НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	Завідувач кафедри комп'ютерних наук
_____ Ігор БОЛБОТ	_____ Белла ГОЛУБ
«01» травня 2026 р.	«01» травня 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Бекенд частина аналітичної системи підрахунку калорій при вживанні їжі людиною»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи асистент	_____ (підпис)	**Ярослав ГОРДІЙ**
Консультант к.т.н., доцент	_____ (підпис)	**Белла ГОЛУБ**
Виконав	_____ (підпис)	**Владислав Бойко**

Київ — 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ

«06» 01.2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Бойку Владиславу Сергійовичу

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Бекенд частина аналітичної системи підрахунку калорій при вживанні їжі людиною»

затверджена наказом від **«06» січня 2026 р**. № **46**«С»

Термін подання завершеної роботи на кафедру **«25» травня 2026** р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):

Веб-застосунок для відстеження калорій та харчування. Бекенд-частина реалізована на ASP.NET Core 8 з використанням Entity Framework Core 9 та PostgreSQL (Supabase). Інтеграція із зовнішнім API.

Перелік питань, що підлягають дослідженню:

Аналіз предметної області; проєктування бази даних; розробка REST API; реалізація JWT-аутентифікації; інтеграція із FatSecret API; розробка адміністративного модуля; тестування системи.

Дата видачі завдання **«06» січня 2026** р.

Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Ярослав ГОРДІЙ**
Консультант	_____ (підпис)	**Белла ГОЛУБ**
Завдання взяв до виконання	_____ (підпис)	**Владислав Бойко**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Позначення	Розшифрування
API	Application Programming Interface — програмний інтерфейс застосунку
ASP.NET	Active Server Pages .NET — платформа розробки веб-застосунків Microsoft
БД	База даних
BMR	Basal Metabolic Rate — базальний рівень метаболізму
CORS	Cross-Origin Resource Sharing — спільний доступ до ресурсів між різними джерелами
CRUD	Create, Read, Update, Delete — основні операції з даними
DTO	Data Transfer Object — об'єкт передачі даних
EF Core	Entity Framework Core — ORM-платформа для .NET
ERD	Entity-Relationship Diagram — діаграма сутність-зв'язок
FK	Foreign Key — зовнішній ключ
HTTP	HyperText Transfer Protocol — протокол передачі гіпертексту
HTTPS	HTTP Secure — захищений протокол передачі даних
JWT	JSON Web Token — відкритий стандарт токена автентифікації
JSON	JavaScript Object Notation — формат обміну даними
ORM	Object-Relational Mapping — об'єктно-реляційне відображення
PK	Primary Key — первинний ключ

Позначення	Розшифрування
ПЕОМ	Персональна електронно-обчислювальна машина
PostgreSQL	Post-relational SQL database — об'єктно-реляційна система управління базами даних
REST	Representational State Transfer — стиль архітектури програмного забезпечення
SPA	Single Page Application — односторінковий застосунок
SQL	Structured Query Language — мова структурованих запитів
СУБД	Система управління базою даних
TLS	Transport Layer Security — протокол захисту транспортного рівня
UI	User Interface — інтерфейс користувача
UX	User Experience — досвід користувача
УМП	Умовне позначення
3НФ	Третя нормальна форма

ВСТУП

Актуальність. Проблема контролю ваги та якості харчування сьогодні є однією з найбільш обговорюваних у сфері громадського здоров'я. Малорухливий спосіб життя, їжа з надмірною кількістю калорій та відсутність культури усвідомленого споживання їжі призвели до того, що надмірна вага стала масовим явищем — за даними ВООЗ, із цією проблемою стикається понад мільярд людей по всьому світу. Це у свою чергу створює попит на зручні цифрові інструменти, які допомагають людям краще розуміти свій раціон харчування.

Ринок подібних застосунків активно розвивається, переважно, за кордоном. Більшість популярних додатків зорієнтовані на англомовну аудиторію та американський або європейський продуктовий ринок, тоді як у вітчизняному сегменті якісних аналогів практично не існує. Водночас попит на персоналізований трекінг харчування серед користувачів із України — як серед тих, хто просто прагне схуднення, так і серед професійних спортсменів, спортсменів аматорів, що контролюють споживання білків, жирів, вуглеводів та мікронутрієнтів.

Мета роботи. Метою бакалаврської кваліфікаційної роботи є розробка бекенд-частини веб-застосунку CalorieTracker, системи для аналітичного підрахунку калорій при вживанні їжі. Реалізована система повинна забезпечувати ведення обліку прийомів їжі, автоматичний розрахунок добової норми калорій за формулою Miffin-St Jeore, взаємодію із зовнішньою базою продуктів харчування, а також функціональний адміністративний модуль.

Завдання роботи. Для виконання поставленої цілі необхідно вирішити такі завдання:

- провести аналіз предметної області та огляд існуючих рішень у сфері обліку харчування;
- спроєктувати реляційну базу даних, що відповідає вимогам третьої нормальної форми (3НФ), засобами PostgreSQL;

- розробити RESTful API на базі платформи ASP.NET Core 8 із застосуванням Entity Framework Core 9;
- реалізувати підсистему автентифікації та авторизації з використанням JWT-токенів;
- організувати інтеграцію із FatSecret Platform API для забезпечення пошуку продуктів харчування;
- розробити модуль досягнень користувача для підвищення мотивації до регулярного використання системи;
- реалізувати адміністративний модуль з аудит-логом дій;
- провести функціональне тестування розроблених компонентів та зафіксувати результати.

Методи дослідження. У ході виконання роботи було застосовано: аналіз і синтез при опрацюванні предметної області; методи структурного та об'єктно-орієнтованого проєктування при побудові архітектури системи; принципи нормалізації реляційних баз даних; підхід RESTful-архітектури при розробці API; тестування за методом «чорного ящика».

Структура роботи. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг становить 70 сторінок і включає таблиці, рисунки та фрагменти програмного коду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд проблематики обліку харчування та підрахунку калорій

Те що ми їмо, впливає на те, як ми себе почуваємо, а отже, і на нашу тривалість життя. Якщо людина споживає більше кілорій, ніж вона встигає витратити, за роки це не проходить безслідно — розвивається ожиріння, потім діабет 2 типу, потім проблеми з серцем із-за надмірної ваги. Ризики усього цього, ймовірно, було б зменшено, якби людина почала більше звертати увагу на свій раціон раніше. Підрахунок калорій — це найпростіший і найефективніший спосіб контролювати свою вагу. Логіка, зрештою, достатньо проста; споживання калорій більше норми, витрати менше — набір ваги та в результаті ожиріння, споживання менше норми — схуднення.

Питання у тому, що у кожного є своя норма. Вік, стать, зріст, спосіб життя — все це враховується для визначення того, скільки саме енергії людині потрібно на день.

Для оцінки базального метаболізму (BMR) фахівці переважно використовують формулу Mifflin-St Jeor, яка була розроблена в 1990 році.

Для чоловіків:

$$BMR = 10 \times W + 6.25 \times H - 5 \times A + 5,$$

для жінок:

$$BMR = 10 \times W + 6.25 \times H - 5 \times A - 161,$$

де W — вага в кілограмах, H — зріст у сантиметрах, A — вік у роках.

Отримане значення множиться на коефіцієнт PAL від 1.2 до 1.9 ,тобто ,для тих хто ледве рухається, або для тих, хто щодня витрачає велику кількість калорій. Підраховувати всі ці дані вручну насправді дуже складно. Тому потрібно постійно переглядати таблиці калорій, що досить часто показують дещо різну інформацію, і це швидко набридає.

Ось чому з'явилися додатки для мобільних телефонів та веб додатки в Інтернеті, які спрощують це в значній мірі: вони беруть дані з перевірених баз даних продуктів, а потім миттєво все розраховують. Ринок таких послуг

активно розвивається, але більшість сервісів створені для англомовних споживачів. Українські продукти часто неможливо знайти в їхній базі даних, а деякі засоби вимірювання просто незручні в нашому повсякденному житті.

1.2 Огляд існуючих пропозицій та їх обмежень

Мною було ретельно проаналізовано три з найпоширеніших платформ: MyFitnessPal, Cronometer та FatSecret. В табл. 1.1 наведений порівняльний аналіз цих платформ.

Таблиця 1.1

Порівняльний аналіз існуючих рішень

Критерій	**MyFitnessPal**	**Cronometer**	**FatSecret**	**CalorieTracker**
Безкоштовний доступ	Частково	Частково	Так	Так
Підтримка UA продуктів	Обмежена	Відсутня	Обмежена	Планується
REST API	Платний	Відсутній	Є	Так (відкритий)
Адмін-панель	Відсутня	Відсутня	Відсутня	Є
Система досягнень	Є	Відсутня	Відсутня	Є
Відкритий код	Ні	Ні	Ні	Так
Сканер штрих-коду	Так	Так	Так	Так

Як ми можемо бачити з таблиці, жоден із розглянутих продуктів не закриває всі потреби одразу: десь немає безкоштовного доступу, десь відсутній відкритий API, десь не вистачає адмін-панелі для безпосереднього керування додатком, а про українські продукти майже не йде мова. CalorieTracker якраз і розроблявся, щоб заповнити ці прогалини.

Окремо варто сказати про архітектуру. Більшість комерційних рішень побудовані за монолітним принципом, це рано чи пізно починає заважати: масштабування з високою ймовірністю стане складним, інтегрування - теж. У CalorieTracker клієнтська і серверна частини з самого початку були розділені за схемою SPA + REST API, що відповідає сучасним підходам до розробки.

У роботі використовувались матеріали кількох типів. Наукові публікації з нутрієнтології та цифрового здоров'я дали теоретичну основу — зокрема, матеріали ВООЗ про поширеність ожиріння та рекомендації щодо збалансованого харчування.

Офіційна документація стала головним орієнтиром у технічних рішеннях. Документація ASP.NET Core , Entity Framework Core і PostgreSQL при проєктуванні архітектури та реалізації бекенду. Документація FatSecret Platform API — при розробці модуля інтеграції із зовнішніми даними.

Підручники з проєктування баз даних допомогли обґрунтувати рішення щодо нормалізації схеми, зокрема, класичні роботи про реляційні бази даних і нормальні форми.

Технічні статті зі Stack Overflow і Microsoft DevBlogs використовувались для вирішення різного роду конкретних та специфічних задач у процесі розробки.

1.3 Моделювання предметної області

До написання коду, потрібно чітко розуміти, що саме система має виконувати і з чим конкретно ми маємо працювати. Моделювання предметної області це саме той крок, де на теорії з’являються основні поняття, зв’язки та правила, яким мають відповідати дані.

У випадку системи CalorieTracker ситуація виглядає наступним чином: Є користувачі з їх фізичними параметрами, продукти харчування з конкретною поживною цінністю, прийоми їжі, динаміка ваги і цілі яких користувач бажає досягти. Окремо варто згадати про адміністраторів системи, які працюють з тими самими даними але вже на іншому рівні доступу та іншими можливостями. Усі ці елементи є частинами одного цілого, і завдання моделювання має показати, як вони між собою пов’язані та яким чином взаємодіють у процесі повсякденного використання додатку.

В результаті аналізу було виділено вісім основних сутностей, кожна відповідає за свою ділянку та знаходиться у чітких відносинах з іншими , утворюючи логічно завершену структуру , що в цілому покриває потреби обраної предметної області:

- **Продукт** (Food) - одиниця бази даних продуктів харчування;
- **Прийом їжі** (Meal) - факт споживання їжі у певний момент часу;
- **Позиція прийому їжі** (MealItem) - конкретний продукт у складі прийому з кількістю;
- **Запис ваги** (WeightRecord) - фіксація ваги та зросту в часі;
- **Ціль** (UserGoal) - цільові показники користувача;
- **Досягнення** (Achievement) - нагороди за активність;
- **Журнал аудиту** (AuditLog).

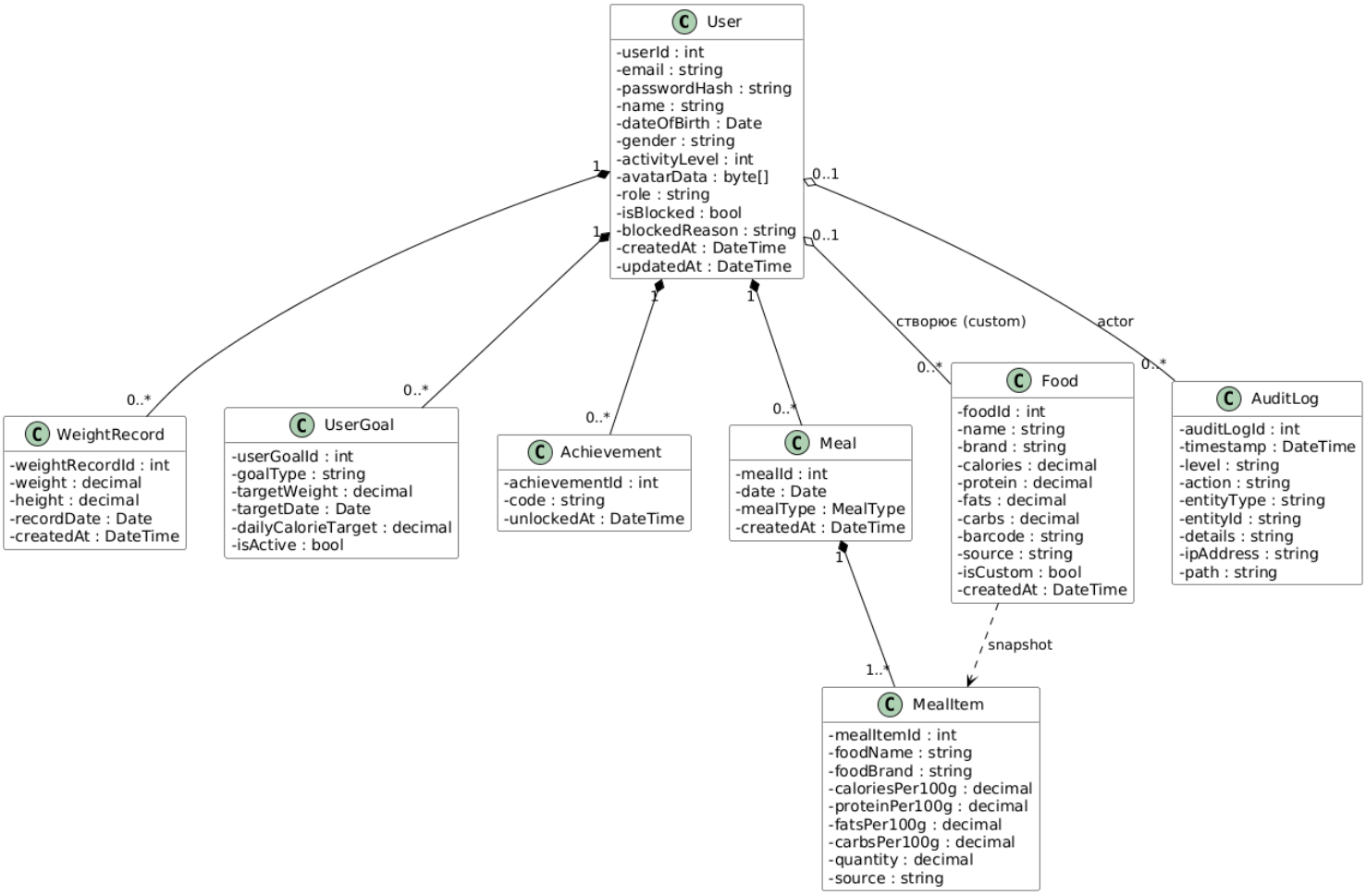


Рис 1.1 — Діаграма класів

Рис 1.2 — Діаграма прецедентів

1.4 Постановка завдання

На основі проведеного аналізу сформульовано технічне завдання на розробку бекенд-частини CalorieTracker.

З точки зору функціоналу система має виконувати:

1. реєстрацію й автентифікацію користувачів через JWT-токени з BCrypt-хешуванням паролів;
2. збереження профілів користувачів з антропометричними даними — стать, дата народження, рівень активності, вага, зріст;
3. автоматичний розрахунок добової норми калорій за формулою Miffin-St Jeor;

4. фіксування прийоми їжі, такі як сніданок, обід, вечерю, перекус — із зазначенням ваги кожного продукту;
5. пошук продуктів за назвою або штрих-кодом через FatSecret API;
6. надання користувачу змоги вести власну базу продуктів;
7. демонстрація щоденної й накопиченої статистики по всім мікро та макро нутрієнтам;
8. відстеження динаміки ваги з хронологією записів;
9. нараховування досягнень — 15 видів, які розблоковуються автоматично по мірі досягнення користувачем певних цілей;
10. надання адміністратору інструментів для управління користувачами, модерації продуктів, перегляду статистики та журналу аудиту.

Щодо нефункціональних вимог:

1. стек: ASP.NET Core 8, Entity Framework Core 9, PostgreSQL на базі Supabase;
2. автентифікація: JWT з HMAC-SHA256, токен діє 24 години;
3. схема бази даних відповідає третій нормальній формі;
4. API будується за принципами REST і документується через Swagger/OpenAPI;
5. час відповіді на звичайні запити — не більше 500 мс;
6. доступ захищений через JWT у парі з HTTPS.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Логічна модель даних

Логічна модель демонструє які атрибути має кожна сутність, якого вони типу, де первинні ключі, де зовнішні ключі і які є обмеження існують. Все це на даний момент демонструється без прив'язки до конкретної бази даних.

Головна вимога, на основі якої була визначена більшість рішень, це відповідність схеми до третьої нормальної форми. Тобто, кожне поле в таблиці повинно залежати виключно від ключа.

В табл 2.1 представлено логічну модель таблиці users:

Таблиця 2.1

Логічна модель таблиці Users

Атрибут	Тип даних	Опис	Обмеження
UserId	SERIAL (PK)	Первинний ключ	NOT NULL, UNIQUE
Email	VARCHAR(256)	Електронна пошта	NOT NULL, UNIQUE
PasswordHash	TEXT	BCrypt-хеш пароля	NOT NULL
Name	VARCHAR(100)	Ім'я користувача	NOT NULL
DateOfBirth	DATE	Дата народження	NOT NULL
Gender	VARCHAR(10)	Стать (Male/Female)	NOT NULL
ActivityLevel	INTEGER	Рівень активності (1–5)	NOT NULL

Атрибут	Тип даних	Опис	Обмеження
AvatarData	BYTEA	Аватар у бінарному форматі	NULL
AvatarContentType	VARCHAR(50)	MIME-тип аватара	NULL
AvatarUpdatedAt	TIMESTAMPTZ	Дата оновлення аватара	NULL
Role	VARCHAR(20)	Роль (User/Admin)	NOT NULL, DEFAULT 'User'
IsBlocked	BOOLEAN	Статус блокування	NOT NULL, DEFAULT false
BlockedAt	TIMESTAMPTZ	Дата блокування	NULL
BlockedReason	TEXT	Причина блокування	NULL
CreatedAt	TIMESTAMPTZ	Дата реєстрації	NOT NULL
UpdatedAt	TIMESTAMPTZ	Дата останнього оновлення	NOT NULL

В табл 2.2 представлено логічну модель таблиці Foods:

Таблиця 2.2

Логічна модель таблиці Foods

Атрибут	Тип даних	Опис	Обмеження
FoodId	SERIAL (PK)	Первинний ключ	NOT NULL, UNIQUE
Name	VARCHAR(200)	Назва продукту	NOT NULL
Brand	VARCHAR(100)	Виробник	NULL

Атрибут	Тип даних	Опис	Обмеження
Calories	DECIMAL	Калорії на 100г	NOT NULL
Protein	DECIMAL	Білки на 100г	NOT NULL
Fats	DECIMAL	Жири на 100г	NOT NULL
Carbs	DECIMAL	Вуглеводи на 100г	NOT NULL
Fiber	DECIMAL	Клітковина на 100г	NULL
Sugar	DECIMAL	Цукри на 100г	NULL
Sodium	DECIMAL	Натрій на 100г (мг)	NULL
Category	VARCHAR(50)	Категорія продукту	NULL
Barcode	VARCHAR(30)	Штрих-код (GTIN)	NULL
ExternalId	VARCHAR(100)	ID у зовнішній системі	NULL
Source	VARCHAR(20)	Джерело (FatSecret/Custom)	NULL
IsCustom	BOOLEAN	Кастомний продукт	NOT NULL
CreatedBy	INTEGER (FK→Users)	Автор продукту	NULL, SET NULL on delete
CreatedAt	TIMESTAMPTZ	Дата додавання	NOT NULL

В табл 2.3 представлено логічну модель таблиці Meals/MealItems:

Таблиця 2.3

Логічна модель таблиці Meals та MealItems

Атрибут	Тип даних	Опис	Обмеження
— Meals —			
MealId	SERIAL (PK)	Первинний ключ	NOT NULL, UNIQUE
UserId	INTEGER (FK→Users)	Власник запису	NOT NULL, CASCADE delete
Date	DATE	Дата прийому їжі	NOT NULL
MealType	INTEGER	Тип (1-Сніданок, 2-Обід, 3-Вечеря, 4-Перекус)	NOT NULL
CreatedAt	TIMESTAMPTZ	Дата створення	NOT NULL
— MealItems —			
MealItemId	SERIAL (PK)	Первинний ключ	NOT NULL, UNIQUE
MealId	INTEGER (FK→Meals)	Прийом їжі	NOT NULL, CASCADE delete
FoodName	VARCHAR(200)	Назва продукту (копія)	NOT NULL
FoodBrand	VARCHAR(100)	Бренд продукту (копія)	NULL
CaloriesPer100g	DECIMAL	Калорії/100г (копія)	NOT NULL
ProteinPer100g	DECIMAL	Білки/100г (копія)	NOT NULL
FatsPer100g	DECIMAL	Жири/100г (копія)	NOT NULL

Атрибут	Тип даних	Опис	Обмеження
CarbsPer100g	DECIMAL	Вуглеводи/100г (копія)	NOT NULL
FiberPer100g	DECIMAL	Клітковина/100г (копія)	NULL
ExternalId	VARCHAR(100)	ID у зовнішній системі	NULL
Source	VARCHAR(20)	Джерело продукту	NULL
Quantity	DECIMAL	Кількість (грами)	NOT NULL

2.2 Вибір системи управління базою даних

Вибір СУБД є дійсно важливим рішенням, яке визначить характеристики продуктивності, а також масштабованості та реальної надійності системи. Для проєкту CalorieTracker було розглянуто кілька реляційних СУБД: PostgreSQL, MySQL та Microsoft SQL Server.

Після порівняльного аналізу, виходячи з результатів, для реалізації проєкту CalorieTracker було обрано PostgreSQL, з розгортанням на платформі Supabase. Вибір було обґрунтовано тим, що PostgreSQL є однією з найбільш функціонально розвинених СУБД з відкритим кодом, та при цьому відзначається суворим дотриманням стандартів SQL та підтримкою розширених типів даних.

Інтеграція PostgreSQL з платформою .NET виконується через провайдер Npgsql для Entity Framework Core, що в свою чергу надає повну підтримку специфічних типів PostgreSQL та генерацію ефективних SQL-запитів.

До остаточного вибору СУБД проведено порівняльний аналіз трьох найпоширеніших реляційних систем, які могли б покрити вимоги CalorieTracker: PostgreSQL, MySQL та Microsoft SQL Server. Критерії для порівняння обиралися з огляду на специфіку проєкту. Результати порівняння наведено в таблиці 2.4.

Таблиця 2.4

Порівняльний аналіз СУБД для проєкту CalorieTracker

Критерій	PostgreSQL	MySQL	MS SQL Server
Ліцензія	Безкоштовна (PostgreSQL License)	Безкоштовна (GPL) / комерційна	Комерційна (є Express-версія)
Дотримання стандартів SQL	Високе	Середнє	Високе
Підтримка JSON / JSONB	Нативна, з індексацією	Базова	Обмежена
Підтримка часових зон (TIMESTAMPTZ)	Так	Часткова	Так
Хмарні платформи з безкоштовним планом	Supabase, Neon, Render	PlanetScale (обмежено)	Azure SQL (обмежено)
Інтеграція з .NET / EF Core	Через Npgsql (офіційний)	Через Pomelo	Нативна, від Microsoft

Критерій	PostgreSQL	MySQL	MS SQL Server
Часткові індекси (Partial Index)	Так	Ні	Так (Filtered Index)

2.3 Фізична модель та створення бази даних

Фізична модель — є наступним логічним кроком після проєктування на рівні сутностей. Ця модель показує, яким чином абстрактні поняття перетворюються на конкретні таблиці PostgreSQL з визначеними типами даних, обмеженнями цілісності та індексами. Простіше кажучи, це рівень, на якому теорія стає схемою, з якою вже може працювати реальна СУБД.

Для створення фізичної схеми використовується інструмент **Entity Framework Core 9** із підходом Code First. Такий підхід передбачає, що схема бази даних не буде писатись вручну у SQL, а буде реалізована автоматично на основі C#-класів моделей і конфігурації через Fluent API.

Будь-яка зміна моделі (додавання поля, перейменування таблиці, зміна типу) буде зафіксована в окремій **міграції** — файлі з двома методами, Up() для застосування змін і Down() для їхньої відміни. У проєкті використано стратегію auto-migrate: метод db.Database.Migrate() викликається на старті застосунку у Program.cs і автоматично застосовує всі ще не використані міграції. Це гарантує, що схема в базі завжди буде синхронізована з кодом, незалежно від середовища розгортання.

Правила каскадного видалення налаштовані з урахуванням бізнес-логіки. Видалення користувача стане причиною видалення всіх його прийомів їжі, записів ваги, цілей і досягнень — оскільки ці записи поза контекстом конкретного користувача не несуть жодної цінності. Натомість для продуктів (Foods) і журналу аудиту (AuditLogs) застосовано стратегію SET NULL: при видаленні автора відповідне посилання просто обнуляється, але сам запис залишається. У випадку з продуктами це дозволяє іншим користувачам

продовжувати ними користуватися, а у випадку з журналом — забезпечує збереження історії дій навіть після видалення облікового запису.

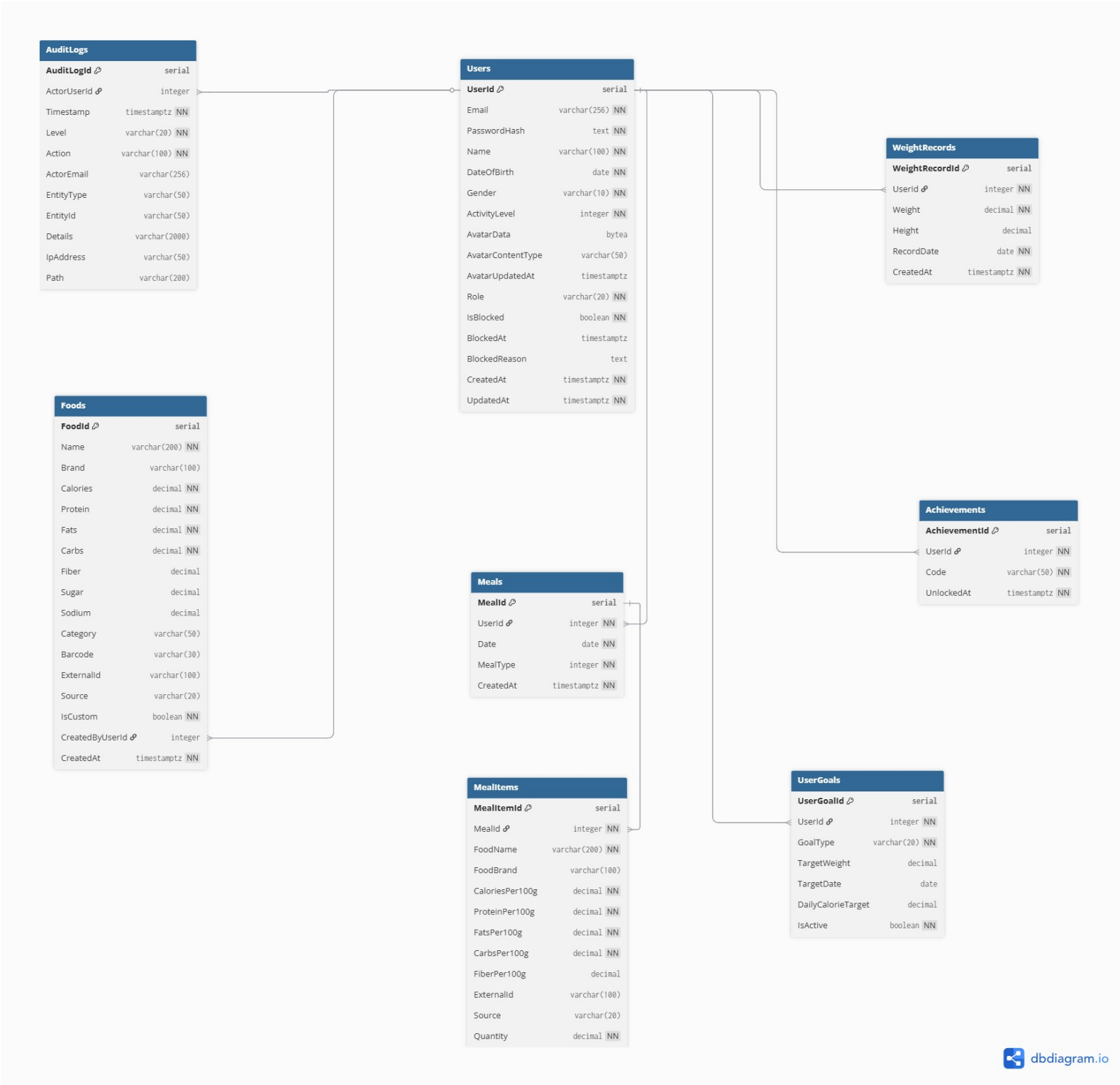


Рис 2.1 — Фізична схема бази даних CalorieTracker

DDL-скрипт створення таблиць бази даних наведено у Додатку А.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Архітектура програмного забезпечення CalorieTracker створена за принципом чіткого розподілення відповідальності між рівнями системи (Separation of Concerns). Рішення виконане у вигляді монорепозиторію, що містить два окремих проєкти: бекенд-частину (CalorieTracker.Server) та фронтенд-частину (calorietracker.client). Моя робота зосереджена на бекенд-частині як на основному предметі дослідження.

CalorieTracker.Server побудований на основі шарової архітектури з наступними рівнями:

- Рівень представлення (Controllers) — обробка HTTP-запитів, валідація вхідних даних, маршрутизація;
- Рівень бізнес-логіки (Services) — реалізація бізнес-правил та алгоритмів;
- Рівень доступу до даних (Data/DbContext) — взаємодія з PostgreSQL через EF Core;
- Рівень моделей (Models, DTOs) — опис доменних сутностей та об'єктів передачі даних.

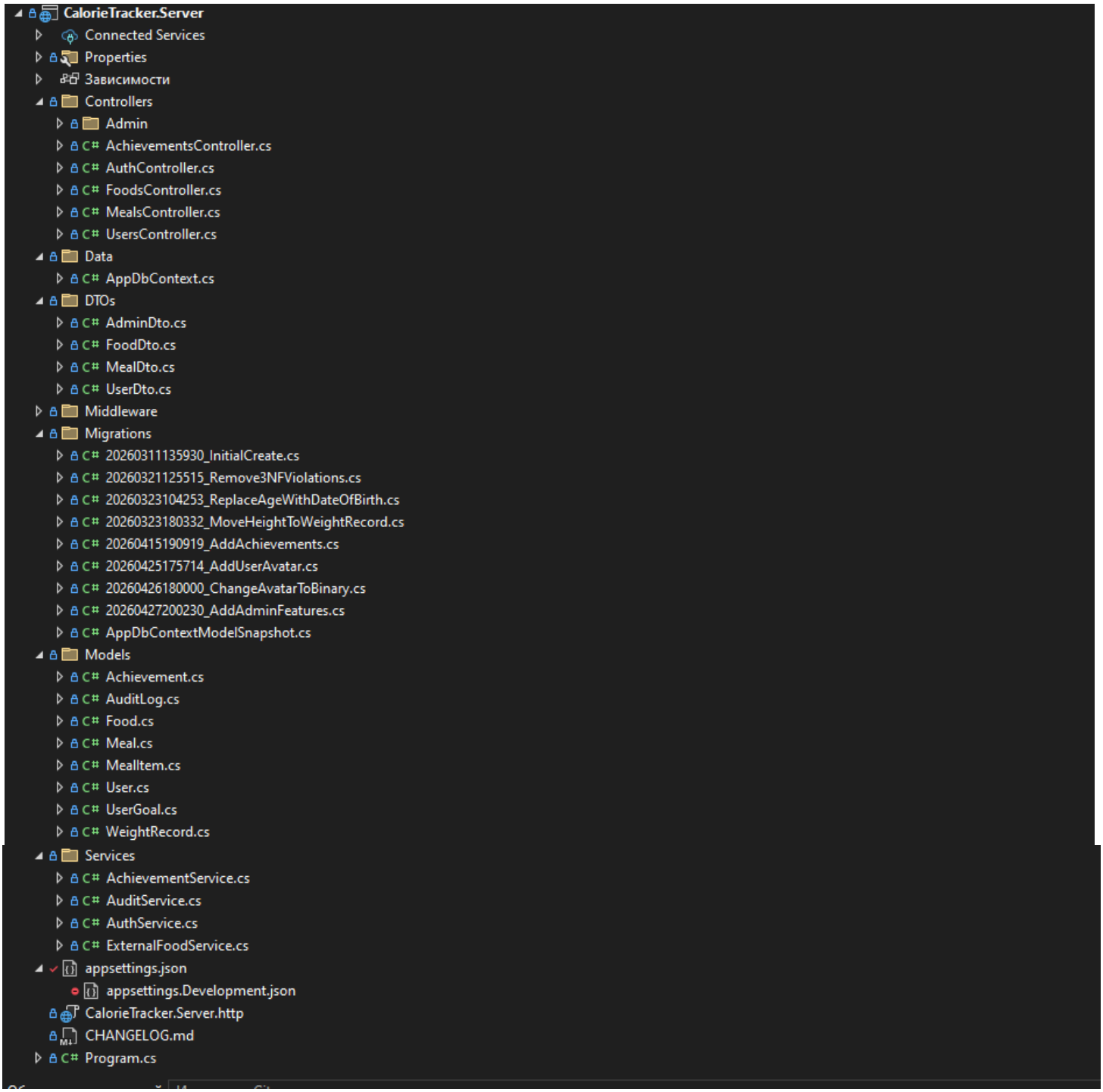


Рис 3.1 — Організаційна структура бекенд-проєкту

Структура папок бекенд-проєкту CalorieTracker.Server організована за функціональним принципом: код розділений за відповідальностями так, щоб

кожна папка містила файли одного призначення. Це значно спрощує навігацію по проєкту та робить зрозумілим, куди саме треба додати новий функціонал.

Папка **«Controllers»** містить контролери REST API, які обробляють вхідні HTTP-запити: «**AuthController**» відповідає за реєстрацію та авторизацію, «**UsersController**» відповідає за роботу з профілем користувача, «**FoodsController**» відповідає за продукти харчування, «**MealsController**» за прийоми їжі, а **«AchievementsController»** відповідає за систему досягнень.

Адміністративні контролери додано в окрему підпапку «**Controllers/Admin/**», що відокремлює функціонал який має бути доступний лише користувачам з роллю Admin, від решти API.

Доменні моделі розміщені у папці Models. До неї входять класи-сутності User, Food, Meal, MealItem, UserGoal, WeightRecord, Achievement та AuditLog, які відповідають таблицям бази даних і використовуються Entity Framework Core для побудови схеми через підхід Code First. Окремо, у папці DTOs, зібрані об'єкти передачі даних, задача яких полягає у тому, щоб приховати внутрішню структуру моделей від клієнта та надавати лише поля, які потрібні у конкретному API-запиті.

Доступом до бази даних керує клас AppDbContext, розміщений у папці Data. У ньому оголошено вісім властивостей типу «DbSet», кожна з яких відповідає за окрему таблицю, а також налаштовані індекси, унікальні обмеження та правила каскадного видалення. В свою чергу Папка Migrations містить історію змін схеми бази даних у вигляді EF Core міграцій, де кожна з них фіксує конкретний крок у змінах проєкту, від початкового створення таблиць до додавання адміністративних можливостей.

Шар бізнес-логіки винесено у папку Services. У ній зосереджено все, що не є простими CRUD-операціями. AuthService хешує паролі алгоритмом BCrypt та генерує JWT-токени, ExternalFoodService працює з зовнішнім FatSecret API через протокол OAuth 2.0, AchievementService займається перевіркою дотримання умов для отримання нагород користувачем, а AuditService веде журнал дій адміністраторів. Такий підхід дозволяє тримати контролери тонкими , тобто вони лише приймають запит, делегують роботу сервісу і повертають результат.

Папка Middleware містить клас ErrorLoggingMiddleware, який перехоплює необроблені винятки на рівні всього застосунку та фіксує їх у журналі аудиту разом з контекстом запиту. Стартова конфігурація проєкту знаходиться у файлі Program.cs, в ньому реєструються залежності в контейнері Dependency Injection, налаштовуються JWT-автентифікація, CORS та pipeline middleware. Параметри підключення до бази даних, секретні ключі для FatSecret та налаштування JWT винесені у файл appsettings.json, що дозволяє змінювати конфігурацію без компіляції коду заново.

Для наочного представлення зв'язків між шарами і зовнішніми залежностями на рисунку 3.2 наведено діаграму компонентів бекенд-частини.

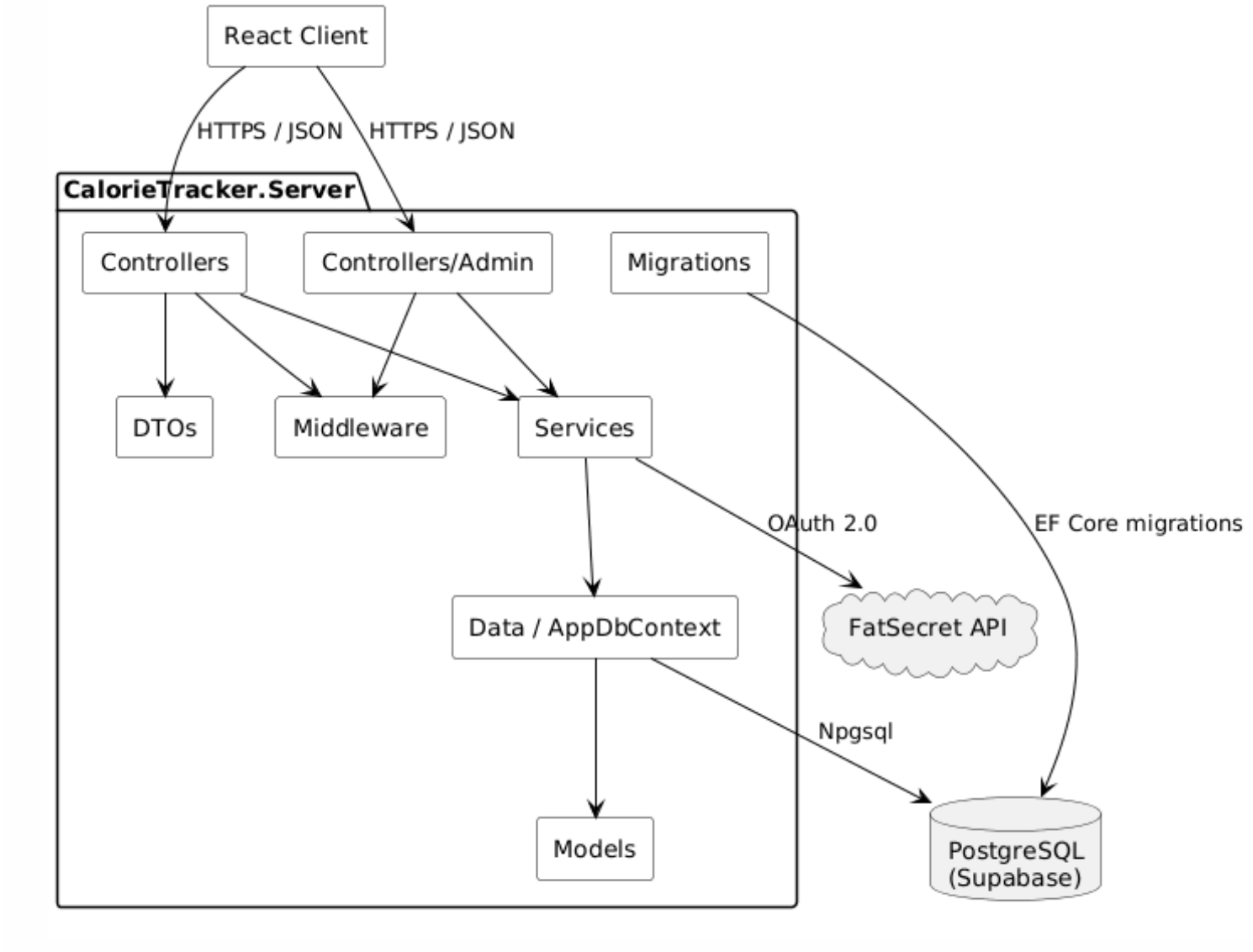


Рис 3.2 — Організаційна структура бекенд-проєкту

3.2 Вибір інструментарію для розробки бекенд-частини

Оскільки вибір технологічного стеку напряму впливає на довгостроковий потенціал розвитку системи, а також на її продуктивність та простоту підтримки. Для бекенд-розробки CalorieTracker я обрав платформу .NET 8 та ASP.NET Core — з ряду вагомих причин.

ASP.NET Core 8 — це сучасний кросплатформний фреймворк для розробки веб-застосунків і API від Microsoft, що є продовженням платформи .NET. Серед основних переваг: висока продуктивність (ASP.NET Core стабільно потрапляє до топу бенчмарків TechEmpower Framework Benchmarks), нативна підтримка внесення залежностей (DI), вбудована підтримка JWT-автентифікації, гнучкий middleware pipeline, а також кросплатформеність (.NET 8 підтримує Windows, Linux, macOS).

Entity Framework Core 9 — ORM (Object-Relational Mapper) для .NET, що забезпечує взаємодію з реляційною базою даних через об'єктно-орієнтований інтерфейс. EF Core дозволяє описувати схему бази даних у вигляді класів C#, автоматично генерувати SQL-запити та керувати міграціями схеми. А провайдер Npgsql забезпечує повну сумісність з PostgreSQL.

Supabase — це хмарна платформа на базі PostgreSQL, що надає готову інфраструктуру з PostgreSQL-сервером, пулінгом з'єднань, дашбордом для управління даними та SQL-редактором. Безкоштовний план Supabase включає достатній обсяг ресурсів для розробки та тестування застосунку.

FatSecret Platform API — зовнішній API для доступу до бази даних продуктів харчування. Обраний через наявність безкоштовного тарифного плану (Premier Free з US data set), підтримку пошуку за штрих-кодом (GTIN-13) та підтримку параметра мови (language=uk).

У таблиці 3.1 представлений технологічний стек бекенд-частини:

Таблиця 3.1

Технологічний стек бекенд-частини

Компонент	**Технологія / Бібліотека**	**Версія / Призначення**
Платформа	ASP.NET Core	.NET 8 — Web API
ORM	Entity Framework Core	v9 + Npgsql провайдер
БД	PostgreSQL / Supabase	Хмарний PostgreSQL, пулінг PgBouncer
Автентифікація	JWT Bearer	Microsoft.AspNetCore.Authentication.JwtBearer
Хешування паролів	BCrypt.Net-Next	BCrypt алгоритм хешування
Документація API	Swagger / OpenAPI	Swashbuckle.AspNetCore — /swagger
Зовнішній API	FatSecret Platform	OAuth 2.0 client_credentials, пошук продуктів
Кешування	IMemoryCache	Кеш токенів та результатів пошуку FatSecret
HTTP-клієнт	HttpClient / IHttpClientFactory	Запити до FatSecret API

3.3 Проєктування та реалізація REST API

REST API (Representational State Transfer Application Programming Interface) це основний інтерфейс взаємодії з клієнтською та серверною частинами. API використовує принципів RESTful архітектури, а саме, використання HTTP-методів відповідно до їх семантики (GET для читання, POST для створення, PUT для оновлення, DELETE для видалення), адресація ресурсів через URI, повернення відповідей у форматі JSON.

API системи CalorieTracker організовано у шість функціональних груп: У таблиці 3.2 наведено ендпоінти **REST API**

Таблиця 3.2

Зведена таблиця ендпоінтів REST API

Метод	URI	Опис	Доступ
POST	/api/auth/register	Реєстрація нового користувача	Public
POST	/api/auth/login	Автентифікація, отримання JWT	Public
GET	/api/users/profile	Профіль поточного користувача	Auth
PUT	/api/users/profile	Оновлення профілю	Auth
PUT	/api/users/avatar	Завантаження аватара (multipart)	Auth
POST	/api/users/weight	Додавання запису ваги/зросту	Auth
GET	/api/users/weight-history	Хронологія ваги (параметр days)	Auth
GET	/api/users/statistics	Статистика споживання нутрієнтів	Auth
GET	/api/foods	Список кастомних	Auth

Метод	URI	Опис	Доступ
		продуктів (пагінація)	
POST	/api/foods	Створення кастомного продукту	Auth
GET	/api/foods/external/search	Пошук продукту (локал + FatSecret)	Auth
GET	/api/foods/external/barcode/{code}	Пошук за штрих-кодом	Auth
GET	/api/meals/daily/{date}	Прийоми їжі за дату + зведення	Auth
POST	/api/meals	Створення прийому їжі з продуктами	Auth
POST	/api/meals/{id}/items	Додавання продукту до прийому	Auth
PUT	/api/meals/items/{itemId}/quantity	Зміна кількості (грами)	Auth
DELETE	/api/meals/items/{itemId}	Видалення позиції прийому	Auth
GET	/api/achievements	Всі досягнення + статус розблокування	Auth
POST	/api/achievements/check	Перевірка і розблокування нових	Auth
GET	/api/admin/users	Список всіх користувачів (з фільтрами)	Admin
POST	/api/admin/users/{id}/block	Блокування користувача	Admin
GET	/api/admin/stats	Статистика системи	Admin
GET	/api/admin/logs	Журнал аудиту (пагінація + фільтри)	Admin

Щоб забезпечити як омога точнішу відповідність контракту між клієнтом та сервером, у системі застосовано автоматичну генерацію OpenAPI-специфікації за допомогою бібліотеки Swashbuckle.AspNetCore. На заміну ручного ведення документації, яка в умовах активної розробки активно застаріває, схема формується динамічно під час кожної збірки та запуску додатка. Механізм генерування спирається на можливості рефлексії платформи .NET.

Процес ініціалізується на етапі конфігурації сервісів: система докладно аналізує код контролерів, сигнатури публічних методів, типи повертаних значень (наприклад, ActionResult<T>) та маршрутні атрибути (на зразок [HttpGet] чи [Route]). Також налаштовано парсинг XML-коментарів із вихідного коду, що дозволяє підтягувати текстові описи безпосередньо з кодової бази і транслювати їх у зрозумілий стандартизований формат. Таким чином, бекенд-код виступає єдиним джерелом істини.

Інтерактивна веб-панель стає доступна за маршрутом /swagger відразу після запуску локального сервера, при цьому вона не просто статично відображає перелік доступних мережевих викликів, а є повноцінним середовищем для пісочниці (sandbox-тестування). Оскільки переважна більшість API-ресурсів CalorieTracker захищені від анонімного доступу, базову конфігурацію Swagger довелося суттєво розширити. Використовуючи методи AddSecurityDefinition та AddSecurityRequirement було інтегровано підтримку авторизації через JWT. Тепер розробнику чи тестувальнику достатньо один раз вставити дійсний Bearer-токен у спеціальне вікно інтерфейсу. Після цього Swagger «під капотом» автоматично буде додавати HTTP-заголовок Authorization до всіх наступних запитів. Це загально знімає потребу розгортати сторонні інструменти на кшталт Postman для швидкої перевірки логіки створення прийомів їжі чи тестування адмін-панелі.

Не менш важливою частиною автоматизації є трансляція моделей. Генератор обходить розроблені класи обміну даними (Data Transfer Objects), формуючи точні JSON-схеми для тіла запитів (request body) та структур відповідей (response payload). Система розпізнає не лише базові типи даних, але й обмеження, задані через механізм Data Annotations у мові C#. Наприклад, якщо властивість у MealItemCreateDto позначена атрибутами [Required] чи [Range], Swagger витягне ці правила і відобразить їх у документації. Відтак розробник клієнтської частини одразу бачить, які поля є обов'язковими, якою має бути максимальна довжина рядка чи допустима кількість грамів для продукту. Подібний рівень прозорості радикально зменшує кількість помилок валідації на етапі зшивання бекенду з фронтендом, адже контракти відомі ще до моменту відправки першого тестового запиту.

Для більш докладної інформації про даному ендпоінті передбачена окрема картка, яка з'являється в результаті натискання назви ендпоінта. У ній наведено опис призначення ендпоінту, схема вхідного JSON об'єкта зі списком типів його полів, перелік можливих кодів відповідей для кожного з них, а також можливість використання інтерактивного інструменту Try it out.

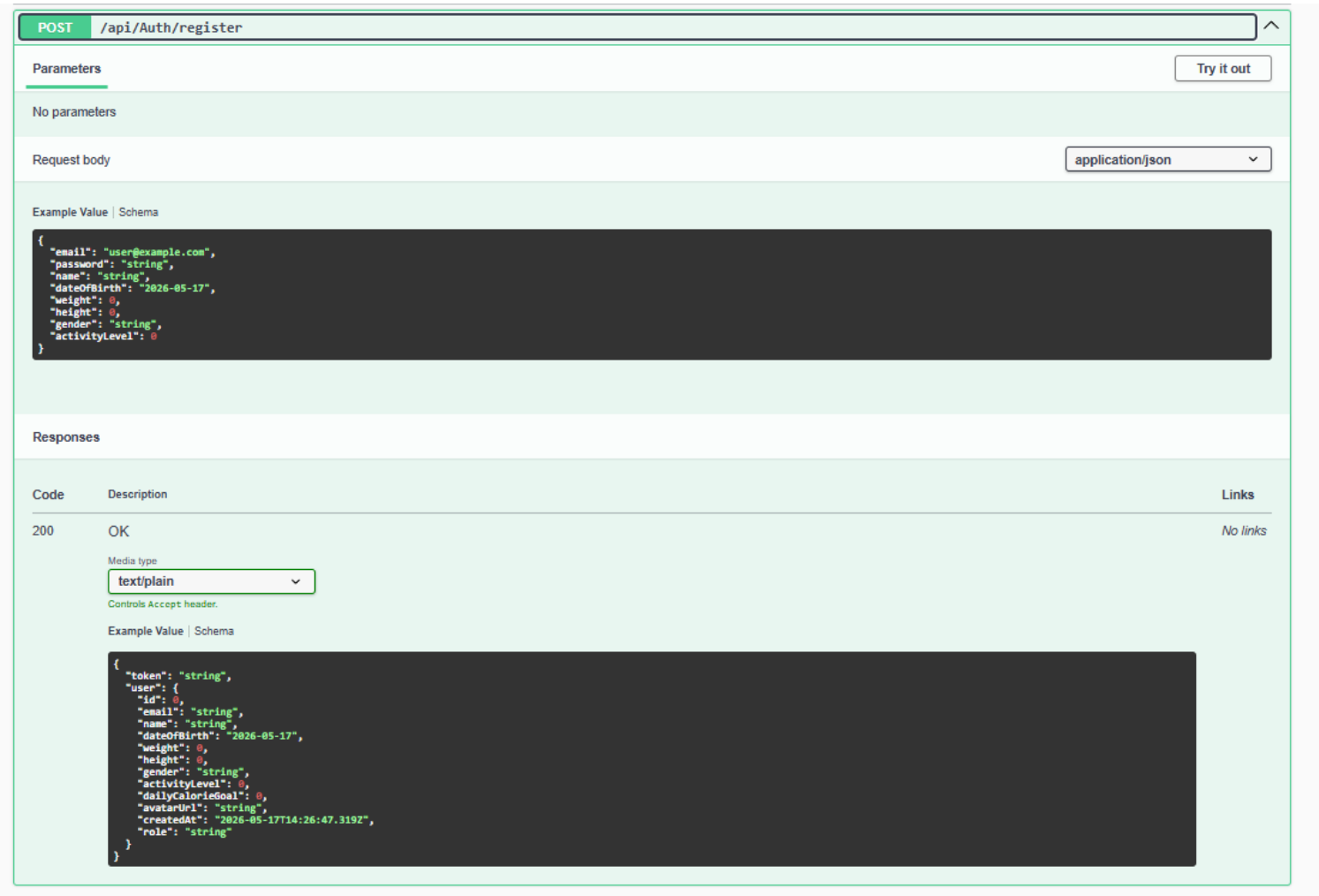


Рис 3.3 — Приклад документації ендпоінту в Swagger UI

Для розуміння того, як саме обробляється типовий запит, на рисунку 3.3(у додатку «Ж») наведено діаграму послідовності для сценарію додавання нового прийому їжі — одного з найчастіших викликів у системі. Діаграма показує проходження запиту через усі шари: від клієнтського React-компонента до запису в PostgreSQL.

3.4 Реалізація бізнес-логіки

Бізнес-логіка CalorieTracker продемонстрована у класах сервісного рівня — вони з себе представляють проміжний шар між контролерами і базою даних. Контролери самі по собі не приймають рішень: вони лише отримують HTTP-запит, передають його у відповідний сервіс і повертають результат. Уся "інтелектуальна" робота, така як: перевірки, розрахунки, інтеграції, формування відповідей , відбувається саме у сервісах. Такий підхід часто називають "thin controllers, fat services", і він дозволяє тримати код у порядку: одна відповідальність на один клас.

Кожен сервіс має чітко визначену зону відповідальності. AuthService відповідає тільки за автентифікацією — нічого, крім хешування паролів і генерації JWT, він не робить. ExternalFoodService являє собою окремий модуль для роботи з FatSecret API, і він нічого "не знає" ні про користувачів, ні про прийоми їжі.

AchievementService інкапсулює логіку розблокування досягнень. AuditService — фіксує дії в журналі. Така ізоляція дозволяє змінювати реалізацію одного сервісу, не зачіпаючи інших.

Взаємодія з базою даних відбувається не напряму, а через AppDbContext. Клас Entity Framework Core, що виступає як посередник між кодом і PostgreSQL. Сервіси отримують DbContext через механізм Dependency Injection, налаштований у Program.cs, і використовують його для виконання LINQ-запитів, які EF Core потім транслює в SQL.

Завдяки цьому в коді бізнес-логіки немає жодного рядка SQL і всі звернення до БД виглядають як звичайна робота з C#-колекціями, що значно підвищує читабельність і знижує ризик помилок.

Нижче розглянуто реалізацію чотирьох ключових сервісів системи.

3.4.1 Сервіс автентифікації (AuthService) AuthService відповідає конкретно за три операції: реєстрацію нових користувачів, перевірку облікових даних під час входу та генерацію JWT-токенів. Це єдина точка входу для логіки автентифікації — інші сервіси з паролями і токенами не працюють.

При реєстрації пароль ніколи не буде збереженим у відкритому вигляді. Він проходить через алгоритм BCrypt з коефіцієнтом складності 11, що означає 2^{11} ітерацій хешування. Це навмисно повільна операція, і саме така повільність робить брутфорс-атаки безкорисними. У базу записується лише підсумковий хеш разом із автоматично згенерованою «сіллю», тобто з випадковим рядком символів, що буде в результаті доданим до паролю на його початку перед хешуванням.

При автентифікації BCrypt-хеш звіряється з тим, що зберігається у БД. Генерація JWT-токена виконується бібліотекою System.IdentityModel.Tokens.Jwt з алгоритмом HMAC-SHA256, секретний ключ для підпису вноситься в appsettings.json. Термін його дії становить 24 години. До токена включаються клейми: NameIdentifier, Email, Name, Role та кастомний userId.

```
public string GenerateJwtToken(User user)
{
    var jwtSettings = _configuration.GetSection("JwtSettings");
    var secretKey = jwtSettings["SecretKey"];
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey!));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);


    var claims = new List<Claim>
    {
        new(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new(ClaimTypes.Email, user.Email),
        new(ClaimTypes.Name, user.Name),
        new(ClaimTypes.Role, string.IsNullOrWhiteSpace(user.Role) ? "User" : user.Role),
        new("userId", user.Id.ToString())
    };

    var token = new JwtSecurityToken(
        issuer: jwtSettings["Issuer"],
        audience: jwtSettings["Audience"],
        claims: claims,
        expires: DateTime.UtcNow.AddHours(double.Parse(jwtSettings["ExpirationHours"]!)),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Рис 3.4 — Реалізація генерації JWT-токена в AuthService

3.4.2 Сервіс зовнішніх продуктів (ExternalFoodService). ExternalFoodService відповідає за надійну інтеграцію системи із зовнішнім провайдером який надає данні про продукти харчування — FatSecret Platform API. Оскільки кожен мережевий виклик до стороннього ресурсу збільшує загальний час відповіді логіки на запити, архітектура цього компонента проєктувалася таким чином, аби кеш використовувався ефективніше та мінімізація не отримувала зайвих HTTP-запитів.

Авторизація взаємодії побудована за стандартом OAuth 2.0 з використанням потоку client_credentials. Бекенд виступає в ролі довіреного клієнта, передаючи свої ідентифікатор та секрет для отримання Bearer-токена.

Щоб уникнути виснаження лімітів зовнішнього API та зменшити затримки, отриманий токен зберігається в оперативній пам'яті сервера через інтерфейс IMemoryCache. Базовий час його життя становить 24 години, проте у код свідомо закладено захисний відступ у 5 хвилин. Це запобігає ситуаціям стану гонитви (race condition), коли термін дії токена міг би закінчитися рівно в ту мілісекунду, коли запит перебуває в дорозі до серверів FatSecret.

Для забезпечення відмовостійкості сервіс має механізм самовідновлення. Якщо зовнішнє API з якихось причин повертає статус 401 Unauthorized (наприклад, через примусове відкликання токенів на стороні провайдера), ExternalFoodService не віддає цю помилку клієнтському додатку. Замість цього він перехоплює виняток, примусово видаляє невалідний токен із локального кешу та виконує одноразовий повторний запит (retry).

Окрім авторизаційних даних, сервіс інтенсивно кешує самі корисні навантаження (payloads). Текстові результати пошуку за ключовими словами зберігаються на 10 хвилин. Натомість відповіді на запити за штрих-кодом (GTIN/EAN) кешуються значно довше — на 1 годину. Такий поділ обґрунтований тим, що прив'язка складу продукту до конкретного штрих-коду є статичною характеристикою, яка майже ніколи не змінюється в реальному часі.

Також окрема увага була приділена парсингу та уніфікації сирих відповідей. FatSecret часто віддає харчову цінність у непередбачуваних форматах: порціях, штуках чи унціях. Для того щоб клієнтський додаток не витрачав ресурси на складні конвертації, на стороні бекенду реалізовано метод PickBestGramServing. Алгоритм перебирає масив доступних порцій конкретного продукту. Він пріоритетно шукає еталонну порцію у 100 грамів. Якщо така відсутня, метод вибирає найближче грамове значення і за допомогою пропорцій математично нормалізує його калорійність та макронутрієнти до 100 грамів. Як наслідок, база даних CalorieTracker та

клієнтський інтерфейс завжди оперують чистими, стандартизованими показниками.

3.4.3 Сервіс досягнень (AchievementService) У додатку, реалізована система мотивації ,яка у свою чергу побудована на наборі з п'ятнадцяти досягнень, які користувач отримує автоматично у міру користування додатком і виконання певних дій пов’язаних з функціоналом програми. Досягнення поділяються на шість тематичних груп, кожна з яких відповідає окремому аспекту здорових звичок, наприклад: реєстрація прийомів їжі, активність користування додатком, стабільність та послідовність ведення щоденника, дотримання добової норми кбжв, контроль ваги та поповнення власної бази продуктів.

Перша група відстежує загальну кількість доданих прийомів їжі. Початковим досягненням є — «Перший крок» (FIRST_MEAL), цей крок розблоковується одразу після того, як користувач додає свій перший прийом їжі, це досягнення покликане заохочувати новачків зробити перші записи під час користування системою. Далі йдуть три послідовних рівні: «Початківець» (MEALS_10) який можна отримати після додавання десяти прийомів їжі, «Досвідчений» (MEALS_50) за п'ятдесят, та «Ветеран» (MEALS_100) за сотню зафіксованих записів про прийом їжі.

Друга група винагороджує загальну активність користування додатком і вимірюється кількістю активних календарних днів, у які користувач хоч раз записав прийом їжі. Досягнення «Тиждень здоров'я» (DAYS_7) видається за активність протягом семи календарних днів, а «Місяць дисципліни» (DAYS_30) — за тридцять. На відміну від попередньої групи, тут важлива не кількість записів, а саме регулярність, що свідчить про стабільне користування додатком з боку користувача.

Окремо я хотів би виділити групу, що відповідає за відстеження безперервної серії щоденного використання. Послідовність буде перервана, якщо користувач пропустить хоча б один день. Досягнення цієї групи мають зростаючий рівень складності: «3 дні поспіль» (STREAK_3) за безпосередньо 3 активних дні, «Тижнева серія» (STREAK_7) за сім днів, та «Два тижні» (STREAK_14) за чотирнадцять днів безперервної активності.

Найбільш складна для виконання група пов'язана з дотриманням розрахованої системою добової норми калорій. День буде вважатись успішним, якщо сума спожитих калорій становить не менше дев'яноста відсотків від індивідуальної норми користувача, вирахуваної за формулою Miffin-St Jeor. Перше досягнення «Ціль досягнута» (CALORIE_GOAL) видається після першого такого дня, наступні рівні як: «П'ять цілей» (CALORIE_GOAL_5) та «Двадцять цілей» (CALORIE_GOAL_20) — за п'ять і двадцять успішних днів відповідно.

Передостання група відноситься до контролю ваги. «Перше зважування» (WEIGHT_FIRST) буде розблоковане одразу після додавання першого запису з вагою, а «10 зважувань» (WEIGHT_10) — після десяти таких записів, що заохочує користувача регулярно фіксувати динаміку. Завершує перелік окреме досягнення «Шеф-кухар» (CUSTOM_FOOD), яке видається за створення першого власного продукту в каталозі особистих продуктів — користувачам, які не знайшли потрібного продукту в базі FatSecret і вирішили доповнити її самостійно.

Така структура досягнень дозволяє охопити різні типи користувацької поведінки: і ентузіастів, які активно записують кожен прийом їжі, і дисциплінованих користувачів, які повертаються в додаток щодня, і навіть тих, хто акцентує увагу на досягненні калорійної норми чи контролі ваги.

Метод CheckAndUnlockAsync виконує підрахунок показників активності (загальна кількість прийомів, кількість активних днів, поточна серія, досягнуті дні) та порівнює їх з умовами досягнень. Нові досягнення зберігаються в таблиці Achievements і повертаються клієнту для відображення сповіщень.

3.4.4 Сервіс аудиту (AuditService). AuditService забезпечує логування дій адміністраторів та системних подій. Кожен запис журналу містить: рівень важливості (Info/Warning/Error), тип дії, ідентифікатор та email актора (з JWT-клейму), ідентифікатор об'єкту дії, деталі, IP-адресу та шлях запиту. Сервіс використовується адміністративними контролерами та ErrorLoggingMiddleware.

3.5 Взаємодія з базою даних (EF Core)

Entity Framework Core є основним інструментом взаємодії бекенду з PostgreSQL. AppDbContext містить 8 DbSet-ів конфігурацію індексів через Fluent API та налаштування каскадного видалення.

Ключові особливості роботи з EF Core у CalorieTracker представляють з себе такі пункти:

- Auto-migrate при старті: Program.cs виконує db.Database.Migrate() при запуску застосунку, що автоматично застосовує всі невиконані міграції;
- Partial index для Foods.ExternalId: реалізований через HasFilter("\"ExternalId\" IS NOT NULL") у конфігурації моделі;
- Upsert-логіка для WeightRecord: пошук існуючого запису за (UserId, Date), оновлення або створення нового;

– Inline-копія nutritional fields у MealItem: при додаванні продукту до прийому всі харчові показники копіюються в рядок MealItem, що гарантує незмінність історичних даних.

3.6 Безпека та автентифікація

Безпека є критично важливим аспектом для будь-якої системи, що обробляє персональні дані користувачів. У CalorieTracker реалізовано комплексний підхід до безпеки на кількох рівнях.

Автентифікація на основі JWT. JSON Web Token (JWT) представляє з себе відкритий стандарт (RFC 7519) для безпечної передачі інформації між сторонами. Токен підписується алгоритмом HMAC-SHA256 з секретним ключем із конфігурації застосунку. Термін дії токена — 24 години. Кожен захищений ендпоінт декодується атрибутом [Authorize], який автоматично перевіряє наявність та дійсність токена.

Авторизація на основі ролей. Система реалізує дворівневу модель авторизації: звичайні користувачі (Role = "User") та адміністратори (Role = "Admin"). Адміністративні ендпоінти захищені атрибутом [Authorize(Roles="Admin")]. Підвищення ролі може бути реалізоване лише через прямий SQL-запит у Supabase — цілеспрямовано для запобігання несанкціонованого отримання прав адміністратора третіми особами.

Хешування паролів. Паролі зберігаються виключно у вигляді BCrypt-хешів (алгоритм Blowfish, коефіцієнт складності 11). Зберігання паролів у відкритому вигляді або у вигляді MD5/SHA-хешів не допускається.

CORS-політика. Для запобігання несанкціонованим міжсайтовим запитам налаштована CORS-політика AllowReactApp, що дозволяє

виконувати запити лише з завчасно дозволених джерел (перелічені в конфігурації). Підтримуються credentials у CORS-запитах для передачі JWT у заголовках.

Журналювання помилок. ErrorLoggingMiddleware перехоплює всі необроблені виключення на рівні middleware pipeline, записує їх у таблицю AuditLogs з рівнем Error та деталями виключення, після чого перекидає виключення далі для стандартної обробки ASP.NET Core (повернення HTTP 500).

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowReactApp", policy =>
    {
        var allowedOrigins = builder.Configuration["Cors:AllowedOrigins"]?.Split(',')
            ?? new[] { "https://localhost:5173", "http://localhost:5173", "http://localhost:5208" };

        policy.WithOrigins(allowedOrigins)
              .AllowAnyHeader()
              .AllowAnyMethod()
              .AllowCredentials();
    });
});
```

Рис 3.5 — Конфігурація безпеки в Program.cs

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Діаграма розгортання

Діаграма розгортання відображає фізичну архітектуру системи та описує розміщення програмних компонентів на обчислювальних вузлах. Система CalorieTracker розгортається у двох конфігураціях: середовище розробки (Development) та продуктивне середовище (Production).

У середовищі розробки бекенд-сервер запускається на localhost:7100 з використанням HTTPS-сертифіката розробника (.NET Dev Certs). Фронтенд Vite-сервер запускається паралельно на localhost:50636. Взаємодія між ними здійснюється через проксі-налаштування Vite.

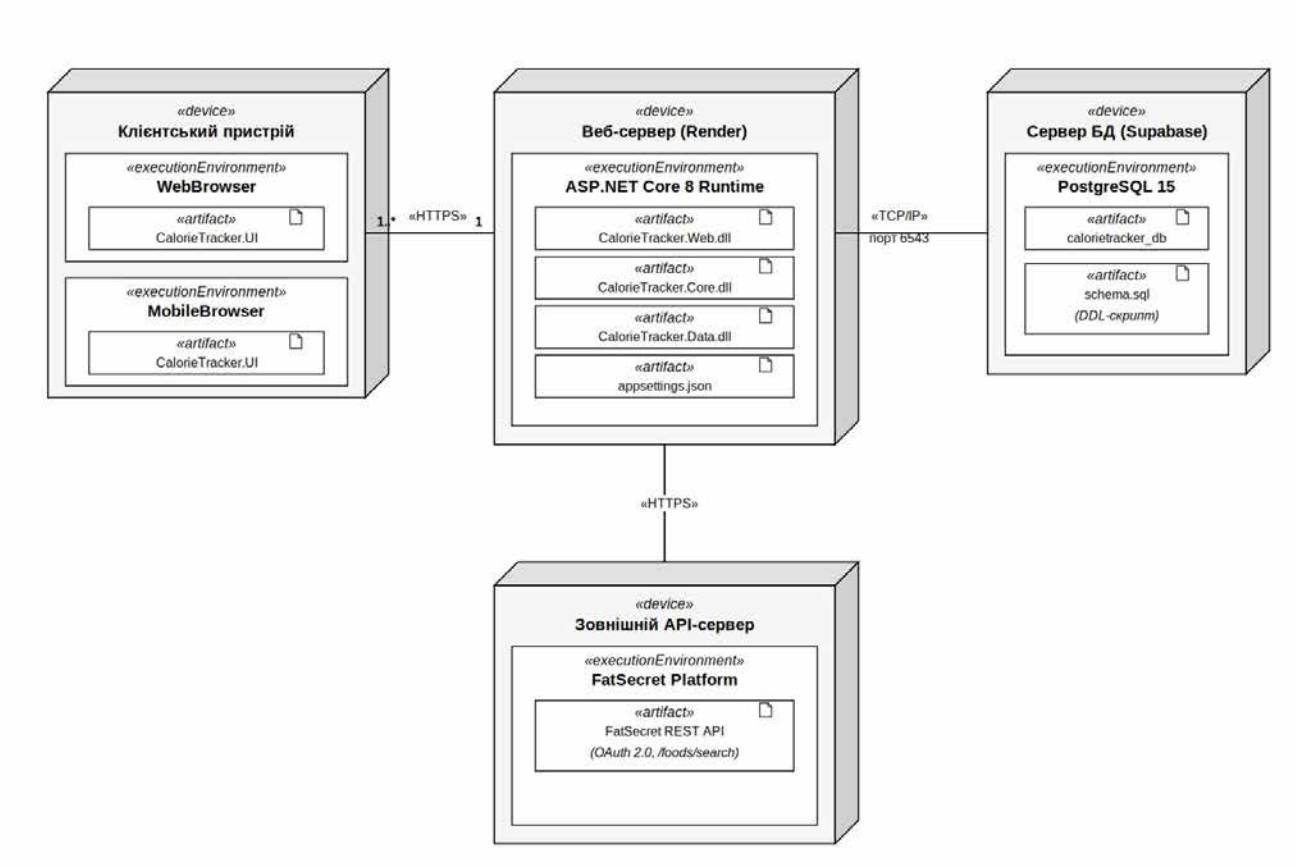


Рисунок 4.1 — Діаграма розгортання системи CalorieTracker

4.2 Апаратні та програмні вимоги до системи

Перед розгортанням системи CalorieTracker необхідно переконатись, що цільове середовище відповідає мінімальним технічним вимогам. Недотримання цих вимог може призвести до зниження продуктивності, нестабільної роботи бекенду або повної неможливості запуску застосунку. Вимоги логічно поділяються на дві групи: серверні — для машини, на якій розгорнуто API і базу даних, і клієнтські — для пристроїв кінцевих користувачів.

Апаратні вимоги до сервера: процесор архітектури x64 або ARM64 з мінімум двома ядрами; оперативна пам'ять — від 512 МБ (рекомендовано 1 ГБ і більше з огляду на необхідність кешування JWT-токенів і результатів FatSecret API через IMemoryCache); дисковий простір — близько 5 ГБ під .NET runtime, образи контейнерів та накопичення журналу AuditLogs.

З програмного забезпечення треба мати: операційну систему з підтримкою .NET 8 (Windows Server 2019 і новіше, Ubuntu 20.04+, Debian 11+ або будь-який дистрибутив із сумісним runtime), власне сам .NET 8 Runtime (або як альтернатива — Docker з офіційним образом mcr.microsoft.com/dotnet/aspnet:8.0) і PostgreSQL 14 або новіший. У моєму випадку PostgreSQL стоїть не на тому ж сервері, а в Supabase — тому локально його ставити не довелось.

Мережеві вимоги передбачають стабільне з'єднання сервера з мережею Інтернет. Воно необхідне у двох ключових сценаріях: підключення до бази даних Supabase (TCP, порт 6543) та виклики до зовнішнього FatSecret API через HTTPS. Низька якість каналу негативно впливає на швидкодію пошуку продуктів і у крайніх випадках може спричинити перебої роботи відповідних ендпоінтів.

Клієнтський пристрій — будь-який сучасний десктопний або мобільний пристрій (комп'ютер, ноутбук, смартфон, планшет) з браузером, що підтримує ECMAScript 2020 та сучасні специфікації CSS. Сюди належать Google Chrome 90+, Mozilla Firefox 88+, Safari 14+ та Microsoft Edge 90+. Окремого

встановлення клієнтського додатку не передбачено — React-застосунок передається сервером у форматі статичних файлів і виконується безпосередньо в браузері.

4.3 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною процесу розробки додатків, воно забезпечує відповідність системи функціональним та нефункціональним вимогам. Для тестування CalorieTracker були застосовані методи тестування, а саме метод чорного ящика (Black Box Testing), а також функціональне тестування через інтерфейс Swagger UI.

4.3.1 Тестування автентифікації та авторизації

В таблиці 4.2 продемонстровані тестові кейси автентифікації.

Таблиця 4.1

Тест-кейси автентифікації

№	Тест-кейс	Вхідні дані	Очікуваний результат	Отриманий результат
1	Реєстрація з коректними даними	Email, пароль, ім'я, дата народження, стать, рівень активності	HTTP 200, JWT токен	HTTP 200, JWT отримано
2	Реєстрація з існуючим email	Вже зареєстрований email	HTTP 400, повідомлення про помилку	HTTP 400, 'Email already exists'
3	Вхід з коректними даними	Email і правильний пароль	HTTP 200, JWT токен	HTTP 200, токен отримано
4	Вхід з неправильним паролем	Email і неправильний пароль	HTTP 401	HTTP 401
5	Запит до захищеного ендпоінту без токена	GET /api/users/profile без заголовку Authorization	HTTP 401	HTTP 401
6	Доступ до адмін-ендпоінту звичайним юзером	GET /api/admin/users з JWT звичайного користувача	HTTP 403	HTTP 403 Forbidden
7	Вхід заблокованого користувача	Email та пароль заблокованого	HTTP 403, причина	HTTP 403, повідомлення

№	Тест-кейс	Вхідні дані	Очікуваний результат	Отриманий результат
		аккаунту	блокування	отримано

4.3.2 Тестування роботи з прийомами їжі

У таблиці 4.2 наведені тестові кейси управління прийомами їжі.

Таблиця 4.2

Тест-кейси управління прийомами їжі

№	Тест-кейс	Вхідні дані	Очікуваний результат	Отриманий результат
1	Отримання прийомів за дату без записів	GET /api/meals/daily/2026-05-01	HTTP 200, порожній список	HTTP 200
2	Створення прийому з продуктом	POST /api/meals з MealType=1, product дані	HTTP 200, MealDto з items	HTTP 200, meal створено
3	Перевірка розрахунку калорій	Продукт 200 ккал/100г, 150г	Calories = 300 ккал	300 ккал (200×150/100)
4	Видалення продукту зі збереженням записів	DELETE /api/foods/{id}, потім перегляд MealItems	MealItems зберігаються з харч. даними	Inline-копія збережена
5	Щоденна статистика нутрієнтів	GET /api/meals/daily/{date}	DailyNutritionSummary з totals	Calories, Protein, Fats, Carbs підраховані

4.3.3 Тестування інтеграції з FatSecret API

У таблиці 4.3 наведені тестові кейси FatSecret API.

Таблиця 4.3

Тест-кейси FatSecret API

№	Тест-кейс	Вхідні дані	Очікуваний результат	Отриманий результат
1	Пошук за назвою продукту	GET /api/foods/external/search?query=apple	Список ExternalFoodDto	✓ Результати отримано
2	Пошук за штрих-кодом	GET /api/foods/external/barcode/4820003290232	FoodDto або 404	✓ Коректна відповідь
3	Кешування результатів	Два однакових запити пошуку	Другий без звернення до FatSecret	✓ IMemoryCache спрацював

4.3.4 Практична перевірка через Swagger UI. Паралельно з табличним описом тестових кейсів було проведено практичну перевірку всіх ендпоінтів через Swagger UI. Він є найзручнішим інструментом, тому що HTTP-запити виконуються прямо з браузера без Postman, curl або інших сторонніх застосунків. Для кожного запиту фіксувались три речі: HTTP-код, який повернув сервер, годину обробки та тіло JSON-відповіді. Такий формат тестування близький до того, як з API буде працювати реальний фронтенд.

Приклад кількох таких запитів показано на рисунку 4.3. Зверху – виклик автентифікації POST /api/auth/login. Сервер дав відповідь код 200 і повернув JWT-токен у JSON. Саме цей токен підставляється в заголовок Authorization: Bearer для захищених запитів. Нижче наведено приклад одного з таких захищених дзвінків, GET /api/meals/daily/date. Він повертає прийоми їжі за конкретну дату разом із зведенням за калоріями та макронутрієнтами. Структура відповіді DailyMealsDto, всередині якої лежить об'єкт Summary з завчасно підрахованими TotalCalories, TotalProtein, TotalFats та TotalCarbs. Підрахунок проводиться за формулою: кількість продукту в грамах, перемножується на його харчову цінність на 100 г і ділиться на 100 г.

Перевірка проведена успішно. Усі заплановані сценарії з тестових таблиць відпрацювали так, як було передбачено у вимогах розділу 1.4. Розбіжностей між очікуваними та реальними відповідями я не зафіксував, тому цілісність даних не порушувалася, харчові показники вважалися коректними, доступ до адмін-ендпоінтів зі звичайним токеном був заблокований — як це має бути.

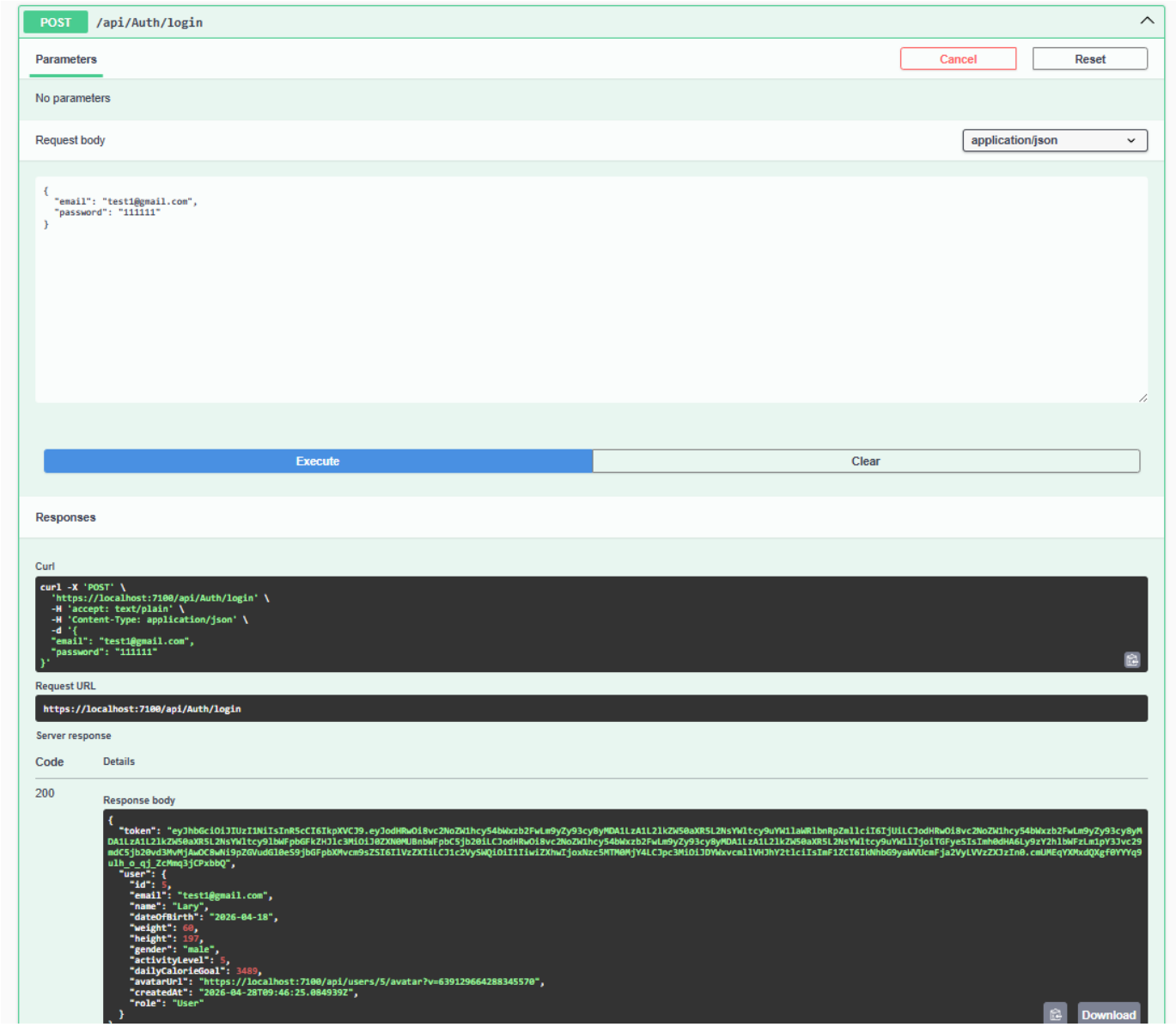


Рисунок 4.2 — Результати тестування API через Swagger UI

4.3.5 Користувацький інтерфейс системи. Попри те, що моя дипломна робота сфокусована на бекенд-частині, я визнав за доцільне навести скріншоти кінцевих сторінок CalorieTracker. Користувацький інтерфейс, реалізований як SPA на React 18 + TypeScript, тут виступає саме клієнтом і головним споживачем розробленого API, наочно демонструючи результати його роботи.

На рис. 4.3 показано Dashboard і сторінку статистики. Дашборд завантажується одразу після входу, використовуючи дані з ендпоінтів профілю та прийомів їжі (GET /api/meals/daily/{date}). Бізнес-логіка підрахунку добової норми калорій (за формулою Miffin-St Jeor), а також агрегація спожитих БЖВ виконується виключно сервером. Клієнт отримує вже готовий об'єкт DailyMealsDto і лише рисує прогрес та статистичні плитки на основі цих обчислень.

Сторінка статистики, що розташована правіше, ілюструє роботу аналітичних запитів до бази даних. Перемикач періодів на 7/14/30 днів передає відповідні параметри до GET /api/users/statistics. Замість того, щоб вантажити браузер сирими масивами, бекенд самостійно групує історію харчування по днях. Сформований сервером масив даних дозволяє швидко вивести «тренд калорій», що суттєво спрощує аналіз загальної тенденції харчування без обчислювального навантаження на пристрій користувача.

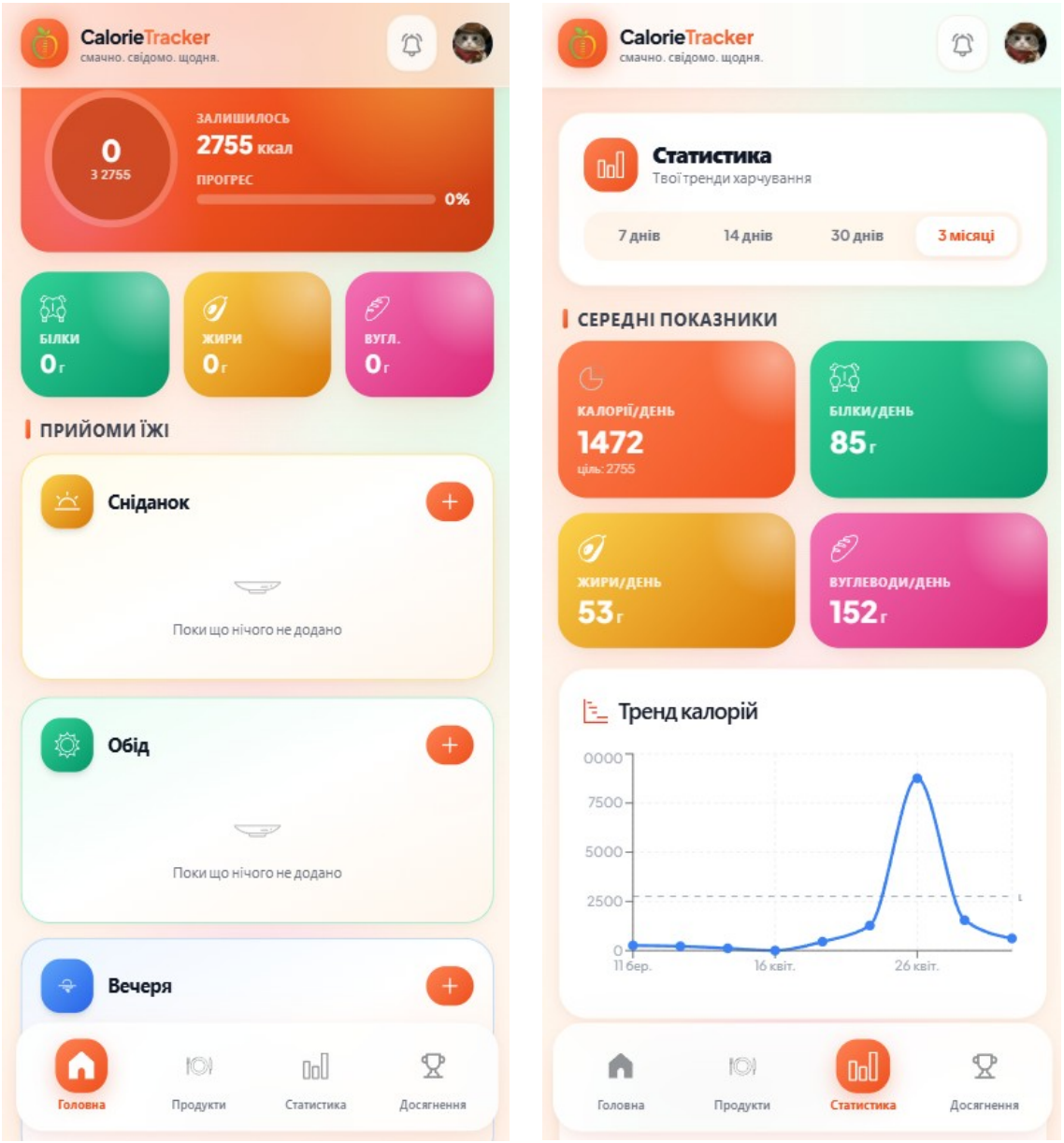


Рисунок 4.3 — Інтерфейс користувача CalorieTracker (Dashboard та Статистика)

4.3.6 Адміністративний модуль. Адмін-модуль побудовано навколо ізольованого набору ендпоінтів із префіксом /api/admin/ і відповідних сторінок у React-клієнті за маршрутом /admin. Захист тут дворівневий, але головним є серверний рубіж. Кожен адмін-контролер жорстко захищено атрибутом [Authorize(Roles="Admin")], який перевіряє наявність відповідного клейму в JWT-токені. Фронтенд-компонент AdminRoute лише допомагає з навігацією, відсіюючи звичайних користувачів і перенаправляючи їх на сторінку входу.

Усього реалізовано п'ять вкладок: Огляд, Користувачі, Продукти, Досягнення, Журнал. На рис. 4.4 показано перші дві. Вкладка Огляду повністю спирається на агрегаційні можливості AdminStatsController. Сервер збирає комплексну статистику по БД: загальну чисельність акаунтів, кількість заблокованих юзерів, MAU та необроблені помилки за добу, віддаючи все це одним AdminStatsDto. Картки оформлено у стилі решти додатку, що дає відчуття цілісного продукту.

Правий скріншот — вкладка користувачів. Це пряма репрезентація роботи GET-запиту /api/admin/users. Для оптимізації передачі даних реалізована серверна пагінація та фільтрація через query-параметри: search, role, blocked, page, pageSize. У стовпцях таблиці виводяться поля безпосередньо з AdminUserListItemDto. Дії над користувачами (блокування, розблокування, видалення) доступні відразу з рядка. Це елементарно швидше, ніж заходити в окрему сторінку профілю, і кожен такий POST/DELETE запит бекенд обов'язково пропускає через сервіс аудиту, фіксуючи зміну стану в базі логів.

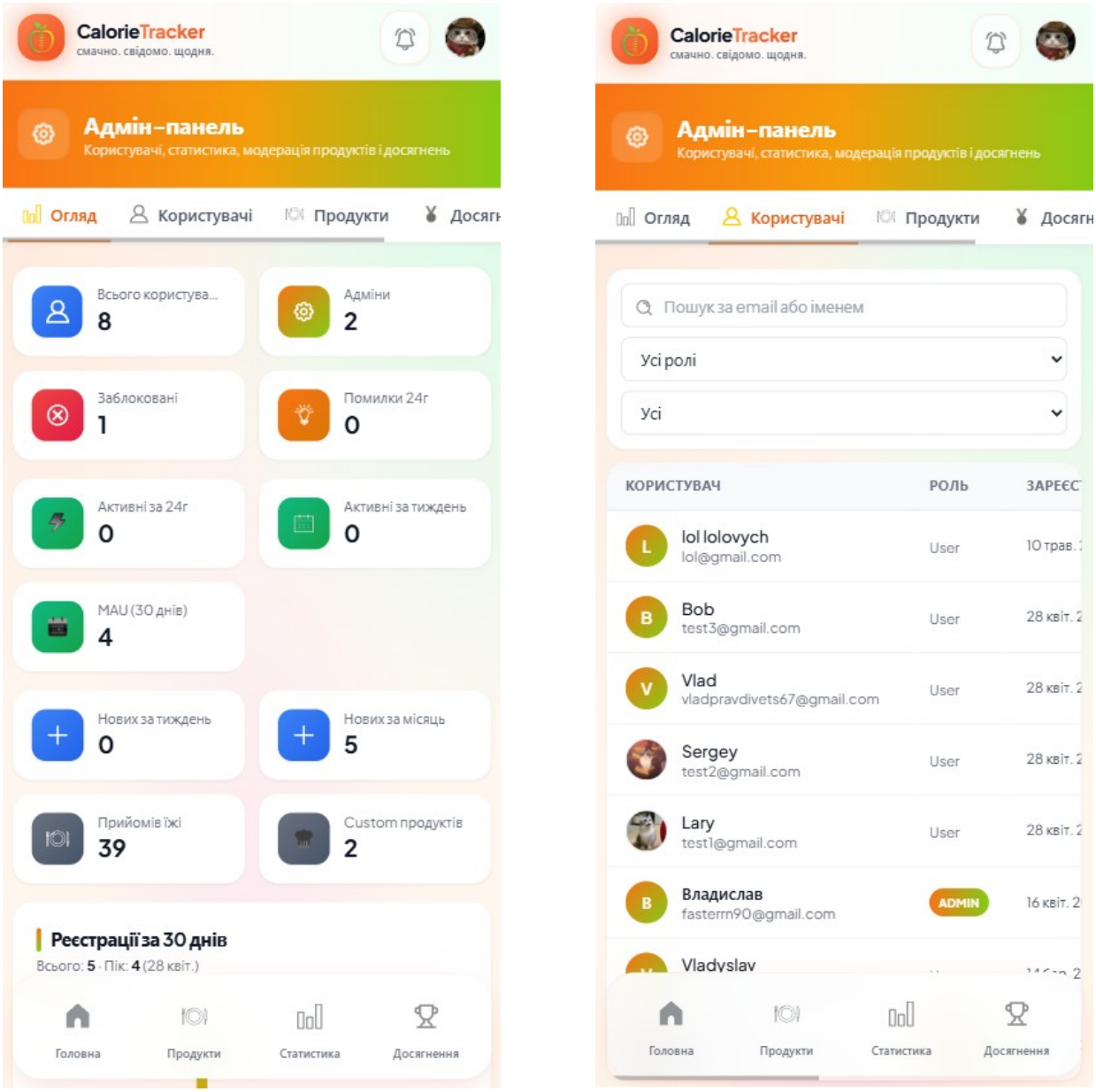


Рисунок 4.4 — Адміністративна панель системи

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було спроєктовано та розроблено бекенд-частину аналітичної системи підрахунку калорій при вживанні їжі людиною — веб-застосунок CalorieTracker.

За результатами виконання роботи можна сформулювати такі висновки:

- Було проведено аналіз предметної області, вивчено принципи підрахунку калорій та розрахунку базального рівня метаболізму за формулою Miffin-St Jeor. Здійснено порівняльний аналіз існуючих рішень (MyFitnessPal, Cronometer, FatSecret), виявлено їх недоліки та обґрунтована актуальність розробки додатку.
- Спроєктовано реляційну базу даних у третій нормальній формі (3НФ). База даних містить 8 таблиць: Users, Foods, Meals, MealItems, UserGoals, WeightRecords, Achievements, AuditLogs. Розроблено фізичну модель з урахуванням оптимізації продуктивності запитів через 9 індексів різних типів.
- Розроблено RESTful API на платформі ASP.NET Core 8 з більш ніж 25 ендпоінтами, організованими у 6 функціональних груп: автентифікація, управління користувачами, продукти харчування, прийоми їжі, досягнення та адміністративний модуль. API задокументовано за допомогою Swagger/OpenAPI.
- Реалізовано систему автентифікації та авторизації на основі JWT-токенів (HMAC-SHA256) з підтримкою рольової моделі (User/Admin). Зі зберіганням паролів у вигляді BCrypt-хешів.
- Реалізовано інтеграцію зі зовнішнім FatSecret Platform API для підтримки пошуку продуктів за назвою та штрих-кодом. Реалізовано кешування OAuth-токенів та результатів пошуку в IMemoryCache.

- Розроблено систему досягнень, що поетапно розблоковується при досягненні таких показників активності як: прийоми їжі, активні дні, серії без пропуску, досягнення добової норми калорій, записи ваги, додавання власних продуктів.
- Реалізовано адміністративний модуль із управлінням користувачами (перегляд, блокування, видалення), модерацією продуктів, переглядом системної статистики та пагінованим журналом аудиту.
- Проведено функціональне тестування розроблених компонентів. Тести підтвердили відповідність системи функціональним вимогам, а саме коректний розрахунок калорій, збереження цілісності даних при видаленні продуктів, правильну роботу авторизації та механізму досягнень.

Розроблений застосунок CalorieTracker є повністю функціональним програмним продуктом, що реалізує повний цикл трекінгу харчування. Бекенд-частина, як основний предмет даної роботи, відповідає всім визначеним функціональним та нефункціональним вимогам.

Перспективами подальшого розвитку системи є: перенесення визначень досягнень з коду до бази даних для спрощення адміністрування; реалізація жорсткого відкликання JWT-токенів при блокуванні користувача; розширення бази продуктів за рахунок прямої інтеграції з Ukrainian Food Database;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- World Health Organization. Obesity and overweight [Електронний ресурс]. — Режим доступу: https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight. — Дата доступу: 01.05.2026.
- Mifflin M.D., St Jeor S.T., Hill L.A., Scott B.J., Daugherty S.A., Koh Y.O. A new predictive equation for resting energy expenditure in healthy individuals // The American Journal of Clinical Nutrition. — 1990. — Vol. 51. — P. 241–247.
- Harris J.A., Benedict F.G. A Biometric Study of Human Basal Metabolism // Proceedings of the National Academy of Sciences of the United States of America. — 1918. — Vol. 4. — No. 12. — P. 370–373.
- Microsoft. ASP.NET Core documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core. — Дата доступу: 05.05.2026.
- Microsoft. Entity Framework Core documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/ef/core. — Дата доступу: 05.05.2026.
- The PostgreSQL Global Development Group. PostgreSQL 16 Documentation [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/16. — Дата доступу: 01.05.2026.
- FatSecret Platform API Documentation [Електронний ресурс]. — Режим доступу: https://platform.fatsecret.com/api. — Дата доступу: 01.05.2026.
- Дейт К.Дж. Введення в системи баз даних. — 8-е вид. — М.: Вільямс, 2005. — 1328 с.
- TechEmpower Framework Benchmarks [Електронний ресурс]. — Режим доступу: https://www.techempower.com/benchmarks. — Дата доступу: 05.05.2026.

- JWT Introduction [Електронний ресурс]. — Режим доступу: https://jwt.io/introduction. — Дата доступу: 01.05.2026.
- Provos N., Mazières D. A Future-Adaptable Password Scheme // USENIX Annual Technical Conference. — 1999.
- Richardson L., Ruby S. RESTful Web Services. — O'Reilly Media, 2007. — 446 p.
- Supabase Documentation [Електронний ресурс]. — Режим доступу: https://supabase.com/docs. — Дата доступу: 01.05.2026.
- Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Pearson, 2017. — 432 p.

- Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures: PhD dissertation. — Irvine: University of California, 2000. — 162 p.
- Internet Engineering Task Force. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. — 2015. — Режим доступу: https://datatracker.ietf.org/doc/html/rfc7519. — Дата доступу: 05.05.2026.
- Internet Engineering Task Force. RFC 6749: The OAuth 2.0 Authorization Framework [Електронний ресурс]. — 2012. — Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749. — Дата доступу: 05.05.2026.
- OWASP Foundation. OWASP Top 10: 2021 [Електронний ресурс]. — Режим доступу: https://owasp.org/Top10/. — Дата доступу: 10.05.2026.
- Smith J. Entity Framework Core in Action. — 2nd ed. — Shelter Island: Manning Publications, 2021. — 624 p.
- Lock A. ASP.NET Core in Action. — 3rd ed. — Shelter Island: Manning Publications, 2024. — 944 p.
- Microsoft. Npgsql Entity Framework Core Provider [Електронний ресурс]. — Режим доступу: https://www.npgsql.org/efcore/. — Дата доступу: 05.05.2026.

- SmartBear Software. Swagger / OpenAPI Specification [Електронний ресурс]. — Режим доступу: https://swagger.io/specification/. — Дата доступу: 05.05.2026.
- ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. — К.: ДП «УкрНДНЦ», 2016. — 26 с.

ДОДАТОК А

DDL-скрипт створення таблиць бази даних

Нижче наведено SQL-скрипт для створення всіх таблиць бази даних системи CalorieTracker у PostgreSQL.

```
-- Таблиця користувачів
CREATE TABLE "Users" (
    "Id" SERIAL PRIMARY KEY,
    "Email" VARCHAR(256) NOT NULL UNIQUE,
    "PasswordHash" TEXT NOT NULL,
    "Name" VARCHAR(100) NOT NULL,
    "DateOfBirth" DATE NOT NULL,
    "Gender" VARCHAR(10) NOT NULL,
    "ActivityLevel" INTEGER NOT NULL,
    "AvatarData" BYTEA,
    "AvatarContentType" VARCHAR(50),
    "AvatarUpdatedAt" TIMESTAMPTZ,
    "Role" VARCHAR(20) NOT NULL DEFAULT 'User',
    "IsBlocked" BOOLEAN NOT NULL DEFAULT false,
    "BlockedAt" TIMESTAMPTZ,
    "BlockedReason" TEXT,
    "CreatedAt" TIMESTAMPTZ NOT NULL,
    "UpdatedAt" TIMESTAMPTZ NOT NULL
);

CREATE UNIQUE INDEX "IX_Users_Email" ON "Users" ("Email");
CREATE INDEX "IX_Users_Role" ON "Users" ("Role");

-- Таблиця продуктів харчування
CREATE TABLE "Foods" (
    "Id" SERIAL PRIMARY KEY,
    "Name" VARCHAR(200) NOT NULL,
    "Brand" VARCHAR(100),
    "CaloriesPer100g" DECIMAL NOT NULL,
    "ProteinPer100g" DECIMAL NOT NULL,
    "FatsPer100g" DECIMAL NOT NULL,
    "CarbsPer100g" DECIMAL NOT NULL,
    "FiberPer100g" DECIMAL,
    "SugarPer100g" DECIMAL,
    "SodiumPer100g" DECIMAL,
    "Category" VARCHAR(50),
    "Barcode" VARCHAR(30),
    "ExternalId" VARCHAR(100),
    "Source" VARCHAR(20),
    "IsCustom" BOOLEAN NOT NULL,
    "CreatedBy" INTEGER REFERENCES "Users"("Id") ON DELETE SET NULL,
    "CreatedAt" TIMESTAMPTZ NOT NULL
);

CREATE INDEX "IX_Foods_ExternalId_Source"
    ON "Foods" ("ExternalId", "Source")
    WHERE "ExternalId" IS NOT NULL;
CREATE INDEX "IX_Foods_CreatedBy" ON "Foods" ("CreatedBy");
```

```
-- Таблиця прийомів їжі
CREATE TABLE "Meals" (
    "Id" SERIAL PRIMARY KEY,
    "UserId" INTEGER NOT NULL REFERENCES "Users"("Id") ON DELETE CASCADE,
    "Date" DATE NOT NULL,
    "MealType" INTEGER NOT NULL,
    "CreatedAt" TIMESTAMPTZ NOT NULL
);

CREATE INDEX "IX_Meals_UserId_Date" ON "Meals" ("UserId", "Date");

-- Таблиця позицій прийомів їжі
CREATE TABLE "MealItems" (
    "Id" SERIAL PRIMARY KEY,
    "MealId" INTEGER NOT NULL REFERENCES "Meals"("Id") ON DELETE CASCADE,
    "FoodName" VARCHAR(200) NOT NULL,
    "FoodBrand" VARCHAR(100),
    "CaloriesPer100g" DECIMAL NOT NULL,
    "ProteinPer100g" DECIMAL NOT NULL,
    "FatsPer100g" DECIMAL NOT NULL,
    "CarbsPer100g" DECIMAL NOT NULL,
    "FiberPer100g" DECIMAL,
    "ExternalId" VARCHAR(100),
    "Source" VARCHAR(20),
    "Quantity" DECIMAL NOT NULL
);

CREATE INDEX "IX_MealItems_MealId" ON "MealItems" ("MealId");

-- Таблиця цілей користувача
CREATE TABLE "UserGoals" (
    "Id" SERIAL PRIMARY KEY,
    "UserId" INTEGER NOT NULL REFERENCES "Users"("Id") ON DELETE CASCADE,
    "GoalType" INTEGER NOT NULL,
    "TargetWeight" DECIMAL,
    "TargetDate" DATE,
    "DailyCalorieTarget" DECIMAL NOT NULL,
    "DailyProteinTarget" DECIMAL,
    "DailyFatsTarget" DECIMAL,
    "DailyCarbsTarget" DECIMAL,
    "IsActive" BOOLEAN NOT NULL,
    "CreatedAt" TIMESTAMPTZ NOT NULL
);

CREATE INDEX "IX_UserGoals_UserId" ON "UserGoals" ("UserId");

-- Таблиця записів ваги
CREATE TABLE "WeightRecords" (
    "Id" SERIAL PRIMARY KEY,
    "UserId" INTEGER NOT NULL REFERENCES "Users"("Id") ON DELETE CASCADE,
    "Weight" DECIMAL NOT NULL,
    "Height" DECIMAL,
    "RecordDate" DATE NOT NULL,
    "CreatedAt" TIMESTAMPTZ NOT NULL
);

CREATE INDEX "IX_WeightRecords_UserId_RecordDate"
    ON "WeightRecords" ("UserId", "RecordDate");
```

```
-- Таблиця досягнень
CREATE TABLE "Achievements" (
    "Id" SERIAL PRIMARY KEY,
    "UserId" INTEGER NOT NULL REFERENCES "Users"("Id") ON DELETE CASCADE,
    "Code" VARCHAR(50) NOT NULL,
    "UnlockedAt" TIMESTAMPTZ NOT NULL
);

CREATE UNIQUE INDEX "IX_Achievements_UserId_Code"
    ON "Achievements" ("UserId", "Code");

-- Таблиця журналу аудиту
CREATE TABLE "AuditLogs" (
    "Id" SERIAL PRIMARY KEY,
    "Timestamp" TIMESTAMPTZ NOT NULL,
    "Level" VARCHAR(20) NOT NULL,
    "Action" VARCHAR(100) NOT NULL,
    "ActorUserId" INTEGER REFERENCES "Users"("Id") ON DELETE SET NULL,
    "ActorEmail" VARCHAR(256),
    "EntityType" VARCHAR(50),
    "EntityId" VARCHAR(50),
    "Details" TEXT,
    "IpAddress" VARCHAR(45),
    "Path" VARCHAR(500)
);

CREATE INDEX "IX_AuditLogs_Timestamp" ON "AuditLogs" ("Timestamp");
CREATE INDEX "IX_AuditLogs_Action" ON "AuditLogs" ("Action");
CREATE INDEX "IX_AuditLogs_ActorUserId" ON "AuditLogs" ("ActorUserId");
```

ДОДАТОК Б

Код ключових сервісів бекенду

AuthService.cs — сервіс автентифікації

```
public class AuthService : IAuthService
{
    private readonly IConfiguration _configuration;

    public AuthService(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public string HashPassword(string password)
    {
        return BCrypt.Net.BCrypt.HashPassword(password, workFactor: 11);
    }

    public bool VerifyPassword(string password, string hash)
    {
        return BCrypt.Net.BCrypt.Verify(password, hash);
    }

    public string GenerateJwtToken(User user)
    {
        var jwtSettings = _configuration.GetSection("JwtSettings");
        var key = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(jwtSettings["SecretKey"]!));

        var claims = new[]
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.Name, user.Name),
            new Claim(ClaimTypes.Role, user.Role),
            new Claim("userId", user.Id.ToString())
        };

        var token = new JwtSecurityToken(
            issuer: jwtSettings["Issuer"],
            audience: jwtSettings["Audience"],
            claims: claims,
            expires: DateTime.UtcNow.AddHours(
                double.Parse(jwtSettings["ExpirationHours"]!)),
            signingCredentials: new SigningCredentials(
                key, SecurityAlgorithms.HmacSha256)
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

AchievementService.cs — метод перевірки та розблокування досягнень

```
public async Task<List<Achievement>> CheckAndUnlockAsync(
    int userId, AppDbContext db)
{
    var now = DateTime.UtcNow;
    var existing = await db.Achievements
        .Where(a => a.UserId == userId)
        .Select(a => a.Code)
        .ToListAsync();

    var meals = await db.MealItems
        .Where(mi => mi.Meal.UserId == userId)
        .Select(mi => mi.Meal.Date)
        .ToListAsync();

    var totalMeals = await db.Meals
        .CountAsync(m => m.UserId == userId);
    var activeDays = meals.Distinct().Count();
    var currentStreak = ComputeCurrentStreak(meals);

    var weightCount = await db.WeightRecords
        .CountAsync(w => w.UserId == userId);
    var customFoods = await db.Foods
        .CountAsync(f => f.CreatedBy == userId && f.IsCustom);

    var user = await db.Users.FindAsync(userId);
    var bmr = ComputeBmr(user);
    var goalHitDays = await CountCalorieGoalHitDays(userId, bmr, db);

    var newlyUnlocked = new List<Achievement>();

    foreach (var def in AllDefinitions)
    {
        if (existing.Contains(def.Code)) continue;

        bool shouldUnlock = def.Code switch
        {
            "FIRST_MEAL"      => totalMeals >= 1,
            "MEALS_10"        => totalMeals >= 10,
            "MEALS_50"        => totalMeals >= 50,
            "MEALS_100"       => totalMeals >= 100,
            "DAYS_7"          => activeDays >= 7,
            "DAYS_30"         => activeDays >= 30,
            "STREAK_3"        => currentStreak >= 3,
            "STREAK_7"        => currentStreak >= 7,
            "STREAK_14"       => currentStreak >= 14,
            "CALORIE_GOAL"    => goalHitDays >= 1,
            "CALORIE_GOAL_5"  => goalHitDays >= 5,
            "CALORIE_GOAL_20" => goalHitDays >= 20,
            "WEIGHT_FIRST"    => weightCount >= 1,
            "WEIGHT_10"       => weightCount >= 10,
            "CUSTOM_FOOD"     => customFoods >= 1,
            _                 => false
        };

        if (shouldUnlock)
        {
            var achievement = new Achievement
            {
```

```
                UserId = userId,
                Code = def.Code,
                UnlockedAt = now
            };
            db.Achievements.Add(achievement);
            newlyUnlocked.Add(achievement);
        }
    }

    if (newlyUnlocked.Any())
        await db.SaveChangesAsync();

    return newlyUnlocked;
}
```

ДОДАТОК В

Код контролерів REST API

MealsController.cs — отримання прийомів їжі за датою

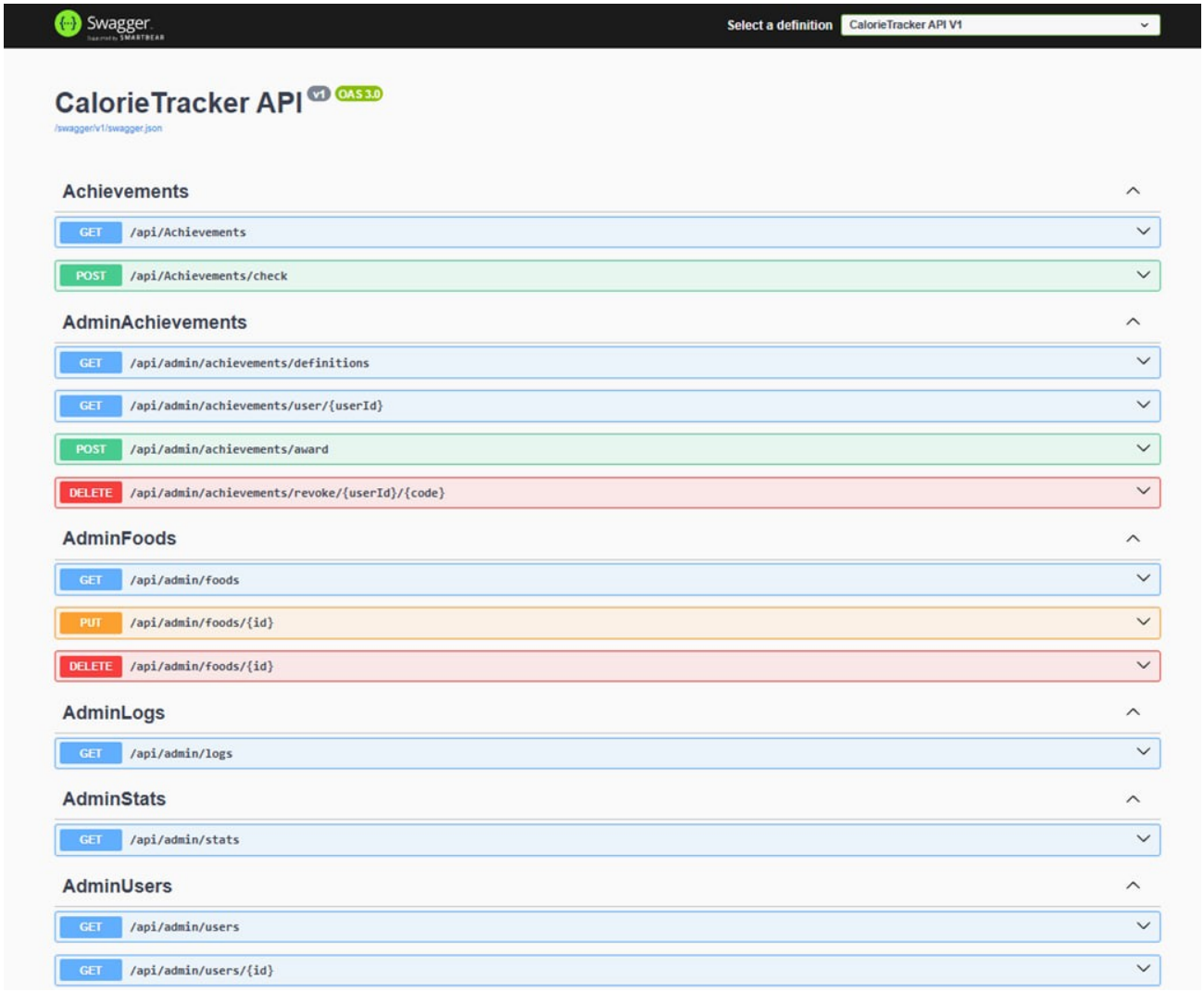
```
[ApiController]
[Route("api/[controller]")]
[Authorize]
public class MealsController : ControllerBase
{
    private readonly AppDbContext _db;

    public MealsController(AppDbContext db)
    {
        _db = db;
    }

    [HttpGet("daily/{date}")]
    public async Task<ActionResult<DailyMealsDto>> GetDailyMeals(
        DateOnly date)
    {
        var userId = int.Parse(
            User.FindFirst("userId")!.Value);

        var meals = await _db.Meals
            .Include(m => m.Items)
            .Where(m => m.UserId == userId && m.Date == date)
            .OrderBy(m => m.MealType)
            .ToListAsync();

        var user = await _db.Users.FindAsync(userId);
        var latestWeight = await _db.WeightRecords
            .Where(w => w.UserId == userId)
            .OrderByDescending(w => w.RecordDate)
            .FirstOrDefaultAsync();

        var dailyCalorieGoal = ComputeDailyCalorieGoal(
            user!, latestWeight);

        var summary = new DailyNutritionSummaryDto
        {
            TotalCalories = meals.SelectMany(m => m.Items)
                .Sum(i => i.Quantity * i.CaloriesPer100g / 100),
            TotalProtein  = meals.SelectMany(m => m.Items)
                .Sum(i => i.Quantity * i.ProteinPer100g / 100),
            TotalFats     = meals.SelectMany(m => m.Items)
                .Sum(i => i.Quantity * i.FatsPer100g / 100),
            TotalCarbs    = meals.SelectMany(m => m.Items)
                .Sum(i => i.Quantity * i.CarbsPer100g / 100),
            DailyCalorieGoal = dailyCalorieGoal
        };

        return Ok(new DailyMealsDto
        {
            Date = date,
```

```
            Meals = meals.Select(MapMealToDto).ToList(),
            Summary = summary
        });
    }

    private static decimal ComputeDailyCalorieGoal(
        User user, WeightRecord? wr)
    {
        if (wr == null) return 2000;
        var age = DateTime.Today.Year - user.DateOfBirth.Year;
        double bmr = user.Gender == "Male"
            ? 10 * (double)wr.Weight + 6.25 * (double)(wr.Height ?? 170)
              - 5 * age + 5
            : 10 * (double)wr.Weight + 6.25 * (double)(wr.Height ?? 165)
              - 5 * age - 161;

        double[] multipliers = { 1.2, 1.375, 1.55, 1.725, 1.9 };
        var idx = Math.Clamp(user.ActivityLevel - 1, 0, 4);
        return (decimal)(bmr * multipliers[idx]);
    }
}
```

ДОДАТОК Д

Скріншоти та діаграми фундаментальних процесів системи

Рис 3.3 — Документація REST API у Swagger UI


Swagger
Select a definition
CalorieTracker API V1
CalorieTracker API
v1
OAS 3.0
/swagger/v1/swagger.json
Achievements
GET /api/Achievements
POST /api/Achievements/check
AdminAchievements
GET /api/admin/achievements/definitions
GET /api/admin/achievements/user/{userId}
POST /api/admin/achievements/award
DELETE /api/admin/achievements/revoke/{userId}/{code}
AdminFoods
GET /api/admin/foods
PUT /api/admin/foods/{id}
DELETE /api/admin/foods/{id}
AdminLogs
GET /api/admin/logs
AdminStats
GET /api/admin/stats
AdminUsers
GET /api/admin/users
GET /api/admin/users/{id}

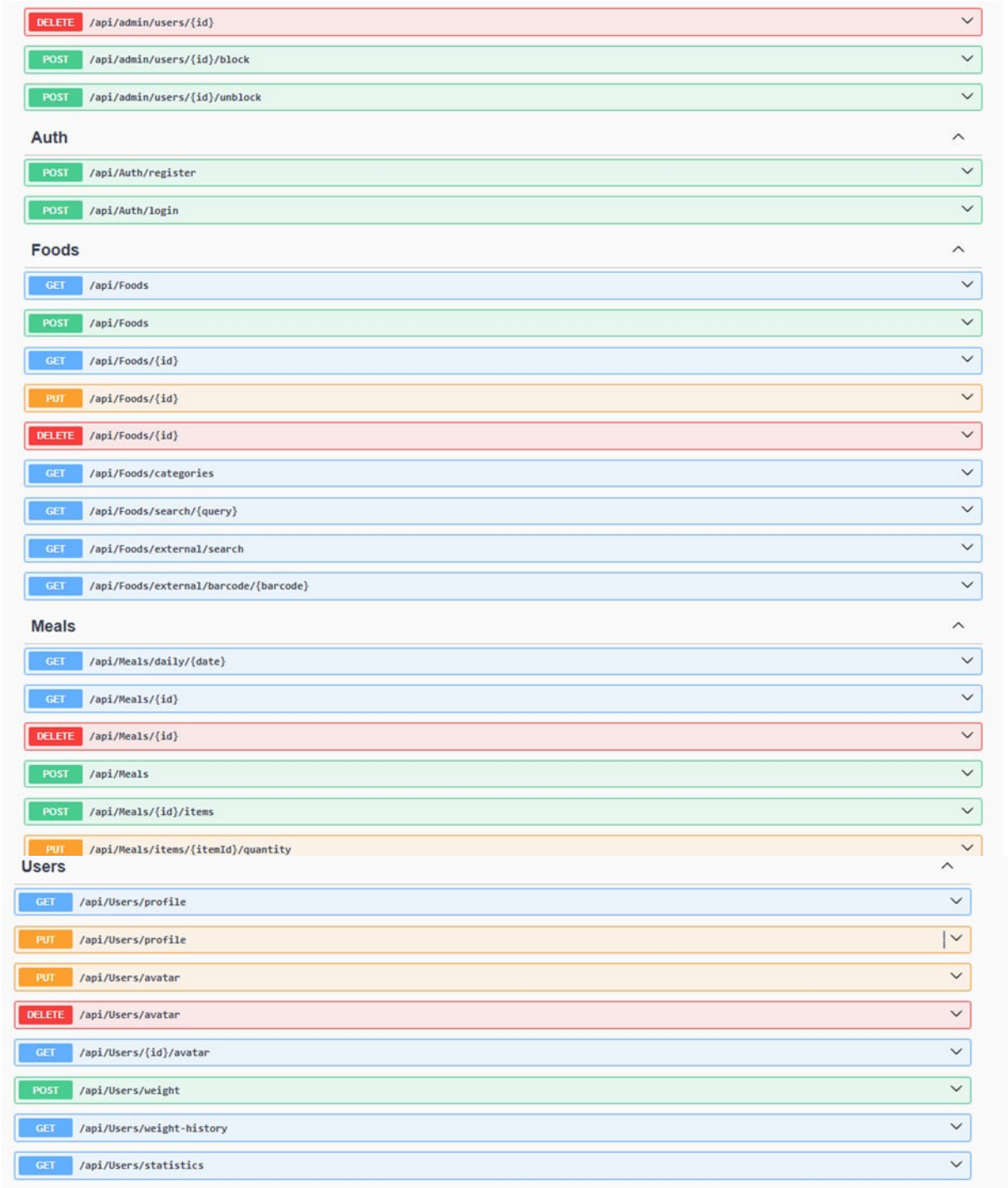


Рис 3.5 — Діаграма послідовності для сценарію «Додавання прийому їжі»

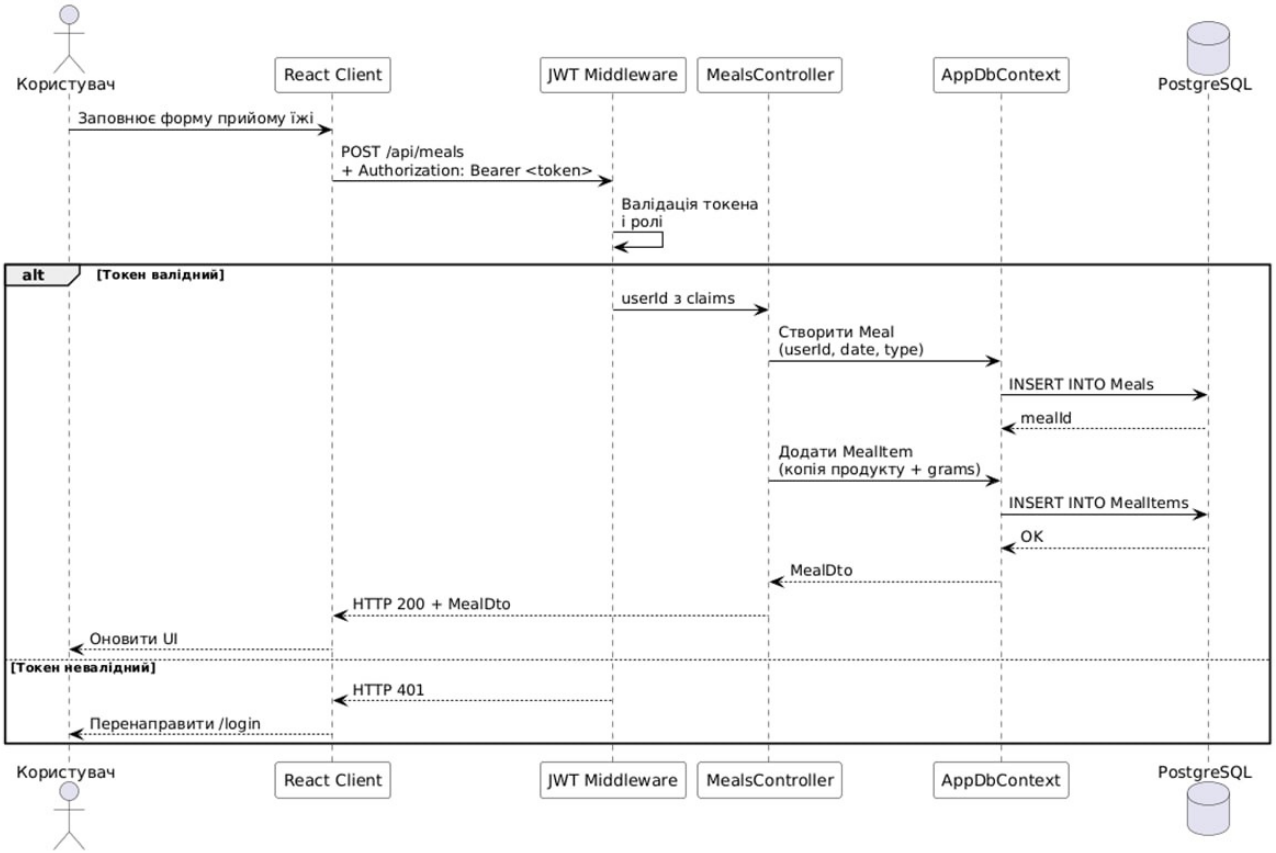


Рис 3.7 — Діаграма послідовності (Реалізація інтеграції з FatSecret API)

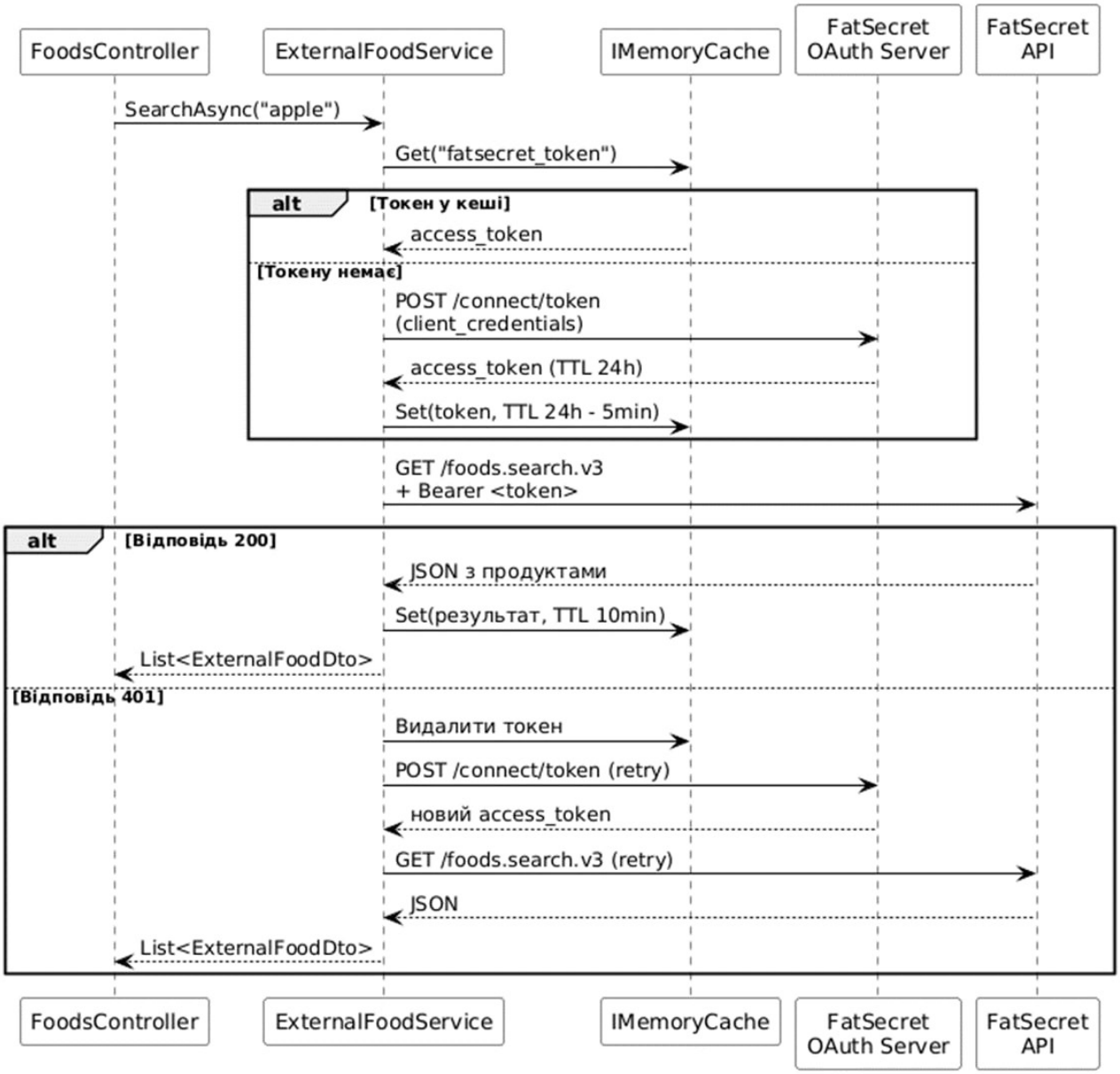
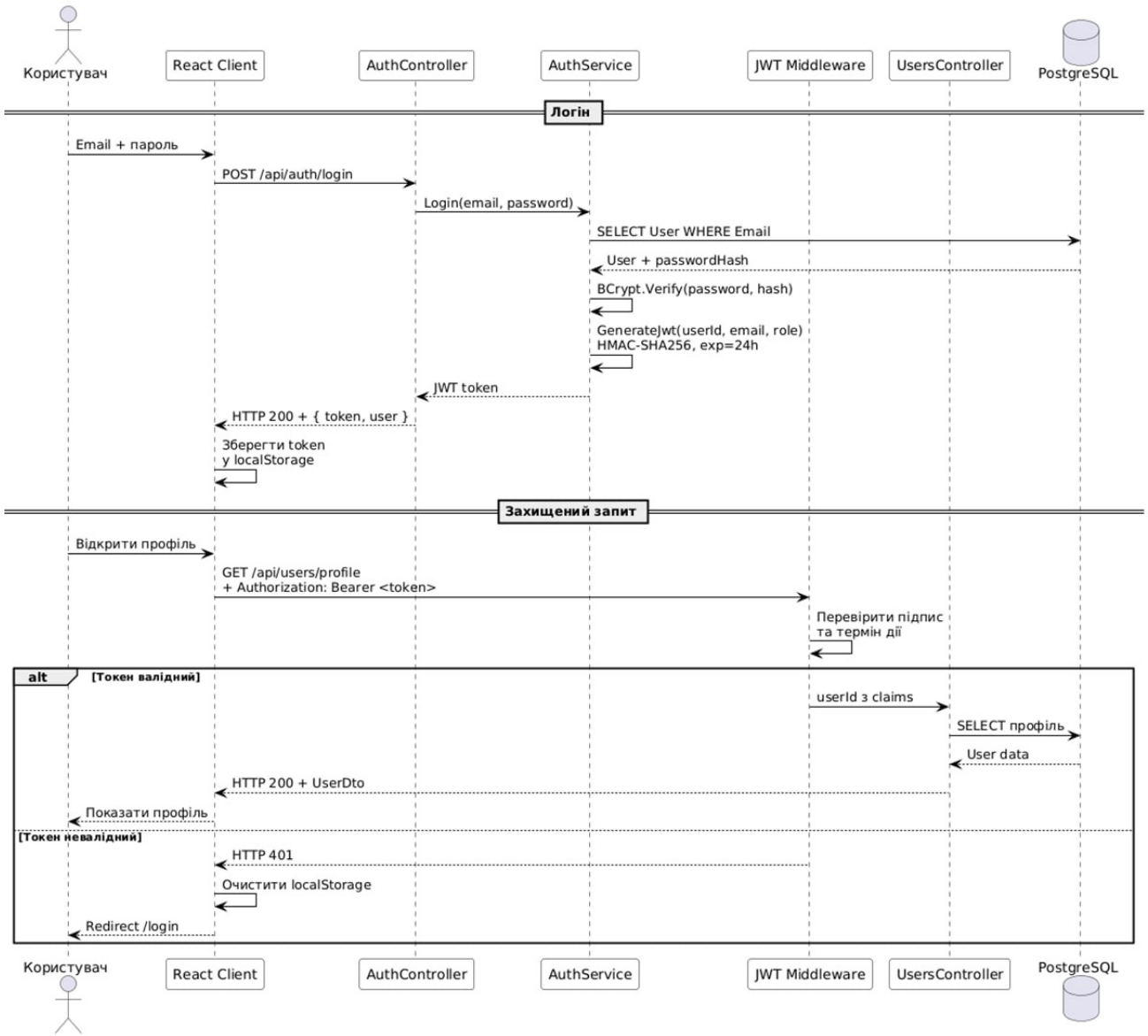


Рис 3.9 — Конфігурація безпеки в Program.cs


Користувач
React Client
AuthController
AuthService
JWT Middleware
UsersController
PostgreSQL
Логін
Email + пароль
POST /api/auth/login
Login(email, password)
SELECT User WHERE Email
User + passwordHash
BCrypt.Verify(password, hash)
GenerateJwt(userId, email, role)
HMAC-SHA256, exp=24h
JWT token
HTTP 200 + { token, user }
Зберегти token
у localStorage
Захищений запит
Відкрити профіль
GET /api/users/profile
+ Authorization: Bearer <token>
Перевірити підпис
та термін дії
alt
[Токен валідний]
userId з claims
SELECT профіль
User data
HTTP 200 + UserDto
Показати профіль
[Токен невалідний]
HTTP 401
Очистити localStorage
Redirect /login
Користувач
React Client
AuthController
AuthService
JWT Middleware
UsersController
PostgreSQL

ДОДАТОК Ж

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, **Бойко Владислав**, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

Моя бакалаврська кваліфікаційна робота (проєкт) на тему «**Бекенд частина аналітичної системи підрахунку калорій при вживанні їжі людиною**» є результатом моєї особистої праці.

1. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
2. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [+] інструменти генеративного ШІ не використовувалися.
    - [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [ ] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше:. _____

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«25» **травня** 2026 р. _____ **Ім'я ПРІЗВИЩЕ**
(підпис)