МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра Програмної інженерії

**КОМПЛЕКСНИЙ КУРСОВИЙ ПРОЄКТ**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський)

Тема роботи: Програмна система ведення харчового щоденника

Виконав:

здобувач 3 курсу, групи ПЗПІ-22-1

Ігор КОРЕЦЬКИЙ

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

Керівник доц. кафедри ПІ Ірина ПОБІЖЕНКО

Члени комісії:

доц. Віктор КАУК   
ст. викл. Олена ОЛІЙНИК   
ст. викл. Костянтин ОНИЩЕНКО

2025р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

Курс 3 Група ПЗПІ-22-1 Семестр 6

**ЗАВДАННЯ**

***на курсовий проект(роботу) студента***

здобувачеві Корецькому Ігору Олександровичу

1. Тема роботи Програмна система ведення харчового щоденника

2. Термін здачі студентом закінченої роботи „\_20\_”\_\_червня\_\_\_\_\_\_\_ 2025 р.\_

3. Вихідні дані до проєкту  *В програмній системі передбачити:*

*додавання нових користувачів, видів їжі, аналітику споживчого кошика,*

*адміністрування середовищем. Використовувати ОС Windows 11, СУБД*

*PostgreSQL, мови програмування C# (ASP.NET Core), бібліотеку React (JavaScript), мобільний застосунок через Capacitor (Android), середовища розробки Visual Studio, Visual Studio Code та Android Studio.*

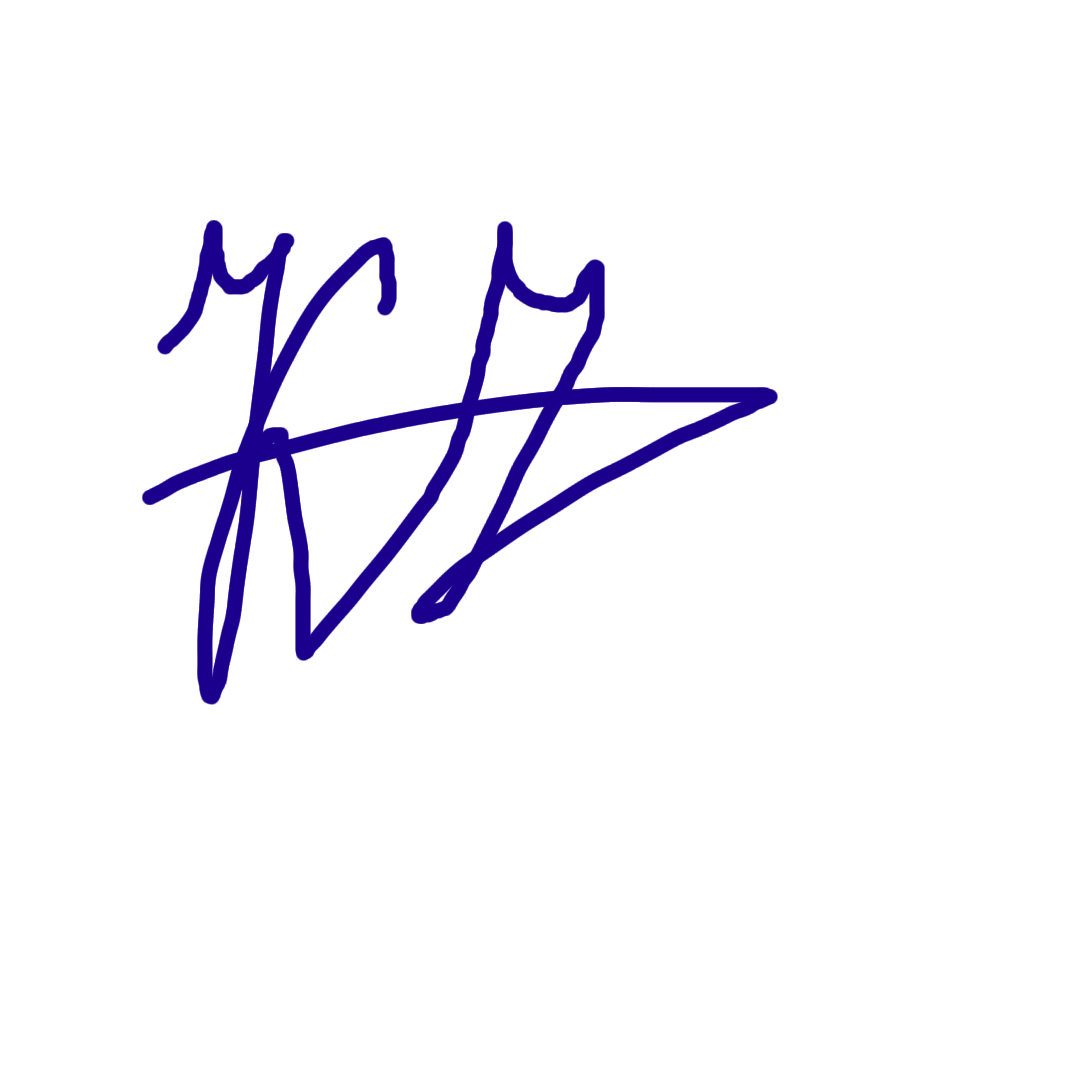
4. Перелік питань, що потрібно опрацювати в роботі *UML діаграма розгортання, UML діаграма прецедентів, ER-модель даних, UML*

*діаграма компонентів, UML діаграма пакетів, UML діаграма взаємодії,*

*UML діаграма діяльності, UML діаграма станів.*

КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| --- | --- | --- | --- |
| 1 | Аналіз предметної галузі | 20.05.2025 | Виконано |
| 2 | Розробка постановки задачі | 20.05.2025 | Виконано |
| 3 | Проектування ПЗ | 21.05.2025 | Виконано |
| 4 | Програмна реалізація | 21.05.2025 - 17.06.2025 | Виконано |
| 5 | Аналіз результатів | 18.06.2025 | Виконано |
| 6 | Підготовка пояснювальної записки | 18.06.2025 - 20.06.2025 | Виконано |
| 7 | Перевірка на академічну доброчесність | 20.06.2025 | Виконано |
| 8 | Захист роботи | 20.06.2025 | Виконано |

Дата видачі завдання «\_01\_» \_\_\_березня\_\_\_ 2025р.

Здобувач

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. кафедри ПІ Побіженко І.О

# РЕФЕРАТ

Пояснювальна записка містить: 73 с., 13 рис., 5 табл., 11 джерел.

СПОЖИВАННЯ ЇЖІ, АНАЛІТИКА, СЕРВЕРНА ЧАСТИНА, ВЕБ-ІНТЕРФЕЙС, ASP.NET CORE, POSTGRESQL, МОБІЛЬНИЙ ДОДАТОК, CAPACITOR.

Об'єктом розробки є програмна система для моніторингу споживання їжі користувачем.

Метою розробки є створення ефективної багатокомпонентної програмної системи, яка дозволяє користувачеві вести облік споживаних продуктів, отримувати рекомендації щодо покращення раціону, аналізувати харчові звички за допомогою аналітичних візуалізацій.

Методи розробки включають: використання ASP.NET Core (C#) для серверної частини, PostgreSQL як системи керування базами даних, Capacitor для створення мобільного застосунку на основі веб-версії, React (JavaScript) для клієнтського інтерфейсу. Розробка здійснювалась у середовищах Visual Studio, Visual Studio Code та Android Studio.

У результаті створено програмну систему, яка надає користувачам можливість контролювати власне харчування, переглядати статистику, отримувати персоналізовані поради та формувати здорові харчові звички.

# 

# ABSTRACT

The explanatory note contains: 73 pages, 13 figures, 5 tables, 11 sources.

FOOD CONSUMPTION, ANALYTICS, BACKEND, WEB INTERFACE, ASP.NET CORE, POSTGRESQL, MOBILE APPLICATION, CAPACITOR.

The object of development is a software system for monitoring users' food consumption.

The aim of this work is to create an efficient multi-component software system that allows users to track their food intake, receive recommendations to improve their diet, and analyze eating habits through visualization tools.

The development methods include the use of ASP.NET Core (C#) for the backend, PostgreSQL as the database management system, Capacitor for building a mobile application based on the web version, and React (JavaScript) for the client interface. Development was carried out using Visual Studio, Visual Studio Code, and Android Studio.

As a result, a software system was created that enables users to manage their nutrition, view statistics, receive personalized recommendations, and build healthy eating habits.

# ЗМІСТ

[**РЕФЕРАТ 3**](#_2ms5ajod3i40)

[**ABSTRACT 4**](#_aigy9kpitpew)

[**ЗМІСТ 5**](#_i9osmbu08ud4)

[**ВСТУП 7**](#_lep9bwfca74i)

[**1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 8**](#_4g7o614g7nki)

[1.1 Аналіз предметної галузі 9](#_b7s1ksjzff3l)

[1.2 Виявлення та вирішення проблем 10](#_bbly01nx788t)

[1.3 Аналіз аналогів програмного забезпечення 13](#_1bzibtehq94u)

[**2 ПОСТАНОВКА ЗАДАЧІ 19**](#_d4vf4scujspg)

[2.1 Загальна постановка задачі 19](#_zqtcypsc6c3)

[2.2 Основний функціонал системи 20](#_wwz2mqm2y804)

[2.3 Очікувані результати 20](#_ujvr69c49g9n)

[**3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 22**](#_g3ntxuszbwd2)

[3.1 UML-проєктування програмного забезпечення 22](#_7cm1agbl19mt)

[3.2 Проєктування архітектури програмного забезпечення 24](#_ono2hekt22fa)

[3.3 Проєктування структури зберігання даних 26](#_eqxqw8kpuzsx)

[3.4 Приклади використаних алгоритмів та методів 28](#_mc8prr5ae524)

[3.5 Проєктування інтерфейсу та зручності користування 32](#_epct0ifzk86e)

[3.6 Підготовка даних 35](#_5pgdktc6ky8b)

[**4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ 37**](#_8euwul38a0p)

[4.1 Серверна частина: запуск, безпека, проміжний конвеєр 37](#_1bjn6n65pdun)

[4.2 Модель предметної області та конфігурація БД 37](#_rv2zo3qho1rs)

[4.3 Доступ до даних: «Сховище» та специфікації 38](#_gel8bl65wv4f)

[4.4 Командно-запитна модель (CQRS) через MediatR 39](#_bf390718gl2f)

[4.5 Веб-API: кінцеві точки на базі FastEndpoints 40](#_96y5br1ke6v5)

[4.6 Клієнтська частина: маршрути, захист, клієнт API, стан, форми 41](#_3ccwbj20ot8l)

[4.7 Алгоритмічні рішення на практиці 46](#_zfcj5nkhvmjq)

[**5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ 47**](#_qphbajp3hx9u)

[5.1 Порівняння результатів із початковими вимогами 47](#_7gvjz3z2oub0)

[5.2 Оцінка якості роботи системи 50](#_1f9c3poif01t)

[5.3 Аналіз ефективності прийнятих рішень 51](#_1sv7m34nyrz6)

[5.4 Обмеження та недоліки 52](#_f6bsx4s5kyp5)

[5.5 Напрями удосконалення 52](#_avwqpfcmieki)

[5.6 Узагальнення 53](#_cmlxxmm69cvq)

[**6 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 54**](#_v77qiv8y249q)

[**7 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 56**](#_ji7om31hpekp)

[**ВИСНОВКИ 56**](#_pkkm1b4e1roe)

[**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 58**](#_baf9wrb0k5u5)

[**ДОДАТОК А - СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SRS) 59**](#_t982snph0e1z)

[**ДОДАТОК Б - ПРЕЗЕНТАЦІЯ 66**](#_byehga8xkh10) **ДОДАТОК В - ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ 72**

# 

# ВСТУП

У сучасному ритмі життя контроль за харчуванням стає дедалі складнішим завданням. Люди стикаються з дефіцитом часу, недостатньою обізнаністю щодо складу продуктів і браком зручних засобів для фіксації раціону. Як наслідок - порушення харчового балансу, що сприяє розвитку хронічних захворювань, зниженню рівня енергії та загального погіршення самопочуття.

Одна з ключових причин - складність самостійного аналізу спожитої їжі: її калорійності, вмісту макро- й мікроелементів, а також відсутність адаптованих під користувача підказок щодо оптимізації харчування. Тому виникає потреба у цифровому інструменті, який дозволить не тільки вести облік спожитого, а й надавати зрозумілу аналітику, персональні рекомендації та бути зручним у щоденному користуванні. Застосування сучасних веб- і мобільних технологій дає змогу створити подібну систему, яка буде доступна з різних пристроїв і враховуватиме індивідуальні особливості користувача.

Метою цієї курсової роботи є розробка багатокомпонентної програмної системи, яка допоможе користувачам вести харчовий щоденник, аналізувати харчові звички та формувати здорові підходи до раціону. Передбачається створення серверної частини на основі ASP.NET Core, розробка Android-додатку за допомогою Kotlin, веб-інтерфейсу з використанням React, а також зберігання й обробка даних у СУБД PostgreSQL. Особлива увага приділяється функціоналу аналітики та персоналізованих рекомендацій.

Очікується, що результати розробки можуть бути використані як індивідуальними користувачами, так і фітнес-центрами, медичними та освітніми установами. Така система дозволяє підвищити рівень обізнаності щодо харчування, покращити самопочуття та підтримувати здоровий спосіб життя за допомогою технологій.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Активний розвиток цифрових технологій та зростаюча увага суспільства до питань здоров’я спричинили стрімке зростання попиту на програмні рішення, які дозволяють користувачам контролювати свій раціон та харчові звички. З кожним роком все більше людей прагнуть вести здоровий спосіб життя, слідкувати за калорійністю їжі, балансом поживних речовин і загальним впливом споживаних продуктів на організм. У зв’язку з цим на ринку інформаційних технологій з’являється дедалі більше мобільних та веб-рішень, спрямованих на покращення якості харчування.

Такі системи можуть бути корисними для найрізноманітніших груп користувачів: спортсменів, людей, які контролюють свою вагу або дотримуються дієт, а також тих, хто просто прагне підтримувати збалансоване харчування. Крім того, подібні сервіси можуть зацікавити медичні установи, дієтологічні центри та фітнес-клуби, де контроль за раціоном є частиною комплексної роботи з клієнтами.

Програмна система, яка дозволяє зручно вести харчовий щоденник, отримувати персоналізовані рекомендації, переглядати статистику й аналітику, має потенціал стати цінним інструментом не лише для кінцевих користувачів, а й для бізнесу. Наприклад, заклади громадського харчування (кафе, ресторани) можуть інтегрувати подібні рішення у свої сервіси, пропонуючи відвідувачам інформацію про склад страв, калорійність та корисність, підвищуючи тим самим рівень довіри та лояльності клієнтів.

Згідно з аналітичними дослідженнями, ринок програм для харчового моніторингу та фітнесу демонструє стабільне зростання. За прогнозами, до 2026 року його обсяг може перевищити 27 мільярдів доларів США, що свідчить про високий інтерес з боку користувачів і перспективність подібних розробок.

Таким чином, створення програмної системи для моніторингу харчування - це не лише актуальна технологічна задача, але й відповідь на реальні запити сучасного суспільства. У цьому розділі буде детально розглянуто предметну область, виявлено наявні проблеми, проаналізовано існуючі рішення та сформульовано завдання для власної розробки.

## 1.1 Аналіз предметної галузі

Галузь цифрових рішень для контролю харчування поєднує в собі елементи медицини, дієтології, спортивної аналітики та інформаційних технологій. В умовах зростання популярності здорового способу життя, усвідомленого підходу до раціону та активної турботи про фізичний стан, програмні системи для моніторингу харчування займають важливе місце на ринку мобільних застосунків та веб-сервісів.

На практиці ці системи охоплюють функціональність ведення харчового щоденника, підрахунку калорій та нутрієнтів, аналізу харчових звичок, формування персоналізованих дієт, візуалізації статистики споживання, а також можуть бути інтегровані з носимими пристроями та трекерами фізичної активності. Такий підхід дозволяє не тільки фіксувати фактичне споживання їжі, але й аналізувати його вплив на стан організму.

Використання подібних систем не обмежується лише бажанням контролювати вагу. Їх активно застосовують для підтримки здоров’я при хронічних захворюваннях, алергіях, індивідуальних дієтах або у процесі медичної реабілітації. У фітнес-індустрії такі рішення стали невіддільним елементом загальної програми тренувань, оскільки правильне харчування напряму впливає на ефективність фізичних навантажень.

Зростаюча обізнаність користувачів щодо ролі макро- та мікронутрієнтів у підтримці здоров’я формує високі очікування до функціональності подібного програмного забезпечення. Сучасний користувач прагне не лише реєструвати дані, але й отримувати розумні аналітичні висновки, рекомендації, персональні поради та мотиваційну підтримку.

Галузь активно рухається у напрямку персоналізації, автоматизації та використання аналітики. Системи дедалі частіше застосовують алгоритми машинного навчання для формування персональних рекомендацій, використовують розширені бази даних продуктів і враховують індивідуальні особливості користувача, такі як стан здоров’я, спосіб життя або навіть емоційний стан.

Таким чином, предметна область контролю харчування є складною, міждисциплінарною та перспективною. Вона потребує поєднання знань у сфері дієтології, програмної інженерії, аналізу даних та UX-дизайну, що відкриває широкі можливості для створення інноваційних та корисних продуктів для різних категорій користувачів.

## 1.2 Виявлення та вирішення проблем

Розробка програмної системи для контролю харчування користувачів вимагає врахування реальних потреб аудиторії, актуальних викликів ринку та потенційних ризиків, які можуть впливати на ефективність впровадження продукту.

Головна мета такого рішення - створення єдиного інтерактивного середовища для відстеження харчових звичок, аналізу раціону та покращення здоров’я за допомогою цифрових інструментів. Для цього потрібно чітко розуміти, які саме очікування мають користувачі з різних цільових груп. Їх основні потреби та очікування узагальнено в таблиці 1.1.

Таблиця 1.1 – Групи користувачів і їх очікування

| **Група користувачів** | **Основна цінність** | **Ключові очікування** |
| --- | --- | --- |
| Загальні користувачі | Покращення здоров’я, контроль харчування | Зручність, точність, персоналізація |
| Ресторани й кафе | Покращення сервісу, аналітика замовлень | Інтеграція з меню, статистика попиту |
| Дослідники та науковці | Масиви харчових даних для аналізу | Анонімність, точність, репрезентативність |
| Організації охорони здоров’я | Вивчення харчових звичок населення | Узагальнена аналітика, динаміка змін |

Успіх проєкту безпосередньо залежить від здатності задовольнити вказані потреби. Для досягнення цілей системи необхідно визначити чіткі критерії ефективності. Ключові з них представлено у таблиці 1.2.

Таблиця 1.2 – Критерії успіху програмної системи

| **Критерій** | **Опис** |
| --- | --- |
| Активність користувачів | Зростання зареєстрованих і регулярно активних обліковок |
| Задоволеність користувачів | Позитивні відгуки, низький рівень відмов |
| Прибутковість | Продаж преміум-функцій, партнерські програми |
| Масштабованість | Можливість обробки великих обсягів даних |

Під час реалізації програмної системи необхідно враховувати потенційні ризики, які можуть вплинути на її ефективність, безпеку та прибутковість. Основні з них, разом із можливими шляхами їх мінімізації, наведено в таблиці 1.3.

Таблиця 1.3 – Основні ризики та рішення

| **Ризик** | **Значимість** | **Рішення** |
| --- | --- | --- |
| Витік персональних даних | Висока | Шифрування, автентифікація, аудит безпеки |
| Невідповідність вимогам законодавства | Висока | Дотримання GDPR, створення політик конфіденційності |
| Мала кількість користувачів | Середня | Маркетинг, партнерства, орієнтація на потреби цільової аудиторії |
| Застарілі технології | Низька | Використання сучасного стеку (ASP.NET Core, Kotlin, PostgreSQL тощо) |
| Залежність від зовнішніх сервісів | Низька | Мінімізація зовнішніх API, можливість автономної роботи |
| Низька прибутковість | Середня | Модель підписок, співпраця з бізнесом, рекламна інтеграція |

Щоб система була конкурентоспроможною, необхідно забезпечити її відповідність сучасним вимогам до продуктивності. Програма має миттєво реагувати на дії користувача, ефективно працювати з великими обсягами даних, забезпечувати стабільність у роботі та зберігати працездатність навіть під час резервного копіювання. Це дозволяє створити надійний цифровий продукт, що відповідає очікуванням користувачів і стандартам сучасного ринку.

## 1.3 Аналіз аналогів програмного забезпечення

Розробка програмного забезпечення для моніторингу харчування потребує врахування вже існуючих рішень на ринку. Аналіз таких аналогів дозволяє глибше зрозуміти очікування користувачів, визначити найкращі практики у сфері дизайну інтерфейсу, організації функціоналу, логіки персоналізації та аналітики, а також ідентифікувати наявні проблеми, які залишаються невирішеними в комерційних продуктах. Такий підхід дозволяє сформувати конкурентну концепцію власної системи.

Одним із найбільш відомих та функціонально наповнених продуктів на ринку є MyFitnessPal - мобільний застосунок та веб-платформа, що з'явилися у 2005 році та згодом були придбані компанією Under Armour. Додаток орієнтований на широку аудиторію користувачів, зацікавлених у веденні харчового щоденника, підрахунку калорій, контролі ваги та досягненні цілей, пов'язаних зі здоровим способом життя. Застосунок доступний як для Android та iOS, так і у браузерній версії, а його база користувачів нараховує понад 200 мільйонів осіб по всьому світу.

MyFitnessPal забезпечує базовий функціонал, необхідний для ефективного контролю харчування: користувач має змогу додавати прийоми їжі, фіксувати кількість спожитих калорій і макроелементів, вести записи щодо вживаної води, змін ваги та рівня фізичної активності. Інформація вводиться вручну або через зчитування штрихкодів, що значно пришвидшує процес. Однією з головних переваг застосунку є обширна база харчових продуктів, яка налічує понад 11 мільйонів позицій, зокрема продукти з різних країн і брендів. Проте її відкрита модель - коли користувачі самостійно додають нові записи - часто призводить до дублювання елементів або появи помилок у вказаних харчових значеннях.

Процес персоналізації реалізовано через опитування під час реєстрації: користувач вказує вік, вагу, зріст, стать, рівень активності та бажану ціль (наприклад, зниження ваги), після чого система автоматично формує щоденну норму калорій та співвідношення білків, жирів і вуглеводів. За потреби ці параметри можна коригувати вручну. У межах безкоштовної версії користувачі мають доступ до базових звітів про харчування та активність, тоді як розширені аналітичні функції, у тому числі аналіз мікронутрієнтів, історія змін за довгі періоди та можливість експорту даних, доступні лише за передплатою у версії Premium.

MyFitnessPal також інтегрується з великою кількістю фітнес-пристроїв, зокрема з Google Fit, Apple Health, Fitbit та Garmin, що дозволяє поєднувати дані про фізичну активність із даними про споживання їжі в межах одного інтерфейсу. Інтерфейс додатку загалом функціональний, проте деякі його елементи можуть здаватися перевантаженими або неочевидними для новачків. Значну критику отримує й наявність нав'язливої реклами в безкоштовній версії, що помітно ускладнює користування системою. Оцінки додатку на популярних платформах свідчать про доволі високий рівень задоволеності - середній рейтинг у Google Play становить 4.1 на основі понад двох мільйонів відгуків, хоча часто згадуються проблеми з надмірною монетизацією, дублюванням продуктів та складністю у налаштуваннях.

У порівнянні з MyFitnessPal, розроблювана нами система робить акцент на точності й верифікації бази продуктів, простоті інтерфейсу та локалізованому підході до української аудиторії. Крім того, базова версія включатиме більш розширену аналітику без потреби оформлення підписки, що робить систему доступною для широкого кола користувачів. Важливою особливістю також стане можливість застосування розробки не лише в особистих цілях, а й в медичних закладах, дослідницьких інститутах та у сфері корпоративного здоров’я, завдяки інтеграції з відкритим API.

Поглиблений аналіз MyFitnessPal дає змогу зрозуміти сильні сторони комерційного продукту - зокрема його популярність, функціональність і багатство бази - та водночас виявити ключові слабкості, які можна врахувати при власній розробці. У той час як MyFitnessPal є прикладом класичного харчового трекера з широким охопленням функцій, наступним прикладом стане Samsung Health - рішення, орієнтоване на комплексний моніторинг здоров’я, що включає в себе аналіз фізичної активності, сну, стресу й харчування.

Наступним конкурентом для аналізу є Samsung Health - безкоштовний мультиплатформовий додаток від Samsung Electronics, доступний на Android, iOS та Tizen-пристроях. Вперше представлений у липні 2012 року під назвою S Health, він згодом став ключовим елементом екосистеми Galaxy, яка була доповнена широким набором функцій для моніторингу здоров’я, тренувань, сну, стресу та харчування.

За допомогою Samsung Health користувач може відстежувати фізичну активність, включно з кроками, сходженням поверхів і різними видами тренувань (біг, плавання, вело), а також автоматично отримувати інформацію про витрачені калорії, базальну метаболічну норму, рівень стресу, змін у серцевому ритмі, сні та складі тіла (процент жирової маси, м’язової активності) . Інтерфейс включає зручну дашборд-секцію з персоналізованими картками та порадами. Наприклад, функція «Energy Score» після аналізу сну та активності надає рекомендації щодо оптимального навантаження на наступний день .

Хоча модуль харчового щоденника не є основним фокусом, він дозволяє додавати прийоми їжі вручну і відстежувати основні показники - такі як калорії та об’єм споживаної води. Інтеграція з пристроями забезпечує автоматичне перенесення даних про кроки і калорії з Galaxy Watch або Galaxy Ring . Водночас користувачі в мережі іноді зазначають, що сучасна версія додатку має лише базовий рівень нутрієнтної аналітики - вона не відображає розподіл білків, жирів і вуглеводів, що обмежує її застосування для детального харчового аналізу .

Однією з головних переваг є глибока інтеграція із сімейством пристроїв Galaxy, що забезпечує автоматичне фіксування сну, тренувань, серцевих показників, стресу та харчування в одному інтерфейсі. Так, користувачі можуть отримувати доступ до моніторингу ЕКГ, пульсу, сну з розбиттям за фазами, рівня кисню в крові та навіть ймовірності апное . Крім того, у 2024–25 роках запуск One UI 6 Watch додав AI-поради щодо здоров’я, нові функції для керування тренуваннями, визначення апное, покращеної підтримки BI-операцій, таких як «Race» для порівняння результатів.

Незважаючи на широкий функціонал, Samsung Health має певні обмеження. По‑перше, базовий харчовий менеджер не надає повної нутрієнтної аналітики та не включає сканування штрихкодів, що користувачі вважають звичною практикою у їм подібних додатках. По‑друге, частини функціоналу є доступними лише через купівлю чи використання суміжних пристроїв - наприклад, функцій Sleep Coaching або апное - що обмежує можливості користувачів без Galaxy Watch. Також, можлива плата за передплату деяких функцій була анонсована у зв’язку з оновленням One UI Watch, що може обмежити доступ до нових можливостей у безкоштовному режимі.

Таким чином, Samsung Health є комплексним рішенням для цілісного моніторингу здоров’я, що вигідно поєднує фізичну активність, сон і певний рівень харчових даних. Проте, через обмежену глибину харчового аналізу, залежність від пристроїв екосистеми Samsung та потенційні обмеження безкоштовної версії, система потребує доповнення для користувачів, які шукають точну та детальну інформацію про харчування.

На цьому прикладі стає зрозуміло, що наступним логічним для порівняння аналогом має стати застосунок Cronometer - орієнтований саме на детальний нутрієнтний облік і може служити протилежністю до Samsung Health у межах наступної частини аналізу.

Cronometer було запущено у 2010 році і пізніше пережило ребрендинг у 2015–2016 роках. Сьогодні він доступний на Android, iOS та у веб-версії, при цьому добре відомий серед дієтологів, спортивних тренерів і користувачів із напрацьованою культурою підрахунку нутрієнтів. Базова функція додатку - це ведення харчового щоденника, де учасник вводить продукти вручну, за допомогою штрих-коду або з бази, і отримує надзвичайно детальні дані як по калоріях і макронутрієнтах, так і по більше ніж 20 мікронутрієнтах.

Cronometer використовує велику та верифіковану базу даних з продуктами, що включає офіційні джерела - національні бази харчових елементів, дані USDA та партійно перевірені записи. Це забезпечує високий рівень точності, який рідко зустрічається в іншому програмному забезпеченні. З іншого боку, така точність ускладнює інтерфейс, і для нових користувачів процес введення даних може здатися громіздким.

Система також пропонує високий рівень персоналізації: користувач може створювати індивідуальні цілі по макро- та мікронутрієнтах, налаштовувати денні ліміти, задавати фізичні активності, вносити показники тіла. Генерація звітів охоплює понад десяток графіків та діаграм – зміна ваги та загального енергобалансу, розподіл нутрієнтів, порівняння звітів, коливання мікроелементів.

У безкоштовній версії крім базових функцій доступні графіки калорій та основних макронутрієнтів, а також експорт CSV. Хоча глибші дані – наприклад, по мікронутрієнтах чи «блоки харчової вартості» – доступні лише у преміум-версії. Тим не менш, для користувачів, які прагнуть максимальної деталізації, навіть базового набору інструментів вистачає для глибокого розуміння власного раціону.

Cronometer підтримує синхронізацію з популярними фітнес-трекерами (Garmin, Fitbit), а також має додаток для Apple Watch. Інтеграція дозволяє поєднувати дані активності, щоденного харчування та вагу в одному вигляді.

Інтерфейс Cronometer сфокусований на максимально ефективному введенні даних, і хоча він може здаватися надто технічним, саме ця особливість приваблює користувачів, які прагнуть високої точності. У відгуках на платформі Google Play та в App Store найчастіше згадується якісний підрахунок нутрієнтів, зокрема мікроелементів, проте також зазначається, що процес введення іноді займає багато часу та вимагає значних зусиль, особливо для новачків.

Однією з основних переваг Cronometer є його надзвичайна точність у підрахунку харчових показників, зокрема завдяки великій базі перевірених продуктів і можливості детального звітування. Цей рівень деталізації дозволяє користувачам отримати глибоке розуміння свого раціону, включно з мікронутрієнтами, які у більшості аналогів часто не враховуються. Водночас не можна оминути низку обмежень: застосунок може здаватися складним у використанні через громіздкий процес введення їжі, наявність частини аналітичних функцій лише у преміум-версії, а також не завжди інтуїтивно зрозумілий інтерфейс, що може ускладнювати перші кроки для нових користувачів.

Таким чином, глибокий аналіз Cronometer завершує порівняння сучасних рішень на ринку, демонструючи високий рівень нутрієнтної точності, але водночас і певну технічну складність у використанні. Враховуючи переваги MyFitnessPal, комплексність підходу Samsung Health і деталізацію Cronometer, у межах нашої розробки передбачається створення гібридної системи, яка поєднує точність, інтуїтивність та гнучкість. Це дозволить уникнути недоліків існуючих рішень та забезпечити комфортну роботу як пересічним користувачам, так і фахівцям зі сфери харчування.

# 2 ПОСТАНОВКА ЗАДАЧІ

Програмна система FoodDiary призначена для полегшення процесу контролю за харчуванням користувачів, формування здорових харчових звичок та підвищення рівня обізнаності щодо складу продуктів. Основною метою є створення інтегрованого рішення, яке поєднує веб-інтерфейс та мобільний додаток, забезпечує зручне введення даних, надання аналітики та рекомендацій, а також підтримує належний рівень безпеки персональної інформації.

## 2.1 Загальна постановка задачі

У межах курсової роботи передбачається реалізація багатокомпонентної системи, яка має виконати такі основні завдання:

* забезпечити користувача можливістю вести харчовий щоденник, включаючи додавання, редагування та видалення записів про продукти і страви;
* надати інструменти для роботи з базою продуктів, включно з можливістю створювати та організовувати власні продукти й категорії;
* реалізувати модуль рецептів, що дозволяє створювати рецепти зі списком інгредієнтів, описом та фото, а також додавати їх у щоденник із автоматичним розрахунком калорійності;
* розробити систему аналітики, яка відображає статистику споживання їжі за день, тиждень та більш тривалий період;
* впровадити механізм персональних рекомендацій, що допомагають оптимізувати раціон відповідно до індивідуальних цілей користувача;
* створити систему сповіщень і нагадувань (наприклад, про воду чи прийом їжі);
* передбачити можливість налаштування профілю користувача з особистими даними та цілями (вага, активність, індивідуальні потреби);
* реалізувати захист даних через автентифікацію, шифрування та використання сучасних протоколів безпеки.

## 2.2 Основний функціонал системи

Для досягнення поставлених цілей система повинна включати наступні компоненти:

1. Серверна частина (ASP.NET Core, PostgreSQL):

* обробка запитів клієнтів через REST API;
* управління базою даних продуктів, рецептів та користувачів;
* формування аналітичних звітів;
* реалізація механізмів безпеки та управління доступом.

1. Клієнтська частина (React):

* інтерфейс для роботи з харчовим щоденником;
* перегляд статистики у вигляді графіків та таблиць;
* управління рецептами та продуктами;
* персоналізація профілю.

1. Мобільний додаток (Capacitor):

* адаптивний інтерфейс для швидкого введення даних;
* push-сповіщення для нагадувань;
* синхронізація даних з сервером;
* відображення аналітики у компактному вигляді.

## 2.3 Очікувані результати

Розробка системи FoodDiary повинна забезпечити:

* зручний і швидкий спосіб введення даних про харчування;
* формування статистики та візуалізацію даних для аналізу харчових звичок;
* отримання користувачем персоналізованих порад;
* підвищення мотивації користувачів до ведення здорового способу життя;
* створення гнучкої архітектури, що дозволяє у майбутньому розширювати функціонал.

Докладна специфікація програмного забезпечення (SRS) наведена у додатку А.

# 

# 3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 UML-проєктування програмного забезпечення

На діаграмі прецедентів (див. рис. 3.1) відображено основні дії користувача в системі та допоміжні взаємодії із службою сповіщень: реєстрація та вхід, керування довідником продуктів і рецептів, ведення щоденника харчування, перегляд аналітики, налаштування профілю та параметрів нагадувань. Діаграма фіксує межі системи та ролі учасників, слугуючи узагальненням функцій, описаних у вимогах.

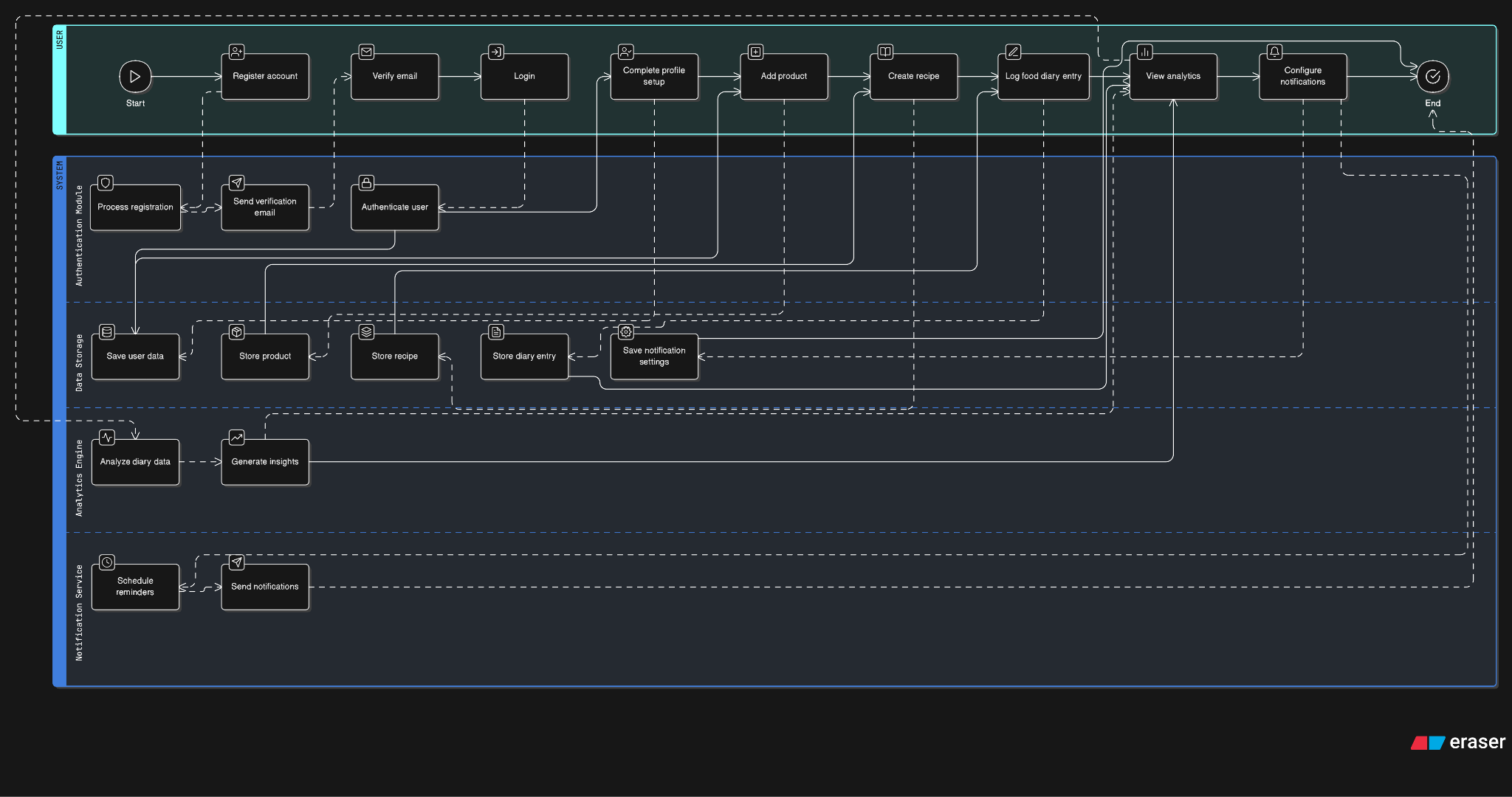


Рисунок 3.1 – Діаграма прецедентів програмної системи FoodDiary

Послідовність дій користувача під час додавання продукту до щоденника подано на діаграмі активностей (див. рис. 3.2). Вона охоплює пошук або створення продукту, перевіряння введених даних, попередній розрахунок поживної цінності та збереження запису.

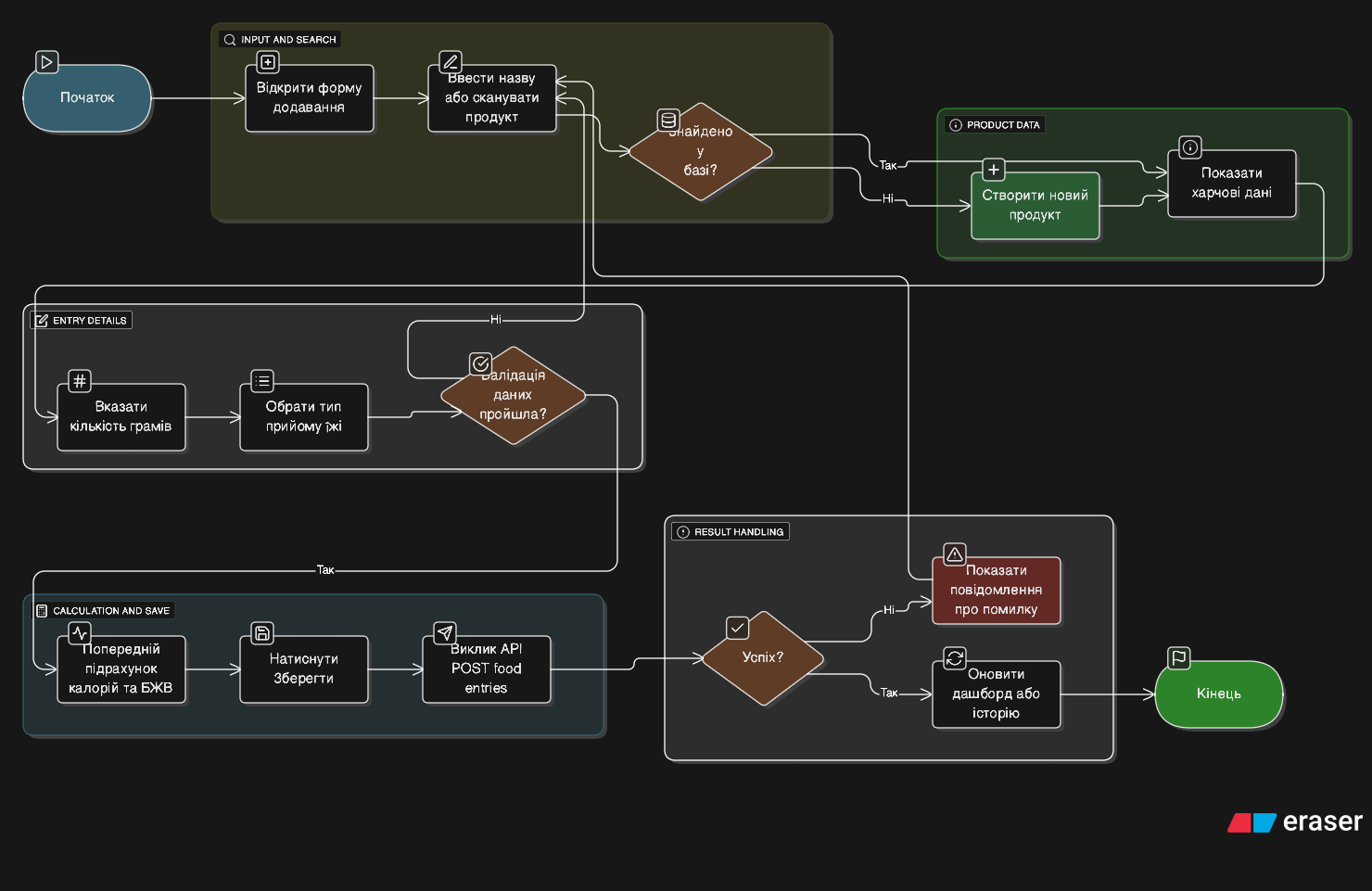


Рисунок 3.2 – Діаграма активностей «Додавання продукту до щоденника»

Логіка створення рецепта і подальшого внесення його до щоденника наведена на другій діаграмі активностей (див. рис. 3.3). Показано формування складу страви з інгредієнтів, обчислення поживності на порцію та додавання запису з урахуванням кількості порцій.

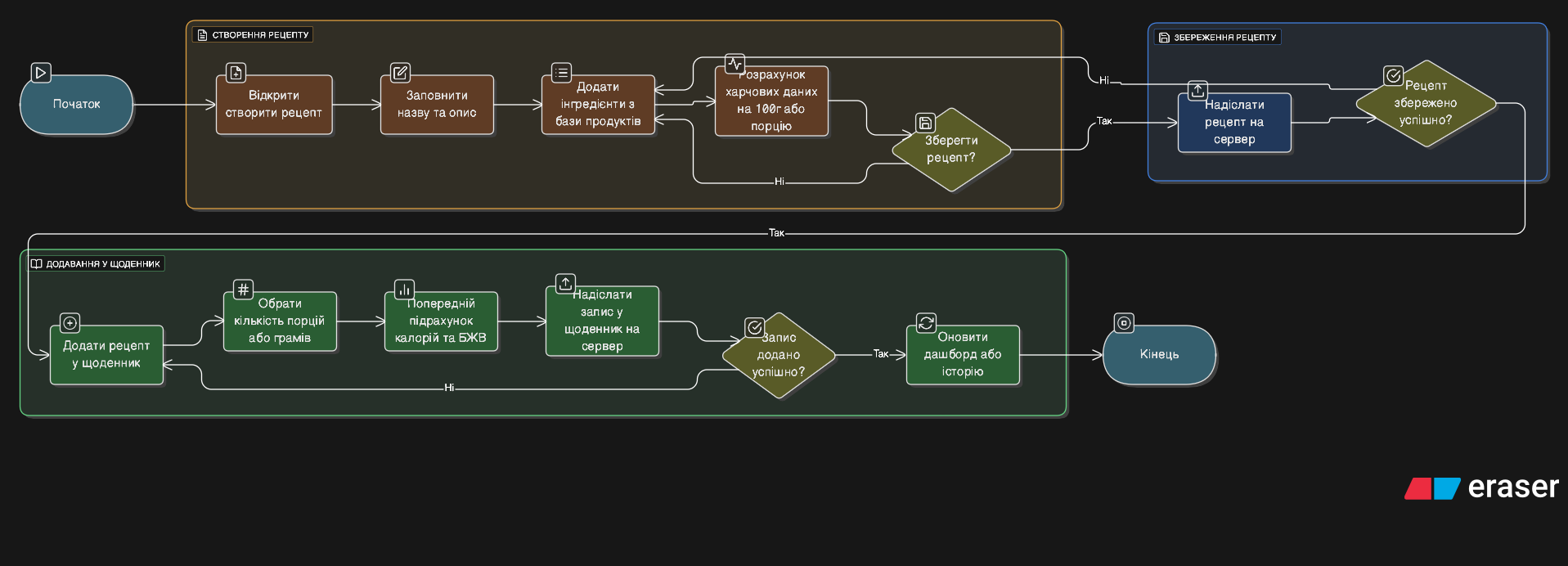


Рисунок 3.3 – Діаграма активностей «Створення рецепта та внесення його до щоденника»

## 3.2 Проєктування архітектури програмного забезпечення

Система реалізована у вигляді клієнт–серверного застосунку зі шаровою організацією та підходу “чиста архітектура”: рівень представлення (веб-клієнт на React і мобільний інтерфейс через Capacitor), прикладний рівень (веб-служба на ASP.NET Core), рівень бізнес-логіки (доменні сервіси, перевіряння, перетворення моделей) та рівень доступу до даних (PostgreSQL із використанням засобів об’єктно-реляційного відображення). [[1]](#_baf9wrb0k5u5) [[2]](#_baf9wrb0k5u5) [[3]](#_baf9wrb0k5u5) [[4]](#_baf9wrb0k5u5) [[5]](#_baf9wrb0k5u5)

Склад і взаємодію основних модулів подано на діаграмі компонентів (див. рис. 3.4). Веб-клієнт і мобільний клієнт звертаються до веб-служби через інтерфейс прикладного програмування на основі REST; зберігання відбувається у реляційному сховищі даних. Передавання даних здійснюється з’єднанням HTTP.

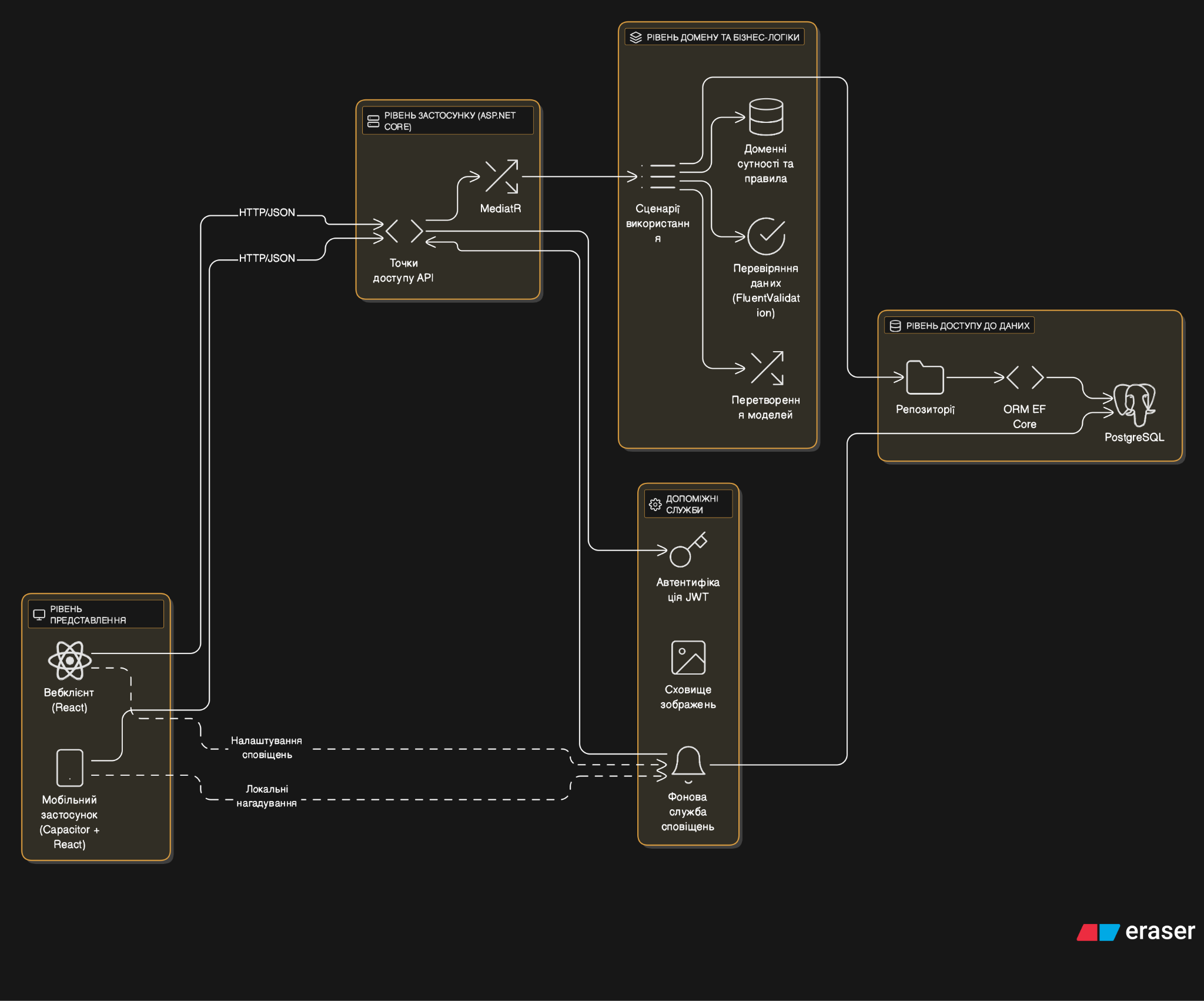


Рисунок 3.4 – Діаграма компонентів програмної системи FoodDiary

Розміщення складників системи наведено на діаграмі розгортання (див. рис. 3.5). Вона демонструє клієнтські пристрої (браузер, мобільні пристрої), веб-службу (ASP.NET Core) та сервер бази даних PostgreSQL, а також зовнішній вузол для надсилання електронних листів.

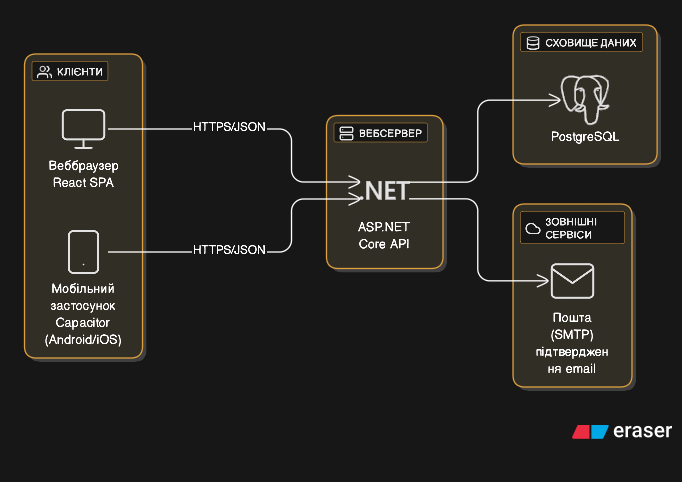


Рисунок 3.5 – Діаграма розгортання програмної системи FoodDiary

У процесі проєктування застосовано такі підходи і засоби: впровадження залежностей для керування життєвим циклом сервісів; шаблон «репозиторій» для ізоляції доступу до даних; об’єкти перенесення даних для відокремлення транспортних моделей від доменних; токени веб-стандарту JSON Web Token для автентифікації; єдині угоди щодо маршрутів і кодів станів для інтерфейсу прикладного програмування. [[6]](#_baf9wrb0k5u5)

## 

## 3.3 Проєктування структури зберігання даних

Логічна модель даних відтворює доменні поняття, що застосовуються у сценаріях користувача й сервісах. На ER-діаграмі (див. рис. 3.6) позначено первинні ключі, показано кардинальності зв’язків «1:1», «1:М» і «M:N»; атрибути, які відповідають зовнішнім ключам, не відображено, оскільки на цьому рівні подається узагальнення предметної області.

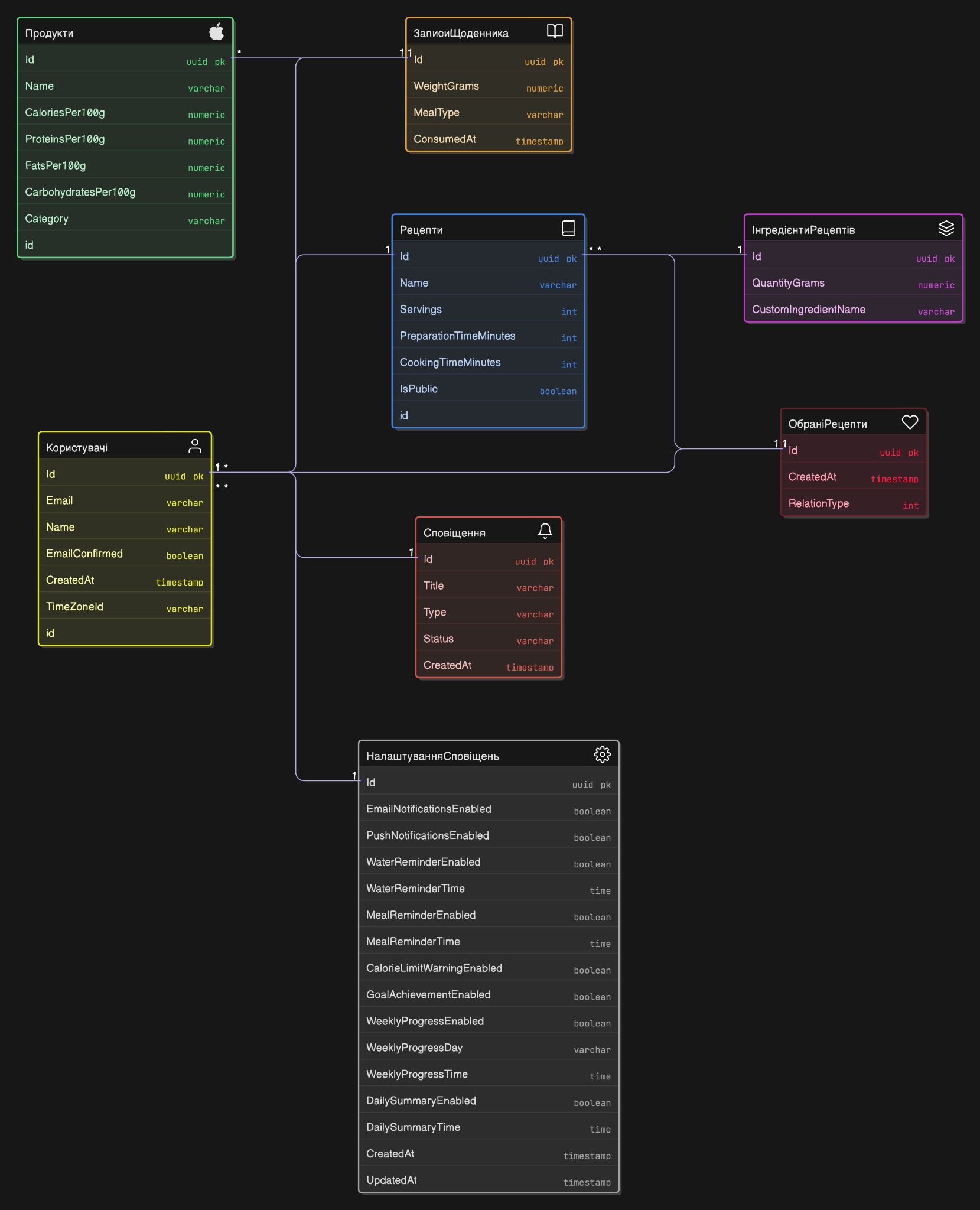


Рисунок 3.6 – ER-діаграма структури даних програмної системи FoodDiary

### 3.3.1 Сутності

1. Користувачі - обліковий профіль: унікальний ідентифікатор, електронна пошта, ім’я, ознака підтвердження пошти, час створення профілю, ідентифікатор часового поясу.
2. Продукти - довідник харчових позицій: назва, енергетична цінність і макронутрієнти на 100 г (калорійність, білки, жири, вуглеводи), категорія; за потреби - метадані зображення.
3. ЗаписиЩоденника - факти споживання: маса в грамах, тип прийому їжі, час споживання, нотатки.
4. Рецепти - користувацькі страви: назва, категорія, кількість порцій, час підготовки та приготування, інструкції, сукупні поживні показники, ознака публічності, мітки створення/оновлення; за потреби - зображення.
5. ІнгредієнтиРецептів - склад рецепта: кількість (грами), можливість вказати кастомний інгредієнт із власними поживними даними, якщо його немає у довіднику продуктів.
6. ОбраніРецепти - факт додавання рецепта користувачем до обраного; містить ідентифікатор та час створення запису.
7. Сповіщення - надіслані повідомлення: назва, тип, статус (прочитане/непрочитане), час створення; за потреби - допоміжні метадані.
8. НалаштуванняСповіщень - персоналізація нагадувань і підсумків:  
   WaterReminderEnabled/Time, MealReminderEnabled/Time,  
   CalorieLimitWarningEnabled, GoalAchievementEnabled,  
   WeeklyProgressEnabled/Day/Time,  
   DailySummaryEnabled/Time,  
   EmailNotificationsEnabled, PushNotificationsEnabled,  
   CreatedAt, UpdatedAt.

### 3.3.2 Взаємозв’язки

1. Користувачі - ЗаписиЩоденника: один користувач має багато записів щоденника (1:М), що забезпечує покриття щоденної фіксації раціону.
2. Користувачі - Рецепти: користувач може створювати багато рецептів (1:М), а рецепти належать автору.
3. Користувачі - Сповіщення: кожному профілю адресуються множинні повідомлення (1:М).
4. Користувачі - НалаштуванняСповіщень: для кожного профілю зберігається рівно один набір параметрів (1:1), що унеможливлює дублювання конфігурацій.
5. Продукти - ЗаписиЩоденника: один продукт може фігурувати у багатьох записах (1:М), що відображає повторюваність уживання.
6. Рецепти - ІнгредієнтиРецептів: рецепт складається з багатьох інгредієнтів (1:М), завдяки чому поживність рецепта обчислюється агрегуванням.
7. Користувачі - Рецепти (обране): зв’язок багато-до-багатьох реалізовано через ОбраніРецепти - відповідно Користувачі 1:М ОбраніРецепти та Рецепти 1:М ОбраніРецепти.

## 3.4 Приклади використаних алгоритмів та методів

У цьому підрозділі подано ключові алгоритми, що реалізують прикладну логіку системи FoodDiary, а також відповідні діаграми активностей і станів. Опис спирається на шари Core / UseCases / Infrastructure / Web та модель даних, наведені вище.

### 3.4.1 Обчислення поживності для запису щоденника

Алгоритм виконує перерахунок показників «на 100 г» у фактично спожиту масу.

Вхідні дані:  
Калорії на 100г, Протеїни на 100г, Жири на 100г, Вуглеводи на 100г, Маса.

Кроки розрахунку:

1. Перевірити, що Маса > 0.
2. Для кожного показника застосувати масштабування:

* калорії = (Калорії на 100г \* Маса)/100;
* білки = (Протеїни на 100г \* WeightGrams)/100;
* жири = (Жири на 100г \* Маса)/100;
* вуглеводи = (Вуглеводи на 100г \* Маса)/100.

1. Округлити до двох десяткових (узгоджено з точностями numeric(8,2) / numeric(10,2) у БД).
2. Зберегти запис; оновити денні підсумки в аналітиці (або обчислювати їх «на льоту» під час запиту аналітики).

Пограничні випадки:

* якщо будь-який з показників продукту відсутній - заборонити збереження або вимагати доповнення даних;
* маса = 0 - відхилити запит з повідомленням про помилку.

### 3.4.2 Обчислення поживності рецепта та масштабування порцій

Вхідні дані: список інгредієнтів рецепта (кожен - або посилання на продукт, або «кастомний» інгредієнт з власними показниками), їх маса у грамах, кількість порцій.

Кроки розрахунку:

1. Для кожного інгредієнта обчислити внесок за формулою з п. 3.4.1.
2. Просумувати всі внески → отримати загальні кількості елементів.
3. Обчислити масу рецепта як суму всіх.
4. За потреби отримати значення на 100 г:  
   де X∈{калорії,білки,жири,вуглеводи}.
5. Для додавання рецепта в щоденник на k порцій:
6. Округлити до двох десяткових; зафіксувати запис у щоденнику.

Транзакційність: створення рецепта та його інгредієнтів виконується в одній транзакції (успіх/відкат цілком).

### 3.4.3 Денний підсумок і попередження про ліміти

На діаграмі активностей (див. рис. 3.7) показано процес побудови щоденного підсумку та формування попереджень про наближення/перевищення індивідуальних цілей.

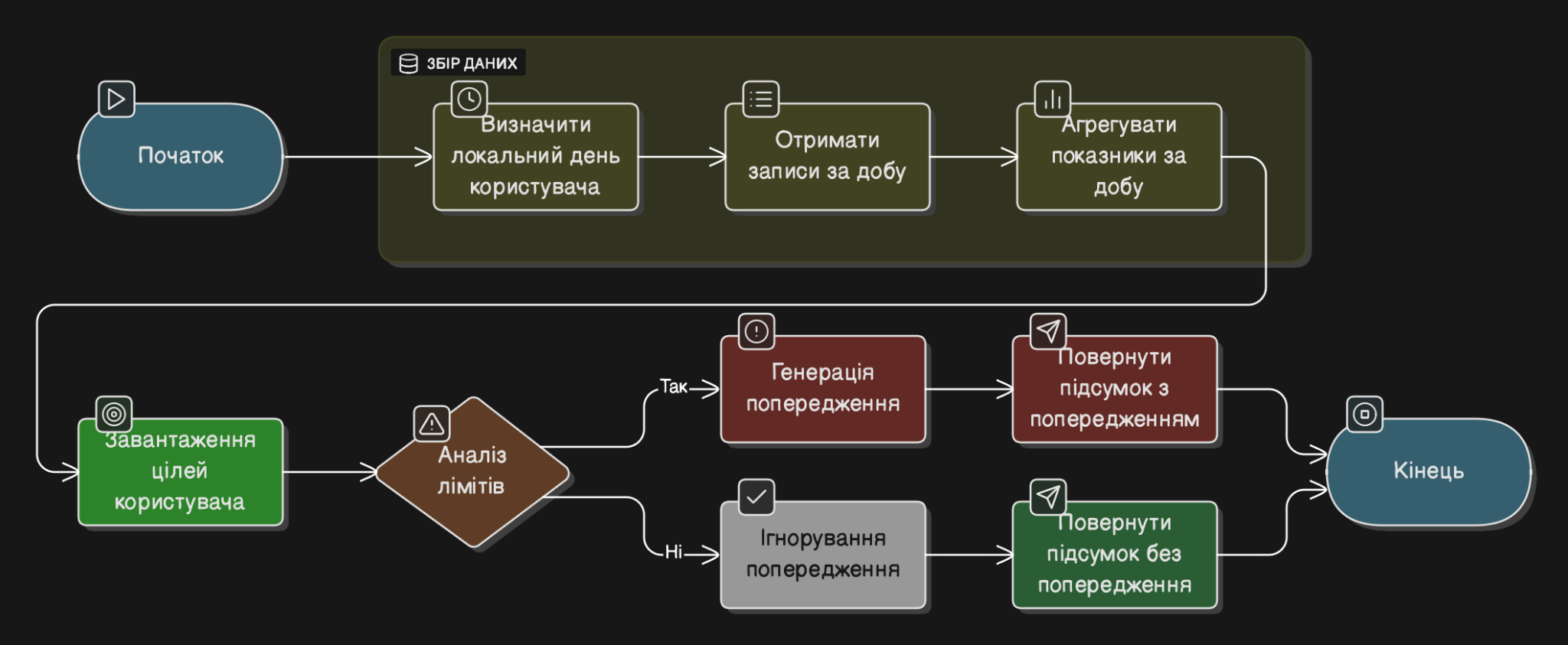


Рисунок 3.7 – Діаграма активностей «Щоденний підсумок і попередження»

Параметри порогів (приклад): попередження «наближення» при ≥ 90 % мети; «перевищення» при > 100 % мети. Точні значення зберігаються в налаштуваннях або конфігурації служби аналітики.

### 3.4.4 Планування нагадувань і життєвий цикл сповіщення

На діаграмі станів (див. рис. 3.8) зображено життєвий цикл сповіщення.

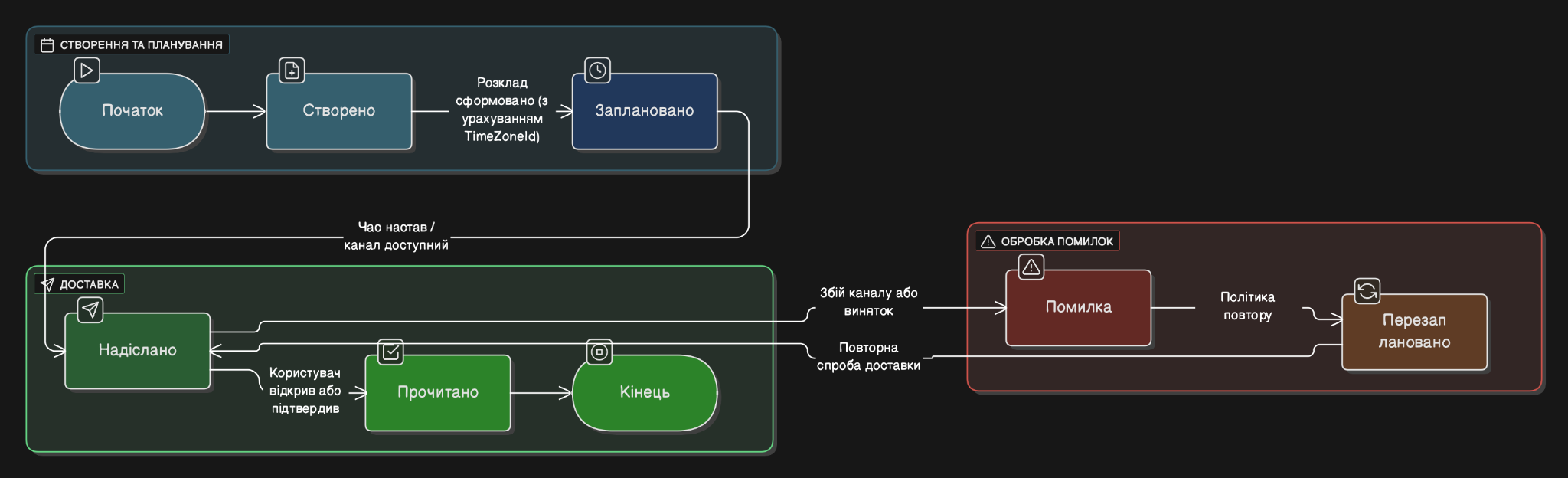


Рисунок 3.8 – Діаграма станів «Життєвий цикл сповіщення»

Алгоритм планування (скорочено):

1. Зчитати налаштування сповіщень користувача.
2. Для кожного ввімкненого типу обчислити наступний локальний час спрацювання.
3. Перевести локальний час у UTC для надійного зберігання/планування.
4. Створити запис сповіщення зі станом «Створено» → «Заплановано».
5. У момент настання події виконати відправлення (локальне для мобільного застосунку або поштове повідомлення) → «Надіслано».
6. За збоїв застосувати політику повтору (обмежена кількість нових спроб з інтервалами).
7. Після взаємодії користувача позначити як «Прочитано».

### 3.4.5 Перевіряння даних (валідація)

Запис щоденника:

* маса > 0;
* тип (сніданок, вечеря, і т.д) із дозволеного переліку;
* дата споживання не з майбутнього;
* нотатки - довжина ≤ 500.

Продукт:

* назва обов’язкова;
* усі поживні показники на 100 г ≥ 0;
* верхні межі - розумні (захист від помилок введення);
* назва унікальна в межах користувача (за вимогою).

Рецепт та інгредієнти:

* щонайменше один інгредієнт;
* маса > 0;
* для кастомного інгредієнта задані базові показники;
* порції ≥ 1.

Користувач/цілі:

* коректна часова зона користувача;
* цілі (ккал/БЖВ) - невід’ємні;
* відповідність взаємним обмеженням (за наявності).

### 3.4.6 Продуктивність та узгодженість

* агрегації виконуються за індексованими полями, що зменшує час формування зведень;
* операції створення рецепта та інгредієнтів - транзакційні;
* для планування сповіщень використовується уніфіковане зберігання часу в UTC із переведенням з/у локальний час користувача під час читання/планування.

## 3.5 Проєктування інтерфейсу та зручності користування

### 3.5.1 Призначення, користувачі та наскрізні сценарії

Інтерфейс *FoodDiary* спрямовано на щоденну рутину користувача: швидке фіксування спожитої їжі, своєчасне коригування раціону та наочне відстеження прогресу щодо індивідуальних цілей. У центрі уваги - три групи користувачів: новачки, які потребують максимально простого входження; досвідчені користувачі, для яких важлива розширена аналітика та робота з рецептами; а також користувачі зі спортивними цілями, які очікують підвищеної точності та гнучкого налаштування індикаторів. Для кожної з груп сформовано наскрізні сценарії: від додавання окремого продукту чи рецепта до щоденника - до планування дня, перегляду підсумків і швидкого виправлення дисбалансів. Інтерфейс передбачає мінімальну кількість кроків у критичних діях, чіткі підказки та однакову логіку на всіх платформах.

### 3.5.2 Інформаційна архітектура та навігація

Структуру інтерфейсу побудовано за принципом тематичних розділів: оглядова панель для щоденних підсумків, щоденник для фіксації спожитого, окремі розділи для рецептів і продуктів, аналітика для огляду тенденцій, а також профіль та налаштування. У вебверсії основні переходи зосереджено в бічному меню, яке забезпечує стабільний «якір» навігації; довільні дії (пошук, сповіщення, контекстні операції) згруповано у верхній панелі, щоб не перевантажувати робочий простір. У мобільній версії застосовано нижню панель швидкого доступу до головних розділів і жести для типових операцій зі списками (видалення, редагування, додавання до вибраного). Логіка шляхів уніфікована: користувач завжди чітко розуміє, де він перебуває, і як повернутися до попереднього стану.

### 3.5.3 Візуальна мова, компоненти та адаптивність

Візуальну мову побудовано на принципах стриманості та читабельності: небагато кольорів, виразна типографіка, послідовні відступи та ієрархія заголовків. Семантичні стани (успішно, помилка, попередження, інформація) виділяються кольором і піктограмою, що полегшує сприйняття навіть за високої щільності даних. Компонентний підхід гарантує однакову поведінку полів введення, карток, списків, модальних вікон і спливаючих повідомлень: користувач очікує однакові реакції системи в подібних ситуаціях, незалежно від розділу. Макети адаптуються до доступного простору без різких перестановок елементів; пріоритет має зміст - ключові показники та дії завжди залишаються в полі зору. Завдяки цьому інтерфейс придатний для використання на різних пристроях - від смартфона до настільного монітора - без втрати зрозумілості.

### 3.5.4 Доступність і локалізація

Інтерфейс проектується з урахуванням доступності: передбачено контрастне відображення основних текстів та індикаторів, видимі фокусні стани для керування з клавіатури, альтернативні описи для зображень і коректний порядок читання для озвучення. Анімації та переходи використовуються помірковано; у разі потреби інтерфейс підлаштовується під режим зменшеного руху. Текстова частина та формати подання даних локалізуються відповідно до мовних налаштувань користувача: інтерфейс доступний українською мовою, із можливістю переключення на інші мови. Формати дат, часу й чисел узгоджені з локацією, одиниці вимірювання - зрозумілі та послідовні в усьому застосунку.

### 3.5.5 Стан системи, повідомлення та стійкість до збоїв

Система чітко відображає свій стан: під час завантаження даних показуються скелетони або індикатори, при відсутності записів - продумані «порожні стани» з пропозиціями наступного кроку. Повідомлення про помилки зрозумілі та контекстні; користувач відразу бачить, що саме треба виправити. Для збоїв мережі передбачено режим роботи з локально збереженими даними та черги на синхронізацію із сервером; після відновлення з’єднання застосунок коректно об’єднує зміни. Нагадування та сповіщення налаштовуються індивідуально, відображаються ненав’язливо та не заважають основним діям; за потреби користувач може швидко змінити їхній режим або тимчасово вимкнути.

### 3.5.6 Продуктивність та підсумок

Продуктивність інтерфейсу підтримується вибором ощадливих засобів відображення, зваженими обсягами завантажуваних ресурсів і поступовою підвантажуваністю даних. Критичні дії - додавання запису, перегляд підсумку, внесення правок - виконуються без відчутних затримок; візуальна стабільність зберігається навіть у процесі оновлення вмісту. У підсумку побудована модель UI/UX поєднує простоту для новачків і глибину для досвідчених користувачів, забезпечує передбачувану поведінку компонентів, стійкість до нестабільної мережі та узгодженість між вебверсією та мобільними пристроями. Це створює послідовний користувацький досвід, у межах якого щоденні завдання виконуються швидко, а стратегічні цілі - залишаються в центрі уваги.

## 3.6 Підготовка даних

### 3.6.1 Джерела, очищення та нормалізація

Дані поділяються на довідникові (продукти), операційні (щоденник, рецепти, інгредієнти, сповіщення, налаштування сповіщень) та облікові (користувачі, профільні метрики, цілі, часовий пояс). Для всіх каналів надходження застосовується єдиний вхідний контроль: обрізання зайвих пробілів, перетворення порожніх необов’язкових полів на NULL, уніфікація числового вводу й десяткового роздільника, перевіряння діапазонів (маса > 0; поживні значення ≥ 0), коректність дат і часу споживання. Запобігання дублям забезпечується нормалізованим порівнянням назв (регістр, пропуски) та унікальністю у межах категорії/автора. Зображення проходять перевірку типу вмісту й розміру, а також визначення контрольної суми. У налаштуваннях сповіщень унеможливлюються суперечливі стани (якщо нагадування вимкнене - час не зберігається).

### 3.6.2 Трансформації та імпорт

Поживні значення продуктів зберігаються «на 100 г», у записах щоденника виконується пропорційний перерахунок до фактичної маси. Категорії продуктів узгоджуються з контрольованим словником; числова точність відповідає полям сховища, щоб уникати накопичення похибок. Усі часові мітки зберігаються у всесвітньому координованому часі; під час введення та відображення виконується перетворення за часовим поясом користувача. Для інгредієнтів рецепта: якщо це довідковий продукт - використовуються його поживні дані; якщо «власний інгредієнт» - зберігається повний локальний набір атрибутів без змішування з довідником. Початкове наповнення виконується пакетно: ідемпотентно (повторний запуск без дублювання), транзакційно (відкат у разі збою) та частинами (зменшення блокувань). Створюються типові записи довідника й налаштувань сповіщень.

### 3.6.3 Цілісність, якість і відновлення

Посилальна цілісність забезпечується порядком завантаження (спочатку «батьківські» сутності - користувачі, продукти, рецепти; далі залежні - записи, інгредієнти, сповіщення) і перевірками перед збереженням (наявність інгредієнтів у рецепті; повний набір атрибутів для «власного інгредієнта»; додатні маси та коректні типи прийомів їжі). Історичні записи не перераховуються після змін у довіднику - зберігається фактичний стан на момент внесення. Для пакетних імпортів формуються звіти якості з переліком відхилених рядків і причинами; усі операції журналюються, зберігаються мітки створення/оновлення та ідентифікатори запусків імпорту. Для небойових середовищ використовуються синтетичні дані; у разі потрапляння фрагментів реальних - застосовується знеособлення. Під час масових завантажень вторинні індекси, що не забезпечують унікальність, за потреби створюються після вставок; великі історичні масиви імпортуються датними блоками. Перед значними змінами виконується резервне копіювання з можливістю повного або вибіркового відновлення; сценарії міграцій і файли початкового наповнення зберігаються у системі контролю версій.

# 

# 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

## 4.1 Серверна частина: запуск, безпека, проміжний конвеєр

Сервер побудовано на ASP.NET Core із розділенням на шари (Core / UseCases / Infrastructure / Web). Під час запуску застосунок реєструє доступ до БД PostgreSQL через EF Core, підключає засоби автентифікації JWT, валідації, мапування об’єктів і формує конвеєр обробки запитів (статичні файли → CORS → автентифікація/авторизація → кінцеві точки API). [[7]](#_baf9wrb0k5u5) Це забезпечує прозоре керування залежностями, єдину точку входу та захищений доступ до ресурсів. Нижче наведено програмну реалізацію конфігурації автентифікації та CORS, а також вмикання відповідних проміжного програмного забезпечення.

builder.Services.AddAuthentication()

.AddJwtBearer(o => o.TokenValidationParameters = new TokenValidationParameters {

ValidateIssuer = true, ValidateAudience = true, ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtIssuer, ValidAudience = jwtAudience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret))

});

builder.Services.AddCors(o => o.AddPolicy("AllowAll", p =>

p.WithOrigins("http://localhost:3000","http://localhost:5173")

.AllowAnyMethod().AllowAnyHeader().AllowCredentials()));

app.UseCors("AllowAll");

app.UseAuthentication();

app.UseAuthorization();

## 4.2 Модель предметної області та конфігурація БД

Доменна модель охоплює користувачів, продукти, записи щоденника, рецепти та їх складники, сповіщення й налаштування сповіщень. Для відображення у PostgreSQL використано EF Core з Fluent API: типи, довжини, точності числових полів, конверсії переліків (enum) у рядки, індекси. Такий підхід дає контроль над фізичною моделлю та продуктивністю запитів. Описані нижче налаштування виконуються у класах конфігурацій EF Core для точного відображення доменної моделі в PostgreSQL. Для сутності користувача задаються обов’язковість і довжини рядків, перетворення переліків у текстові значення (задля читабельності та стабільності міграцій), точність числових полів (уникнення похибок під час обчислень поживності), а також індекси й унікальні обмеження для пришвидшення пошуку та гарантування цілісності даних. Витяг із конфігурації подано нижче.

builder.Property(u => u.Email).IsRequired().HasMaxLength(255);

builder.Property(u => u.Gender).HasConversion<string>().HasMaxLength(10);

builder.Property(u => u.DailyCalorieGoal).HasPrecision(8,2);

builder.HasIndex(u => u.Email).IsUnique();

builder.HasIndex(u => u.Name);

## 4.3 Доступ до даних: «Сховище» та специфікації

Щоб відокремити бізнес-логіку від деталей доступу до даних і зробити обробку помилок уніфікованою, звернення до PostgreSQL інкапсульовано в узагальнений репозиторій. Методи репозиторію повертають типізовані результати (успіх / помилка / не знайдено), що спрощує контролювання потоку виконання у сценаріях застосунку та полегшує модульне тестування. Для складних вибірок (фільтрація за кількома полями, підвантаження навігаційних властивостей, сортування, пагінація) використовується патерн специфікації: критерії та включення описуються в окремому класі й передаються в репозиторій як єдиний параметр запиту. Витяг інтерфейсу та приклад специфікації наведено нижче.

public async Task<Result<T>> AddAsync(T entity) {

await \_dbSet.AddAsync(entity);

await \_dbContext.SaveChangesAsync();

return Result.Success(entity);

}

public async Task<Result<List<T>>> ListAsync(params string[] includes) {

var q = \_dbSet.AsQueryable();

foreach (var inc in includes) q = q.Include(inc);

return Result.Success(await q.ToListAsync());

}

## 4.4 Командно-запитна модель (CQRS) через MediatR

Зміна стану (створення/оновлення/видалення) і читання розведені на окремі обробники. Напр., створення продукту: перевірка дублікатів → формування сутності → (за потреби) оброблення зображення → збереження → мапування відповіді. Така декомпозиція зменшує зв’язаність і полегшує еволюцію логіки [[8]](#_baf9wrb0k5u5). Щоб продемонструвати розведення записів і читання в рамках CQRS, нижче наведено приклад команди створення продукту та ключових кроків її обробника. Команда (код наведено нижче) оформлена у вигляді незмінного запису і містить лише дані, потрібні для створення сутності: назву, поживність на 100 г та (опційно) бінарні дані зображення. Відправлення відбувається через посередник MediatR, що зменшує зв’язаність між веб-шаром і бізнес-логікою та спрощує тестування окремих сценаріїв.

public record CreateProductCommand : IRequest<Result<CreateProductResponse>> {

public string Name { get; init; } = "";

public double CaloriesPer100g { get; init; }

public byte[]? ImageData { get; init; }

public string? ImageContentType { get; init; }

}

Обробник команди (код наведено нижче) реалізує покроковий конвеєр: попередня перевірка на дублікати (унікальність назви), побудова доменної сутності, опційна обробка зображення (перевірка типу/розміру, витяг метаданих), збереження через репозиторій у транзакційній операції та мапування у DTO відповіді. Повернення здійснюється уніфікованим типом Result<T>, що спрощує обробку помилок; у продуктивному коді до цього конвеєра додаються валідація FluentValidation і підтримка CancellationToken.

var duplicate = await CheckForDuplicateProduct(request.Name);

if (!duplicate.IsSuccess) return duplicate;

var product = CreateProductFromRequest(request);

var img = await HandleProductImage(request, product);

if (!img.IsSuccess) return img;

return await SaveProduct(product)

.Map(\_ => Result.Success(\_mapper.Map<CreateProductResponse>(product)));

## 4.5 Веб-API: кінцеві точки на базі FastEndpoints

Для веб-шару маршрути, політика доступу та приймання файлів описуються безпосередньо у класах кінцевих точок. Це дозволяє тримати декларативну частину (шлях, метод, авторизація, формат вхідних даних) поруч із точкою входу, а предметну логіку - в окремих обробниках, що спрощує читання й супровід. У наведеному нижче прикладі клас кінцевої точки створення продукту задає маршрут POST /api/products, дозволяє приймання файлів (multipart/form-data) та приймає валідаваний запит моделі; реалізацію бізнес-кроків у скороченому листингу опущено.

[Authorize]

public class CreateProductEndpoint: Endpoint<CreateProductRequest,CreateProductResponse> {

public override void Configure() { Post("/api/products"); AllowFileUploads(); }

public override async Task HandleAsync(CreateProductRequest req, CancellationToken ct) {

}

}

Для завантаження зображення продукту використовується окремий маршрут з увімкненим прийманням файлів; у тілі обробника (не наведено) виконується перевірка типу та розміру, зчитування потоку і делегування в команду завантаження зображення.

Post("/api/products/{ProductId}/image"); AllowFileUploads();

// валідація типу/розміру → читання потоку → команда UploadProductImageCommand

Процес створення продукту в інтерфейсі та результат збереження показано нижче (див. рис. 4.1): у частині а наведено форму заповнення, у частині б - відображення щойно нового продукту у довіднику.

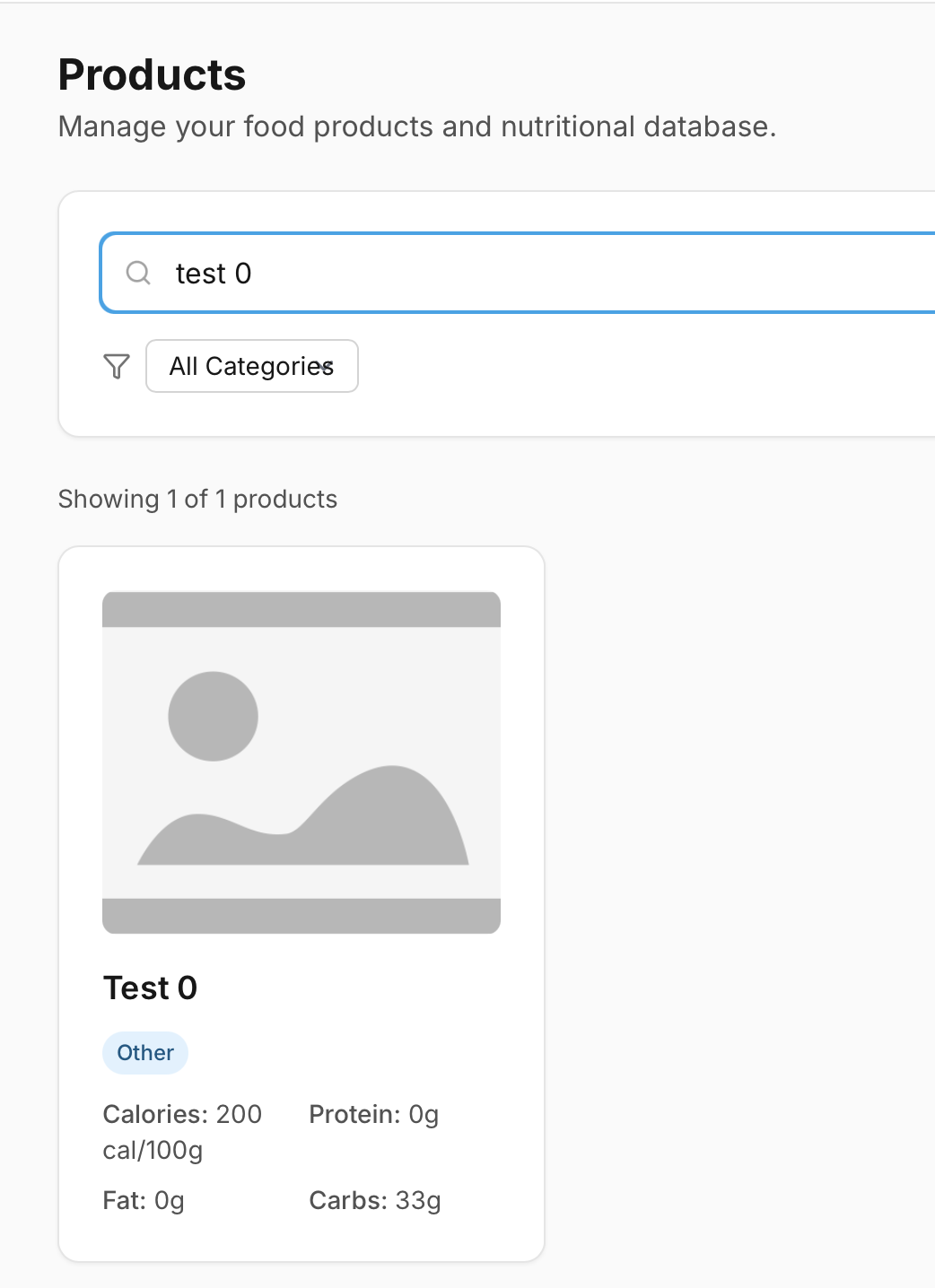
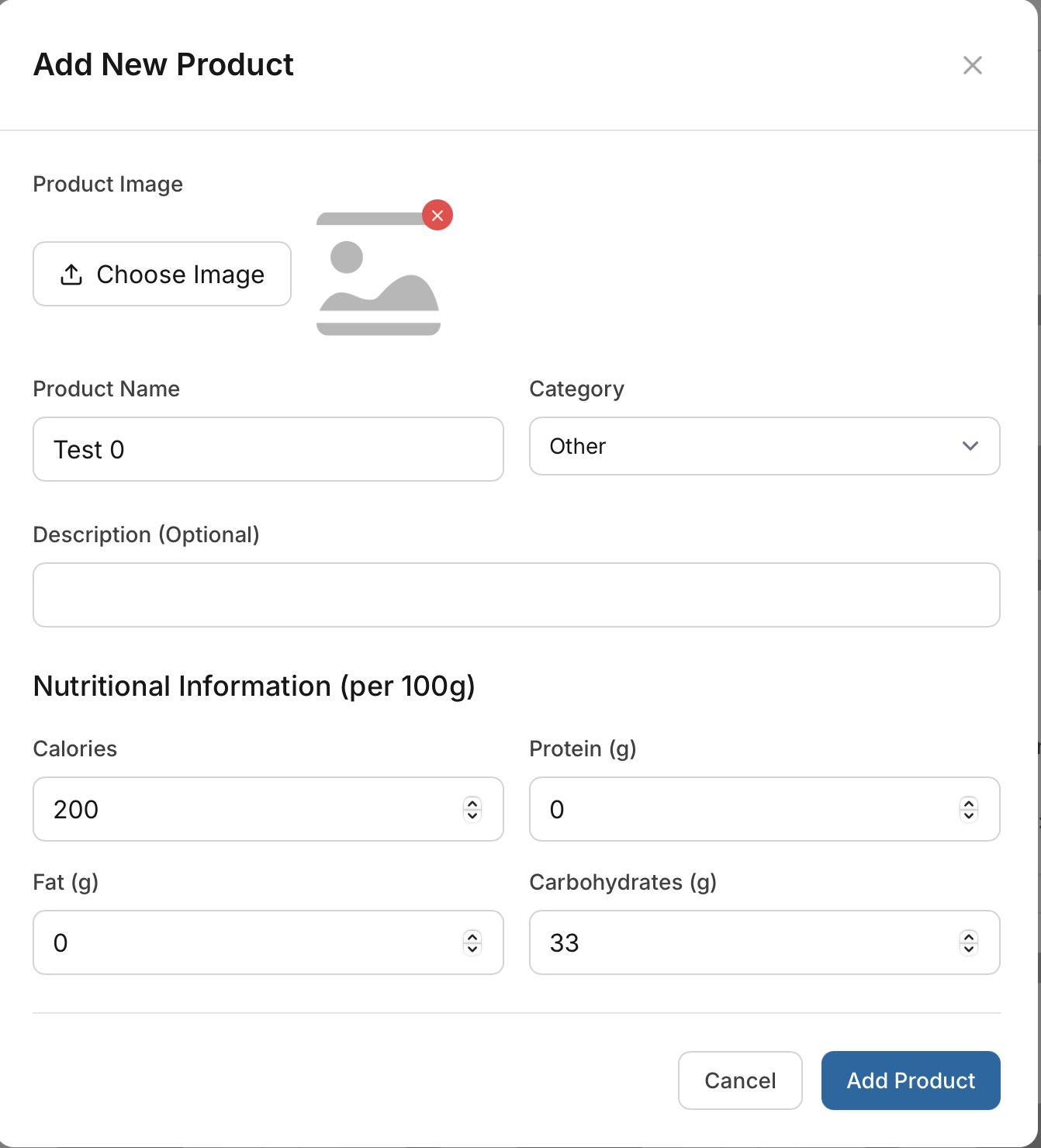


Рисунок 4.1 – Створення продукту в системі FoodDiary: а - форма заповнення; б - продукт додано до довідника

## 4.6 Клієнтська частина: маршрути, захист, клієнт API, стан, форми

### 4.6.1 Маршрутизація та охорона розділів

Для клієнтської частини навігацію організовано через вкладену маршрутизацію з базовим макетом Layout, що уніфікує шапку, навігаційні елементи та робочу область сторінок. Доступ до функціональних розділів (дашборд, щоденник тощо) контролює охоронний компонент ProtectedRoute: він перевіряє стан автентифікації користувача та, за відсутності чинної сесії, виконує перенаправлення на сторінку входу. Такий підхід зосереджує перевірку доступу в одному місці, спрощує супровід і запобігає дублюванню перевірок у кожній сторінці. Фрагмент реалізації наведено нижче.

<Routes>

<Route path="/login" element={<LoginPage/>}/>

<Route element={<Layout/>}>

<Route path="/dashboard" element={<ProtectedRoute><DashboardPage/></ProtectedRoute>}/>

<Route path="/diary" element={<ProtectedRoute><FoodDiaryPage/></ProtectedRoute>}/>

</Route>

</Routes>

На рисунку 4.2 показано перенаправлення користувача на форму входу за відсутності маркера доступу; на рисунку 4.3 - доступ до «Огляду» після успішної аутентифікації.

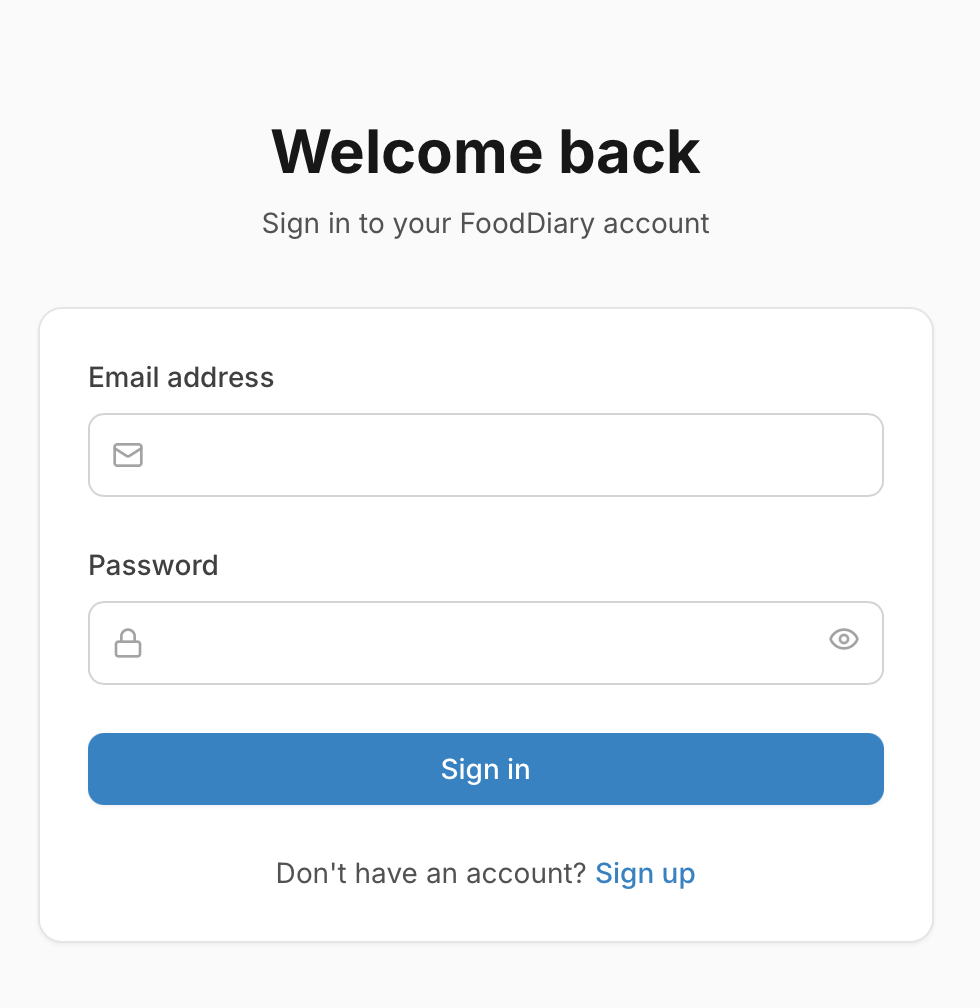


Рисунок 4.2 – Перенаправлення на сторінку входу за відсутності JWT

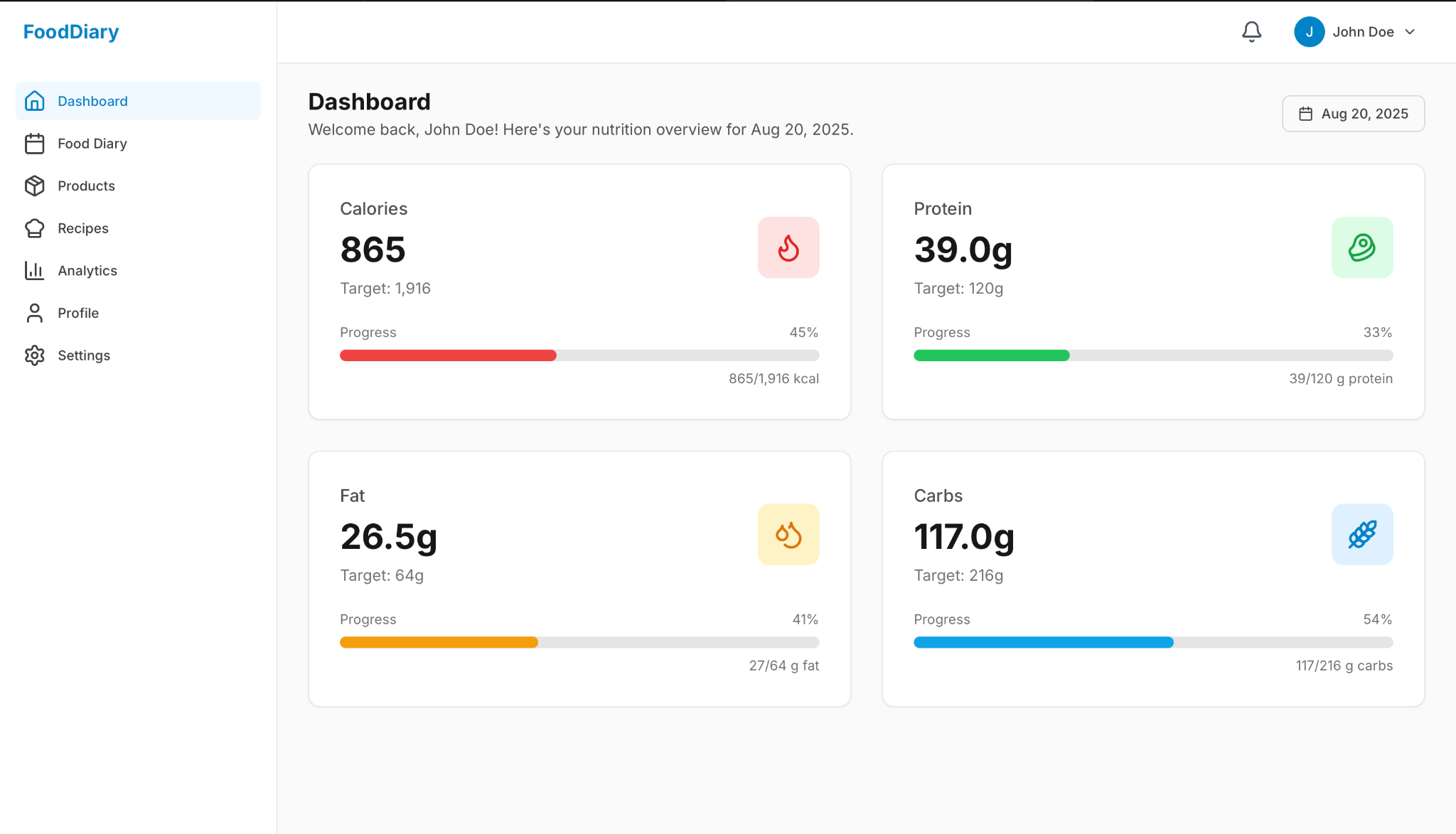


Рисунок 4.3 – Доступ до панелі «Огляд» після входу (JWT дійсний)

### 4.6.2 Єдиний клієнт до API

HTTP-клієнт (axios) із перехоплювачами додає JWT до кожного запиту та централізовано обробляє 401-відмови (скидання токена, повернення на сторінку входу). Це спрощує підтримку безпеки на клієнті. Фрагмент реалізації наведено нижче.

api.interceptors.request.use(cfg => {

const t = getAuthToken(); if (t) cfg.headers.Authorization = `Bearer ${t}`; return cfg;

});

api.interceptors.response.use(r => r, err => {

if (err.response?.status === 401) { removeAuthToken(); window.location.href = '/login'; }

return Promise.reject(err);

});

### 4.6.3 Сховище стану (Zustand)

Для записів щоденника реалізовані дії завантаження/створення/оновлення/видалення, фільтри та похідні обчислення (загальна поживність). Помилки від API відображаються в єдиному полі сховища, що полегшує керування повідомленнями в інтерфейсі. Фрагмент реалізації наведено нижче.

fetchEntries: async (date) => {

const resp = await fetch(`${API\_BASE\_URL}/api/food-entries?date=${encodeURIComponent(date)}`, { headers:{Authorization:`Bearer ${getAuthToken()}`}});

if (!resp.ok) handleApiError(resp);

const data = await resp.json();

set({ entries: Array.isArray(data.foodEntries) ? data.foodEntries : data });

}

### 

### 4.6.4 Форми та перевіряння даних

Для побудови форм застосовано зв’язку React Hook Form і Zod: схема перевіряння описується декларативно, а інтегрований вирішувач забезпечує миттєвий зворотний зв’язок під час введення. За рахунок спостереження за полями (watch) інтерфейс обчислює та показує попередній перегляд поживності відповідно до введеної ваги, що пришвидшує роботу користувача та зменшує кількість помилок. Фрагмент оголошення схеми та ініціалізації форми наведено нижче.

const schema = z.object({

productId: z.string().min(1, 'Оберіть продукт'),

quantity: z.number().min(1).max(10000),

mealType: z.number().min(0).max(3)

});

const { register, handleSubmit, watch } = useForm({ resolver: zodResolver(schema) });

Динамічну зміну попереднього розрахунку поживності під час редагування маси продукту показано на рисунку 4.4: у частині а наведено стан форми до зміни поля «Маса», у частиніб - після зміни, коли значення калорійності та Б/Ж/В перераховано.

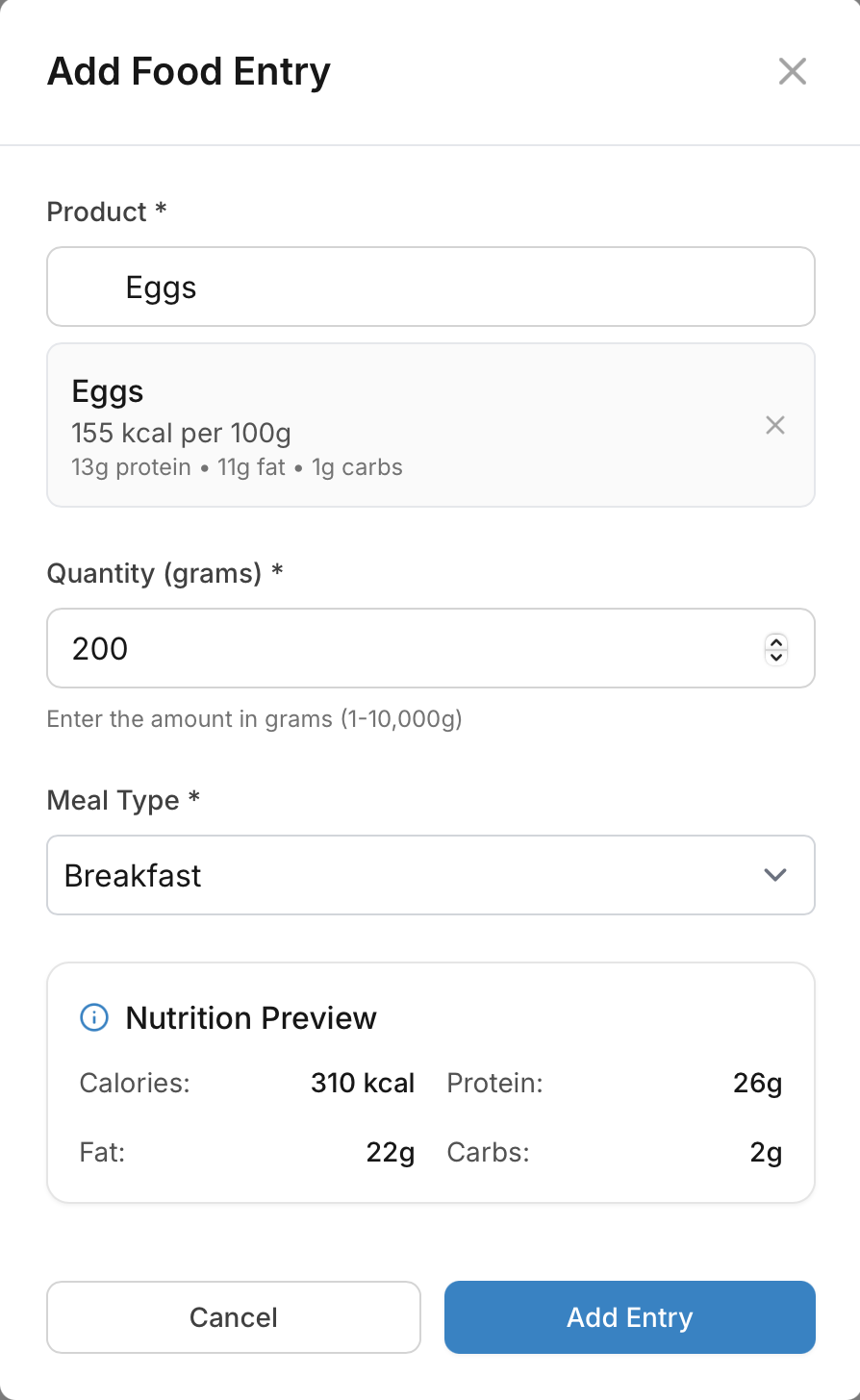
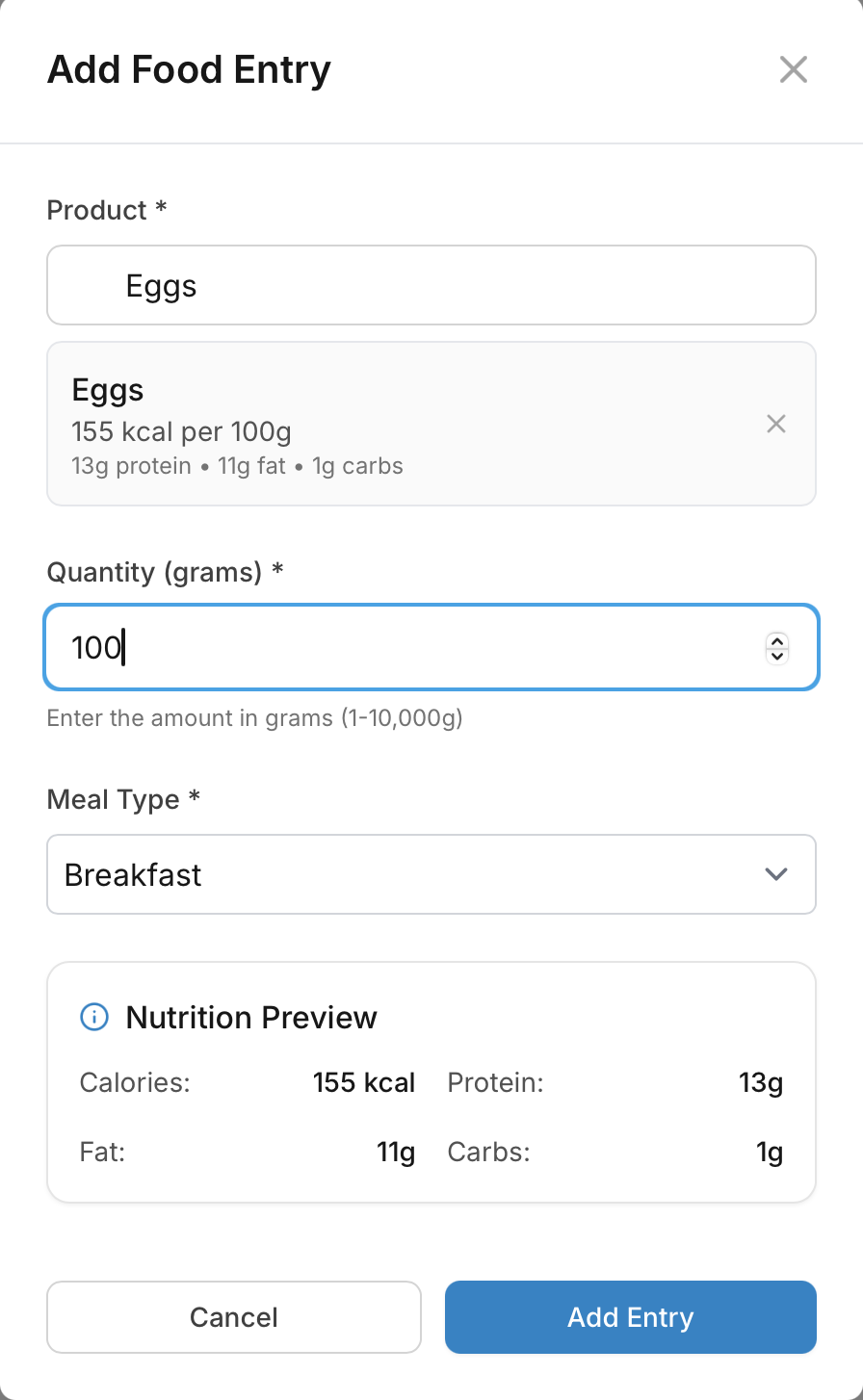


Рисунок 4.4 – Форма додавання запису з попереднім розрахунком поживності: а - до зміни маси; б - після зміни маси

### 4.6.5 Повторно вживані компоненти та доступність

Інтерфейс побудовано з невеликих багаторазових компонентів (картки, модалі, кнопки), що підтримують стани loading/error/disabled. Модальні вікна мають ARIA-атрибути і керування з клавіатури (фокус-менеджмент), списки - «скелетони» під час завантаження. Це покращує доступність і сприйняття швидкодії. Результат відображення щоденних записів із фільтрами та станами завантаження наведено на рисунку 4.5. [[9]](#_baf9wrb0k5u5)

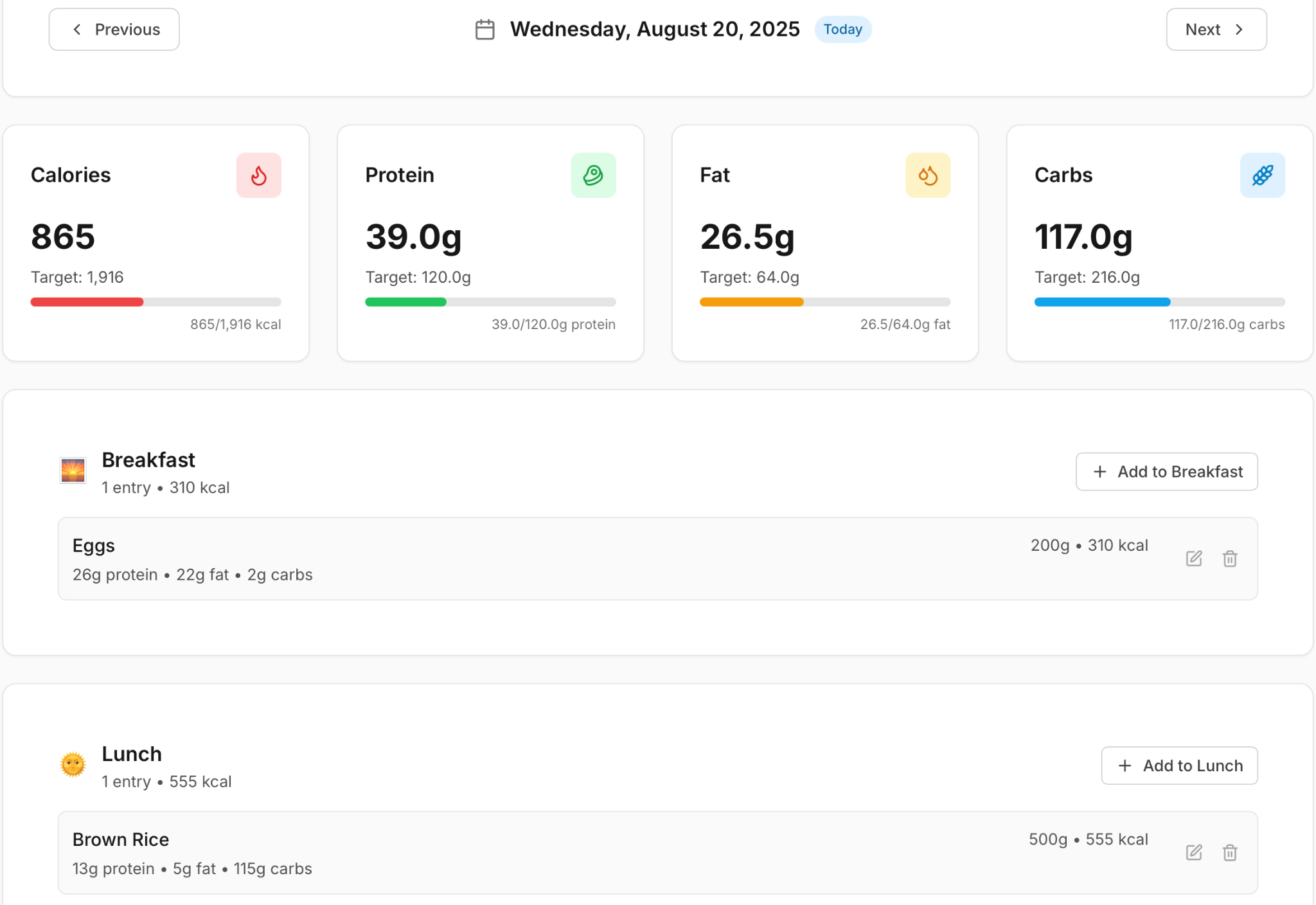


Рисунок 4.5 – Список записів за добу: групування за прийомами їжі, «скелетони» під час завантаження

## 4.7 Алгоритмічні рішення на практиці

У серверній частині використано сервісні методи для агрегування харчової цінності (за день/період), валідації даних та виявлення дублікатів. На клієнті ці розрахунки дублюються для миттєвого попереднього перегляду (без очікування відповіді API), а «джерелом істини» лишається сервер - це зменшує сприйняту затримку, не порушуючи цілісності даних. Концепція - «оптимістичний UX» зі строгою перевіркою на бекенді.

# 

# 5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 5.1 Порівняння результатів із початковими вимогами

Нижче узагальнено стан реалізації вимог, визначених у специфікації вимог до програмного забезпечення (SRS), та фактичний набір можливостей, який отримано у поточній версії системи.

### 5.1.1 Функціональні вимоги

Таблиця 5.1 – Відповідність функціональним вимогам SRS

| **Підсистема / вимога (SRS)** | **Стан виконання** | **Коментар** |
| --- | --- | --- |
| Харчовий щоденник: додавання/редагування/видалення записів; категорії прийомів їжі; денна статистика | виконано | Підтримуються сніданок/обід/вечеря/перекус, попередній перегляд поживності перед збереженням, миттєве оновлення підсумків за день. |
| База продуктів: пошук/перегляд; додавання нових продуктів; категоризація | виконано | Є пошук за назвою, зображення та опис; організація за категоріями; пагінація для великих вибірок. |
| Рецепти: створення/редагування; інгредієнти з перерахунком калорійності; збереження та повторне використання | виконано | Є улюблені рецепти; обчислення загальної поживності на основі інгредієнтів і порцій. |

Продовження таблиці 5.1 – Відповідність функціональним вимогам SRS

| Аналітика: день/тиждень/місяць; візуалізації; попередження про перевищення норм | виконано (базово) | Розрахунки підсумків і базові графіки. Розширені рекомендації винесено в окрему підзадачу еволюції. |
| --- | --- | --- |
| Сповіщення: нагадування про їжу та воду; персоналізація частоти | виконано | Керування параметрами в профілі; є типи нагадувань і часові налаштування. |
| Профіль: особисті дані та цілі; автоматичний розрахунок добових норм | виконано | Підтримуються метрики тіла, цілі та індивідуальні денні ліміти. |
| Безпека: авторизація через маркери доступу (JWT), хешування паролів | виконано | Паролі зберігаються у вигляді хешів; доступ до захищених розділів лише для автентифікованих користувачів. |



### 

### 5.1.2 Нефункціональні вимоги

Таблиця 5.2 – Відповідність нефункціональним вимогам SRS

| **Атрибут якості (SRS)** | **Оцінка відповідності** | **Коментар** |
| --- | --- | --- |
| Продуктивність інтерфейсу (до ~1 с) та обробка запитів до БД (до ~2 с) | загалом відповідає | На робочому стенді затримки не перевищують комфортні межі для типових сценаріїв (пошук, додавання, перегляд денної стрічки). |
| Надійність (безперервна робота) та резервне копіювання | частково виконано | Архітектурно підтримується розгортання у контейнерах; резервування БД передбачено на рівні середовища розгортання. |
| Масштабованість | виконано (архітектурно) | Шарова побудова та окреме сховище дають можливість горизонтального масштабування вебшару та винесення статичних об’єктів. |

Продовження таблиці 5.2– Відповідність нефункціональним вимогам SRS

| Зручність використання | виконано | Інтерфейс із зрозумілою навігацією, адаптивними екранами та індикаторами станів; швидкі дії для щоденних сценаріїв. |
| --- | --- | --- |



## 5.2 Оцінка якості роботи системи

Оцінювання якості системи виконано за ключовими характеристиками якості програмного забезпечення, зокрема з урахуванням моделі ISO/IEC 25010 [[10]](#_baf9wrb0k5u5). З погляду коректності та цілісності даних усі операції створення, редагування й видалення підтримують послідовний стан сховища: поживність перераховується відповідно до маси та складу, рецепти агрегують показники інгредієнтів, а денні підсумки синхронно оновлюються на дашборді. Зручність користування забезпечується зрозумілою структурою розділів (щоденник, продукти, рецепти, аналітика, профіль), попереднім переглядом поживності та контекстними підказками; критичні дії підтверджуються, а помилки відображаються безпосередньо біля відповідних полів введення. Рівень безпеки підтримується застосуванням маркерів доступу (JWT) для роботи з приватними ресурсами та зберіганням паролів у вигляді криптографічних хешів; розмежування доступу на рівні кінцевих точок серверної частини мінімізує ризик несанкціонованого перегляду. Продуктивність у типових сценаріях (пошук продуктів, додавання запису, перегляд статистики) сприймається як миттєва; водночас потенційними вузькими місцями лишаються масові операції (імпорт довідників) та обробка великих зображень, що потребують налаштування кешування і раціоналізації медіафайлів. Узагальнюючи ці спостереження, далі розглянемо, наскільки обрані архітектурні та технологічні рішення сприяли досягненню зазначених властивостей системи.

## 5.3 Аналіз ефективності прийнятих рішень

Аналіз ухвалених рішень засвідчує їхню практичну ефективність для задач FoodDiary. Шарова архітектура з чітким розмежуванням відповідальностей між рівнями подання, прикладної логіки, бізнес-правил і доступу до даних зробила код прозорим для супроводу, полегшила модульне тестування та дала змогу розвивати окремі частини незалежно, не порушуючи цілісності всього застосунку. Запровадження сценаріїв використання та посередника для запитів і команд із розділенням операцій читання та змін (CQRS) підвищило керованість бізнес-логіки: правила валідації та транзакційні межі локалізовано в одному місці, а читальні запити можна оптимізувати без ризику побічних ефектів. Обрана реляційна модель даних із чітко визначеними сутностями (користувачі, продукти, рецепти, інгредієнти, записи щоденника, налаштування сповіщень) забезпечує узгодженість і можливість ефективної індексації, що спрощує побудову аналітичних звітів і перевірок цілісності. Єдина кодова база інтерфейсу з адаптацією під мобільні пристрої зменшила витрати на підтримку та забезпечила однакові сценарії для веба й мобільного середовища; локальні нагадування покривають потреби користувачів поза браузером.

Водночас компроміси окреслюють подальші кроки вдосконалення. Зберігання зображень безпосередньо в базі даних спрощує забезпечення цілісності, проте збільшує обсяг резервного копіювання й навантаження на сховище; для масштабування доцільно винести медіафайли в об’єктне сховище з кешуванням і доставленням через CDN, залишивши в БД лише посилання та метадані. У разі зростання навантаження закладені логічні межі полегшують виділення окремих вузлів (аналітики чи сповіщень) у самостійні служби. Загалом ухвалені рішення виправдали себе в експлуатаційних сценаріях і задають зрозумілу траєкторію еволюції системи.

## 

## 5.4 Обмеження та недоліки

Попри досягнуті результати, система має низку обмежень, що впливають на досвід користування та подальше масштабування. По-перше, більшість функцій розрахована на постійне з’єднання з мережею: офлайн-сценарії розглядаються як допоміжні, тож без доступу до сервера можливості користувача звужуються. По-друге, точність розрахунків напряму залежить від якості довідника продуктів: наявність дубльованих позицій або неповних записів знижує достовірність підсумкової поживності та рекомендацій, а отже потребує механізмів верифікації та злиття даних. По-третє, зберігання зображень безпосередньо в базі даних спрощує підтримання цілісності, але підвищує вимоги до ємності сховища й резервного копіювання; з ростом кількості користувачів це стає чинником навантаження. Додатково, модуль порад нині працює в базовому режимі: персоналізація потребує навчання на історичних даних та врахування активності користувача. Нарешті, окрема консоль адміністрування відсутня; адміністративні операції виконуються опосередковано через сховище даних, що ускладнює щоденне обслуговування.

## 5.5 Напрями удосконалення

Визначені обмеження окреслюють практичні кроки розвитку. Насамперед доцільно посилити джерела харчових даних, інтегрувавши відкриті довідники та запровадивши модерацію записів разом із напівавтоматичним злиттям дублікатів-це підвищить точність підрахунків і якість пошуку. Для медіафайлів варто винести зберігання у зовнішнє об’єктне сховище з кешуванням і стисненням, а також обмежити розміри під час завантаження; у базі даних лишити лише посилання та метадані. Модуль рекомендацій доцільно розвинути у бік глибшого аналізу: виявлення дисбалансів за період, пропозиції страв під індивідуальні цілі, врахування типових часових шаблонів користувача. Окремим напрямом є спостережність і надійність: централізовані журнали подій, метрики часу відповіді та частки помилок, а також регулярні перевірки відновлення з резервних копій. Для масштабування-рознести сервер застосунку та базу даних на окремі вузли, додати проміжне кешування і зворотний проксі для статичних ресурсів. Щодо зручності, корисними стануть «швидкі дії» у щоденнику (додавання з історії, шаблони денного меню) та розширені довідкові підказки безпосередньо в інтерфейсі.

## 5.6 Узагальнення

Система реалізує головні можливості, визначені у специфікації: ведення щоденника, роботу з продуктами та рецептами, базову аналітику, нагадування, профіль і цілі, а також дотримання вимог до захисту даних. Якість взаємодії відповідає очікуванням: інтерфейс послідовний і зрозумілий, ключові сценарії виконуються без відчутних затримок, а перерахунок поживності та денні підсумки працюють узгоджено. Водночас окреслені напрями покращення-робота з медіа, розширення персоналізованих рекомендацій, запровадження спостережності та підготовка до масштабування-дають чітку траєкторію подальшої еволюції без зміни базових архітектурних рішень. Такий підхід забезпечує поступовий приріст цінності для користувачів і підтримуваність рішення у довгостроковій перспективі.

# 6 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмної системи FoodDiary виконувалось як комбінація модульних, інтеграційних і ручних наскрізних перевірок для трьох складових: серверної частини (ASP.NET Core із локальною БД), веб-клієнта (React у сучасних браузерах) і мобільного застосунку під Android (Capacitor). Робоче середовище - локальне; деплою на віддалені середовища немає, CI/CD не налаштовано. На сервері застосовано xUnit як основний фреймворк юніт-тестів у поєднанні з NSubstitute (мокінг залежностей) і FluentAssertions (твердження). Для ізоляції доменної логіки від інфраструктури використано in-memory провайдер EF Core; тестові дані ініціалізуються детермінованими фікстурами (seed), що забезпечує відтворюваність прогонів. Інтеграційні перевірки REST-API виконуються на базі WebApplicationFactory/TestServer: піднімається ізольований інстанс додатка з тестовою конфігурацією, проганяються сценарії автентифікації (позитивні/негативні, у т.ч. відсутній або протермінований JWT), CRUD-операції з доменними сутностями, перевіряється коректність контрактів JSON, ідемпотентність, обробка граничних та помилкових вхідних даних і очікувана авторизаційна відповідь (401/403) на захищених ендпойнтах. Зовнішні інтеграції інкапсулюються за інтерфейсами й замокані в тестах; зокрема сервіс відправки листів підміняється NSubstitute-реалізацією для фіксації викликів без реальних мережевих операцій. За результатами актуального прогону dotnet test отримано підсумок: total: 455, failed: 0, succeeded: 455, skipped: 0, duration: ~4,9 s.

Фронтенд перевірявся під час роботи, за стабільним смоук-чеклістом і короткими регресійними прогонами після суттєвих змін. Перевірені критичні користувацькі потоки: вхід/вихід/відновлення сесії; навігація між основними екранами; пошук і вибір продуктів; додавання, редагування та видалення записів у щоденнику; створення рецепта з інгредієнтами та додавання його за порціями; перегляд денної/тижневої аналітики; повідомлення про помилки та валідація форм (обов’язкові поля, діапазони значень). Окремо контролювались базові аспекти доступності (керування з клавіатури, видимість фокуса), поведінка в умовах нестабільної мережі (повтори запитів, індикація станів) і відпрацювання «порожніх станів» інтерфейсу.

Мобільний застосунок під Android перевірявся смоук- і короткими регресійними прогонами на емуляторах Android Studio. Перевірено запуск і холодний старт, стабільність навігації, збереження стану під час згортання/повороту екрана; локальне створення записів у офлайн-режимі з подальшою синхронізацією після відновлення мережі; обробку типових помилок мережевої взаємодії.

Середовища розділяються профілями конфігурацій; для автотестів використовується EF Core InMemory, для ручних перевірок - локальний екземпляр БД. Час у серверних сутностях зберігається в UTC, відображення у клієнтах відповідає часовому поясу користувача, що враховано в тестах обробки дат та агрегацій. Формалізовані нефункціональні випробування (замір продуктивності за методикою, безпекові аудити, SLA/SLI) не виконувались; у межах ручних прогонів підтверджено прийнятну чуйність основних екранів і відсутність витоку чутливих полів у типових відповідях API, а також очікувані статус-коди доступу.

Запуски тестів проводились локально (dotnet test); деплою та CI/CD немає.

# 7 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Впровадження FoodDiary виконується локально і зводиться до мінімальних змін конфігурацій та послідовного запуску трьох частин: API, веб-клієнта і мобільного клієнта. Спочатку в src/FoodDiary.Web/appsettings.jsonзаповніть ConnectionStrings:DefaultConnection (ваш PostgreSQL), EmailSettings (робочий SMTP для підтвердження пошти) і JwtSettings (Secret, Issuer, Audience). Профілі запуску вже задані в launchSettings.json: профіль http слухає http://0.0.0.0:5001/ (стартова сторінка - Swagger), https - https://0.0.0.0:57679/, також є IIS Express з http://localhost:57678/. Далі виконайте dotnet restore, dotnet build, застосуйте міграції dotnet ef database update і запустіть API командою cd src/FoodDiary.Web && dotnet run; інтерфейс Swagger буде за адресою http://localhost:5001/swagger. Для веб-клієнта в каталозі фронтенду створіть або змініть .env, вказавши VITE\_API\_BASE\_URL=http://localhost:5001 (якщо відкриватиметесь із телефона або емулятора в тій самій мережі - використайте свою локальну IP: http://<ваша\_IP>:5001), після чого запустіть npm install і npm run dev та працюйте за адресою, яку покаже Vite (типово http://localhost:5173). Для мобільного клієнта в .env мобайлу так само пропишіть VITE\_API\_BASE\_URL=http://<ваша\_IP>:5001, щоб застосунок бачив API через LAN; далі послідовно виконайте npm install, запустіть на Android npm run android, за потреби зберіть веб-ресурси npm run build, синхронізуйте з нативними платформами npm run sync, а для відкриття проєкту в Android Studio скористайтесь npm run open:android. Після старту достатньо відкрити Swagger, зареєструвати користувача (лист підтвердження прийде через ваш SMTP), увійти у веб або мобільний клієнт і додати тестовий запис у щоденник, щоб пересвідчитись у коректній взаємодії з локальним API.

# ВИСНОВКИ

У межах курсового проєкту створено програмну систему FoodDiary для ведення харчового щоденника, роботи з продуктами та рецептами, перегляду підсумків і отримання нагадувань. Реалізовано ключові сценарії: додавання/редагування/видалення записів, попередній перегляд поживності, довідник продуктів із пошуком, конструктор рецептів із перерахунком поживності, базова аналітика за періодами, персональні сповіщення та профіль із цілями.

Архітектура клієнт–серверна, шарова: серверна частина на ASP.NET Core, інтерфейс на React (з мобільною збіркою через Capacitor), сховище - PostgreSQL. Передбачено автентифікацію маркерами доступу, хешування паролів, валідацію даних та індекси для типових запитів. За підсумками зіставлення з SRS, основні функції виконано; інтерфейс працює швидко в типових сценаріях і придатний до подальшого розширення.

Виявлені обмеження стосуються якості довідника продуктів (потребує джерел перевірених даних і злиття дублікатів), зберігання зображень у базі (доцільне винесення у зовнішнє сховище), а також базового рівня персональних рекомендацій. Подальший розвиток варто зосередити на: інтеграції зовнішніх довідників харчової цінності, розширенні модуля рекомендацій, оптимізації зберігання медіафайлів і підсиленні спостережності (журнали подій, метрики продуктивності).

Репозиторій вихідних кодів: [[11]](#_baf9wrb0k5u5)

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. **Martin R. C.** Clean Architecture: A Craftsman’s Guide to Software Structure and Design. - Boston : Addison–Wesley Professional, 2017. - 432 p. (дата звернення: 22.05.2025)
2. **React Team.** React Documentation. - URL: <https://react.dev> (дата звернення: 15.06.2025).
3. **Microsoft.** ASP.NET Core documentation. - URL: <https://learn.microsoft.com/aspnet/core> (дата звернення: 25.05.2025).
4. **Evans E.** Domain-Driven Design: Tackling Complexity in the Heart of Software. - Boston : Addison–Wesley Professional, 2003. - 560 p. (дата звернення: 22.05.2025)
5. **PostgreSQL Global Development Group.** PostgreSQL 16 Documentation. - URL: <https://www.postgresql.org/docs/16/> (дата звернення: 25.05.2025).
6. **Jones M., Bradley J., Sakimura N.** JSON Web Token (JWT) : RFC 7519. - IETF, 2015. - URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 01.06.2025).
7. **Microsoft.** Entity Framework Core documentation. - URL: <https://learn.microsoft.com/ef/core> (дата звернення: 30.05.2025).
8. **MediatR Project.** MediatR (CQRS/Mediator for .NET) : документація. - URL: <https://github.com/jbogard/MediatR> (дата звернення: 09.06.2025).
9. **W3C.** Web Content Accessibility Guidelines (WCAG) 2.2 : W3C Recommendation. - 2023. - URL: <https://www.w3.org/TR/WCAG22/> (дата звернення: 16.06.2025).
10. **ISO/IEC 25010:2011.** Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. - Geneva : ISO/IEC, 2011. - 34 p. (дата звернення: 20.06.2025)
11. **FoodDiary Project.** Репозиторій вихідних кодів. - URL: [GitHub](https://github.com/NureKoretskyiIgor/2025_B_KKP_PZPI-22-1_Koretskyi_I_O/tree/main/2025_%D0%91_%D0%9A%D0%9A%D0%9F_%D0%9F%D0%97%D0%9F%D0%86-22-1_%D0%9A%D0%BE%D1%80%D0%B5%D1%86%D1%8C%D0%BA%D0%B8%D0%B9_%D0%86_%D0%9E) (дата звернення: 20.06.2025).

# ДОДАТОК А - СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SRS)

# 1 ВСТУП

## 1.1 Огляд продукту

FoodDiary - програмна система для фіксації щоденного харчування, ведення довідника продуктів та рецептів, перегляду аналітики й отримання персональних підказок. Рішення включає веб-застосунок, мобільний застосунок та серверну частину з єдиною базою даних.

## 1.2 Мета

Забезпечити користувача зручним інструментом контролю раціону (калорійність і макронутрієнти), плануванням прийомів їжі та формуванням здорових звичок за рахунок нагадувань і візуалізацій прогресу.

## 1.3 Межі

Специфікація охоплює функції й обмеження користувацького рівня, інтерфейси та вимоги до якості. Окрема консоль адміністрування не передбачається; технічне супроводження виконується засобами СУБД та інфраструктури.

## 1.4 Посилання

* модель якості ISO/IEC 25010;
* веб-доступність WCAG 2.2;
* рекомендації OWASP щодо веб-безпеки;
* специфікації HTTP/HTTPS, JSON;
* керівництва з використання ASP.NET Core, PostgreSQL, React, Android/Capacitor.

## 1.5 Означення та абревіатури

* ККП - курсова кваліфікаційна робота;
* SRS - специфікація програмних вимог;
* API - інтерфейс прикладного програмування;
* JWT - маркер доступу JSON Web Token;
* СУБД - система керування базами даних;
* БЖВ - білки, жири, вуглеводи;
* БД - база даних.

# 2 ЗАГАЛЬНИЙ ОПИС

## 2.1 Перспективи продукту

Система позиціонується як особистий помічник з харчування: ведення щоденника, власних рецептів, аналіз трендів і нагадування. Подальший розвиток - інтеграції з відкритими довідниками поживності та фітнес-пристроями.

## 2.2 Функції продукту

Облік продуктів і рецептів; додавання записів до щоденника; автоматичний перерахунок поживності; аналітичні звіти за періоди; персональні нагадування; керування профілем і цілями.

## 2.3 Характеристики користувачів

Індивідуальні користувачі без спеціальної підготовки у дієтології. Інтерфейс має бути простим, з підказками та попереднім переглядом розрахунків.

## 2.4 Загальні обмеження

Потрібен доступ до мережі Інтернет; актуальність розрахунків залежить від якості довідника продуктів; мобільна версія - для сучасних Android-пристроїв.

## 2.5 Припущення й залежності

* користувач має сучасний браузер або Android-пристрій;
* серверна інфраструктура підтримує масштабування;
* зображення надсилаються у прийнятних розмірах;
* часові пояси коректно налаштовані.

# 3 КОНКРЕТНІ ВИМОГИ

## 3.1 Вимоги до зовнішніх інтерфейсів

### 3.1.1 Інтерфейс користувача

Розділи:

* «Щоденник»;
* «Продукти»;
* «Рецепти»;
* «Аналітика»;
* «Профіль»;
* «Налаштування».

Ключові сценарії:

* швидке додавання запису;
* створення рецепту з інгредієнтів;
* перегляд денних/тижневих підсумків;
* редагування цілей;
* керування нагадуваннями.

Інші деталі інтерфейсу:

* адаптивність для настільних і мобільних екранів;
* доступність згідно з базовими принципами WCAG.

### 3.1.2 Апаратний інтерфейс

* підтримуються сучасні Android-пристрої;
* камера використовується лише для фото страв (за дозволом користувача).

### 3.1.3 Програмний інтерфейс

* публічний інтерфейс прикладного програмування (REST) з переданням даних у форматі JSON;
* автентифікація - маркерами JWT;
* стабільні версії маршрутів (v1).

### 3.1.4 Комунікаційний протокол

* обмін даними відбувається через HTTP;
* часові мітки передаються в UTC із урахуванням часового поясу користувача на рівні профілю.

### 3.1.5 Обмеження пам’яті

* зберігання медіафайлів - у межах політики розміру;
* у БД зберігаються зображення й метадані (для спрощення цілісності).

### 3.1.6 Операції

* типові операції: створення/редагування/видалення продуктів, рецептів та записів;
* додавання рецепту до щоденника;
* перегляд звітів;
* керування налаштуваннями сповіщень;
* зміна цілей.

### 3.1.7 Функції продукту (детальні вимоги)

* автоматичний перерахунок поживності для записів і рецептів;
* пошук і фільтрація за назвою, категорією, датою;
* вибір порції/маси з миттєвим попереднім переглядом;
* улюблені рецепти;
* нагадування про воду та прийоми їжі, щоденні підсумки;
* аналітика за день/тиждень/місяць.

### 3.1.8 Припущення й залежності (для інтерфейсів)

* стійкий мережевий канал;
* синхронізація даних між веб- і мобільною версіями;
* коректна робота служб сповіщень ОС.

## 3.2 Властивості програмного продукту

* цілісність даних: каскадні правила видалення, перевіряння полів, перерахунок поживності при зміні маси/складу;
* узгодженість інтерфейсу: єдині шаблони форм і таблиць, попередні перегляди, підтвердження критичних дій;
* спостережність: журнали подій сервера, базові метрики (помилки, час відповіді);

## 3.3 Атрибути програмного продукту

### 3.3.1 Надійність

* безперервна робота серверної частини;
* процедури резервного копіювання БД;
* відновлення після збоїв.

### 3.3.2 Доступність

* підтримка основних настільних браузерів;
* адаптація для мобільних пристроїв;
* оброблення відсутності мережі на клієнті (кеш мінімальних даних).

### 3.3.3 Безпека

* автентифікація JWT;
* хешування паролів;
* розмежування доступу до ресурсів користувача;
* захист від типових веб-загроз (ін’єкції, підміна запитів тощо).

### 3.3.4 Супроводжуваність

* шарова побудова;
* виділення доменних сутностей;
* стандартизовані інтерфейси;
* міграції БД;
* автоматизовані тести основних сценаріїв.

### 3.3.5 Переносимість

* розгортання у контейнерах;
* незалежність від конкретного постачальника хмарних послуг.

### 3.3.6 Продуктивність

* швидкий відгук інтерфейсу на типові дії;
* ефективні індекси БД;
* пагінація та фільтрація списків;
* економне використання мережі.

## 3.4 Вимоги бази даних

Модель даних охоплює сутності: користувачі продукти, рецепти, інгредієнти рецептів, записи щоденника, налаштування сповіщень, сповіщення, улюблені рецепти (асоціативна сутність).

Цілісність:

* первинні та зовнішні ключі;
* унікальність електронної пошти;
* обмеження типів;
* каскадні правила видалення згідно з призначенням.

Продуктивність:

* індекси для пошуку за назвою, датою, користувачем;
* складені індекси для частих відборів.

Резервування:

* регулярні резервні копії;
* засоби відновлення.

## 3.5 Інші вимоги

* технологічна база: серверна частина - ASP.NET Core, СУБД - PostgreSQL, клієнт - React, мобільний - Capacitor/Android;
* конфіденційність: персональні дані використовуються виключно для роботи сервісу; дотримання політики приватності;

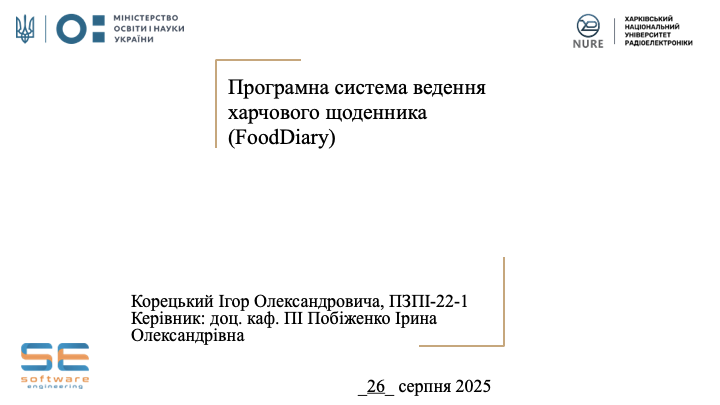
# 4 РОБОЧЕ СЕРЕДОВИЩЕ

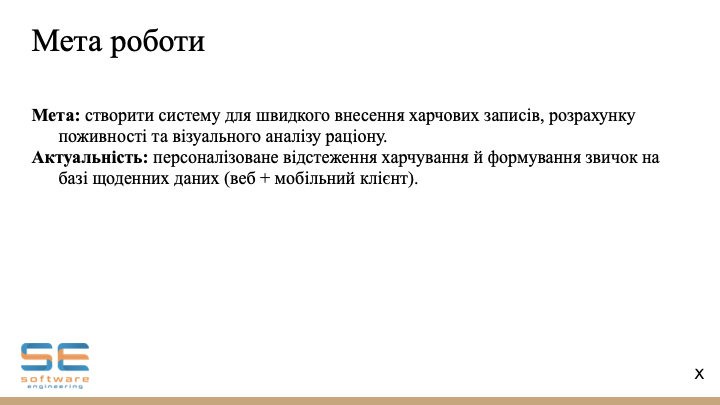
Сервер: веб-сервер застосунку на ASP.NET Core; СУБД PostgreSQL.  
Клієнт: сучасні браузери (настільні/мобільні); мобільний застосунок на Android (через Capacitor).  
Протоколи: HTTP, REST, обмін даними у форматі JSON.

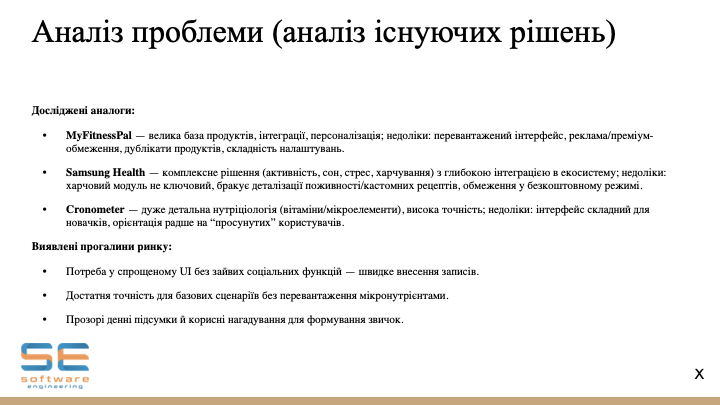
# 5 МАЙБУТНІ ВДОСКОНАЛЕННЯ

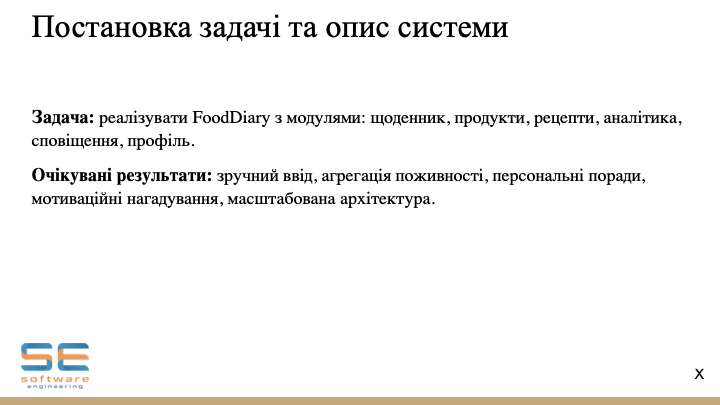
Інтеграції з відкритими довідниками харчової цінності та носимими пристроями; автоматичне формування списку покупок; розширені рекомендації на основі історичних даних; винесення зберігання медіафайлів у зовнішнє об’єктне сховище; поглиблена спостережність (журнали, метрики).

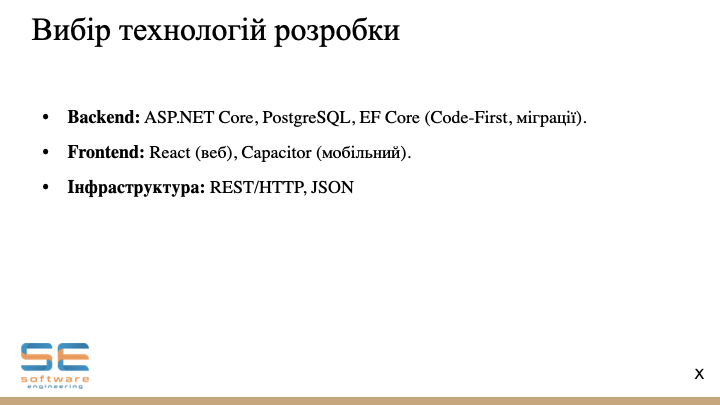
# ДОДАТОК Б - ПРЕЗЕНТАЦІЯ



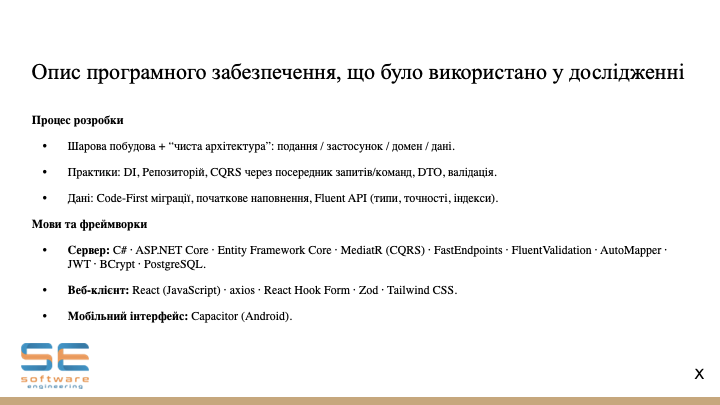


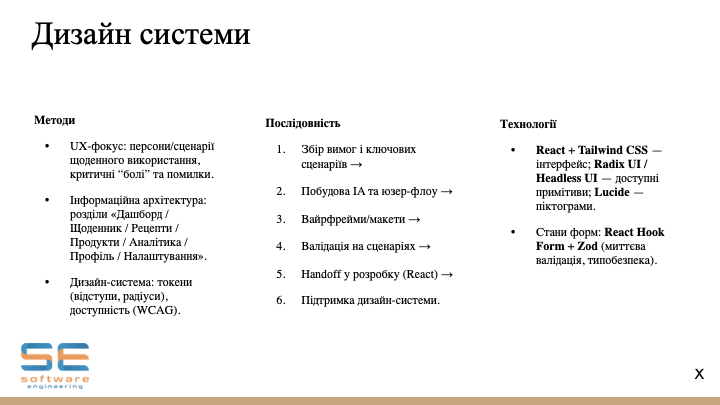


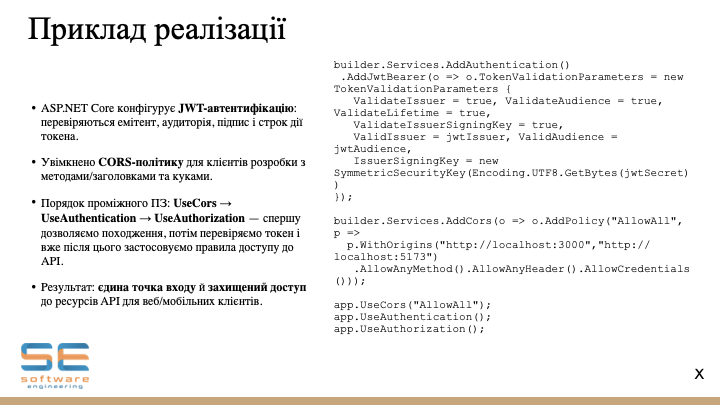


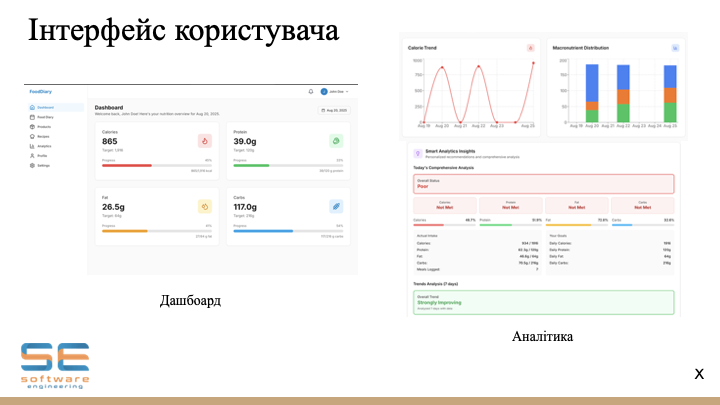


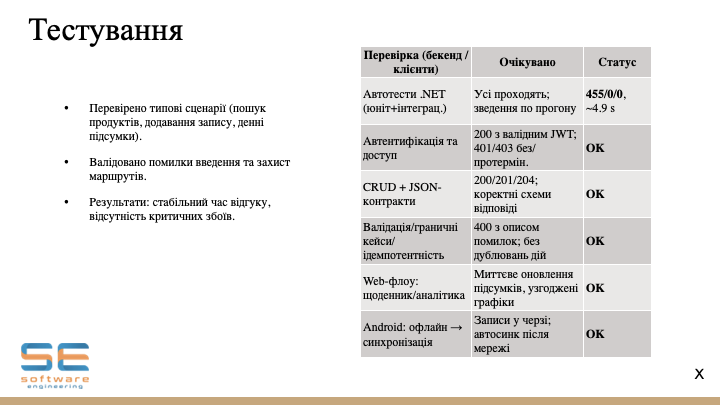


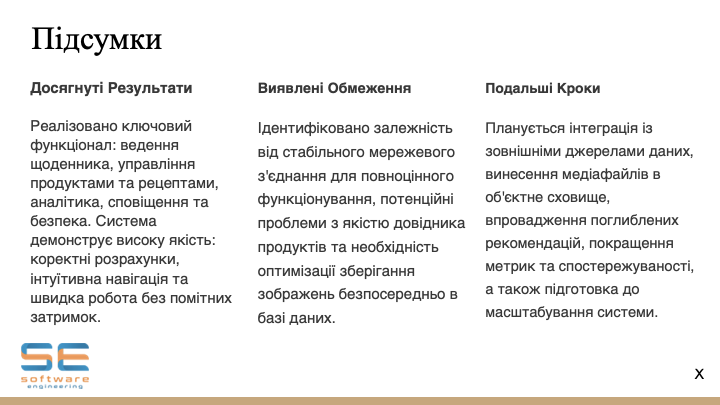












# ДОДАТОК В - ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ

