12:01 | - | 1천 미만 | 0% |

<Firebase에 프로젝트를 생성해서 푸시알림 구현>

푸시알림은 Firebase Cloud Messaging을 활용하여, 클라이언트 앱의 FCM Token을 저장하고 Firebase Function을 활용하여 Push를 전송한다. 밤 12시에 업데이트 되는 핸드폰 사용 횟수와 공부시간 데이터를 받아서 사용자에게 알려주고, 사용자는 이를 통해서 하루가 끝날 때 자신의 공부량을 알아볼 수 있으며, 해당 정보는 한 주의 무지개를 생성하는 데에 활용된다.

B. Implementations Issues

- 네트워크 어댑터에서 패킷을 캡쳐하는 것을 최소화 하는 것이 네트워크 패킷 캡처를 구현하면서 가장 중요시 했던 부분이다. 모니터 모드로 전환된 어댑터에는 일정 범위 (20m이상) 안에 있는 모든 패킷들이 잡히므로 불필요한 데이터가 쌓일 수 있다. 이를 해결하기 위해 AP에 접속 및 연결을 시도하는 Probe Request 패킷이 발생할 경우를 인터넷 접속으로 감지하여 해당 패킷만 필터링 하여 캡쳐하도록 하였고, 해당 프로그램의 사용자는 적어도 2~3m 내에 있기 때문에 신호세기인 RSSI 값이 일정수준 이상인 패킷들을 필터링 하여 캡쳐하였다.
- 패킷의 구조체를 직접 만드는 부분이 네트워크 패킷 캡처를 구현하면서 어려웠던 부분이다. 네트워크 어댑터마다 패킷을 캡쳐할때 radiotap_header의 구조체 형식이 조금씩 달라서, 해당 판다 네트워크 어댑터에 맞춘 radiotap_header를 만들어 사용하였다. 또한 구조체 내부에서 uint8_t, uint16_t, uint8_t [6] 과 같이 형식이 모두 다른 단위값들을 사용자가 char 형태로 입력하는 MAC주소와 비교하는 과정에서도 어려움을 겪었고, 많은 시행착오 끝에 해결하였다.
- API gateway의 역할과 작동 방식에 대한 경험이 없어서 이해에 어려움을 겪었지만, 수업을 통해 이해했고 완벽하게 해결하였다.
- 주간 평가 알고리즘 구현 시, 원하는 데이터를 얻기 위한 효과적인 sql문을 작성하는데 어려움을 겪었다. 먼저 첫 번째 일요일을 찾은 뒤, 그 후 3주 동안의 성취여부를 불러오는 방식으로 sql문을 작성하였다.
- 임의의 날짜에 점을 찍고 요일 별로 다른 색으로 설정하고자 하였으나, 설정하여 찍은 점은 색을 변경하는 방법을 찾지 못했다. 요일 로별 색이 다른 모든 점을 일괄적으로 찍고 난 뒤, 점을 지우는 방식으로 구현하였다.
- RDS에서 안드로이드 스튜디오 내장데이터베이스로 데이터를 복사하고 어플에서는 내장 데이터베이스를 쓰려고 했다. 그러나, 데이터 불일치 문제, 바로 연결했을 때의 장점등의 문제를 이유로 사용하지 않기로 하였다. RESTful API의 쉬운 연동을 위해 Square 사에서 제공하는 오픈소스 라이브러리 Retrofit2를 사용하였다.

9. Result

A. Experiments

1. 라즈베리파이 wifi 연결 및 센서 설치
2. 라즈베리파이 코드 구동
3. 목표시간 설정
4. 라즈베리파이가 설치된 의자에 앉아서 공부 시작
5. 핸드폰 디바이스로 인터넷 사용
6. Buzzer 센서, 푸시 알림 작동 확인
7. 어플로 공부 시간과 공부 외 활동 시간 확인
8. 해당 과정을 며칠간 반복
9. 자정에 공부시간에 대한 알림을 받고 누적된 정보를 확인

B. Result Analysis and Discussion

데모는 20.11.23(월) ~ 20.11.29(일) 동안 진행하였다.

- 11.23(월)에는 0시간 00분 공부했고, 목표시간은 2시간 00분, 핸드폰 사용 횟수는 0회였다.
- 11.24(화)에는 3시간 57분 공부했고, 목표시간은 4시간 00분, 핸드폰 사용 횟수는 22회였다.
- 11.25(수)에는 1시간 59분 공부했고, 목표시간은 3시간 00분, 핸드폰 사용 횟수는 6회였다.
- 11.26(목)에는 2시간 02분 공부했고, 목표시간은 1시간 30분, 핸드폰 사용 횟수는 4회였다.
- 11.27(금)에는 1시간 37분 공부했고, 목표시간은 1시간 30분, 핸드폰 사용 횟수는 0회였다.
- 11.28(토)에는 1시간 54분 공부했고, 목표시간은 1시간 30분, 핸드폰 사용 횟수는 5회였다.
- 11.29(일)에는 2시간 44분 공부했고, 목표시간은 2시간 00분 핸드폰 사용 횟수는 0회였다.

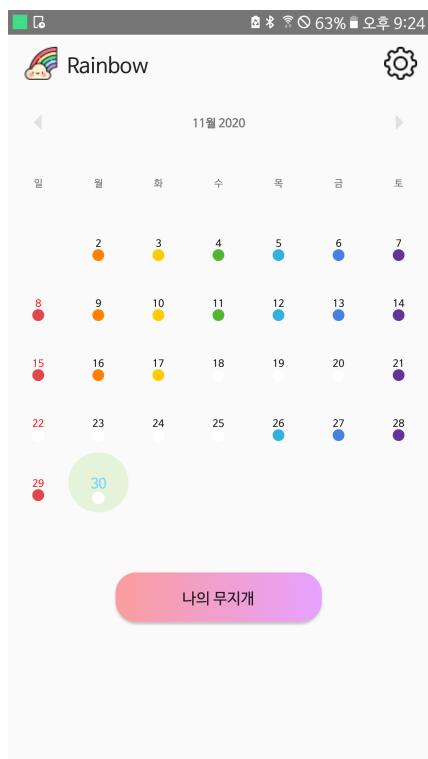
20.11.24(화)에 목표 공부시간을 4시간 00분으로 변경하였다.

20.11.25(수)에 목표 공부시간을 3시간 00분으로 변경하였다.

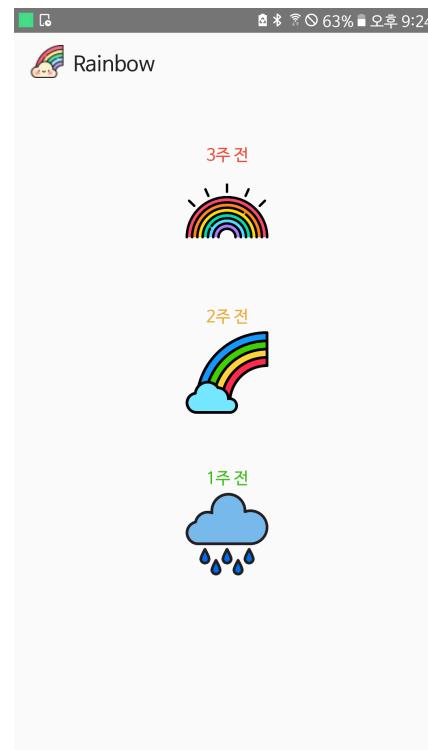
20.11.26(목)에 목표 공부시간을 1시간 30분으로 변경하였다.

20.11.29(일)에 목표 공부시간을 2시간 00분으로 변경하였다.

데모가 끝난 뒤, 20.11.30(월)에 달력 화면을 캡처한 화면이다.

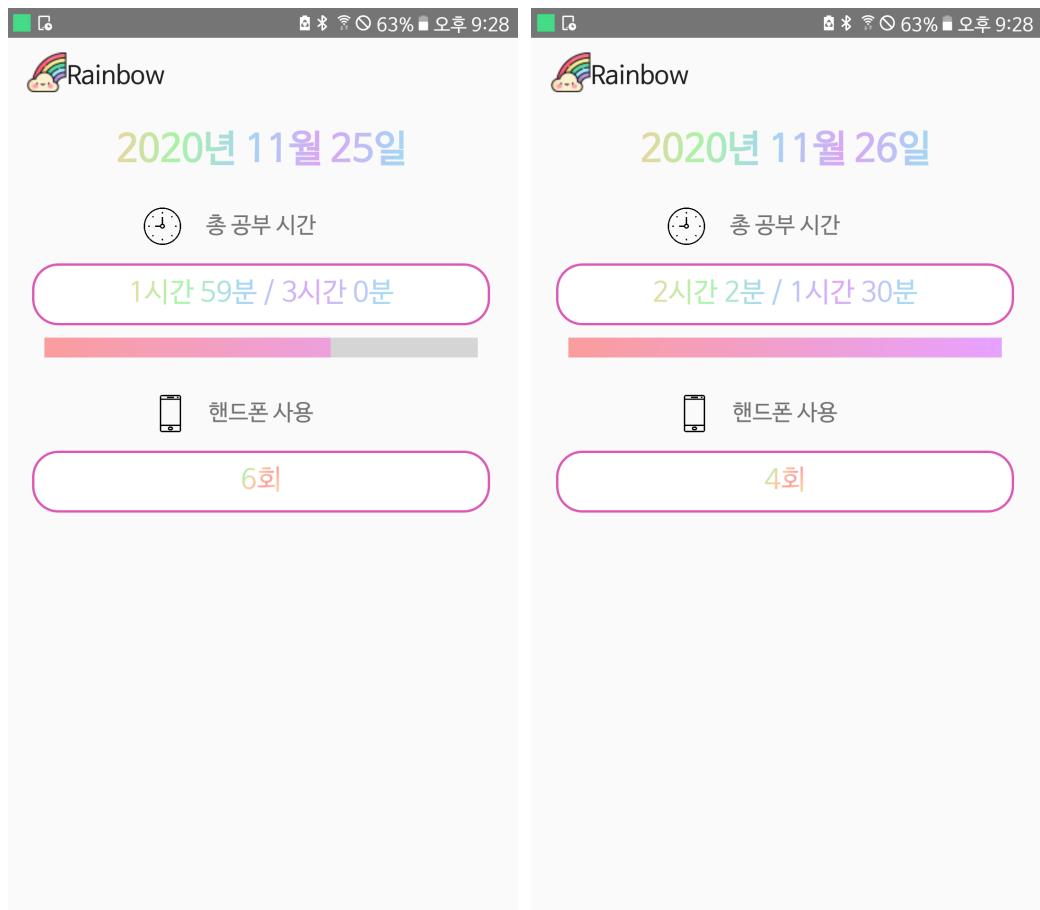


공부목표를 달성하지 못한 22(일), 23(월), 24(화), 25(수)에는 무지개 닷을 받지 못했다. 그러나, 공부목표를 달성한 26(목), 27(금), 28(토)에는 무지개 닷을 받았다.



데모가 끝난 11.29일(일)에, 11.22(일) - 11.28(토)에 대한 주간 평가가 등록된 모습을 볼 수 있다. 일주일 동안 달성한 일자가 총 3일이므로, '1주 전' 항목에 구름 스티커를 받은 모습을 확인할 수 있다.

다음은 일주일 동안의 공부량과 핸드폰 사용량이 가장 잘 나타나는 일자화면이다.



10. Division & Assignment of Work

| 담당자 | 항목 |
|-----|----------------------------------|
| 김성수 | 라즈베리파이 개발, 센서 연동, 모듈 연결 및 데모 |
| 서예진 | 네트워크 패킷 캡처, Firebase 푸시 알림 |
| 최용욱 | 람다 작성, API gateway 구축 등 AWS 구성 |
| 강세희 | 달력 페이지 기능 구성, 어플 데이터베이스화, API 연동 |
| 이진아 | 일간, 시간 설정, 주간 평가 페이지 기능 구성, 디자인 |

◆ Notion

회의록

↓ Click **List View** to create and see other views, including a board organized by meeting type.

속성 필터 정렬 X 검색 ...

새로 만들기

- 20201129 오전 12시 (진아작성 - 최종 발표 관련)
- 20201122 오전 12시 (세희작성 - 최종 안드로이드 개발 관련)
- 20201115 오전 12시 (성수작성 - 모듈 연동 관련)
- 20201108 오후 10시 (예진작성 - FCM 연동 및 Rambda 함수 관련)
- 20201018 오전 12시 (용욱작성 - 2차 발표 준비)
- 20201011 오후 10시 (진아작성 - 현재까지 개발 과정 동기화)
- 20201004 오후 10시 (세희작성 - 2차 발표 내용 확정 및 역할 분담)
- 20200927 오전 12시 (성수작성 - 개발 진행 정도 공유)
- 20200920 오후 10시 (용욱작성 - 개발과정 동기화 및 아이디어 회의)
- 20200913, 20200914 정리 (예진작성 - 주제 확정)
- 20200914 오후 4시(주제회의3)
- 20200913 오후 2시(주제회의2)
- 20200912 오후 10시(주제회의1)

+ 새로 만들기



한일

◆ Github

<https://github.com/KHU-Rainbow>

The screenshot shows a GitHub application window with the following details:

- Repository:** Application (jina)
- Branch:** master
- Commits:** 313 (including 1 merge pull request)
- Pull Requests:** 13
- Issues:** 0
- Dependency Graph:** A complex network of nodes representing Java classes and methods. Nodes are color-coded by package: blue for `com.rainbow`, purple for `com.rainbow.util`, and grey for `java.util`. Edges represent method calls. A prominent purple node at the top is labeled `com.rainbow.util.MonitoringUtil`.
- Commits:** A list of 313 commits, with commit #13 being the most recent. Commit #13 is titled "Merge pull request #13 from KHL-Rainbow/behee".
- File Changes:** A list of files changed in commit #13, including `src/main/java/com/rainbow/util/MonitoringUtil.java`, `src/main/java/com/rainbow/util/ResourceUtil.java`, `src/main/java/com/rainbow/util/StringUtil.java`, `src/main/java/com/rainbow/util/Util.java`, and `src/main/java/com/rainbow/util/UUIDUtil.java`.

11. Conclusion

사용자가 자신의 공부량에 대해 수치적으로 성취감을 느낄 수 있도록, 핸드폰 사용 시간을 알려줌으로써 의식적으로 공부하는 습관을 만들 수 있도록 하는 공부량 입력 자동화 시스템 'Rainbow' 프로젝트를 진행하였다. 프로젝트 진행 초반에 고려하였던 기능들을 모두 구현을 하였고, UI적인 측면에서는 계획보다 더 나아진 결과를 보여줬다. 데모를 통해서, 프로젝트의 기능들이 정상 동작함을 증명하였고, 데모자가 12시에 자신의 공부량에 대한 알림을 받고 그 다음날 더 공부하게 됐다는 점에서 프로젝트가 의의가 있음을 보여줬다.

이 프로젝트에서 구현했던 앱의 기능과 UI만으로도 사용자가 공부를 하며 성취감을 충분히 느낄 수 있다. 그러나, 이 앱의 기능이나 UI를 수정하거나 다른 기능들을 추가로 구현할 수도 있을 것이다. 예를 들자면, 이 앱의 성취도를 표현해주는 방식은 성취 여부를 표현하는 달력 페이지의 점과 어느 정도를 성취했는지를 표현하는 일자페이지의 프로그래스바로 구성되어 있다. 이렇듯, 단순 성취 여부와 진행정도만을 표현해주는 게 아니라, 이를 단계별로 나누어 1/7 분량별로 성취했을 때, 무지개 색 하나를 얹는 방식으로 구현도 가능하면 사용자가 공부를 하며 더욱 큰 성취를 느낄 수 있을 것이다.

또한, 해당 프로젝트에서 설계한 모듈들을 이용하여 다른 프로젝트를 진행에 적용한다면 더욱 좋은 프로젝트가 될 수 있을 것이다. 예를 들자면, 본 프로젝트의 IoT 구현 부분이 사용자의 부재를 알 수 있으므로, 화면 보호기가 효과적으로 작동할 수 있게 도와주는 프로젝트를 진행하는데 본 프로젝트를 연계할 수도 있을 것이다.

◆ [Appendix] User Manual



당신을 위한 집중 도우미

사용설명서

최신기술프로젝트2 B조
강세희, 김성수, 서예진, 이진아, 최용욱

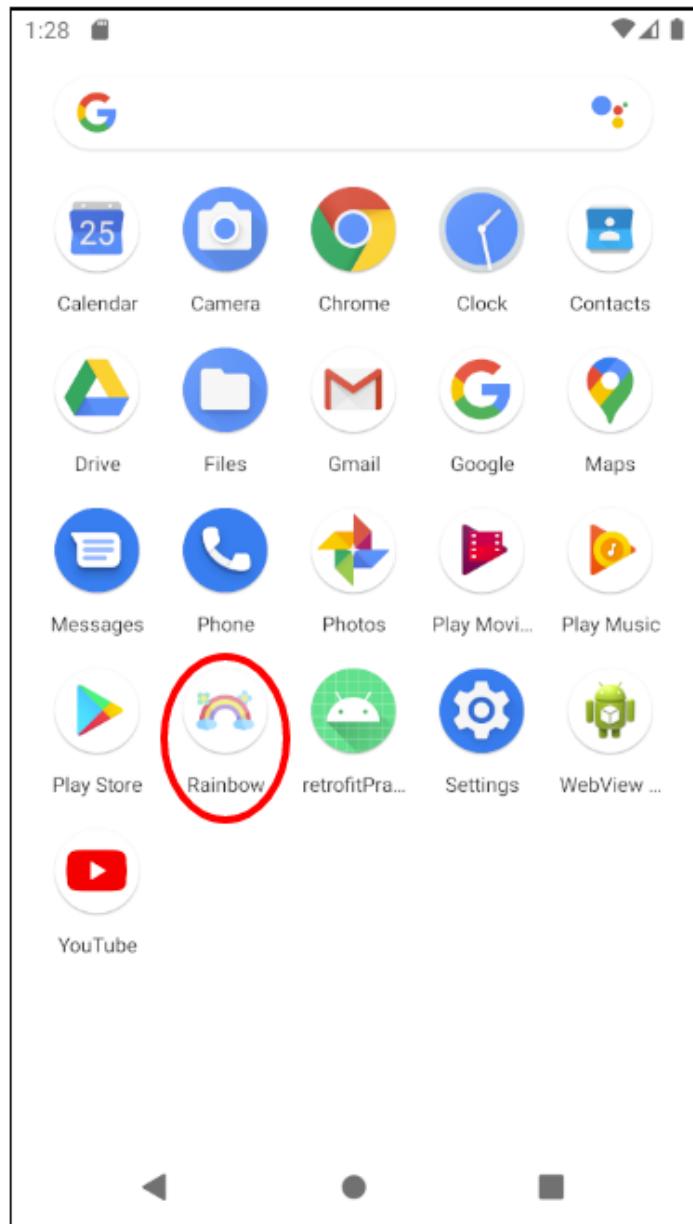




CONTENTS

1. 프로그램 실행
2. 인트로 화면
3. 메인 화면
4. 시간 설정 화면
5. 일자 화면
6. 주간 무지개 화면

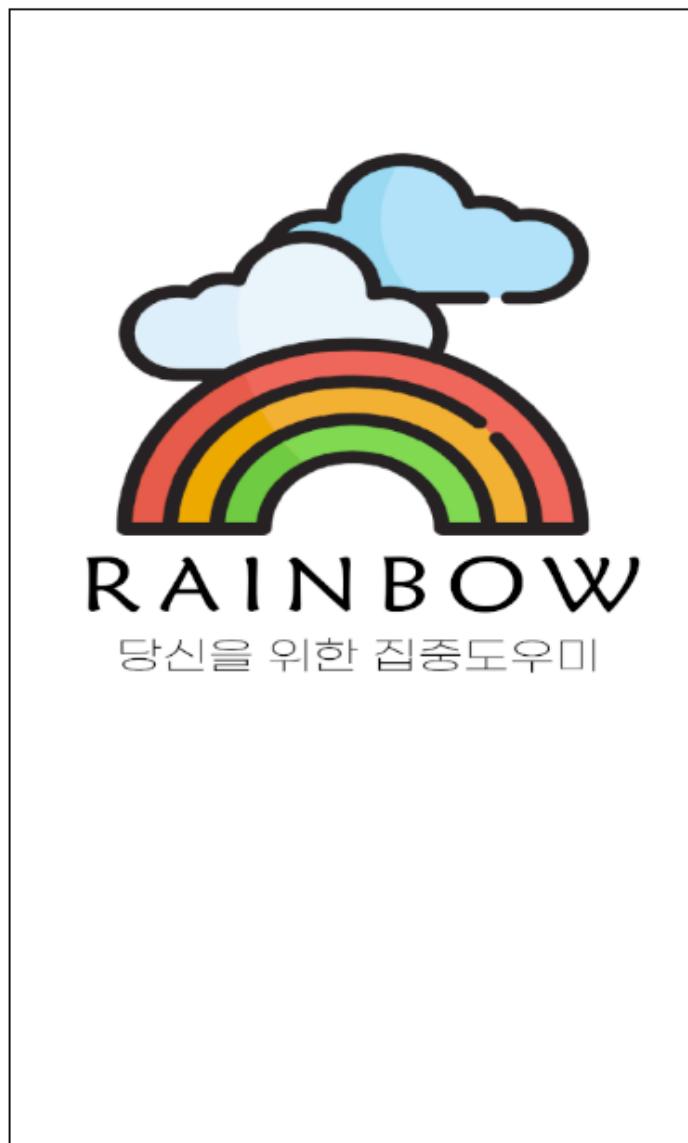
프로그램 실행



앱 설치가 완료되면,
스마트폰 메뉴에 왼쪽 화면과 같은
무지개 아이콘이 생깁니다.

아이콘을 선택하면
프로그램이 실행됩니다.

인트로 화면



위와 같은 인트로 화면이 표시되며
앱이 실행됩니다.

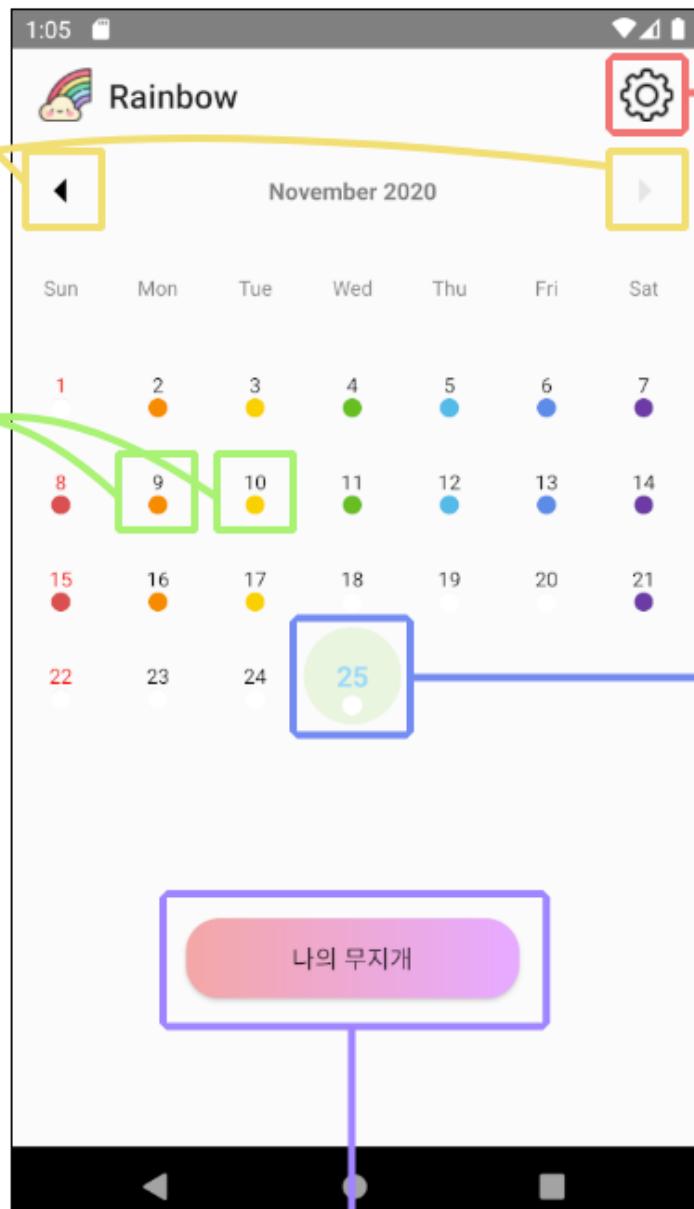
메인 화면

어플 실행에 성공하면, 사용자의 공부량을 한 눈에 볼 수 있는 달력이 나타납니다.

_ 여러가지 기능들을 한눈에 보고, 사용해보세요.

달력 넘기기 버튼

날짜 하나하나가
일자 페이지로
넘어가는 버튼



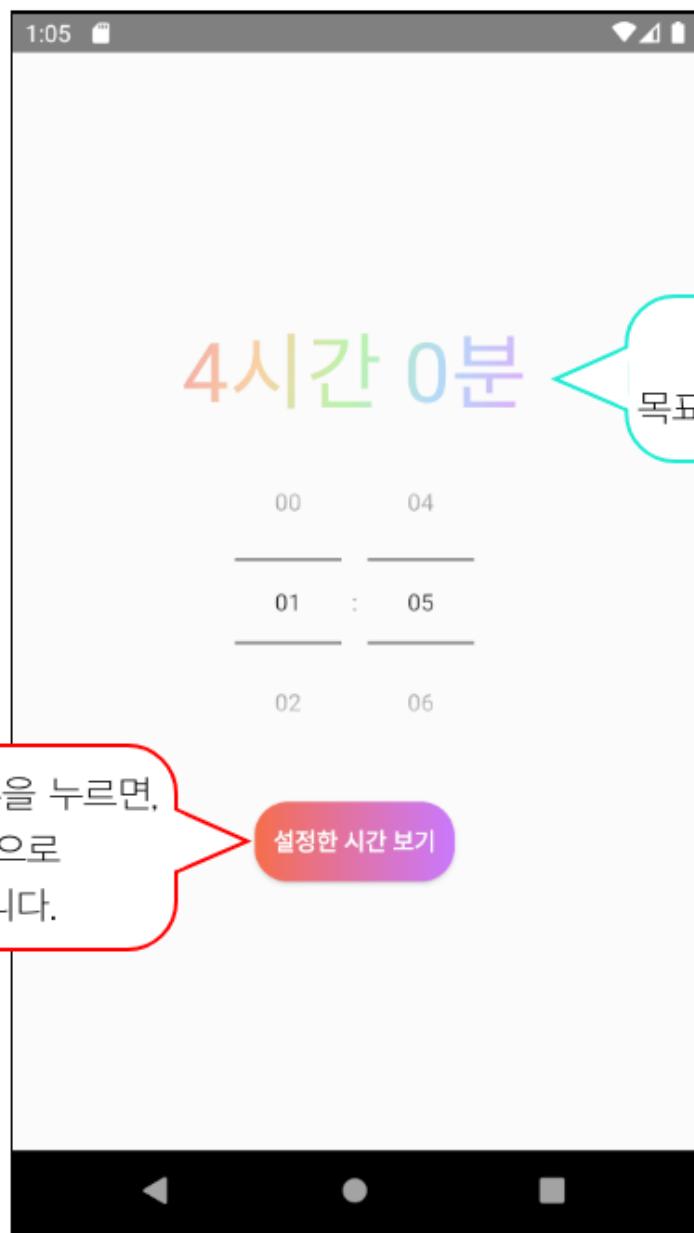
목표 시간 설정
할 수 있는 페이지로
이동하는 버튼

오늘 일자를 기준으로
28일 전까지
볼 수 있습니다.

오늘로부터
최근 3주 동안의
공부량을 바탕으로 주간 무지개를
볼 수 있는 페이지로 이동하는 버튼

시간 설정 화면

목표 시간을 설정할 수 있는 화면입니다.
하루하루의 목표시간을 자신의 상황에 맞게 조절하세요.



‘설정한 시간 보기’ 버튼을 누르면,
위에서 설정한 시간으로
목표시간이 설정됩니다.

설정한 시간 보기

일자 화면

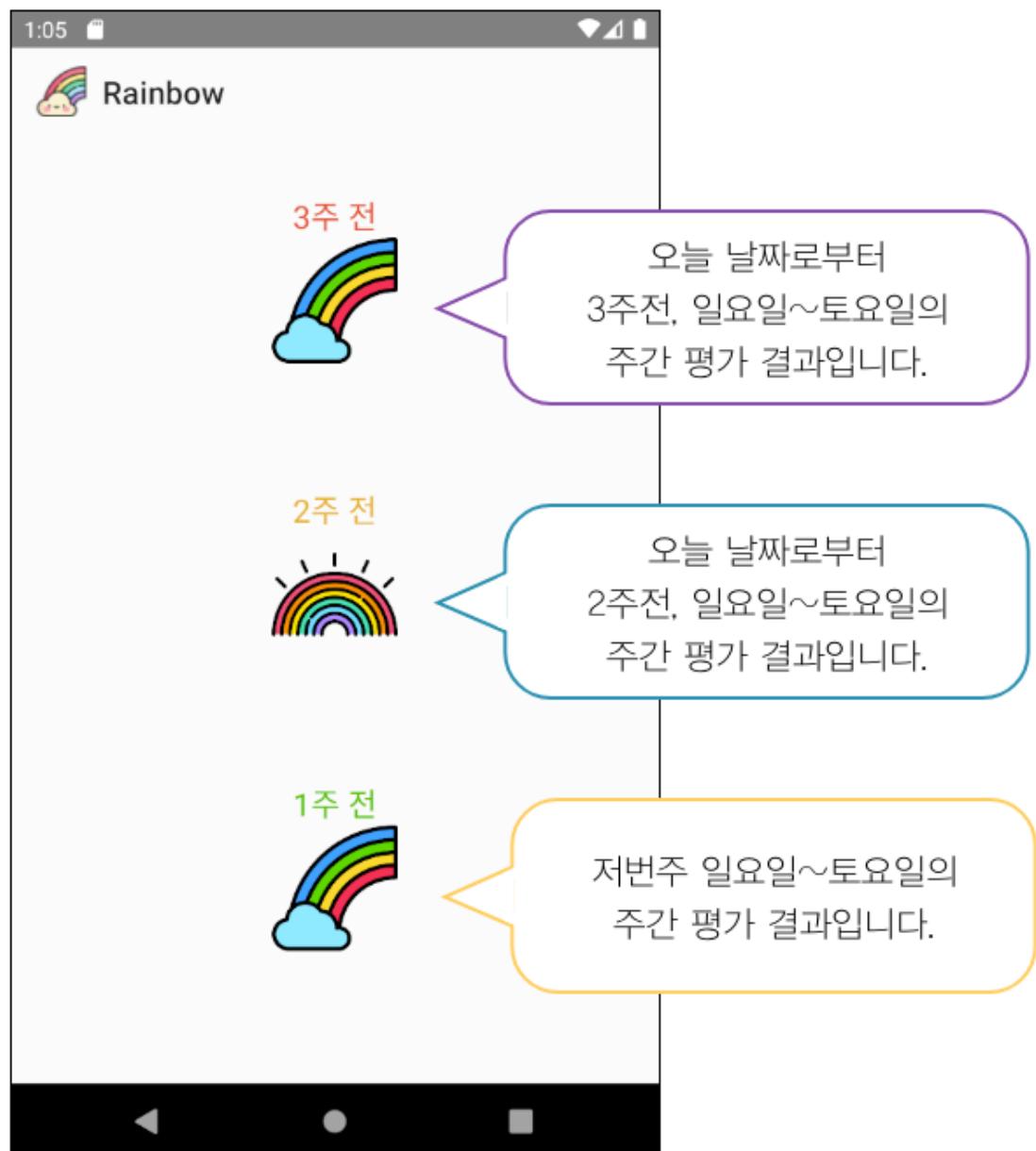
선택한 일자의 공부량과 목표량, 핸드폰 사용 횟수를 볼 수 있는 화면입니다.
_ 이 페이지를 통해 얼마나 공부 했었는지 확인해보세요



달력화면에서 일자페이지로 들어오는데 성공했다면,
선택한 일자, 해당 일자의 공부시간, 목표시간,
공부 중 핸드폰 사용 횟수를 볼 수 있습니다.

주간 평가 화면

공부량을 주간으로 평가해 볼 수 있는 페이지입니다.
이 페이지를 통해 얼마나 공부 했었는지 확인해보세요



사용자의 성취감을 위해 주간 평가를 준비했습니다.

일요일~토요일 기준으로 7일 모두 달성 시 완벽한 무지개를,

4~6일 달성 시 반 무지개를,

3일 이하 달성 시 구름 이미지를 받게 됩니다.