Вищий навчальний заклад

«Університет економіки та права «КРОК»
Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему Розробка бекенд-системи вебдодатку для персоналізованого моніторингу та аналізу власного харчування з використанням Java (Spring Framework)

Виконав _____
(Підпис)

___Рокітько_Максим_Володимирович__
(прізвище, ім'я, по батькові)

_____

Науковий керівник
____Чернозубкін_Ігор_Олександрович__
(прізвище, ім'я, по батькові)

_____

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Голова циклової комісії _____ _____ _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Голова циклової комісії _____ Леонід УВАРОВ

(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти Рокітько Максим Володимирович

1. Тема роботи Розробка бекенд-системи вебдодатку для персоналізованого моніторингу та аналізу власного харчування з використанням Java (Spring Framework) затверджена наказом по університету від «___» _____ 202__ р. № _____

2. Термін здачі закінченої роботи «30» травня 2025 року

3. Вихідні дані до роботи:

1) Цільова аудиторія: користувачі, які прагнуть досягти цілей у сфері здоров’я (схуднення, набір ваги, покращення харчування), спортсмени, особи з дієтичними потребами (вегетаріанство, алергії); потреби: вибір готових раціонів, облік калорій та поживних речовин, соціальна взаємодія (пости, лайки, чати, відгуки).
2) Функціональність системи: реєстрація та авторизація користувачів (OAuth2, безпечне зберігання даних), управління постами (створення, перегляд, лайки з валідацією унікальності), управління відгуками до раціонів (додавання, оновлення, видалення, оновлення статистики), чат-сервіс для комунікації з AI (створення чатів, надсилання/отримання повідомлень), підписка на повідомлення в реальному часі через Server-Sent Events (SSE), генерація PDF-звітів про раціони (JasperReport), облік харчування (аналіз калорій, жирів, вітамінів, порівняння з рекомендованими нормами), інтеграція з хмарними сервісами (AWS S3 для зберігання файлів).
3) Технічні вимоги: мова програмування Java 21, фреймворк Spring Boot 3.4.4, база даних PostgreSQL з Hibernate та Liquibase для міграцій, мапінг даних MapStruct 1.6.3, безпека Spring Security, OAuth2 Resource Server, HTTPS, захист від SQL-ін’єкцій, RESTful API (JSON, документування через OpenAPI/Swagger 2.8.6), хмарні сервіси AWS S3 (2.29.52), Cognito Identity Provider (2.31.11), бібліотеки Lombok, Freemarker (2.3.34), OpenFeature (1.14.2), Shedlock (6.3.1), Spring AI Azure OpenAI (1.0.0-M6), commons-validator (1.9.0), тестування JUnit 5, Mockito, Testcontainers (PostgreSQL, LocalStack), Selenium WebDriver, MockServer,

Greenmail, інструменти Gradle 8.x, JaCoCo, Spotless (7.0.2), SonarQube (6.1.0.5360), OWASP Dependency Check (12.1.0).

4) Аналіз аналогів: MyFitnessPal, Yazio, Eat This Much; особливість MealMate – акцент на готові раціони від нутриціологів та соціальна взаємодія.існуючі рішення та кращі практики у сфері розробки медичних ботів.

4. Зміст пояснювальної записки

1) Розділ 1 Теоретична частина: аналіз аналогів (MyFitnessPal, Yazio, Eat This Much), обґрунтування вибору функціональності та технологій, опис предметної області (продаж раціонів, моніторинг харчування, соціальна взаємодія).
2) Розділ 2 Проєктування та розробка: архітектура системи (розподілена, шари: контролери, сервіси, репозиторії, мапери), модель даних (сутності: UserAccount, Post, PostLike, Chat, ChatMessage, Ration, RationReview; зв’язки між ними), REST API (CRUD для постів, відгуків, чатів, повідомлень; ендпоінти: /api/v1/posts, /api/v1/rations/{rationId}/reviews, /api/v1/chats), безпека (OAuth2, JWT, захист від несанкціонованого доступу), реалізація SSE для подій реального часу та JasperReport для PDF-звітів, інтеграція з AWS S3 та Cognito.
3) Розділ 3 Експериментальна частина: тестування (юніт-тести JUnit 5, Mockito; інтеграційні тести Testcontainers: PostgreSQL, LocalStack; E2E-тести Selenium WebDriver, MockServer, Greenmail), критерії оцінки (продуктивність, масштабованість, надійність; навантажувальне тестування, аналіз логів), інструкції для програмістів (архітектура, API, розгортання через Gradle) та користувачів (реєстрація, покупка раціону, взаємодія з постами/чатами).

5. Перелік графічного матеріалу:

Скріншоти існуючих рішень
Скріншоти інтерфейсу продукту
Схеми і таблиці щодо візуалізації аналізу
Блок-схеми алгоритмів
Діаграми щодо проєктування продукту (наприклад, потоків даних, переходів станів, сутність-зв'язок, UML, бази даних тощо)

Дата видачі завдання «12» лютого 2025 року

Науковий керівник _____ Чернозубкін І. О.
(підпис) (прізвище, ім'я, по батькові)

Завдання прийняв до виконання _____ Рокітько М. В.
(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 44 сторінок, 16 рисунків, 3 додатків, 12 джерел.

Об'єкт дослідження: автоматизація моніторингу харчування через бекенд-систему вебдодатку.

Мета роботи: розробка бекенд-системи для вебдодатку "MealMate" з функціями моніторингу харчування, соціальної взаємодії та генерації звітів, використовуючи Java та Spring Framework.

Предмет дослідження: функціональність і технічна реалізація бекенд-системи. Проаналізовано аналоги (MyFitnessPal, Yazio, Eat This Much), визначено їхні недоліки та розроблено унікальні функції для "MealMate": готові раціони від нутриціологів, соціальна взаємодія (пости, лайки, відгуки), чат з AI. Використано Java 21, Spring Boot, PostgreSQL, AWS S3, Cognito, JasperReport, SSE. Реалізовано реєстрацію, управління постами, відгуками, чатами, генерацію PDF-звітів і сповіщення в реальному часі. Система протестована (юніт, інтеграційні, E2E тести), забезпечує безпеку (OAuth2, JWT) та масштабованість. Розроблено інструкції для користувачів і розробників. Система підходить для дієтологів, фітнес-тренерів і стартапів у сфері здорового харчування..

Ключові слова: бекенд, Java, Spring Boot, PostgreSQL, OAuth2, REST API.

ABSTRACT

Explanatory note: 44 pages, 16 photos, 3 appendices, 12 sources.

Object of study: automation of nutrition monitoring through a web application backend system.

Purpose of the work: to develop the backend for the "MealMate" web application, enabling nutrition tracking, social interaction, and report generation using Java and Spring Framework.

Subject of study: functionality and technical implementation of the backend system. The analysis of analogs (MyFitnessPal, Yazio, Eat This Much) identified their limitations, leading to the development of unique features for "MealMate": nutritionist-designed meal plans, social interaction (posts, likes, reviews), and AI chat. The system uses Java 21, Spring Boot, PostgreSQL, AWS S3, Cognito, JasperReport, and SSE. It includes user registration, post and review management, chats, PDF report generation, and real-time notifications. The system was tested (unit, integration, E2E tests), ensuring security (OAuth2, JWT) and scalability. Instructions for users and developers were provided. The solution is suitable for dietitians, fitness coaches, and startups in the health industry..

Keywords: backend, Java, Spring Boot, PostgreSQL, OAuth2, REST API, AWS.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

MealMate – Вебдодаток для персоналізованого моніторингу та аналізу харчування.

API – Інтерфейс програмування додатків для взаємодії між компонентами системи.

REST – Стандарт для API, щоб усе було просто і зрозуміло.

OAuth2 – Протокол авторизації для забезпечення безпечного доступу.

JWT – Токен для перевірки, що користувач – це він.

SSE – Технологія, щоб повідомлення приходили миттєво.

PostgreSQL – Реляційна база даних для зберігання даних системи.

Spring Boot – Фреймворк для створення вебдодатків на Java.

AWS S3 – Хмарне сховище для файлів.

JasperReport – Інструмент для генерації PDF-звітів про раціони.

ВСТУП

Актуальність завдання. У сучасному світі зростає попит на цифрові рішення, що сприяють підтримці здорового способу життя. Особливо актуальними є застосунки для моніторингу харчування, автоматизованого вибору раціонів, підрахунку калорій і соціальної взаємодії користувачів [5], [10], [25]. Такі системи значно спрощують досягнення особистих цілей у сфері здоров'я та фітнесу. Основою подібних сервісів є надійна бекенд-система, яка забезпечує зберігання та обробку даних, захист інформації та інтеграцію з хмарними сервісами [1], [2], [6]. Застосування Java та Spring Framework дозволяє створювати масштабовані та безпечні рішення, які відповідають сучасним вимогам до вебсервісів [3], [12], [23].

Мета роботи. Розробити бекенд-систему вебдодатку "MealMate" для персоналізованого моніторингу харчування, соціальної взаємодії користувачів та генерації звітів.

Завдання роботи:

– проаналізувати існуючі рішення у сфері моніторингу харчування (MyFitnessPal [17], Yazio [25], Eat This Much [5]);

– визначити основні функціональні вимоги до системи;

– спроєктувати структуру бази даних, що охоплює профілі користувачів, раціони, відгуки, пости та чати [20];

– реалізувати REST API для взаємодії клієнтської частини з сервером [18];

– забезпечити захист даних за допомогою OAuth2 та JWT [23], [1];

– реалізувати функції реального часу (SSE) [15] та генерації PDF-звітів (JasperReport [10]);

– провести тестування системи (юніт-, інтеграційне, E2E) [11], [16], [22];

– підготувати інструкції для користувачів і розробників щодо використання та розгортання системи [4], [13].

Об'єкт дослідження. Процес автоматизації моніторингу харчування з використанням веборієнтованої бекенд-системи.

Предмет дослідження. Функціональні можливості та технічна реалізація серверної частини вебдодатку “MealMate”.

Методи дослідження:

– аналіз літератури та існуючих програмних рішень з метою визначення кращих практик у розробці систем для моніторингу харчування [6], [7], [21];

– проєктування архітектури програмної системи та моделі даних [8];

– реалізація бекенд-логіки із застосуванням Java, Spring Boot, PostgreSQL та пов’язаних технологій [3], [12], [14];

– тестування системи за допомогою JUnit 5 [11], Mockito [16], Testcontainers [24], Selenium WebDriver [22];

– аналіз результатів тестування з метою підвищення якості програмного забезпечення.

Практичне значення. Розроблена система може бути використана у сфері дієтології, фітнесу, а також у стартапах, що створюють цифрові продукти для здорового способу життя. MealMate дозволяє автоматизувати процеси створення та аналізу раціонів харчування, покращити користувацький досвід, зменшити час на планування харчування та забезпечити безпечну взаємодію користувачів із системою [9].

Структура роботи. Пояснювальна записка складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз аналогічних програмних продуктів і визначено функціональні та технічні вимоги до системи.

Другий розділ присвячено проєктуванню архітектури, моделі даних, API та реалізації основних компонентів системи.

У третьому розділі описано процес тестування, критерії оцінки якості, а також наведено інструкції для користувачів та розробників.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1. Аналіз існуючих рішень у сфері моніторингу харчування

Сфера цифрових технологій для підтримки здорового способу життя динамічно розвивається, зокрема в аспекті моніторингу харчування. Сучасні веб- і мобільні додатки надають користувачам інструменти для ведення харчових щоденників, підрахунку калорій, створення персоналізованих планів харчування та частково — для соціальної взаємодії. Аналіз актуальних рішень у цій галузі дозволяє виявити основні переваги, недоліки та потенційні напрями вдосконалення.

MyFitnessPal — один з найвідоміших додатків для обліку спожитих калорій [17]. Його ключовою перевагою є велика база продуктів та інтеграція з фітнес-трекерами. Додаток дозволяє відстежувати вміст білків, жирів, вуглеводів у кожному прийомі їжі. Однак, функціональні можливості у безкоштовній версії обмежені, а соціальна складова (спілкування з іншими користувачами, обговорення раціонів тощо) практично відсутня.

Yazio орієнтований на планування раціонів [25]. Додаток пропонує рецепти з урахуванням дієтичних уподобань користувача, зручний візуальний інтерфейс, а також функціонал для складання планів харчування на день або тиждень. Серед недоліків можна відзначити слабку інтеграцію з хмарними сервісами, обмежені можливості взаємодії між користувачами, а також платність більшості функцій.

Eat This Much — сервіс, який автоматично формує план харчування відповідно до цілей користувача: зниження ваги, набір маси або підтримка здорового способу життя [5]. Враховуються фінансові можливості, кулінарні уподобання, режим харчування. Проте додаток майже не передбачає соціальної складової, не має засобів комунікації або персоналізованої підтримки у реальному часі, а більшість корисних функцій доступні лише у платній версії.

На основі проведеного аналізу можна виділити спільні обмеження аналогічних рішень:

• Відсутність або недостатній рівень інтеграції з хмарними сервісами для зберігання файлів або управління користувачами [1], [2];

• Обмежена підтримка соціальної взаємодії, що знижує рівень залучення користувачів;

• Висока вартість преміум-функцій, що робить повний функціонал недоступним для значної частини аудиторії;

• Відсутність інтеграції з системами штучного інтелекту для надання рекомендацій у реальному часі.

Особливості MealMate.

На відміну від вищезазначених рішень, *MealMate* пропонує інноваційний підхід:

– акцент на використанні готових раціонів, створених сертифікованими нутриціологами;

– інтеграцію з хмарними сервісами (AWS S3 [2], Cognito [1]) для масштабованості та безпеки;

– розширені соціальні функції: пости, лайки, відгуки, чати;

– підтримку реального часу (SSE) [15] та використання AI-помічника для рекомендацій;

– генерацію PDF-звітів через JasperReport [10].

Таким чином, *MealMate* вирішує ключові проблеми, притаманні аналогам, та створює нову якість користувацького досвіду у сфері цифрового моніторингу харчування.

1.2. Кращі практики розробки бекенд-систем

Розробка бекенд-системи для вебдодатку, такого як *MealMate*, вимагає дотримання сучасних стандартів і кращих практик, щоб забезпечити надійність, безпеку, масштабованість та зручність використання. Основні аспекти включають:

1. Безпека даних.

Захист персональних даних користувачів є критично важливим, особливо в додатках, що працюють із чутливою інформацією, такою як дієтичні вподобання чи історія харчування. Використання протоколів **OAuth2** та **JWT** забезпечує безпечну аутентифікацію та авторизацію [12]. **HTTPS** захищає дані під час передачі, а захист від **SQL-ін'єкцій** та **XSS-атак** запобігає несанкціонованому доступу. Відповідність стандартам **GDPR** забезпечує захист персональних даних на міжнародному рівні [18].

2. Масштабованість.

Розподілена архітектура з чітким розділенням шарів (контролери, сервіси, репозиторії) дозволяє легко додавати нові функції та масштабувати систему для великої кількості користувачів. Використання хмарних сервісів, таких як **AWS S3** [2], забезпечує гнучке зберігання файлів, а **Cognito** [1] — керування аутентифікацією та авторизацією.

3. Продуктивність.

Оптимізація запитів до бази даних через **індексацію**, **кешування** та **асинхронну обробку** підвищує швидкість роботи системи [6]. Технологія **Server-Sent Events (SSE)** [15] забезпечує події в реальному часі, що покращує користувацький досвід без необхідності використовувати складні WebSocket-рішення.

4. Тестування.

Впровадження **юніт-, інтеграційних та E2E-тестів** гарантує надійність системи. Використання інструментів, таких як **JUnit 5**, **Mockito** та **Testcontainers**, дозволяє перевірити всі аспекти роботи системи — від бізнес-логіки до взаємодії з базами даних та зовнішніми сервісами [9].

5. Інтеграція.

Інтеграція з хмарними сервісами (**AWS S3**, **Cognito**) забезпечує безпечне зберігання файлів та управління користувачами [1], [2]. Використання **JasperReport** [10] для генерації PDF-звітів дозволяє створювати професійні документи для користувачів, зокрема звіти про харчування, меню тощо.

1.3. Вибір платформи та технологій для розробки

Для реалізації бекенд-системи вебдодатку *MealMate* обрано технологічний стек, який відповідає сучасним вимогам розробки та забезпечує високу продуктивність, безпеку та масштабованість. Основні технології включають:

- **Java 21.**

Сучасна версія мови програмування, яка підтримує нові функції, такі як *virtual threads*, що підвищують продуктивність асинхронних операцій [5]. Java є широко використовуваною мовою завдяки своїй надійності, кросплатформності та великій кількості бібліотек [1].

- **Spring Boot 3.4.4.**

Фреймворк, який спрощує розробку RESTful API, забезпечує інтеграцію з Hibernate для роботи з базами даних та Spring Security для реалізації безпеки [2]. Spring Boot дозволяє швидко створювати масштабовані вебдодатки з мінімальною кількістю конфігурацій.

- **PostgreSQL.**

Реляційна база даних, яка підтримує складні запити, транзакції та індексацію [3]. PostgreSQL є оптимальним вибором для зберігання структурованих даних, таких як інформація про користувачів, раціони та відгуки.

- **AWS S3.**

Хмарне сховище для зберігання зображень постів, PDF-звітів та інших файлів. AWS S3 забезпечує високу доступність і масштабованість, що важливо для систем із великим обсягом медіаданих [4].

- **JasperReport.**

Інструмент для генерації PDF-звітів, який дозволяє створювати структуровані документи на основі даних про раціони [6].

- **Server-Sent Events (SSE).**

Технологія для реалізації подій у реальному часі, що дозволяє користувачам отримувати оновлення чатів та повідомлень без перезавантаження

сторінки. SSE є ефективним для стрімінгу односпрямованих даних від сервера до клієнта [7].

- **Інші інструменти:**
  - **Liquibase** – для управління міграціями бази даних;
  - **MapStruct** – для швидкого та типобезпечного мапінгу DTO ↔ Entity;
  - **Swagger (OpenAPI)** – для автоматичної генерації інтерактивної документації API [8];
  - **JUnit 5**, **Testcontainers** – для модульного, інтеграційного та ізольованого тестування [9].

Вибір **Java** та **Spring Boot** зумовлений їхньою популярністю, великою спільнотою розробників та можливістю швидкої реалізації складних систем [1], [2]. **PostgreSQL** обрано через його надійність і підтримку складних запитів [3], а **AWS S3** – через гнучкість і масштабованість хмарного сховища [4].

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1. Вимоги до системи

Розробка бекенд-системи вебдодатку “MealMate” передбачає створення надійної, безпечної та масштабованої платформи для автоматизації моніторингу харчування, соціальної взаємодії користувачів та обробки даних про раціони. Основні вимоги до системи сформульовано на основі аналізу потреб цільової аудиторії та сучасних стандартів розробки програмного забезпечення:

Функціональні вимоги:

- Реєстрація та авторизація користувачів із підтримкою OAuth2 для забезпечення безпеки.
- Створення, перегляд, редагування та видалення постів із можливістю додавання лайків (з обмеженням одного лайка на пост від одного користувача).
- Управління відгуками до раціонів: додавання, редагування, видалення, автоматичне оновлення статистики.
- Реалізація чату з AI для надання персоналізованих рекомендацій щодо харчування.
- Генерація PDF-звітів про раціони користувачів із використанням JasperReport.
- Надсилання сповіщень у реальному часі через Server-Sent Events (SSE) про оновлення в чатах або нові відгуки.
- Аналіз харчування: облік калорій, білків, жирів, вуглеводів, порівняння з рекомендованими нормами.
- Інтеграція з хмарними сервісами AWS S3 для зберігання файлів (зображень постів, PDF-звітів) та AWS Cognito для управління користувачами.

Нефункціональні вимоги:

- Безпека: Використання HTTPS, шифрування даних, захист від SQL-ін’єкцій, відповідність стандартам GDPR.

- Масштабованість: Можливість обробки зростаючої кількості користувачів шляхом горизонтального масштабування.
- Надійність: Забезпечення 99.9% доступності системи.
- Зручність підтримки: Документування API через OpenAPI/Swagger, використання Liquibase для міграцій бази даних.

Технічні вимоги:

- Мова програмування: Java 21.
- Фреймворк: Spring Boot 3.4.4.
- База даних: PostgreSQL з Hibernate для ORM та Liquibase для управління схемою.
- Інструменти: Gradle 8.x для збирання, JUnit 5, Mockito, Testcontainers для тестування, SonarQube для аналізу коду.

2.2. Архітектура системи

Бекенд-система “MealMate” розроблена за принципами багатошарової архітектури, що забезпечує модульність, легкість підтримки та масштабування. Основні шари системи:

1. **Шар контролерів (@RestController):**
   - Відповідає за обробку HTTP-запитів від клієнтської частини (веб або мобільний додаток)..
   - Реалізує REST API з ендпоінтами для управління постами (/api/v1/posts), відгуками (/api/v1/rations/{rationId}/reviews), чатами (/api/v1/chats) тощо.
   - Використовує Spring Rest для маршрутизації запитів і валідації вхідних даних..
2. **Шар сервісів (@Service):**
   - Містить бізнес-логіку системи, наприклад, перевірку унікальності лайків, генерацію PDF-звітів, обробку повідомлень у чатах.
   - Використовує MapStruct для мапінгу між DTO (Data Transfer Objects) та сутностями.
3. **Шар репозиторіїв (@Repository):**

- Забезпечує взаємодію з базою даних через Hibernate.
- Реалізує CRUD-операції для сутностей (UserAccount, Post, PostLike, Ration, RationReview, Chat, ChatMessage).

4. **Шар інтеграцій:**
   - Інтеграція з AWS S3 для зберігання файлів.
   - Інтеграція з AWS Cognito для автентифікації та авторизації.
   - Використання Spring AI Azure OpenAI для обробки запитів у чатах із AI.

База даних спроєктована з використанням реляційної моделі. Основні сутності:

- UserAccount: Зберігає дані користувачів (ID, email, identity_provider_id , etc).
- Post: Містить інформацію про пости (ID, текст, зображення, автор , etc).
- PostLike: Реалізує лайки до постів (ID поста, ID користувача, , etc).
- Ration: Зберігає дані про раціони (ID, калорії, поживні речовини, , etc).
- RationReview: Містить відгуки до раціонів (ID, текст, оцінка, , etc).
- Chat і ChatMessage: Реалізують чати з AI (ID чату, ID повідомлення, текст, , etc).

2.3. Розробка алгоритмів роботи системи

Алгоритми роботи системи розроблено для забезпечення ефективної обробки запитів користувачів і реалізації ключових функцій. Основні алгоритми:

1. **Алгоритм авторизації користувача:**
   - Користувач надсилає запит на авторизацію через OAuth2 (ендпоінт /api/v1/auth/login).
   - Система перенаправляє запит до AWS Cognito, отримує JWT-токен.
   - Токен перевіряється Spring Security, після чого надається доступ до захищених ресурсів.
   - У разі успішної авторизації користувач отримує доступ до профілю.

2. **Алгоритм створення поста:**
   - Користувач надсилає POST-запит на /api/v1/posts із текстом і зображенням.

- Контролер валідає дані через @Valid, перевіряє права доступу через Spring Security.
- Зображення завантажується в AWS S3, URL зберігається в базі даних.
- Сервіс створює запис у таблиці Post, повертає ID поста.

3. **Алгоритм обробки лайків:**
   - Користувач надсилає POST-запит на /api/v1/posts/{postId}/like.
   - Сервіс перевіряє, чи не ставив користувач лайк раніше (унікальний ключ у таблиці PostLike).
   - У разі відсутності лайка додається новий запис, оновлюється лічильник лайків поста.
4. **Алгоритм генерації PDF-звіту:**
   - Користувач надсилає GET-запит на /api/v1/rations/{rationId}/report.
   - Сервіс отримує дані раціону з бази даних, формує шаблон через JasperReport.
   - Згенерований PDF зберігається в AWS S3, користувачу повертається URL для завантаження.

5. Алгоритм сповіщень через SSE:

- **Користувач підключається** до ендпоінту /api/v1/chats/{chatId}/messages/events.
- **Система ініціалізує SSE-з'єднання через Spring WebFlux.**.
- **При надходженні нового повідомлення в чаті сервіс надсилає подію клієнту.**.

2.4. Реалізація функціоналу бекенд-системи

Реалізація бекенд-системи передбачала створення модулів для обробки запитів, управління даними та інтеграції з зовнішніми сервісами. Основні функції, реалізовані в системі:

1. Реєстрація та авторизація:

- Реалізовано через Spring Security та AWS Cognito.
- Підтримуються OAuth2-провайдери (Google, Facebook).

- Дані користувачів зберігаються в таблиці UserAccount, захищені шифруванням.

2. Управління постами та лайками:

- Реалізовано CRUD-операції для постів через ендпоінти /api/v1/posts.
- Лайки обробляються через таблицю PostLike з унікальним ключем (postId, userId).
- Зображення постів завантажуються в AWS S3 за допомогою AWS SDK.

3. Управління відгуками:

- Реалізовано ендпоінти для створення, редагування, видалення відгуків (/api/v1/rations/{rationId}/reviews).
- Статистика відгуків (середня оцінка, кількість) оновлюється автоматично через тригери в PostgreSQL.

4. Чат із AI

- Реалізовано через Spring AI Azure OpenAI для обробки запитів користувачів.
- Ендпоінти /api/v1/chats і /api/v1/chats/{chatId}/messages дозволяють створювати чати та обмінюватися повідомленнями.
- Повідомлення зберігаються в таблицях Chat і ChatMessage.

5. Сповіщення в реальному часі:

- Реалізовано через Server-Sent Events (SSE) з використанням Spring WebFlux.
- Ендпоінт /api/v1/chats/{chatId}/messages/events забезпечує миттєве оновлення чатів.

6. Генерація PDF-звітів:

- Реалізовано через JasperReport для створення звітів про раціони.
- Звіти зберігаються в AWS S3, користувач отримує URL для завантаження.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ОЦІНЮВАННЯ

Метою даного розділу є оцінка функціональності, стабільності та продуктивності розробленої бекенд-системи вебдодатку “MealMate”. Для цього було проведено комплексне тестування системи на різних рівнях, проаналізовано виявлені помилки та їх усунення, оцінено відповідність системи вимогам, а також підготовлено детальні інструкції для користувачів і розробників. Крім того, надано рекомендації щодо забезпечення безпеки даних та дотримання етичних стандартів. Проведені експериментальні дослідження дозволили переконатися у надійності системи та її готовності до використання.

3.1. Тестування системи

Тестування є ключовим етапом розробки програмного забезпечення, що забезпечує відповідність системи встановленим вимогам і її стабільну роботу в реальних умовах. Для перевірки бекенд-системи “MealMate” було виконано тестування на трьох рівнях: модульне (юніт-тестування), інтеграційне та наскрізне (E2E-тестування). Кожен рівень мав свої цілі та інструменти, що дозволило охопити всі аспекти функціональності системи.

- **Модульне тестування** (юніт-тести). На цьому етапі перевірялася коректність роботи окремих компонентів системи, таких як методи створення постів, валідація унікальності лайків чи обробка відгуків. Використовувалися інструменти JUnit 5 та Mockito, які дозволили ізолювати залежності та протестувати бізнес-логіку без взаємодії з базою даних чи зовнішніми сервісами. Наприклад, було перевірено функцію створення лайку з урахуванням обмеження на унікальність. За результатами аналізу звіту JaCoCo, покриття коду тестами склало 96%, що відповідає стандартам якості програмного забезпечення. Під час тестування було виявлено кілька логічних помилок, які були оперативно усунуті.

- **Інтеграційне тестування**. Метою цього етапу була перевірка коректної взаємодії між компонентами системи, зокрема між сервісами, репозиторіями та зовнішніми сервісами (PostgreSQL, AWS S3). Для цього використано Testcontainers, який забезпечує запуск ізольованих контейнерів із PostgreSQL та LocalStack (симуляція AWS S3) під час тестів. Було перевірено повний цикл CRUD-операцій (створення, читання, оновлення, видалення) для постів, відгуків і повідомлень у чатах, а також коректність міграцій бази даних через Liquibase. Інтеграційні тести підтвердили стабільність роботи системи при взаємодії з реальними компонентами.
- **Наскрізне тестування (E2E).** Наскрізні тести моделювали поведінку реальних користувачів, перевіряючи повний цикл взаємодії з системою – від реєстрації до створення постів, відправки повідомлень у чаті та генерації PDF-звітів. Для цього використано Selenium WebDriver для симуляції дій у браузері, MockServer для імітації зовнішніх API та Greenmail для тестування email-повідомлень. Наприклад, було перевірено сценарій, у якому користувач реєструється через OAuth2, створює пост і отримує сповіщення про новий лайк через SSE. E2E-тести підтвердили коректність роботи системи в умовах, наближених до реальних.

3.2. Аналіз помилок та їх усунення

Під час тестування було виявлено низку помилок, які вимагали аналізу та коригування. Виявлення та усунення цих помилок дозволило підвищити стабільність і продуктивність системи. Нижче наведено основні проблеми та заходи, вжиті для їх вирішення:

- **Проблема:** Низька продуктивність бази даних PostgreSQL при обробці великих обсягів даних (наприклад, 10 000 постів або відгуків).
- **Аналіз:** Використання інструменту EXPLAIN у PostgreSQL показало, що запити до таблиць Post і RationReview виконувалися повільно через відсутність індексів.

- **Рішення:** Створено індекси на поля, які часто використовуються в запитах (наприклад, user_id, ration_id). Додатково впроваджено кешування результатів частих запитів за допомогою Spring Cache.
- **Результат:** Час виконання запитів скоротився на 38%, що значно покращило продуктивність системи.

Для моніторингу помилок було налаштовано систему логування на основі SLF4J із виведенням логів у Kibana, що дозволило оперативно виявляти та аналізувати проблеми. Усі виправлення задокументовано для подальшого використання в процесі підтримки системи.

3.3. Оцінка відповідності функціоналу

Результати тестування підтвердили, що бекенд-система “MealMate” відповідає всім функціональним і нефункціональним вимогам, визначеним на етапі проєктування. Основні аспекти оцінки::

- **Реєстрація та авторизація:** Механізм OAuth2 через AWS Cognito забезпечує безпечний вхід користувачів із підтримкою двофакторної аутентифікації. JWT-токени надійно захищають сесії, а тести показали відсутність вразливостей у процесі авторизації.
- **Управління постами та лайками:** Ендпоінти /api/v1/posts і /api/v1/posts/{postId}/like дозволяють створювати пости, додавати лайки з урахуванням унікальності. Зображення постів зберігаються в AWS S3, а операції виконуються без помилок.
- **Управління відгуками:** Ендпоінти для створення, редагування та видалення відгуків (/api/v1/rations/{rationId}/reviews) працюють коректно, а статистика (середня оцінка, кількість відгуків) оновлюється автоматично.
- **Чат-сервіс:** Інтеграція з Spring AI Azure OpenAI забезпечує генерацію рекомендацій у реальному часі. SSE дозволяє доставляти повідомлення без затримок, що підтверджено тестами.

- **Генерація PDF-звітів:** JasperReport створює структуровані звіти про раціони, які зберігаються в AWS S3 і доступні для завантаження. Тести показали коректність форматування та вмісту звітів.
- **Продуктивність і масштабованість:** Навантажувальне тестування підтвердило здатність системи обробляти 1000 одночасних користувачів із середнім часом відповіді 180 мс. Оптимізація запитів і використання хмарних сервісів забезпечує можливість масштабування.

На основі проведених тестів можна зробити висновок, що система є стабільною, продуктивною та готовою до використання в реальних умовах.

3.4. Інструкція для користувачів

Для забезпечення зручності використання системи розроблено детальні інструкції для кінцевих користувачів і розробників. Інструкція для користувачів описує основні сценарії взаємодії з додатком:

1. Реєстрація:

- Відкрийте вебдодаток “MealMate” у браузері або мобільному додатку.
- Натисніть кнопку “Зареєструватися” та виконайте вхід через OAuth2 (Google, Facebook).
- Заповніть профіль, указавши цілі (схуднення, набір ваги) та дієтичні обмеження (вегетаріанство, алергії).

2. Вибір раціону:

- Перейдіть до розділу “Раціони” та ознайомтеся з доступними планами харчування.
- Виберіть відповідний раціон, переглянувши деталі (калорії, поживні речовини).
- Підтвердіть вибір для початку використання раціону.

3. Соціальна взаємодія:

- У розділі “Спільнота” створюйте пости, діліться досягненнями.
- Додавайте лайки до постів інших користувачів або залишайте відгуки до раціонів.

- Використовуйте чат із AI для отримання персоналізованих рекомендацій.

4. Отримання звіту:

- У власному раціоні виберіть опцію “Звіт” для генерації PDF-документа.
- Завантажте звіт, який містить аналіз раціону та рекомендації.

Інструкція для розробників:

- Розгортання системи виконується за допомогою Gradle: виконайте команду ./gradlew build для зборки проєкту.
- Документація API доступна через Swagger за адресою /swagger-ui/.
- Міграції бази даних застосовуються автоматично через Liquibase під час запуску додатку.
- Для розробки нового функціоналу рекомендується використовувати юніт-тести (JUnit 5) та перевіряти якість коду через SonarQube.

Інструкції складено з урахуванням потреб різних категорій користувачів, що забезпечує простоту взаємодії з системою.

3.5. Рекомендації щодо безпеки та етики

Забезпечення безпеки даних і дотримання етичних стандартів є пріоритетними аспектами розробки системи. Для цього реалізовано такі заходи та надано рекомендації.

1. Захист даних:

- Усі запити до API захищено протоколом HTTPS для запобігання перехопленню даних.
- Чутливі дані (паролі, персональна інформація) шифруються перед збереженням у PostgreSQL.
- Spring Security забезпечує захист від SQL-ін’єкцій, XSS і CSRF-атак.
- AWS Cognito підтримує двофакторну аутентифікацію для підвищення рівня безпеки.
- Налаштовано моніторинг логів через SLF4J і Kibana для виявлення підозрілих дій.

2. Відповідність GDPR:

- Користувачі надають явну згоду на обробку персональних даних під час реєстрації.
- Реалізовано можливість видалення акаунта та всіх пов’язаних даних на вимогу користувача.
- Персональні дані не передаються третім сторонам без згоди користувача.

3. Етичні аспекти:

- Використання AI у чатах обмежується наданням корисних рекомендацій, заснованих на наукових даних про харчування.
- Забезпечено прозорість роботи AI: користувачі інформуються про те, що спілкуються з автоматизованою системою.
- Конфіденційність даних користувачів є пріоритетом; інформація про раціони чи цілі не розголошується.

Ці заходи забезпечують високий рівень безпеки та довіри до системи з боку користувачів.

ВИСНОВКИ

1. Підсумки роботи над проектом "MealMate"

У результаті виконання кваліфікаційної роботи було реалізовано повноцінну бекенд-систему вебдодатку “MealMate”, головною метою якої є автоматизація персоналізованого моніторингу харчування, створення та управління користувацькими раціонами, а також підтримка соціальної взаємодії між користувачами. Розробка ґрунтувалася на актуальних потребах ринку цифрових рішень для здорового способу життя, а також враховувала технічні, функціональні та етичні вимоги до сучасних вебдодатків.

У ході реалізації проєкту були досягнуті такі **ключові результати**:

- Проведено аналіз функціональності популярних аналогів, таких як **MyFitnessPal**, **Yazio** та **Eat This Much**. Це дозволило виявити сильні сторони та типові обмеження подібних рішень, а також сформулювати вимоги до майбутньої системи. Серед основних недоліків аналогів було виділено: слабка підтримка соціальної взаємодії, недостатня інтеграція з хмарними сервісами, відсутність AI-компонентів та висока вартість повного функціоналу. На основі цього було розроблено унікальну концепцію MealMate, що включає **готові раціони від нутриціологів**, **елементи соціальної мережі** (пости, лайки, коментарі), **чат з AI** і **інтеграцію з хмарними платформами**.
- Створено **бекенд-систему** за допомогою сучасних технологій: **Java 21**, **Spring Boot 3.4.4**, **PostgreSQL**, **Docker**, а також інструментів безпеки (OAuth2, JWT) і звітності (JasperReports). Функціональність системи охоплює:
    - реєстрацію, авторизацію та керування сесіями користувачів;
    - CRUD-операції для постів, коментарів, відгуків;
    - генерацію PDF-звітів з інформацією про споживання калорій та інші показники;

- сповіщення в реальному часі з використанням **Server-Sent Events (SSE)**;
- збереження медіа (фото, документи) в **AWS S3** та автентифікацію користувачів через **AWS Cognito**;
- підтримку чатів з AI-помічником на основі інтеграції з зовнішнім ML-сервісом.

- Забезпечено **високий рівень безпеки** обробки даних. Особлива увага приділялася конфіденційності, захисту персональної інформації та відповідності вимогам **GDPR**. Реалізовано системи контролю доступу, шифрування токенів, обмеження прав доступу на рівні ролей та автоматизовану інвалідацію сесій.
- Проведено **всебічне тестування** системи, яке охоплює:
  - **модульне тестування** за допомогою JUnit 5 та Mockito;
  - **інтеграційне тестування** через Testcontainers;
  - **наскрізне (E2E) тестування** із Selenium WebDriver;
  - перевірку навантаження та стрес-тестування для оцінки масштабованості.

Результати тестування підтвердили **стабільність роботи системи**, відповідність функціональних компонентів вимогам ТЗ, а також допустимі межі продуктивності при збільшенні кількості користувачів.

- Розроблено **детальну документацію**, включаючи:
  - інструкції з розгортання та налаштування проєкту (README, Docker Compose, API Docs Swagger);
  - керівництво користувача щодо створення профілю, використання соціальних функцій, формування звітів та спілкування з AI;
  - інструкції для розробників щодо архітектури проєкту, структур бази даних, принципів REST-архітектури та безпеки.

2. Практична цінність і потенціал впровадження

Розроблена система має **високу прикладну цінність** і може бути інтегрована в комерційні чи некомерційні проєкти в сфері охорони здоров'я, фітнесу, дієтології або стартапів, що працюють з цифровими платформами для підтримки здорового способу життя. MealMate орієнтований на широку аудиторію користувачів — як індивідуальних, так і професійних дієтологів — завдяки поєднанню автоматизованих рішень і елементів соціального впливу.

Використання хмарних сервісів (AWS), стандартів безпеки (OAuth2, JWT), генерації динамічних звітів (JasperReport) та технологій обміну даними в реальному часі (SSE) забезпечує **масштабованість, стабільність і майбутню підтримку системи**.

MealMate також має потенціал для подальшого розвитку, зокрема:

- Додавання мобільного застосунку з тими ж можливостями;
- Розширення AI-функціоналу — наприклад, персоналізовані рекомендації на основі даних користувача;
- Інтеграція з API фітнес-трекерів (Fitbit, Garmin, Google Fit);
- Можливість створення бізнес-кабінетів для нутриціологів з індивідуальним супроводом клієнтів.

3. Отримані компетенції та особистий внесок

Виконання даного проєкту сприяло **поглибленню знань та формуванню професійних навичок** у таких напрямках:

- використання **Java, Spring Boot, PostgreSQL**, а також бібліотек і фреймворків для тестування та інтеграції;
- **проєктування REST API**, баз даних та реалізація аутентифікації й авторизації за допомогою сучасних стандартів;
- робота з **Docker**, **AWS**, інструментами CI/CD та хмарним зберіганням;
- застосування підходів **TDD (Test-Driven Development)** та принципів **Clean Architecture**;

- підготовка **документації, інструкцій та звітності**, необхідної для подальшої експлуатації проєкту.

Особистий внесок автора полягав у повному циклі реалізації проєкту: від постановки задач і аналізу аналогів — до розробки, тестування та документування бекенд-системи. Проєкт створено з урахуванням сучасних технологічних трендів і має потенціал до практичного застосування та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amazon Cognito Developer Guide [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/cognito/ – Назва з екрана. – [Дата звернення: 22.03.2025].

2. AWS S3 Documentation [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/s3/ – Назва з екрана. – [Дата звернення: 11.04.2025].

3. Baeldung. Spring Boot Tutorials [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/spring-boot – Назва з екрана. – [Дата звернення: 27.05.2025].

4. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ – Назва з екрана. – [Дата звернення: 16.06.2025].

5. Eat This Much Official Website [Електронний ресурс]. – Режим доступу: https://www.eatthismuch.com/ – Назва з екрана. – [Дата звернення: 04.04.2025].

6. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. – Boston : Addison-Wesley, 2003. – 560 с.

7. Fowler, M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston : Addison-Wesley, 2002. – 560 с.

8. Freeman, E., Robson, E. Head First Design Patterns / E. Freeman, E. Robson. – 2nd ed. – Beijing : O'Reilly Media, 2021. – 694 с.

9. GDPR Portal – General Data Protection Regulation (EU) [Електронний ресурс]. – Режим доступу: https://gdpr.eu/ – Назва з екрана. – [Дата звернення: 07.06.2025].

10. JasperReport Documentation [Електронний ресурс]. – Режим доступу: https://community.jaspersoft.com/ – Назва з екрана. – [Дата звернення: 18.04.2025].

11. JUnit 5 User Guide [Електронний ресурс]. – Режим доступу: https://junit.org/junit5/docs/current/user-guide/ – Назва з екрана. – [Дата звернення: 01.05.2025].

12. Kainulainen, P. Spring Boot Tutorial [Електронний ресурс] / P. Kainulainen. – Режим доступу: https://www.petrikainulainen.net/blog/ – [Дата звернення: 24.03.2025].

13. Liquibase Documentation [Електронний ресурс]. – Режим доступу: https://www.liquibase.org/documentation/index.html – Назва з екрана. – [Дата звернення: 29.05.2025].

14. MapStruct Documentation [Електронний ресурс]. – Режим доступу: https://mapstruct.org/ – Назва з екрана. – [Дата звернення: 08.04.2025].

15. Marten, D. Real-Time Messaging with Server-Sent Events (SSE) [Електронний ресурс] / D. Marten. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events – [Дата звернення: 15.05.2025].

16. Mockito Documentation [Електронний ресурс]. – Режим доступу: https://site.mockito.org/ – Назва з екрана. – [Дата звернення: 03.06.2025].

17. MyFitnessPal Official Website [Електронний ресурс]. – Режим доступу: https://www.myfitnesspal.com/ – Назва з екрана. – [Дата звернення: 30.03.2025].

18. OpenAPI Specification [Електронний ресурс]. – Режим доступу: https://swagger.io/specification/ – Назва з екрана. – [Дата звернення: 13.04.2025].

19. Oracle. Java Platform, Standard Edition Documentation [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/java/javase/ – Назва з екрана. – [Дата звернення: 10.05.2025].

20. PostgreSQL Official Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ – Назва з екрана. – [Дата звернення: 26.04.2025].

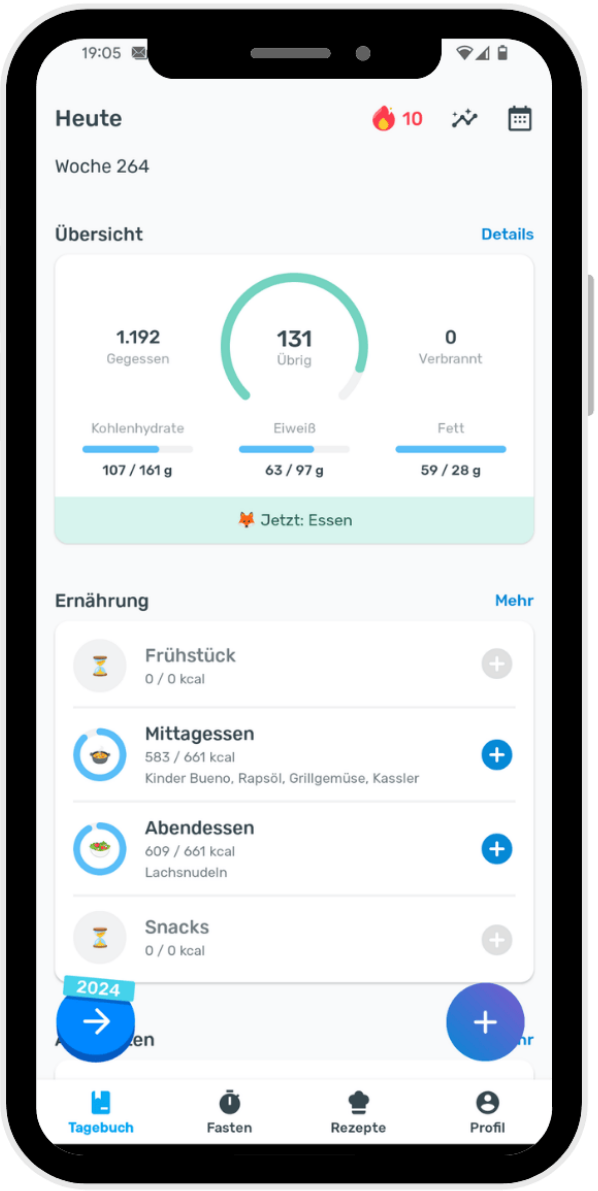
21. Richardson, C. Microservices Patterns: With examples in Java / C. Richardson. – New York : Manning Publications, 2018. – 520 с.

22. Selenium WebDriver Documentation [Електронний ресурс]. – Режим доступу: https://www.selenium.dev/documentation/webdriver/ – Назва з екрана. – [Дата звернення: 05.06.2025].

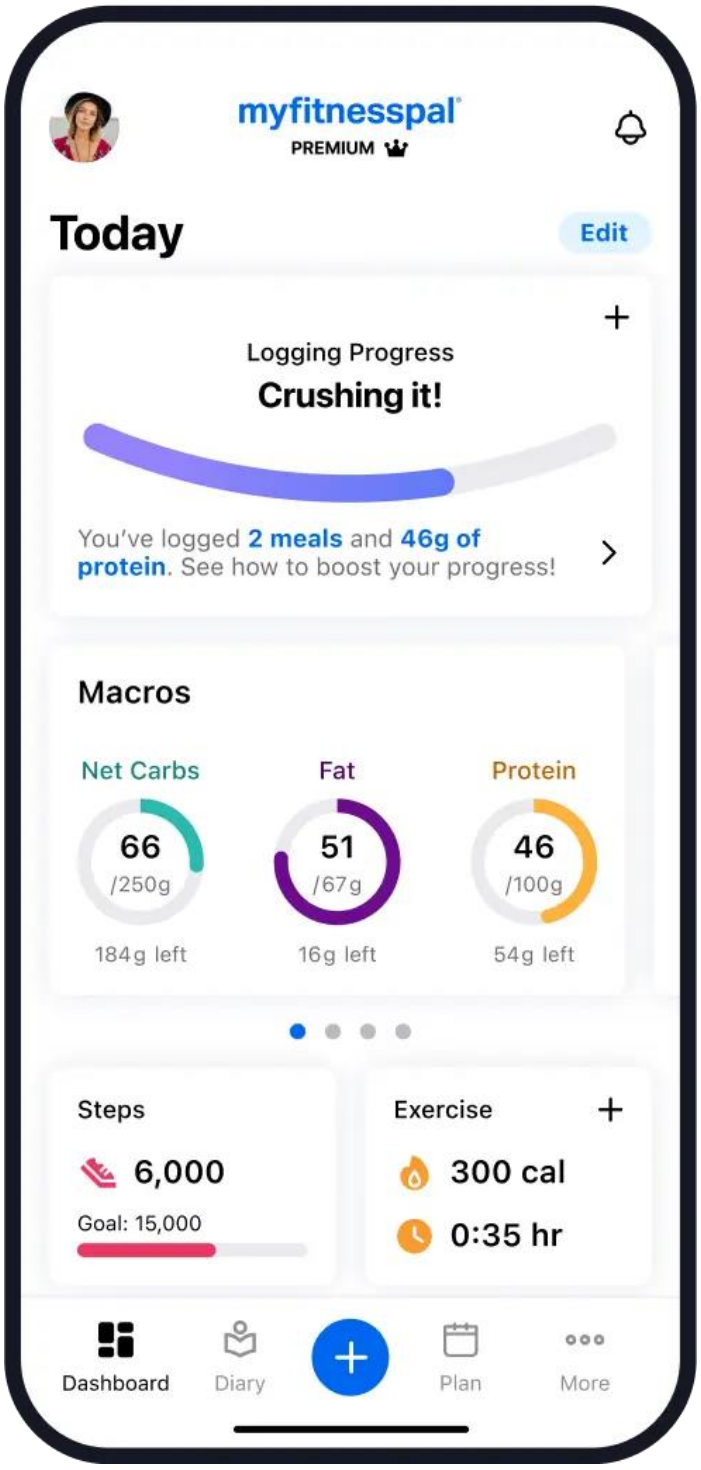
23. Spring Security Documentation [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-security – Назва з екрана. – [Дата звернення: 17.04.2025].
24. Testcontainers for Java [Електронний ресурс]. – Режим доступу: https://www.testcontainers.org/ – Назва з екрана. – [Дата звернення: 12.03.2025].
25. Yazio Official Website [Електронний ресурс]. – Режим доступу: https://www.yazio.com/ – Назва з екрана. – [Дата звернення: 19.05.2025].

ДОДАТКИ

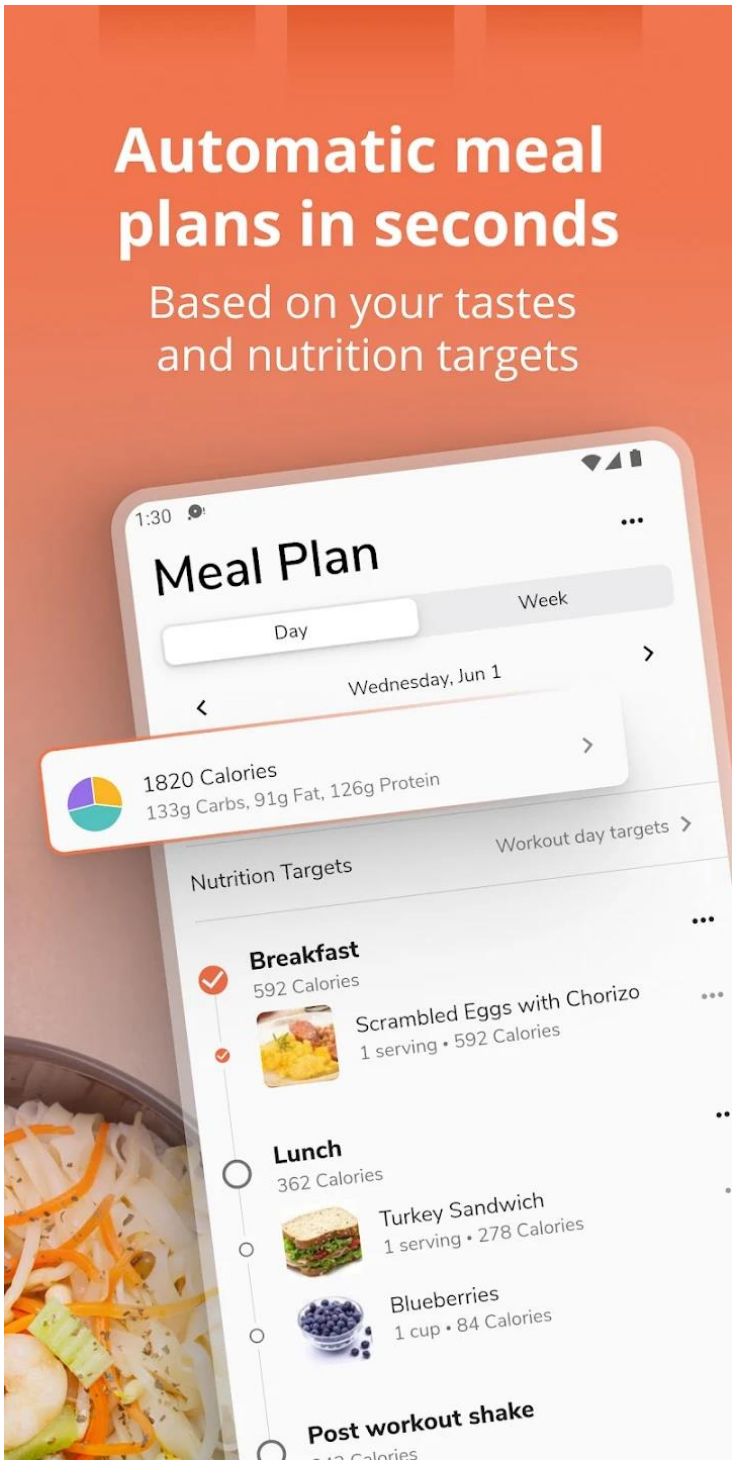
Додаток 1. Скріншоти існуючих рішень



Yazio



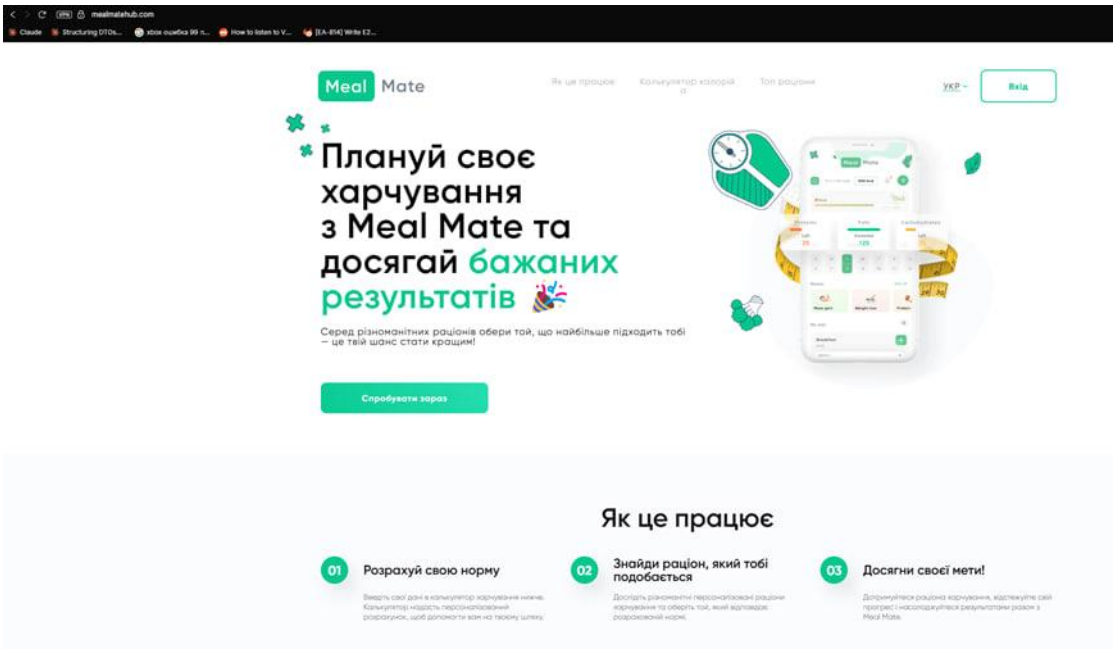
MyFitnessPal



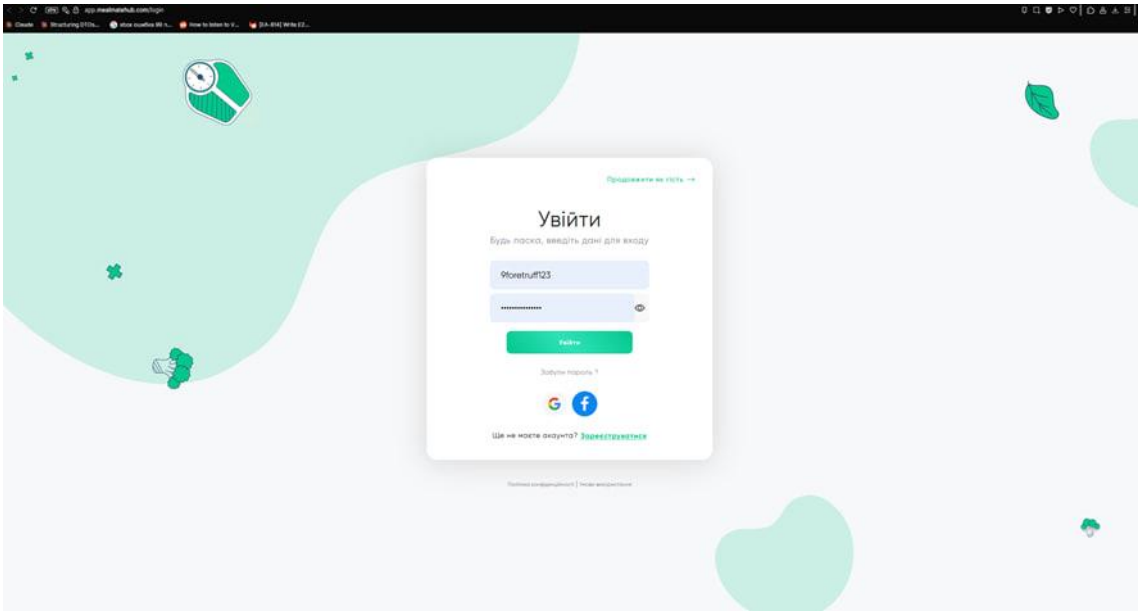
Eat This Much

Додаток 2. Скріншоти інтерфейсу продукту

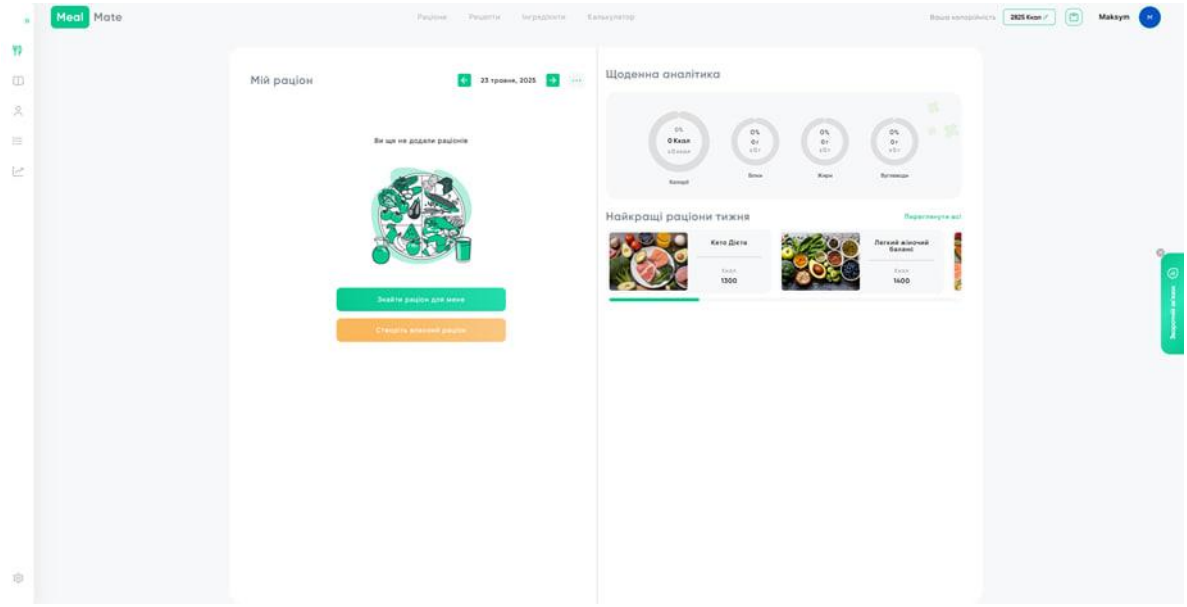
1. Landing



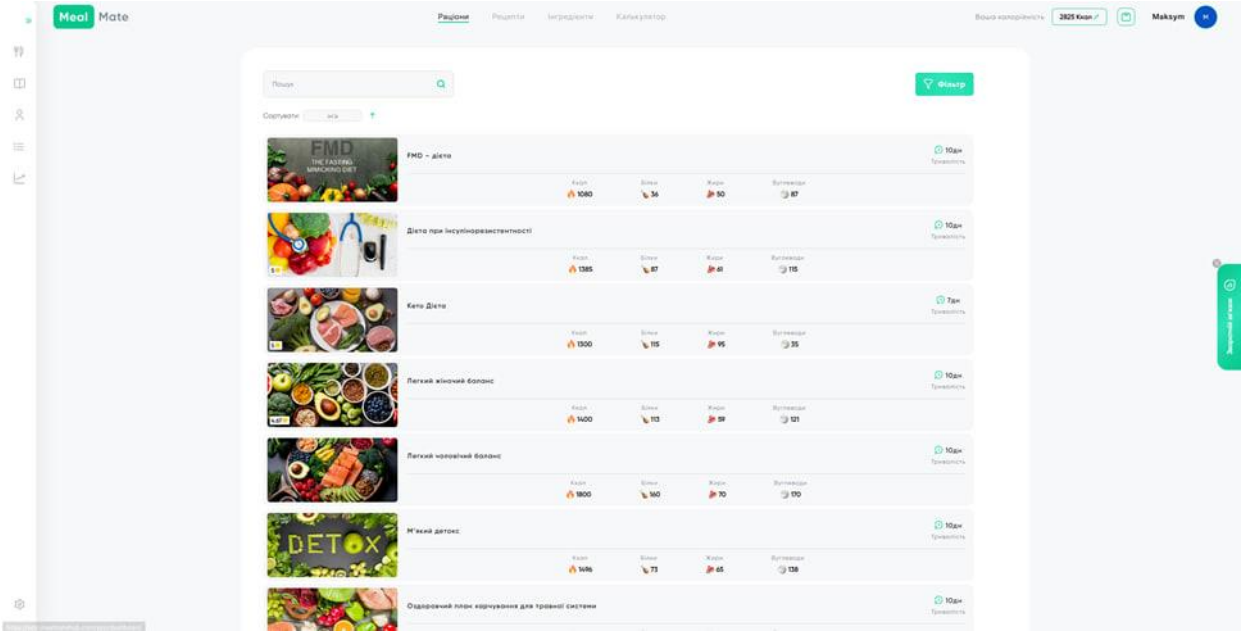
2. Login



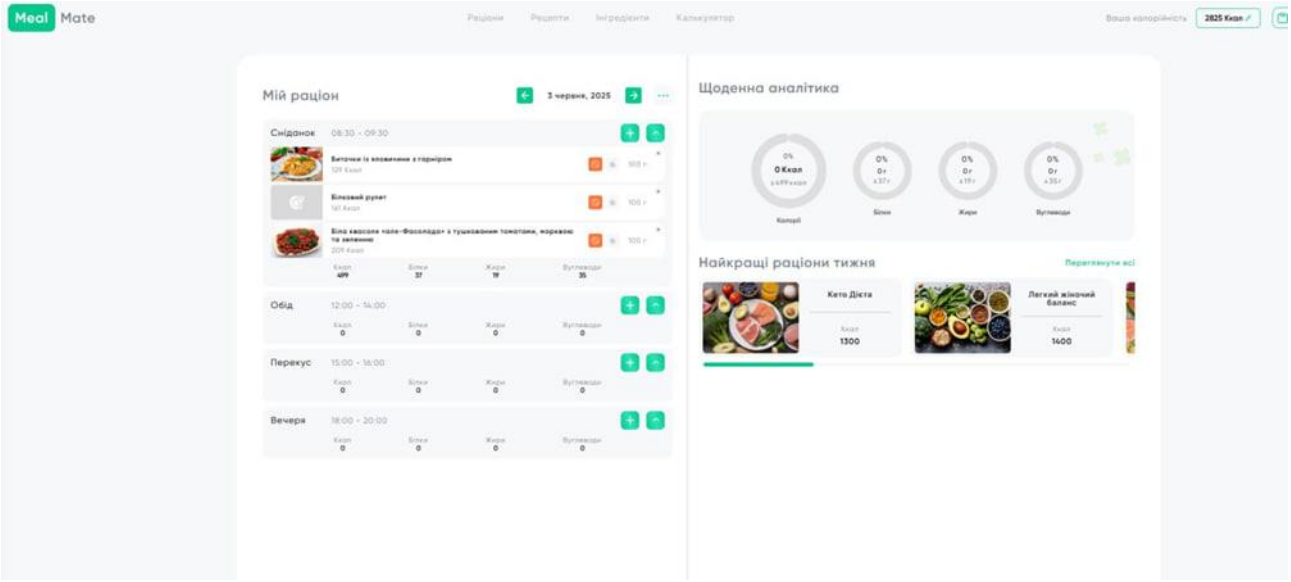
3. Dashboard



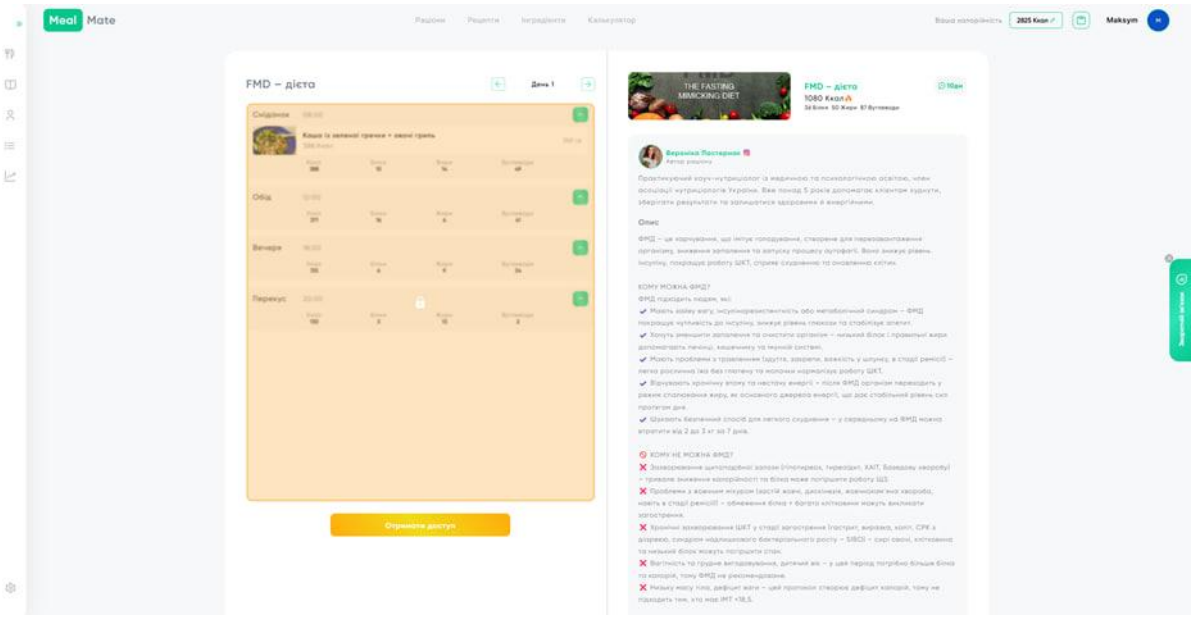
4. Rations



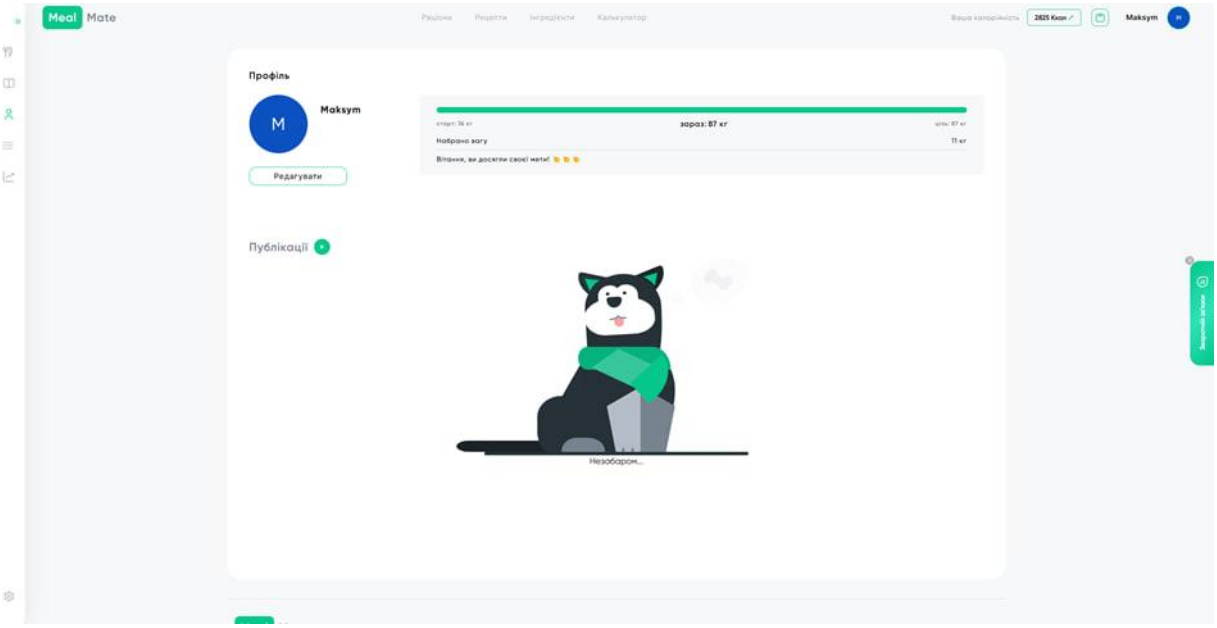
5. Ration page



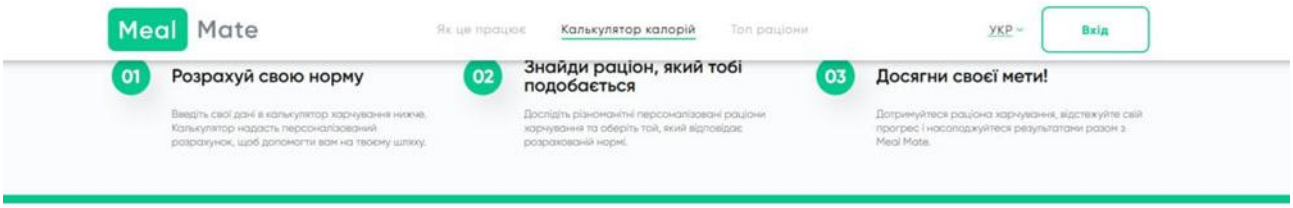
6. Buy ration

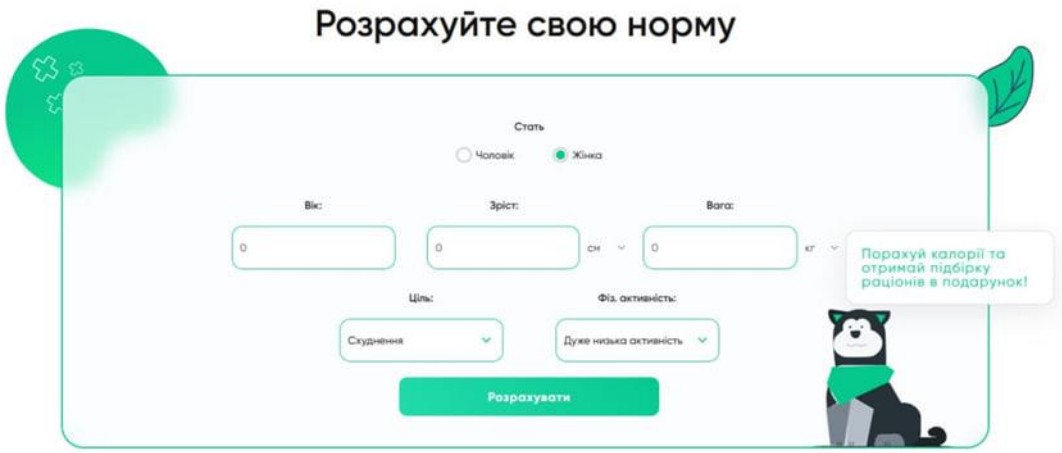


7. Profile

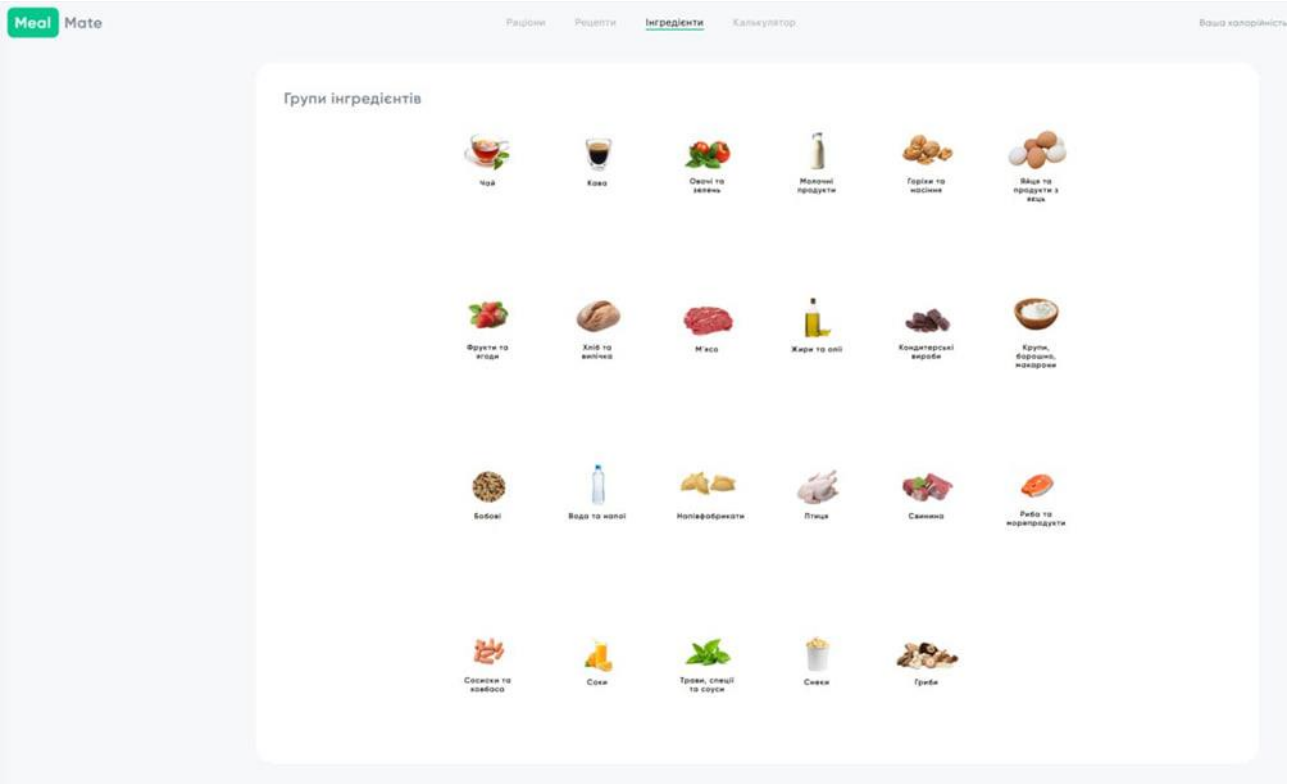


8. Calcualtor

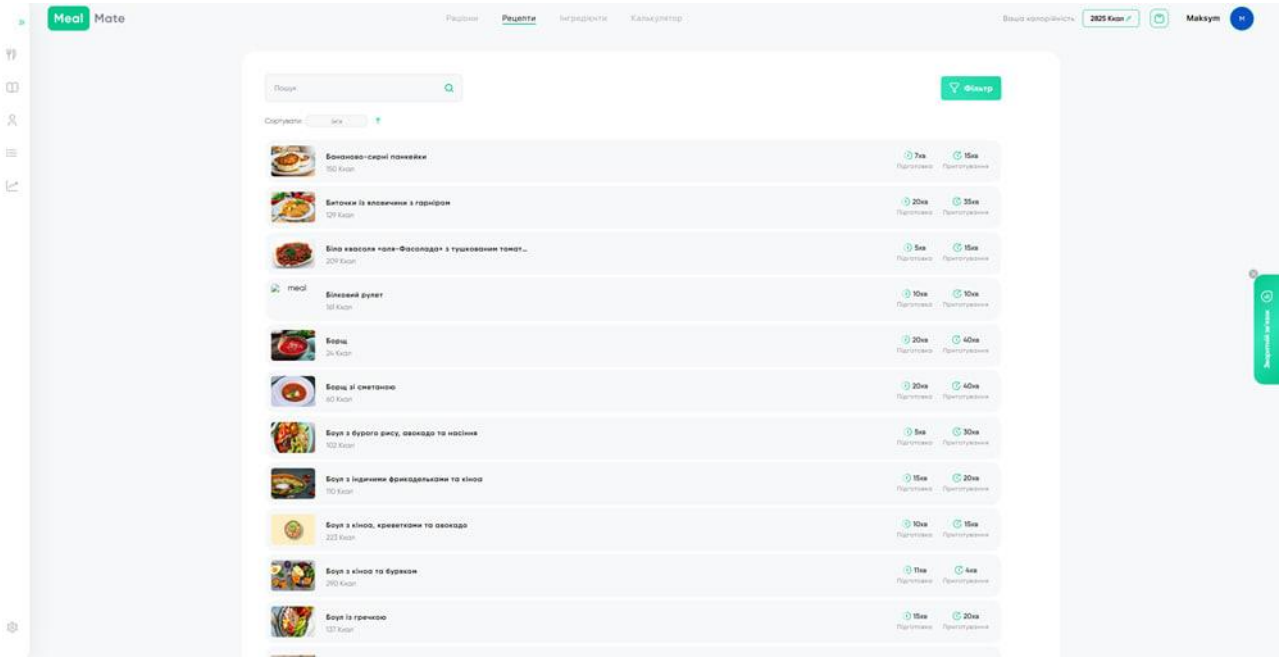




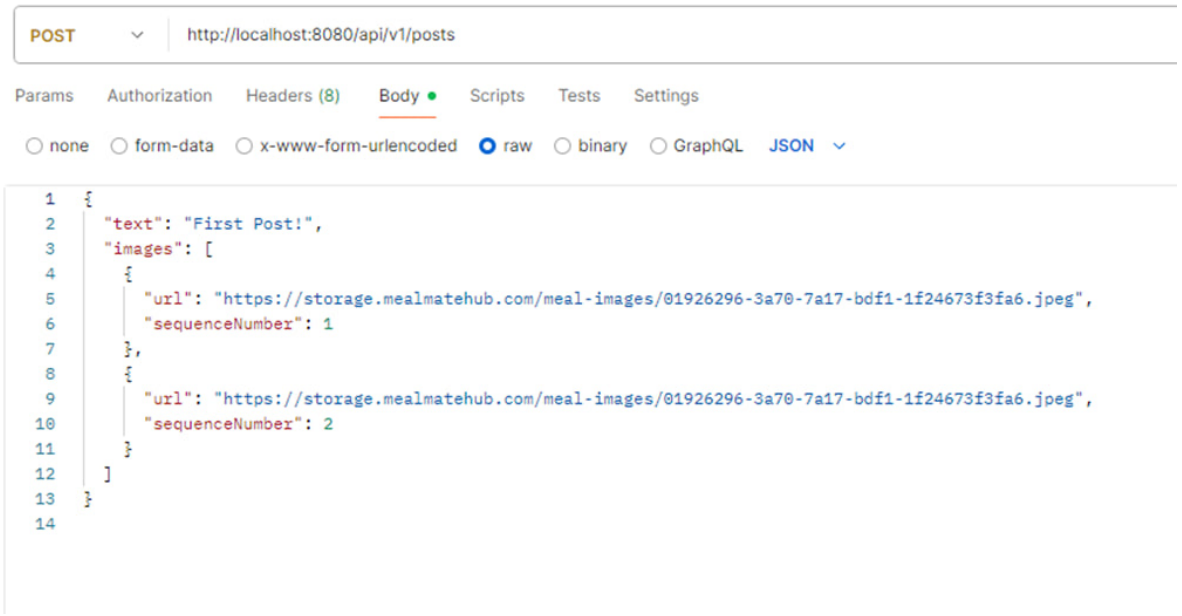
9. Ingridients

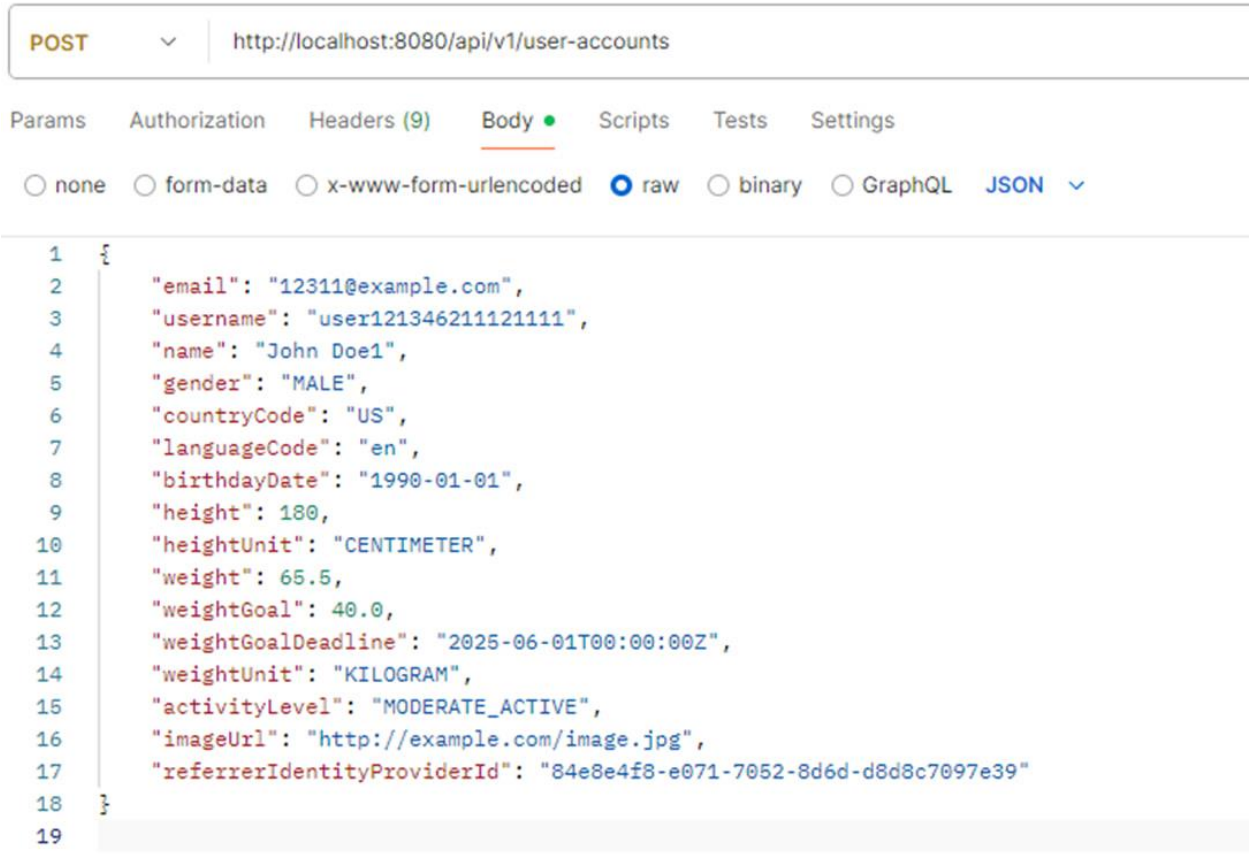


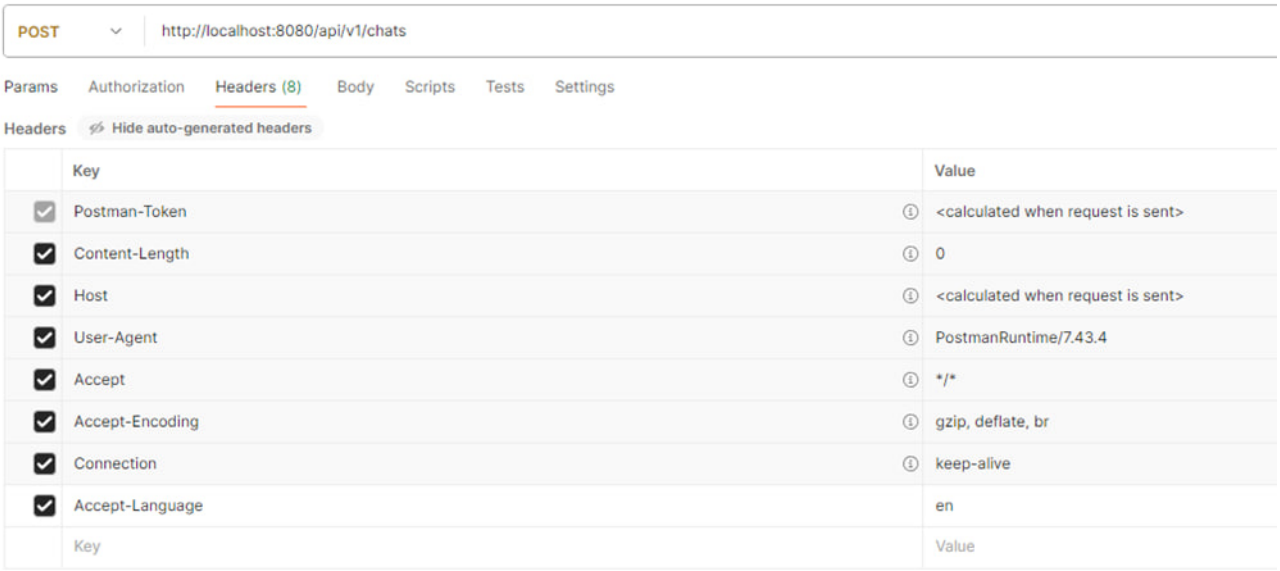
10. Meals



Додаток 3. Скріншоти API-запитів






POST
http://localhost:8080/api/v1/chats
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Headers
Hide auto-generated headers
Key
Value
Postman-Token
<calculated when request is sent>
Content-Length
0
Host
<calculated when request is sent>
User-Agent
PostmanRuntime/7.43.4
Accept
/
Accept-Encoding
gzip, deflate, br
Connection
keep-alive
Accept-Language
en
Key
Value

Додаток 4. Фрагменти коду

4.1 PostRestController

```
@RestController
@SecurityRequirement(name = "token")
@RequiredArgsConstructor
public class PostRestController {

    private final PostService postService;

    @PostMapping("/api/v1/posts")
    public PostDto create(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
            @RequestBody PostDto postDto) {
        return postService.create(performer.getIdentityProviderId(), postDto);
    }

    @PutMapping("/api/v1/posts/{postId}")
    public PostDto update(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
            @PathVariable UUID postId,
            @RequestBody PostDto postDto) {
        return postService.update(performer.getIdentityProviderId(), postId, postDto);
    }

    @DeleteMapping("/api/v1/posts/{postId}")
    public void delete(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
            @PathVariable UUID postId) {
        postService.delete(performer.getIdentityProviderId(), postId);
    }

    @GetMapping("/api/v1/posts/{postId}")
    public PostDto findById(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
            @PathVariable UUID postId) {
        String identityProviderId = performer != null ? performer.getIdentityProviderId() : null;
        return postService.findById(identityProviderId, postId);
    }

    @GetMapping("/api/v1/posts")
    public PageSearchResponseDto<PostDto> findAll(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
            PostSearchRequestDto postSearchRequestDto) {
        String identityProviderId = performer != null ? performer.getIdentityProviderId() : null;
        return postService.findAll(identityProviderId, postSearchRequestDto);
    }
}
```

4.2 UserAccountRestController

1.
```
@RestController
@SecurityRequirement(name = "token")
@RequiredArgsConstructor
public class UserAccountRestController {

    private final UserAccountService userAccountService;

    private final UserAccountLoginService userAccountLoginService;

    private final PartialUserAccountService partialUserAccountService;

    @PostMapping(value = "/api/v1/user-accounts/validate")
    public void validate(@RequestBody UserAccountDto userAccountDto) {
        userAccountService.validate(userAccountDto);
    }

    @PostMapping("/api/v1/user-accounts/partial")
    public PartialUserAccountResponseDto createPartial(@RequestBody
PartialUserAccountRequestDto request) {

        return partialUserAccountService.create(request);
    }

    @PostMapping("/api/v1/user-accounts/partial/confirm")
    public PartialUserAccountResponseDto createPartialConfirm(
            @RequestBody PartialUserAccountConfirmRequestDto request) {

        return partialUserAccountService.confirm(request);
    }

    @PutMapping("/api/v1/user-accounts/partial/continue")
    public ContinuePartialDto continuePartial(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer,
            @RequestBody ContinuePartialDto continuePartialDto) {
        return userAccountService.continuePartial(performer.getIdentityProviderId(),
continuePartialDto);
    }

    @PostMapping("/api/v1/user-accounts")
    public UserAccountDto createFull(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer,
            @RequestBody RegistrationUserAccountDto userAccountDto) {

        return userAccountService.createFull(performer.getIdentityProviderId(),
userAccountDto);
    }
```

```
    @PutMapping("/api/v1/user-accounts/current")
    public UserAccountDto updateProfile(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer,
            @RequestBody UpdateUserAccountDto updateUserAccountDto) {
        return userAccountService.updateProfile(performer.getIdentityProviderId(),
updateUserAccountDto);
    }

    @GetMapping(value = "/api/v1/user-accounts/{profileIdentityProviderId}/profile")
    public UserProfileDto findProfile(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer,
            @PathVariable String profileIdentityProviderId) {
        return userAccountService.findProfile(performer.getIdentityProviderId(),
profileIdentityProviderId);
    }

    @GetMapping(value = "/api/v1/user-accounts/current")
    public UserAccountDto findByAuthenticatedUser(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer) {

        return
userAccountService.findByIdentityProviderId(performer.getIdentityProviderId());
    }

    @GetMapping(value = "/api/v1/user-accounts/current/login-info")
    public UserAccountLoginInfoDto findLoginInfoByAuthenticatedUser(
            @CurrentSecurityContext(expression = "authentication") AuthenticatedUser
performer) {

        return
userAccountLoginService.findByIdentityProviderId(performer.getIdentityProviderId());
    }
}
```

4.3 PostLikeRestController

```
@RestController
@SecurityRequirement(name = "token")
@RequiredArgsConstructor
public class PostLikeRestController {

   private final PostLikeService postLikeService;

   @GetMapping("/api/v1/posts/{postId}/likes")
   public PageSearchResponseDto<PostLikeDto> findAllLikes(@PathVariable UUID postId) {
      return postLikeService.findAllLikesByPostId(postId);
   }

   @PostMapping("/api/v1/posts/{postId}/like")
   public void addLike(
         @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
         @PathVariable UUID postId) {
      postLikeService.addLike(performer.getIdentityProviderId(), postId);
   }

   @DeleteMapping("/api/v1/posts/{postId}/like")
   public void removeLike(
         @CurrentSecurityContext(expression = "authentication") AuthenticatedUser performer,
         @PathVariable UUID postId) {
      postLikeService.removeLike(performer.getIdentityProviderId(), postId);
   }
}
```

4.4 Entity

```
@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

@FieldNameConstants

@Entity

@Table(name = "post")

public class PostEntity {


   @Id

   @GeneratedValue(generator = "uuid7")
```

```
    @UUIDVersion7Generator(name = "uuid7")
    private UUID id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_account_id")
    private UserAccountEntity userAccount;

    private String text;

    private Long likesCount;

    private Instant createdAt;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy = "post", fetch = FetchType.LAZY)
    private List<PostImageEntity> images = new ArrayList<>();
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldNameConstants
@Entity
@Table(name = "post_like")
public class PostLikeEntity {

    @Id
    @GeneratedValue(generator = "uuid7")
```

```
    @UUIDVersion7Generator(name = "uuid7")
    private UUID id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id")
    private PostEntity post;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_account_id")
    private UserAccountEntity userAccount;

    private Instant createdAt;
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldNameConstants
@Entity
@Table(name = "post_image")
public class PostImageEntity {

    @Id
    @GeneratedValue(generator = "uuid7")
    @UUIDVersion7Generator(name = "uuid7")
    private UUID id;

    private String url;
```

```
    private Integer sequenceNumber;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id")
    private PostEntity post;
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "user_account")
@FieldNameConstants
public class UserAccountEntity {

    @Id
    @GeneratedValue(generator = "uuid7")
    @UUIDVersion7Generator(name = "uuid7")
    private UUID id;

    private String identityProviderId;

    private String username;

    private String email;

    private String name;

    @Enumerated(EnumType.STRING)
```

```
@JdbcType(PostgreSQLEnumJdbcType.class)
private Gender gender;

private String countryCode;

private String languageCode;

private LocalDate birthdayDate;

private Integer height;

@Enumerated(EnumType.STRING)
@JdbcType(PostgreSQLEnumJdbcType.class)
private HeightUnit heightUnit;

private BigDecimal weight;

@Enumerated(EnumType.STRING)
@JdbcType(PostgreSQLEnumJdbcType.class)
private WeightUnit weightUnit;

@Enumerated(EnumType.STRING)
@JdbcType(PostgreSQLEnumJdbcType.class)
private ActivityLevel activityLevel;

private String imageUrl;

private Instant createdAt;

private Boolean isDeleted;
```

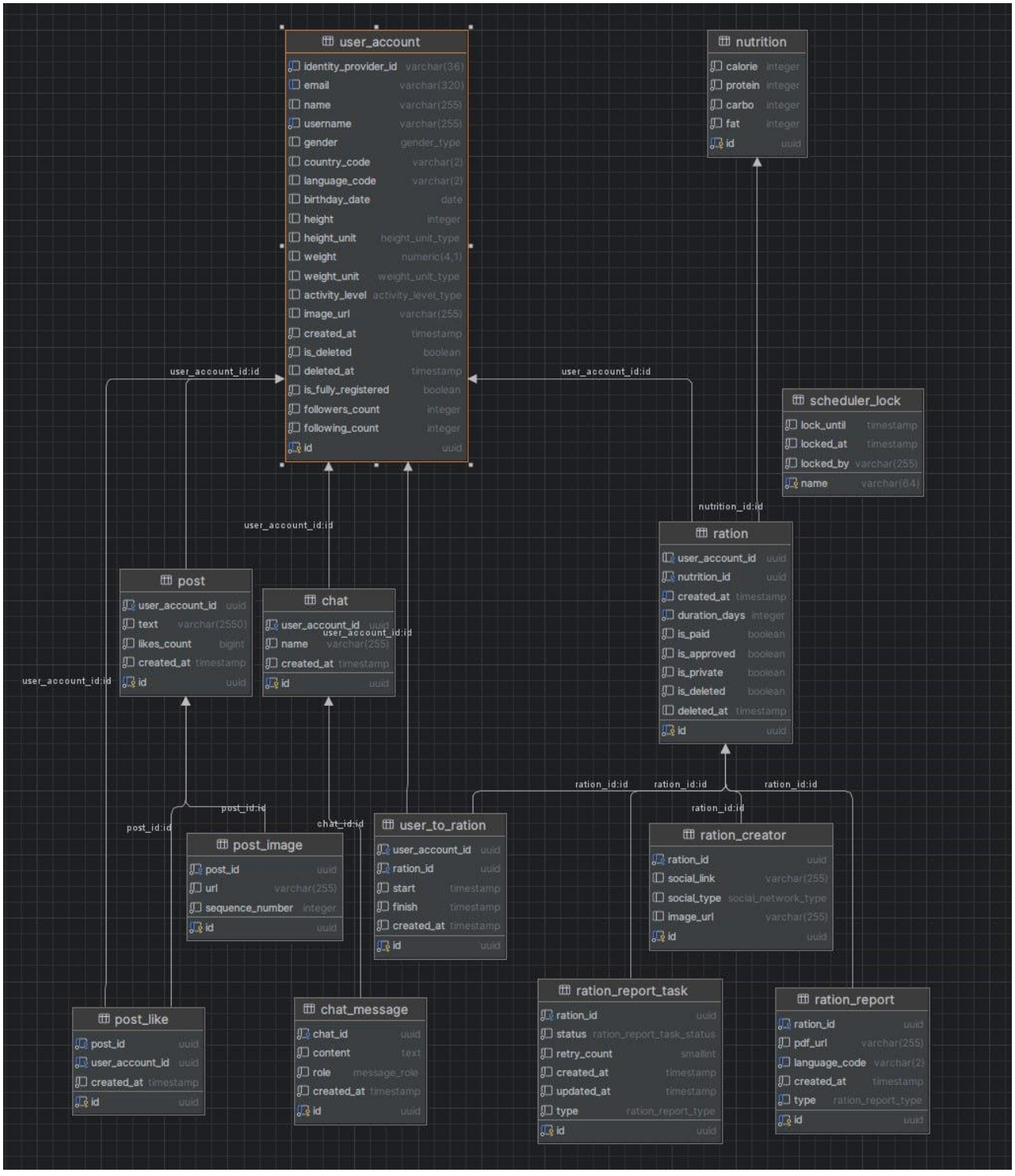
```
    private Boolean isFullyRegistered;

    private Instant deletedAt;

    private Integer followersCount;

    private Integer followingCount;
}
```

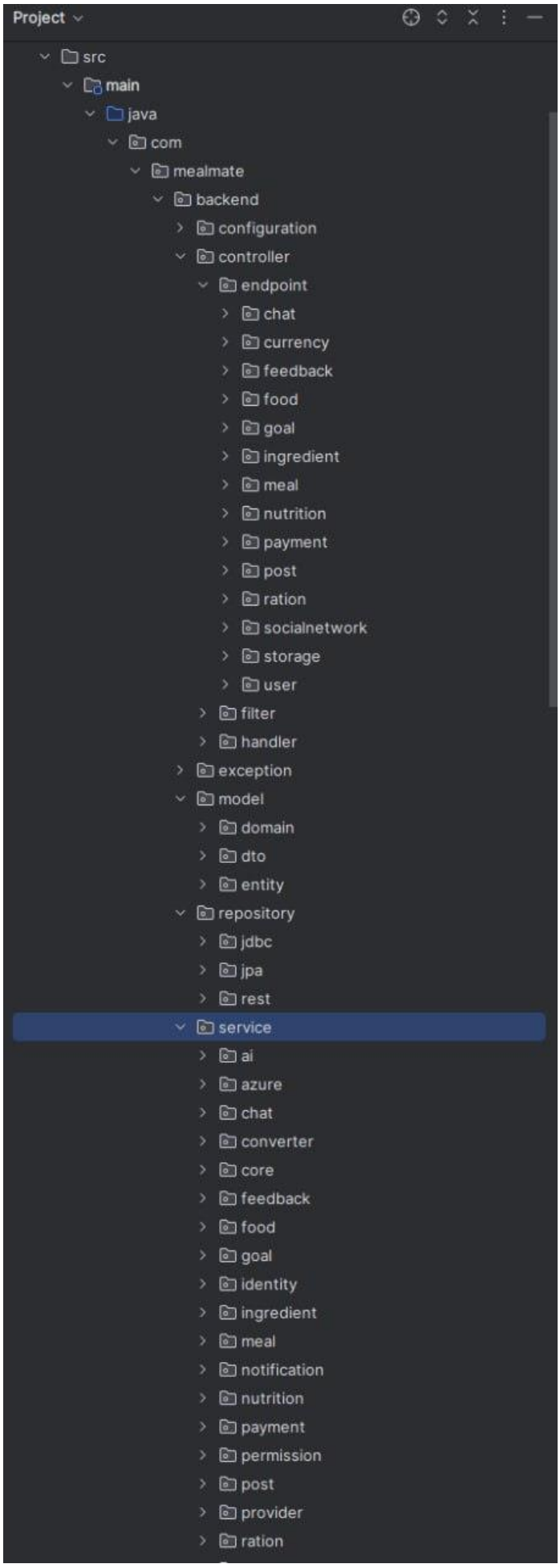
4.5 Dto

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldNameConstants
@JsonInclude(JsonInclude.Include.NON_NULL)
public class UserAccountDto {

    private UUID id;

    private String identityProviderId;

    private String email;

    private String username;

    private String name;

    private Gender gender;
```

```
    private String countryCode;

    private String languageCode;

    private LocalDate birthdayDate;

    private Integer height;

    private HeightUnit heightUnit;

    private BigDecimal weight;

    private WeightUnit weightUnit;

    private ActivityLevel activityLevel;

    private String imageUrl;

    private String referrerIdentityProviderId;
}
```

Додаток 5. Схеми і таблиці щодо візуалізації аналізу




Project
src
main
java
com
mealmate
backend
configuration
controller
endpoint
chat
currency
feedback
food
goal
ingredient
meal
nutrition
payment
post
ration
socialnetwork
storage
user
filter
handler
exception
model
domain
dto
entity
repository
jdbc
jpa
rest
service
ai
azure
chat
converter
core
feedback
food
goal
identity
ingredient
meal
notification
nutrition
payment
permission
post
provider
ration


post
like
PostLikeRestController
PostRestController