

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

(підпис) **Ігор БОЛБОТ**

(підпис) **Белла ГОЛУБ**

«01» _____ червня _____ 2026 р.

«01» _____ червня _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Фронтенд 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**

(підпис) **Ярослав ГОРДІЙ**

**Консультант бакалаврської
кваліфікаційної роботи**
к.т.н., доцент

(підпис) **Белла ГОЛУБ**

Виконав

(підпис) **Владислав ПРАВДИВЕЦЬ**

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ **Белла ГОЛУБ**

«06» _____ січня _____ 2026 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Правдивцю Владиславу Едуардовичу

_____ Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи: «Фронтенд 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): Користувацький інтерфейс веб-застосунку для візуалізації аналітичних даних та взаємодії з системою моніторингу харчування і раціону.

Перелік питань, що підлягають дослідженню:

- Аналіз предметної області та існуючих рішень обліку харчування.
- Проєктування архітектури клієнтської частини та інтерфейсу користувача.
- Програмна реалізація компонентів інтерфейсу, автентифікації, пошуку продуктів та візуалізації статистики.
- Тестування клієнтської частини та підготовка інсталяційного пакету.

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

**Керівник бакалаврської
кваліфікаційної роботи**

_____ **Ярослав ГОРДІЙ**

**Консультант
бакалаврської
кваліфікаційної роботи**

_____ **Белла ГОЛУБ**

**Завдання взяв до
виконання**

_____ **Владислав ПРАВДИВЕЦЬ**

ЗМІСТ

ВСТУП.....	6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМ ОБЛІКУ ХАРЧУВАННЯ.....	9
1.1 Характеристика предметної області.....	9
1.2 Моделювання предметної області.....	10
1.3 Огляд інформаційних джерел та існуючих рішень.....	13
1.4 Постановка завдання.....	17
2 ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ.....	22
2.1 Односторінковий застосунок та його складові.....	22
2.2 Взаємодія клієнтської та серверної частин системи.....	23
2.3 Архітектура клієнтської частини.....	25
3 ПРОЄКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	27
3.1 Організаційна структура програмного забезпечення.....	27
3.2 Вибір інструментарію для розробки клієнтської частини.....	29
3.3 Алгоритмізація та програмування основних модулів.....	32
3.4 Опис інтерфейсу користувача.....	36
4 ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ.....	51
4.1 Тестування клієнтської частини.....	51
4.2 Вимоги до апаратного та програмного забезпечення.....	54
4.3 Склад інсталяційного пакету.....	54
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	58
ДОДАТКИ.....	61
ДОДАТОК А.....	61
ДОДАТОК Б.....	66
ДОДАТОК В.....	72
ДОДАТОК Д.....	94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- API (Application Programming Interface) – прикладний програмний інтерфейс
- BMI (Body Mass Index) – індекс маси тіла
- CLS (Cumulative Layout Shift) – сукупне зміщення макета
- CSS (Cascading Style Sheets) – каскадні таблиці стилів
- DOM (Document Object Model) – об’єктна модель документа
- DTO (Data Transfer Object) – об’єкт передачі даних
- FCP (First Contentful Paint) – перше відмальовування контенту
- HMR (Hot Module Replacement) – гаряча заміна модулів
- HTML (HyperText Markup Language) – мова розмітки гіпертексту
- HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі даних
- JSON (JavaScript Object Notation) – формат обміну даними
- JWT (JSON Web Token) – стандарт токenu автентифікації
- LCP (Largest Contentful Paint) – відмальовування найбільшого контенту
- MAU (Monthly Active Users) – щомісячна активна аудиторія користувачів
- ORM (Object-Relational Mapping) – об’єктно-реляційне відображення
- PAL (Physical Activity Level) – рівень фізичної активності
- PWA (Progressive Web App) – прогресивний веб-застосунок
- REST (Representational State Transfer) – архітектурний стиль веб-сервісів
- SPA (Single Page Application) – односторінковий застосунок
- SVG (Scalable Vector Graphics) – масштабована векторна графіка
- TBT (Total Blocking Time) – загальний час блокування
- UI (User Interface) – інтерфейс користувача
- UML (Unified Modeling Language) – уніфікована мова моделювання
- БД – база даних
- ВООЗ – Всесвітня організація охорони здоров’я
- ЗСЖ – Здоровий спосіб життя

КБЖВ – калорії, білки, жири, вуглеводи

ОС – операційна система

СУБД – система управління базами даних

ВСТУП

Актуальність теми. Наразі все більше зростає інтерес людей щодо питань збалансованого харчування та ЗСЖ. Контроль власного раціону є одним із основних інструментів підтримання чи покращення здоров'я – зниження ваги, набору м'язової маси або підтримка поточного стану. Але ручний підрахунок поживних речовин залишається кропітким завданням, що вимагає систематичного підходу та значних часових витрат. Існуючі програмні рішення у цій сфері, як правило, пропонують широкий функціонал, проте є складними у налаштуванні під індивідуальні потреби або обмежують частину можливостей у безкоштовних версіях. Тому розробка власного веб-застосунку для персонального обліку харчування, адаптованого під конкретні вимоги користувача, є актуальним практичним завданням.

Мета розробки. Мета роботи – спроектувати та реалізувати клієнтську частину аналітичної системи CalorieTracker, що забезпечить зручний і сучасний інтерфейс для щоденного обліку харчування, підрахунку споживаних калорій і макронутрієнтів, відстеження динаміки ваги та перегляду статистичних даних шляхом розробки односторінкового веб-застосунку (SPA) з використанням стеку React + TypeScript + Vite.

Методи та технології. У роботі застосовано методи об'єктно-орієнтованого проєктування та UML-моделювання для опису архітектури системи, а також компонентний підхід до розробки інтерфейсу. Клієнтська частина реалізована з використанням бібліотеки React 18 – для побудови декларативного компонентного UI з реактивним оновленням стану. Мову TypeScript обрано для забезпечення статичної типізації та узгодженості типів між клієнтом і серверними DTO, що суттєво знижує кількість помилок на етапі розробки. Засіб збірки Vite забезпечує миттєвий запуск середовища розробки з підтримкою Hot Module Replacement. CSS-фреймворк Tailwind CSS застосовано для швидкої реалізації адаптивного мобільно-орієнтованого інтерфейсу через utility-first підхід. HTTP-клієнт Axios налаштовано з interceptor-ами для

автоматичної передачі JWT-токену та обробки помилок автентифікації. Бібліотека Recharts використовується для побудови інтерактивних графіків статистики харчування. Маршрутизацію між сторінками реалізовано через React Router DOM v6. Для сканування штрих-кодів продуктів через камеру пристрою підключено бібліотеку @zxing/browser, а взаємодія з бекендом (ASP.NET Core 8) здійснюється через REST API у форматі JSON із JWT-автентифікацією.

Апробація результатів. Основні результати роботи апробовано шляхом участі у Всеукраїнській науково-практичній конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем» (ТАACSD 2026, НУБіП України, Київ, 2026), де опубліковано тези доповіді за темою «Застосування технології Progressive Web App для оптимізації продуктивності односторінкових фітнес-додатків»

Структура роботи. Пояснювальна записка складається зі вступу, чотирьох розділів, списку використаних джерел та додатків. Загальний обсяг записки становить 60 сторінок (без додатків), список джерел налічує 29 позицій. Додатки містять графічну частину, лістинги коду та декларацію про академічну доброчесність.

У першому розділі виконано системний аналіз предметної області обліку харчування. Описані ключові сценарії взаємодії користувача з системою, потреби та очікування від інтерфейсу, наведено UML-діаграми. Проведено огляд інтерфейсів існуючих програмних рішень та сформульовано функціональні і нефункціональні вимоги до системи.

Другий розділ присвячений проектуванню клієнтської частини системи. Описана архітектура SPA-застосунку, його структурні складові та механізм взаємодії з серверною частиною через REST API.

Третій розділ присвячений розробці програмного забезпечення: організаційній структурі застосунку, обґрунтуванню технологічного стеку, алгоритмам ключових модулів (автентифікація, пошук продуктів, підрахунок

КБЖВ, статистика, досягнення), а також детальному опису інтерфейсу кожного екрана.

У четвертому розділі проведено тестування та описано впровадження системи. Наведено результати функціонального тестування, аудит продуктивності засобами Google Lighthouse, вимоги до апаратного та програмного забезпечення і склад інсталяційного пакету.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМ ОБЛІКУ ХАРЧУВАННЯ

1.1 Характеристика предметної області

Ця предметна область даного дослідження охоплює механізми запису, обробки та аналізу даних про харчування людини. Основа цієї сфери – це розрахунок енергетичного балансу на основі фізіологічних показників. Обробка інформації базується на таких змінних, як вік, стать, поточна вага, зріст та рівень щоденної фізичної активності. Ці вхідні дані використовуються для визначення базового обміну речовин та формування індивідуальної добової норми споживання.

Центральною характеристикою є постійна взаємодія з великими об'ємами харчових даних. Кожен продукт виступає як інформаційна одиниця, яка містить відомості про загальну енергетичну цінність та точне співвідношення макронутрієнтів. Процес обліку зводиться до ідентифікації такого продукту в глобальних базах за допомогою текстового запиту або зчитування штрих-коду. Після знаходження продукт додається до конкретного прийому їжі із зазначенням його фактичної маси, що забезпечує миттєвий перерахунок усіх пов'язаних денних показників.

Процес обліку має бути безперервним. Аналізувати лише один день недостатньо. Система повинна обробляти інформацію за тижні, місяці або навіть роки. Усі ці записи перетворюються на зручні графіки. Вони наочно показують загальну динаміку харчування, співвідношення поживних речовин та зміни у вазі. Такий тривалий аналіз допомагає зрозуміти, чи дійсно працює обраний план харчування.

Щоб регулярний запис їжі став корисною звичкою, а не обов'язком, що виснажує, такі системи активно працюють з мотивацією користувача. Для цього використовуються ігрові елементи. Наприклад, програми видають віртуальні нагороди за виконані цілі, рахують дні безперервного ведення щоденника та

яскраво показують досягнуті успіхи. Завдяки цьому рутинне введення цифр перетворюється на цікавий процес, що значно полегшує підтримання самодисципліни.

Специфічним є і середовище взаємодії. Введення інформації зазвичай відбувається епізодично та в умовах дефіциту часу, наприклад перед прийомом їжі або одразу після нього. Тому архітектура та логіка рішень у цій предметній області вимагають створення інтерфейсів, які гарантують максимальну швидкість виконання цільових дій, мінімізують кількість переходів між екранами та надають миттєвий візуальний відгук.

1.2 Моделювання предметної області

Для формального опису системи CalorieTracker розроблено комплекс UML-діаграм: діаграму прецедентів, діаграму послідовності та діаграму діяльності.

Діаграма прецедентів відображає взаємодію основних учасників системи та сценарії їх взаємодії. Виділено двох акторів предметної області – Користувача та Нутриціолога, з основними процесами обліку харчування. Користувач виконує введення параметрів тіла та цілей, ведення щоденника харчування та перегляд статистики. Нутриціолог, у свою чергу, забезпечує створення планів харчування та аналіз харчування користувача. Прецедент «Перегляд статистики» є спільним для обох акторів. Діаграма прецедентів зображена на рис. 1.



Рис. 1 Діаграма прецедентів предметної області системи обліку харчування

Діаграма послідовності ілюструє процес ведення щоденника харчування з точки зору предметної області. Користувач обирає дату та тип прийому їжі, після чого здійснює пошук продукту за назвою у каталозі продуктів. Каталог повертає список продуктів із харчовим складом, користувач обирає потрібний та отримує показники КБЖВ. Далі користувач вказує кількість спожитого продукту в грамах та записує прийом їжі до щоденника. Після підтвердження запису щоденник автоматично підраховує спожиті калорії за день і надає користувачу підсумок раціону з денними показниками КБЖВ. Діаграма прецедентів представлена на рис. 2.

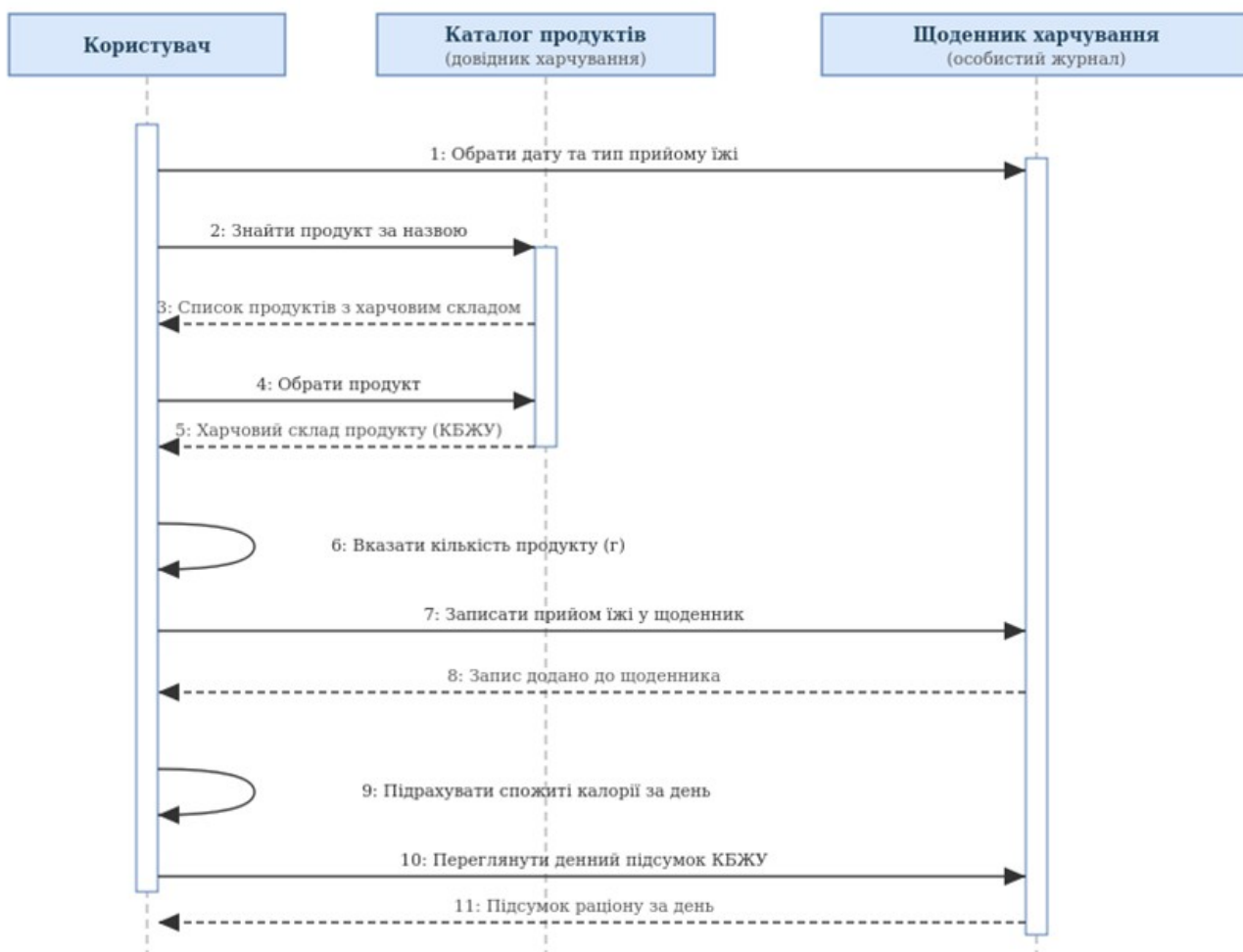


Рис. 2 Діаграма послідовності процесу ведення щоденника харчування

Діаграма діяльності описує процес ведення щоденника харчування в предметній області. Діяльність починається з введення користувачем параметрів тіла (вага, зріст, мета). Далі виконується аналіз отриманих даних та створення індивідуального плану харчування, який користувач переглядає і затверджує. Після цього користувач щоденно вносить спожиті продукти до щоденника – система виконує підрахунок КБЖВ та аналіз дотримання плану харчування. За результатами аналізу формується звіт з рекомендаціями. Якщо користувач вирішує продовжувати ведення щоденника, то цикл повторюється з кроку внесення продуктів, інакше діяльність завершується. Діаграму діяльності представлено на рис. 3.



Рис. 3 Діаграма діяльності процесу ведення щоденника харчування

1.3 Огляд інформаційних джерел та існуючих рішень

Аналіз існуючих застосунків для обліку харчування є важливим етапом проєктування, оскільки дозволяє виявити сильні та слабкі сторони наявних рішень з точки зору досвіду користувача, і сформулювати вимоги до власної

системи. Для огляду обрано три найпоширеніші трекери харчування: MyFitnessPal, Cronometer та FatSecret.

1.3.1 MyFitnessPal. Є найпопулярнішим у світі трекером калорій з базою понад 14 млн продуктів (рис. 4).

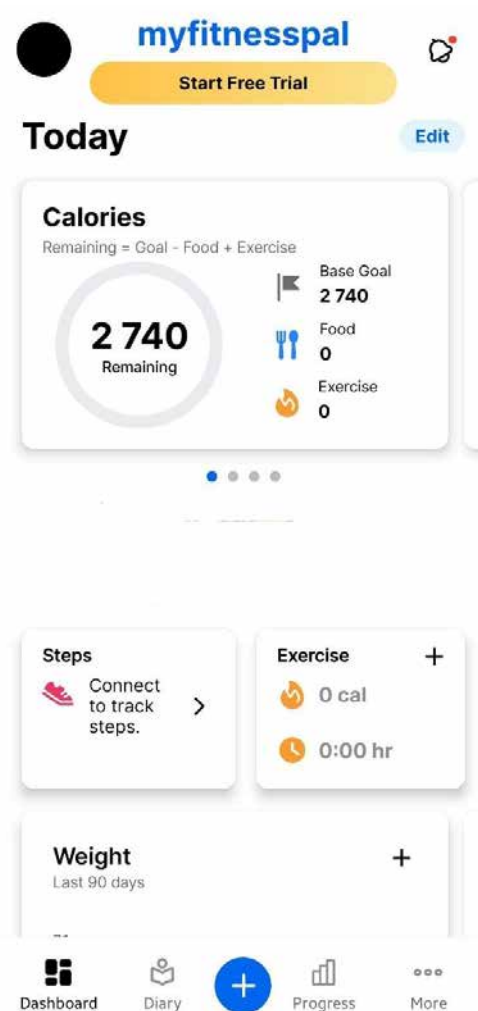


Рис. 4 Головний екран застосунку MyFitnessPal

З точки зору інтерфейсу, головний екран MyFitnessPal побудований навколо кільцевого індикатора «Калорії за день»: у центрі відображається залишок до денної норми, а зовні – дуги спожитих калорій. Нижче розміщено горизонтальну панель макронутрієнтів з трьома прогрес-барами для білків, жирів та вуглеводів. Прийоми їжі згруповані у розкладні секції – «Breakfast», «Lunch», «Dinner», «Snacks» – кожна з кнопкою «Add Food». Навігація між розділами здійснюється через нижню панель з п'ятьма іконками. Сканер штрих-

кодів доступний безпосередньо з екрана пошуку одним натисканням на іконку камери.

Серед недоліків слід відзначити перевантаженість інтерфейсу рекламними банерами у безкоштовній версії, а також складність налаштування цілей для нових користувачів – процес onboarding містить більше десяти кроків. Деякі розширені аналітичні функції (звіти по мікронутрієнтах, детальні діаграми) приховані за платною підпискою.

1.3.2 Cronometer. Застосунок з акцентом на точний облік мікронутрієнтів (рис. 5).

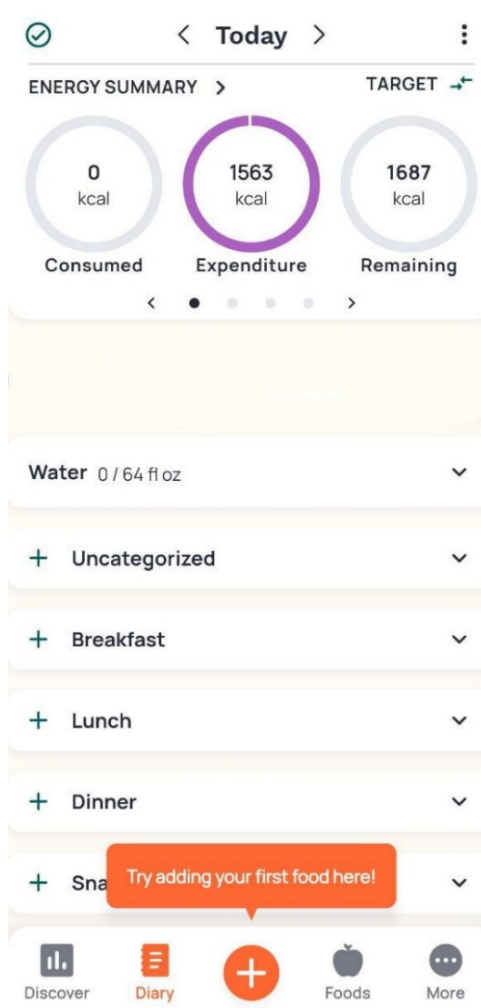


Рис. 5 Головний екран застосунку Cronometer

Головний екран Cronometer реалізований як вертикальний список: кругова діаграма калорій у верхній частині, нижче – розгорнутий список усіх мікронутрієнтів (залізо, кальцій, вітамін С тощо) з прогрес-барами. Для кожного

нутрієнта відображається фактичне значення, добова норма та відсоток виконання. Такий підхід підходить для медичного контролю харчування, але перевантажує екран для пересічного користувача, якому потрібна лише базова інформація про КБЖВ. Пошук продуктів є зручним, однак результати відображаються у вигляді щільного списку без візуальних розділювачів, що ускладнює швидкий вибір.

Загалом інтерфейс Cronometer орієнтований на досвідченого користувача, знайомого з нутриціологією. Для початківця він є складним у освоєнні через велику кількість показників і відсутність спрощеного режиму відображення. Відсутня також україномовна локалізація.

1.3.3 FatSecret. Трекер харчування з широкою базою продуктів і простим інтерфейсом (рис. 6).

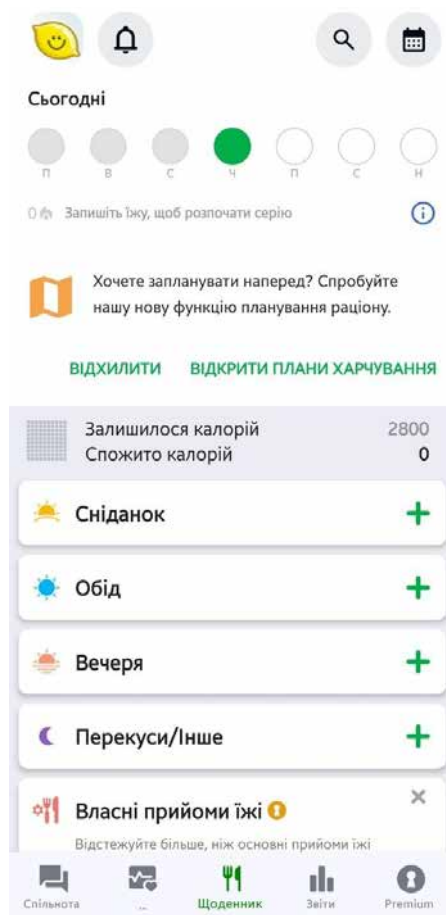


Рис. 6 Головний екран додатку FatSecret

FatSecret пропонує мінімалістичний підхід до відображення даних: на головному екрані відображається лише загальна кількість спожитих калорій та три значення макронутрієнтів у числовому вигляді без графічних індикаторів. Журнал харчування розбито на типові прийоми їжі. Пошук продуктів підтримує введення тексту та сканування штрих-кодів. Однак відсутність прогрес-барів і кругових індикаторів робить сприйняття даних менш наочним порівняно з конкурентами. Сторінка статистики реалізована у вигляді простих стовпчастих діаграм без можливості вибору детального часового діапазону.

Аналіз трьох застосунків дозволив виявити кілька спільних тенденцій. Всі рішення використовують поділ прийомів їжі на подібні категорії: сніданок, обід, вечеря та перекус. Такий підхід зрозумілий користувачам і був запозичений у CalorieTracker. Кругові або кільцеві індикатори денної норми калорій є ефективним і звичним для користувачів способом відображення прогресу. Але жоден із розглянутих безкоштовних рішень не пропонує одночасно простого зручного мобільного інтерфейсу, вбудованої системи досягнень та повністю відкритого функціоналу без обмежень.

На основі огляду визначено ключові орієнтири для проєктування інтерфейсу CalorieTracker. Мінімальна кількість кроків для типових дій, наочна візуалізація прогресу через кругові індикатори та кольорові картки макронутрієнтів, зручний пошук з підтримкою сканера штрих-кодів, адаптивний mobile-first дизайн та система досягнень, яка буде слугувати елементом мотивації регулярного ведення обліку.

1.4 Постановка завдання

На підставі проведеного аналізу предметної області та огляду існуючих рішень сформульовано вимоги до розроблюваної клієнтської частини системи.

1.4.1 Функціональні вимоги. Нижче наведені функціональні вимоги.

Підсистема автентифікації та профілю:

- реєстрація нового облікового запису з вказанням електронної пошти, пароля та імені;
- вхід в систему та отримання JWT-токену; автоматичне збереження токenu у localStorage та передача у заголовок Authorization: Bearer кожного запиту;
- перегляд та редагування профілю: дата народження, стать, рівень фізичної активності;
- завантаження аватара користувача через multipart/form-data;
- автоматичний перехід на сторінку входу при HTTP 401 через axios response interceptor.

Підсистема щоденного журналу харчування:

- відображення денних показників КБЖВ відносно персональної цілі для обраної дати;
- навігація між датами;
- відображення чотирьох секцій прийомів їжі: сніданок, обід, вечеря, перекус;
- додавання продукту до прийому їжі через модальне вікно з трьома режимами пошуку;
- редагування кількості (маса в грамах) вже доданого продукту;
- видалення продукту з прийому їжі.

Підсистема пошуку продуктів:

- текстовий пошук за назвою продукту через FatSecret API;
- пошук продукту за штрих-кодом через камеру пристрою;
- пошук серед власних продуктів, створених користувачем.

Підсистема управління продуктами:

- перегляд списку власних продуктів із можливістю текстового пошуку;
- створення власного продукту з повним набором показників КБЖВ;
- редагування та видалення автором власного продукту.

Підсистема статистики:

- відображення карток середніх показників КБЖВ за 7, 14, 30 та 90 днів;
- лінійний графік зміни калорій з позначкою денної норми;
- перемикач часового діапазону для оновлення графіку та карток.

Підсистема відстеження ваги та цілей:

- додавання нового запису поточної ваги та зросту;
- відображення графіка динаміки ваги за обраний період;
- встановлення цілі (наприклад, схуднення) та бажаної ваги.

Підсистема досягнень:

- відображення всіх досягнень із розподілом на розблоковані та заблоковані, із зазначенням дати отримання;
- перевірка умов і нарахування нових досягнень при відкритті сторінки.

Адміністративна панель (лише для ролі Admin):

- перегляд зведеної статистики системи: кількість користувачів, прийомів їжі, записів у журналі аудиту;
- управління обліковими записами: блокування, розблокування, видалення;
- модерація власних продуктів користувачів;
- перегляд журналу аудиту системи.

1.4.2 Нефункціональні вимоги. Нижче представлені нефункціональні вимоги:

- Продуктивність: час першого завантаження SPA – не більше 3 с на мобільній мережі 4G ($FCP \leq 3$ с, $LCP \leq 3$ с за метриками Google Lighthouse).
- Адаптивність: інтерфейс повністю адаптований для мобільних пристроїв.

- Безпека: JWT-токен зберігається в localStorage; при HTTP 401 – автоматичне очищення токена та перенаправлення на /login через axios response interceptor.
- Масштабованість: модульна архітектура React-компонентів дозволяє додавати нові сторінки та функції без зміни існуючого коду.
- Зручність: навігація між основними розділами – не більше двох кліків від головного екрана через нижню навігаційну панель MobileBottomNavbar.
- Сумісність: коректне відображення у браузерях Chrome 93+, Firefox 90+, Safari 14+, Edge 93+.

Окрім функціональних і нефункціональних вимог, при проектуванні інтерфейсу CalorieTracker визначено також вимоги до якості взаємодії з користувачем, що теж дуже впливають на архітектурні рішення клієнтської частини.

Найчастіші операції – додавання продукту до журналу та перегляд денного прогресу. Вони мають виконуватися не більш ніж у три кроки:

1. Відкрити застосунок;
2. Натиснути «Додати» у потрібній секції;
3. Знайти продукт та підтвердити.

Це вимагає розміщення кнопки додавання безпосередньо на картці кожного прийому їжі без додаткових переходів між сторінками.

У будь-який момент користувач повинен розуміти, скільки калорій залишилося до денної норми і наскільки виконано план по кожному макронутрієнту. Реалізація цієї вимоги – постійно видимий кільцевий індикатор та три кольорові картки КБЖВ на головному екрані, що оновлюються одразу після додавання продукту.

Будь-яка мережева операція має супроводжуватись індикатором завантаження, а у разі помилки – зрозумілим повідомленням. Це запобігає ситуаціям, коли користувач не розуміє, чи збереглися його дані.

Навігаційна панель розміщена внизу екрана, оскільки більшість користувачів тримають смартфон однією рукою. Кнопки мають мінімальний розмір 44×44 пікселі відповідно до рекомендацій Apple Human Interface Guidelines та Material Design.

2 ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ

2.1 Односторінковий застосунок та його складові

Клієнтська частина системи CalorieTracker реалізована у вигляді односторінкового застосунку. Це застосунок, у якому весь інтерфейс завантажується одноразово при першому зверненні до сервера, а подальша навігація між розділами виконується без повного перезавантаження сторінки. Замість цього браузер динамічно замінює лише ту частину DOM, яка змінилась, що суттєво прискорює роботу інтерфейсу та покращує загальний досвід взаємодії з додатком.

Такий підхід є особливо доцільним для мобільно-орієнтованих застосунків обліку харчування, де користувач часто переходить між розділами – щоденним журналом, статистикою, профілем та сторінкою продуктів – і очікує миттєвої реакції інтерфейсу на свої дії. Традиційні багатосторінкові застосунки (МРА) при кожному такому переході ініціювали б повний HTTP-запит та перемалювання сторінки, що призводило б до помітних затримок та мерехтіння контенту.

Застосунок CalorieTracker складається з таких основних структурних частин:

- Точка входу (index.html та main.tsx) – вхідний HTML-документ, який підключає JavaScript-бангл. При першому завантаженні браузер отримує порожній контейнер `<div id="root">`, у який React монтує весь інтерфейс застосунку.
- Кореневий компонент (App.tsx) – визначає структуру маршрутизації застосунку. Тут оголошені три типи маршрутів: публічні (Login, Register), захищені (Dashboard, Foods, Statistics, Profile, Achievements) та адміністративні (Admin). Захист маршрутів реалізовано через компоненти PrivateRoute та AdminRoute.

- Сторінки (pages/) – кожна сторінка є самодостатнім React-компонентом, що відповідає за завантаження даних та відображення відповідного екрану. Зокрема, Dashboard відповідає за щоденний журнал харчування, Statistics – за аналітику раціону, Profile – за налаштування профілю та відстеження ваги, Achievements – за систему досягнень, а Admin – за адміністрування системи.
- Компоненти (components/) – багаторазово використовувані елементи інтерфейсу, такі як AddFoodModal, BarcodeScannerModal, MealCard, TopHeader та MobileBottomNavbar. Кожен компонент є незалежним та може бути використаний на кількох сторінках.
- Сервісний шар (services/) – модуль api.ts містить усі функції для взаємодії з REST API серверної частини. Функції згруповані за ресурсами: authApi, mealsApi, foodsApi, usersApi, achievementsApi та adminApi.
- Типи та утиліти (types/, utils/) – TypeScript-інтерфейси, що описують структури даних та допоміжні функції для роботи з JWT-токенами, localStorage і перевірки прав доступу.

Навігація між сторінками реалізована засобами бібліотеки React Router DOM v6. Нижня навігаційна панель MobileBottomNavbar забезпечує перехід між основними розділами в один дотик, а компонент TopHeader відображає поточну сторінку та надає доступ до панелі сповіщень. Таким чином, структура застосунку забезпечує чіткий поділ відповідальностей між шарами та дозволяє незалежно розробляти, тестувати й розширювати кожен модуль.

2.2 Взаємодія клієнтської та серверної частин системи

Клієнтська та серверна частини системи CalorieTracker є повністю незалежними застосунками, що взаємодіють між собою через REST API у форматі JSON. Такий підхід відповідає принципу розділення відповідальностей: фронтенд відповідає виключно за відображення даних та обробку дій

користувача, тоді як бекенд забезпечує бізнес-логіку, збереження даних та їх обробку. Це надає можливість незалежно розвивати обидві частини системи.

Для виконання HTTP-запитів на стороні клієнта використовується бібліотека Axios. На відміну від вбудованого Fetch API, Axios надає зручний механізм перехоплювачів (interceptors), що дозволяє централізовано вирішувати наскрізні завдання – автоматичне додавання заголовка авторизації до кожного запиту та обробку відповідей з помилками автентифікації. Усі HTTP-функції зосереджені в одному модулі `services/api.ts`, що усуває дублювання логіки роботи з мережею у компонентах.

Автентифікація в системі реалізована на основі JWT-токенів (JSON Web Token). Після успішного входу сервер видає токен, який клієнт зберігає у `localStorage` браузера. При кожному наступному запиті `request-interceptor` Axios автоматично зчитує токен і додає його до заголовка `Authorization` у форматі `Bearer`. Якщо сервер повертає статус HTTP 401 (Unauthorized), `response-interceptor` очищає токен та перенаправляє користувача на сторінку входу – без будь-якої додаткової логіки у компонентах.

Обмін даними між клієнтом і сервером відбувається у форматі JSON. Кожен ресурс системи має відповідну групу API-функцій у сервісному шарі. Наприклад, `mealsApi.getDailyMeals(date)` виконує GET-запит для отримання усіх прийомів їжі за обрану дату разом із зведеними денними показниками, а `mealsApi.addFoodToMeal(mealId, data)` – POST-запит для додавання продукту до журналу. Усі відповіді сервера типізовані за допомогою TypeScript-інтерфейсів, що гарантує узгодженість структур даних між клієнтом і сервером.

Окрім взаємодії з власним бекендом, клієнтська частина також звертається до зовнішнього API для пошуку продуктів харчування. Запити до зовнішнього сервісу проксуються через серверну частину, тому клієнт взаємодіє лише з одним базовим URL. Це дозволяє приховати ключі доступу до зовнішнього API від браузера та спростити логіку клієнтського коду.

2.3 Архітектура клієнтської частини

Архітектура клієнтської частини системи CalorieTracker базується на компонентному підході, який є центральною ідеєю бібліотеки React. Відповідно до цього підходу, інтерфейс розбивається на невеликі, незалежні та повторно використовувані блоки – компоненти. Кожен компонент інкапсулює власну розмітку, стилі та логіку взаємодії з даними, що суттєво спрощує розробку, тестування та подальшу підтримку застосунку.

Управління станом застосунку реалізовано за допомогою вбудованих хуків React: `useState` та `useEffect`. Без застосування зовнішніх менеджерів стану. `useState` зберігає локальний стан компонента (завантажені дані, стан форм, прапори відображення модальних вікон тощо), а `useEffect` відповідає за виконання побічних ефектів – HTTP-запитів при монтуванні компонента або при зміні залежностей. Такий підхід є достатнім для поточного масштабу застосунку і не вимагає додаткових залежностей.

Маршрутизація між сторінками визначена у кореневому компоненті `App.tsx` за допомогою `React Router DOM v6`. Архітектурно маршрути поділено на три категорії. Публічні маршрути (`/login`, `/register`) доступні без автентифікації. Захищені маршрути (`/dashboard`, `/foods`, `/statistics`, `/profile`, `/achievements`) огорнуті компонентом `PrivateRoute`, який перевіряє наявність JWT-токену і у разі його відсутності перенаправляє на сторінку входу. Адміністративний маршрут (`/admin`) додатково перевіряє роль користувача через компонент `AdminRoute`.

Стилізація інтерфейсу виконана з використанням CSS-фреймворку `Tailwind CSS` за принципом `utility-first`: стилі застосовуються безпосередньо у `JSX`-розмітці через атомарні класи. Це прискорює розробку, виключає конфлікти між CSS-файлами та автоматично видаляє невикористані стилі у виробничій збірці через механізм `purging`. Застосунок розроблений за принципом `mobile-first`. Максимальна ширина контентної зони обмежена значенням `max-w-md` (448 пікселів), що забезпечує коректне і зручне відображення як на смартфонах, так і на десктопах.

Збірка та середовище розробки організовані за допомогою інструменту Vite. Vite використовує нативну підтримку ES-модулів браузером для миттєвого запуску dev-сервера та механізм HMR, завдяки якому зміни у коді відображаються в браузері без повного перезавантаження. Виробнича збірка виконується через esbuild з оптимізацією Rollup, що дає мінімальний та оптимізований бандл. Вихідна папка dist/ містить статичні файли, готові до розгортання на будь-якому статичному хостингу.

Таким чином, архітектура клієнтської частини забезпечує чіткий поділ між шарами представлення, логіки та взаємодії з сервером, підтримує зручне масштабування за рахунок компонентної структури та надає швидкий і адаптивний інтерфейс для кінцевого користувача.

3 ПРОЄКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Програмне забезпечення системи CalorieTracker реалізоване за клієнт-серверною архітектурою. У межах бакалаврської кваліфікаційної роботи розробляється клієнтська частина, SPA-застосунок на базі React 18, що взаємодіє з серверною частиною через REST API. Детальний опис архітектури та принципів взаємодії компонентів наведено у розділі 2.

Діаграма розгортання описує фізичну топологію системи і містить чотири вузли: Клієнтський Пристрій з браузером, де виконується CalorieTracker SPA та зберігається JWT-токен у localStorage; Веб Сервер з середовищем ASP.NET Core 8 Runtime, де розміщено CalorieTracker.Server.dll; Сервер БД з PostgreSQL та базою calorietracker_db; зовнішній FatSecret API-сервер, що надає REST API для пошуку продуктів за назвою та штрих-кодом через OAuth 2.0. Усі канали зв'язку захищено протоколом HTTPS. Діаграма розгортання представлена на рис. 7.

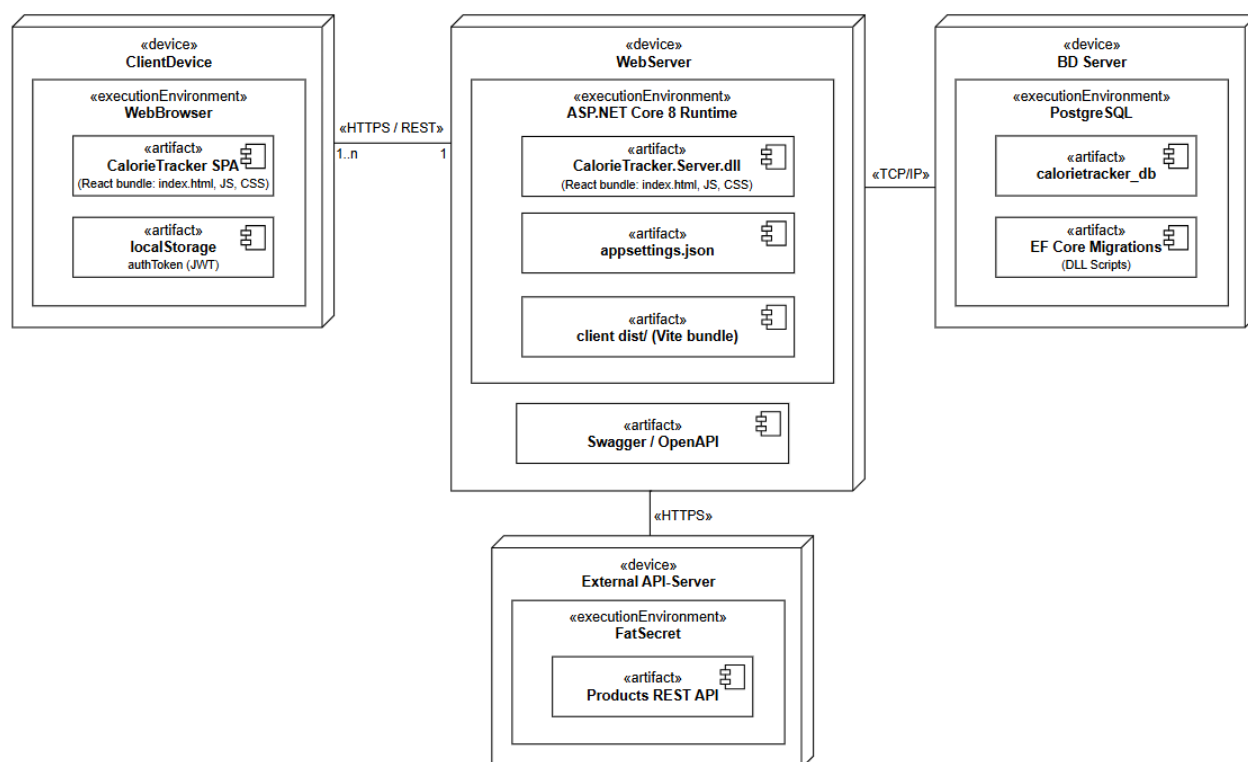


Рис. 7 Діаграма розгортання

Діаграма пакетів відображає організаційну структуру програмного забезпечення у вигляді єдиного рішення CalorieTracker.sln і зображена на рис. 8.

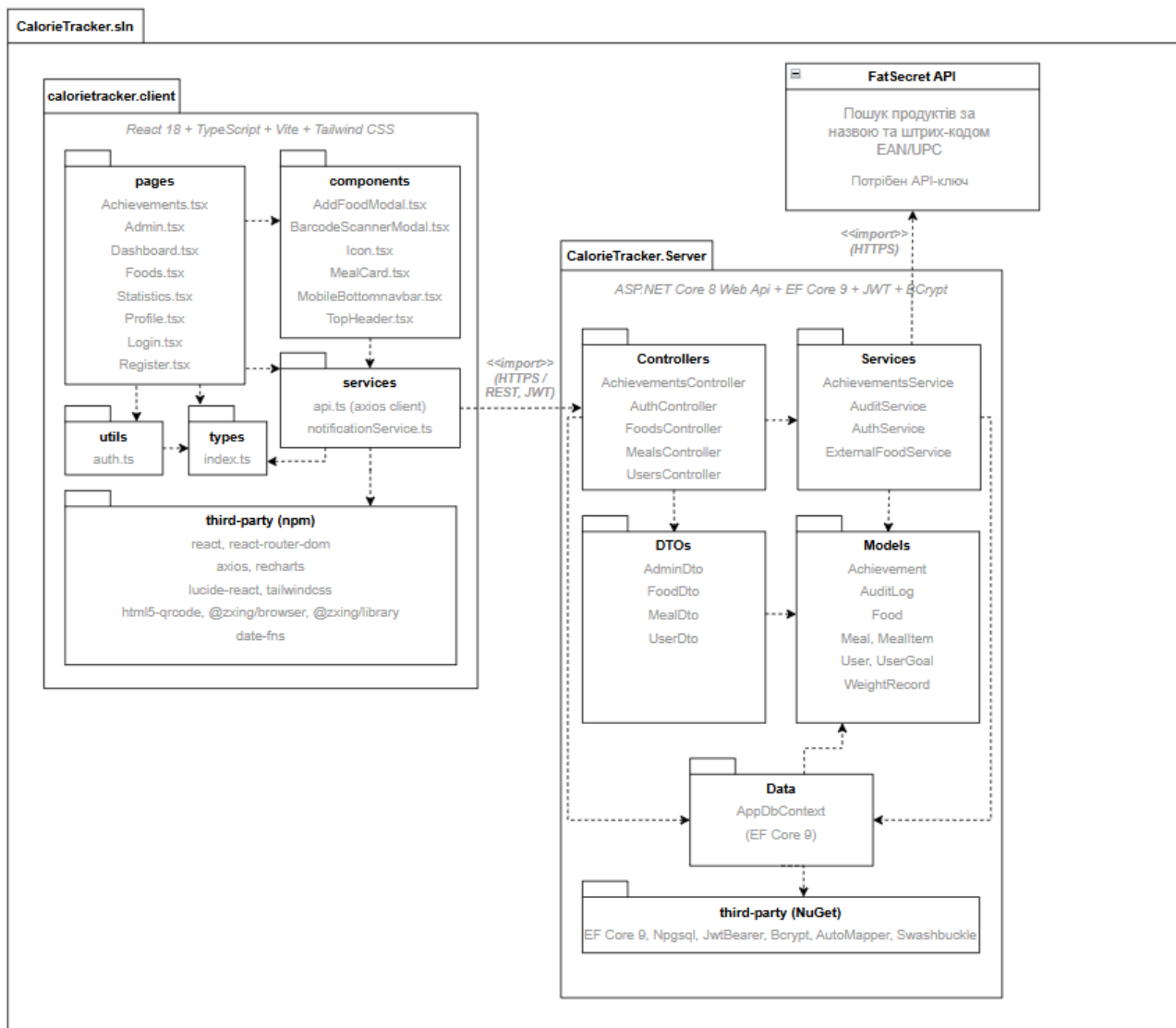


Рис. 8 Діаграма пакетів

Клієнтська частина **calorietracker.client** містить пакети **pages** (Dashboard, Foods, Statistics, Profile, Achievements, Login, Register), **services** (**api.ts** з **axios**-клієнтом та **JWT-interceptor**), **types** (TypeScript-інтерфейси) та **third-party npm** (react, recharts, html5-qrcode тощо). Серверна частина **CalorieTracker.Server** містить пакети **Controllers**, **Services**, **Models**, **DTOs**, **Data** та **third-party NuGet** (EF Core 9, Npgsql, JwtBearer, BCrypt). Взаємодія між частинами – через **HTTPS** з **JWT-автентифікацією**.

Організаційна структура клієнтської частини включає такі основні модулі:

- pages – сторінки застосунку (Dashboard, Foods, Statistics, Profile, Achievements, Login, Register, Admin);
- components – UI-компоненти (AddFoodModal, BarcodeScannerModal, MealCard, TopHeader, MobileBottomNavbar, Icon), які використовуються багаторазово;
- services – модуль взаємодії з серверною частиною api.ts, що містить функції HTTP-запитів, згруповані за ресурсами (authApi, mealsApi, foodsApi, usersApi, achievementsApi, adminApi);
- utils – допоміжні функції для роботи з JWT-токенами, localStorage, повідомленнями системи та перевіркою прав доступу;
- types – TypeScript-інтерфейси та enum-типи для опису DTO-моделей і структури даних.

Такий підхід дозволяє зберігати чисту архітектуру проєкту, уникати дублювання коду, спрощує тестування окремих модулів та полегшує подальшу підтримку системи.

3.2 Вибір інструментарію для розробки клієнтської частини

Вибір технологічного стеку для клієнтської частини здійснювався з урахуванням вимог до продуктивності, підтримуваності коду та зручності розробки. Список стеку клієнтської частини наведений у таблиці 3.1

Таблиця 3.1

Технологічний стек клієнтської частини

Технологія	Версія	Призначення
React	18.3.1	Компонентна бібліотека для побудови UI
TypeScript	5.6.3	Статична типізація JavaScript
Vite	5.4.10	Засіб збірки з HMR

Технологія	Версія	Призначення
Tailwind CSS	3.4.14	Utility-first CSS-фреймворк
React Router DOM	6.26.2	Клієнтська маршрутизація
Axios	1.7.7	HTTP-клієнт з interceptor-ами
Recharts	3.8.1	SVG-графіки на основі React
html5-qrcode	2.3.8	Сканування QR та штрих-кодів
@zxing/browser	0.1.5	Альтернативний декодер штрих-кодів
date-fns	4.1.0	Утиліти роботи з датами
lucide-react	0.446.0	Бібліотека SVG-іконок

React 18 обрано як основну бібліотеку завдяки компонентній архітектурі, декларативному підходу до опису UI та широкій екосистемі. Хуки `useState` та `useEffect` забезпечують управління локальним станом компонента і виконання побічних ефектів (HTTP-запити при монтуванні або зміні залежностей) без необхідності в зовнішньому state-менеджері.

TypeScript надає статичну типізацію, що дозволяє виявляти помилки на етапі компіляції. Усі DTO серверної частини відображені у файлі `src/types/index.ts` у вигляді TypeScript-інтерфейсів (`User`, `Food`, `Meal`, `MealItem` тощо), що гарантує узгодженість між клієнтом і сервером.

Vite забезпечує миттєвий запуск dev-сервера завдяки нативній підтримці ES-модулів та механізму Hot Module Replacement (HMR). Продакшн-збірка будується через `esbuild` з подальшою оптимізацією Rollup, що дає мінімальний розмір бандлу.

Tailwind CSS реалізує підхід utility-first: стилі застосовуються через атомарні класи безпосередньо в JSX-розмітці (`flex`, `p-4`, `text-green-500` тощо). Це прискорює розробку, усуває конфлікти стилів та автоматично видаляє невикористані правила у фінальній продакшн-збірці через механізм purging.

Axios налаштовано з двома interceptor-ами. Request-interceptor автоматично додає заголовок `Authorization: Bearer {token}` до кожного запиту, зчитуючи токен з `localStorage`. Response-interceptor перехоплює відповіді зі статусом 401 (Unauthorized), видаляє токен та перенаправляє користувача на сторінку входу.

Налаштування interceptor-ів реалізовано у файлі `services/api.ts`. Request interceptor перехоплює кожен вихідний запит та автоматично додає JWT-токен з `localStorage` у заголовок `Authorization`. Фрагмент коду request interceptor наведено на рис 9.

```
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('authToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Рис. 9 Фрагмент коду налаштування request interceptor в Axios

Response interceptor перехоплює вхідні відповіді та обробляє помилку автентифікації: при отриманні статусу HTTP 401 токен та дані користувача видаляються з `localStorage`, після чого виконується автоматичне перенаправлення на сторінку входу. Фрагмент коду response interceptor наведено на рис 10.

```
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('authToken');
      localStorage.removeItem('user');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

Рис. 10 Фрагмент коду налаштування response interceptor в Axios

Це централізує логіку автентифікації в одному місці та звільняє компоненти від необхідності самостійно перевіряти наявність токена при кожному запиті. Повний лістинг файлу `api.ts` наведено у Додатку Б.

3.3 Алгоритмізація та програмування основних модулів

3.3.1 Модуль автентифікації. Процес входу в систему реалізовано у компоненті `Login.tsx`. Після введення облікових даних клієнт надсилає POST-запит на `/api/auth/login` через `authApi.login()`. При успішній відповіді (HTTP 200) JWT-токен та об'єкт користувача зберігаються у `localStorage` за допомогою функцій `setAuthToken()` та `setCurrentUser()` з модуля `utils/auth.ts`. Далі відбувається перенаправлення на `/dashboard`. Повний лістинг файлу `auth.ts` надано у Додатку Б.

Процес збереження сесії користувача побудований на основі JWT-токенів. Після успішного входу система отримує токен і зберігає його у локальному сховищі браузера (`localStorage`). Це дозволяє користувачу не вводити пароль заново при кожному оновленні сторінки. Щоб не додавати цей токен вручну до кожного нового запиту, використовується спеціальний механізм перехоплювачів (`interceptors`) у HTTP-клієнті `Axios`. Він автоматично прикріплює токен до заголовків усіх вихідних запитів до сервера. Якщо час дії токена закінчився, сервер повертає помилку доступу (статус 401). У такому випадку система автоматично очищає локальне сховище та перенаправляє користувача назад на сторінку входу. Це забезпечує надійний захист особистих даних.

3.3.2 Модуль підрахунку КБЖВ. Підрахунок денних показників КБЖВ реалізований на стороні серверної частини. Клієнт отримує вже обчислені значення у готовому вигляді. При відкритті сторінки `Dashboard` або зміні дати виконується GET-запит через `mealsApi.getDailyMeals(date)`, який повертає об'єкт типу `DailyMeals` з двома ключовими полями: масивом прийомів їжі `meals` та зведеним об'єктом `summary` з денними показниками.

На основі отриманих даних компонент `Dashboard.tsx` обчислює та відображає показники денної норми калорій. Фрагмент коду розрахунку показників зображено на рис. 11.

```
const remainingKcal = Math.max(goalKcal - consumedKcal, 0);
const progressPct = Math.min((consumedKcal / Math.max(goalKcal, 1)) * 100, 100);
```

Рис. 11. Логіка розрахунку показників денної норми калорій в компоненті `Dashboard.tsx`

`goalKcal` – денна ціль калорій з профілю користувача, `consumedKcal` – фактично спожита кількість за день. Використання `Math.max(goalKcal, 1)` запобігає діленню на нуль у випадку, якщо денна ціль ще не встановлена. Значення `progressPct` використовується для відображення кругового прогрес-індикатора, а `remainingKcal` – для картки «Залишилось». Аналогічно розраховується прогрес для білків, жирів та вуглеводів відносно відповідних денних цілей з `UserGoals`.

Кожна секція прийому їжі відображає підсумкову калорійність, що є сумою калорій усіх `MealItem` у межах цього прийому. Ці значення також надходять у готовому вигляді від сервера – клієнт не виконує власних арифметичних обчислень з харчовими даними. Лістинг компонента `Dashboard.tsx` наведено у Додатку В.

3.3.3 Модуль пошуку продуктів. Компонент `AddFoodModal.tsx` реалізує три режими пошуку продуктів: пошук за назвою, сканування штрих-коду та вибір із власних продуктів. Між ними користувач перемикається за допомогою вкладок у верхній частині модального вікна, а також іконки штрих-коду.

Пошук за назвою є основним режимом. Користувач вводить назву продукту в текстове поле, після чого компонент формує GET-запит до ендпоінту `/api/foods/external/search` з параметрами `query` та `source`. Щоб уникнути надмірної кількості запитів при швидкому введенні тексту, реалізовано затримку у 800 мілісекунд. Запит до сервера відправляється лише тоді, коли користувач припинив введення на цей час. Результати з'являються саме в момент, коли користувач очікує на них, а не після кожного натискання клавіші.

Результати пошуку відображаються у вигляді прокручуваного списку карток. Кожна картка містить назву продукту та скорочений КБЖВ на 100 г. При виборі продукту з'являється форма введення кількості у грамах з числовим полем і кнопкою підтвердження. Після підтвердження компонент викликає `mealsApi.addFoodToMeal()` і закривається, а батьківський компонент `Dashboard.tsx` перезавантажує денні дані.

Пошук за штрих-кодом реалізовано у дочірньому компоненті `BarcodeScannerModal.tsx`. При відкритті вкладки браузер запитує дозвіл на використання камери. Після підтвердження ініціалізується об'єкт `BrowserMultiFormatReader` з бібліотеки `@zxing/browser`, який безперервно аналізує кадри з відеопотоку у пошуку штрих-кодів. Розпізнавання відбувається повністю на стороні браузера, без передачі відеопотоку на сервер. Це забезпечує конфіденційність і не потребує мережевого з'єднання на етапі сканування.

Як тільки штрих-код розпізнано, сканування зупиняється і числовий рядок передається у запит `GET /api/foods/external/barcode/{barcode}`. Сервер проксує запит до зовнішньої бази і повертає клієнту об'єкт з назвою продукту та показниками КБЖВ. Якщо продукт за штрих-кодом не знайдено, компонент відображає відповідне повідомлення і пропонує перейти до текстового пошуку або створити продукт вручну. Це виключає ситуації, коли користувач не може знайти потрібний товар.

Третій спосіб пошуку – серед власних продуктів. Список власних продуктів завантажується одноразово при відкритті вкладки і фільтрується на стороні клієнта при введенні тексту без додаткових звернень до сервера. Це дозволяє користувачеві швидко знаходити продукти, які він регулярно вживає і вже вніс у систему раніше, без необхідності щоразу шукати їх у зовнішній базі.

3.3.4 Модуль статистики. Компонент `Statistics.tsx` отримує агреговані дані через GET-запит `/api/users/statistics?days={N}`, де `N` – обраний період (7, 14, 30 або 90 днів). Сервер повертає масив денних записів з датою та сумарними

показниками КБЖВ, а також середні значення за весь період. Клієнт передає ці дані до компоненту `Recharts LineChart` для відображення тренду калорій.

Процес відображення статистичних графіків: коли користувач відкриває сторінку статистики, система відправляє запит на сервер для отримання даних за обраний період (за замовчуванням 7 днів). Після отримання відповіді від сервера, дані зберігаються у внутрішньому стані сторінки за допомогою `useState`. Тоді дані автоматично передаються у бібліотеку `Recharts`, яка відповідає за відображення графіків на екрані. Якщо користувач натискає на перемикач і обирає інший період, попередні дані очищаються, відправляється новий запит, а графік переробляється заново відповідно до нової інформації. Лістинг компонента `Statistics.tsx` наведено у Додатку В.

3.3.5 Модуль досягнень. Підсистема досягнень реалізована як елемент гейміфікації, що стимулює користувача до регулярного ведення журналу харчування. Логіка перевірки умов розблокування реалізована на стороні серверної частини у класі `AchievementService`, клієнтська частина відповідає виключно за відображення результатів.

Усього в системі визначено 15 досягнень, згрупованих у п'ять категорій. Категорія «Харчування» містить досягнення за кількість прийомів їжі: від першого запису до 100 прийомів. Категорія «Активність» нагороджує за кількість днів з хоча б одним записом – 7 та 30 днів відповідно. Категорія «Серія» відстежує безперервні дні ведення щоденника: 3, 7 та 14 днів поспіль. Категорія «Цілі» нараховує досягнення за дотримання денної норми калорій – перше виконання, 5 та 20 разів. Категорія «Вага» заохочує до відстеження динаміки ваги, а «Продукти» – до додавання власного продукту.

При відкритті сторінки `Achievements.tsx` виконується GET-запит через `achievementsApi.getAll()`, який повертає повний список усіх 15 досягнень з відміткою `isUnlocked` для кожного. Досягнення відображаються згрупованими за категоріями. Розблоковані виділяються кольоровим фоном та зеленою

позначкою із датою отримання, заблоковані відображаються тьмяним кольором із хрестиком.

Прогрес відображається у вигляді прогрес-бару з лічильником розблокованих досягнень. При натисканні кнопки «Перевірити нові досягнення» виконується POST-запит `achievementsApi.check()`, сервер аналізує поточну статистику користувача та повертає список щойно розблокованих досягнень, які відображаються у виділеному банері. Повний лістинг компонента `Achievements.tsx` показано у Додатку В.

3.4 Опис інтерфейсу користувача

Клієнтська частина `CalorieTracker` реалізована як мобільно-орієнтований SPA з фіксованою нижньою навігаційною панеллю та верхнім хедером. Максимальна ширина контентної зони – 448 пікселів (`max-w-md`), що забезпечує коректне відображення на мобільних пристроях.

Система сповіщень – важливий елемент загального інтерфейсу, який інтегрований у хедер. При натисканні на іконку сповіщень відкривається спливаюча панель (рис. 12), де відображається список останніх системних подій та підказок для користувача.

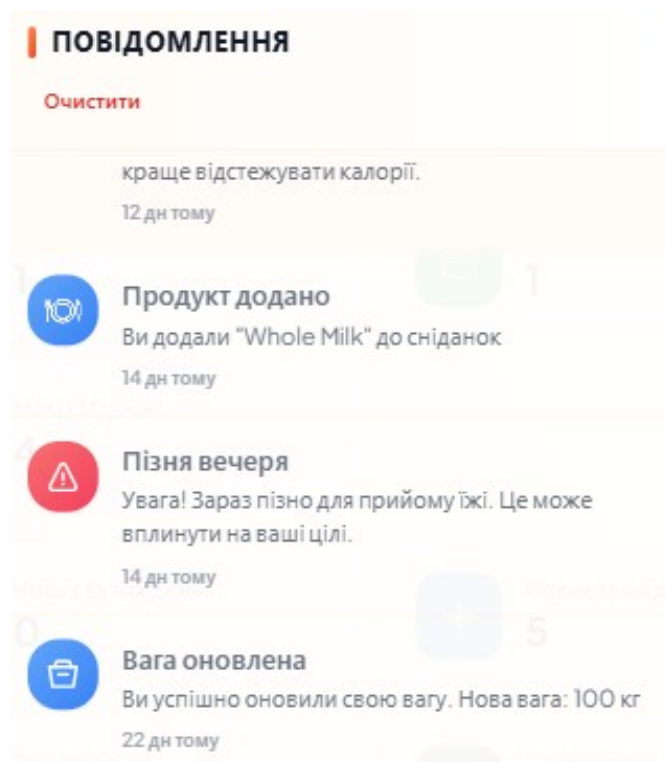


Рис. 12 Панель сповіщень користувача

Система сповіщень інформує та попереджує користувача. Вона повідомляє про успішне додавання продукту до журналу, оновлення показників ваги, а також надсилає автоматичні попередження. Як приклад, про пізню вечерю чи час для сніданку.

Усі сповіщення містять час, що минув з моменту події. Користувач легко відстежує свою активність і отримує зворотний зв'язок від системи. При цьому не потрібно переходити на інші сторінки. Для зручності передбачена кнопка «Очистити». Вона швидко видаляє всю історію прочитаних повідомлень.

Перед реалізацією інтерфейсу розроблено візуальний макет застосунку у сервісі для прототипування Figma. Це дозволило до початку написання коду визначити загальну стилістику інтерфейсу, колірну схему, типографіку та розташування елементів на кожному екрані. Скріншоти всіх сторінок застосунку, крім наявних в цьому розділі, наведено у Додатку Б.

3.4.1 Сторінка входу та реєстрації. Неавторизований користувач потрапляє на сторінку Login.tsx, де розміщено форму з полями Email та Пароль, а також посилання на сторінку реєстрації. Реалізовану базову клієнтську

валідацію: обидва поля є обов'язковими, при порожньому значенні відображається підказка. При успішному вході JWT-токен зберігається у localStorage та відбувається перенаправлення на /dashboard. У разі помилки (HTTP 401) відображається повідомлення «Невірний email або пароль»

Сторінка Register.tsx містить розширену форму: ім'я, email, пароль, дата народження, стать та рівень фізичної активності. (Від «Малорухливий» до «Дуже активний»). Рівень активності впливає на розрахунок денної норми калорій через коефіцієнт PAL. Обидві сторінки захищені PublicRoute-компонентом: авторизований користувач автоматично перенаправляється на /dashboard.

Скріншоти сторінок логіну та реєстрації зображені на рис. 13.

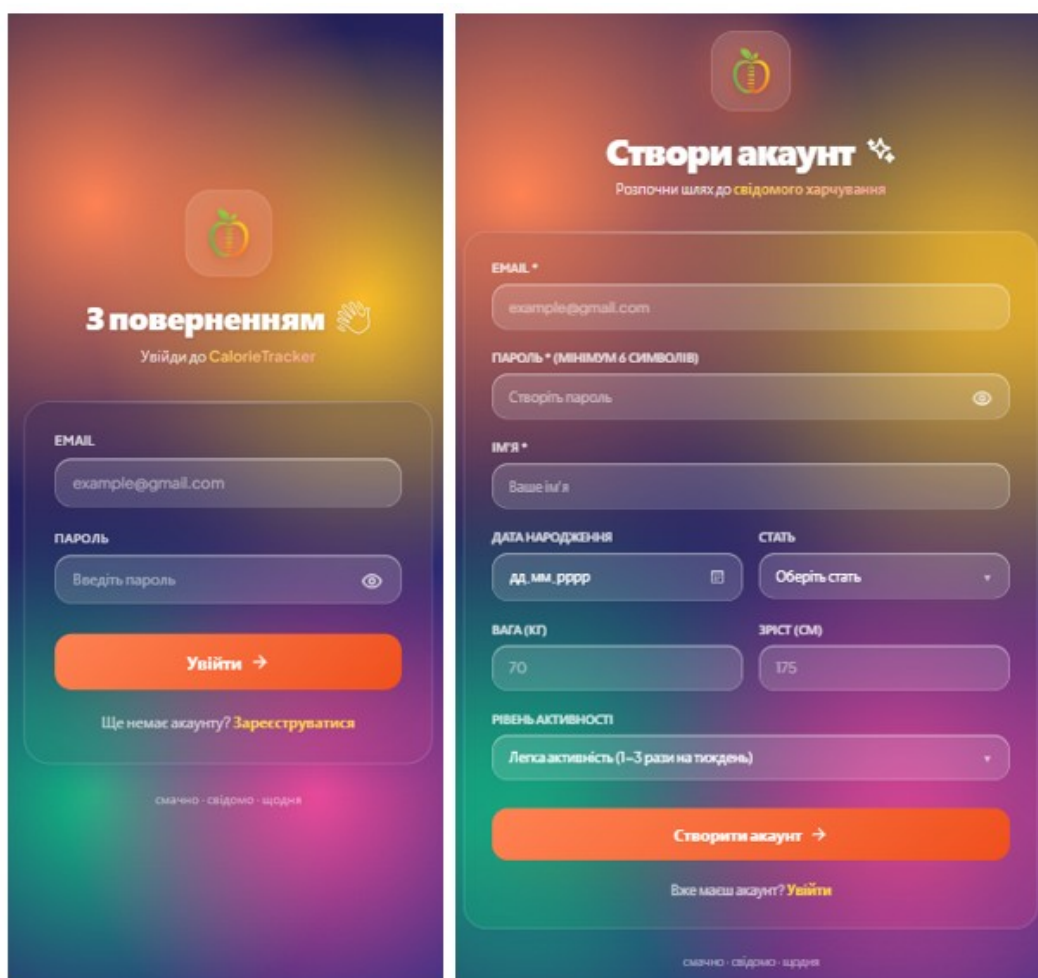


Рис. 13 Сторінки входу в систему CalorieTracker

3.4.2 Головна сторінка (щоденний журнал). Головна картка центрального екрану застосунку (рис. 14) містить привітання користувача та

перемикачі дати. У верхній частині відображається круговий прогрес-індикатор денної норми калорій із числовим значенням спожитого та залишку до виконання денної норми. Нижче розміщено три картки макронутрієнтів (білки, жири, вуглеводи) з прогрес-барами відносно денних цілей.

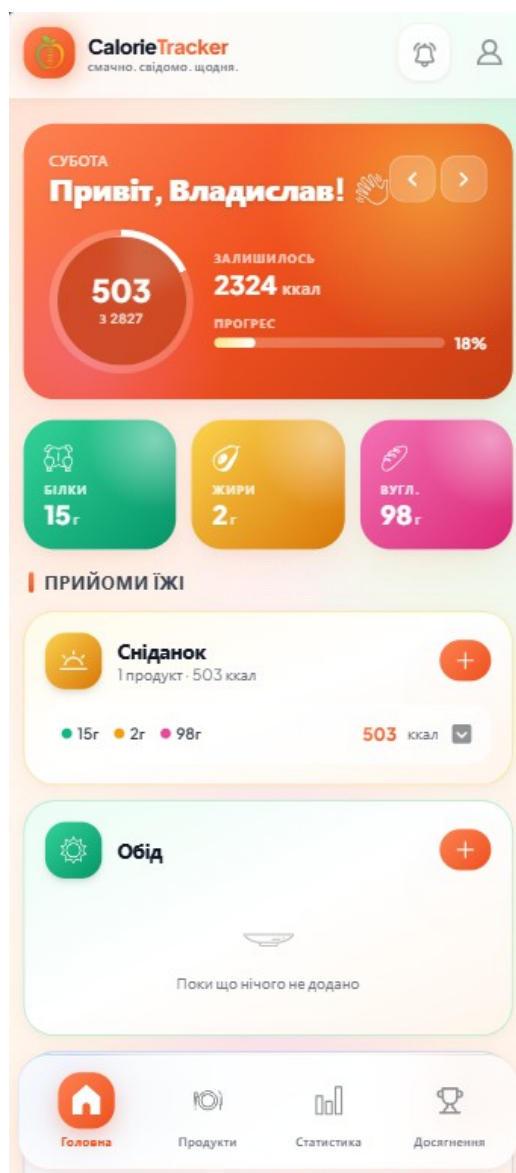


Рис. 14 Головна сторінка – щоденний журнал харчування

Основну частину екрана займають чотири секції прийомів їжі: Сніданок, Обід, Вечеря та Перекус. Кожна секція відображає список доданих продуктів із назвою, кількістю в грамах та розрахованою калорійністю. При натисканні на продукт відкривається модальне вікно редагування кількості. Кнопка видалення прибирає запис із журналу. Кнопка «Додати» відкриває компонент

AddFoodModal, зображений на рис. 15. Містить дві вкладки пошуку: «Пошук у базі» та «Мої продукти».

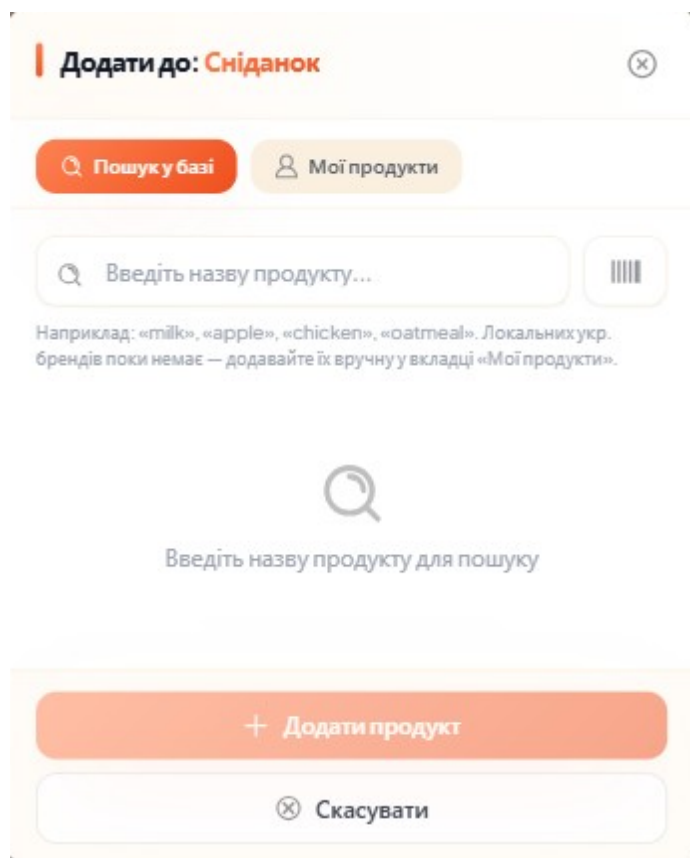


Рис. 15 Модальне вікно AddFoodModal для додавання продуктів

На вкладці «Пошук у базі» користувач вводить назву продукту в текстове поле. Кожен результат відображає назву, бренд, калорійність на 100 г та значення КБЖВ. Праворуч від поля вводу розміщено кнопку сканування штрих-коду – при натисканні відкривається BarcodeScannerModal, який активує камеру пристрою. Після успішного зчитування штрих-коду знайдений продукт автоматично обирається без необхідності пошуку вручну. На вкладці «Мої продукти» відображається список власних продуктів користувача з фільтрацією на стороні клієнта.

Після вибору продукту з будь-якої вкладки відображається форма введення кількості в грамах із попереднім розрахунком КБЖВ у реальному часі. Після успішного додавання автоматично ініціюється перевірка нових досягнень.

3.4.3 Сторінка управління продуктами. Сторінка `Foods.tsx` надає користувачу контроль над власною базою продуктів. У верхній частині знаходиться текстове поле пошуку. При введенні запиту список фільтрується в реальному часі без додаткових HTTP-запитів, оскільки фільтрація відбувається на стороні клієнта по завантажених даних.

Кожен продукт відображається у вигляді картки з назвою, категорією та чотирма числовими значеннями калорій, білків, жирів та вуглеводів на 100 г. Кнопки редагування та видалення з'являються при натисканні на картку. Операція видалення вимагає підтвердження, щоб запобігти випадковому вилученню даних.

Кнопка «Додати продукт» відкриває модальну форму з полями назви, категорії та числових показників КБЖВ. Форма містить валідацію. Усі числові поля допускають лише невід'ємні значення, а поле назви є обов'язковим. Після успішного збереження список оновлюється автоматично. Скріншот сторінки наведено у Додатку А (рис. А.1-А.2).

3.4.4 Сторінка статистики. Сторінка статистики надає користувачу аналітичний огляд раціону за обраний часовий діапазон. У верхній частині картки розміщено перемикач діапазону з чотирма варіантами: 7 днів, 14 днів, 30 днів та 3 місяці. При зміні діапазону виконується новий GET-запит через `usersApi.getStatistics(days)`, сервер агрегує дані журналу харчування за відповідний період та повертає масив денних записів і зведені середні значення.

Нижче перемикача розміщено чотири кольорові картки середніх показників за обраний період: середня калорійність на день з відображенням денної цілі, середнє споживання білків, жирів та вуглеводів у грамах (рис. 16).

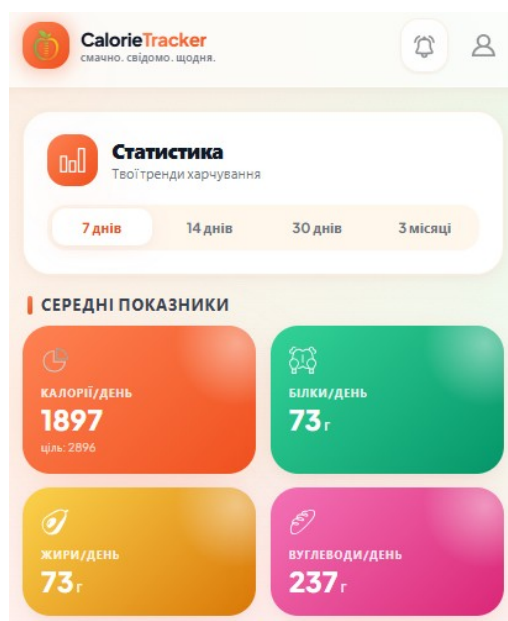


Рис. 16 Вибір діапазону статистики та середні показники сторінки
«Статистика»

Центральним елементом сторінки є лінійний графік тренду калорій, побудований засобами бібліотеки Recharts (компонент LineChart). Вісь X відображає дати обраного діапазону, вісь Y – калорійність за кожен день. На графіку нанесено горизонтальну пунктирну референсну лінію з денною ціллю калорій, що дозволяє візуально оцінити скільки днів раціон відповідав нормі, а скільки був нижче або вище цілі. При наведенні виборі точки графіку відображається підказка з точним значенням калорій за обраний день.

Нижче графіку розміщено секцію «Розподіл макронутрієнтів» з кільцевою діаграмою, яка відображає співвідношення білків, жирів та вуглеводів у середньодобовому раціоні. Кожен сектор підписано із зазначенням середньої маси в грамах та відсотком від загального споживання. Під кільцевою діаграмою розміщено стовпчасту діаграму «Макроси по днях», де кожен стовпець відображає денне споживання макронутрієнтів у вигляді кольорових сегментів. Ці елементи зображені на рис. 17.

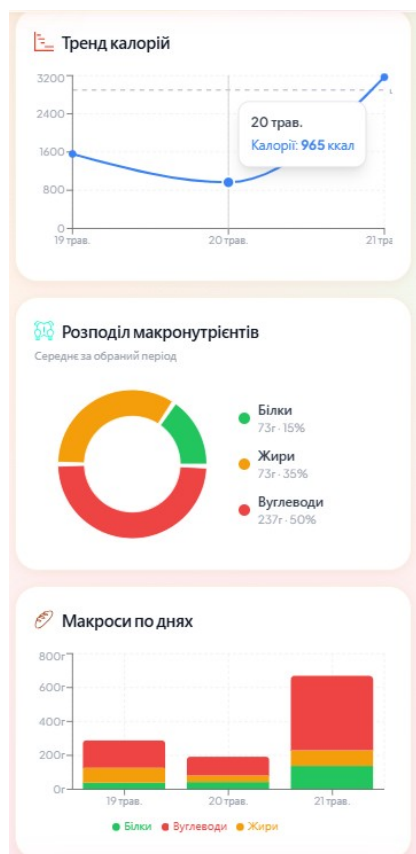


Рис. 17 Графіки тренду калорій, розподілу макронутрієнтів і макросів по днях сторінки Статистика

Далі знаходяться ще три секції: «Прогрес до цілей» містить два прогрес-бари: для калорій (фактичне середнє відносно денної цілі) та для білків (середнє споживання відносно рекомендованої норми). «Активність» відображає кількість активних днів (днів з хоча б одним записом у журналі), прийомів їжі за обраний період та середню кількість прийомів на день. А в кінці сторінки статистики розташована секція «Поради». Це динамічний блок рекомендацій, що формується на стороні клієнта згідно завантажених статистичних даних. Логіка формування порад реалізована безпосередньо у компоненті Statistics.tsx.

Якщо середня денна калорійність за обраний період становить менше 80% від цільової норми – відображається попередження про недостатнє споживання калорій, якщо середнє споживання білка нижче 80 г – пропонується збільшити частку білкових продуктів, а якщо регулярність ведення щоденника перевищує 80% днів обраного діапазону – відображається мотиваційне повідомлення. Такий

підхід дозволяє надавати персоналізовані рекомендації без додаткових запитів до сервера. (рис. 18)

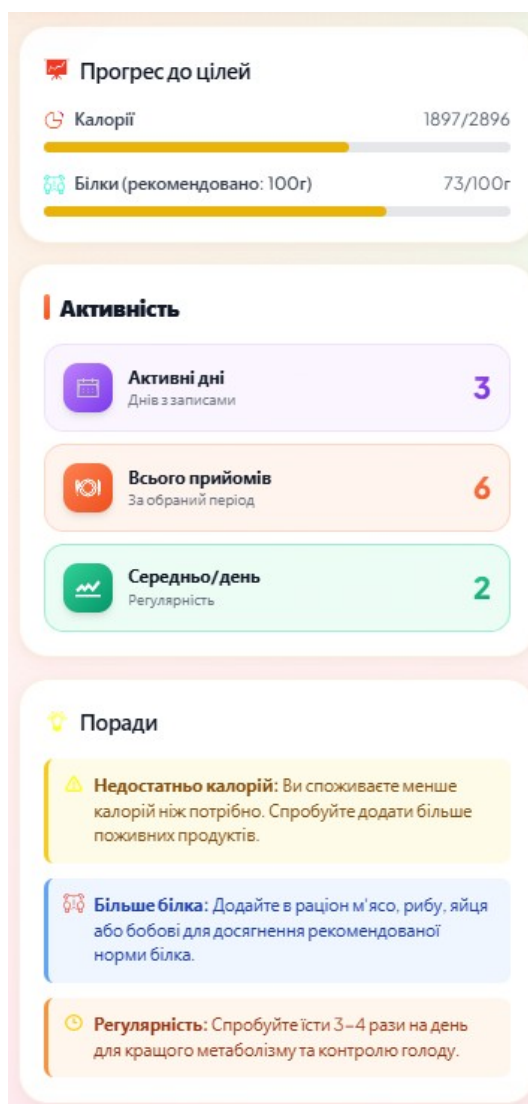


Рис. 18 Секції прогресу, активності та порад сторінки Статистика

3.4.5 Сторінка профілю та відстеження ваги. Сторінка профілю складається з двох вкладок – «Профіль» і «Вага». У верхній частині сторінки можна завантажити фото користувача, яке після можна змінити або видалити. Також тут зображені ім'я користувача та електронна пошта (рис. 19). Завантаження фото реалізовано через запит `usersApi.updateAvatar()` з передачею файлу у форматі `multipart/form-data`.

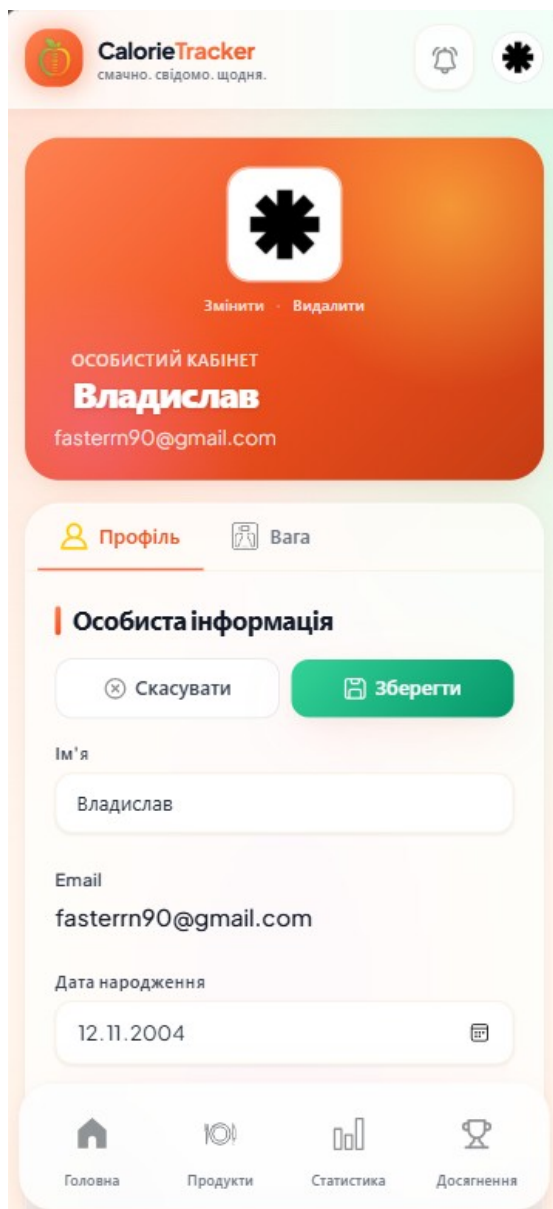


Рис. 19 Сторінка профілю, вкладка «Профіль»

Вкладка «Профіль» містить особисту інформацію користувача: ім'я, електронну пошту, дату народження, стать, вагу, зріст та рівень активності. При натисканні кнопки «Редагувати» поля переходять у режим редагування, після збереження виконується PUT-запит через `usersApi.updateProfile()`. У нижній частині вкладки відображається розрахована денна норма калорій та ВМІ з відповідною категорією за шкалою ВООЗ (рис. 20).

Дата народження

12.11.2004

Стать

Чоловіча

Вага (кг)

95

Зріст (см)

201

Рівень активності

Легка активність (1–3 рази на тиждень)

Денна норма калорій

2896 ккал

Розраховується автоматично з ваших параметрів

Індекс маси тіла (BMI)

23.5

Нормальна вага

Icon of a person on a scale

Рис. 20 Особиста інформація, денна норма калорій та індекс маси тіла на сторінці профілю

Вкладка «Вага» містить форму для додавання нового запису ваги та хронологічний список попередніх вимірювань з датою та значенням. При збереженні виконується POST-запит через `usersApi.addWeightRecord()` (рис. 21).

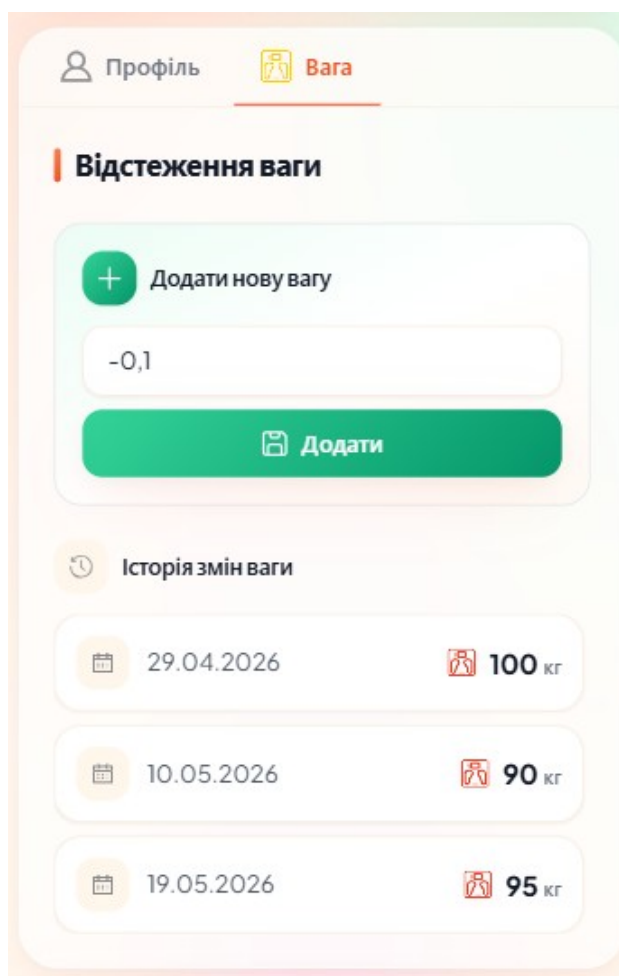


Рис. 21 Вкладка «Вага» з формою та історією вимірювань

3.4.6 Сторінка досягнень. Сторінка досягнень відображає загальний прогрес у вигляді прогрес-бару з лічильником розблокованих досягнень та кнопкою перевірки нових досягнень. При натисканні кнопки система перевіряє умови та відображає банер з переліком щойно розблокованих досягнень, який закривається вручну. Нижче досягнення згруповані за категоріями – розблоковані відображаються з кольоровою іконкою та датою отримання, заблоковані – тьмяним кольором (рис. 22).

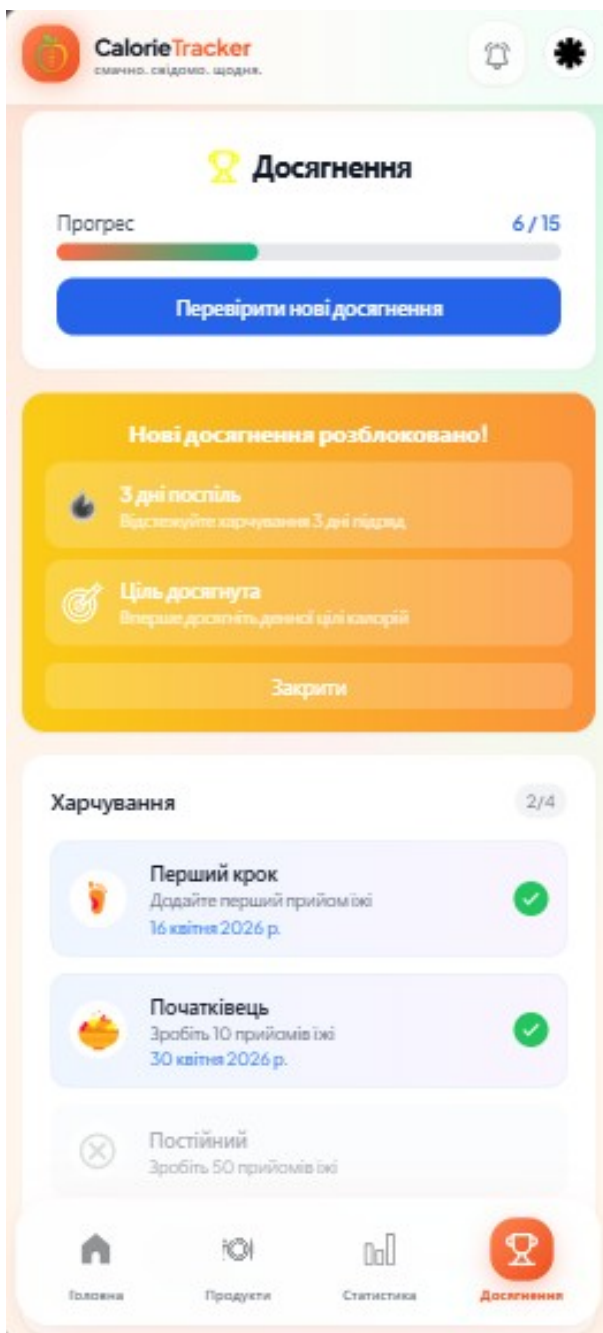


Рис. 22 Сторінка досягнень користувача

3.4.7 Адміністративна панель. Доступна користувачам, які мають роль адміністратора. Складається з п'ятих вкладок. Вкладка «Огляд» відображає зведену статистику системи: активні користувачі за 24 години та за тиждень, MAU (місячна аудиторія за 30 днів), нові користувачі за тиждень та місяць, загальна кількість прийомів їжі та власних продуктів. Також відображається стовпчастий графік реєстрацій за останні 30 днів (рис. 23).

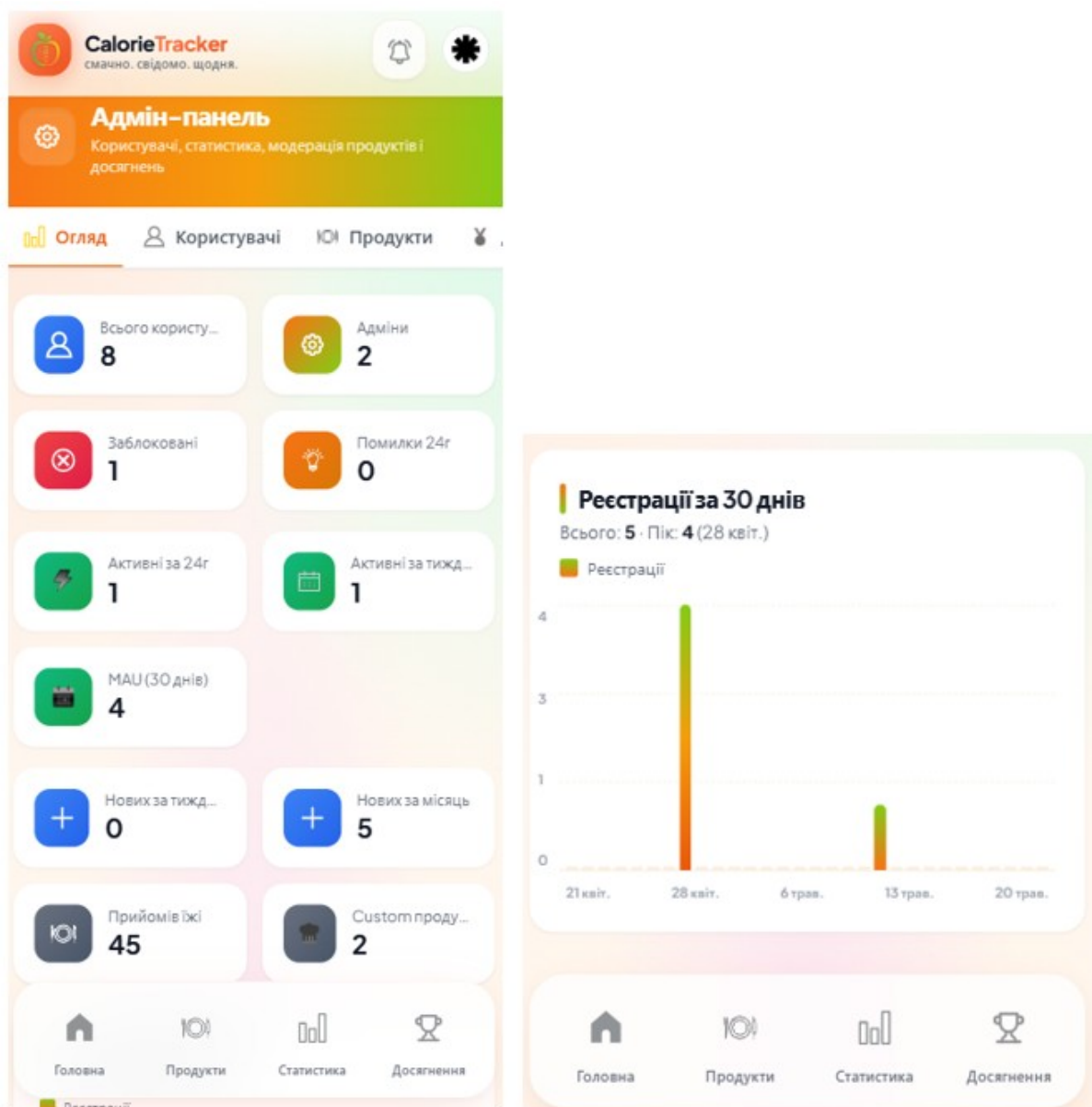


Рис. 23 Вкладка «Огляд» адміністративної панелі

Вкладка «Користувачі» містить таблицю всіх облікових записів з пошуком за email або іменем та фільтрами за роллю і статусом. Для кожного користувача відображається ім'я, email, роль, дата реєстрації, остання активність, кількість прийомів їжі та статус. При натисканні «Перегляд» відкривається модальне вікно з детальною інформацією про користувача: антропометричні дані, кількість досягнень, власних продуктів та записів ваги. У нижній частині вікна розміщено поле для вказання причини блокування та кнопки «Заблокувати»/«Розблокувати» і «Видалити» (рис. 24).

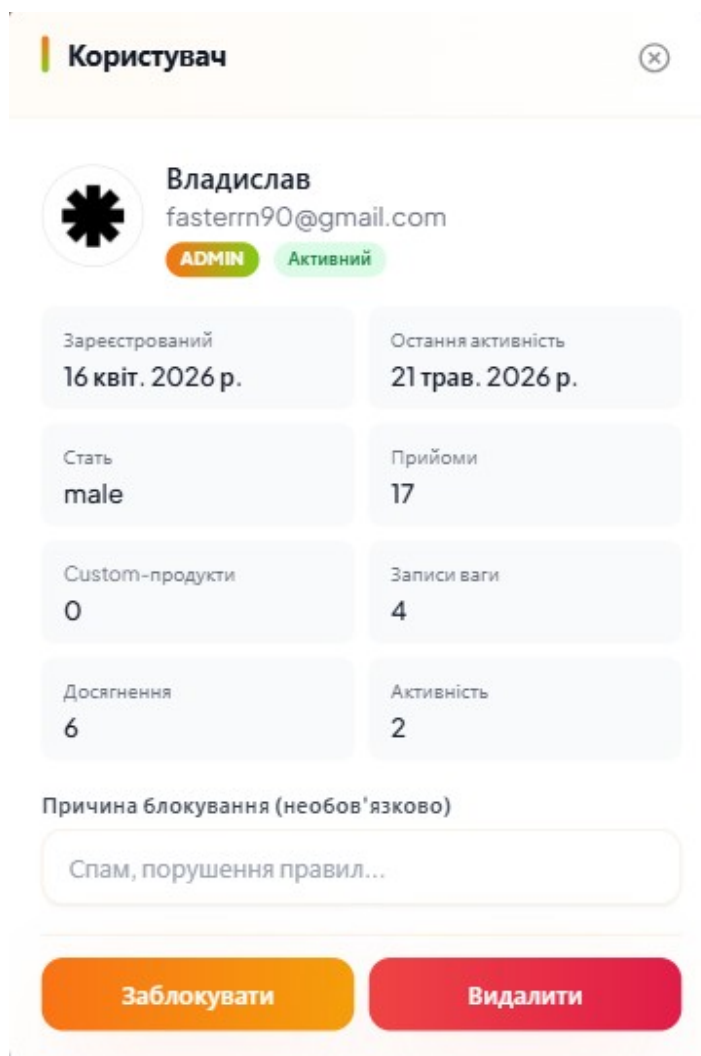


Рис. 24 Модальне вікно перегляду облікового запису користувача

Вкладка «Продукти» містить таблицю власних продуктів користувачів з пошуком за назвою або брендом, відображає автора, калорійність та макронутрієнти, а також кнопки редагування і видалення. Вкладка «Досягнення» відображає перелік усіх визначених досягнень із кількістю користувачів що їх отримали та можливістю нагородити вручну. Вкладка «Логи» містить журнал системних подій з фільтрами за рівнем та типом дії, де кожен запис відображає час, рівень, тип події та актора. Скріншоти вкладок Користувачі, Продукти, Досягнення та Логи наведено у Додатку А (рис. А.3-А.6).

4 ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Тестування клієнтської частини

Тестування клієнтської частини CalorieTracker охопило кілька видів: функціональне тестування сценаріїв використання, тестування інтерфейсу та тестування продуктивності.

Функціональне тестування проводилось шляхом перевірки входу/виходу із системи, реєстрації з різними комбінаціями даних, додавання та видалення продуктів з журналу, пошуку за назвою та штрих-кодом, відображення статистики за різні діапазони дат, коректності розрахунку залишку калорій відносно денної цілі тощо. Всі перевірені сценарії завершилися успішно. Результати наведено у табл. 4.1

Таблиця 4.1

Результати функціонального тестування клієнтської частини

№	Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
1	Реєстрація з коректними даними	Валідні email, пароль, ім'я	Перехід на /dashboard	Пройшов
2	Вхід з невірним паролем	Коректний email, невірний пароль	Повідомлення про помилку	Пройшов
3	Вхід з коректними даними	Валідні email та пароль	JWT збережено, перехід на /dashboard	Пройшов
4	Доступ до захищеної сторінки без токена	Відсутній токен у localStorage	Перенаправлення на /login	Пройшов
5	Додавання продукту до журналу	Пошук за назвою, введення кількості	Продукт з'являється у секції прийому їжі	Пройшов
6	Пошук продукту за штрих-кодом	Сканування штрих-коду	Продукт знайдено і обрано автоматично	Пройшов

№	Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
		через камеру		
7	Редагування кількості продукту	Нове значення в грамах	Значення КБЖВ перераховано	Пройшов
8	Перегляд статистики за різні діапазони	Перемикач 7/14/30/90 днів	Графік та картки оновлюються	Пройшов
9	Додавання запису ваги	Значення в кг	Запис з'являється в історії	Пройшов
10	Створення власного продукту	Назва та показники КБЖВ	Продукт з'являється у «Мої продукти»	Пройшов
11	Блокування користувача (Admin)	Причина блокування	Статус користувача змінено на «Заблокований»	Пройшов

Тестування продуктивності виконувалось засобами Google Lighthouse у режимі емуляції мобільного пристрою на мережі 4G. Результати наведено у табл. 4.2.

Таблиця 4.2

Результати аудиту продуктивності Lighthouse

Метрика	Значення	Норма
Performance Score	77 / 100	50-89 – потребує покращення
First Contentful Paint (FCP)	2.5 с	≤ 3 с – задовільно
Largest Contentful Paint (LCP)	4.6 с	> 4 с – потребує покращення
Total Blocking Time (TBT)	110 мс	≤ 200 мс – добре
Cumulative Layout Shift (CLS)	0.003	≤ 0.1 – відмінно

Скріншот результату тестування за допомогою засобів Google Lighthouse показано на рис. 25.

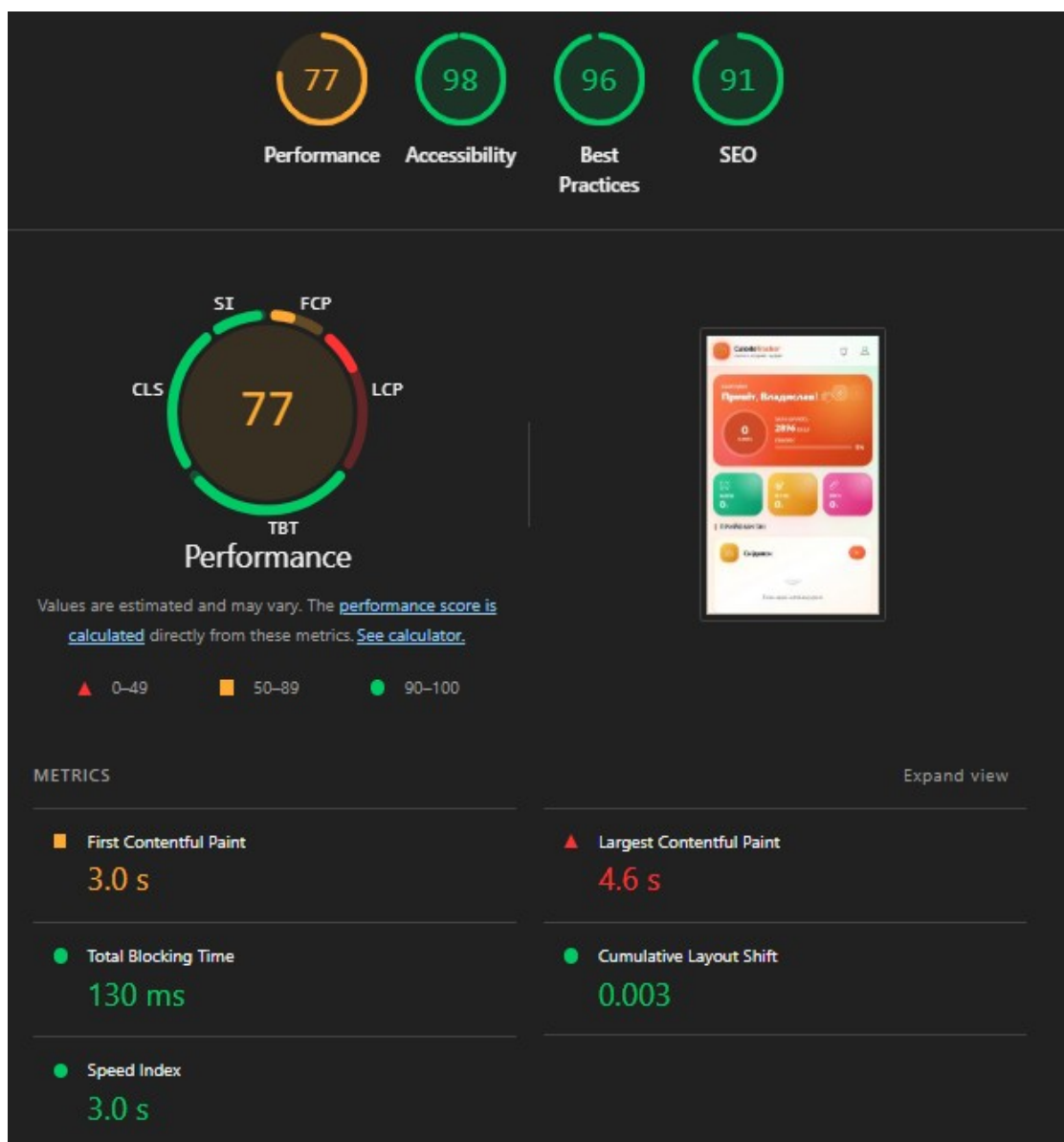


Рис. 25 Результати аудиту Lighthouse для CalorieTracker

Після фінального оновлення дизайну, аудит показав бал продуктивності 77/100. Аналіз звіту Lighthouse виявив три основні причини зниження: наявність render-blocking запитів, неоптимізована доставка зображень та недостатнє кешування статичних ресурсів. Для підвищення балу рекомендується конвертація зображень у формат WebP, налаштування Cache-Control заголовків на сервері та усунення render-blocking ресурсів. Зазначені недоліки стосуються виключно продуктивності завантаження і не впливають на функціональність

застосунку – метрики TBT (110 мс) та CLS (0.003) залишились у зеленій зоні норми. А також потрібно враховувати, що зараз додаток знаходиться на безкоштовному хостингу Render.com, що може додатково впливати на час відповіді API і загальні показники продуктивності.

4.2 Вимоги до апаратного та програмного забезпечення

Для користувача клієнтської частини системи CalorieTracker вимоги є мінімальними. Застосунок функціонує у браузері без необхідності встановлення.

Мінімальні вимоги до пристрою користувача:

- процесор: будь-який сучасний (ARM або x86) з тактовою частотою ≥ 1 ГГц;
- оперативна пам'ять: ≥ 512 МБ;
- мережеве з'єднання: $\geq 3G$ (рекомендовано 4G/Wi-Fi);
- браузер: Chrome 93+, Firefox 90+, Safari 14+, Edge 93+;
- для сканування штрих-кодів: камера пристрою та дозвіл браузера на її використання.

Для розгортання серверної частини:

- СУБД: PostgreSQL через Supabase;
- середовище виконання: .NET 8 Runtime;
- ОС: Ubuntu 20.04 і краще (Render Docker-середовище).

4.3 Склад інсталяційного пакету

Розроблений веб-застосунок розгорнуто у хмарному середовищі та доступний через URL. Для розгортання клієнтської частини використовується статичний хостинг.

Склад дистрибутиву клієнтської частини (папка dist/ після команди vite build):

- index.html – точка входу SPA;

- `assets/index-[hash].js` – основний JavaScript-бандл (мінімізований);
- `assets/index-[hash].css` – стилі Tailwind CSS (після purging);
- `public/icons/` – SVG та PNG іконки інтерфейсу;
- `public/logo.png` – логотип застосунку.

Склад дистрибутиву серверної частини (Dockerfile, розгортається на Render.com):

- `CalorieTracker.Server.dll` – серверна частина (ASP.NET Core 8);
- `appsettings.json` – конфігурація (рядок підключення до Supabase, JWT-ключ, FatSecret API keys);
- `Dockerfile` – інструкція збірки Docker-образу.

ВИСНОВКИ

Метою бакалаврської кваліфікаційної роботи було спроектувати та реалізувати клієнтську частину аналітичної системи підрахунку калорій при вживанні їжі людиною. Поставлену мету досягнуто в повному обсязі – розроблено функціональний клієнтський застосунок CalorieTracker, розгорнутий у хмарному середовищі.

У процесі виконання роботи встановлено, що персональний облік харчування є систематичним процесом фіксації спожитих продуктів з метою контролю енергетичного балансу організму. Аналіз існуючих рішень показав, що вони є переважно закритими комерційними продуктами з обмеженою безкоштовною функціональністю або перевантаженим інтерфейсом. Це підтвердило доцільність розробки власної зручної системи з фокусом на користувацький досвід.

Спроектовано та реалізовано архітектуру клієнтської частини у вигляді односторінкового застосунку (SPA). Взаємодію з серверною частиною налаштовано через REST API у форматі JSON за допомогою HTTP-клієнта Axios. Важливим технічним рішенням стала реалізація механізму перехоплювачів для автоматичного керування JWT-токенами та безпечної автентифікації користувачів, що забезпечило надійний захист сесій.

Програмна реалізація виконана на базі бібліотеки React 18 з використанням мови TypeScript. Це гарантувало статичну типізацію та узгодженість із серверними моделями даних DTO. Застосунок реалізує повний цикл обліку харчування, починаючи налаштуванням особистого профілю, закінчуючи щоденним веденням журналу, переглядом статистики та отриманням досягнень. Інтеграція із зовнішнім сервісом пошуку продуктів дозволяє здійснювати пошук за назвою, а сканування штрих-кодів реалізовано на стороні клієнта засобами бібліотеки @zxing/browser. Також розроблено адміністративну панель для управління обліковими записами, модерації продуктів та перегляду системних логів.

Функціональне тестування підтвердило коректну роботу всіх реалізованих сценаріїв використання на стороні клієнта. Аудит продуктивності засобами Google Lighthouse показав бал 77/100, при цьому метрики інтерактивності та стабільності інтерфейсу відповідають нормі. Виявлені причини зниження загального балу продуктивності завантаження (render-blocking запити, неоптимізована доставка зображень та особливості роботи безкоштовного хостингу) мають можливість усунення на етапі подальшої оптимізації і не впливають на функціональність застосунку.

Практична цінність роботи полягає у створенні сучасного, адаптивного та швидкого інструменту персонального обліку харчування, придатного для реального використання на мобільних і десктопних пристроях. Основні результати апробовано на Всеукраїнській науково-практичній конференції TAACSD 2026, де опубліковано тези доповіді. Перспективами подальшого розвитку є впровадження технології PWA для можливості роботи в офлайн-режимі, а також розробка нативного мобільного клієнта.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. Your First Component [Електронний ресурс]. – Режим доступу: <https://react.dev/learn/your-first-component>.
2. TypeScript. TypeScript for JavaScript Programmers [Електронний ресурс]. – Режим доступу: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.
3. Responsive Web Design Basics [Електронний ресурс] / web.dev. – Режим доступу: <https://web.dev/learn/design/>.
4. Apple Inc. Human Interface Guidelines [Електронний ресурс]. – Режим доступу: <https://developer.apple.com/design/human-interface-guidelines>.
5. Google. Material Design Guidelines [Електронний ресурс]. – Режим доступу: <https://m3.material.io>.
6. Vite. Getting Started [Електронний ресурс]. – Режим доступу: <https://vitejs.dev/guide/>.
7. Tailwind CSS. Core Concepts [Електронний ресурс]. – Режим доступу: <https://tailwindcss.com/docs/utility-first>.
8. Axios. Promise based HTTP client for the browser and node.js [Електронний ресурс]. – Режим доступу: <https://axios-http.com/docs/intro>.
9. React Router DOM. Tutorial [Електронний ресурс]. – Режим доступу: <https://reactrouter.com/en/main/start/tutorial>.
10. Google Lighthouse. Overview [Електронний ресурс]. – Режим доступу: <https://developer.chrome.com/docs/lighthouse/overview>. – Дата звернення: 20.05.2026.
11. Supabase. PostgreSQL hosting platform [Електронний ресурс]. – Режим доступу: <https://supabase.com/docs>.
12. ASP.NET Core Web API. Tutorial [Електронний ресурс] / Microsoft Docs. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/web-api/>.
13. JWT Introduction. JSON Web Tokens [Електронний ресурс]. – Режим доступу: <https://jwt.io/introduction>.

14. Recharts: A composable charting library built on React components [Електронний ресурс]. – Режим доступу: <https://recharts.org/>.
15. PostgreSQL 16 Documentation [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/16/>.
16. UML 2.5 Specification [Електронний ресурс] / Object Management Group. – Режим доступу: <https://www.omg.org/spec/UML/2.5.1>.
17. Render. Cloud Application Hosting [Електронний ресурс]. – Режим доступу: <https://render.com/docs>. – Дата звернення: 16.05.2026.
18. BCrypt.Net. Password hashing library [Електронний ресурс]. – Режим доступу: <https://github.com/BcryptNet/bcrypt.net>.
19. MyFitnessPal [Електронний ресурс]. – Режим доступу: <https://www.myfitnesspal.com/>. – Дата звернення: 21.05.2026.
20. Cronometer: Track nutrition & count calories [Електронний ресурс]. – Режим доступу: <https://cronometer.com/>. – Дата звернення: 21.05.2026.
21. FatSecret. Calorie Counter and Diet Tracker for Weight Loss [Електронний ресурс]. – Режим доступу: <https://www.fatsecret.com/>. – Дата звернення: 21.05.2026.
22. Open Food Facts. The free food database for all [Електронний ресурс]. – Режим доступу: <https://world.openfoodfacts.org/>. – Дата звернення: 21.05.2026.
23. FoodData Central [Електронний ресурс] / U.S. Department of Agriculture. – Режим доступу: <https://fdc.nal.usda.gov/>. – Дата звернення: 21.05.2026.
24. World Health Organization. Body mass index - BMI [Електронний ресурс]. – Режим доступу: <https://www.who.int/data/gho/data/themes/topics/topic-details/GHO/body-mass-index>. – Дата звернення: 19.05.2026.
25. Збалансоване харчування, баланс БЖУ [Електронний ресурс] // Медичний центр «Daily Medical». – Режим доступу: <https://daily-med.com.ua/uk/blog-uk/zbalansovane-harchuvannya-balans-bzhu>. – Дата звернення: 19.05.2026.

26. Figma: The Collaborative Interface Design Tool [Электронный ресурс]. – Режим доступа: <https://www.figma.com/>. – Дата звернення: 01.03.2026.
27. ZXing barcode scanning library for Browser [Электронный ресурс]. – Режим доступа: <https://github.com/zxing-js/browser>.
28. Progressive Web Apps [Электронный ресурс] / MDN Web Docs. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.
29. Web Vitals: Essential metrics for a healthy site [Электронный ресурс] / Google Developers. – Режим доступа: <https://web.dev/articles/vitals>.

ДОДАТКИ

ДОДАТОК А

Скріншоти інтерфейсу системи CalorieTracker

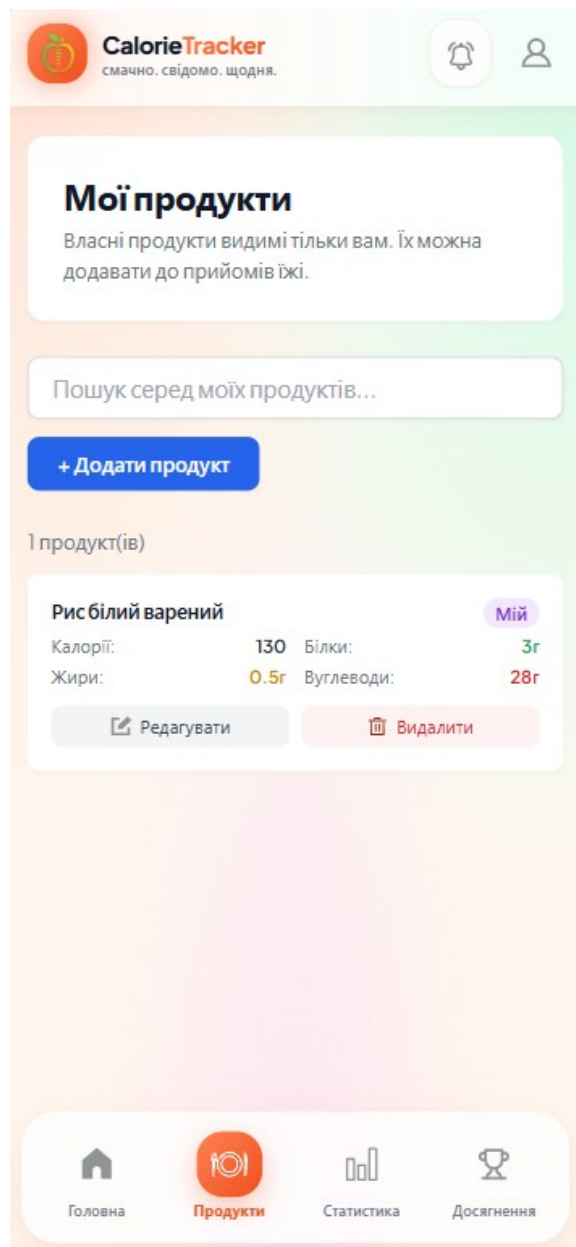


Рис. А.1 Сторінка управління продуктами

Додати власний продукт

×

Назва *

Назва продукту

Бренд

Категорія

Без категорії

Калорії/100г *

0

Білки/100г

0

Жири/100г

0

Вуглеводи/100г

0

Клітковина/100г

0

Цукор/100г

Зберегти

×

Скасувати

Рис. А.2 Форма додавання продукту на сторінці продуктів

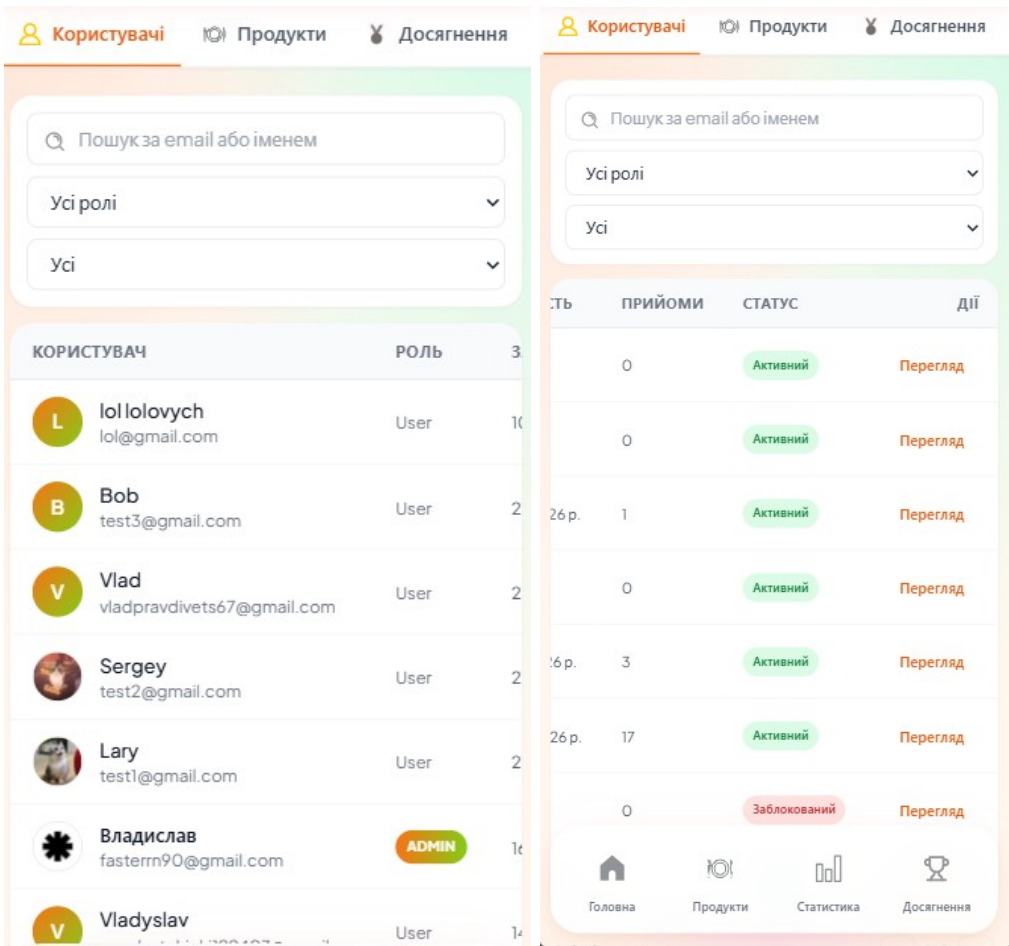


Рис. А.3 Вкладка управління користувачами в адмін-панелі

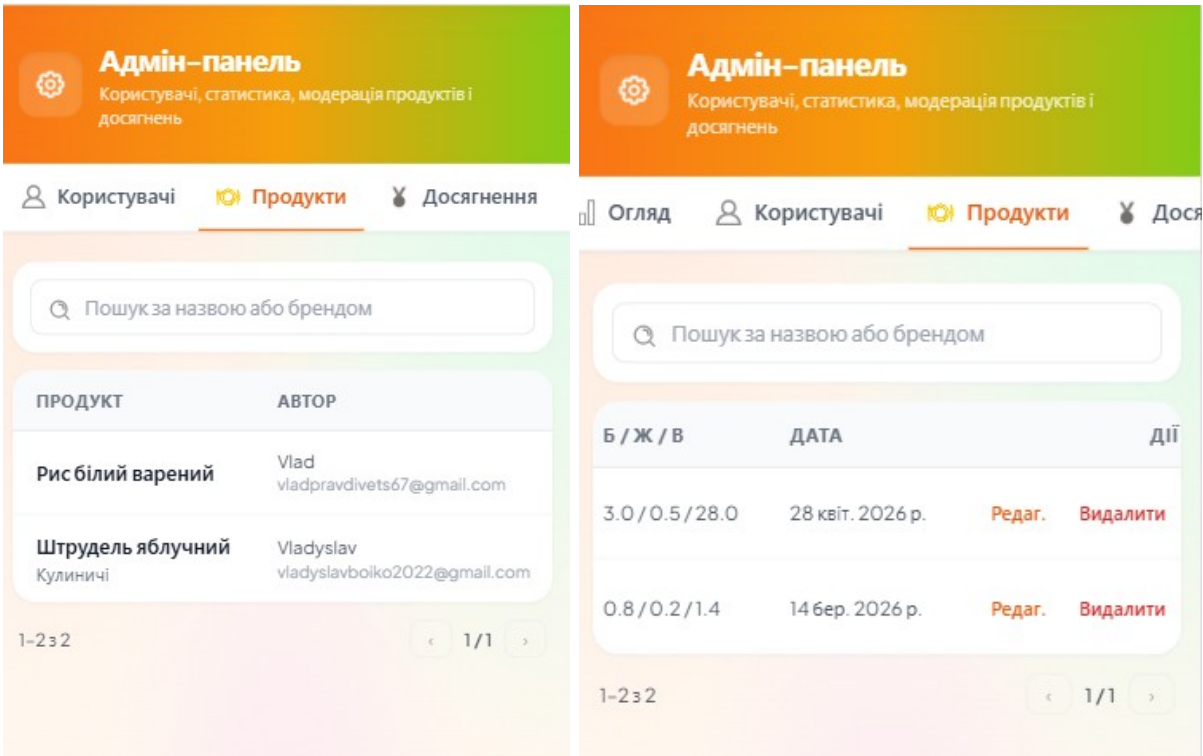


Рис. А.4 Вкладка управління продуктами користувачів в адмін-панелі

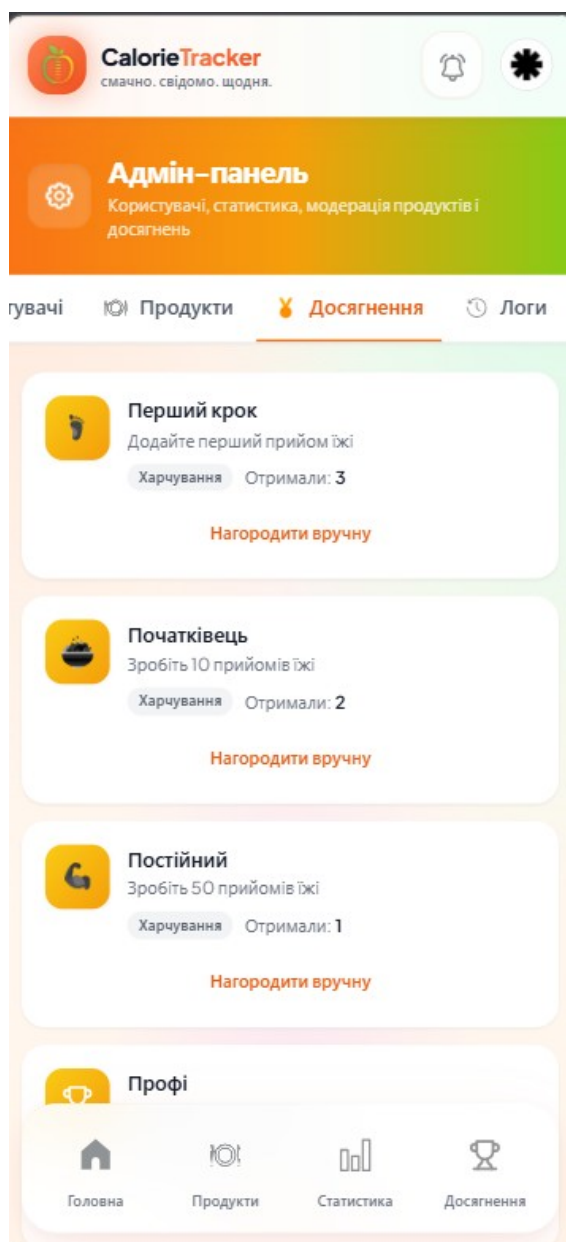


Рис. А.5 Вкладка адмін-панелі з управління досягненнями

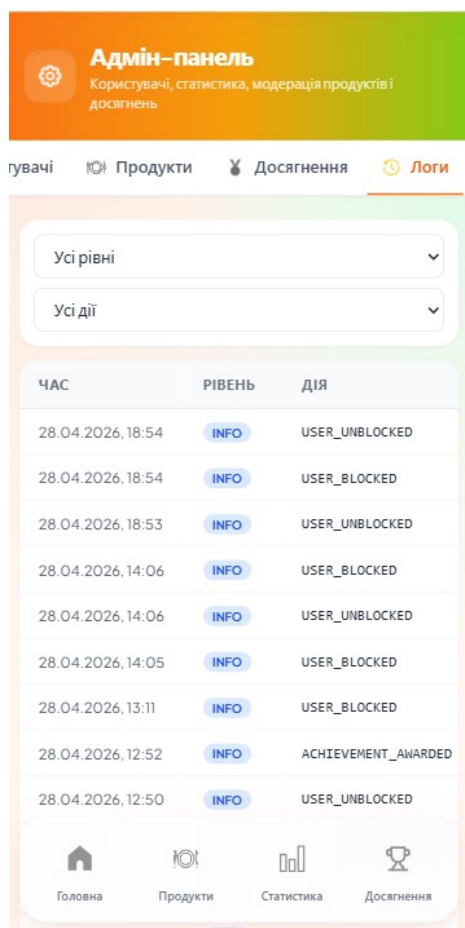


Рис. А.6 Вкладка адмін-панелі з логами

ДОДАТОК Б

Лістинги коду сервісного шару клієнтської частини

Api.ts – сервіс HTTP-запитів клієнтської системи:

```
import axios from 'axios';
import type { AxiosResponse } from 'axios';
import type {
  User,
  UserRegistration,
  UserLogin,
  AuthResponse,
  Food,
  FoodCreate,
  FoodSearchParams,
  ExternalFood,
  DailyMeals,
  Meal,
  AddFoodToMeal,
  MealType,
  AchievementDto,
  AdminUserListItem,
  AdminUserDetails,
  AdminFood,
  AdminStats,
  AdminAchievementDefinition,
  AdminAuditLog,
  PagedResult,
} from '../types';

const API_BASE_URL = import.meta.env.DEV ? 'https://localhost:7100/api' : '/api';

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('authToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('authToken');
      localStorage.removeItem('user');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export const authApi = {
  register: async (data: UserRegistration): Promise<AxiosResponse<AuthResponse>> => {
    console.log('API: Відправляємо запит реєстрації на:', `${API_BASE_URL}/auth/register`);
```

```

    console.log('API: Дані для реєстрації:', data);
    try {
      const response = await api.post('/auth/register', data);
      console.log('API: Успішна реєстрація:', response.data);
      return response;
    } catch (error) {
      console.error('API: Помилка реєстрації:', error);
      throw error;
    }
  },

  login: async (data: UserLogin): Promise<AxiosResponse<AuthResponse>> => {
    console.log('API: Відправляємо запит логіну на:', `${API_BASE_URL}/auth/login`);
    try {
      const response = await api.post('/auth/login', data);
      console.log('API: Успішний логін:', response.data);
      return response;
    } catch (error) {
      console.error('API: Помилка логіну:', error);
      throw error;
    }
  },
};

export const foodsApi = {
  getFoods: (params?: FoodSearchParams): Promise<AxiosResponse<Food[]>> =>
    api.get('/foods', { params }),

  getFood: (id: number): Promise<AxiosResponse<Food>> =>
    api.get(`/foods/${id}`),

  createFood: (data: FoodCreate): Promise<AxiosResponse<Food>> =>
    api.post('/foods', data),

  updateFood: (id: number, data: FoodCreate): Promise<AxiosResponse<Food>> =>
    api.put(`/foods/${id}`, data),

  deleteFood: (id: number): Promise<AxiosResponse<void>> =>
    api.delete(`/foods/${id}`),

  searchFoods: (query: string): Promise<AxiosResponse<Food[]>> =>
    api.get(`/foods/search/${encodeURIComponent(query)}`),

  getCategories: (): Promise<AxiosResponse<string[]>> =>
    api.get('/foods/categories'),
};

export const externalFoodsApi = {
  searchByName: (query: string, source: 'off' | 'ukraine' = 'ukraine'): Promise<AxiosResponse<ExternalFood[]>> =>
    api.get('/foods/external/search', { params: { query, source } }),

  searchByBarcode: (barcode: string): Promise<AxiosResponse<ExternalFood>> =>
    api.get(`/foods/external/barcode/${encodeURIComponent(barcode)}`),
};

export const mealsApi = {
  getDailyMeals: (date: string): Promise<AxiosResponse<DailyMeals>> =>
    api.get(`/meals/daily/${date}`),

  createMeal: (data: {
    date: string;
    mealType: MealType;
    items: AddFoodToMeal[];
  }): Promise<AxiosResponse<Meal>> =>

```

```

    api.post('/meals', data),

    getMeal: (id: number): Promise<AxiosResponse<Meal>> =>
      api.get(`/meals/${id}`),

    addFoodToMeal: (mealId: number, data: AddFoodToMeal): Promise<AxiosResponse<any>> =>
      api.post(`/meals/${mealId}/items`, data),

    updateMealItemQuantity: async (itemId: number, quantity: number): Promise<AxiosResponse<void>> => {
      console.log(`API: Оновлення кількості для item ${itemId} на ${quantity}г`);
      console.log(`API URL: ${API_BASE_URL}/meals/items/${itemId}/quantity`);
      try {
        const response = await api.put(`/meals/items/${itemId}/quantity`, { quantity });
        console.log('API: Кількість успішно оновлена', response);
        return response;
      } catch (error) {
        console.error('API: Помилка оновлення кількості:', error);
        throw error;
      }
    },

    removeFoodFromMeal: async (itemId: number): Promise<AxiosResponse<void>> => {
      console.log(`API: Видалення item ${itemId}`);
      console.log(`API URL: ${API_BASE_URL}/meals/items/${itemId}`);
      try {
        const response = await api.delete(`/meals/items/${itemId}`);
        console.log('API: Продукт успішно видалено', response);
        return response;
      } catch (error) {
        console.error('API: Помилка видалення:', error);
        throw error;
      }
    },

    deleteMeal: (id: number): Promise<AxiosResponse<void>> =>
      api.delete(`/meals/${id}`),
  };

export const usersApi = {
  getProfile: (): Promise<AxiosResponse<User>> =>
    api.get('/users/profile'),

  updateProfile: (data: Partial<User>): Promise<AxiosResponse<User>> =>
    api.put('/users/profile', data),

  updateAvatar: (file: Blob, filename = 'avatar.jpg'): Promise<AxiosResponse<{ avatarUrl: string }>> => {
    const formData = new FormData();
    formData.append('file', file, filename);
    return api.put('/users/avatar', formData, {
      headers: { 'Content-Type': 'multipart/form-data' },
    });
  },

  deleteAvatar: (): Promise<AxiosResponse<{ avatarUrl: null }>> =>
    api.delete('/users/avatar'),

  addWeightRecord: (weight: number, height?: number): Promise<AxiosResponse<any>> =>
    api.post('/users/weight', { weight, height }),

  getWeightHistory: (days = 30): Promise<AxiosResponse<{ date: string; weight: number }[]>> =>
    api.get('/users/weight-history', { params: { days } }),

  getStatistics: (days = 7): Promise<AxiosResponse<any>> =>
    api.get('/users/statistics', { params: { days } }),
};

```

```

};

export const achievementsApi = {
  getAll: (): Promise<AxiosResponse<AchievementDto[]>> =>
    api.get('/achievements'),

  check: (): Promise<AxiosResponse<AchievementDto[]>> =>
    api.post('/achievements/check'),
};

export const adminApi = {
  getStats: (): Promise<AxiosResponse<AdminStats>> =>
    api.get('/admin/stats'),

  listUsers: (params: {
    search?: string;
    role?: string;
    blocked?: boolean;
    page?: number;
    pageSize?: number;
  }): Promise<AxiosResponse<PagedResult<AdminUserListItem>>> =>
    api.get('/admin/users', { params }),

  getUser: (id: number): Promise<AxiosResponse<AdminUserDetails>> =>
    api.get(`/admin/users/${id}`),

  blockUser: (id: number, reason?: string): Promise<AxiosResponse<void>> =>
    api.post(`/admin/users/${id}/block`, { reason }),

  unblockUser: (id: number): Promise<AxiosResponse<void>> =>
    api.post(`/admin/users/${id}/unblock`),

  deleteUser: (id: number): Promise<AxiosResponse<void>> =>
    api.delete(`/admin/users/${id}`),

  listFoods: (params: {
    search?: string;
    userId?: number;
    page?: number;
    pageSize?: number;
  }): Promise<AxiosResponse<PagedResult<AdminFood>>> =>
    api.get('/admin/foods', { params }),

  updateFood: (id: number, data: FoodCreate): Promise<AxiosResponse<void>> =>
    api.put(`/admin/foods/${id}`, data),

  deleteFood: (id: number): Promise<AxiosResponse<void>> =>
    api.delete(`/admin/foods/${id}`),

  getAchievementDefinitions: (): Promise<AxiosResponse<AdminAchievementDefinition[]>> =>
    api.get('/admin/achievements/definitions'),

  getUserAchievements: (userId: number): Promise<AxiosResponse<{ code: string; unlockedAt: string }[]>> =>
    api.get(`/admin/achievements/user/${userId}`),

  awardAchievement: (userId: number, code: string): Promise<AxiosResponse<void>> =>
    api.post('/admin/achievements/award', { userId, code }),

  revokeAchievement: (userId: number, code: string): Promise<AxiosResponse<void>> =>
    api.delete(`/admin/achievements/revoke/${userId}/${encodeURIComponent(code)}`),

  listLogs: (params: {
    level?: string;
    action?: string;
  })

```

```

    actorUserId?: number;
    page?: number;
    pageSize?: number;
  }): Promise<AxiosResponse<PagedResult<AdminAuditLog>>> =>
    api.get('/admin/logs', { params }),
  });

export default api;

```

auth.ts – сервіс автентифікації та управління сесією:

```

import type { User } from '../types';

export const getAuthToken = (): string | null => {
  return localStorage.getItem('authToken');
};

export const setAuthToken = (token: string): void => {
  localStorage.setItem('authToken', token);
};

export const removeAuthToken = (): void => {
  localStorage.removeItem('authToken');
  localStorage.removeItem('user');
};

export const getCurrentUser = (): User | null => {
  const userStr = localStorage.getItem('user');
  if (userStr) {
    try {
      return JSON.parse(userStr);
    } catch {
      return null;
    }
  }
  return null;
};

export const setCurrentUser = (user: User): void => {
  localStorage.setItem('user', JSON.stringify(user));
  window.dispatchEvent(new Event('auth-user-changed'));
};

export const isAuthenticated = (): boolean => {
  return !!getAuthToken() && !!getCurrentUser();
};

export const isAdmin = (): boolean => {
  const u = getCurrentUser();
  return u?.role === 'Admin';
};

export const logout = (): void => {
  removeAuthToken();
  window.location.href = '/login';
};

```

ДОДАТОК В

Лістинги коду основних компонентів клієнтської частини

Dashboard.tsx — компонент щоденного журналу харчування:

```
import React, { useState, useEffect } from 'react';
import { getCurrentUser } from '../utils/auth';
import { mealsApi } from '../services/api';
import AddFoodModal from '../components/AddFoodModal';
import Icon from '../components/Icon';

interface DailyMeals {
  date: string;
  meals: any[];
  summary: {
    totalCalories: number;
    totalProtein: number;
    totalFats: number;
    totalCarbs: number;
    dailyCalorieGoal?: number;
  };
};

const Dashboard = () => {
  const [selectedDate, setSelectedDate] = useState(new Date());
  const [dailyMeals, setDailyMeals] = useState<DailyMeals | null>(null);
  const [loading, setLoading] = useState(false);
  const [showAddFoodModal, setShowAddFoodModal] = useState(false);
  const [selectedMealType, setSelectedMealType] = useState<number>(1);
  const [expandedMeals, setExpandedMeals] = useState<{ [key: number]: boolean }>({});
  const user = getCurrentUser();

  const loadDailyMeals = async (date: Date) => {
    try {
      setLoading(true);
      const dateStr = date.toISOString().split('T')[0];
      const response = await mealsApi.getDailyMeals(dateStr);
      setDailyMeals(response.data);
    } catch (error) {
      console.error('Error loading daily meals:', error);
      setDailyMeals({
        date: date.toISOString().split('T')[0],
        meals: [],
        summary: {
          totalCalories: 0,
          totalProtein: 0,
          totalFats: 0,
          totalCarbs: 0,
          dailyCalorieGoal: user?.dailyCalorieGoal
        }
      });
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    loadDailyMeals(selectedDate);
  }, [selectedDate]);

  const handleEditQuantity = async (itemId: number, currentQuantity: number) => {
    const newQuantityStr = prompt('Введіть нову кількість (грам):', currentQuantity.toString());
```

```

if (newQuantityStr === null) return;

const newQuantity = parseFloat(newQuantityStr);

if (isNaN(newQuantity) || newQuantity <= 0) {
  alert('Введіть коректну кількість більше 0');
  return;
}

try {
  await mealsApi.updateMealItemQuantity(itemId, newQuantity);
  await loadDailyMeals(selectedDate);
  alert('Кількість успішно оновлена!');
} catch (error: any) {
  console.error('Помилка при оновленні:', error);
  alert('Помилка при оновленні кількості: ' + (error?.response?.data?.message || error?.message || 'Невідома
помилка'));
}
};

const handleDeleteItem = async (itemId: number, foodName: string) => {
  const confirmed = confirm(`Видалити "${foodName}" з прийому їжі?`);

  if (!confirmed) return;

  try {
    await mealsApi.removeFoodFromMeal(itemId);
    await loadDailyMeals(selectedDate);
    alert('Продукт успішно видалено!');
  } catch (error: any) {
    console.error('Помилка при видаленні:', error);
    alert('Помилка при видаленні продукту: ' + (error?.response?.data?.message || error?.message || 'Невідома
помилка'));
  }
};

const handleAddFood = (mealType: number) => {
  setSelectedMealType(mealType);
  setShowAddFoodModal(true);
};

const handleFoodAdded = () => {
  setShowAddFoodModal(false);
  loadDailyMeals(selectedDate);
};

const toggleMealExpansion = (mealType: number) => {
  setExpandedMeals(prev => ({
    ...prev,
    [mealType]: !prev[mealType]
  }));
};

const getMealTypeName = (mealType: number): string => {
  switch (mealType) {
    case 1: return 'Сніданок';
    case 2: return 'Обід';
    case 3: return 'Вечеря';
    case 4: return 'Перекус';
    default: return 'Прийом їжі';
  }
};

```

```

const getMealTypeIcon = (mealType: number): string => {
  switch (mealType) {
    case 1: return 'breakfast';
    case 2: return 'lunch';
    case 3: return 'dinner';
    case 4: return 'snack';
    default: return 'plate';
  }
};

const getMealCardClass = (mealType: number): string => {
  switch (mealType) {
    case 1: return 'meal-card meal-card-breakfast';
    case 2: return 'meal-card meal-card-lunch';
    case 3: return 'meal-card meal-card-dinner';
    case 4: return 'meal-card meal-card-snack';
    default: return 'meal-card';
  }
};

const getMealIconBgClass = (mealType: number): string => {
  switch (mealType) {
    case 1: return 'meal-icon-breakfast';
    case 2: return 'meal-icon-lunch';
    case 3: return 'meal-icon-dinner';
    case 4: return 'meal-icon-snack';
    default: return 'meal-icon-lunch';
  }
};

const getMealByType = (mealType: number) => {
  return dailyMeals?.meals.find(meal => meal.mealType === mealType);
};

const changeDateBy = (days: number) => {
  const newDate = new Date(selectedDate);
  newDate.setDate(newDate.getDate() + days);
  setSelectedDate(newDate);
};

const isToday = (date: Date) => {
  const today = new Date();
  return date.toDateString() === today.toDateString();
};

const goalKcal = dailyMeals?.summary.dailyCalorieGoal || user?.dailyCalorieGoal || 2000;
const consumedKcal = Math.round(dailyMeals?.summary.totalCalories || 0);
const remainingKcal = Math.max(goalKcal - consumedKcal, 0);
const progressPct = Math.min((consumedKcal / Math.max(goalKcal, 1)) * 100, 100);

return (
  <div className="min-h-screen pb-24">
    <div className="max-w-md mx-auto px-4 py-5">
      <div className="hero-card p-5 mb-4 animate-fade-in-up">
        <div className="relative z-10">
          <div className="flex items-center justify-between mb-4">
            <div>
              <p className="text-white/80 text-xs font-semibold tracking-wide uppercase">
                {isToday(selectedDate) ? 'Сьогодні' : selectedDate.toLocaleDateString('uk-UA', { weekday: 'long' })}
              </p>
              <h1 className="text-2xl font-extrabold text-white drop-shadow leading-tight flex items-center gap-2">
                <span>Привіт, {user?.name?.split('')[0] || 'друзе'}!</span>
                <Icon name="wave" size={26} color="white" />
              </h1>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
);

```

```

</div>
<div className="flex items-center gap-1">
  <button
    onClick={() => changeDateBy(-1)}
    className="w-9 h-9 rounded-xl bg-white/20 hover:bg-white/30 backdrop-blur-sm border border-
white/30 text-white transition-all active:scale-95 flex items-center justify-center"
    aria-label="Попередний день"
  >
    <Icon name="chevron-left" size={18} color="white" />
  </button>
  <button
    onClick={() => changeDateBy(1)}
    disabled={isToday(selectedDate)}
    className="w-9 h-9 rounded-xl bg-white/20 hover:bg-white/30 backdrop-blur-sm border border-
white/30 text-white transition-all active:scale-95 disabled:opacity-30 disabled:cursor-not-allowed flex items-center justify-
center"
    aria-label="Наступний день"
  >
    <Icon name="chevron-right" size={18} color="white" />
  </button>
</div>
</div>

<div className="flex items-center gap-4">
  <div className="relative flex-shrink-0">
    <div
      className="w-28 h-28 rounded-full p-1.5"
      style={{
        background: `conic-gradient(#FFFFFF ${progressPct * 3.6}deg, rgba(255,255,255,0.2) $
{progressPct * 3.6}deg)`
      }}
    >
      <div className="w-full h-full rounded-full bg-brand-700/80 backdrop-blur flex flex-col items-center
justify-center">
        <div className="text-2xl font-extrabold text-white leading-none">{consumedKcal}</div>
        <div className="text-[10px] text-white/80 font-semibold uppercase tracking-wider mt-0.5">3
{goalKcal}</div>
      </div>
    </div>
  </div>

  <div className="flex-1 space-y-2.5 text-white">
    <div>
      <div className="text-[10px] uppercase tracking-wider text-white/70 font-bold">Залишилось</div>
      <div className="text-xl font-extrabold leading-tight">{remainingKcal} <span className="text-sm
font-semibold text-white/80">ккал</span></div>
    </div>
    <div>
      <div className="text-[10px] uppercase tracking-wider text-white/70 font-bold">Порпес</div>
      <div className="flex items-center gap-2">
        <div className="flex-1 h-2 rounded-full bg-white/25 overflow-hidden">
          <div
            className="h-full rounded-full bg-gradient-to-r from-amber-200 to-white transition-all
duration-500"
            style={{ width: `${progressPct}%` }}
          />
        </div>
        <span className="text-xs font-bold">{Math.round(progressPct)}%</span>
      </div>
    </div>
  </div>
</div>
</div>
</div>

```

```

    {!loading && dailyMeals && (
      <div className="grid grid-cols-3 gap-3 mb-4 animate-fade-in-up">
        <div className="tile-protein">
          <div className="relative z-10">
            <Icon name="protein" size={22} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold mt-1.5">Білки</div>
            <div className="text-xl font-extrabold leading-none mt-0.5">
              {Math.round(dailyMeals.summary.totalProtein)} <span className="text-xs font-semibold
text-white/80 ml-0.5">r</span>
            </div>
          </div>
        </div>
        <div className="tile-fats">
          <div className="relative z-10">
            <Icon name="fats" size={22} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold mt-1.5">Жири</div>
            <div className="text-xl font-extrabold leading-none mt-0.5">
              {Math.round(dailyMeals.summary.totalFats)} <span className="text-xs font-semibold text-white/80
ml-0.5">r</span>
            </div>
          </div>
        </div>
        <div className="tile-carbs">
          <div className="relative z-10">
            <Icon name="carbs" size={22} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold mt-1.5">Вугл.</div>
            <div className="text-xl font-extrabold leading-none mt-0.5">
              {Math.round(dailyMeals.summary.totalCarbs)} <span className="text-xs font-semibold text-white/80
ml-0.5">r</span>
            </div>
          </div>
        </div>
      </div>
    )}

    {!loading && (
      <div className="space-y-3">
        <h2 className="text-sm font-bold text-gray-700 uppercase tracking-wider px-1 flex items-center gap-2">
          <span className="inline-block w-1 h-4 rounded-full bg-brand-gradient" />
          Прийоми їжі
        </h2>
        {[1, 2, 3, 4].map((mealType) => {
          const meal = getMealByType(mealType);
          const isExpanded = expandedMeals[mealType];
          const hasItems = meal && meal.items && meal.items.length > 0;

          return (
            <div
              key={mealType}
              className={getMealCardClass(mealType)}
            >
              <div className="p-4">
                <div className="flex justify-between items-center mb-3">
                  <div className="flex items-center gap-3">
                    <div className={`w-11 h-11 rounded-2xl flex items-center justify-center shadow-md $
{getMealIconBgClass(mealType)}`>
                      <Icon name={getMealTypeIcon(mealType)} size={22} color="white" />
                    </div>
                    <div>
                      <h3 className="text-base font-bold text-gray-900 leading-tight">
                        {getMealTypeName(mealType)}
                      </h3>
                      {hasItems && (

```

```

        <p className="text-xs text-gray-500">
            {meal.items.length} продукт{meal.items.length > 1 ? 'и' : ''} ·
        </p>
    </div>
    </div>

    <button
        onClick={() => handleAddFood(mealType)}
        className="px-3 py-2 bg-brand-gradient text-white rounded-full text-sm font-bold shadow-pop
        hover:brightness-110 hover:-translate-y-0.5 active:translate-y-0 transition-all flex items-center gap-1"
    >
        <Icon name="add" size={16} color="white" />
        <span className="hidden xs:inline">Додати</span>
    </button>
</div>

{hasItems && (
    <button
        onClick={() => toggleMealExpansion(mealType)}
        className="w-full flex justify-between items-center p-3 bg-white/70 backdrop-blur
        rounded-2xl border border-white/80 hover:bg-white transition-all"
    >
        <div className="flex items-center gap-3 flex-1">
            <div className="flex items-center gap-1 text-xs">
                <span className="w-2 h-2 rounded-full bg-fresh-500" />
                <span className="font-semibold text-gray-700">{Math.round(meal.totalProtein || 0)}
            </div>
            <div className="flex items-center gap-1 text-xs">
                <span className="w-2 h-2 rounded-full bg-sun-500" />
                <span className="font-semibold text-gray-700">{Math.round(meal.totalFats || 0)}
            </div>
            <div className="flex items-center gap-1 text-xs">
                <span className="w-2 h-2 rounded-full bg-berry-500" />
                <span className="font-semibold text-gray-700">{Math.round(meal.totalCarbs || 0)}
            </div>
        </div>
        <div className="flex items-center gap-2">
            <span className="text-sm font-extrabold text-gradient-brand">
                {Math.round(meal.totalCalories)}
            </span>
            <span className="text-xs text-gray-500">ккал</span>
            <Icon
                name={isExpanded ? "expand-less" : "expand-more"}
                size={20}
                color="gray"
            />
        </div>
    </button>
)}

{!hasItems && (
    <div className="text-center py-4 text-gray-500">
        <div className="mb-1.5 flex justify-center opacity-60">
            <Icon name="empty-plate" size={40} color="gray" />
        </div>
        <p className="text-xs">Поки що нічого не додано</p>
    </div>
)}
</div>

```

```

{hasItems && isExpanded && (
  <div className="border-t border-white/80 bg-white/50 backdrop-blur p-3 space-y-2 animate-fade-
in-up">
    {meal.items.map((item: any, index: number) => (
      <div
        key={index}
        className="p-3 bg-white rounded-2xl shadow-sm border border-cream-100 hover:shadow-
soft transition-shadow"
      >
        <div className="flex justify-between items-start mb-2">
          <div className="flex-1 min-w-0">
            <div className="font-semibold text-gray-900 text-sm truncate">
              {item.foodName}
            </div>
            <div className="text-xs text-gray-500 mt-0.5 flex items-center gap-1.5">
              <span>{item.quantity}</span>
              <span className="text-gray-300">·</span>
              <span className="font-bold text-brand-600">{Math.round(item.calories)}
ккал</span>
            </div>
          </div>

          <div className="flex items-center gap-1 ml-2">
            <button
              onClick={() => handleEditQuantity(item.id, item.quantity)}
              className="p-1.5 text-brand-600 hover:bg-brand-50 rounded-xl transition-colors"
              title="Змінити кількість"
            >
              <Icon name="edit" size={16} color="blue" />
            </button>
            <button
              onClick={() => handleDeleteItem(item.id, item.foodName)}
              className="p-1.5 text-red-500 hover:bg-red-50 rounded-xl transition-colors"
              title="Видалити"
            >
              <Icon name="delete" size={16} color="red" />
            </button>
          </div>
        </div>

        <div className="flex gap-3 text-xs">
          <div className="flex items-center gap-1">
            <span className="w-1.5 h-1.5 rounded-full bg-fresh-500" />
            <span className="text-gray-600">Б {Math.round(item.protein)}г</span>
          </div>
          <div className="flex items-center gap-1">
            <span className="w-1.5 h-1.5 rounded-full bg-sun-500" />
            <span className="text-gray-600">Ж {Math.round(item.fats)}г</span>
          </div>
          <div className="flex items-center gap-1">
            <span className="w-1.5 h-1.5 rounded-full bg-berry-500" />
            <span className="text-gray-600">Б {Math.round(item.carbs)}г</span>
          </div>
        </div>
      </div>
    ))
  </div>
)}
);
}}}
</div>
)}

```

```

    {loading && (
      <div className="flex justify-center items-center py-12">
        <div className="spinner"></div>
      </div>
    )}
  </div>

  {showAddFoodModal && (
    <AddFoodModal
      date={selectedDate.toISOString().split("T")[0]}
      mealType={selectedMealType}
      onClose={() => setShowAddFoodModal(false)}
      onFoodAdded={handleFoodAdded}
    />
  )}
</div>
);
};

export default Dashboard;

```

Statistics.tsx – компонент статистики та аналітики харчування:

```

import React, { useState, useEffect } from 'react';
import {
  LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip,
  ResponsiveContainer, ReferenceLine, PieChart, Pie, Cell, Legend,
  BarChart, Bar
} from 'recharts';
import { getCurrentUser } from '../utils/auth';
import { usersApi } from '../services/api';
import Icon from '../components/Icon';

interface UserStats {
  averageCalories: number;
  averageProtein: number;
  averageFats: number;
  averageCarbs: number;
  totalMeals: number;
  daysWithData: number;
}

interface DailyStats {
  date: string;
  totalCalories: number;
  totalProtein: number;
  totalFats: number;
  totalCarbs: number;
  mealsCount: number;
}

const MACRO_COLORS = {
  protein: '#22c55e',
  fats: '#f59e0b',
  carbs: '#ef4444',
  calories: '#3b82f6',
};

const formatShortDate = (dateString: string) => {
  const date = new Date(dateString);
  return date.toLocaleDateString('uk-UA', { day: 'numeric', month: 'short' });
};

```

```

};

const CaloriesTooltip = ({ active, payload, label }: any) => {
  if (!active || !payload?.length) return null;
  return (
    <div className="bg-white border border-gray-200 rounded-xl shadow-lg p-3 text-sm">
      <p className="font-semibold text-gray-700 mb-1">{label}</p>
      {payload.map((entry: any) => (
        <p key={entry.dataKey} style={{ color: entry.color }}>
          {entry.name}: <strong>{Math.round(entry.value)}</strong>
          {entry.dataKey !== 'goal' ? ' ккал' : ' ккал'}
        </p>
      ))}
    </div>
  );
};

const MacroTooltip = ({ active, payload, label }: any) => {
  if (!active || !payload?.length) return null;
  return (
    <div className="bg-white border border-gray-200 rounded-xl shadow-lg p-3 text-sm">
      <p className="font-semibold text-gray-700 mb-1">{label}</p>
      {payload.map((entry: any) => (
        <p key={entry.dataKey} style={{ color: entry.color }}>
          {entry.name}: <strong>{Math.round(entry.value)}</strong>
        </p>
      ))}
    </div>
  );
};

const Statistics = () => {
  const [userStats, setUserStats] = useState<UserStats | null>(null);
  const [dailyStats, setDailyStats] = useState<DailyStats[]>([]);
  const [selectedPeriod, setSelectedPeriod] = useState(7);
  const [loading, setLoading] = useState(false);
  const user = getCurrentUser();

  useEffect(() => {
    loadStatistics();
  }, [selectedPeriod]);

  const loadStatistics = async () => {
    try {
      setLoading(true);
      const response = await usersApi.getStatistics(selectedPeriod);
      setUserStats(response.data.summary);
      setDailyStats(response.data.dailyStats || []);
    } catch (error) {
      console.error('Error loading statistics:', error);
    } finally {
      setLoading(false);
    }
  };

  const periods = [
    { value: 7, label: '7 днів' },
    { value: 14, label: '14 днів' },
    { value: 30, label: '30 днів' },
    { value: 90, label: '3 місяці' },
  ];

  const calorieGoal = user?.dailyCalorieGoal || 2000;

```

```

const calorieChartData = dailyStats.map(d => ({
  label: formatShortDate(d.date),
  calories: Math.round(d.totalCalories),
  goal: calorieGoal,
}));

const macroChartData = dailyStats.map(d => ({
  label: formatShortDate(d.date),
  Білки: Math.round(d.totalProtein),
  Жири: Math.round(d.totalFats),
  Вуглеводи: Math.round(d.totalCarbs),
}));

const totalMacroKcal = userStats
  ? userStats.averageProtein * 4 + userStats.averageFats * 9 + userStats.averageCarbs * 4
  : 0;

const donutData = userStats && totalMacroKcal > 0
  ? [
    { name: 'Білки', value: Math.round(userStats.averageProtein * 4), grams: Math.round(userStats.averageProtein) },
    { name: 'Жири', value: Math.round(userStats.averageFats * 9), grams: Math.round(userStats.averageFats) },
    { name: 'Вуглеводи', value: Math.round(userStats.averageCarbs * 4), grams: Math.round(userStats.averageCarbs) },
  ]
  : [];
const donutColors = [MACRO_COLORS.protein, MACRO_COLORS.fats, MACRO_COLORS.carbs];

const calculateProgress = (current: number, goal: number) => {
  if (!goal) return 0;
  return Math.min((current / goal) * 100, 100);
};

const getProgressColor = (progress: number) => {
  if (progress < 50) return 'bg-red-500';
  if (progress < 80) return 'bg-yellow-500';
  return 'bg-green-500';
};

return (
  <div className="min-h-screen pb-24">
    <div className="max-w-md mx-auto px-4 py-5">

      <div className="card-rich p-5 mb-4 animate-fade-in-up">
        <div className="flex items-center gap-3 mb-4">
          <div className="w-11 h-11 rounded-2xl bg-brand-gradient flex items-center justify-center shadow-pop">
            <Icon name="chart" size={22} color="white" />
          </div>
          <div>
            <h1 className="text-lg font-extrabold text-gray-900 leading-tight">Статистика</h1>
            <p className="text-xs text-gray-500">Твої тренди харчування</p>
          </div>
        </div>
        <div className="flex gap-1.5 bg-cream-100 p-1 rounded-2xl">
          {periods.map((period) => (
            <button
              key={period.value}
              onClick={() => setSelectedPeriod(period.value)}
              className={`flex-1 py-2 px-2 rounded-xl text-xs font-bold transition-all ${
                selectedPeriod === period.value
                  ? 'bg-white text-brand-600 shadow-soft'
                  : 'text-gray-500 hover:text-gray-800'
              }`}
            >
              {period.label}
            </button>
          ))}
        </div>
      </div>
    </div>
  </div>

```

```

    )))
  </div>
</div>

{loading ? (
  <div className="flex justify-center items-center py-12">
    <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600" />
  </div>
) : userStats ? (
  <>
    <div className="mb-4 animate-fade-in-up">
      <h2 className="text-sm font-bold text-gray-700 uppercase tracking-wider px-1 mb-2 flex items-center
gap-2">
        <span className="inline-block w-1 h-4 rounded-full bg-brand-gradient" />
        Середні показники
      </h2>
      <div className="grid grid-cols-2 gap-3">
        <div className="tile-kcal">
          <div className="relative z-10">
            <Icon name="calories" size={24} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold
mt-2">Калорії/день</div>
            <div className="text-2xl font-extrabold leading-none mt-1">
              {Math.round(userStats.averageCalories)}
            </div>
            <div className="text-[10px] text-white/75 mt-1">ціль: {calorieGoal}</div>
          </div>
        </div>
        <div className="tile-protein">
          <div className="relative z-10">
            <Icon name="protein" size={24} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold
mt-2">Білки/день</div>
            <div className="text-2xl font-extrabold leading-none mt-1">
              {Math.round(userStats.averageProtein)}<span className="text-sm font-semibold text-white/85
ml-0.5">г</span>
            </div>
          </div>
        </div>
        <div className="tile-fats">
          <div className="relative z-10">
            <Icon name="fats" size={24} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold
mt-2">Жири/день</div>
            <div className="text-2xl font-extrabold leading-none mt-1">
              {Math.round(userStats.averageFats)}<span className="text-sm font-semibold text-white/85
ml-0.5">г</span>
            </div>
          </div>
        </div>
        <div className="tile-carbs">
          <div className="relative z-10">
            <Icon name="carbs" size={24} color="white" />
            <div className="text-[10px] uppercase tracking-wider text-white/85 font-bold
mt-2">Вуглеводи/день</div>
            <div className="text-2xl font-extrabold leading-none mt-1">
              {Math.round(userStats.averageCarbs)}<span className="text-sm font-semibold text-white/85
ml-0.5">г</span>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
) : null}

```

```

{calorieChartData.length > 0 && (
  <div className="card-rich p-5 mb-4">
    <h2 className="text-lg font-semibold text-gray-900 mb-4 flex items-center space-x-2">
      <Icon name="timeline" size={20} color="blue" />
      <span>Тренд калорій</span>
    </h2>
    <ResponsiveContainer width="100%" height={200}>
      <LineChart data={calorieChartData} margin={{ top: 5, right: 8, left: -20, bottom: 5 }}>
        <CartesianGrid strokeDasharray="3 3" stroke="#f0f0f0" />
        <XAxis
          dataKey="label"
          tick={{ fontSize: 11, fill: '#9ca3af' }}
          interval={selectedPeriod <= 7 ? 0 : Math.ceil(calorieChartData.length / 6)}
        />
        <YAxis tick={{ fontSize: 11, fill: '#9ca3af' }} />
        <Tooltip content={<CaloriesTooltip />} />
        <ReferenceLine
          y={calorieGoal}
          stroke="#94a3b8"
          strokeDasharray="5 5"
          label={{ value: 'ціль', position: 'right', fontSize: 10, fill: '#94a3b8' }}
        />
        <Line
          type="monotone"
          dataKey="calories"
          name="Калорії"
          stroke={MACRO_COLORS.calories}
          strokeWidth={2.5}
          dot={{ r: 4, fill: MACRO_COLORS.calories, strokeWidth: 0 }}
          activeDot={{ r: 6 }}
        />
      </LineChart>
    </ResponsiveContainer>
  </div>
)}

{donutData.length > 0 && (
  <div className="card-rich p-5 mb-4">
    <h2 className="text-lg font-semibold text-gray-900 mb-1 flex items-center space-x-2">
      <Icon name="protein" size={20} color="green" />
      <span>Розподіл макронутрієнтів</span>
    </h2>
    <p className="text-xs text-gray-400 mb-4">Середнє за обраний період</p>
    <div className="flex items-center">
      <ResponsiveContainer width="55%" height={180}>
        <PieChart>
          <Pie
            data={donutData}
            cx="50%"
            cy="50%"
            innerRadius={50}
            outerRadius={78}
            paddingAngle={3}
            dataKey="value"
          />
          {donutData.map((_, index) => (
            <Cell key={index} fill={donutColors[index]} />
          ))}
        </Pie>
        <Tooltip
          formatter={(value: number, name: string) => [
            `${value} ккал (${Math.round(value / totalMacroKcal * 100)}%)`,
            name
          ]}
        />
      </div>
    </div>
  </div>
)}

```

```

    />
  </PieChart>
</ResponsiveContainer>
<div className="flex-1 space-y-3 pl-2">
  {donutData.map((item, i) => (
    <div key={item.name} className="flex items-center space-x-2">
      <div
        className="w-3 h-3 rounded-full flex-shrink-0"
        style={{ backgroundColor: donutColors[i] }}
      />
      <div>
        <p className="text-sm font-medium text-gray-700">{item.name}</p>
        <p className="text-xs text-gray-400">
          {item.grams}г · {Math.round(item.value / totalMacroKcal * 100)}%
        </p>
      </div>
    </div>
  ))}
</div>
</div>
</div>
))}

{macroChartData.length > 0 && (
  <div className="card-rich p-5 mb-4">
    <h2 className="text-lg font-semibold text-gray-900 mb-4 flex items-center space-x-2">
      <Icon name="carbs" size={20} color="red" />
      <span>Макроси по днях</span>
    </h2>
    <ResponsiveContainer width="100%" height={200}>
      <BarChart data={macroChartData} margin={{ top: 5, right: 8, left: -20, bottom: 5 }}>
        <CartesianGrid strokeDasharray="3 3" stroke="#f0f0f0" />
        <XAxis
          dataKey="label"
          tick={{ fontSize: 11, fill: '#9ca3af' }}
          interval={selectedPeriod <= 7 ? 0 : Math.ceil(macroChartData.length / 6)}
        />
        <YAxis tick={{ fontSize: 11, fill: '#9ca3af' }} unit="г" />
        <Tooltip content={<MacroTooltip />} />
        <Legend
          wrapperStyle={{ fontSize: 12 }}
          iconType="circle"
          iconSize={8}
        />
        <Bar dataKey="Білки" stackId="a" fill={MACRO_COLORS.protein} radius={[0, 0, 0, 0]} />
        <Bar dataKey="Жири" stackId="a" fill={MACRO_COLORS.fats} />
        <Bar dataKey="Вуглеводи" stackId="a" fill={MACRO_COLORS.carbs} radius={[4, 4, 0, 0]} />
      </BarChart>
    </ResponsiveContainer>
  </div>
))}

<div className="card-rich p-5 mb-4">
  <h2 className="text-lg font-semibold text-gray-900 mb-4 flex items-center space-x-2">
    <Icon name="progress" size={20} color="blue" />
    <span>Прогрес до цілей</span>
  </h2>
  <div className="space-y-4">
    <div>
      <div className="flex justify-between items-center mb-2">
        <div className="flex items-center space-x-2">
          <Icon name="calories" size={16} color="blue" />
          <span className="text-sm font-medium text-gray-700">Калорії</span>
        </div>

```

```

    <span className="text-sm text-gray-600">
      {Math.round(userStats.averageCalories)}/{calorieGoal}
    </span>
  </div>
  <div className="w-full bg-gray-200 rounded-full h-2">
    <div
      className={`h-2 rounded-full transition-all duration-300 $
        {getProgressColor(calculateProgress(userStats.averageCalories, calorieGoal))}`}
      style={{ width: `${calculateProgress(userStats.averageCalories, calorieGoal)}%` }}
    </div>
  </div>
  <div>
    <div className="flex justify-between items-center mb-2">
      <div className="flex items-center space-x-2">
        <Icon name="protein" size={16} color="green" />
        <span className="text-sm font-medium text-gray-700">Білки (рекомендовано: 100r)</span>
      </div>
      <span className="text-sm text-gray-600">
        {Math.round(userStats.averageProtein)}/100r
      </span>
    </div>
    <div className="w-full bg-gray-200 rounded-full h-2">
      <div
        className={`h-2 rounded-full transition-all duration-300 $
          {getProgressColor(calculateProgress(userStats.averageProtein, 100))}`}
        style={{ width: `${calculateProgress(userStats.averageProtein, 100)}%` }}
      </div>
    </div>
  </div>
</div>

<div className="card-rich p-5 mb-4">
  <h2 className="text-lg font-extrabold text-gray-900 mb-4 flex items-center gap-2">
    <span className="inline-block w-1 h-5 rounded-full bg-brand-gradient" />
    Активність
  </h2>
  <div className="space-y-2.5">
    <div className="flex items-center justify-between p-3.5 bg-gradient-to-br from-purple-50 to-purple-100/40 rounded-2xl border border-purple-200/60">
      <div className="flex items-center gap-3">
        <div className="w-10 h-10 rounded-xl bg-gradient-to-br from-purple-400 to-violet-600 flex items-center justify-center shadow-md">
          <Icon name="calendar" size={20} color="white" />
        </div>
        <div>
          <p className="font-bold text-gray-900 text-sm">Активні дні</p>
          <p className="text-xs text-gray-500">Днів з записами</p>
        </div>
      </div>
      <span className="text-2xl font-extrabold bg-gradient-to-br from-purple-500 to-violet-700 bg-clip-text text-transparent">{userStats.daysWithData}</span>
    </div>
    <div className="flex items-center justify-between p-3.5 bg-gradient-to-br from-brand-50 to-brand-100/40 rounded-2xl border border-brand-200/60">
      <div className="flex items-center gap-3">
        <div className="w-10 h-10 rounded-xl bg-brand-gradient flex items-center justify-center shadow-md">
          <Icon name="plate" size={20} color="white" />
        </div>
        <div>
          <p className="font-bold text-gray-900 text-sm">Всього прийомів</p>
          <p className="text-xs text-gray-500">За обраний період</p>
        </div>
      </div>
    </div>
  </div>

```

```

        </div>
      </div>
      <span className="text-2xl font-extrabold text-gradient-brand">{userStats.totalMeals}</span>
    </div>
    <div className="flex items-center justify-between p-3.5 bg-gradient-to-br from-fresh-50 to-fresh-100/40
rounded-2xl border border-fresh-300/60">
      <div className="flex items-center gap-3">
        <div className="w-10 h-10 rounded-xl bg-fresh-gradient flex items-center justify-center shadow-
md">
          <Icon name="average" size={20} color="white" />
        </div>
        <div>
          <p className="font-bold text-gray-900 text-sm">Середньо/день</p>
          <p className="text-xs text-gray-500">Регулярність</p>
        </div>
      </div>
      <span className="text-2xl font-extrabold text-gradient-fresh">
        {userStats.daysWithData > 0
        ? Math.round((userStats.totalMeals / userStats.daysWithData) * 10) / 10
        : 0}
      </span>
    </div>
  </div>
</div>

<div className="card-rich p-5 mb-4">
  <h2 className="text-lg font-semibold text-gray-900 mb-4 flex items-center space-x-2">
    <Icon name="lightbulb" size={20} color="yellow" />
    <span>Поради</span>
  </h2>
  <div className="space-y-3">
    {userStats.averageCalories < calorieGoal * 0.8 && (
      <div className="p-3 bg-yellow-50 border-l-4 border-yellow-400 rounded-lg">
        <div className="flex items-start space-x-2">
          <Icon name="warning" size={16} color="yellow" />
          <p className="text-sm text-yellow-800">
            <strong>Недостатньо калорій:</strong> Ви споживаєте менше калорій ніж потрібно.
            Спробуйте додати більше поживних продуктів.
          </p>
        </div>
      </div>
    )}
    {userStats.averageProtein < 80 && (
      <div className="p-3 bg-blue-50 border-l-4 border-blue-400 rounded-lg">
        <div className="flex items-start space-x-2">
          <Icon name="protein" size={16} color="blue" />
          <p className="text-sm text-blue-800">
            <strong>Більше білка:</strong> Додайте в раціон м'ясо, рибу, яйця або бобові
            для досягнення рекомендованої норми білка.
          </p>
        </div>
      </div>
    )}
    {userStats.daysWithData / selectedPeriod > 0.8 && (
      <div className="p-3 bg-green-50 border-l-4 border-green-400 rounded-lg">
        <div className="flex items-start space-x-2">
          <Icon name="success" size={16} color="green" />
          <p className="text-sm text-green-800">
            <strong>Відмінно!</strong> Ви регулярно відстежуєте своє харчування.
            Продовжуйте в тому ж дусі!
          </p>
        </div>
      </div>
    )}
  </div>
</div>

```

```

        {userStats.daysWithData > 0 && userStats.totalMeals / userStats.daysWithData < 3 && (
          <div className="p-3 bg-orange-50 border-l-4 border-orange-400 rounded-lg">
            <div className="flex items-start space-x-2">
              <Icon name="clock" size={16} color="orange" />
              <p className="text-sm text-orange-800">
                <strong>Регулярність:</strong> Спробуйте їсти 3-4 рази на день для
                кращого метаболізму та контролю голоду.
              </p>
            </div>
          </div>
        )}
      </div>
    </div>
  </>
): (
  <div className="card-rich p-8 text-center">
    <div className="mb-4 flex justify-center">
      <Icon name="chart" size={96} color="gray" />
    </div>
    <h3 className="text-lg font-medium text-gray-900 mb-2">Статистика недоступна</h3>
    <p className="text-gray-600">Почніть відстежувати харчування для перегляду статистики</p>
  </div>
)
</div>
</div>
);
};

export default Statistics;

```

Achievements.tsx – КОМПОНЕНТ ПІДСИСТЕМИ ДОСЯГНЕНЬ:

```

import React, { useState, useEffect, useCallback } from 'react';
import { achievementsApi } from '../services/api';
import type { AchievementDto } from '../types';
import Icon from '../components/Icon';

const Achievements: React.FC = () => {
  const [achievements, setAchievements] = useState<AchievementDto[]>([]);
  const [loading, setLoading] = useState(true);
  const [checking, setChecking] = useState(false);
  const [newlyUnlocked, setNewlyUnlocked] = useState<AchievementDto[]>([]);

  const load = useCallback(async () => {
    try {
      const res = await achievementsApi.getAll();
      setAchievements(res.data);
    } catch (e) {
      console.error('Failed to load achievements', e);
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => { load(); }, [load]);

  const handleCheck = async () => {
    setChecking(true);
    try {
      const res = await achievementsApi.check();
      if (res.data.length > 0) {
        setNewlyUnlocked(res.data);
      }
    }
  };

```

```

    await load();
  }
} catch (e) {
  console.error('Failed to check achievements', e);
} finally {
  setChecking(false);
}
};

const unlockedCount = achievements.filter(a => a.isUnlocked).length;
const totalCount = achievements.length;

const categories = Array.from(new Set(achievements.map(a => a.category)));

const formatDate = (dateStr?: string) => {
  if (!dateStr) return "";
  return new Date(dateStr).toLocaleDateString('uk-UA', {
    day: 'numeric', month: 'long', year: 'numeric'
  });
};

return (
  <div className="min-h-screen pb-24">
    <div className="max-w-md mx-auto px-4 py-6">

      <div className="bg-white rounded-2xl shadow-sm p-6 mb-4">
        <h1 className="text-xl font-bold text-gray-900 text-center mb-4 flex items-center justify-center gap-2">
          <Icon name="trophy" size={24} color="yellow" />
          <span>Досягнення</span>
        </h1>

        <div className="mb-3">
          <div className="flex justify-between text-sm text-gray-600 mb-1">
            <span>Прогрес</span>
            <span className="font-semibold text-blue-600">{unlockedCount} / {totalCount}</span>
          </div>
          <div className="w-full bg-gray-200 rounded-full h-3">
            <div
              className="h-3 rounded-full bg-gradient-to-r from-brand-500 to-fresh-500 transition-all duration-500"
              style={{ width: totalCount > 0 ? `${(unlockedCount / totalCount) * 100}%` : '0%' }}
            </div>
          </div>
        </div>

        <button
          onClick={handleCheck}
          disabled={checking}
          className="w-full py-2.5 bg-blue-600 text-white rounded-xl text-sm font-semibold hover:bg-blue-700
disabled:opacity-60 transition-colors"
        >
          {checking ? 'Перевіряємо...' : 'Перевірити нові досягнення'}
        </button>
      </div>

      {newlyUnlocked.length > 0 && (
        <div className="bg-gradient-to-r from-yellow-400 to-orange-400 rounded-2xl shadow-sm p-4 mb-4">
          <p className="text-white font-bold text-center mb-2">Нові досягнення розблоковано!</p>
          <div className="space-y-2">
            <div key={a.code} className="bg-white bg-opacity-20 rounded-xl p-3 flex items-center space-x-3">
              <Icon name={a.iconName} size={28} color="white" />
              <div>
                <p className="text-white font-semibold text-sm">{a.title}</p>
                <p className="text-white text-xs opacity-80">{a.description}</p>
              </div>
            </div>
          </div>
        </div>
      )}
    </div>
  </div>
);

```

```

        </div>
      </div>
    )}
  </div>
  <button
    onClick={() => setNewlyUnlocked([])}
    className="w-full mt-3 py-1.5 bg-white bg-opacity-20 text-white text-sm rounded-lg hover:bg-opacity-30
transition"
  >
    Закрити
  </button>
</div>
)}

{loading ? (
  <div className="flex justify-center items-center py-12">
    <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600" />
  </div>
) : (
  categories.map(category => {
    const items = achievements.filter(a => a.category === category);
    const catUnlocked = items.filter(a => a.isUnlocked).length;

    return (
      <div key={category} className="bg-white rounded-2xl shadow-sm p-5 mb-4">
        <div className="flex items-center justify-between mb-4">
          <h2 className="text-base font-semibold text-gray-900">{category}</h2>
          <span className="text-xs text-gray-500 bg-gray-100 px-2 py-1 rounded-full">
            {catUnlocked}/{items.length}
          </span>
        </div>

        <div className="space-y-3">
          {items.map(achievement => (
            <div
              key={achievement.code}
              className={`flex items-center space-x-4 p-3 rounded-xl transition-all ${
                achievement.isUnlocked
                  ? 'bg-gradient-to-r from-blue-50 to-purple-50 border border-blue-100'
                  : 'bg-gray-50 opacity-60'
              }`}
            >
              <div className="flex-shrink-0 w-10 h-10 rounded-full bg-white flex items-center justify-center"
                style={{ filter: achievement.isUnlocked ? 'none' : 'grayscale(1) opacity(0.5)' }}>
                <Icon
                  name={achievement.isUnlocked ? achievement.iconName : 'cancel'}
                  size={28}
                  color={achievement.isUnlocked ? 'blue' : 'gray'}
                />
              </div>

              <div className="flex-1 min-w-0">
                <p className={`font-semibold text-sm ${achievement.isUnlocked ? 'text-gray-900' : 'text-
gray-500'}`}>
                  {achievement.title}
                </p>
                <p className="text-xs text-gray-500 truncate">{achievement.description}</p>
                {achievement.isUnlocked && achievement.unlockedAt && (
                  <p className="text-xs text-blue-500 mt-0.5">
                    {formatDate(achievement.unlockedAt)}
                  </p>
                )}
              </div>
            </div>
          )}
        </div>
      </div>
    )
  )}

```

```

    {achievement.isUnlocked && (
      <div className="flex-shrink-0">
        <div className="w-6 h-6 bg-green-500 rounded-full flex items-center justify-center">
          <svg className="w-4 h-4 text-white" fill="none" stroke="currentColor" viewBox="0 0 24
24">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2.5} d="M5 13l4
4L19 7" />
          </svg>
        </div>
      </div>
    )}
  </div>
))}
</div>
</div>
);
})
})
</div>
</div>
);
};

export default Achievements;

```

ДОДАТОК Д
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ
ДЕКЛАРАЦІЯ ЗДОБУВАЧА

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

1. Моя бакалаврська кваліфікаційна робота (проект) на тему «Фронтенд частина аналітичної системи підрахунку калорій при вживанні їжі людиною» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - [*] інструменти генеративного ШІ не використовувалися.
 - [*] інструменти ШІ Gemini, Claude та DeepL 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 відрядження з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____

(підпис)

Владислав ПРАВДИВЕЦЬ