

# Backend-DB 구성

# DB History : 초안

## Before

- 1조, 4조에서 구현된 값 반영하여 제작
- 테이블은 모두 1:n 관계로 이루어짐(ID : health, sensors, details)
- useraccount table :** 유저 정보
- details table :** 유저 상세 정보
  - useraccount, details 테이블을 따로 생성한 이유 : useraccount(유저)만 만들고 details(사용자 입력. 즉, 측정)까지는 생성하지 않는 유저가 있을 것이라 예상
- health table :** 사용자의 건강 상태
  - health 테이블을 detail 테이블과 따로 둔 이유 : details(사용자 입력)값만 생성하고 health(출력)값은 생성하지 않는 유저 있을 것이라 예상
- sensors table :** 2조의 측정값 데이터

### 상세설명 - database 구성

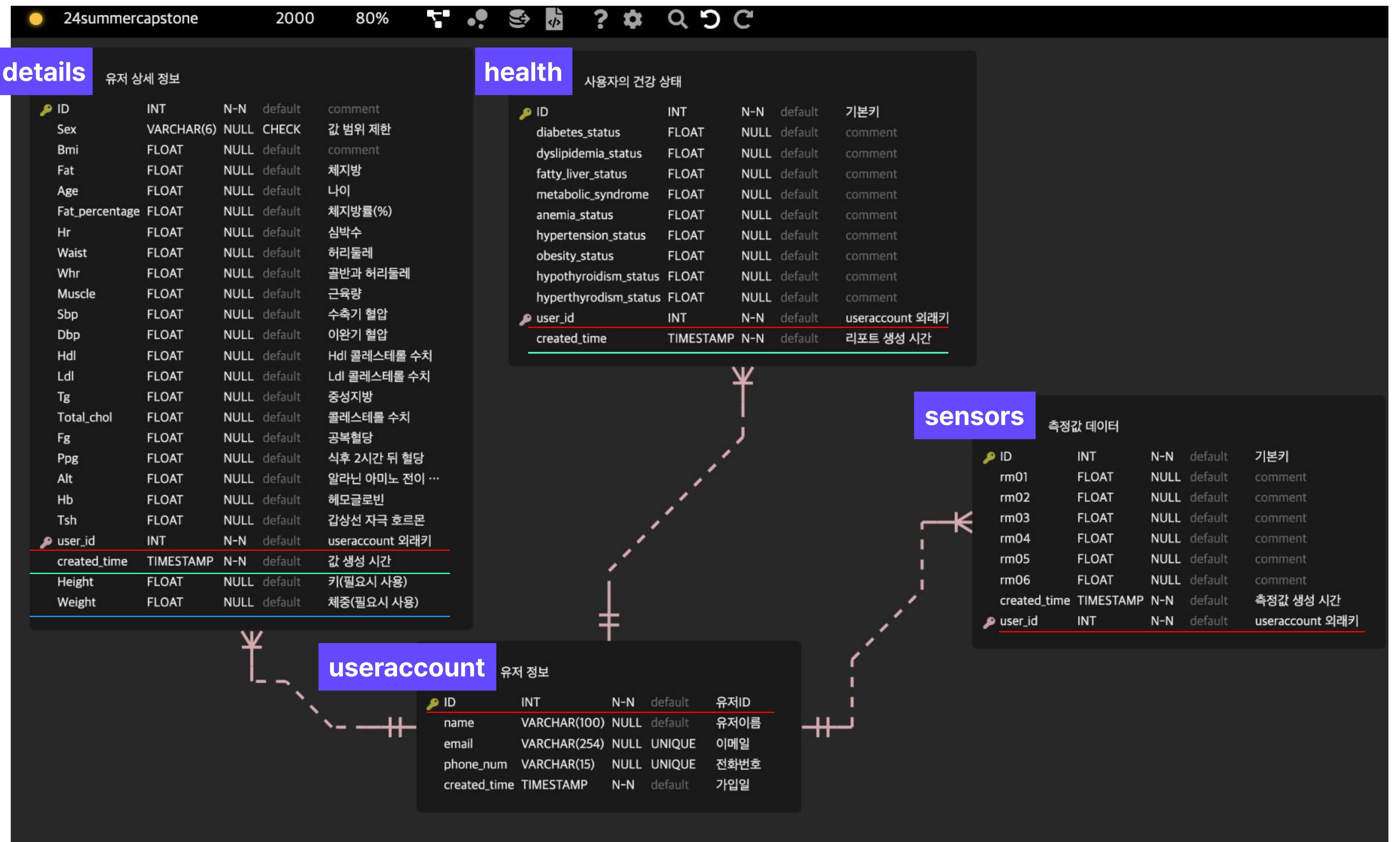
- details, health, sensors 테이블 등의 수치 속성은 FLOAT로 함 (추후 파일 합칠 때 혼동 방지를 위해)

### created\_time 속성 (모든 테이블에서)

- 언제 가입했는지, 언제 값을 입력했는지, 언제 리포트가 작성되었는지 등 표시 필요
  - TIMESTAMP 데이터 타입 : 데이터가 특정 이벤트 발생 시점에 자동으로 생성되는 날짜와 시간 정보를 저장함
  - 입력할 때 timestamp 찍어주도록 만들어준 상태 (최초 생성 시간)

### detail 테이블

- 1조에서 받아온 속성이 아닌 키(Height), 체중(Weight) 속성을 detail 테이블에 추가함
- 사용자의 기본 정보에 속할 것이라 예상했기 때문



# DB History : 최종

- 1조, 4조에서 구현된 값 반영하여 제작
- 테이블은 모두 1:n 관계로 이루어짐(ID : health, sensors, details)
- useraccount table의 기본키인 ID가 3개의 테이블의 user\_id이라는 외래키로 연결됨

- useraccount table : 유저 정보
- details table : 유저 상세 정보
  - useraccount, details 테이블을 따로 생성한 이유 : useraccount(유저)만 만들고 details(사용자 입력. 즉, 측정)까지는 생성하지 않는 유저가 있을 것이다 예상
- health table : 사용자의 건강 상태
  - health 테이블을 detail 테이블과 따로 둔 이유 : details(사용자 입력)값만 생성하고 health(출력)값은 생성하지 않는 유저 있을것이다 예상
- sensors table : 2조의 측정값 데이터
- recommend table(1조의 추천 문구 가져옴)

## 상세설명 - database 구성

- details, health, sensors 테이블 등의 수치 속성은 FLOAT로 함 (추후 파일 합칠때 혼동 방지를 위해)

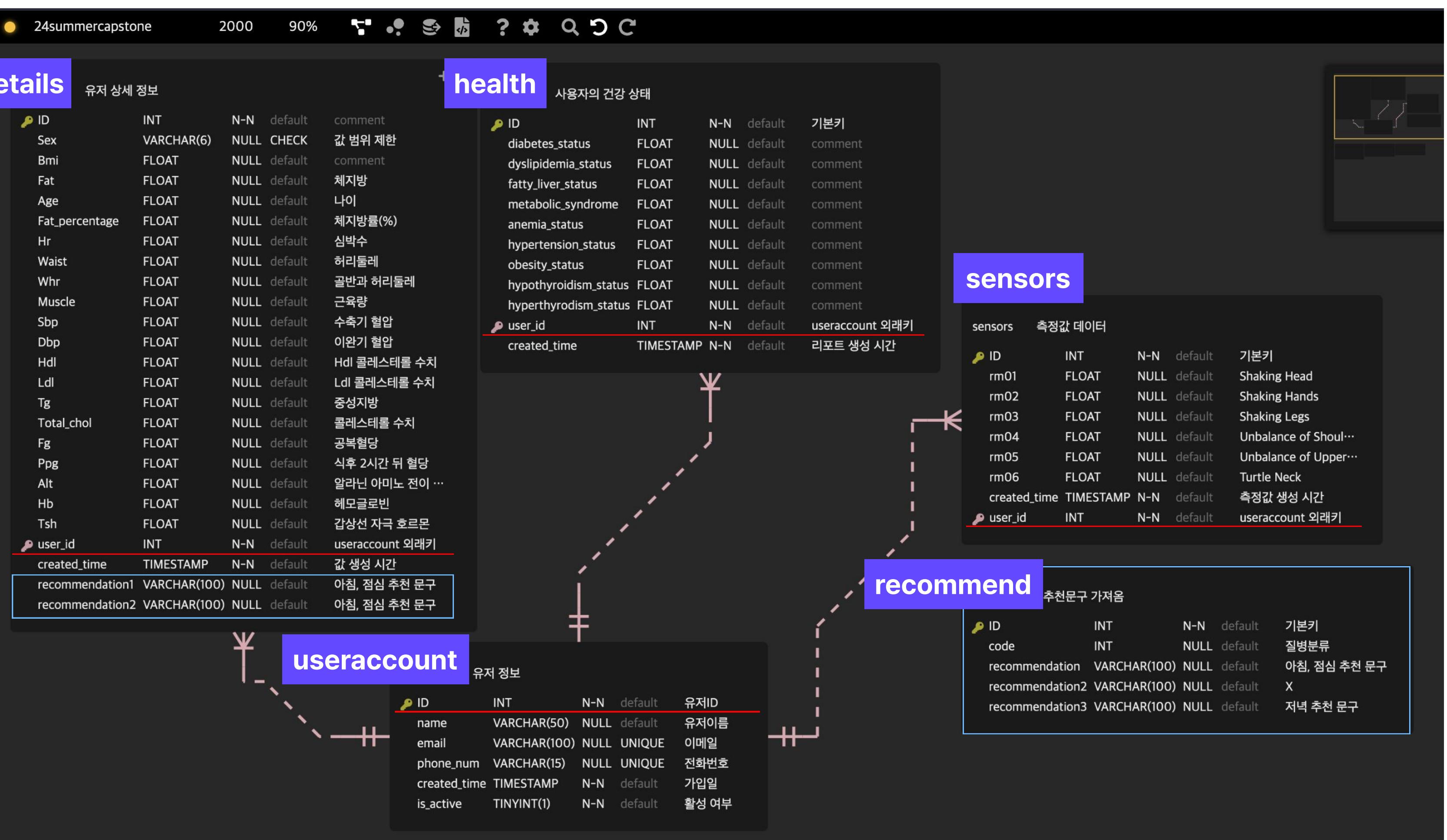
## created\_time 속성 (모든 테이블에서)

- 언제 가입했는지, 언제 값을 입력했는지, 언제 리포트가 작성되었는지 등 표시 필요
- TIMESTAMP 데이터 타입 : 데이터가 특정 이벤트 발생 시점에 자동으로 생성되는 날짜와 시간 정보를 저장함
- 입력할 때 timestamp 짹어주도록 만들어준 상태 (최초 생성 시간)

## detail 테이블

- 1조에서 받아온 속성이 아닌 키(Height), 체중(Weight) 속성을 detail 테이블에 추가함
- 사용자의 기본 정보에 속할 것이라 예상했기 때문

## After



# DB History : 초안 ~ 최종 변화

## Before

### useraccount 테이블:

- email이나 phone\_num 중 하나는 꼭 NOT NULL로, 하나는 반드시 입력되게 함
  - 이때, 두 속성 모두 UNIQUE으로 중복되지 않도록 함

```
CREATE TABLE useraccount (
    ID INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(254) UNIQUE,
    phone_num VARCHAR(15) UNIQUE,
    created_time TIMESTAMP NOT NULL,
    CHECK (email IS NOT NULL OR phone_num IS NOT NULL)
);
```

### details 테이블:

- Sex는 'Male', 'Female'이라는 값만 들어올 수 있도록 값 범위 제한함
- 값이 어떻게 들어올지 정확히 모르는 상태이므로 FLOAT로 받아올 수 있도록 함

```
CREATE TABLE details (
    ID INT PRIMARY KEY,
    user_id INT NOT NULL,
    Sex VARCHAR(6) CHECK (sex IN ('Male', 'Female')),
    Bmi FLOAT,
    Fat FLOAT,
    Age FLOAT,
    Fat_percentage FLOAT,
    Hr FLOAT,
    Waist FLOAT,
    Whr FLOAT,
    Muscle FLOAT,
    Sbp FLOAT,
    Dbp FLOAT,
    Hdl FLOAT,
    Ldl FLOAT,
    Tg FLOAT,
    Total_chol FLOAT,
    Fg FLOAT,
    Ppg FLOAT,
    Alt FLOAT,
    Hb FLOAT,
    Tsh FLOAT,
    Created_time TIMESTAMP NOT NULL,
    Height FLOAT,
    Weight FLOAT,
    FOREIGN KEY (user_id) REFERENCES useraccount(ID)
);
```

## After

### useraccount 테이블:

- email이나 phone\_num 중 하나는 꼭 NOT NULL로, 하나는 반드시 입력되게 함
  - 이때, 두 속성 모두 UNIQUE으로 중복되지 않도록 함
- 계정 활성상태는 Django login 기능 구현시 꼭 필요한 속성값

```
CREATE TABLE useraccount (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_num VARCHAR(15) UNIQUE,
    created_time TIMESTAMP NOT NULL,
    is_active TINYINT(1) NOT NULL,
    CHECK (email IS NOT NULL OR phone_num IS NOT NULL)
);
```

### details 테이블:

- Sex는 'Male', 'Female'이라는 값만 들어올 수 있도록 값 범위 제한함
- 값이 어떻게 들어올지 정확히 모르는 상태이므로 FLOAT로 받아올 수 있도록 함
- created\_time 변수 수정 - 백엔드 코드 작성의 용이성을 위해
  - 모든 테이블에서 똑같은 값(가장 최근 생성된 데이터)을 불러오기 위해

```
CREATE TABLE details (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    Sex VARCHAR(6) CHECK (sex IN ('Male', 'Female')),
    Bmi FLOAT,
    Fat FLOAT,
    Age FLOAT,
    Fat_percentage FLOAT,
    Hr FLOAT,
    Waist FLOAT,
    Whr FLOAT,
    Muscle FLOAT,
    Sbp FLOAT,
    Dbp FLOAT,
    Hdl FLOAT,
    Ldl FLOAT,
    Tg FLOAT,
    Total_chol FLOAT,
    Fg FLOAT,
    Ppg FLOAT,
    Alt FLOAT,
    Hb FLOAT,
    Tsh FLOAT,
    created_time TIMESTAMP NOT NULL,
    recommendation1 VARCHAR(100),
    recommendation2 VARCHAR(100),
    recommendation3 VARCHAR(100),
    metabolicper FLOAT,
    bodypoint FLOAT,
    FOREIGN KEY (user_id) REFERENCES useraccount(ID)
);
```

## No Changed

### health 테이블:

- 값이 어떻게 들어올지 정확히 모르는 상태이므로 FLOAT로 받아올 수 있도록 함

```
CREATE TABLE health (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    diabetes_status FLOAT,
    dyslipidemia_status FLOAT,
    fatty_liver_status FLOAT,
    metabolic_syndrome FLOAT,
    anemia_status FLOAT,
    hypertension_status FLOAT,
    obesity_status FLOAT,
    hypothyroidism_status FLOAT,
    hyperthyroidism_status FLOAT,
    created_time TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES useraccount(ID)
);
```

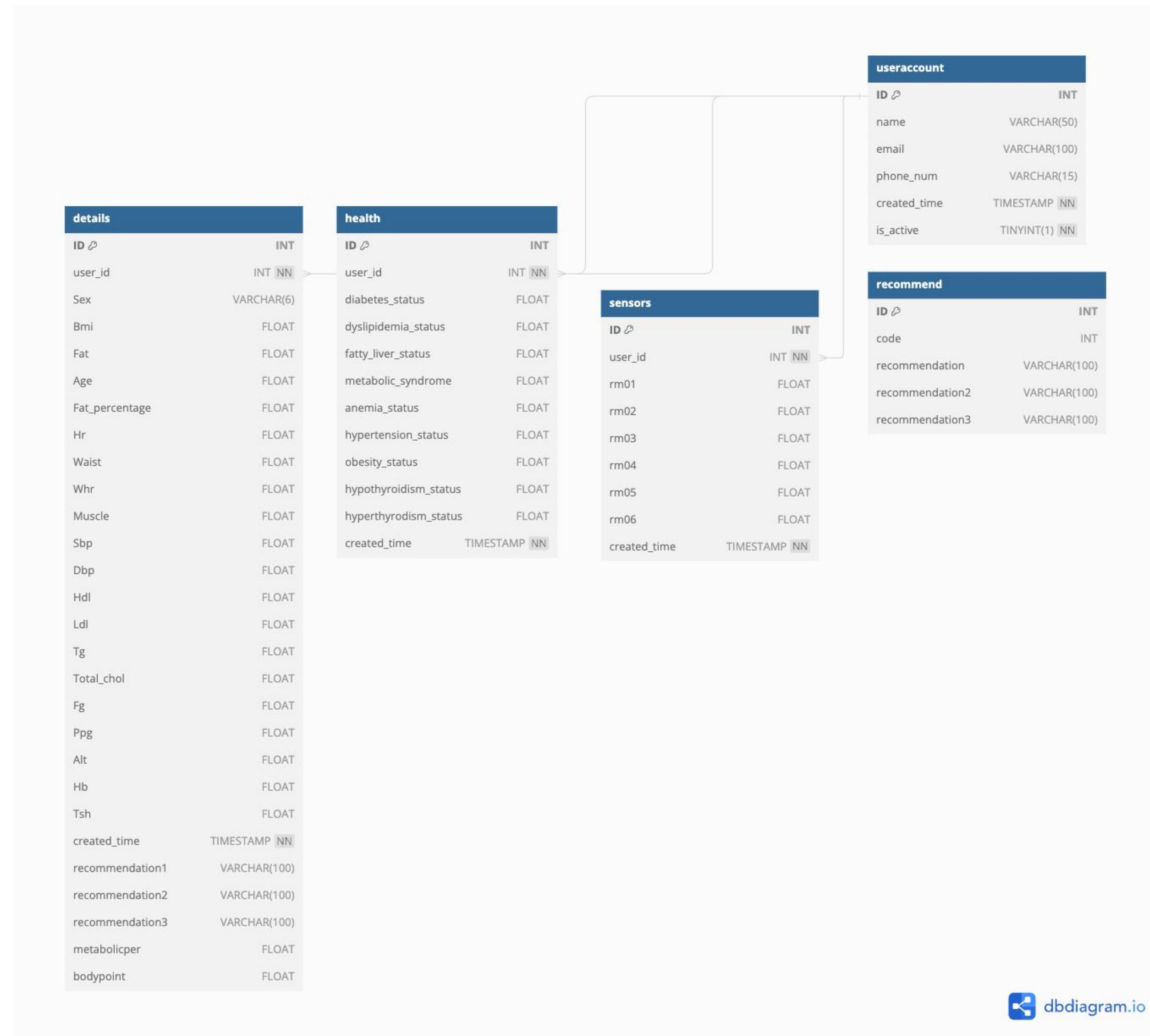
### sensors 테이블:

- 센서 측정값과 측정시간 저장

```
CREATE TABLE sensors (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    rm01 FLOAT,
    rm02 FLOAT,
    rm03 FLOAT,
    rm04 FLOAT,
    rm05 FLOAT,
    rm06 FLOAT,
    created_time TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES useraccount(ID)
);
```

# 최종 ERD

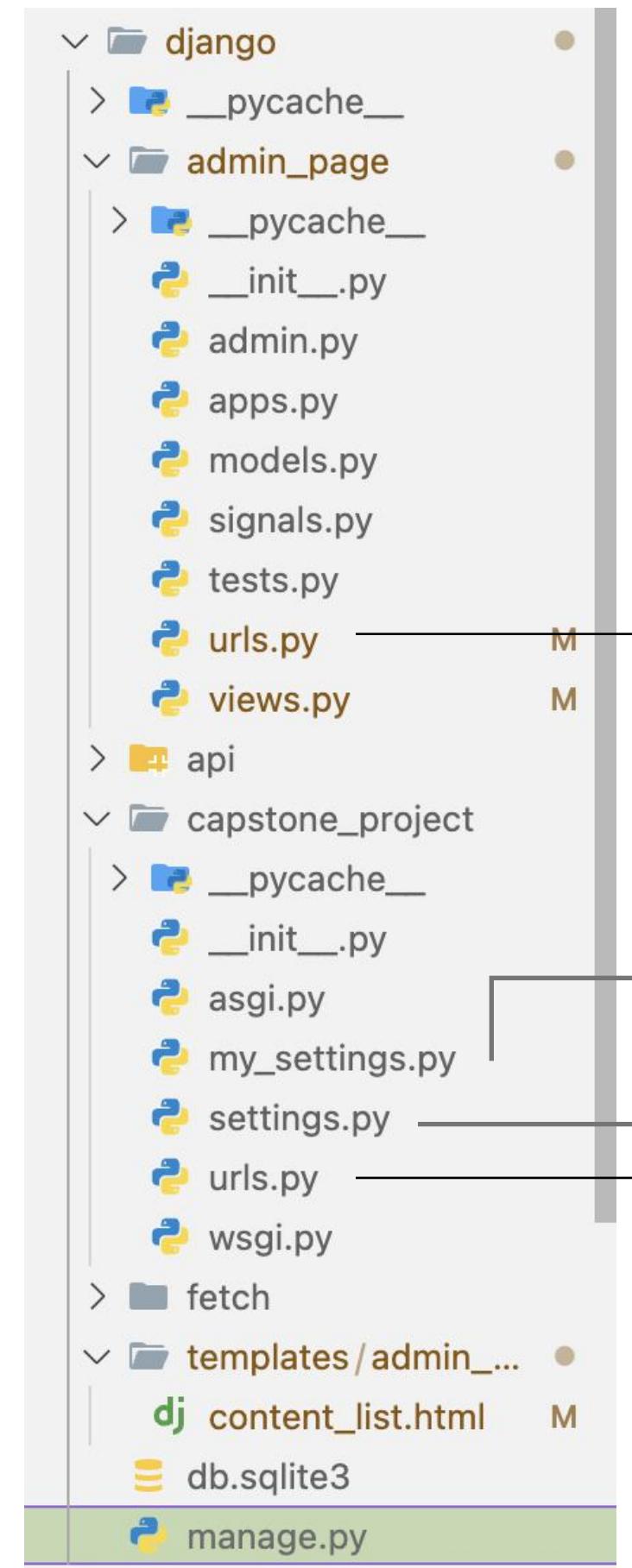
최종 생성된 DB는 다음과 같음



# Django 서버와 DB 연동 1

Django와 MariaDB 연결

## django 폴더구조



## capstone\_project/settings.py

```
django > capstone_project > 🐍 settings.py > ...
12
13 import os, sys
14 from datetime import datetime
15 from pathlib import Path
16 from datetime import timedelta
17 from .my_settings import mySECRET_KEY, myDATABASES
18
19 # Database
20 # https://docs.djangoproject.com/en/4.2/ref/settings/#databases
21 SECRET_KEY = mySECRET_KEY
22 DATABASES = myDATABASES
23
24 # Build paths inside the project like this: BASE_DIR / 'subdir'.
25 BASE_DIR = Path(__file__).resolve().parent.parent
26
27
28 # Quick-start development settings - unsuitable for production
29 # See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/
30
```

## capstone\_project/urls.py

```
django > capstone_project > 🐍 urls.py > ...
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('admin_page/', include('admin_page.urls')),
7     path('api/', include('api.urls')),
8     path('fetch/', include('fetch.urls'))
9 ]
```

## admin\_page/urls.py

```
django > admin_page > 🐍 urls.py > ...
1 from django.urls import path
2 from .views import combined_view
3
4 urlpatterns = [
5     path('', combined_view), # 127.0.0.1/admin_page에 들어오면 combined_view함수를 실행.
6 ]
7
```

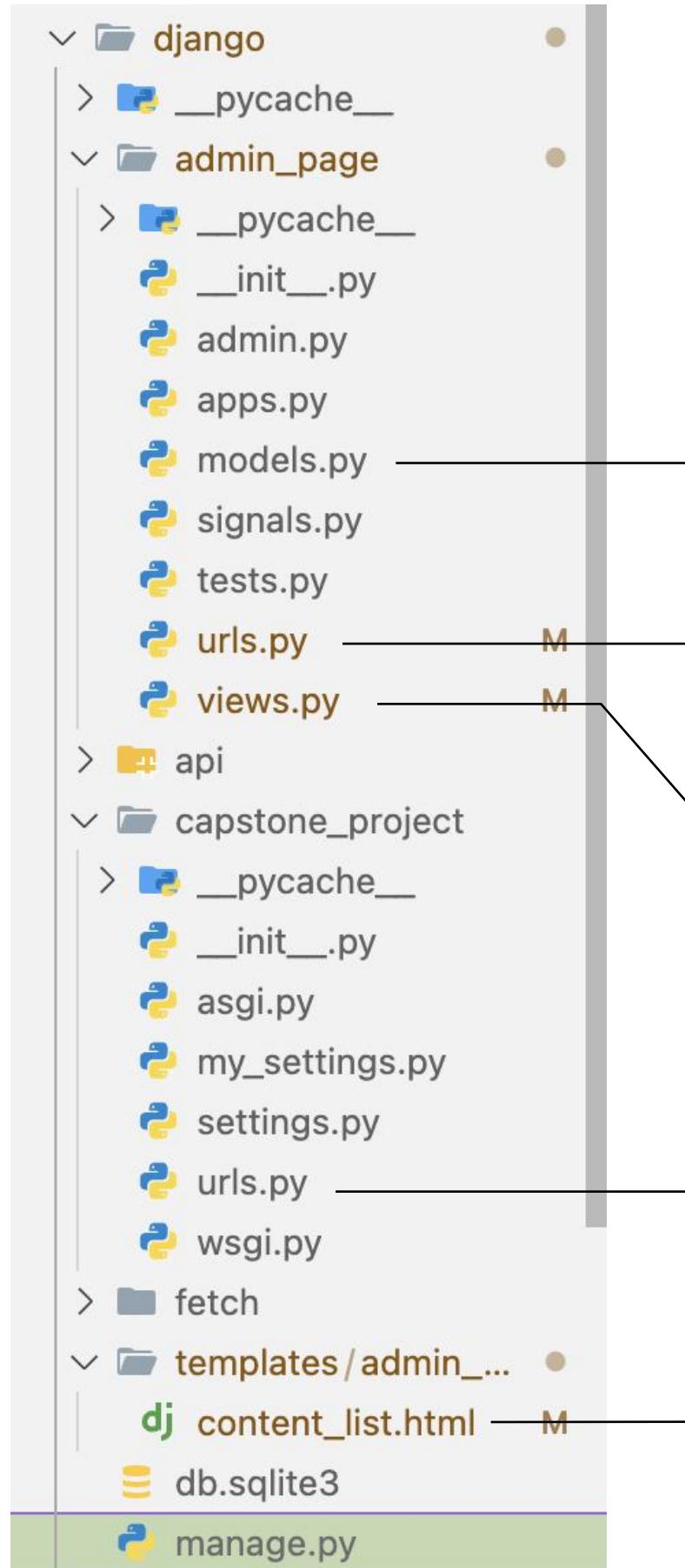
## capstone\_project/my\_settings.py

```
django > capstone_project > 🐍 my_settings.py > ...
1 mySECRET_KEY = 's!dl3qffp)_33!@#$!@*=5dp#sdd%^&*(jfi!@#$@#dfjskfl'
2
3 myDATABASES = [
4     'default' : {
5         'ENGINE' : 'django.db.backends.mysql', # 백엔드 엔진
6         'NAME' : 'capstonedb', # 데이터베이스 이름
7         'USER' : 'capstone', # 계정
8         'PASSWORD' : 'summer', # rootpassword로 지정할 숫자
9         '# HOST' : '127.0.0.1',
10        'HOST' : 'eloo.iptime.org',
11        'PORT' : '33306'
12    }
13]
14
```

# Django 서버와 DB 연동 2

## Django와 MariaDB 연결

### django 폴더구조



### admin\_page/models.py

```

django > admin_page > models.py > ...
1  from django.db import models
2
3  # Create your models here.
4  class UserAccount(models.Model):
5      id = models.AutoField(primary_key=True) # 기본 키, 자동 생성되는 필드
6      name = models.CharField(max_length=50)
7      email = models.EmailField(unique=True, null=True, blank=True) # 고유한, NULL 허용, 빈 문자열 허용
8      phone_num = models.CharField(max_length=15, unique=True, null=True, blank=True)
9      created_time = models.DateTimeField(auto_now_add=True) # 자동으로 현재 시간 기록
10     is_active = models.BooleanField(default=True)
11
12     # login() 함수에서 사용자를 식별하는 데 사용할 필드를 정의
13     USERNAME_FIELD = 'id'
14     REQUIRED_FIELDS = ['phone_num', 'email']
15     is_anonymous = False
16     is_authenticated = True
17     class Meta:
18         db_table = 'useraccount' # 실제 데이터베이스 테이블의 이름
19
20     def __str__(self):
21         return self.name
  
```

### admin\_page/views.py

```

django > admin_page > views.py > ...
1  from django.shortcuts import render
2  from .models import UserAccount, Details, Health, Sensors, Recommend
3
4  def combined_view(request):
5      users = UserAccount.objects.all().reverse()[:5]
6      details = Details.objects.all().reverse()[:5]
7      health = Health.objects.all().reverse()[:5]
8      sensors = Sensors.objects.all().reverse()[:5]
9      recommend = Recommend.objects.all().reverse()[:5]
10     context = {
11         'users': users,
12         'details': details,
13         'health': health,
14         'sensors': sensors,
15         'recommends': recommend
16     }
17
18     return render(request, 'admin_page/content_list.html', context)
  
```

### admin\_page/urls.py

```

django > admin_page > urls.py > ...
1  from django.urls import path
2  from .views import combined_view
3
4  urlpatterns = [
5      path('', combined_view), # 127.0.0.1/admin_page에 들어오면 combined_view함수를 실행.
6  ]
  
```

### capstone\_project/urls.py

```

django > capstone_project > urls.py > ...
1  from django.contrib import admin
2  from django.urls import path, include
3
4  urlpatterns = [
5      path('admin/', admin.site.urls),
6      path('admin_page/', include('admin_page.urls')),
7      path('api/', include('api.urls')),
8      path('fetch/', include('fetch.urls'))
9  ]
  
```

### templates/content\_list.html

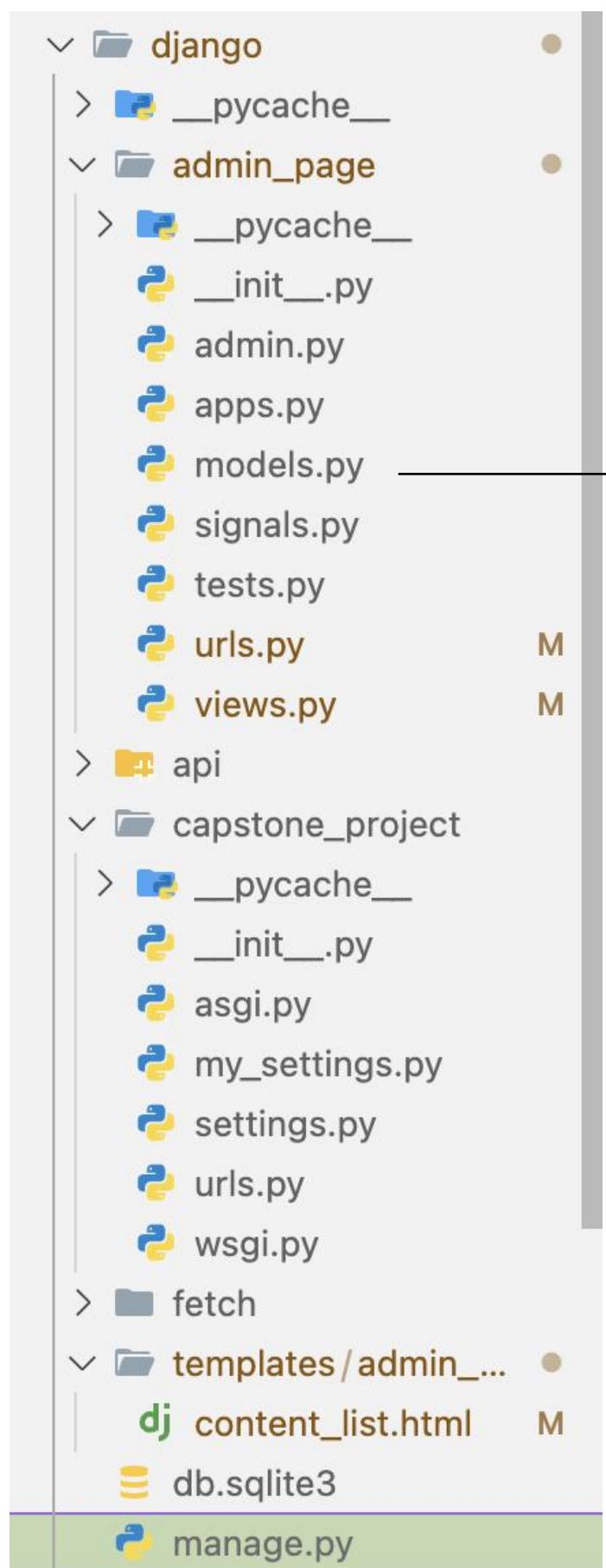
```

django > templates > admin_page > dj_content_list.html
2  <html>
3  </head>
4  <body>
5      <h1>Admin page : Data</h1>
6
7      <h2>Users</h2>
8      <table>
9          <thead>
10             <tr>
11                 <th>ID</th>
12                 <th>Name</th>
13                 <th>Email</th>
14                 <th>Phone Number</th>
15                 <th>Created Time</th>
16                 <th>Is Active</th>
17             </tr>
18         </thead>
19         <tbody>
20             <% for user in users %>
21                 <tr>
22                     <td>{{ user.id }}</td>
23                     <td>{{ user.name }}</td>
24                     <td>{{ user.email }}</td>
25                     <td>{{ user.phone_num }}</td>
26                     <td>{{ user.created_time }}</td>
27                     <td>{{ user.is_active }}</td>
28                 </tr>
29             <% empty %>
30             <tr>
31                 <td colspan="5">No recommendations found.</td>
32             </tr>
33         <% endfor %>
34     </tbody>
35 </table>
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
  
```

# Django 서버와 DB 연동 3

MariaDB에서 만들어준 table을 django와 연동

## django 폴더구조



## admin\_page/models.py

```
django > admin_page > models.py > ...
1  from django.db import models
2
3  # Create your models here.
4  class UserAccount(models.Model):
5      id = models.AutoField(primary_key=True) # 기본 키, 자동 생성되는 필드
6      name = models.CharField(max_length=50)
7      email = models.EmailField(unique=True, null=True, blank=True) # 고유한, NULL 허용, 빈 문자열 허용
8      phone_num = models.CharField(max_length=15, unique=True, null=True, blank=True)
9      created_time = models.DateTimeField(auto_now_add=True) # 자동으로 현재 시간 기록
10     is_active = models.BooleanField(default=True)
11
12     # login() 함수에서 사용자를 식별하는 데 사용할 필드를 정의
13     USERNAME_FIELD = 'id'
14     REQUIRED_FIELDS = ['phone_num', 'email']
15     is_anonymous = False
16     is_authenticated = True
17     class Meta:
18         db_table = 'useraccount' # 실제 데이터베이스 테이블의 이름
19
20     def __str__(self):
21         return self.name
```

```
24  class Details(models.Model):
25      id = models.AutoField(primary_key=True)
26      user = models.ForeignKey(UserAccount, on_delete=models.CASCADE) # 해당 사용자 계정이 삭제될 경우 관련된 Details 데이터도 함께 삭제됨
27      SEX_CHOICES = [
28          ('Male', 'Male'),
29          ('Female', 'Female'),
30      ]
31      sex = models.CharField(max_length=6, choices=SEX_CHOICES) # 필드에 사용할 수 있는 선택 옵션을 정의
32      bmi = models.FloatField(null=True, blank=True) # 필드에 대해 NULL 값을 허용, 빈 값도 허용
33      fat = models.FloatField(null=True, blank=True)
34      age = models.FloatField(null=True, blank=True)
35      fat_percentage = models.FloatField(null=True, blank=True)
36      hr = models.FloatField(null=True, blank=True)
37      waist = models.FloatField(null=True, blank=True)
38      whr = models.FloatField(null=True, blank=True)
39      muscle = models.FloatField(null=True, blank=True)
40      sbp = models.FloatField(null=True, blank=True)
41      dbp = models.FloatField(null=True, blank=True)
42      hdl = models.FloatField(null=True, blank=True)
43      ldl = models.FloatField(null=True, blank=True)
44      tg = models.FloatField(null=True, blank=True)
45      total_chol = models.FloatField(null=True, blank=True)
46      fg = models.FloatField(null=True, blank=True)
47      ppg = models.FloatField(null=True, blank=True)
48      alt = models.FloatField(null=True, blank=True)
49      hb = models.FloatField(null=True, blank=True)
50      tsh = models.FloatField(null=True, blank=True)
51      created_time = models.DateTimeField(auto_now_add=True)
52      recommendation1 = models.CharField(max_length=100, null=True, blank=True)
53      recommendation2 = models.CharField(max_length=100, null=True, blank=True)
54      recommendation3 = models.CharField(max_length=100, null=True, blank=True)
55      metabolicper = models.FloatField(null=True, blank=True)
56      bodypoint = models.FloatField(null=True, blank=True)
57
58     class Meta:
59         db_table = 'details'
60
61     def __str__(self):
62         return f"Details for {self.user.name}"
```

```
65  class Health(models.Model):
66      id = models.AutoField(primary_key=True)
67      user = models.ForeignKey(UserAccount, on_delete=models.CASCADE)
68      diabetes_status = models.FloatField(null=True, blank=True)
69      dyslipidemia_status = models.FloatField(null=True, blank=True)
70      fatty_liver_status = models.FloatField(null=True, blank=True)
71      metabolic_syndrome = models.FloatField(null=True, blank=True)
72      anemia_status = models.FloatField(null=True, blank=True)
73      hypertension_status = models.FloatField(null=True, blank=True)
74      obesity_status = models.FloatField(null=True, blank=True)
75      hypothyroidism_status = models.FloatField(null=True, blank=True)
76      hyperthyroidism_status = models.FloatField(null=True, blank=True)
77      created_time = models.DateTimeField(auto_now_add=True)
78
79     class Meta:
80         db_table = 'health'
81
82     def __str__(self):
83         return f"Health status for {self.user.name}"
```

```
86  class Sensors(models.Model):
87      id = models.AutoField(primary_key=True)
88      user = models.ForeignKey(UserAccount, on_delete=models.CASCADE)
89      rm01 = models.FloatField(null=True, blank=True)
90      rm02 = models.FloatField(null=True, blank=True)
91      rm03 = models.FloatField(null=True, blank=True)
92      rm04 = models.FloatField(null=True, blank=True)
93      rm05 = models.FloatField(null=True, blank=True)
94      rm06 = models.FloatField(null=True, blank=True)
95      created_time = models.DateTimeField(auto_now_add=True)
96
97     class Meta:
98         db_table = 'sensors'
99
100    def __str__(self):
101        return f"Sensors data for {self.user.name}"
102
103   class Recommend(models.Model):
104       id = models.AutoField(primary_key=True)
105       code = models.IntegerField()
106       recommendation = models.CharField(max_length=100)
107       recommendation2 = models.CharField(max_length=100)
108       recommendation3 = models.CharField(max_length=100)
109
110     class Meta:
111         db_table = 'recommend'
112
113     def __str__(self):
114         return f"Recommendation {self.id}"
```

# Django 서버에서 DB 조회 ([http://127.0.0.1:8000/admin\\_page/](http://127.0.0.1:8000/admin_page/))

# 담당. 윤가현

가장 최근에 들어온 값 5개만 출력되고 있음

http://127.0.0.1:8000/admin_page/																											
Admin page : Data																											
Users																											
ID	Name	Email	Phone Number									Created Time						Is Active									
7	이은영	jiyeong21@example.net	5535493602.0									2024년 7월 16일 6:03 오전						True									
8	김정호	jieun38@example.net	None									2024년 7월 16일 6:03 오전						True									
9	홍우진	None	1715563835.0									2024년 7월 16일 6:03 오전						False									
10	백정수	jeongnammun@example.com	None									2024년 7월 16일 6:03 오전						False									
11	이현숙	jeongjagim@example.com	4419483584.0									2024년 7월 16일 6:03 오전						True									
Details																											
ID	User ID	Sex	BMI	Fat	Age	Fat Percentage	Hr	Waist	Whr	Muscle	Sbp	Dbp	Hdl	Ldl	Tg	Total Chol	Fg	Ppg	Alt	Hb	Tsh	Created Time	Recommendation1	Recommendation2	Recommendation3	Metabolic Per	Body Point
1	7	Female	26.0131	23.9122	36.4621	25.8801	115.313	83.6614	0.617999	31.2344	178.954	110.07	86.6031	81.0593	219.438	197.691	74.948	177.463	46.7164	10.299	1.46597	2024년 7월 17일 12:21 오전	철분이 많이 함유된 통곡물 시리얼과 딸기를 함께 섭취하시고, 우유보다는 아몬드 밀크를 드시는 것이 좋습니다.	철분이 많이 함유된 콩나물 무침과 비타민 C가 풍부한 브로콜리를 함께 드세요.	주기적으로 건강 검진을 받아 조기 발견과 예방에 신경 쓰세요.	80.0	44.0
2	8	Female	31.9263	44.8682	52.3814	32.6547	76.7771	97.6337	0.818637	43.2449	97.4686	118.152	35.837	63.0934	283.698	165.712	190.447	180.236	30.0046	14.2164	9.82473	2024년 7월 17일 12:21 오전	통곡물 시리얼과 딸기를 함께 드시길 권장드립니다.	피로를 풀기 위해 커피 대신 카페인이 적은 허브차를 드시는 것을 추천드립니다. 허브차는 진정 효과가 있습니다.	가벼운 운동을 일상 생활에 통합해 보세요. 예를 들어, 엘리베이터 대신 계단을 이용하세요.	80.0	56.0
3	9	Male	17.6018	6.55808	20.588	28.9801	64.6237	70.8779	0.697193	31.6634	164.445	71.1866	84.2303	142.078	141.246	243.363	168.741	283.188	147.675	16.2914	5.55445	2024년 7월 17일 12:23 오전	식사 후 디저트로는 케이크 대신 과일 샐러드를 준비해보세요. 과일 샐러드는 비타민과 항산화제가 풍부합니다.	겸기나 수영과 같은 저Impact 운동을 통해 체력을 유지하고 철분을 효과적으로 이용하세요.	채소와 과일을 충분히 섭취하여 비타민과 미네랄을 보충하세요.	40.0	56.0
4	10	Female	39.4269	6.41237	42.7464	29.1176	55.3377	84.1231	0.801808	71.9808	169.06	92.2518	77.7005	111.098	267.962	278.424	123.777	174.651	61.1299	14.2264	7.44959	2024년 7월 17일 2:12 오전	닭 가슴살과 녹색 채소를 곁들인 퀴노아 샐러드를 드시고, 허브차나 자몽 티를 마시는 것을 추천드립니다.	잡곡밥과 함께 시금치와 토마토를 곁들인 가벼운 훈제 연어를 섭취하시고, 물 대신 철분 흡수를 돋는 레몬 워터를 마시는 것이 좋습니다.	하루에 정해진 시간에 규칙적으로 식사하세요.	80.0	50.0
5	11	Female	39.2835	23.741	24.7794	44.192	100.107	64.8089	0.654005	56.6494	96.0523	65.1622	78.972	80.7295	245.743	272.651	78.4195	86.071	63.2669	13.3634	6.41685	2024년 7월 16일 6:07 오전	None	None	None	None	None
Health																											
ID	User ID	User	Diabetes Status	Dyslipidemia Status	Fatty Liver Status	Metabolic Syndrome	Anemia Status	Hypertension Status	Obesity Status	Hypothyroidism Status	Hyperthyroidism Status	Created Time															
1	7	이은영	0.6	0.5	0.1	0.7	0.4	0.5	0.8	0.3	0.9	2024년 7월 16일 6:28 오전															
2	8	김정호	0.6	0.5	0.8	0.1	0.1	0.3	0.2	0.8	0.9	2024년 7월 16일 6:28 오전															
3	9	홍우진	0.8	0.5	0.9	0.2	0.4	0.6	0.6	0.2	0.8	2024년 7월 16일 6:28 오전															
4	10	백정수	0.3	0.9	0.1	0.4	0.3	0.2	0.1	1.0	0.9	2024년 7월 16일 6:28 오전															
5	7	이은영	0.6	0.7	0.4	0.9	0.5	0.5	0.0	0.1	0.7	2024년 7월 16일 6:28 오전															
Sensors																											
ID	User_ID	User	RM01	RM02	RM03	RM04	RM05	RM06	created_time																		
1	7	이은영	19.7478	91.7965	47.1301	98.555	97.337	33.3646	2024년 7월 16일 6:38 오전																		
2	7	이은영	55.6083	97.7639	57.5334	35.7703	40.0632	61.3334	2024년 7월 16일 6:38 오전																		
3	8	김정호	61.3843	83.3297	38.7355	50.6452	71.5321	76.5503	2024년 7월 16일 6:38 오전																		
4	9	홍우진	41.6941	12.8262	94.417	96.3954	66.8366	28.5828	2024년 7월 16일 6:38 오전																		
5	10	백정수	25.3725	64.4507	98.4532	8.78608	7.86166	77.0714	2024년 7월 16일 6:38 오전																		
Recommendations																											
ID	Code	Recommendation	Recommendation2									Recommendation3															
1169	0	단백질이 풍부한 계란 오믈렛과 함께 비타민이 많은 신선한 과일을 드시기를 권장드립니다.	“규칙적인 식사와 충분한 수면이 건강 유지에 중요합니다.”									규칙적인 식사와 충분한 수면이 건강 유지에 중요합니다.															
1170	0	흰 쌀밥 대신 섬유질이 풍부한 퀴노아 샐러드를 선택해보세요.	“식사를 거르지 말고 꼭챙겨 드세요. 하루를 시작하는 좋은 에너지원이 됩니다.”									식사를 거르지 말고 꼭챙겨 드세요. 하루를 시작하는 좋은 에너지원이 됩니다.															
1171	0	포화지방이 적은 닭가슴살과 오메가-3가 풍부한 연어를 함께 드시기를 추천드립니다.	“매일 30분 정도 가벼운 산책을 해보세요. 심신이 상쾌해집니다.”									매일 30분 정도 가벼운 산책을 해보세요. 심신이 상쾌해집니다.															
1172	0	고당도의 과자 대신 비타민 C가 많은 오렌지를 드셔보세요.	“물을 충분히 마시고, 카페인 섭취는 줄이는 것이 좋습니다.”									물을 충분히 마시고, 카페인 섭취는 줄이는 것이 좋습니다.															
1173	0	주스 대신에 당분이 적고 칼로리가 낮은 녹차를 마셔보세요.	“스트레칭을 자주 해주세요. 근육의 긴장을 풀어주고 피로를 예방합니다.”									스트레칭을 자주 해주세요. 근육의 긴장을 풀어주고 피로를 예방합니다.															

# Chat GPT 검토 (질문)

입력 프롬프트 1 : 다음의 SQL 쿼리에 대해 DB 생성이 충돌없이 잘 되었는지 평가해줘.

```
∅ 다음의 SQL 쿼리에 대해 DB 생성이 충돌없이 잘 되었는지 평가해줘.

CREATE TABLE useraccount (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_num VARCHAR(15) UNIQUE,
    created_time TIMESTAMP NOT NULL,
    is_active TINYINT(1) NOT NULL,
    CHECK (email IS NOT NULL OR phone_num IS NOT NULL)
);

CREATE TABLE details (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    Sex VARCHAR(6) CHECK (sex IN ('Male', 'Female')),
    Bmi FLOAT,
    Fat FLOAT,
    Age FLOAT,
    Fat_percentage FLOAT,
    Hr FLOAT,
    Waist FLOAT,
    Whr FLOAT,
    Muscle FLOAT,
    Sbp FLOAT,
    Dbp F↓ T,
    HdL FLOAT
);
```

입력 프롬프트 2 : 다음의 SQL 쿼리에 대해 DB 생성이 충돌없이 잘 되었는지 채점해서 100점 만점으로 점수 매겨줘.

```
다음의 SQL 쿼리에 대해 DB 생성이 충돌없이 잘 되었는지 채점해서 100점 만점으로 점수 매겨줘.

CREATE TABLE useraccount (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_num VARCHAR(15) UNIQUE,
    created_time TIMESTAMP NOT NULL,
    is_active TINYINT(1) NOT NULL,
    CHECK (email IS NOT NULL OR phone_num IS NOT NULL)
);

CREATE TABLE details (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    Sex VARCHAR(6) CHECK (sex IN ('Male', 'Female')),
    Bmi FLOAT,
    Fat FLOAT,
    Age FLOAT,
    Fat_percentage FLOAT,
    Hr FLOAT,
    Waist FLOAT,
    Whr FLOAT,
    Muscle FLOAT
);
```

검토 링크 1 : <https://chatgpt.com/share/3ab76160-f49f-4ffc-9c0b-d5f53627c6fc>

검토 링크 2 : <https://chatgpt.com/share/5b6d5ca0-916d-4869-b6ff-c78a9b8e3f54>

# Chat GPT 검토 (답변)

프롬프트 1에 대한 답변 : DB의 모든 Table이 논리적 충돌 없이 잘 구성됨

테이블: `recommend`

sql

```
CREATE TABLE recommend (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    code INT,
    recommendation VARCHAR(100),
    recommendation2 VARCHAR(100),
    recommendation3 VARCHAR(100)
);
```

- `ID`는 기본 키로 자동 증가 설정이 잘 되어 있음.

**종합 평가**

모든 테이블이 기본 키와 자동 증가, 외래 키 제약 조건, 데이터 타입 및 크기, NOT NULL 제약 조건, 그리고 CHECK 제약 조건 등을 잘 설정하고 있습니다. 현재 제공된 SQL 쿼리들은 논리적 충돌 없이 잘 구성되어 있는 것으로 보입니다.

프롬프트 2에 대한 답변 : 구조와 제약 조건, 참조 무결성을 종합적으로 고려할 때, 이 SQL 쿼리는 충돌 없이 잘 작동할 것으로 보임. 이에 따라, 100점 만점에 95점을 줌

## 종합 평가

- 모든 테이블에 PRIMARY KEY와 AUTO\_INCREMENT 설정이 올바르게 되어 있습니다.
- 각 테이블 간의 FOREIGN KEY 관계가 잘 설정되어 있어 참조 무결성이 보장됩니다.
- 데이터 타입과 제약 조건이 적절하게 설정되어 있습니다.
- 테이블 간의 관계와 데이터 구조가 잘 설계되어 있습니다.

## 점수

구조와 제약 조건, 참조 무결성을 종합적으로 고려할 때, 이 SQL 쿼리는 충돌 없이 잘 작동할 것으로 보입니다. 이에 따라, 100점 만점에 95점을 주겠습니다.

## 개선 제안

- details 테이블의 일부 컬럼들에 대해서도 NOT NULL 제약 조건을 추가하여 데이터의 완전성을 보장할 수 있습니다.
- 필요하다면, 추가적인 인덱스를 생성하여 검색 성능을 향상시킬 수 있습니다.

검토 링크 1 : <https://chatgpt.com/share/3ab76160-f49f-4ffc-9c0b-d5f53627c6fc>

검토 링크 2 : <https://chatgpt.com/share/5b6d5ca0-916d-4869-b6ff-c78a9b8e3f54>

# Github Repository 생성

<https://github.com/kimeloo/24SummerCapstone>

 **24SummerCapstone** Public

 main  4 Branches  0 Tags

 Go to file  Add file  Code

 **kimeloo** Merge branch 'main' of github.com:kimeloo/24SummerCapstone into main 1b6b76b · 3 hours ago  160 Commits

	MOD : details 테이블 수정	13 hours ago
	FIX : SERVER Recommendation 문구 기능 추가 완료	3 hours ago
	TEST : example 추가	yesterday
	FIX : CLIENT Recommendation 문구 시간대 추가	3 hours ago
	MOD : django 파일에 admin, models, views 추가. FIX : settin...	4 days ago
	Update Readme.md	13 hours ago
	MOD : details 테이블 수정	13 hours ago
	ADD : 파일 트리구조로 보기	12 hours ago
	FIX : django model에 recommend 데이터베이스 지정, view와 t...	yesterday

# Django 서버 구성

## URL 구조 및 기능

### ServerUrl /

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('admin_page/', include('admin_page.urls')),
    path('api/', include('api.urls')),
    path('fetch/', include('fetch.urls'))
]
```

로그인 및 메일 전송 담당 ( api / )

```
urlpatterns = [
    path('token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('login/', LoginView, name='login'),
    path('email/', sendEmail, name='email'),
]
```

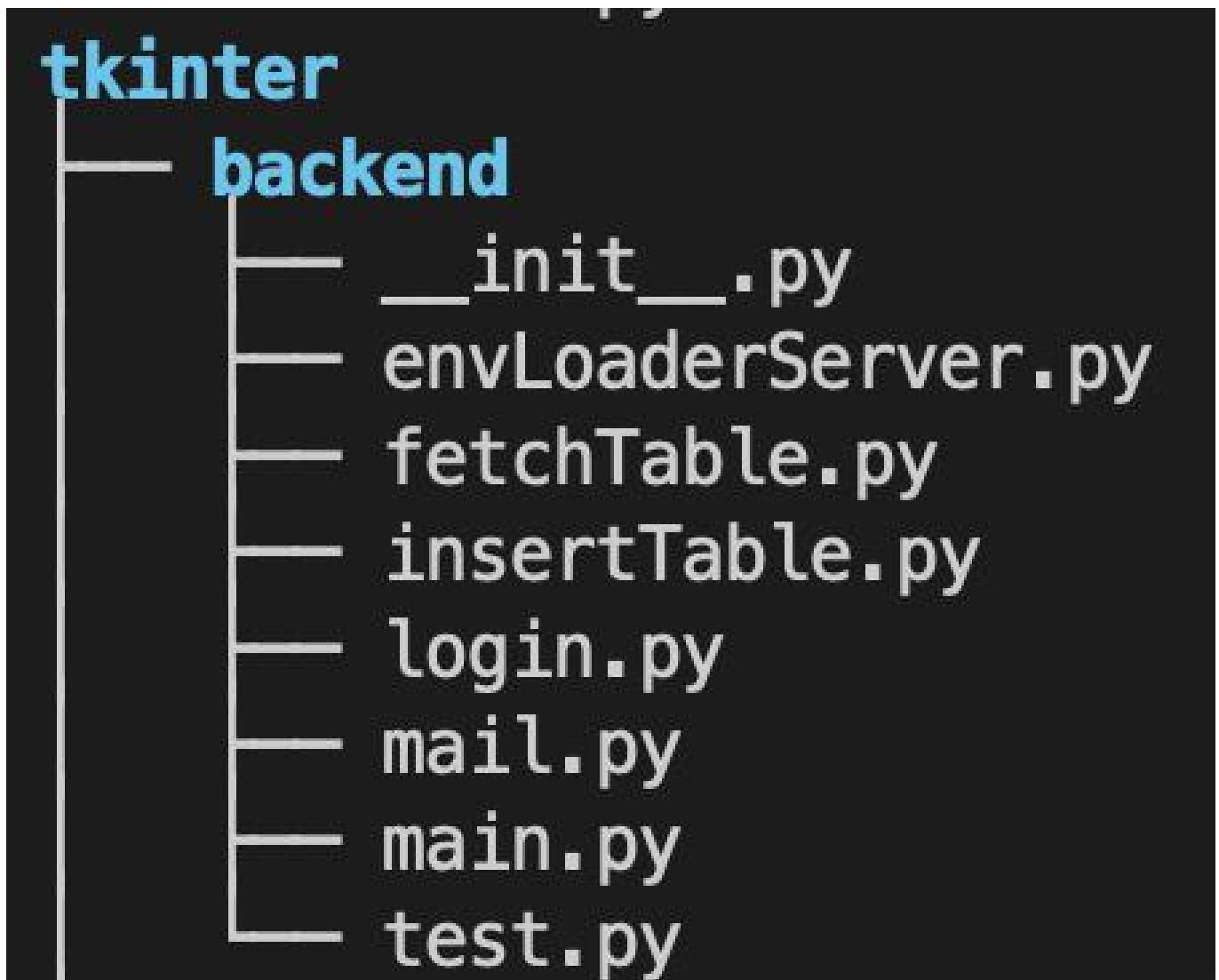
DB 조회 및 추가 담당 ( fetch / )

```
urlpatterns = [
    path('select/<str:table>/', TableView, name='fetchTable'),
    path('insert/<str:table>/', TableInsert, name='insertTable'),
]
```

# Jetson 클라이언트 구성

## 파일 구조 및 기능

### 파일 구조



### UI ⇌ Client 연결 코드

```

19  class ConnectUI():
20      '''frontend UI/UX와 연결'''
21      def __init__(self):
22          self.server = ConnectServer()
23
24      def login(self, phone=None, email=None):
25          loginStatus = self.server.login(phone=phone, email=email)
26          if loginStatus:
27              logger.info('Login Success!')
28          else:
29              logger.error('Login Failed.')
30
31
32      def fromUI(self, table=str(), data=dict()):
33          '''frontend에서 호출하여, 데이터를 backend로 전송'''
34          logger.debug(f'Fetched data:table={table}\tdata={data}')
35          result = self.server.toServer(table, data)
36          if result!=False:
37              logger.info('Data transmission success!')
38              return result
39          else:
39              logger.error('Data transmission failed.')
40              return '결과 전송 실패.\nData transmission failed.'
41
42
43      def toUI(self, table=str(), columns=list()):
44          '''frontend에서 호출하여, DB 데이터를 UI로 전송'''
45          logger.debug(f'Request data:table={table}\tcolumns={columns}')
46          result = self.server.fromServer(table)
47          if result!=False:
48              logger.info('Data reception success!')
49              filteredByCols = {key:result[key] for key in columns if key in result}
50              logger.info('Data filtering success!')
51              logger.debug(f'Filtered data:table={table}\tdata={filteredByCols}')
52              return filteredByCols
53          else:
54              logger.error('Data reception failed.')
55              return {'error':'No Data'}
56
57      def sendEmail(self):
58          if self.server.sendEmail():
59              logger.info('Email delivery success!')
60          else:
61              logger.error('Email delivery failed.')

```

### Client ⇌ Server 연결 코드

```

63  class ConnectServer():
64      '''서버 데이터 송수신'''
65      def __init__(self):
66          self.token = None
67          self.refToken = None
68
69      def login(self, phone, email):
70          '''서버에 로그인'''
71          try:
72              self.loginServer = Login.JwtAuthClient(SERVER_HOST)
73              self.loginServer.login(phone_number=phone, email_address=email)
74              self.token, self.refToken = self.loginServer.get_token()
75              return True
76          except:
77              return False
78
79      def refreshLogin(self):
80          self.token = self.loginServer.refresh_token()
81
82      def toServer(self, table, data):
83          '''서버로 데이터 전송'''
84          try:
85              insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
86              result = insertServer.insertTable(table, data)
87              if result == 401:
88                  self.refreshLogin()
89                  insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
90                  result = insertServer.insertTable(table, data)
91              if 'success' in result:
92                  return result['returnList']
93          except:
94              pass
95          return False
96
97      def fromServer(self, table):
98          try:
99              fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
100             result = fetchServer.fetchTable(table)
101             if result == 401:
102                 self.refreshLogin()
103                 fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
104                 result = fetchServer.fetchTable(table)
105             if 'error' in result:
106                 raise
107                 return result
108         except:
109             return False

```

# 로그인 구현

전화번호 or 메일주소 로그인 (JWT 활용)

UI

```
backend.login(phone=phone_number) backend.login(email=email_address)
```

Jetson 클라이언트

```
def login(self, phone=None, email=None):
    loginStatus = self.server.login(phone=phone, email=email)
    if loginStatus:
        logger.info('Login Success!')
    else:
        logger.error('Login Failed.')

def login(self, phone, email):
    '''서버에 로그인'''
    try:
        self.loginServer = Login.JwtAuthClient(SERVER_HOST)
        self.loginServer.login(phone_number=phone, email_address=email)
        self.token, self.refToken = self.loginServer.get_token()
        return True
    except:
        return False

def login(self, phone_number=None, email_address=None):
    data = {
        'phone_num' : phone_number,
        'email' : email_address
    }
    try:
        response = requests.post(self.serverUrl + '/api/login/', data=data)
        response_data = response.json()
        if response.status_code == 200:
            # 로그인 요청 결과로 받은 JWT 토큰은 클래스 내의 jwt_token 속성에 저장됨
            self.jwt_token = response_data.get('token')
            self.ref_token = response_data.get('refresh_token')
            self.user_id = response_data.get('ID')
            print("로그인 성공!")
        elif response.status_code == 401:
            self.jwt_token = self.refresh_token()
        else:
            print("로그인 실패:", response_data.get('message'))
    except requests.exceptions.RequestException as e:
        print("요청 중 오류 발생:", e)
```

Django 서버

```
@csrf_exempt
def LoginView(request):
    if request.method == 'POST':
        phone_num = request.POST.get('phone_num')
        email = request.POST.get('email')
        if not(phone_num or email):
            logger.error(f'User failed to log in. phone_num: {phone_num}, email: {email}')
            return JsonResponse({'error': 'POST phone_num or email'}, status=400)

        # 사용자 검색
        try:
            user = UserAccount.objects.get(phone_num=phone_num, email=email)
        except UserAccount.DoesNotExist:
            user = UserAccount.objects.create(phone_num=phone_num, email=email)
            user.save()

        # JWT 토큰 생성
        access_token = AccessToken.for_user(user)
        refresh_token = RefreshToken.for_user(user)

        # 사용자 로그인 처리
        login(request, user)
        logger.info(f'User {user.id} logged in successfully.')
        return JsonResponse({'ID': user.id, 'token':str(access_token), 'refresh_token':str(refresh_token)}, status=200)
    else:
        return JsonResponse({'error': 'Method not allowed'}, status=405)
```

DB UserAccount 테이블

# 로그인 구현

전화번호 or 메일주소 로그인 (JWT 활용)

## 클라이언트 UI

전화번호를 입력하세요.

010 12345678

1	2	3
4	5	6
7	8	9
0	←	

뒤로가기

검사 시작

이메일을 입력하세요.

example@example.com

직접 입력

@

뒤로가기

검사 시작

## 서버 Log

```
2024-07-17 03:46:32,188 - INFO - User 15 logged in successfully.
[17/Jul/2024 03:46:32] "POST /api/login/ HTTP/1.1" 200 504
```

## DB UserAccount 테이블

14		None	01012345678	2024년 7월 16일 7:37 오후	True
15		None	[REDACTED]	2024년 7월 16일 7:42 오후	True

# User 입력값 DB로 전송

RM07, RM08

UI

```
recommendations = backend.fromUI('details', physicalData)
```

Jetson 클라이언트

```
def fromUI(self, table=str(), data=dict()):
    '''frontend에서 호출하여, 데이터를 backend로 전송'''
    logger.debug(f'Fetched data:table={table}\tdata={data}')
    result = self.server.toServer(table, data)
    if result!=False:
        logger.info('Data transmission success!')
        return result
    else:
        logger.error('Data transmission failed.')
        return '결과 전송 실패.\nData transmission failed.'
```

```
def toServer(self, table, data):
    '''서버로 데이터 전송'''
    try:
        insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
        result = insertServer.insertTable(table, data)
        if result == 401:
            self.refreshLogin()
            insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
            result = insertServer.insertTable(table, data)
        if 'success' in result:
            return result['returnList']
    except:
        pass
    return False
```

```
class InsertTable:
    def __init__(self, token, serverUrl):
        self.serverUrl = serverUrl
        self.headers = {
            'Authorization': f'Bearer {token}'
        }

    def insertTable(self, table, data=dict()):
        self.Endpoint = f'/fetch/insert/{table}/'
        url = self.serverUrl+self.Endpoint

        response = requests.post(url, headers=self.headers, data=data)
        if response.status_code == 401:
            return 401
        return response.json()
```

Django 서버

```
@api_view(['POST'])
@authentication_classes([JWTAuthentication])
def TableInsert(request, table):
    if request.method == 'POST':
        user = request.user
        logger.info(f'User {user.id} requested to insert data.')
        if InsertOrUpdate(table, user, request.data):
            logger.error(user.id)
            returnList = rm0708.main(user_id=user.id) ← 1조 코드 실행지점
            # returnList = ['(TEST 문구) ○○○ 항목의 수치 관리가 필요합니다.']
            return Response({'success': 'Data inserted successfully', 'returnList': returnList}, status=status.HTTP_200_OK)
        else:
            return Response({'error': 'Data not inserted'}, status=status.HTTP_404_NOT_FOUND)
    else:
        return Response({'error': 'Method not allowed'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
```

```
def InsertOrUpdate(table, user, data):
    Table = toTable(table)
    data = dict(data)
    for key in data:
        if type(data[key])==list:
            data[key] = ', '.join(data[key])
    try:
        latest = Table.objects.filter(user=user).order_by('-created_time').first()
        if latest:
            latestTime = datetime.fromisoformat(str(latest.created_time)[:19])
            currentTime = datetime.fromisoformat(str(datetime.now())[:19])
            timeDiff = currentTime - latestTime
            if timeDiff <= timedelta(hours=2):
                latest.__dict__.update(**data)
                latest.save()
                return True
        except:
            pass
    Table.objects.create(user=user, **data)
    return True
```

DB Details 테이블

# User 입력값 DB로 전송

RM07, RM08

## 클라이언트 UI

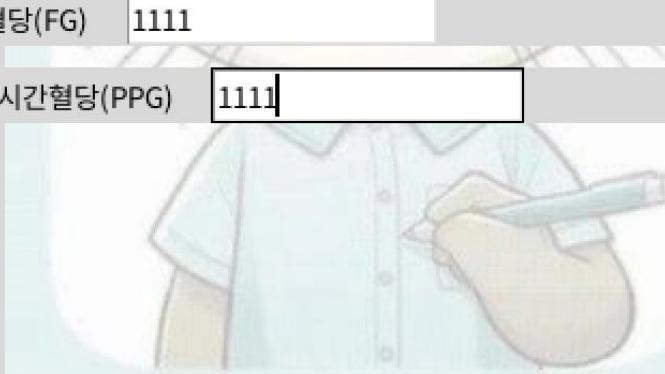
신체 측정 정보를 입력하세요.

나이	9999
성별	Male
키(cm)	9999
체중(kg)	9999
체지방지수(BMI)	9999
체지방(Kg)	9999
체지방률(%)	9999
심장박동수(bpm)	9999
허리둘레(cm)	9999
골반과 허리둘레(WHR)	9999
근육량(kg)	9999
수축기 혈압(SBP)	9999
이완기 혈압(DBP)	9999

[뒤로가기](#) [결과보기](#)

혈액 측정 정보를 입력하세요.

저밀도콜레스테롤(LDL)	1111
고밀도콜레스테롤(HDL)	1111
총콜레스테롤	1111
중성지방(TG)	1111
알라닌아미노전이효소(ALT)	1111
헤모글로빈(Hb)	1111
갑상선자극호르몬(TSH)	1111
공복혈당(FG)	1111
식후2시간혈당(PPG)	1111



[뒤로가기](#) [결과보기](#)

## 서버 Log

2024-07-17 03:47:29,327 - INFO - User 15 requested to insert data.

[17/Jul/2024 03:47:29] "POST /fetch/insert/details/ HTTP/1.1" 200 271

2024-07-17 03:49:24,867 - INFO - User 15 requested to insert data.

[17/Jul/2024 03:49:25] "POST /fetch/insert/details/ HTTP/1.1" 200 433

## DB Details 테이블

27	15	Male	9999.0	9999.0	9999.0	9999.0	9999.0	9999.0	9999.0	9999.0	9999.0	1111.0	1111.0	1111.0

# User 입력값 반환

RM07, RM08

## 클라이언트 UI

사용자의 입력값:

나이: 9999  
성별: Male  
키(cm): 9999  
체중(kg): 9999

신체 측정 결과:

고혈압에 대한 관리가 필요합니다.  
비만에 대한 관리가 필요합니다.  
대사증후군에 대한 관리가 필요합니다.  
빈혈에 대한 관리가 필요합니다.

검사 재선택      결과 전송

사용자의 입력값:

저밀도콜레스테롤(LDL): 1111  
고밀도콜레스테롤(HDL): 1111  
총콜레스테롤: 1111  
중성지방(TG): 1111

혈액 측정 결과:

고혈압에 대한 관리가 필요합니다.  
비만에 대한 관리가 필요합니다.  
당뇨병에 대한 관리가 필요합니다.  
이상지질혈증에 대한 관리가 필요합니다.  
지방간에 대한 관리가 필요합니다.  
대사증후군에 대한 관리가 필요합니다.  
갑상선기능저하증에 대한 관리가 필요합니다.

검사 재선택      결과 전송

## 서버 Log

2024-07-17 03:49:24,954 - INFO - 고혈압에 대한 관리가 필요합니다.  
비만에 대한 관리가 필요합니다.  
당뇨병에 대한 관리가 필요합니다.  
이상지질혈증에 대한 관리가 필요합니다.  
지방간에 대한 관리가 필요합니다.  
대사증후군에 대한 관리가 필요합니다.  
  
갑상선기능저하증에 대한 관리가 필요합니다.

# DB에서 UI로 전송

R&R Report, email 발송

Django 서버

DB Details 테이블

```
@api_view(['POST'])
@authentication_classes([JWTAuthentication])
def TableView(request, table):
    if request.method == 'POST':
        user = request.user
        logger.info(f'User {user.id} requested {table} Table.')
        fromDB = returnValue(table, user)
        if fromDB:
            return Response(fromDB, status=status.HTTP_200_OK)
        else:
            return Response({'error': 'Data not found'}, status=status.HTTP_404_NOT_FOUND)
    else:
        return Response({'error': 'Method not allowed'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
```

```
@api_view(['GET'])
@authentication_classes([JWTAuthentication])
def sendEmail(request):
    if request.method == 'GET':
        user = request.user
        logger.info(f'User {user.id} requested to send an email.')
        body, address = mailBody(user)
        if body==False:
            return JsonResponse({'error': 'No email address'}, status=400)

        email = EmailMessage('PROJECT NAME R&R Report', body, to=address)
        email.send()
        return JsonResponse({'message': 'Email sent successfully'}, status=200)
```

Jetson 클라이언트

```
def fetchTable(self, table):
    self.Endpoint = f'/fetch/select/{table}/'
    url = self.serverUrl+self.Endpoint

    response = requests.post(url, headers=self.headers)
    if response.status_code == 401:
        return 401
    return response.json()
```

```
def fromServer(self, table):
    try:
        fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
        result = fetchServer.fetchTable(table)
        if result == 401:
            self.refreshLogin()
            fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
            result = fetchServer.fetchTable(table)
        if 'error' in result:
            raise
        return result
    except:
        return False
```

```
def toUI(self, table=str(), columns=list()):
    '''frontend에서 호출하여, DB 데이터를 UI로 전송'''
    logger.debug(f'Request dataTable={table}\tcolumns={columns}')
    result = self.server.fromServer(table)
    if result!=False:
        logger.info('Data reception success!')
        filteredByCols = {key:result[key] for key in columns if key in result}
        logger.info('Data filtering success!')
        logger.debug(f'Filtered data:table={table}\tdata={filteredByCols}')
        return filteredByCols
    else:
        logger.error('Data reception failed.')
        return {'error': 'No Data'}
```

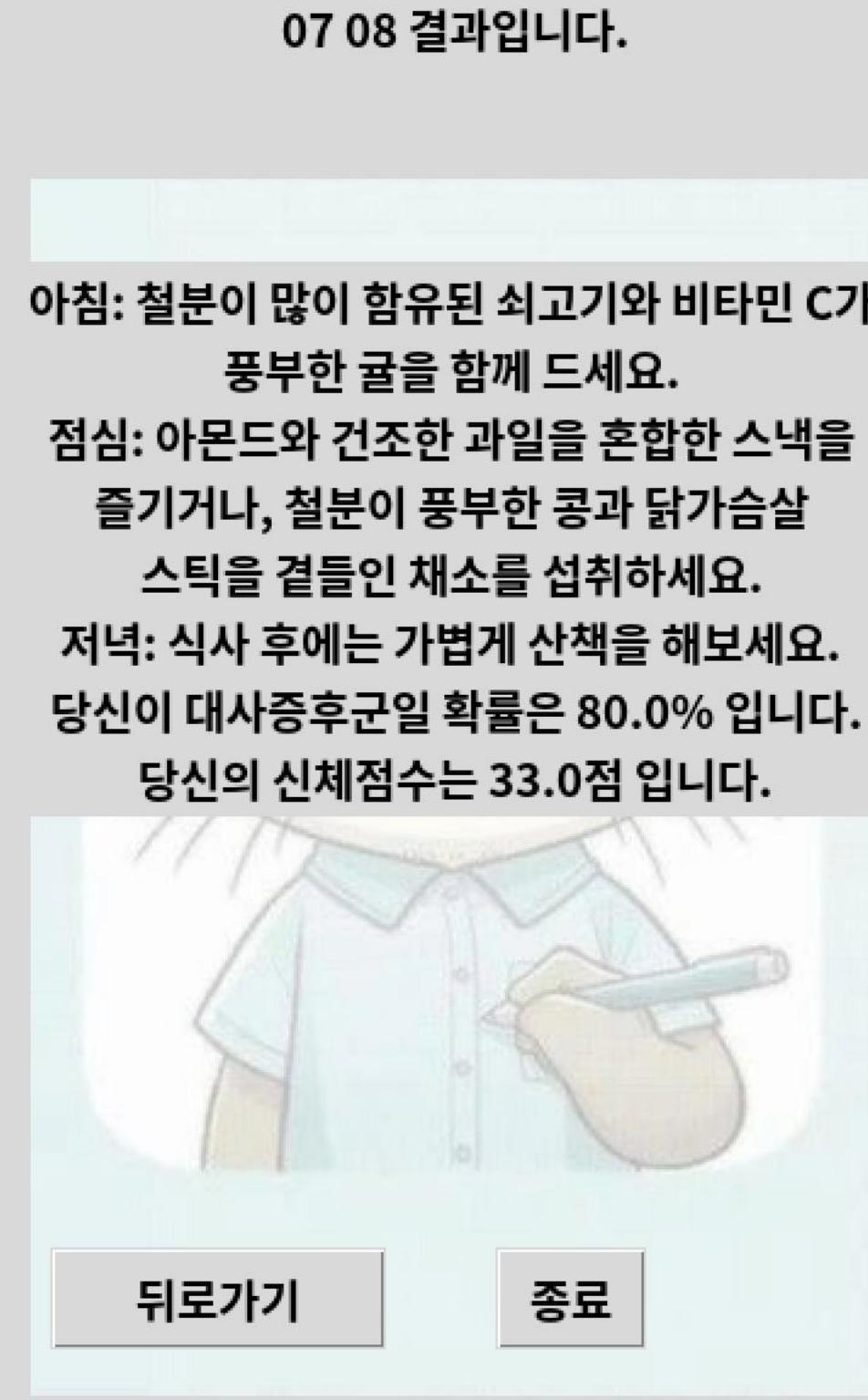
UI

```
recommendationsFromServer = backend.toUI('details', ['recommendation1', 'recommendation2', 'recommendation3', 'metabolicper', 'bodypoint'])
recommendTextSet = ""
recommendTextSet += f"아침: {recommendTextSplitter(str(recommendationsFromServer['recommendation1']))}\n"
recommendTextSet += f"점심: {recommendTextSplitter(str(recommendationsFromServer['recommendation2']))}\n"
recommendTextSet += f"저녁: {recommendTextSplitter(str(recommendationsFromServer['recommendation3']))}\n"
recommendTextSet += f"당신이 대사증후군일 확률은 {str(recommendationsFromServer['metabolicper'])}% 입니다.\n"
recommendTextSet += f"당신의 신체점수는 {str(recommendationsFromServer['bodypoint'])}점입니다."
recommendText.set(recommendTextSet)
backend.sendEmail()
```

# DB에서 UI로 전송

R&R Report, email 발송

## 클라이언트 UI



07 08 결과입니다.

아침: 철분이 많이 함유된 쇠고기와 비타민 C가 풍부한 귤을 함께 드세요.

점심: 아몬드와 건조한 과일을 혼합한 스낵을 즐기거나, 철분이 풍부한 콩과 닭가슴살 스틱을 곁들인 채소를 섭취하세요.

저녁: 식사 후에는 가볍게 산책을 해보세요.

당신이 대사증후군일 확률은 80.0%입니다.

당신의 신체점수는 33.0점입니다.

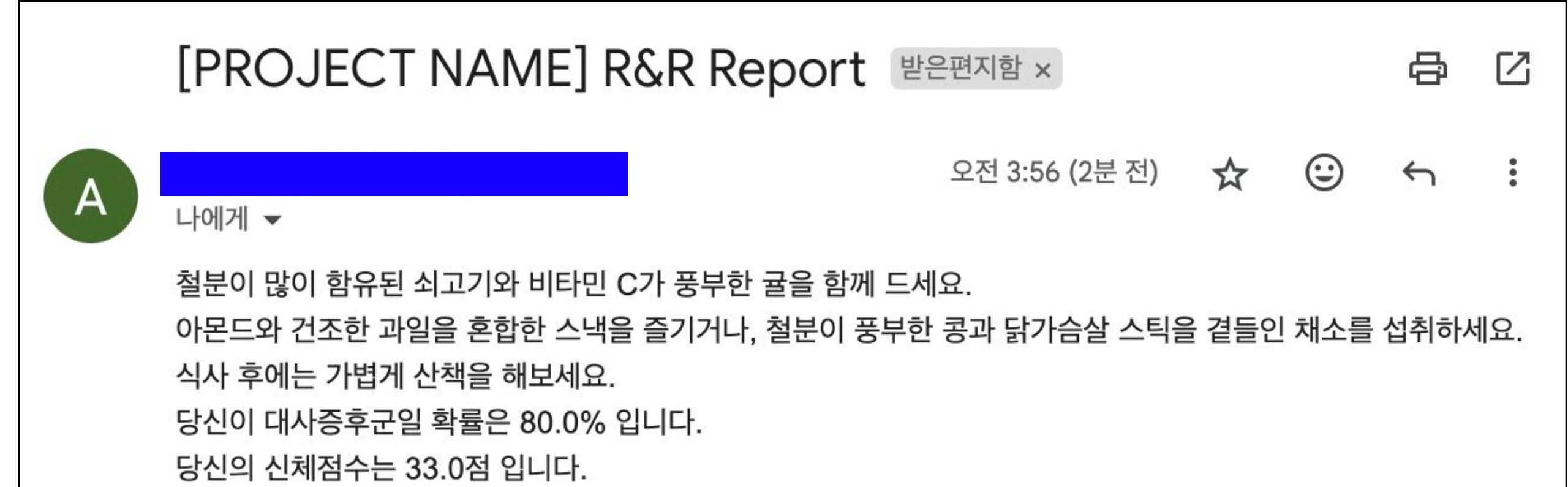
**A**

뒤로가기 종료

## 서버 Log

```
2024-07-17 03:56:18,327 - INFO - User 15 requested details Table.
[17/Jul/2024 03:56:18] "POST /fetch/select/details/ HTTP/1.1" 200 690
2024-07-17 03:56:18,429 - INFO - User 15 requested to send an email.
[17/Jul/2024 03:56:22] "GET /api/email/ HTTP/1.1" 200 38
```

## e-mail 수신



[PROJECT NAME] R&R Report

발은편지함 ×

오전 3:56 (2분 전)

A [REDACTED]

나에게 ▾

철분이 많이 함유된 쇠고기와 비타민 C가 풍부한 귤을 함께 드세요.  
아몬드와 건조한 과일을 혼합한 스낵을 즐기거나, 철분이 풍부한 콩과 닭가슴살 스틱을 곁들인 채소를 섭취하세요.  
식사 후에는 가볍게 산책을 해보세요.  
당신이 대사증후군일 확률은 80.0%입니다.  
당신의 신체점수는 33.0점입니다.

# Chat GPT 검토

<https://chatgpt.com/share/ac8e9139-d982-450e-bf3f-dba1f7541e11>

캡스톤디자인 과목 수강생의 코드를 채점해줘. UI로부터 데이터를 받아온 클라이언트가 서버를 통해 DB와 소통하는 코드를 작성했어. 100점 만점으로 채점하여 보여줘.

<클라이언트>

```
class ConnectUI():
    """frontend UI/UX와 연결"""
    def __init__(self):
        self.server = ConnectServer()

    def login(self, phone=None, email=None):
        loginStatus = self.server.login(phone=phone, email=email)
        if loginStatus:
            logger.info('Login Success!')
        else:
            logger.error('Login Failed.')

    def fromUI(self, table=str(), data=dict()):
        """frontend에서 호출하여, 데이터를 backend로 전송"""
        logger.debug(f'Fetched data:table={table}\tdata={data}')
        result = self.server.toServer(table, data)
        if result!=False:
            logger.info('Data transmission success!')
            return result
        else:
            logger.error('Data transmission failed.')
            return '결과 전송 실패.\nData transmission failed.'

    def toUI(self, table=str(), columns=list()):
        """frontend에서 호출하여, DB 데이터를 UI로 전송"""
        logger.debug(f'Request data:table={table}\tcolumns={columns}')
        result = self.server.fromServer(table)
        if result!=False:
            logger.info('Data reception success!')
            filteredByCols = {key:result[key] for key in columns if key in result}
            logger.info('Data filtering success!')
            logger.debug(f'Filtered data:table={table}\tdata={filteredByCols}')
            return filteredByCols
        else:
            logger.error('Data reception failed.')
            return {'error':'No Data'}

    def sendEmail(self):
        if self.server.sendEmail():
            logger.info('Email delivery success!')
        else:
            logger.error('Email delivery failed.')

def loginServer():
    """서버 데이터 송수신"""
    def __init__(self):
        self.token = None
        self.refToken = None

    def login(self, phone, email):
        """서버에 로그인"""
        try:
            self.loginServer = Login.JwtAuthClient(SERVER_HOST)
            self.loginServer.login(phone_number=phone, email_address=email)
            self.token, self.refToken = self.loginServer.get_token()
            return True
        except:
            return False

    def refreshLogin(self):
        self.token = self.loginServer.refresh_token()

    def toServer(self, table, data):
        """서버로 데이터 전송"""
        try:
            insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
            result = insertServer.insertTable(table, data)
            if result == 401:
                self.refreshLogin()
                insertServer = InsertTable.InsertTable(self.token, SERVER_HOST)
                result = insertServer.insertTable(table, data)
            if 'success' in result:
                return result['returnList']
            except:
                pass
            return False
        except:
            pass
            return False

    def fromServer(self, table):
        try:
            fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
            result = fetchServer.fetchTable(table)
            if result == 401:
                self.refreshLogin()
                fetchServer = FetchTable.FetchTable(self.token, SERVER_HOST)
                result = fetchServer.fetchTable(table)
            if 'error' in result:
                raise
            return result
            except:
                return False
        except:
            pass
            return False

    def sendEmail(self):
        emailServer = Mail.Email(self.token, SERVER_HOST)
        result = emailServer.email()
        if result==401:
            self.refreshLogin()
            emailServer = Mail.Email(self.token, SERVER_HOST)
            result = emailServer.email()
        return result
```

```
<서버>
from django.contrib.auth import login
from django.contrib.views.decorators.csrf import csrf_exempt
from admin_page.models import UserAccount
from rest_framework_simplejwt.tokens import AccessToken, RefreshToken
from rest_framework.decorators import api_view, authentication_classes
from rest_framework.authentication import JWTAuthentication
from django.core.mail import EmailMessage
from admin_page.models import Details
from django.forms.models import model_to_dict
import logging

logger = logging.getLogger(__name__)

@csrf_exempt
def LoginView(request):
    if request.method == 'POST':
        phone = request.POST.get('phone_num')
        email = request.POST.get('email')
        if not phone or not email:
            logger.error('User failed to log in. phone_num: (phone_num), email: (email)')
            return JsonResponse({'error': 'POST phone_num or email'}, status=400)
        # 사용자 검색
        user = UserAccount.objects.get(phone_num=phone_num, email=email)
        except UserAccount.DoesNotExist:
            user = UserAccount.objects.create(phone_num=phone_num, email=email)
            user.save()
        # JWT 토큰 생성
        access_token = AccessToken.for_user(user)
        refresh_token = RefreshToken.for_user(user)
        # 사용자 로그인 처리
        login(request, user)
        logger.info(f'User (user.id) logged in successfully.')
        return JsonResponse({'D': user.id, 'token': str(access_token), 'refresh_token': str(refresh_token)}, status=200)
    else:
        return JsonResponse({'error': 'Method not allowed'}, status=405)

def multiBody(user):
    userAddress = user.email
    data = model_to_dict(Details.objects.filter(user_id=user.id).order_by('-created_time').first())
    body = f'[data][recommendation1]]\n[body][recommendation2]]\n[body][recommendation3]]\n당신이 대사증후군일 확률은 (data["metabolicper"])입니다.\n당신의 신체점수는 (data["bodypoint"])입니다.'
    if userAddress == None:
        return False, None
    return True, userAddress

@api_view(['GET'])
@authentication_classes([JWTAuthentication])
def sendEmail(request):
    if request.method == 'GET':
        user = request.user
        logger.info(f'User (user.id) requested to send an email.')
        body, address = multiBody(user)
        if body == None:
            return JsonResponse({'error': 'No email address'}, status=400)
        email = EmailMessage(
            [PROJECT_NAME] R&R Report',
            body,
            to=[address]
        )
        email.send()
        return JsonResponse({'message': 'Email sent successfully'}, status=200)
    from rest_framework.decorators import api_view, authentication_classes
    from rest_framework.response import Response
    from rest_framework_simplejwt.authentication import JWTAuthentication
    from rest_framework import status
    from django.forms.models import model_to_dict
    from datetime import datetime, timedelta
    import logging

    from admin_page.models import Details, Health, Sensors, Recommend
    from static import rm7078
    logger = logging.getLogger(__name__)

    def toTable(table):
        if table.lower() == 'details':
            return Details
        elif table.lower() == 'health':
            return Health
        elif table.lower() == 'sensors':
            return Sensors
        elif table.lower() == 'recommend':
            return Recommend
        else:
            return None

    def returnValue(table, user):
        Table = toTable(table)
        try:
            data = Table.objects.filter(user=user).order_by('-created_time').first()
            return model_to_dict(data)
        except:
            return None

    def insertOrUpdate(table, user, data):
        Table = toTable(table)
        data = dict(data)
        for key in data:
            if type(data[key]) == list:
                data[key] = ','.join(data[key])
        try:
            latest = Table.objects.filter(user=user).order_by('-created_time').first()
            if latest:
                latestTime = datetime.strptime(str(latest.created_time)[19], '%Y-%m-%d %H:%M:%S')
                currentTime = datetime.strptime(str(datetime.now())[19], '%Y-%m-%d %H:%M:%S')
                timeDiff = abs(currentTime - latestTime).total_seconds()
                if timeDiff >= timedelta(hours=2):
                    latest._dict_.update(**data)
                    latest.save()
            except:
                pass
            Table.objects.create(user=user, **data)
            return True
        except:
            pass
            return False

    @api_view(['POST'])
    @authentication_classes([JWTAuthentication])
    def TableView(request, table):
        if request.method == 'POST':
            user = request.user
            logger.info(f'User (user.id) requested (table) Table.')
            fromDB = returnValue(table, user)
            if fromDB:
                return Response(fromDB, status=status.HTTP_200_OK)
            else:
                return Response({'error': 'Data not found'}, status=status.HTTP_404_NOT_FOUND)
        else:
            return Response({'error': 'Method not allowed'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)

    @api_view(['POST'])
    @authentication_classes([JWTAuthentication])
    def TableUpdate(request, table):
        if request.method == 'POST':
            user = request.user
            logger.info(f'User (user.id) requested to insert data.')
            if fromDB := returnValue(table, user, request.data):
                logger.error(user.id)
                returnList = rm7078.main(user_id=user.id)
                if returnList == [TEST 템플릿]:
                    return Response({'success': 'Data inserted successfully', 'returnList': returnList}, status=status.HTTP_200_OK)
                else:
                    return Response({'error': 'Data not inserted'}, status=status.HTTP_404_NOT_FOUND)
            else:
                return Response({'error': 'Method not allowed'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
```

# Chat GPT 검토

<https://chatgpt.com/share/ac8e9139-d982-450e-bf3f-dba1f7541e11>

## ### 코드 채점 (100점 만점)

이 코드는 클라이언트와 서버 간의 통신을 다루는 Python 애플리케이션으로, 여러 측면에서 평가해 볼 수 있습니다. 각 항목에 대해 점수를 부여하고, 종합 점수를 도출해보겠습니다.

### #### 1. 코드 구조 및 가독성 (20점)

- \*\*클라이언트 코드\*\*: 클래스와 메서드를 적절히 사용하여 구조화되어 있습니다. `ConnectUI` 와 `ConnectServer`의 역할이 명확히 분리되어 있습니다. 로깅을 통해 상태를 확인할 수 있도록 하여 디버깅에 용이합니다. (20/20)
- \*\*서버 코드\*\*: Django 뷰와 REST 프레임워크를 사용하여 API 엔드포인트를 구현했습니다. 함수와 메서드가 적절히 분리되어 있으며, 로깅을 통해 상태를 추적할 수 있습니다. (20/20)

### #### 2. 기능 구현 (40점)

- \*\*클라이언트 코드\*\*: 로그인, 데이터 전송 및 수신, 이메일 전송 등의 기능이 잘 구현되어 있습니다. 토큰 갱신과 같은 예외 상황도 잘 처리하고 있습니다. (40/40)
- \*\*서버 코드\*\*: 로그인, 이메일 전송, 데이터 조회 및 삽입 기능이 잘 구현되어 있습니다. JWT 인증을 사용하여 보안을 강화하였습니다. (40/40)

### #### 3. 예외 처리 및 오류 로그 (20점)

- \*\*클라이언트 코드\*\*: 로그인 실패, 데이터 전송 실패 등 다양한 예외 상황을 잘 처리하고 있으며, 이를 로깅하여 문제를 쉽게 파악할 수 있도록 했습니다. (20/20)
- \*\*서버 코드\*\*: 존재하지 않는 사용자 처리, 잘못된 메서드 요청 처리 등 다양한 예외 상황을 처리하고 있습니다. 그러나, `InsertOrUpdate` 함수에서 발생할 수 있는 모든 예외를 구체적으로 처리하지 않고 있습니다. (18/20)

### #### 4. 보안 (20점)

- \*\*클라이언트 코드\*\*: 토큰 기반 인증을 사용하고, 토큰 갱신을 통해 세션 관리를 하고 있습니다. 그러나 토큰 만료 시점에 대한 명확한 처리가 부족합니다. (18/20)
- \*\*서버 코드\*\*: JWT 인증을 사용하여 보안을 강화했으며, CSRF 예외 처리를 통해 보안을 유지하고 있습니다. 그러나 이메일 본문에 민감한 정보를 포함할 경우 보안에 취약할 수 있습니다. (18/20)

#### 총점: 96점

## ### 종합 피드백

- \*\*클라이언트 코드\*\*는 전반적으로 매우 잘 작성되어 있으며, 예외 처리와 로깅이 잘 되어 있습니다.
- \*\*서버 코드\*\*도 전반적으로 잘 작성되어 있으며, 특히 JWT 인증을 사용한 보안 강화가 인상적입니다. 그러나 예외 처리를 좀 더 세밀하게 하면 더 좋을 것입니다.
- 데이터베이스 모델에서 민감한 정보는 가능한 한 직접 노출되지 않도록 주의해야 합니다. 예를 들어, 이메일 본문에 포함되는 정보는 적절히 필터링되거나 암호화되어야 합니다.

## ### 추가 고려 사항

- \*\*유닛 테스트\*\*: 유닛 테스트를 추가하여 각 기능이 의도대로 작동하는지 확인할 수 있습니다.
- \*\*문서화\*\*: 각 클래스와 메서드에 대한 주석과 설명을 추가하여 코드의 이해도를 높일 수 있습니다.
- \*\*토큰 만료 시점 처리\*\*: 클라이언트에서 토큰 만료 시점에 대한 명확한 처리 로직을 추가하면 보안을 강화할 수 있습니다.

---

\*\*Q1:\*\* 유닛 테스트를 추가하여 각 기능이 의도대로 작동하는지 확인하려면 어떻게 해야 하나요?

\*\*Q2:\*\* 서버 코드에서 발생할 수 있는 예외 상황을 구체적으로 처리하기 위한 방법에는 어떤 것들이 있을까요?

\*\*Q3:\*\* 클라이언트에서 토큰 만료 시점을 처리하는 로직을 어떻게 구현할 수 있을까요?