2023 년 2 학기 IIT-4204
SW 프로젝트 최종 보고서

OCR 을 활용한 식단 관리 앱

박종현(2020193005), 신비(2022193006)
연세대학교 IT 융합공학과

요 약

건강한 식단에 대한 사람들의 관심이 많아지면서 식단 기록 및 관리를 위한 다수의 앱들이 출시되었다. 그러나 이러한 앱들은 영양성분 기록에 있어서 데이터가 부족하거나, 부정확하다는 문제점이 있다. 본 프로젝트는 이러한 문제점을 OCR 기술을 활용하여 해결하고, 다른 앱에서는 제공하고 있지 않은 기능들을 구현해보며 한 학기동안 end-to-end 로, 사용자의 요구 및 편리성을 고려한 서비스를 만들어보는 것이 목표이다.

프로젝트의 주요 구성 요소로는, 사용자 인터페이스의 편의성을 고려한 Flutter 기반의 프론트엔드, 이미지 처리와 OCR 을 통한 데이터 추출, 그리고 사용자 데이터의 안정적인 처리와 저장을 위한 Django 백엔드 및 PostgreSQL 데이터베이스가 있다. OCR 은 Google Cloud Vision API 를 사용했다.

프로젝트의 주요 쟁점은 추출된 데이터의 정확성과 검증, 데이터 처리 속도, 그리고 사용자 맞춤형 정보 제공이었다. 데이터 처리의 정확성을 보장하기 위해 OCR 정보를 바탕으로 맞춤형 알고리즘을 설계했으며, 앱과 서버 양쪽에서 전처리 과정을 포함하여 데이터 처리의 정확성과 속도를 개선했다. 또한 사용자로부터 정보를 받아 이를 기반으로 맞춤형 정보를 제공하고, 계정 연동과 함께 데이터베이스에 저장했다. 사용자의 편의성을 위해 UI 는 직관성을 주 목표로 설계하였고, 앱에서 제공하는 기능들은 기존에 출시된 앱들의 사용자 경험을 참고하며 구현하였다.

종합적으로 프로젝트는 기술적 목표를 달성했으며, 최소 기능 제품(MVP)을 만족한다. 그러나 계획한 모든 기능을 전부 구현한 것이 아니기 때문에, 향후 계획과 목표를 제시하는 것으로 보고서를 마무리한다.

1. 프로젝트 배경 및 목적 (작성자 : 박종현)

현재 존재하는 식단 관리 앱들의 핵심적인 기능과 목적은 사용자가 평소에 먹는 음식들을 기록하여 권장량 만큼의 영양소를 섭취할 수 있도록 하는 것이다. 본 프로젝트는 이런 앱들이 공통적으로 갖고 있는 문제점을 해결하고, 차별화된 기능을 추가하는 것이다.

기존의 앱들은 영양성분표가 있는 제품들의 영양 성분을 기록할 때 우선 제품의 영양성분 정보를 데이터베이스에 저장한다. 그리고 사용자가 바코드를 찍거나 제품명을 검색하면, 데이터베이스에 저장된 정보를 바탕으로 기록하게 된다.

제품명을 직접 검색하는 방법으로 기록하는 것은 사용자 입장에서 귀찮게 느껴지기에 대부분의 사용자들은 바코드를 사용하여 기록하는 방법을 선호한다. 이러한 편리함이 바코드 인식 기능을 보유하고 있는 앱이 가장 많은 다운로드 수를 갖고 있는 이유 중 하나라고 생각된다.

앱	다운로드 수	제품 검색 방법
Yazio	1000만+	바코드 사진 / 직접 검색
밀리그램	10만+	직접 검색
인아웃	10만+	직접 검색
닥터 다이어리	100만+	직접 검색
필라이즈	10만+	직접 검색

[그림 1] Top 5 식단 관리 앱들의 플레이 스토어 다운로드 수 및 제품 검색 방법

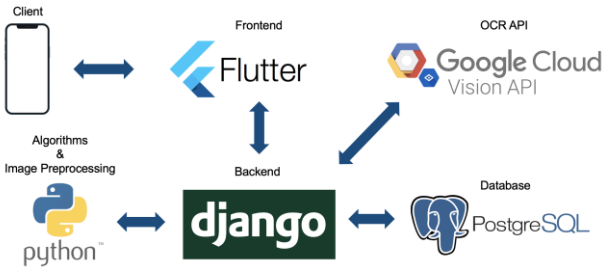
가장 많은 다운로드 수를 보유하고 있는 Yazio 의 후기(그림 2)를 참고하면, 데이터베이스를 바탕으로 제품의 영양 정보를 기록했을 때 크게 두 가지 문제점이 존재한다. 우선 첫째로, 인기가 없거나 새로 출시된 제품의 경우 데이터베이스에 없는 경우가 많아 사용자가 직접 모든 영양 정보를 입력해야 한다는 불편함이 존재한다. 둘째로, 이렇게 데이터베이스에 존재하지 않는 제품은 사용자가 요청하여 등록할 수 있게 되는데, 등록 과정에서 많은 오류가 발생한다.

[그림 2] Yazio 후기

우리는 이번 프로젝트를 통해, 사용자의 편리성은 유지하면서 데이터베이스 기록 방식의 단점을 해결하기 위해 OCR 기술을 활용하기로 하였다. OCR 을 사용하여 별도의 데이터베이스 없이 영양성분표로부터 영양 정보를 바로 추출할 수 있다면, 제품이 등록되지 않아 정보를 직접 입력해야 한다는 문제점을 해결할 수 있을 것으로 예상했다.

기존 앱들에는 존재하지 않는 추가적인 기능으로는, 사용자가 자신과 비슷한 체형, 목적을 가진 다른 사용자들의 식단을 참고할 수 있도록 하는 것을 도입할 예정이다.

2. 시스템 아키텍처 (작성자 : 박종현)



[그림 3] 시스템 아키텍처

Frontend Framework – Flutter

한 학기 내에 Android, iOS 에서 모두 사용할 수 있는 앱 제작이 목표이다 보니 cross-platform framework 를 사용하기로 하였다. 가장 많이 사용하는 Flutter 와 React-Native 중에서 최종적으로 Flutter 를 사용하게 된 이유는, 사용자들에게 식단 관련 정보를 보여주는 데 최적화된 우리만의 디자인을 사용하기 위함이다. 우리가 사용하고자 하는 독특한 디자인 구현을 위해서는, 화면에 나타나는 거의 모든 UI 를 native os 가 아닌 Dart 가 렌더링하는 Flutter 를 사용하는 것이 React-Native 보다 유리하다고 생각했다.

Backend Web Framework – Django

OCR 을 활용하여 영양 성분 정보를 정확히 추출해내기 위해서는 사진에 대한 전처리가 필요하다. 사진 전처리에는 파이썬 라이브러리를 활용할 계획인데, Node js 와 같이 다른 언어로 된 framework 를 사용하면 외부 script 를 실행하는 데 overhead 가 발생할 것으로 예상되었다. 따라서 파이썬 기반의 framework 인 Django 를 채택하였다.

Database – PostgreSQL

Django 에서 공식적으로 지원하는 데이터베이스는 모두 RDBMS 계열이다. MongoDB 와 같은 NoSQL 도 별도의 연동 과정을 통해 사용할 수는 있지만, 공식적으로 지원하고 있지는 않아 안정성이 떨어진다고 알려져 있다. 따라서 꼭 NoSQL 과 같이 다른 구조를 가진 데이터베이스를 사용해야 하는 것이 아니라면, 공식적으로 지원하고 있는 데이터베이스를 사용하는 것이 권장된다. 추가적으로 이번 프로젝트에서 저장해야 하는 데이터는 뒤에서 더 자세히 다루겠지만, 테이블 구조를 사용하는 것이 적합하다고 판단했기에 PostgreSQL 을 사용하였다.

Image Preprocessing – Python

앞서 언급했듯이 OCR 을 활용하여 영양 성분 정보를 정확히 추출해내기 위해서는 사진에 대한 전처리가 필요하다. 파이썬에는 이미지 처리와 관련하여 많은 라이브러리들이 존재하고, OpenCV 와 같은 라이브러리는 C/C++ 기반으로 동작하기 때문에 매우 빠르다는 장점이 있다. 따라서 이러한 장점을 이용하기 위해 이미지 전처리 과정에서는 파이썬을 사용하기로 하였다.

OCR – Google Cloud Vision API

이미지 전처리를 제대로 진행해도, OCR 이 정확하게 되지 않으면 소용이 없다. 따라서 OCR API 를 선택하는 데 있어서 가장 중요하게 여겼던 것은 OCR 정확도였다. 한글에 대해 안정적인 정확도를 내는 API 에는 Google Cloud Vision, Naver Clova 크게 두 가지가 있다. 이 둘의 정확도 차이는 크지 않지만, Google Cloud Vision API 가 저렴하면서도 응답 속도가 빠르다고 알려져 있기에 Google Cloud Vision 을 사용하게 되었다.

3. System Components

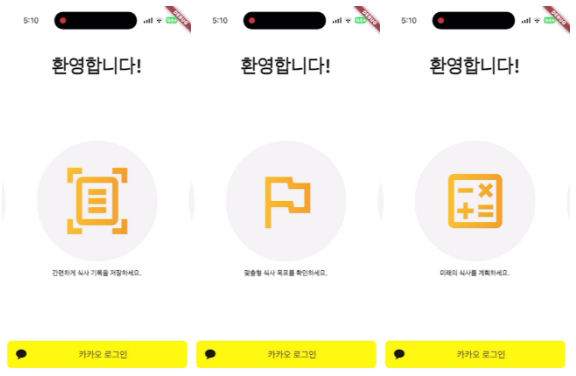
3-1) Frontend Components (작성자 : 신비)

(a) 앱 디자인

프로젝트에서 개발한 앱은 식단 및 영양과 관련되었기 때문에 전반적으로 난색 계열의 색상을 사용하였다. 또한 사용자와 상호작용하는 위젯은 둥근 모서리를 포함하고, 그렇지 않은 위젯은 사각 모서리를 사용하여 디자인의 일관성을 유지했다.

로그인 화면 디자인

로그인 화면에서 앱의 주기능을 아이콘과 함께 소개하여, 앱을 처음 사용하는 사람도 앱의 목적을 이해하고 어떻게 이용하면 되는 지에 대한 방향을 제시했다. 횡축 방향으로 페이지가 자동으로 넘어가며 기능이 하나씩 소개된다.



[그림 4] 로그인 화면

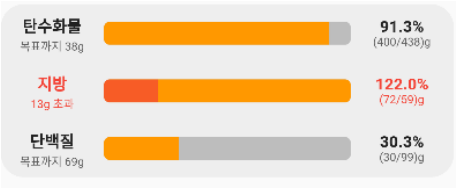
홈 화면 디자인

앱의 핵심적인 기능인 영양소 추적을 위한 그래프는 홈 화면 상단에 배치했다. 이는 사용자가 앱을 실행했을 때 시선이 가장 먼저 가는 곳에 중요한 영양 정보를 표시함으로써 즉시 확인할 수 있도록 하기 위함이다. 그래프는 주요 영양소와 그 외 영양소를 색상을 구분하여 시각적으로 명확하게 표시하였으며, 이는 사용자가 필요한 정보를 빠르게 접근할 수 있도록 도와준다.

영양소 그래프 위젯은 확장 기능을 포함하여, 사용자가 탄수화물, 단백질, 지방 외의 영양소에 대한 정보를 원할 때 쉽게 열어볼 수 있도록 하였다. 접힌 상태와 확장된 상태 사이의 부드러운 전환을 위해 애니메이션 효과를 적용함으로써 사용자 경험을 극대화하고 인터페이스의 직관성을 강화했다.

[그림 5] 홈 화면, 접힌 상태(좌)와 확장된 상태(우)

사용자가 권장 영양소 섭취량을 초과할 경우, 그래프에서 라벨과 초과된 양을 적색으로 표시하여 시각적 피드백을 제공한다. 이러한 경고 메커니즘은 사용자가 자신의 식습관에 대해 보다 의식적인 결정을 내릴 수 있도록 유도한다.



[그림 6] 영양소 그래프 위젯

앱 하단에는 다양한 기능을 수행하는 버튼 그룹이 배치되어 있다. 각 버튼은 중요도에 따라 색상과 크기가 다르게 설정하였다.

폼 제출 화면 디자인

영양소 양에 대한 폼 제출 화면의 각 영양소는 영양성분표의 순서를 그대로 사용하여 정보 검증 시 자연스러운 시선으로 검증이 가능하도록 했다. 필수로 입력해야 하는 영양소는 적색 별 기호를 따로 표기하여 사용자가 한눈에 알아볼 수 있게 유도하였다.

[그림 7] 폼 제출 화면

사용자가 항상 영양성분표에 해당하는 양만큼의 음식을 섭취하는 것이 아니기 때문에, 영양성분표와는 별개로 사용자가 먹은 양을 따로 기록해야 한다. 따라서 기록 수단으로서 슬라이더 위젯을 채택했다. 슬라이더 위젯은 단위 영양성분표를 기반으로 하여, 사용자가 먹은 양을 정수 배수(x1, x2, x3...)로 기록한다. 만약 사용자가 단위 영양성분표보다 적은 양을 섭취했을 경우, 사용자는 슬라이더를 x1 에서 x0 방향으로 조정하도록 시도할 것이다. 이때 슬라이더의 값이 x0 에 도달하면, 슬라이더의 색상이 변함과 동시에 슬라이더의 단위가 백분율 단위(25%, 50%, 75%)로 전환된다. 이를 통해 사용자는 1 개 미만의 섭취량을 보다 세밀하게 기록할 수 있다. 또한, 백분율 슬라이더를 100%까지 당길 경우, 다시 정수 배수 슬라이더로 전환되는 디자인을 사용했다.

[그림 8]정수 배수 슬라이더(위)와 백분율 슬라이더(아래)

(b) Image Preprocessing

앱 내에서의 이미지 전처리 과정은 서버로의 사진 전송 시간을 최소화하고, OCR 과 알고리즘을 이용한 영양정보 추출 시간을 단축하기 위해 필수적이다. 앱 수준에서 실행될 전처리로 Perspective Transformation, Rotation, Grayscale, Resizing 이 제안되었으나 Perspective Transformation 과 Rotation 은 사용자가 직접 조정해야 하는 불편함이 있고, 이 두 개의 전처리 유무와 관계없이 OCR 정확도는 그대로였기에 Grayscale 과 Resizing 처리만 채택되었다. Grayscale 처리는 이미지의 색상을 회색조로 변환하여 데이터 크기를 줄이는 과정이고 Resizing 은 이미지의 해상도를 줄여 데이터 크기를 줄이는 과정이다.

초기에는 모든 사진을 동일한 비율(60%)로 축소하는 전략을 사용했다. 그러나 저해상도 이미지에서 OCR 성능에 문제가 발생하는 것을 확인했다. 이미 저해상도인 이미지[그림 9]를 한번 더 축소한 결과[그림 10], 글자가 심하게 뭉개진 사진을 서버에 전송하는 문제가 있었다. 따라서 일괄적 축소 대신 해상도에 따라 유동적으로 이미지 크기를 축소하는 방식으로 전략을 수정했다. 개선된 resizing 과정에서는 이미지의 가로 또는 세로 길이 중 더 짧은 쪽을 기준으로 해상도를 1000 픽셀로 조정한다. 이미 1000 픽셀 미만인 저해상도 이미지는 이 압축 과정에서 제외된다.

[그림 9] 원본 저해상도 이미지(800x534)

[그림 10] 저해상도 이미지 축소 결과(좌), 축소하지 않은 결과(우)

	원본 이미지	전처리된 이미지
정확도	149/160 (93.12%)	150/160 (93.75%)
평균 응답시간	4.68 초	2.65 초

20 장의 영양성분표 사진에 대한 테스트 진행 결과

앱 수준에서 먼저 진행되는 전처리 과정은 앱과 서버의 전반적인 성능을 향상시키는 데 크게 기여한다. 위 표를 참고하면, 원본 이미지를 사용했을 때보다 전처리를 진행했을 때 정확도는 유지하며 사용자 응답 시간을 43.38% 단축시킬 수 있었다. 이미지 처리 시간이 단축

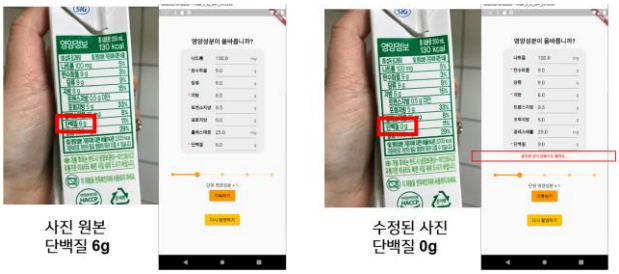
됨으로써 사용자 경험은 보다 원활해지고, 서버 부담도 줄어든다.

(c) OCR 정보 검증

앱에서 서버에 전처리된 사진을 보내고 OCR 정보를 받으면 영양성분 정보를 폼 수정 페이지에 미리 작성한다. 영양성분 추출에 실패하거나, 잘못된 정보를 추출했을 경우를 대비해 사용자의 2 차 검증을 받아야 한다. 폼 제출 페이지를 표시하기 전 앱에서도 먼저 정보의 유효성을 1 차적으로 검증하는데, 아래의 공식을 이용한다.

$$총열량(kcal) = 4(kcal/g) \times 탄수화물(g) + 9(kcal/g) \times 지방(g) + 4(kcal/g) \times 단백질(g)$$

해당 식에 OCR 정보를 대입했을 때 문제가 발생하면, 사용자에게 잘못된 데이터가 포함되어 있을 수도 있다는 것을 알려 사용자가 정보를 확인할 수 있도록 하였다.



[그림 11] OCR 정보 검증

(d) Provider

Flutter 를 이용하여 앱을 개발할 때, 위젯의 state 관리를 위해 자주 사용하는 방식은 SetState 이다. SetState 는 Flutter 의 기본 state 관리 기법으로, 한 위젯에서 SetState 를 호출하면 Flutter 는 해당 위젯을 다시 빌드한다. 이는 작은 규모의 앱 또는 단일 위젯에서는 효율적이지만, 위젯이 복잡한 구조로 이루어져 있거나 요소가 많이 포함되어 있다면 state 를 업데이트하기 위해 위젯의 모든 요소를 리빌드해야 한다는 점에서 비효율적이다.

Provider 는 보다 복잡한 state 관리를 위해 사용되는 패키지로, 앱 전체에 걸친 데이터와 위젯 state 를 공유하고 관리하는 데에 적합하다. SetState 와 달리, Provider 패키지를 사용하게 되면 Provider 에 연결된 위젯만을 다시 빌드할 수 있다. 이는 state update 가 필요한 위젯만 다시 빌드한다는 점에서 효율적이며, Notify 를 이용해 Provider 와 연결된 model 의 데이터를 업데이트하면 listener 가 포함된 위젯이 자동으로 리빌드된다.

Provider 는 선택적으로 위젯을 다시 빌드한다는 점에서 성능적으로도 효율적이며, 코드 작성시 위젯의 state 를 관리하는 코드와 위젯을 디자인하는 코드를 분리하여 작업할 수 있기 때문에 가독성과 유지 보수적 측면에서도 용이하다.

가령, 폼 제출시 유저 정보를 홈 화면에 업데이트해야 하는 기능을 구현해야 할 때, Provider 를 사용하기 전에는 SetState 를 이용해 영양성분 그래프를 업데이트해야 했다. 폼 제출 화면은 홈 화면으로부터 push 된 별개의 위젯이기 때문에, 폼 제출 화면이 pop 되면서 홈 화면에서 SetState 를 호출하기 위해서는 폼 제출 화면을 push 할 때 인자로서 콜백함수를 따로 추가하거나, 홈 화면 위젯의 글로벌 키를 따로 설정해야 한다. 그러나 Provider 를 사용하게 되면, 폼 제출 화면을 pop 하면서 상위 Provider 에 notifyListner 를 호출하여 간단하게 위젯을 리빌드할 수 있다. 앱에서는 Nutrients Model 과 UserInfo Model 을 생성하여 각 Model 에 대한 Provider 를 생성했다.

(e) 카카오톡 로그인

앱에서 카카오톡으로 로그인하게 되면 내부 데이터에 토큰이 남는다. 토큰에는 회원 id 가 포함되어 있기 때문에, 토큰을 이용하여 로그인 여부를 판단한다. 서버에 회원 id 를 포함하여 사용자 정보를 요청했을 때, 받는 정보에 따라 기존 회원과 신규 회원을 구분한다. 로그인과 관련된 앱의 전체적인 로직은 아래와 같다.

앱을 처음 실행
1. 앱에 토큰이 있고 유효한가? -> 있으면 2 번, 없으면 로그인 페이지로 2. 서버에 사용자 정보를 요청 -> 올바른 정보를 받으면 3 번, 특정 코드를 받으면 회원가입 페이지로 3. 받은 정보와 함께 홈 화면으로 이동

로그인 페이지
1. 카카오톡 로그인 후 토큰 발급 요청 2. 서버에 사용자 정보를 요청 -> 올바른 정보를 받으면 3 번, 특정 코드를 받으면 회원가입 페이지로 3. 받은 정보와 함께 홈 화면으로 이동

회원가입 페이지
1. 사용자 정보 폼 작성 후 제출 2. 서버에서 신규 회원을 등록하고, 목표 영양소를 계산 후 앱으로 전송 3. 받은 정보와 함께 홈 화면으로 이동

정리를 했을 때는 간단해 보이지만, 여러 시행착오를 거친 끝에 다음과 같은 로직을 작성할 수 있었다. 로그인 구현 중에서 발생한 문제는 크게 2 가지가 있었다.

앱을 처음 실행했을 때, 1 차로 토큰이 있고 유효한지 확인한 후, 2 차로 서버에 사용자 정보를 요청한다. 로직에 따르면 비동기 함수가 조건에 따라 1 번 또는 2 번 실행되는데, 비동기 함수의 결과에 따라 비동기적으로 다른 페이지를 연결하는 것이 해결해야 할 문제였다. 앱이 처음 실행되자마자 비동기 함수가 실행되기 때문에 try catch 문과 같은 예외 처리에 최적화된 함수를 사용할 수 없어, 각 예외에 대한 경우를 모두 고려하여 직접 예외를 처리해야 했다. 이를 위해 FutureBuilder 를 이중으로 사용하는 방법을 택했다. 비동기 함수가 종료되기 전에 다른 페이지로 대체되는 경우를 막기 위함이다.

사용자가 신규 회원인지 기존 회원인지를 구분할 방법이 필요했다. 그러나 앞선 문제의 연장선상에서, 앱을 처음 실행했을 때 비동기 함수의 추가는 많은 제약이 따른다. 만약 서버에 추가적인 통신을 넣어 신규 회원 여부를 확인하는 절차를 추가하게 되면 비동기 함수가 하나 더 추가되어 앱의 복잡성이 크게 증가하고, 그에 따른 예외 처리를 모두 해야 한다. 따라서 기존 서버 통신 로직을 활용하여, 서버에 사용자 정보를 요청했을 때, 사용자 정보 내부 데이터가 특별한 값(목표영양성분이 -1)을 포함하고 있으면 신규 회원으로 구분하는 방법을 택했다.

(f) 예외 처리

서버와 통신 시, 예상치 못한 상황이 발생할 때마다 앱이 freezing 되는 문제가 있었다. 대부분은 try catch 로 묶어 해결했지만, 일부 예외 처리는 직접 처리해야 했다. 다음은 freezing 이 발생하는 목록이고, 각 항목에 대해 예외 처리를 진행했다.

영양성분표 촬영 및 폼 제출
1. 카메라 및 미디어 권한이 없는 경우 2. 권한 요청을 했으나 권한 획득에 실패할 경우 3. 이미 있던 권한이 제거된 경우 4. 카메라를 실행하고 사용자가 사진을 찍지 않은 경우 5. 미디어에서 사용자가 사진을 선택하지 않은 경우 6. 유효하지 않은 형식의 이미지를 선택한 경우 7. 서버 통신에 오류가 발생한 경우

카카오 로그인 및 회원가입
1. 휴대전화에 카카오톡이 설치되지 않은 경우 2. 카카오 토큰이 만료되거나 유효하지 않은 경우 3. 카카오 토큰을 발급받았으나 정보를 인식할 수 없는 경우 4. 신규 회원이 회원가입 폼을 작성하지 않고 앱을 종료한 경우 5. 서버 통신에 오류가 발생한 경우

(g) 코드 리팩토링

미래의 유지 보수를 용이하게 하고, 다른 팀원과의 협업을 원활하게 하기 위해 코드 리팩토링을 진행했다. 프로젝트가 진행되면서, Flutter 개발에 익숙해지면서 기존 코드에서 불필요한 부분이 보였고, 코드가 길어짐에 따라 가독성이 매우 떨어지는 문제가 발생하여, 이를 해결하고자 했다.

리팩토링 과정에서 주요 초점은 폴더 구조를 체계화하는 것이었다. 이를 위해, 기존의 main.dart 에서 작성되었던 코드를 여러 하위 폴더 내의 dart 파일로 세분화했다. 이 과정에서 model, provider, screen 등 각각의 기능별로 코드를 분류하고 정리했다. 이를 통해 각 기능과 구성 요소를 명확하게 분리하여, 코드의 가독성을 크게 향상시킬 수 있었다. 또한 중복되는 코드를 재활용함으로써 불필요한 코드도 크게 줄일 수 있었다.

3-2) Backend Components (작성자 : 박종현)

(a) Image Preprocessing

이미지 전처리는 정확한 OCR 을 위해 필요하다. 영양성분표의 일부 글자가 그림자로 인해 잘 보이지 않는 경우가 가끔 있고, 비닐이나 캔 등의 재질로 만들어진 제품들은 빛 반사에 의해 글자가 가려지는 경우가 생긴다. 추가적으로, 사용자가 사진을 수평에 맞게 찍지 않고 틀어서 찍더라도 텍스트 인식이 제대로 되도록 하기 위해, skew correction 을 진행하였다.

Denoising

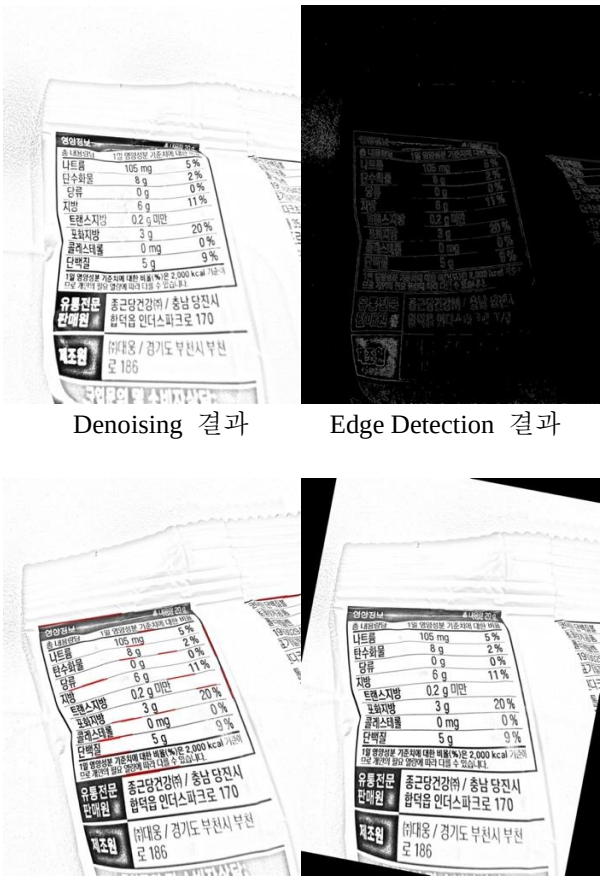
따라서, 우선 그림자나 빛과 같은 noise 을 제거하기 위해 gaussian blur denoising 을 사용하였다. Denoising 이 적용되기 전후 사진은 아래와 같으며, 불필요한 그림자나 빛 반사 영역이 없어진 것을 확인할 수 있다.

Denoising 전 Denoising 후

[그림 12] Denoising 과정

Skew Correction

틀어서 찍힌 사진은 우선 가로로 된 선들을 찾은 후, x 축 기준으로 틀어진 각도의 평균만큼 사진을 회전시켜 수평을 맞추었다. 사진에 존재하는 가로 선들을 찾기 위해서는 사진에 존재하는 edge 를 찾는 과정이 선행되어야 하며, 이를 위해서는 canny edge detection 을 사용하였다. 이후 hough line transform 을 통해 사진에 존재하는 선들을 찾았다. Hough line transformation 은 OpenCV 라이브러리에 존재하는데, cv2.HoughLines()와 cv2.HoughLinesP() 두 가지가 있다. 둘의 가장 큰 차이점은 직선임을 판단하기 위해 검사하는 점의 개수이다. 전자는 모든 점에 대해 계산을 하기 때문에 시간이 오래 걸리지만, 후자는 임의의 점을 이용하여 직선을 찾기 때문에 속도가 빠르다. 따라서 후자를 통해 사진에 존재하는 선들을 찾았다. X 축을 기준으로 하였을 때 45° 보다 작은 각도로 기울어진 선들을 가로 선으로 추정하고, 이들의 평균 각도만큼 사진을 회전시켰다. Skew correction 과정은 [그림 13]에 나타내었다.



Denoising 결과 Edge Detection 결과

Hough line transform 결과 Rotation 결과

[그림 13] Skew correction 과정

(b) 영양성분 매칭

위치 기반 추출 알고리즘

이미지 전처리 후, 영양성분표에 존재하는 각 영양소의 양을 파악하기 위해 Google Cloud Vision API 를 사용하여 사진에 있는 텍스트를 추출하였다. Google Cloud Vision API 는 응답으로 텍스트 뿐만 아니라 각 단어의 bounding box 위치까지 제공하기 때문에, bounding box 의 위치를 기반으로 각 영양소 양을 매칭하기로 하였다. 아래와 같이 가로로 된 영양성분표의 경우에는 세로 좌표를 비교함으로써, 세로로 된 영양성분표는 가로 좌표 비교를 통해 영양성분 양을 매칭하였다.

영양성분표에 있는 영양소 양은 실제 양인 g, 그리고 영양성분에서 차지하는 비율 % 두 가지가 있다. g 에 대응되는 숫자는 영양소 단어의 옆에(가로형 성분표인 경우) 있는 단어 4 개를 합친 다음, regular expression 을 통해 첫 번째 'g' 앞에 나오는 숫자로 판단하였다. 옆에 있는 단어 4 개를 보는 이유는 [그림 14]에서 '25g'이라는 문자열이 '25', 'g'으로 분리되어 검출될 수도 있고, '25g' 하나로 검출될 수도 있기 때문이다. '25g 8%'가 아닌 '8% 25g'으로 배치되어 있는 영양성분표의 경우, '8', '%', '25', 'g', 4 개의 단어로 분리되어 검출될 가능성이 있기 때문에 4 개를 포함시켜 regular expression 에 입력해야 한다.

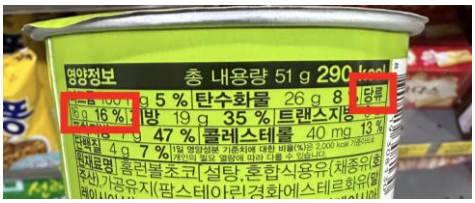
세로형 성분표인 경우에는 [그림 15]와 같이 아래에 있는 2 개의 숫자 중, 옆에 'g'라는 문자가 있는 숫자로 매칭하였다.

[그림 14] 가로형 성분표

[그림 15] 세로형 성분표

이 방법을 사용했을 때 일반적인 영양성분표에서는 문제가 없다. 그러나 [그림 16]과 같이 영양성분 정보가 다른 줄에 걸쳐서 표시된 경우 위치 기반 알고리즘으로

추출되지 않는다는 점이다. 이를 해결하기 위해, 문자열 기반으로 영양성분을 추출하는 알고리즘을 추가하였다.



[그림 16] 줄 바꿈이 일어나는 성분표

문자열 기반 추출 알고리즘

위치 기반 추출 알고리즘이 문제가 되었던 것은 텍스트 중간에 줄 바꿈이 일어나기 때문이다. 따라서 줄 바꿈이 일어나더라도 각 줄을 하나의 문자열로 합치면, 영양 성분과 양을 차례대로 매칭할 수 있을 것으로 예상했다.

Google Cloud Vision API OCR 의 응답에 들어있는 단어들의 순서는 이미지 상에서 사람이 글을 읽는 순서와 동일하다. 따라서 응답에 들어있는 모든 단어들을 하나의 배열로 합치면 줄 바꿈이 일어나더라도 다음 단어들을 보면서 정보 해석이 가능하다.

하지만 영양성분 정보를 파악하는 데 필요한 단어들만 해석하면 되므로, 모든 단어를 배열에 포함시킬 필요는 없다. 따라서 해당 단어가 영양소 단어의 2 글자 이상의 substring 이거나, 알파벳 ‘g’가 포함되어 있거나, 숫자가 포함되어 있으면 단어 배열에 추가하도록 하였다.

[그림 16]의 위의 이미지를 가지고 단어 필터링을 하였을 때, 다음과 같은 배열이 생성된다.

['B2', '1 일', '10kcal,', '탄수화물', '2g(1%),', '단백', '0g(0%),', '지방', '0g(0%),', '나트륨', '0mg(0%),', '500', 'mg(500%),', '비타민 B10.36', 'mg(30%),', '비타민 B2', '0.42', 'mg(30%),', '비타민 B6', '0.45', 'mg(30%),', '비타민 B12', '0.72']

생성된 배열에서의 영양소 양 매칭 과정은 위치 기반 추출 알고리즘에서 사용한 regular expression 을 사용하였다.

알고리즘 동작 순서

위치 기반 추출 알고리즘이 줄 바꿈 문제를 해결하지 못해 문자열 기반 추출 알고리즘을 사용한 것처럼, 문자열 기반 알고리즘 역시 세로형 영양성분표가 주어지면 영양소 양을 매칭하지 못한다. 즉, 두 가지 알고리즘이 모두 필요하다. 따라서 문자열 기반으로 먼저 영양소 양 매핑을 시도하고, 파악하지 못한 영양소들에 대해서는 추가적으로 위치 기반 알고리즘을 사용하였다.

(c) Database

데이터베이스는 RDBMS 계열인 PostgreSQL 을 사용한다. 이번 프로젝트에서 저장해야 하는 요소는 크게 세 가지이다.

사용자 기본 정보

사용자 정보는 나이, 체중, 몸무게 등 기초/유지 대사량 계산을 위한 정보와 목표로 하는 체형, 오늘 먹은 영양소 양 등 사용자의 식단을 위한 정보를 포함한다. 이를 저장하기 위한 테이블 정보는 아래와 같다.

Field	저장 정보	Type	저장 목적 / (세부정보)
id (Primary)	사용자 고유 id	Char	사용자를 구분 / 'k' + '123456789' 형태로 저장되며, k 는 회원가입 하는 데 사용된 소셜 로그인 제공자, 숫자는 제공자가 사용자마다 부여한 고유 id 이다.
name	사용자 이름	Char	메인 화면에 사용자 이름을 띄우기 위함
is_male	사용자 성별	Int	사용자와 비슷한 체형과 목표를 가진 다른 사용자들의 식단 정보를 제공할 때 남녀를 구분
birth_year	사용자 출생년도	Int	추후 나이대에 따른 정보를 제공
height	사용자 키 (cm)	Float	BMI 계산
weight	사용자 몸무게 (kg)	Float	BMI, 권장 영양 섭취량 계산
fat_ratio	사용자 체지방률 (%)	Float	사용자의 체형을 구분, 체성분 변화 그래프를 시각화
muscle	사용자 근육량 (kg)	Float	사용자의 체성분 변화 그래프 시각화
activity_rate	사용자의 평소 활동량	Int	권장 영양 섭취량 계산 / 2 : 평소에 많은 활동량을 가진 사용자 1 : 평소에 평균적인 활동량을 가진 사용자 0 : 평소에 적은 활동량을 가진 사용자
body_goal	사용자의 식단 목적	Int	권장 영양 섭취량 계산에 필요, 사용자와 비슷한 체형과 목표를 가진 다른 사용자들의 식단 정보 조회
body_state	사용자 체형	Int	사용자와 비슷한 체형과 목표를 가진 다른 사용자들의 식단 정보 조회 / 사용자의 BMI, 체지방률에 따라 11 개의 체형으로 구분 0 : 마름, 1 : 약간 마름, 2 : 마른 비만, 3 : 근육형 날씬, 4 : 날씬, 5 : 적정, 6 : 근육형, 7 : 통통_type1, 8 : 운동선수, 9 : 통통_type2, 10 : 비만
nutrient_today	사용자가 오늘 섭취한 영양성분	Json	사용자가 오늘 섭취한 영양성분 정보 조회 / Ex) {"fat": 28.0, "carb": 100.0, "prot": 8.0, "sugar": 56.0, "trans": 0.0, "sodium": 360.0, "saturated": 16.4, "cholesterol": 0.0}
nutrient_goal	사용자에게 권장되는 영양성분	Json	사용자의 체성분, 식단 목적에 따라 권장되는 영양성분 정보 조회 Re / Ex) {"fat": 62, "carb": 456, "prot": 104, "sugar": 70, "trans": 3, "sodium": 2000.0, "saturated": 31, "cholesterol": 300.0}

사용자들의 날짜별 섭취기록

사용자가 자신의 과거 식단 정보를 조회하거나, 자신과 비슷한 체형과 목표를 가진 다른 사용자들의 식단 정보를 조회하는 데 필요한 데이터를 저장하기 위한 테이블이다.

Field	저장 정보	Type	저장 목적 / (세부정보)
user_info	해당 record 의 사용자 정보	Foreign_key	현재 사용자 정보와 연동
record_date	테이블 record 가 저장된 날짜	Date	Record 저장 날짜 조회
weight	이 날 몸무게 (kg)	Float	Record 저장 당시 사용자 몸무게 조회
fat_ratio	이 날 체지방률 (%)	Float	Record 저장 당시 사용자 체지방률 조회
muscle	이 날 근육량 (kg)	Float	Record 저장 당시 사용자 근육량 조회
nutrient_eaten	이 날 섭취한 영양성분	Json	사용자가 이 날 섭취한 영양성분 정보 조회
nutrient_goal	이 날 권장되었던 영양성분	Json	이 날 사용자의 체성분, 식단 목적에 따라 권장되었던 영양성분 정보 조회

사용자들이 업로드한 이미지

사용자들이 영양성분표를 촬영하여 업로드하였을 때, OCR 결과가 부정확했다면, 알고리즘에 어떤 문제가 있었는지 디버깅하기 위해 이미지를 저장한다.

Field	저장 정보	Type	저장 목적 / (세부정보)
orig_img	사용자가 보낸 이미지	Image	부정확했던 OCR 결과에 대해 분석

(d) Handling Requests

URI Path	Method	Body	목적	Response
/signup	Post	회원가입 페이지에서 요구되는 사용자 정보 json	사용자가 입력한 정보를 바탕으로 데이터베이스의 사용자 테이블에 새로운 사용자에 대한 record 등록	사용자 record 가 정상적으로 저장되었다면 1, 문제가 발생하면 0 반환
/get_user_all	Post	카카오, 네이버 등에서 사용자별로 발급하는 고유 id	앱에서 사용자 정보를 조회하기 위함	User ID 에 해당하는 사용자의 정보를 json 형태로 반환
/send_image	Post	영양성분표 사진	사용자가 섭취한 음식의 영양성분표 분석	사진에 존재하는 영양성분들의 양을 json 형태로 반환
/client_nutrients	Post	사용자 id, 사용자가 입력한 영양성분 json	사용자가 확인한 영양성분 양 기록	사용자가 현재까지 먹은 영양소 양, 목표 영양소 양을 json 형태로 반환

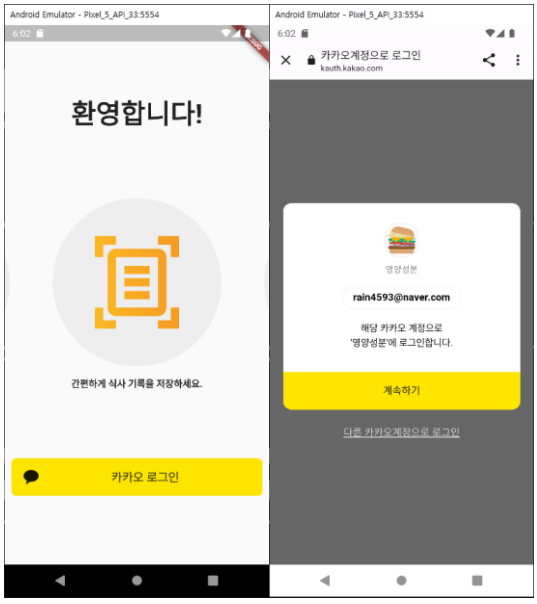
4. 프로젝트 일정 (작성자 : 박종현, 신비)

기간	Frontend	Backend
Week 1 9/22~9/28	Flutter 'Hello, world!' 카메라 및 미디어 권한 설정(AOS, IOS)	이미지 전처리 - Denoising - Skew Correction
Week 2 9/29~10/5	이미지 전처리 (Perspective transformation, Grayscale & resize)	Bounding Box 를 활용한 위치 기반 영양성분 매칭 알고리즘 구현
Week 3 10/6~10/12	앱과 서버 통신 실습 Github Repository 생성	위치 기반 알고리즘으로 해결하지 못하는 경우를 위해 문자열 기반 영양성분 매칭 알고리즘 추가
Week 4 10/13~10/19	MVP1 일부 기능 구현: 사진 전처리 및 전송 후 OCR 정보 받기	1. 클라이언트로부터 사진을 받아 영양성분 결과 보내주는 API 제작 2. 영양성분 매칭 알고리즘 Bug Fixes
Week 5 10/20~10/26	통신 예외 처리 Nutrients Model 생성: 영양성분 정보 관리 영양성분표 폼 제작: OCR 정보 선입력	OCR 최적화 - 이미지 전처리 parameter tuning - 이미지 축소 비율 tuning
Week 6 10/27~11/2	필수 항목 설정, 작성된 폼 서버로 전송	낱개 단위로 소분된 제품의 영양성분표 처리
Week 7 11/3~11/9	홈 화면 디자인	1. 영양성분 Form Handling 2. Debug 를 위해 사용자가 보낸 사진 서버에 저장되도록 설정
Week 8 11/10~11/16	코드 리팩토링, 폴더 구조 체계화 Provider 패키지 추가 -> 위젯 state 최적화	Test DB 생성하여 가상의 사용자가 먹는 음식들 누적해서 기록
Week 9 11/17~11/23	사용자 DB 요청, 받은 정보를 Provider 에 연결 영양성분 데이터 검증(열량 공식) 먹은 양 슬라이더 위젯 구현 여러 위젯에 애니메이션 효과 추가	1. 사용자가 지금까지 먹은 양, 하루 목표량을 전달하는 request handling 2. 사용자 정보를 기록할 DB 테이블 설계
Week 10 11/24~11/30	이미지 압축 로직 최적화 카카오 로그인 구현 UserInfo Model 생성: 사용자 정보 관리	사용자 회원가입 코드 작성
Week 11 12/1~12/7	회원가입 작성 폼 구현 기존 서버 통신에 kakaoId 추가	회원가입 구현, 클라이언트 id 에 맞는 데이터 전송
Week 12 12/8~12/14	카카오 토큰, 로그인 관련 예외 처리	1. 회원가입 시 사용자 체형 구분 추가 2. 다양한 소셜 로그인 제공을 고려하여 사용자 id 저장 방식 수정

5. 결론

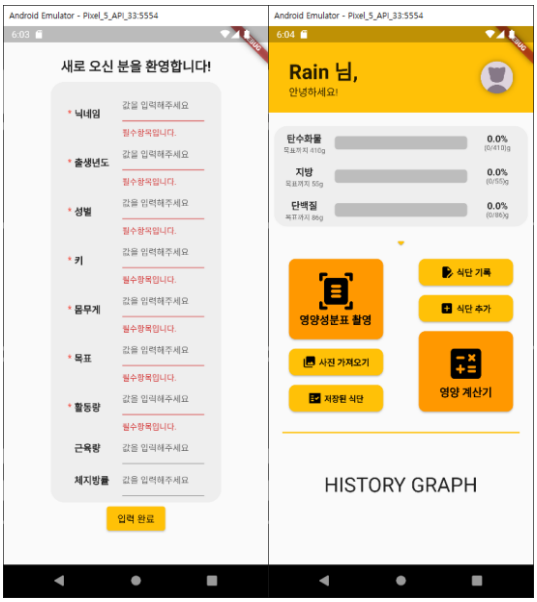
본 보고서의 프로젝트는 기존 식단 기록/관리 앱의 불편함을 해결하고 사용자에게 맞춤형 정보를 제공하고자 하는 목적 아래 진행되었다. 기존 앱은 식단 기록 시 부정확한 DB 에서 음식을 직접 검색해야 하는 불편함이 있었다. 이 프로젝트에서는 식단을 기록하는 과정을 간편하게 하고자 영양성분표 사진으로부터 영양 정보를 추출하는 알고리즘을 제작했다. 음식 양에 대한 기준은 직관성을 위해 영양성분표 자체를 단위로 사용했다. 사용자별 맞춤 영양 정보를 제공하기 위해, 카카오톡 계정과 사용자 정보를 연동하여 사용자 정보를 관리했다. 개개인의 식단 기록과 목표 영양소 양을 화면에 표시하며, 사용자는 영양성분표를 촬영한 후 OCR 분석 결과만 확인하면 된다. 이후 사용자가 제출한 영양 정보를 DB 에 저장 후, 업데이트된 정보를 화면에 표시한다. 이를 통해 사용자는 오늘 섭취한 영양소 양을 확인하고 자신의 식단을 관리할 수 있다.

앱의 전체적인 흐름을 살펴보면 다음과 같다. 앱을 실행하면, 로그인 화면이 나타나며 앱의 기능을 소개한다. 카카오 로그인 버튼을 통해 로그인할 수 있다.



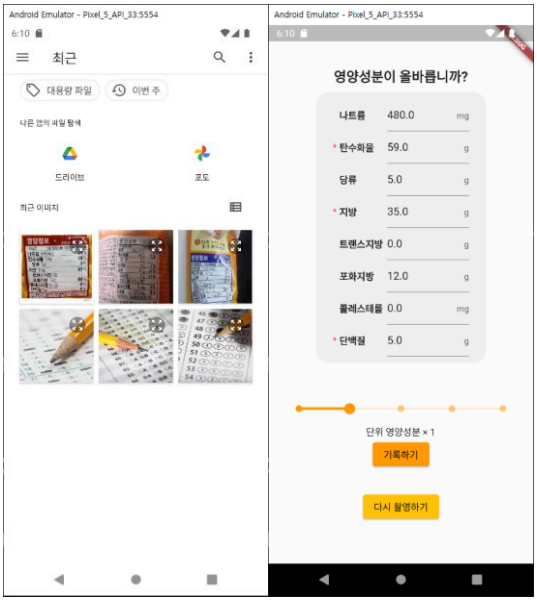
[그림 17] 로그인 화면(좌)와 카카오톡 로그인(우)

신규 회원인 경우 회원가입 화면이 나타나며 사용자 정보를 입력 받는다. 회원가입이 완료되면 홈 화면으로 이동한다.



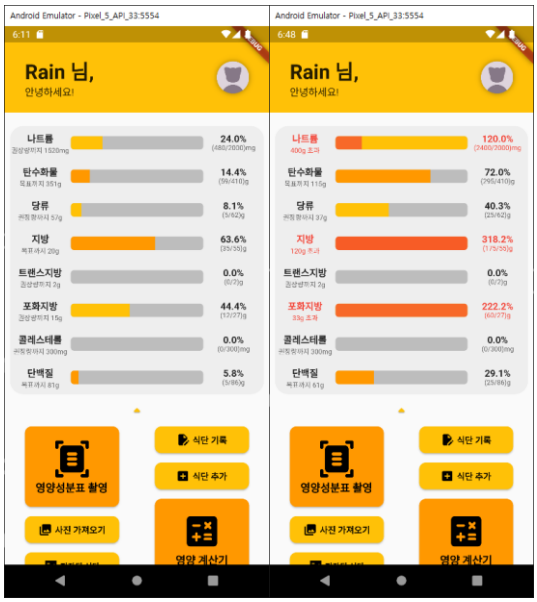
[그림 18] 회원가입 화면(좌)과 홈 화면(우)

홈 화면에서 '영양성분표 촬영' 또는 '사진 가져오기' 버튼을 누르면, 사진이 서버로 전송되어 영양 성분 정보가 추출된다. 추출된 정보는 폼 작성 화면에 선입력된다.



[그림 19] '사진 가져오기' 화면(좌)과 폼 제출 화면(우)

사용자가 영양성분표를 확인하고 먹은 양을 기록하면 홈 화면에서는 이를 반영하여 영양성분 그래프를 업데이트한다. 사용자가 권장섭취량을 초과하면 그래프의 라벨과 초과 섭취량이 적색으로 표시된다.



[그림 20] 영양성분 정보가 업데이트된 홈 화면

이번 프로젝트를 통해 사용자 요구사항을 바탕으로 한 end-to-end 서비스 개발 경험을 해볼 수 있었으며, frontend 와 backend 의 협업 과정을 몸소 느낄 수 있었다. Frontend 에서는 사용자 중심의 디자인과 interactive 한 사용자 경험을 구축하는 데 집중했으며, backend 에서는 알고리즘 설계 및 서버 DB 설계, 안정적이면서도 효율적인 서버 로직을 개발하는 데 주력했다.

협업 과정에서 소통의 중요성이 강조되었다. 매주 최소 1 회씩 대면으로 만나 프로젝트의 방향성에 대한 회의를 진행했으며, 특히 앱과 서버 간의 API 를 설계할 때 많은 소통이 필요했다. Frontend 관련 작업을 하더라도 어느정도 backend 구조를 이해하고 있어야 했고, backend 에서 역시 frontend 에서 데이터를 어떻게 처리할 예정인지 알고 있어야 했다. 자신의 영역 바깥의 부족한 정보는 팀원끼리 서로 질문하며 채워 나갔으며, 각자의 영역에서 작업한 것은 하나의 Github repository 에 commit 하며 진행상황을 공유했다.

시간 관계상 구현하지 못한 기능으로는 같은 체형의 사용자 간 식단 기록 공유 기능, 잔여 영양소 기반 식단 추천 기능, 영양 섭취 추이 그래프 기능이 있다. SW 프로젝트는 마무리되었지만, 방학 때 개발에 집중하여, 해당 기능을 모두 구현한 뒤 앱스토어와 플레이스토어에 등록할 예정이다.