Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 − Інженерія програмного забезпечення

На тему: Розробка вебзастосунку для планування раціону та фізичної активності

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студентка гр. ІПЗ-21-2

_____ / Д. В. Сокол /

Керівник
кваліфікаційної роботи _____ / О. Г. Рибальченко /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 − Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«___»_____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студентці групи ІПЗ-21-2 Сокол Дар'ї Вікторівні

1. Тема: Розробка вебзастосунку для планування раціону та фізичної активності затверджено наказом по КНУ №200с від «10» квітня 2025 р.
2. Термін подання студентом закінченої роботи: «10» червня 2025 р.
3. Вихідні дані по роботі: розроблений вебзастосунок має ефективно оптимізувати щоденні процеси комплексного управління й моніторингу харчування та фізичної активності, надавати персоналізовані рекомендації, мати інтуїтивно-зрозумілий адаптивний інтерфейс.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): вивчити літературні джерела за темою, провести аналіз наявних аналогів, розробити математичну модель, сформувати функціональні можливості та обмеження профілів користувачів, спроєктувати архітектуру та інтерфейс застосунку, реалізувати інтерфейс користувача вебзастосунку, забезпечити кросбраузерність та адаптивність, створити цілісну базу даних, реалізувати серверну частину, виконати тестування продукту.
5. Перелік ілюстративного матеріалу: структурна схема, діаграма прецедентів (варіантів використання), контекстна діаграма, діаграма послідовностей, схема бази даних, вайрфрейми сторінок, знімки екрану інтерфейсів розробленого вебзастосунку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літературних джерел відповідно до теми кваліфікаційної роботи	31.03.2025 – 04.04.2025
2	Пошук та проведення аналізу існуючих у предметній області аналогів програмних продуктів	05.04.2025– 10.04.2025
3	Формулювання актуальності та мети і визначення завдань роботи	11.04.2025– 13.04.2025
4	Підготовка матеріалів першого розділу роботи	14.04.2025 – 17.04.2025
5	Розробка математичного апарату, діаграм архітектури програмного продукту, проєктування бази даних	18.04.2025– 21.04.2025
6	Створення вайрфреймів UI-компонентів вебзастосунку	22.04.2025– 23.04.2025
7	Підготовка матеріалів другого розділу роботи	24.04.2025 – 28.04.2025
8	Вибір технологій реалізації проєкту, розробка front-end частини вебзастосунку	29.04.2025– 13.05.2025
9	Створення бази даних, розробка back-end частини вебзастосунку	14.05.2025– 30.05.2025
10	Тестування розробленого програмного продукту, усунення виявлених дефектів	31.05.2025– 01.06.2025
11	Підготовка матеріалів третього розділу роботи	02.06.2025– 04.06.2025
12	Оформлення пояснювальної записки	05.06.2025– 09.06.2025

Дата видачі завдання: «11» квітня 2025 р.

Студентка: _____ / Д. В. Сокол /

Керівник роботи: _____ / О. Г. Рибальченко /

РЕФЕРАТ

ВЕБЗАСТОСУНОК, АНАЛІЗ, ПЛАНУВАННЯ, РАЦІОН, ФІЗИЧНА АКТИВНІСТЬ, ВІДСЛІДКОВУВАННЯ ПРОГРЕСУ, DJANGO

Пояснювальна записка: 64 с., 1 табл., 38 рис., 2 дод., 21 джерел.

Метою роботи є розробка вебзастосунку «Nutrizen» для планування раціону та фізичної активності. Він призначений допомагати користувачам, які прагнуть оптимізувати власну вагу або ж просто підтримувати себе у формі та дотримуватися принципів раціонального харчування, витрачаючи мінімум часу на пошук рецептів і складання збалансованого меню. Застосунок дозволяє відстежувати рівень активності та вміст нутрієнтів у спожитих стравах, а також на основі кількості засвоєних калорій упродовж дня пропонує необхідний обсяг фізичних навантажень.

Об’єкт проєктування – вебзастосунок, який забезпечує сучасний інтуїтивно-зрозумілий адаптивний інтерфейс і дозволяє ефективно оптимізувати щоденні процеси комплексного управління й моніторингу харчування та фізичної активності.

У даній кваліфікаційній роботі проведено комплексну оцінку переваг і недоліків впровадження програмних продуктів у предметній області, виконано аналіз наявних аналогів і реалізовано вебзастосунок для планування раціону та фізичної активності. У ході розробки особливу увагу було приділено створенню зрозумілого та зручного застосунку, який за допомогою діаграм і таблиць візуалізує показники, що характеризують спосіб життя та харчування. Зроблено акцент на багатофункціональності: користувачі отримують рекомендації фізичної активності та можуть вносити інформацію з фітнес-браслетів чи мобільних додатків, мають змогу самостійно вписувати страви у денний раціон або ж автоматично генерувати плани харчування відповідно до своїх потреб, у вебзастосунку представлено корисні статті про спорт, важливість макроелементів та ін., а також є велика база рецептів і калькулятори харчової цінності продуктів, індексу маси тіла тощо.

ABSTRACT

WEB APPLICATION, ANALYSIS, PLANNING, MEAL, PHYSICAL ACTIVITY, PROGRESS TRACKING, DJANGO

Thesis in 64 p., 1 tab., 38 fig., 2 app., 21 references.

The purpose of this work is to develop a web application “Nutrizen” for diet and physical activity planning. It is designed to help users who seek to optimize their own weight or just keep themselves fit and adhere to the principles of rational nutrition, spending a minimum of time searching for recipes and creating a balanced menu. The application allows to track the level of activity and the content of nutrients in the dishes consumed, and also, based on the number of calories consumed during the day, suggests the necessary amount of physical activity.

The object of the design is a web application that provides a modern user-friendly adaptive interface and allows to effectively optimize the daily process of complex management and monitoring of nutrition and physical activity.

In this qualifying paper, a comprehensive assessment of the advantages and disadvantages of implementing software products in the subject area was caried out, an analysis of existing analogues was performed and a web application for meal and physical activity planning was implemented. During the development special attention was paid to an intuitively understandable and convenient application interface that visualizes indicators characterizing lifestyle and nutrition using diagrams and tables. The emphasis was placed on multifunctionality: users receive physical activity recommendations and can enter information from fitness bracelets or mobile applications, are able to independently enter meals into their daily diet or automatically meal plans according to their needs, the web application presents useful articles about sports, the importance of macronutrients etc., and also has a large database of recipes and calculators of the nutrition value of products, body mass index etc.

ЗМІСТ

ВСТУП

Сьогодні типовий день більшості людей проходить у постійному поспіху, тому питання збалансованого харчування нерідко відкладається у довгий ящик. Одним не вистачає часу на підбір продуктів для приготування корисних страв, тому вони надають перевагу швидким перекусам, а інші – взагалі можуть пропускати деякі прийоми їжі. При цьому люди часто не відслідковують свою фізичну активність, що в комплексі із розбалансованим харчуванням може не лише негативно вплинути на фігуру чи стан шкіри, а й призвести до погіршення загального самопочуття та навіть виникнення серйозних проблем зі здоров'ям.

Технології зараз так чи інакше набули поширення на усі сфери нашого життя. Навіть у частині харчування та фізичної активності вже існує велике різноманіття застосунків, які покликані допомогти рахувати калорії, слідкувати за балансом нутрієнтів, складати меню та контролювати фізичну активність. У результаті більше людей можуть приділяти увагу своєму раціону та стилю життя, витрачаючи при цьому зовсім мало часу, а це дуже важливо у шаленому ритмі сучасного світу.

Мета роботи – розробити вебзастосунок, який буде поєднувати ключові функції для зручного і зрозумілого планування раціону та фізичної активності, а також міститиме додаткові можливості, які б вирізняли його на ринку серед уже наявних подібних рішень.

Реалізація цього проєкту дозволить створити інноваційний конкурентоспроможний інструмент, який відповідатиме потребам користувачів, поєднуючи новітні технології та користь у повсякденному житті.

1 СУЧАСНИЙ СТАН ЦИФРОВИХ ІНСТРУМЕНТІВ ПЛАНУВАННЯ РАЦІОНУ ТА АКТУАЛЬНІСТЬ РОБОТИ

1.1 Актуальний стан професійної області

Згідно із даними Центру громадського здоров'я МОЗ України близько 20% українців працездатного віку страждають на ожиріння та майже 25% – мають надлишкову масу тіла. Причинами виникнення та стрімкого поширення проблеми є: малорухливий спосіб життя, зловживання продуктами харчування із високим вмістом жиру, солі та цукру і нестача в раціоні овочів, фруктів, риби, яєць, молочних продуктів тощо [1].

Всесвітня організація охорони здоров'я (ВООЗ) вже тривалий час вважає ожиріння глобальною проблемою, а неправильне харчування – її ключовою причиною. Наслідки, спричинені набором надлишкової ваги, становлять серйозну загрозу здоров'ю осіб будь-якого віку і включають, зокрема, підвищення ризику розвитку серцево-судинних захворювань, виникнення діабету, онкологічних захворювань та проблем із шлунково-кишковим трактом і опорно-руховим апаратом.

Щорічно запускається велика кількість інформаційних кампаній на тему здорового харчування і фізичної активності та розробляються інформаційні бюлетені, плакати і листівки з метою донесення рекомендацій, що допоможуть знизити ризик розвитку вищезгаданих неінфекційних захворювань, які виникають внаслідок нездорового набору ваги.

Загалом усі рекомендації зводяться до того, що необхідно контролювати КБЖВ спожитої їжі, дотримуватися збалансованого розпорядку харчування, урізноманітнювати щоденний раціон продуктами із різних категорій, підтримувати водний баланс, а також кожного дня приділяти час тренуванням чи іншим видам фізичної активності [2]. Проте сучасній людині буває досить важко дотримуватися таких, здавалося б, простих порад, насамперед, через брак часу. І тут допомагають смартфони, планшети та розумні годинники – відслідковувати отриману енергетичну цінність, спожиті нутрієнти та рівень

фізичної активності стає набагато простіше і швидше, адже все це робить один або декілька різних застосунків, а людина лише вносить дані та контролює свій прогрес.

Розвиток сфери інформаційних технологій важко не помітити, адже автоматизація навіть таких буденних речей як планування раціону й активності дозволяє не лише спростити ці задачі, а й побачити загальну картину споживання і витрати калорій, макроелементів, коли і скільки людина рухається – знання про це дають змогу підходити більш структуровано до питань організації харчування і дозвілля самостійно або ж спонукають звернутися за допомогою до лікарів, якщо показники незадовільні. Більшість застосунків дозволяють врахувати індивідуальні потреби, як наприклад, наявність алергії, харчові вподобання та інші. Також ще однією перевагою використання подібних додатків є отримання мотивації продовжувати вести здоровий спосіб життя через виконання щоденних цілей і перегляд графіків прогресу, так із часом у людини вибудовується дисциплінованість, яка згодом переростає у корисну звичку. Крім того, такі застосунки виконують ще й освітню функцію для тих, хто бажає дізнатися більше про здорове харчування та фізичні навантаження.

Водночас, використання застосунків для моніторингу харчування та фізичної активності може мати не лише переваги, а й ряд недоліків, серед яких: можливі неточності у розрахунках через введення даних користувачем вручну або ж застарілі дані в базі; виникнення тривожності чи навіть розладів харчової поведінки (РХП) внаслідок надмірної залежності від показників у додатку; питання надійності збереження чутливої інформації про стан здоров’я, активність та харчові вподобання користувачів.

1.2 Аналіз наявних аналогів вебзастосунків у предметній галузі

Нині існує велика кількість мобільних додатків для планування харчування та відстеження фізичної активності, проте у розрізі саме

веборієнтованих застосунків спостерігається порівняне зменшення кількості існуючих програмних продуктів.

Для аналізу було обрано три найпопулярніші вебдодатки, які допомагають користувачам складати плани харчування, відслідковувати спожиті КБЖВ, контролювати масу тіла та активність або ж просто дізнатися більше про здорове харчування.

Першим було розглянуто сервіс «My Diet Meal Plan» [3]. На рисунку 1.1 показано, що додаток має інтуїтивно-зрозумілий інтерфейс та доволі сучасний дизайн. Для неавторизованих користувачів доступні статті на теми харчування та здоров'я, калькулятори КБЖВ і відсотка жиру в організмі, а також можна обрати бажану дієту з-поміж існуючих або ж згенерувати власний план харчування під конкретні потреби. Для доступу до планів та можливості відслідковування прогресу необхідно авторизуватися у системі, вказавши електронну адресу, ім'я користувача та пароль.

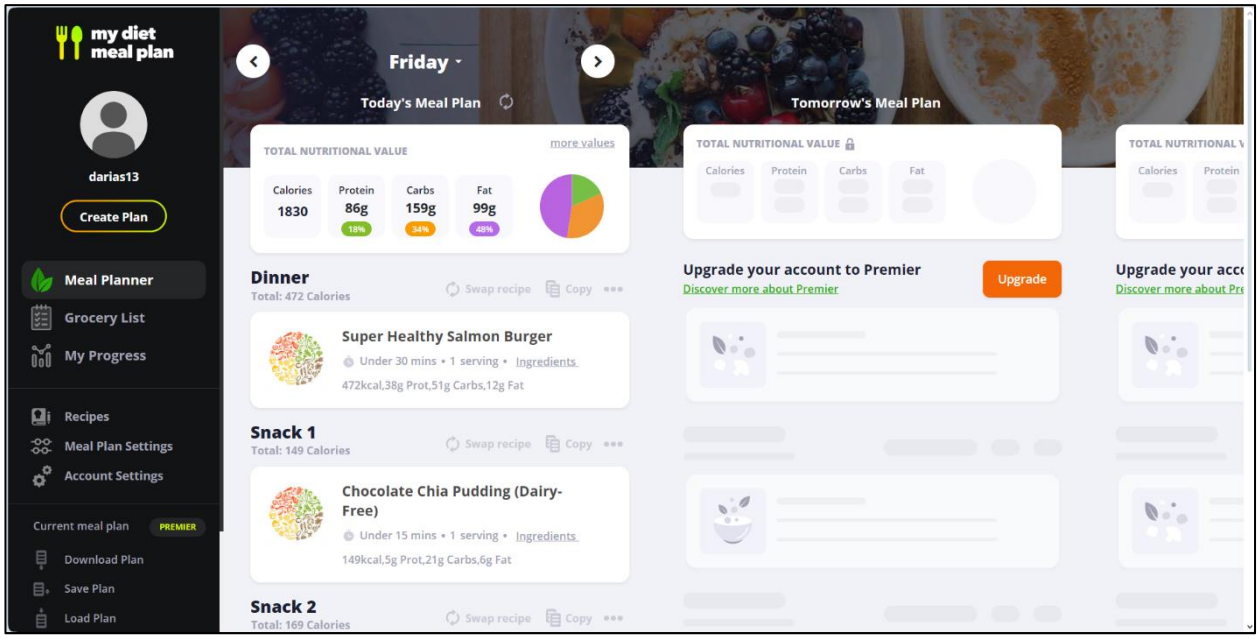

Рисунок 1.1 – Інтерфейс сторінки планування раціону в «My Diet Meal Plan»

Основними перевагами цього вебзастосунку є: надання високого рівня персоналізації для задоволення індивідуальних потреб та смаків користувачів (наприклад, налаштування у планах оптимального співвідношення білків,

жирів і вуглеводів для схуднення чи, навпаки, набору м’язової маси); велика база рецептів (безглютенові, веганські та інші страви); генерація раціону на тиждень разом зі списком покупок необхідних для приготування продуктів.

До недоліків «My Diet Meal Plan» варто віднести відсутність модулів для внесення інформації про активність та її відслідковування; неточне врахування потреб користувачів – певні рецепти зі згенерованого плану можуть не зовсім точно відповідати смакам користувача або ж містити продукти, на які в людини алергія; відсутність української мови інтерфейсу; а також те, що статті та калькулятори доступні лише неавторизованим користувачам.

Наступним було досліджено вебдодаток «Таблиця калорійності» [4]. Інтерфейс цього сервісу є доволі простим і зрозумілим (див. рис. 1.2), проте часто трапляються проблеми із відображенням елементів навігації чи цілих сторінок, що має негативний вплив на користувацький досвід. Відсутність готових планів дієт і можливості їхнього генерування під конкретні потреби пояснюється тим, що цей продукт має на меті лише контроль та аналіз харчування, тобто кожен користувач має вибудовувати свій раціон самостійно на основі розрахованої у додатку норми КБЖВ або ж із допомогою дієтолога.

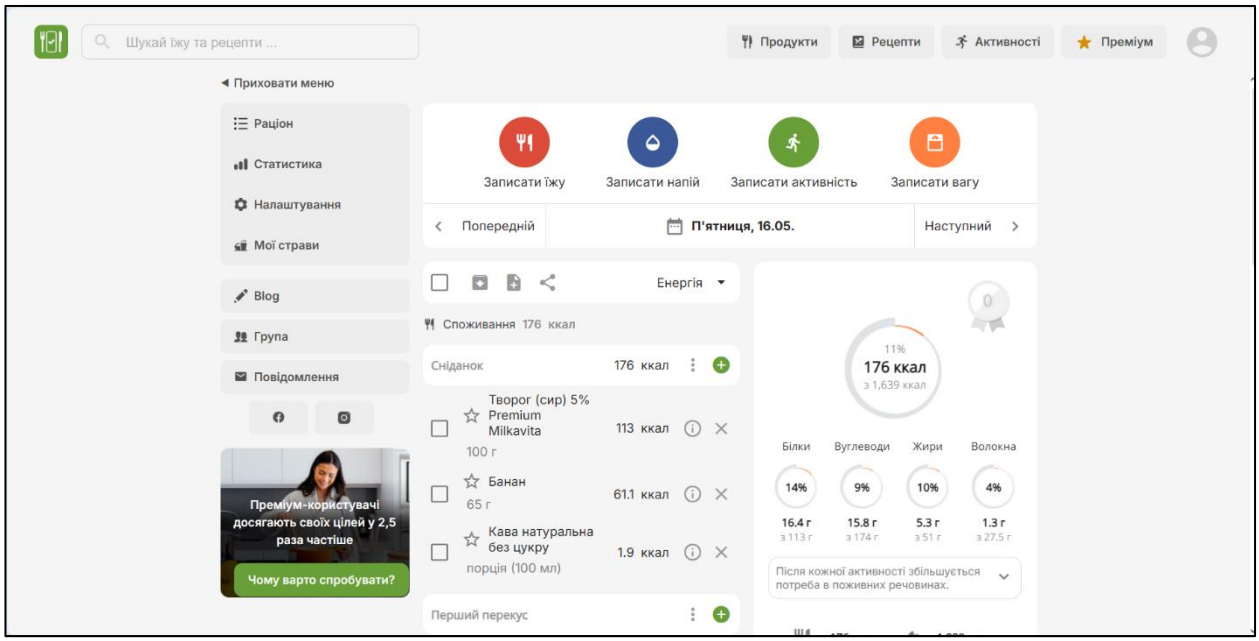

Рисунок 1.2 – Аналіз денного раціону у вебзастосунку «Таблиця калорійності»

До переваг цього програмного продукту можна віднести: детальну статистику та аналіз раціону і фізичних навантажень – це дозволяє не лише слідкувати за дотриманням денної норми КБЖВ і активності, а також відслідковувати прогрес зміни ваги, додержання водного балансу тощо; детальне подання інформації не лише у табличному та цифровому вигляді, а й за допомогою діаграм; блог із великою кількістю статей та рубрик на різні теми, які стосуються здорового способу життя.

Натомість серед недоліків «Таблиці калорійності» слід виділити проблеми у стабільності роботи вебзастосунку і тривалості очікування завантаження деяких сторінок; відсутність у рецептах позначок про наявність продуктів-алергенів; а також нагромадження реклами у безкоштовній версії, яка часто заважає переглядати інформацію (користувачів попереджають, що придбавши преміум підписку цього можна уникнути).

Останній, третій продукт, «Eating Well» [5] – електронний журнал про здорове харчування. Зовнішній вигляд його сторінок (див. рис. 1.3) нагадує новинний вебсайт, навігація зрозуміла та вже звична для користувачів. На сайті зібрано велику кількість статей, які охоплюють різні теми у сфері здорового харчування, навіть є новини про корисні харчові звички зірок і їхні кулінарні лайфхаки, а також тут цікаво та просто описано наукові дослідження у царині раціонального харчування. Плани відомих дієт із рецептами відповідних страв окремо зібрані та винесені в якості пунктів навігаційного меню. Також для людей із діабетом наявні спеціальні підбірки рецептів і статті від медиків з порадами щодо підтримання оптимального рівня цукру в крові.

Перевагами додатку є: велика кількість корисної інформації про здорове харчування, яка точно стане в нагоді людям, які тільки починають вибудовувати свій раціон; добірки кулінарних секретів і лайфхаків, які спрощують новачкам процес приготування їжі; посилання на першоджерела наукових досліджень, наведені під статтями та оглядами, які дозволяють користувачам глибше розібратися у темі або ж просто бути впевненими у достовірності прочитаної інформації.

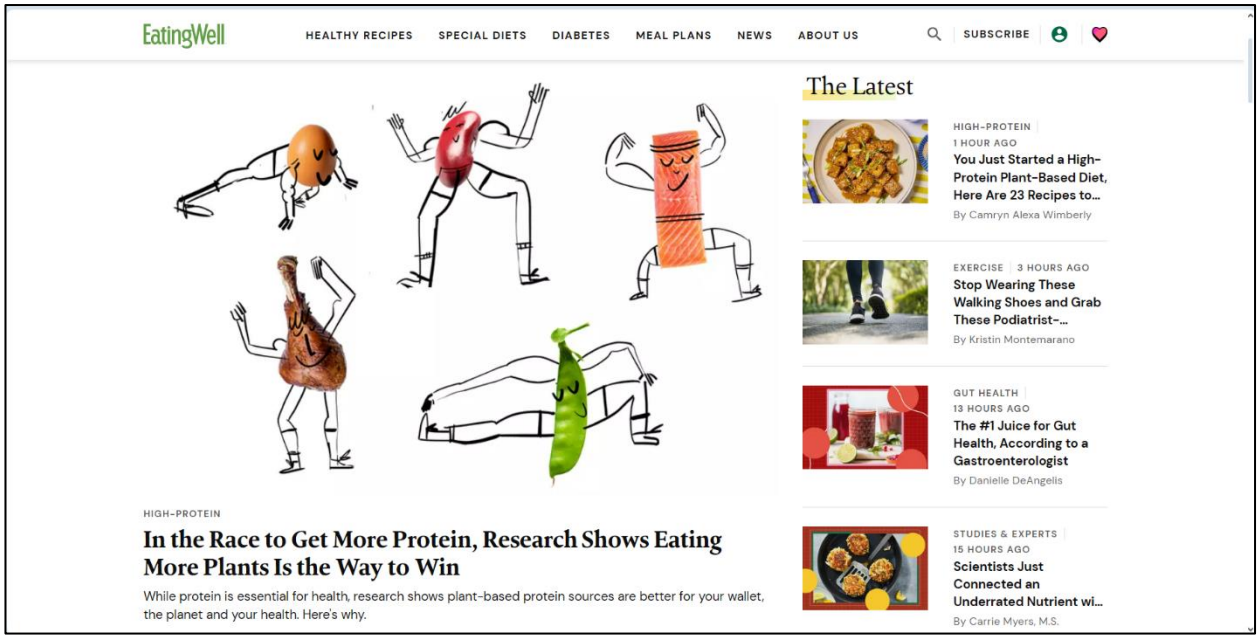

Рисунок 1.3 – Головна сторінка «Eating Well»

Основними недоліками «Eating Well» є відсутність можливості коментування статей та оглядів, лише позначки лайк і дизлайк, а також представлення інформації про вміст макроелементів у стравах виключно у цифровому та табличному вигляді без візуалізації, що ускладнює сприйняття.

Нижче у таблиці 1.1 наведено узагальнений порівняльний аналіз розглянутих вебзастосунків.

Таблиця 1.1 – Узагальнений порівняльний аналіз існуючих аналогів

Критерій	**Назва вебзастосунку**		
	My Diet Meal Plan	**Таблиця калорійності**	**Eating Well**
1	**2**	**3**	**4**
Адаптивність інтерфейсу	Частково	Частково	Повністю
Наявність безкоштовної версії	Наявна	Наявна	Повністю безкоштовний
Можливість генерації прийомів їжі чи планів харчування	Наявна	Відсутня	Відсутня

Продовження таблиці 1.1

1	2	3	4
Персоналізація налаштувань застосунку	Наявна	Наявна	Відсутня
Калькулятори КБЖВ, ІМТ і ін.	Наявні	Наявні	Відсутні
Завантаження або друк планів чи окремих рецептів	Лише для користувачів із преміум підпискою	Лише для користувачів із преміум підпискою	Відсутнє
Автоматичне створення списків покупок на основі плану харчування	Наявне	Відсутнє	Відсутнє
Наявність бази рецептів, статей	Наявна	Наявна	Наявна
Наявність готових планів харчування	Наявні	Відсутні	Наявні
Самостійне планування раціону, додавання рецептів	Наявне	Наявне	Відсутнє
Внесення інформації про активність, відстеження прогресу	Відсутнє	Наявне	Відсутнє
Персоналізовані пропозиції активності	Відсутні	Відсутні	Відсутні
Аналіз та збереження історії харчування й активності	Наявне	Наявне	Відсутнє
Зворотний зв’язок	Відсутній	Наявний	Наявний
Українська мова інтерфейсу	Відсутня	Наявна	Відсутня

Проведений аналіз показав, що існують різні вебсервіси, які допомагають користувачам дотримуватися принципів здорового харчування, й усі вони мають як переваги, так і недоліки. Найбільше необхідного функціоналу сконцентровано у перших двох вебзастосунках – «My Diet Meal Plan» і «Таблиця калорійності», адже вони дозволяють не лише шукати рецепти чи збагачувати свої знання у сфері здорового харчування, а й аналізувати та навіть створювати раціон під конкретні потреби. Під час розробки власного вебзастосунку будуть враховані результати аналізу цих програмних продуктів.

1.3 Формулювання актуальності та мети роботи

Незважаючи на глобальний розвиток у різних сферах, ми досі стикаємося із проблемою нераціонального харчування, більш того у розвинених країнах продовжує спостерігатися тенденція зростання кількості хронічних неінфекційних захворювань, причиною яких є малорухливий спосіб життя та незбалансований раціон. За даними ВООЗ більше половини населення країн із високим рівнем розвитку мають надлишкову вагу та, відповідно, знаходяться у групі ризику появи діабету, патологій серцево-судинної, травної, опорно-рухової систем, а також виникнення різних видів онкологічних захворювань.

Статистика лякає, але разом із тим спонукає до пошуку шляхів вирішення проблеми. За даних обставин як для окремих людей, так і для урядів держав ключовою є необхідність формування звичок здорового харчування та регулярної фізичної активності. Для цього доцільним є застосування цифрових інструментів, зокрема вебзастосунків, які дозволять автоматизувати рутинні процеси планування, обчислення і аналізу, значно спростивши задачу людям.

Аналіз наявних рішень показав, що вже розроблені вебдодатки, хоч і виконують більшість потрібних функцій, також мають низку недоліків – часто не надають підтримку індивідуальних цілей користувачів і не реалізують комплексний підхід до здорового способу життя, роблячи акцент лише на

харчуванні або лише на фізичній активності. Тож важливо створити продукт, у якому буде все необхідне для користувачів, які бажають харчуватися правильно і турбуватися про своє здоров’я.

Мета роботи полягає у розробці вебзастосунку, який забезпечить комплексний підхід до формування збалансованого раціону та достатнього рівня фізичної активності, а також поєднуватиме у собі як інформативну, так і аналітичну та планувальну складові. Персоналізовані рекомендації фізичного навантаження, що доповнюють харчовий раціон, сприятимуть більш швидкому досягненню оптимальних результатів. При цьому додаток повинен мати зрозумілий для користувача інтерфейс, візуалізацію даних у вигляді діаграм і в табличному поданні та багато корисної і пізнавальної інформації про здорове харчування та фізичну активність.

1.4 Визначення завдань кваліфікаційної роботи

Для досягнення мети роботи виділимо наступні завдання, які були сформульовані за результатами аналізу існуючих аналогів вебзастосунків для планування і моніторингу харчування та фізичної активності:

– розробити математичну модель вебзастосунку;

– сформувати профілі користувачів, визначити їхні функціональні можливості та обмеження;

– розробити UML-діаграми для проєктування майбутнього продукту;

– створити вайрфрейми візуального інтерфейсу вебзастосунку;

– розглянути й обрати інструменти та технології реалізації і середовище розробки;

– реалізувати інтерфейс користувача вебзастосунку;

– забезпечити кросбраузерність та адаптивність до різних розмірів екранів;

– створити базу даних, для забезпечення цілісності даних і каскадного оновлення та видалення пов’язаних полів встановити зв’язки між таблицями;

– реалізувати серверну частину вебзастосунку;

– забезпечити безпеку конфіденційної інформації користувачів;

– виконати тестування реалізованого програмного продукту.

Розробка вебзастосунку для планування раціону та фізичної активності забезпечить користувачам повноцінний персоналізований підхід до здорового способу життя, зробить доступнішим і значно спростить процес аналізу та моніторингу харчування і навантажень, а також сприятиме формуванню здорових звичок та допоможе запобігти виникненню і розвитку поширених захворювань. У ході реалізації програмного продукту особливу увагу необхідно приділити розробці математичного апарату, створенню логічно структурованої і масштабованої бази даних, яка забезпечить цілісність і зв’язність даних. Крім того, необхідно створити інтуїтивно зрозумілий інтерфейс, аби залучити якомога більше користувачів і максимально спросити для них процес взаємодії із застосунком та уникнути додаткових проблем.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Розробка математичної моделі

Одним із ключових етапів створення вебзастосунку для планування раціону та фізичної активності є розробка математичної моделі, адже обчислень у подібному програмному рішенні передбачається багато й усі вони мають бути максимально ефективними і точними.

Математичною моделлю називають сукупність математичних об’єктів (формул, рівнянь, нерівностей тощо), які використовуються для опису суттєвих властивостей, станів або процесів досліджуваного об’єкта чи системи задля його аналізу або прогнозування [6]. У контексті розробки даного вебзастосунку така модель необхідна для формалізації процесу обчислення показників КБЖВ раціону кожного користувача з урахуванням індивідуальних потреб, а також для забезпечення раціонального підґрунтя персоналізованих рекомендацій активності.

Насамперед, для ефективного планування раціону необхідно визначити загальні добові енергетичні витрати організму (TDEE), а для цього слід знати базальну швидкість обміну речовин в організмі (BMR) або ж близький за значенням показник енерговитрат в умовах мінімальної активності (REE), який розраховується за менш суворих вимог [7]. REE – це кількість калорій, необхідна організму у стані повного спокою для підтримання основних життєвих функцій (дихання, терморегуляції тощо) та процесів травлення і всмоктування поживних речовин. Формули Харріса – Бенедикта та Мафліна – Сан Жеора [8] є найвідомішими для розрахунку цього показника. Загалом вони подібні за своєю логікою, проте, незважаючи на можливі ускладнення в процесі обчислень, буде використовуватися саме формула Мафліна – Сан Жеора (2.1), адже вона серед іншого враховує рівень активності людини, тому отриманий результат буде більш орієнтованим на індивідуальні особливості кожного користувача [9].

$$\text{Для жінок: } \mathrm{REE} = (10 \times \mathrm{W}) + (6{,}25 \times \mathrm{H}) - (5 \times \mathrm{A}) - 161$$
$$\text{Для чоловіків: } \mathrm{REE} = (10 \times \mathrm{W}) + (6{,}25 \times \mathrm{H}) - (5 \times \mathrm{A}) - 5, \quad (2.1)$$

де *W* – маса тіла (кг), *H* – зріст (см), *A* – вік (років).

Для точного визначення загальної добової витрати енергії організмом необхідно вказати рівень активності людини, розрахунки виконуються за формулою (2.2), яка наведена нижче.

$$\text{Для обох статей: } \mathrm{TDEE} = \mathrm{REE} \times \mathrm{PA}, \quad (2.2)$$

де *PA* – коефіцієнт фізичної активності: 1,2 – низька активність; 1,3 – середня; 1,4 – висока [10].

Далі значення TDEE коригується залежно від мети користувача: якщо ціллю є схуднення – необхідно створити дефіцит калорій, тобто споживати на 15% менше від загальної денної витрати; якщо потрібно наростити м’язи, слід споживати на 10% більше калорій, ніж показник TDEE; у разі потреби підтримання поточної форми розрахунок калорій залишається без змін.

Оптимально збалансованим вважається розподіл у кількості 30% білків, 30% жирів і 40% вуглеводів від денної норми калорій. Таке співвідношення базується на стандартних енергетичних еквівалентах, які враховують середню кількість енергії, отриману організмом при засвоєнні кожного з нутрієнтів [9].

Ще одним важливим показником, який має розраховувати вебзастосунок, є індекс маси тіла (ІМТ). Він обчислюється за формулою (2.3) та використовується для орієнтовної оцінки відповідності ваги людини її зросту, а також для визначення ризику розвитку захворювань, пов’язаних із надлишковою або недостатньою масою тіла.

$$\mathrm{IMT} = \frac{\mathrm{W}}{\mathrm{H}^2}, \quad (2.3)$$

де *W* – маса тіла (кг), *H* – зріст (м).

Результат обчислення інтерпретується відповідно до даних МОЗ України, згідно із якими: якщо показник ІМТ менше ніж 18,5 – маса тіла є недостатньою; за умови, що він знаходиться у межах від 18,5 до 24,9 – вага нормальна; про наявність надлишкової ваги свідчить отримане значення в діапазоні між 25,0 та 29,9; у разі виявлення ожиріння – сигналом буде одержання у підсумку числа більше за 30. Важливо зазначити, що ці показники вважаються еталонними лише для дорослих людей, аби оцінити ІМТ у дітей чи підлітків слід використовувати окремі таблиці та консультуватися зі спеціалістами [11].

Відсоток жиру в тілі обчислюють задля більш точної оцінки фізичної форми. Для розрахунку цього показника найпопулярнішими методами є US Navy [12] та YMCA. Ми будемо використовувати перший спосіб – формулу, розроблену ВМС США (2.4), бо вона є більш точною за рахунок використання більшої кількості параметрів при обчисленні, крім обхвату талії тут враховується зріст, і обхват шиї та стегон.

$$\begin{gathered}\text{Для жінок: } \%\mathrm{F} = 495 \,/\, (1{,}29579 - 0{,}35004 \times \log_{10}(\mathrm{W} + \mathrm{H} - \mathrm{N}) + \\ 0{,}22100 \times \log_{10}(\mathrm{Hg})) - 450 \\ \text{Для чоловіків: } \%\mathrm{F} = 495 \,/\, (1{,}0324 - 1{,}9077 \times \log_{10}(\mathrm{W} - \mathrm{N}) + \\ 0{,}15456 \times \log_{10}(\mathrm{Hg})) - 450,\end{gathered} \tag{2.4}$$

де *W* – обхват талії (см), *H* – обхват стегна (см), *N* – обхват шиї (см), *Hg* – зріст (см).

Отримані результати розглядаються як орієнтовна оцінка жирової маси в тілі, відповідно, нижчі значення свідчать про менший міст жиру, а вищі – про більший. Обчислені значення порівнюються із нормами відповідно статі та віку для отримання інформації про фізичну форму та можливі ризики для здоров’я, при цьому оптимальні показники для дорослих жінок становлять 18-28%, а для чоловіків – 10-20%.

Щоб реалізувати підбір страв у раціон відповідно до денної потреби в нутрієнтах необхідно розраховувати КБЖВ для кожного інгредієнта за формулою (2.5). Для цього використовують метод енергетичних еквівалентів макронутрієнтів 4-9-4: на 1 грам білків – 4 ккал, на 1 грам жирів – 9 ккал, на 1 грам вуглеводів – 4 ккал [9].

$$N = \frac{P \times TDEE}{E}, \tag{2.5}$$

де N – кількість нутрієнта (г), P – частка калорійності (%) цього нутрієнта, $TDEE$ – загальна добова витрата енергії (ккал), E – енергетична цінність 1 г нутрієнта (ккал).

Розроблена математична модель спирається на загальноприйняті наукові дослідження, тому стане хорошою основою для всіх розрахунків у вебзастосунку та дозволить забезпечити високу точність обчислень, що є вкрай важливим фактором для досягнення користувачами бажаних результатів без шкоди здоров’ю.

2.2 Визначення користувацьких ролей і рівнів доступу до системи

Визначення типів користувачів – важливий етап проєктування, оскільки від цього залежить інтерфейс, функціонал та обмеження для певної ролі у системі, а отже, це впливає на загальну якість і ефективність продукту.

Загалом розрізняють три види суб’єктів системи: адміністратори, авторизовані користувачі та гості (ті, хто не виконали вхід в акаунт) – усі вони мають певні можливості, що визначаються рівнями або це ще називають правами доступу. Відповідно, для виконання певної дії слід мати дозвіл. Щоб надати всім користувачам необхідні дозволи зручно спершу розподілити їх за ролями і потім для них визначити права.

У вебзастосунку для планування раціону та фізичної активності найбільш доцільно виділити чотири рівні доступу за ролями:

– гість – неавторизований користувач, який може лише переглядати рецепти, статті, готові плани харчування та має доступ до калькуляторів оцінки фізичних параметрів тіла і розрахунку нутрієнтів без збереження даних у системі;

– адміністратор – у базі даних створюється «нульовий» користувач із повним набором дозволів, визначених в усіх інших ролях, і, зокрема, з можливістю змінювати ролі решти користувачів, а також видаляти їхні акаунти чи створювати нові;

– авторизований користувач – призначається автоматично при реєстрації та подальшій авторизації, надає такий же набір дозволів, що і у гостя, але разом із тим дає можливість згенерувати або самостійно створити план харчування, відслідковувати прогрес в особистому кабінеті;

– редактор – надається адміністратором і включає дозволи на додавання нових, редагування та видалення вже існуючих рецептів, статей і готових планів харчування.

Чотири рівні доступу до системи – це оптимальний варіант для даного вебзастосунку. Подібне визначення ролей і розподіл дозволів між ними дозволяє логічно розділити доступ до функціоналу, адаптувати інтерфейс під різні типи користувачів і забезпечити безпеку даних. У майбутньому за потреби можна додати нові ролі та наділити їх необхідними правами.

2.3 Створення структурної схеми та діаграми прецедентів

Структурні схеми програмного забезпечення відображають ієрархічний розподіл основних компонентів системи та зв’язки між ними. Такі діаграми мають важливе значення для розуміння структури майбутньої програми, адже вони демонструють які частини слід втілювати як окремі функціональні блоки (модулі), тим самим спрошуючи сприйняття та аналіз побудови програмного продукту та дозволяючи уникнути надмірної складності в архітектурі системи і неузгодженості між її модулями.

На рисунку 2.1 наведено структурну схему створюваного вебзастосунку.

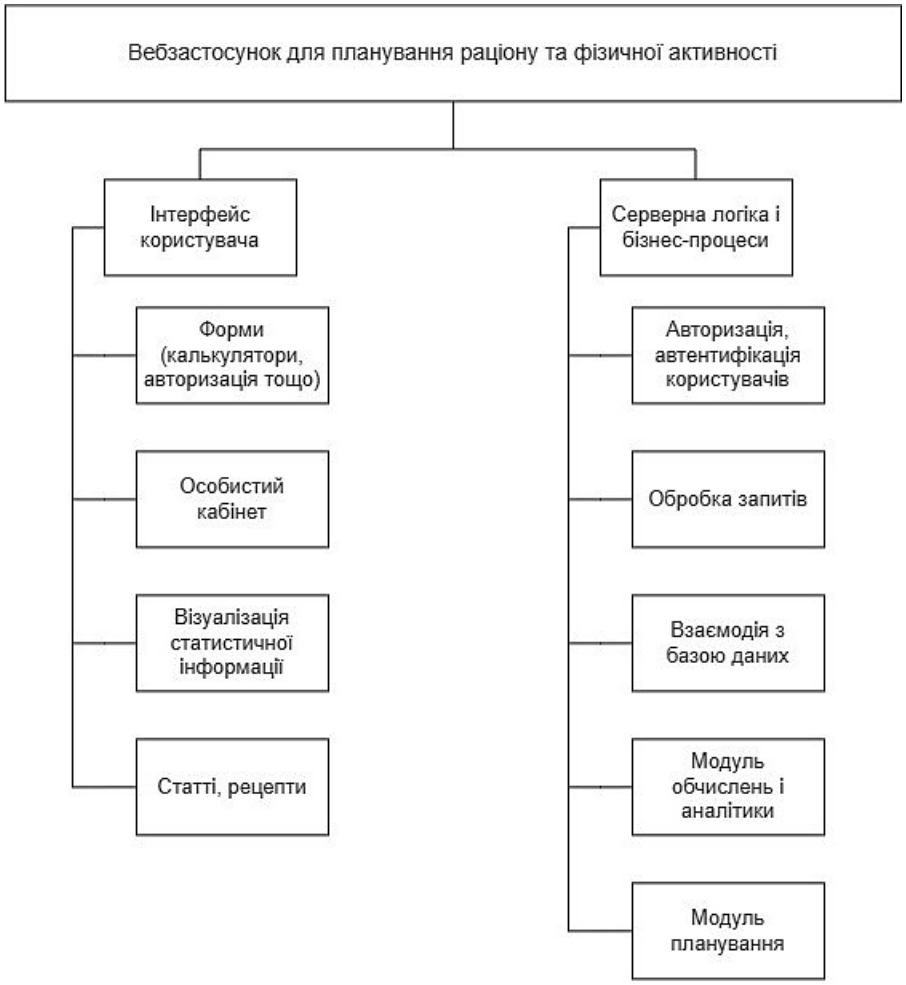

Рисунок 2.1 – Структурна схема вебзастосунку планування раціону та фізичної активності

Створена схема відображає загальну організацію програмного комплексу, а також включає основні модулі, які належать до клієнтської та серверної частини. Таким чином можна незалежно розробляти, підтримувати і масштабувати компоненти інтерфейсу та внутрішньої логіки.

Наступною було розроблено діаграму варіантів використання або як її ще називають діаграму прецедентів, що відображає типові сценарії взаємодії різних ролей користувачів із системою. Відповідно, дана схема складається із: акторів, які представляють користувачів, що взаємодіють із системою; прецедентів – дій або функцій, які виконуються у відповідь на запити акторів; зв’язків – ліній і стрілок, що поєднують акторів із прецедентами, відображаючи взаємодію між ними; також може відображатися система у

вигляді великого прямокутника, який обмежує всі прецеденти, проте це опціонально. Побудова цієї діаграми необхідна, аби подивитися на систему з боку тих, хто буде працювати із нею, та конкретно визначити до якого функціоналу будуть надані права доступу користувачам із певними ролями.

Нижче на рисунку 2.2 показана діаграма прецедентів вебзастосунку для планування раціону та фізичної активності, на якій визначено 4 актори, система та прецеденти, що виникають при взаємодії із нею.

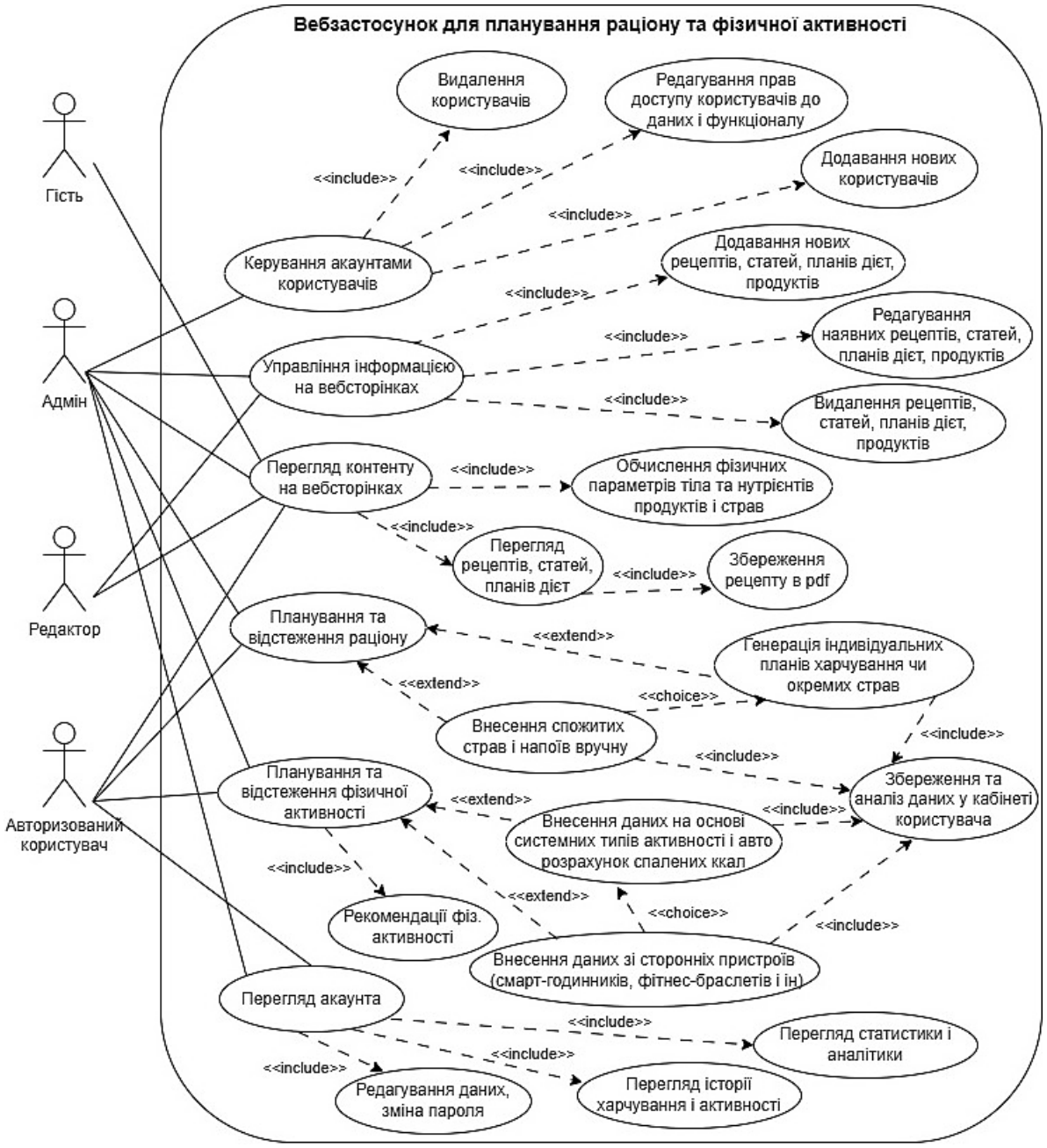

Рисунок 2.2 – Діаграма прецедентів створюваного вебзастосунку

Суцільні лінії з’єднують акторів та доступний їм функціонал, а стрілки з підписами <<include>>, <<extend>> та <<choice>> показують загальні підпроцеси у наведених функціях, додаткові кроки, які виконуються за певних умов, та варіанти подальших шляхів на вибір користувача відповідно.

2.4 Розробка функціональної діаграми

Функціональна схема IDEF0 дає змогу структуризувати процеси, краще зрозуміти логіку системи. З її допомогою можливо розбити складні функції на більш простіші, а також визначити як вони взаємодіють між собою.

На рисунку 2.3 представлене контекстне уявлення створюваного вебзастосунку. Це найвищий рівень абстракції, на якому відображено загальний огляд системи та її взаємодію із зовнішніми даними і користувачами.

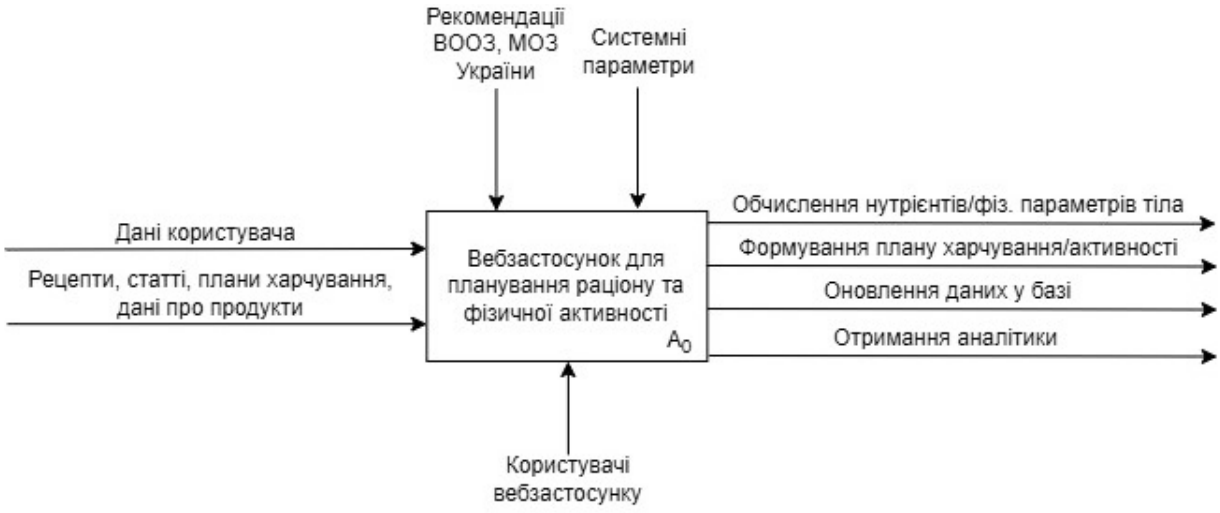

Рисунок 2.3 – Контекстне уявлення вебзастосунку планування раціону та фізичної активності

Стрілками ліворуч показано вхідні дані, праворуч – отриманий результат, зверху зображено, що керує, а внизу – виконавець.

Нижче на рисунку 2.4 наведена діаграма першого рівня декомпозиції, що розширює контекстне уявлення A0, та більш детально показує окремі функції, які виконує програмний продукт.

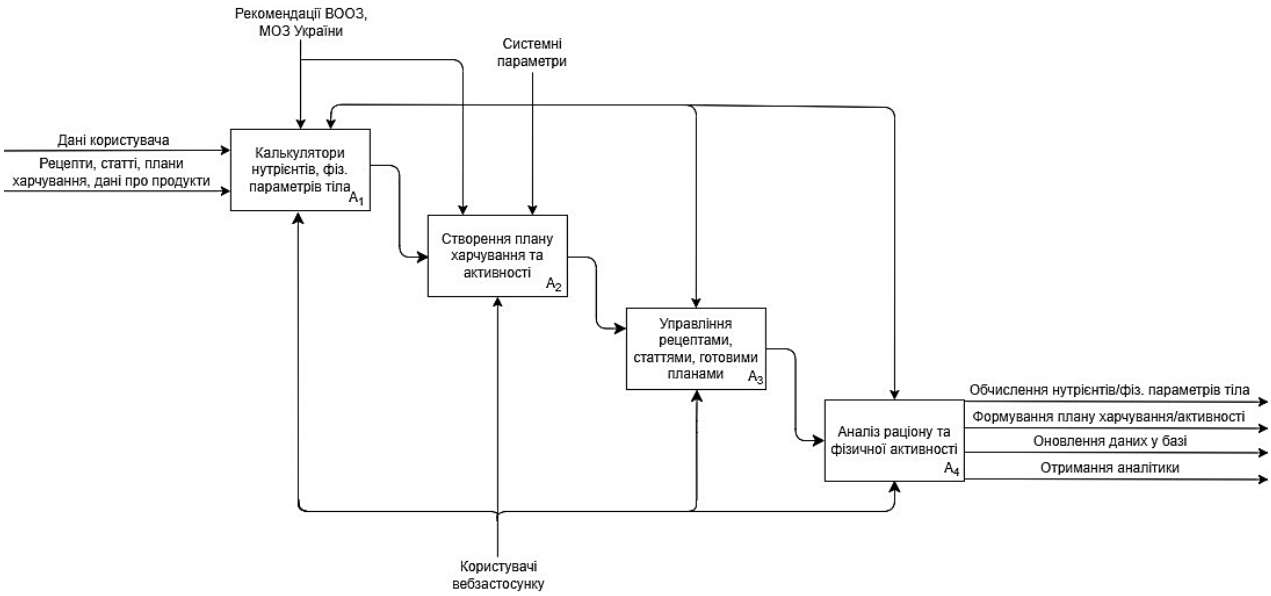

Рисунок 2.4 – Діаграма першого рівня декомпозиції

На ній зображено ті ж дуги, що і на контекстній, але загальна функція розбита на менші.

Діаграма другого рівня декомпозиції, показана на рисунку 2.5, розширює діаграму першого рівня, розбиваючи блок А2 на більш дрібні функції. Таким чином отримуємо його складові частини більш детально.

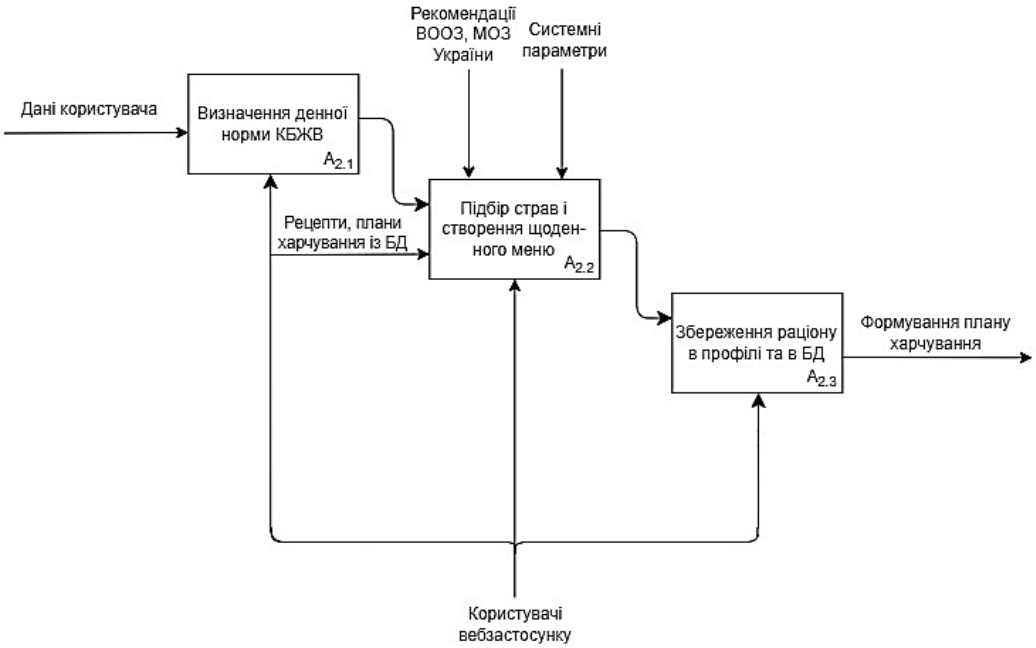

Рисунок 2.5 – Діаграма другого рівня декомпозиції

Рівнів декомпозиції може бути більше, адже кожен із них дозволяє докладніше розібратися у функціоналі системи та яким чином між собою взаємодіють її складові, але щоб загалом зрозуміти логіку роботи вебзастосунку для планування раціону та фізичної активності виявилося достатньо і двох ступенів деталізації. Отримане розуміння того, з яких частин складається система, значно спростить і пришвидшить процес розробки.

2.5 Створення діаграми послідовностей

Діаграма послідовностей необхідна, аби наочно відобразити процес обміну повідомленнями між різними частинами системи у ході виконання конкретного сценарію. Такі діаграми описують які дії і в якому порядку відбуваються між об’єктами в межах визначеного сценарію.

Зазвичай діаграми послідовностей складаються із: учасників взаємодії, які беруть участь у процесі, що ілюструється, та розташовані вгорі; часових ліній – пунктирних осей, що відображаються вниз від учасників і тривають допоки вони існують в конкретному сценарії, та потрібні для демонстрації хронології виконання дій. Безпосередньо сама взаємодія показується за допомогою суцільних і пунктирних стрілок, які зображають запити, відповіді та інші дії. Інколи на діаграмах послідовностей за допомогою прямокутників на часових осях позначають коли саме блок активний – це допомагає швидше визначати та простіше простежувати які частини співпрацюють у певний момент.

На рисунку 2.6 показана діаграма послідовностей процесу використання калькулятора КБЖВ страв у вебзастосунку для планування раціону та фізичної активності. Завдяки наявності часових ліній на цій діаграмі можна не лише отримати уявлення про дії, які виконуються в ході визначеного процесу, а й дізнатися їхню хронологію, зрозуміти в який момент відбувається взаємодія, наприклад, користувацького інтерфейсу із сервером чи коли сервер надсилає запити до бази даних.

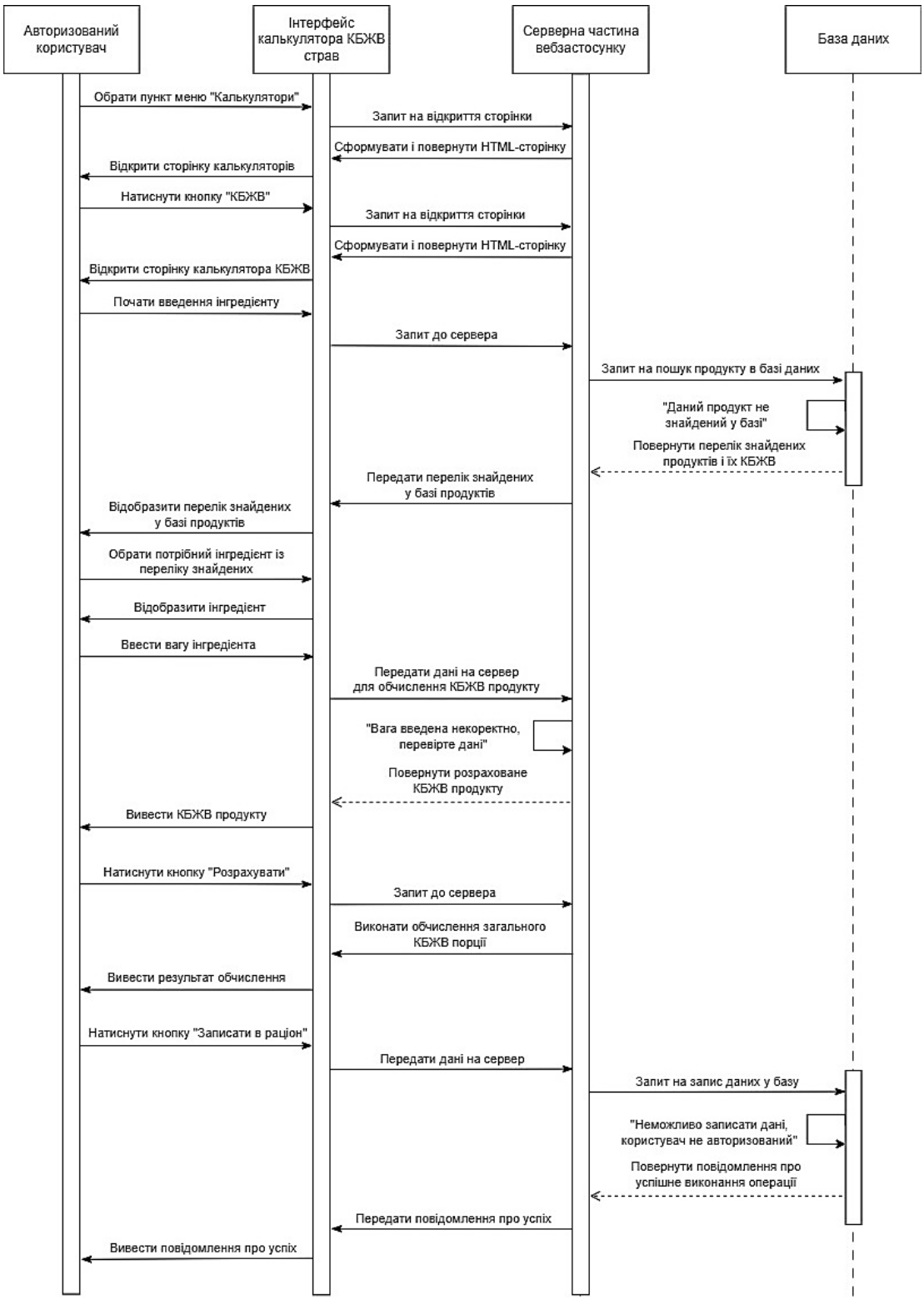

Рисунок 2.6 – Діаграма послідовностей обчислення КБЖВ спожитої страви у створюваному вебзастосунку

Таким чином, маючи уявлення про те, коли і як конкретні об’єкти взаємодіють між собою, можна уникнути проблем із логікою побудови взаємозв’язків у системі ще до написання коду.

2.6 Побудова моделі «сутність-зв’язок» бази даних вебзастосунку

ER-діаграми дозволяють змоделювати сутності у базі даних та взаємозв’язки між ними. Власне, зв’язками тут виступають логічні відношення «один-до-одного», «один-до-багатьох» та ін. На рисунку 2.7 наведено діаграму сутність-зв’язок бази даних вебзастосунку планування раціону та активності.

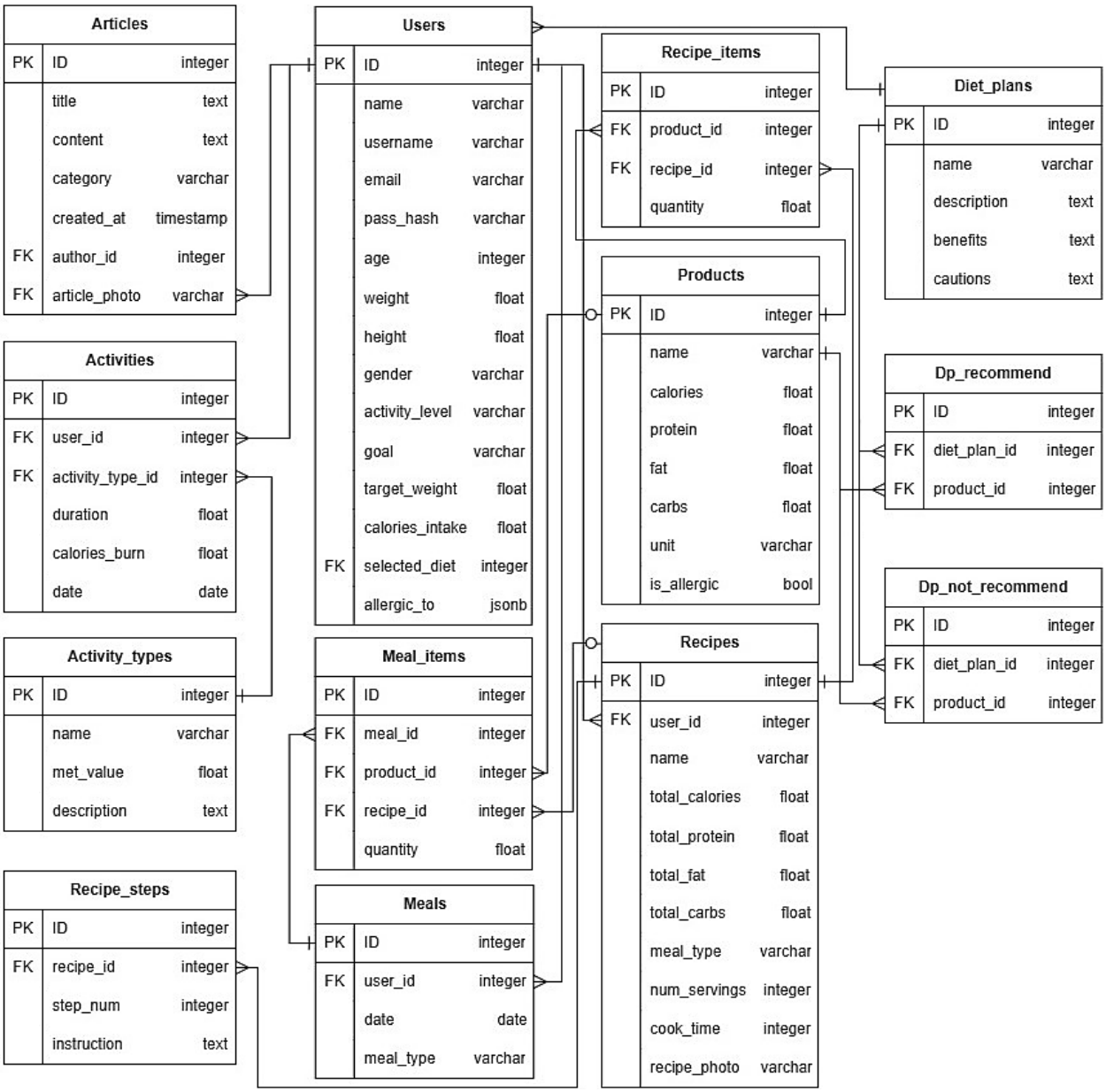

Рисунок 2.7 – ER-діаграма бази даних створюваного вебзастосунку

Для розроблюваного програмного продукту визначено 13 сутностей, що втілюють основні об’єкти, з якими буде працювати вебзастосунок. Кожна сутність має свій набір атрибутів, які описують її властивості, унікальний

ідентифікатор (первинний ключ) – для однозначної ідентифікації записів, а також зовнішній ключ для встановлення зв’язків з іншими таблицями бази даних і забезпечення її цілісності.

Між сутностями визначені наступні логічні залежності:

– Users – Activities («один-до-багатьох»): користувач може мати багато записів про свою активність;

– Users – Diet_plans («один-до-багатьох»): один користувач може мати одну обрану дієту, але дієту можуть обрати багато користувачів. Кожен план дієти має рекомендовані та не рекомендовані продукти;

– Activities – Activity_types («багато-до-одного»): кожен запис про активність посилається на її тип у довіднику;

– Users – Recipes («один-до-багатьох»): один користувач може створити багато рецептів, але кожен рецепт має лише одного автора;

– Recipes – Recipe_items («один-до-багатьох»): один рецепт може складатися із багатьох інгредієнтів;

– Recipe_items – Products («багато-до-одного»): кожен інгредієнт у рецепті визначений у довіднику продуктів;

– Recipes – Recipe_steps («один-до-багатьох»): один рецепт може мати багато етапів приготування;

– Users – Meals («один-до-багатьох»): користувач може мати багато прийомів їжі;

– Meals – Meal_items («один-до-багатьох»): кожен прийом включає декілька складових, наприклад, страва і напій;

– Meal_items – Products («багато-до-одного»): складові прийому їжі можуть включати споживання лише окремих продуктів, які визначені у довіднику продуктів;

– Meal_items – Recipes («багато-до-одного»): складовими прийому їжі можуть бути цілі страви, а не просто певні продукти;

– Article – Users («багато-до-одного»): кожна стаття має одного автора серед користувачів, а один користувач може написати багато статей.

За допомогою діаграми сутність-зв’язок вдалося збудувати реальну модель предметної області вебзастосунку. Усі зв’язки між сутностями організовані так, аби уникнути дублювання або втрати даних і мати змогу будувати коректні та ефективні запити до бази даних.

2.7 Створення вайрфреймів інтерфейсу основних сторінок

Перед початком реалізації користувацького інтерфейсу продукту необхідно розробити вайрфрейми – швидкі нариси, результатом яких є створення «скелету» майбутніх сторінок, що дасть змогу приблизно розуміти їхню структуру та розміщення таких ключових елементів, як: навігаційне меню, кнопки, поля форм, великі зображення, текстові блоки та ін., причому тут не важлива колірна гама чи шрифтова композиція, а тільки розташування.

На рисунку 2.8 відображено вайрфрейм початкової сторінки продукту.

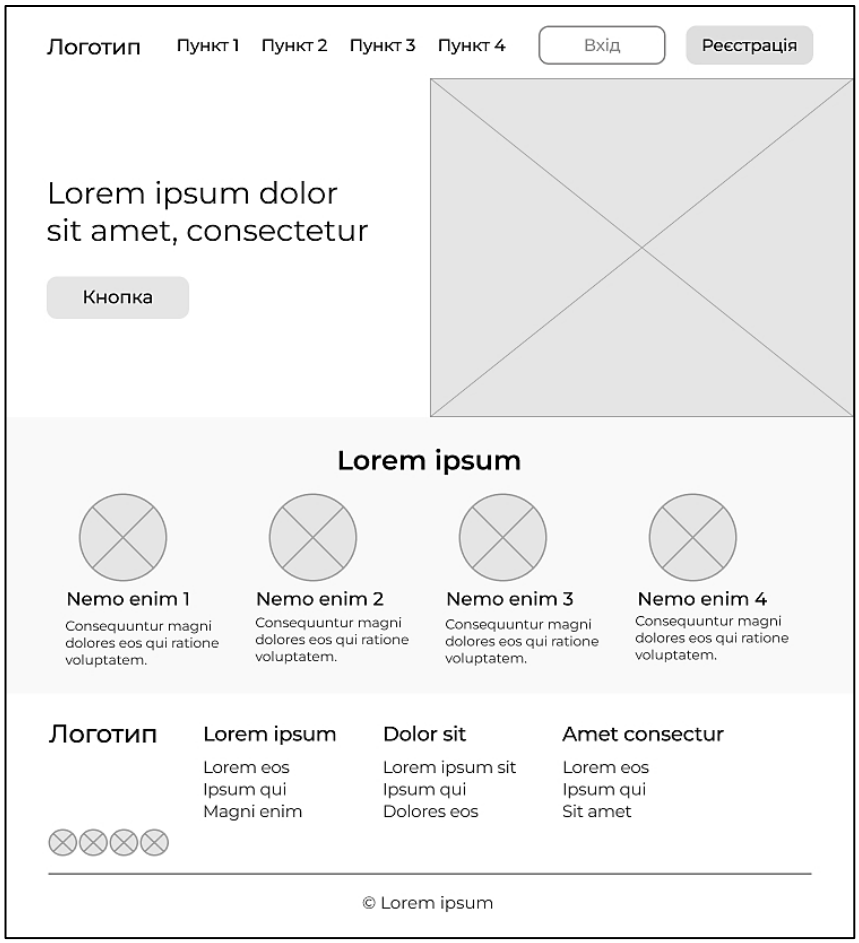

Рисунок 2.8 – Вайрфрейм головної сторінки вебзастосунку

Головна сторінка вебзастосунку, перш за все, має бути інформативною, аби користувачі, які вперше знайомляться із сервісом, змогли швидко дізнатися про його функціонал, тому обов’язково всі важливі факти у доступній формі викладаються саме тут. Вгорі розміщений логотип, навігаційне меню та кнопки для реєстрації або входу в існуючий акаунт. Нижче під основним контентом із текстом і зображеннями знаходиться футер.

Коли користувач натискає на кнопку Увійти, перед ним відкривається сторінка зі стандартною формою авторизації (див. рис. 2.9), де він повинен ввести свої ідентифікаційні дані – ім’я користувача та пароль. Якщо ж акаунт ще не створений, слід перейти на сторінку реєстрації за посиланням унизу.

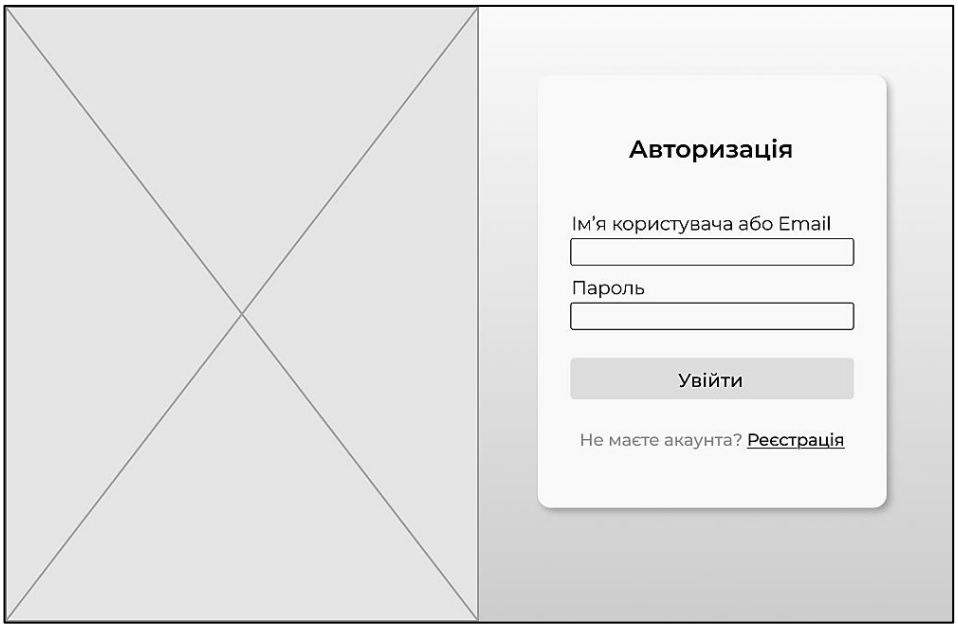

Рисунок 2.9 – Вайрфрейм сторінки із формою авторизації у системі

На більш великих дисплеях форма буде розташована праворуч від тематичного зображення, а на смартфонах – займатиме місце посередині екрану для зручності користувачів.

У процесі авторизації перевіряється роль користувача у системі та відповідно до неї надається доступ до певного функціоналу. На рисунку 2.10 показано панель адміністратора для керування акаунтами користувачів.

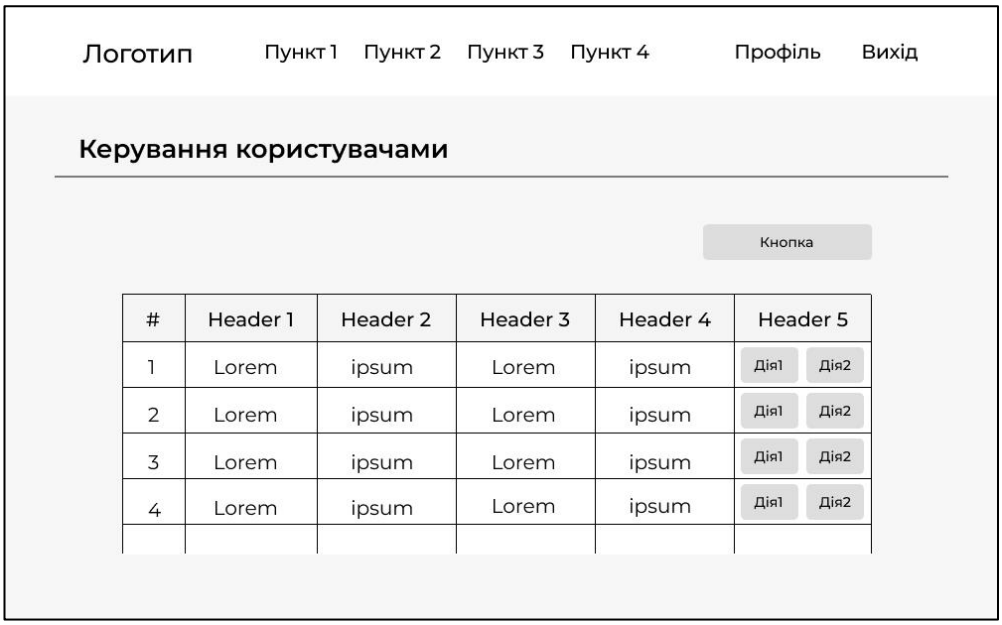

Рисунок 2.10 – Вайрфрейм адміністраторської панелі

Кнопка вгорі відкриває вікно додавання нового користувача у систему, а кнопки у таблиці здійснюють редагування ролей і видалення наявних акаунтів.

Аналітика харчування та активності (див. рис. 2.11) – одна з основних функцій вебзастосунку, вона доступна лише авторизованим користувачам.

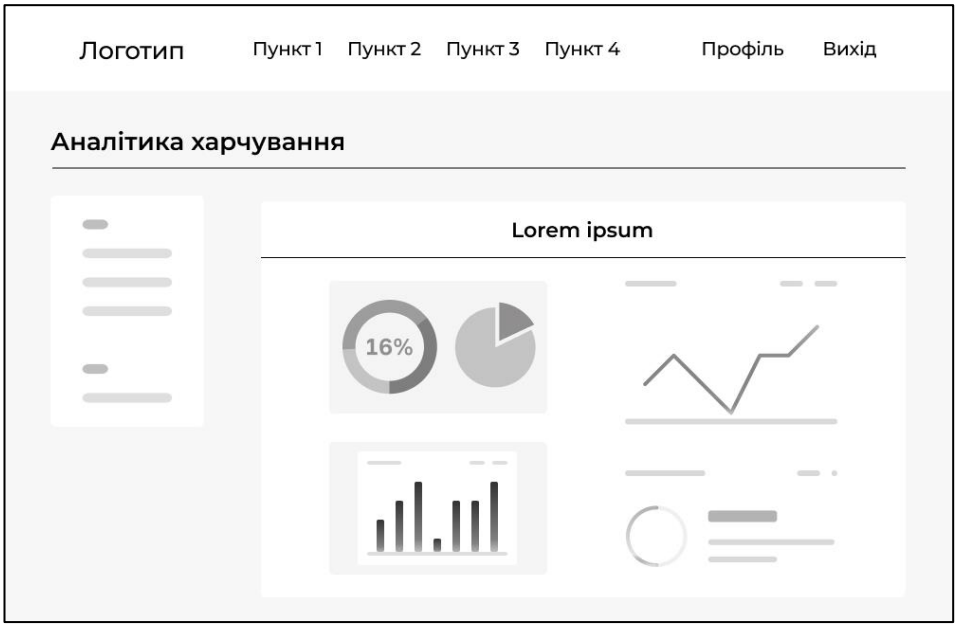

Рисунок 2.11 – Вайрфрейм сторінки з аналітикою

Статистична інформація подається у вигляді діаграм та схем, що значно спрощує її сприйняття.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ ПРОЄКТУ

3.1 Вибір технологій реалізації та середовища розробки

Для доступу до бази даних, розробки серверної частини і бізнес-логіки вебзастосунку для планування раціону та фізичної активності було обрано мову Python, а саме використано фреймворк Django [13], який широко застосовується для реалізації веборієнтованих додатків. Він є надзвичайно зручним у тому, що поставляється із вбудованим сервером, який не потребує тривалого налаштування вручну й автоматично перезавантажується при змінах коду, тож дозволяє повністю зосередитися на розробці функціональних можливостей і графічного інтерфейсу користувача майбутнього продукту.

У якості системи управління базою даних (СУБД) використовується об’єктно-реляційна PostgreSQL [14]. Рішення обрати саме її було прийняте із декількох причин. По-перше, вона підтримує складні фільтрації та агрегації; дозволяє напряму зберігати масиви у полі, не створюючи окрему таблицю; у ній можна зберігати та індексувати структуровані JSON-дані; завдяки повнотекстовому пошуку можна швидко знайти щось за ключовими словами, наприклад, рецепти за інгредієнтами. По-друге, ця СУБД підтримується Django, тож можна використовувати можливості обох технологій на максимум, конфліктів не виникне і всі функції працюватимуть коректно. По-третє, PostgreSQL легко масштабується, добре працює із великою кількістю даних і, що важливо, підходить для продакшену, адже підтримує одночасно багато клієнтів та стійка до збоїв – усе це надзвичайно важливо для реалізації стабільного вебзастосунку для планування раціону та фізичної активності.

Щоб запускати разом Django та PostgreSQL без тривалого налаштування і встановлення вручну використовується Docker [15] – інструмент, який «запаковує» все в ізольовані контейнери. Таким чином, кожна частина працює у своєму контейнері, при цьому весь стек запускається однією командою, конфігурується у Dockerfile та зв’язується між собою у docker-compose.yml. Завдяки цьому не потрібно турбуватися через несумісності, бо все працює в

однакових умовах на будь-якому сервері або комп’ютері. Також Docker створює окрему папку для даних, дозволяючи легко зберігати і переносити БД. Для великих проєктів можна запустити декілька контейнерів із однаковим бекендом і підключити їх до одного контейнера із базою даних, так вдасться розподілити частину запитів, щоб сайт не гальмував, при цьому дані не загубляться та не будуть дублюватися, адже база даних одна.

Для реалізації візуального інтерфейсу вебзастосунку було обрано перевірені загальноприйняті технології: мову гіпертекстової розмітки HTML [16], CSS [17] для стилізації сторінок та JavaScript [18] для динамічного оновлення вмісту без перезавантаження, відкриття модальних вікон тощо. Аби в результаті отримати не лише візуально привабливий, а ще й адаптивний UI, було прийняте рішення щодо необхідності використання фреймворку Bootstrap [19], щоб взаємодія користувачів із вебзастосунком була максимально зручною з будь-якого пристрою чи браузера. Вибір такого набору технологій обумовлений не лише їхньою зрозумілістю і зручністю, а й безшовною інтеграцією із Django. Усі html-файли у Django – це шаблони, у які завдяки шаблонізатору підставляються дані із Python через змінні, цикли та умовні блоки. Таким чином можна формувати вебсторінки із динамічним вмістом.

У якості середовища розробки спершу розглядалося декілька варіантів: Visual Studio Code [20], PyCharm та Sublime Text. Основними критеріями для подальшого вибору стали підтримка усіх використовуваних технологій та наявність різноманітних зручностей, які полегшують написання коду, – автодоповнення, підсвічування синтаксису тощо. У результаті було обрано VS Code, адже крім зазначених параметрів, він повністю безкоштовний, на відміну від двох інших інструментів, і має більшу підтримку технологій розробки фронтенду, ніж PyCharm. Також він більше підходить для реалізації такого великого проєкту за рахунок широкого спектру додаткових функцій «із коробки», що не можна відзначити у Sublime Text.

Для зручної взаємодії із Docker через графічний інтерфейс замість командного рядка, використано додаток Docker Desktop. Аналогічно для перегляду структури бази даних, слідкування за її роботою та виконання SQL-запитів використано вебінтерфейс pgAdmin.

Комбінація із вищерозглянутих технологій і середовищ для взаємодії з ними забезпечує все необхідне для реалізації вебзастосунку для планування раціону та фізичної активності, бо поєднує стабільний бекенд, зручний фронтенд і ефективну організацію процесу розробки.

3.2 Створення, контейнеризація та налаштування проєкту на Django

Незважаючи на вибір, здавалося б, одних із найбільш зручних технологій, процес створення та налаштування проєкту може стати першим ускладненням на шляху до реалізації продукту, адже на цьому етапі потрібна не лише максимальна уважність і сконцентрованість, а й ґрунтовне розуміння того, як працює контейнеризація, що потрібно писати у файлах налаштувань, через який порт здійснюється підключення до бази даних тощо. Також потрібно організувати роботу одразу декількох компонентів і Docker для них, що теж здається не таким легким завданням.

Перед початком роботи Docker Desktop було завантажено з офіційного сайту та інстальовано, а встановлення Django відбувалося через командний рядок за допомогою менеджера пакетів Python.

Наступним кроком було створення Django-проєкту. Для цього у терміналі в потрібній папці було виконано команду *django-admin startproget nutrizen*. У результаті цього була створена структура майбутнього вебзастосунку – головна директорія, файли для запуску та файл із налаштуваннями.

Після цього командою *python manage.py startapp nutrizenApp* було додано основний додаток проєкту, тобто його окремий функціональний модуль. Таким чином була розроблена папка з файлами, у яких далі будуть розроблятися моделі, представлення, форми та адреси створюваного продукту.

Щоб інтегрувати Docker та виконати налаштування контейнера з Django було створено Dockerfile – своєрідну інструкцію, що описує яку версію та бібліотеки Python потрібно встановити, як запускати сервер тощо. Особливостями цього етапу є те, що потрібно вказати правильну версію Python, не забути додати встановлення потрібних бібліотек у requirements.txt, вказати що потрібно копіювати у контейнер, а що ні.

Далі у файлі-конфігурації було налаштовано одночасний запуск контейнерів із Django та PostgreSQL, підключено pgAdmin для зручності. Тут було дуже важливо правильно прописати порти для доступу до сервісів – 8000:8000 для Django та 5432:5432 для PostgreSQL, адже якщо змінити стандартний порт для бази даних у контейнері, бекенд не зможе підключитися. Пароль для бази даних, як і інші змінні оточення, були прописані в окремому файлі .env, щоб уникнути втрати чутливих даних при їх передачі на сервер.

Наступним етапом було встановлення підключення до бази даних у файлі налаштувань Django-проєкту, а також задання інших важливих параметрів, зокрема, активація режиму відладки, налаштування часової зони.

Потім був запуск усіх сервісів у своїх контейнерах командою *docker compose up --build*. Таким чином все працює ізольовано, але взаємодіє між собою. Паралельно із цим було виконано міграції, щоб Django додав у базу даних стандартні таблиці для авторизації, роботи із сесіями тощо. Без цих таблиць неможливе повноцінне використання усіх можливостей фреймворку. Завершальним кроком перед безпосереднім початком розробки було створення суперкористувача із правом доступу до адмін-панелі Django.

Хоч на початку налаштування здається складним, насправді, після вивчення особливостей використовуваних технологій, воно проходить доволі швидко, без Docker це було б ще простіше, але в подальшому із ним буде легше розгортати проєкт у хмарі та масштабувати. Загалом, він спрощує навіть локальну розробку, адже можна змінювати версії бази даних без ризику щось безповоротно «зламати», через залишки попередніх версій не виникнуть конфлікти тощо.

3.3 Реалізація візуального інтерфейсу

Якісний візуальний інтерфейс є важливою складовою будь-якого програмного продукту, його «візитівкою». Оскільки це перший елемент системи, із яким взаємодіє користувач, важливо розробити стильний дизайн, щоб застосунок впадав в очі та вирізнявся серед аналогів та, що найголовніше, забезпечував якісний користувацький досвід і позитивне сприйняття продукту.

Зручна навігація, динамічний адаптивний інтерфейс та відсутність нагромадження елементів на сторінках – безпосередньо впливають на формування хорошого користувацького досвіду. У контексті розробки із застосуванням Django актуальність і динамічність контенту на вебсторінках забезпечується завдяки використанню шаблонізатора.

Основою для генерації сторінок вебзастосунку є файл base.html, Фрагмент його вмісту наведений на рисунку 3.1 нижче.

```
<body class="">
    {% if topbar %} {% block navbar %} {% include "navbar.html" %} {% endblock navbar %} {% endif %}
    {% block pageContent %}{% endblock pageContent %}
    {% block ScriptBlock %} {% endblock ScriptBlock %}
    <div class="modal fade" id="confirm_modal" role='dialog'>
        <div class="modal-dialog modal-md modal-dialog-centered" role="document">···
        </div>
    </div>
    {% if footer %}
    <footer id="footer" class="mt-5">
        <div class="container">
            <div class="row d-flex flex-wrap justify-content-between py-5">
                <div class="col-md-3 col-sm-6">
                    <div class="footer-menu footer-menu-001">···
                    </div>
                </div>
                <div class="col-md-3 col-sm-6">
                    <div class="footer-menu footer-menu-002">
                        <h5 class="widget-title text-uppercase mb-4">Швидкий доступ</h5>
                        <ul class="menu-list list-unstyled text-uppercase border-animation-left fs-6">
                        <li class="menu-item">
                            <a href="{% url 'home' %}" class="item-anchor">Головна</a></li>
                        <li class="menu-item">
                            <a href="{% url 'calculators-page' %}" class="item-anchor">Калькулятори</a></li>
                        <li class="menu-item">
                            <a href="{% url 'diet-plans-page' %}" class="item-anchor">Готові плани</a></li>
                        <li class="menu-item">
                            <a href="{% url 'blog-page' %}" class="item-anchor">Статті</a></li>
                        <li class="menu-item">
                            <a href="{% url 'recipes-page' %}" class="item-anchor">Рецепти</a></li>
                        </ul>
                    </div>
                </div>
            </div>
```

Рисунок 3.1 – Структура основного шаблону проєкту

Саме у хедері цього файлу описуються метадані, відбувається підключення стилів, скриптів, створюється футер, а всередині тегу <body> визначається місце для виведення навігаційної панелі {% block navbar %}, яка реалізована в іншому файлі, та безпосередньо контенту решти окремих сторінок через {% block pageContent %}, що визначається в усіх інших створених шаблонах.

Логіка відображення інтерфейсу для гостей та авторизованих користувачів відбувається через отримання відповідної інформації з бази даних за допомогою шаблонізатора, фрагмент верстки відповідного файлу показано нижче на рисунку 3.2.

```
<div class="col-3 col-lg-auto">
    <ul class="list-unstyled d-flex m-0">
      {% if user.is_authenticated and not can_edit %} <!-- Користувач -->
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'profile-page' %}">Профіль</a>
        </li>
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'logout' %}">Вийти</a>
        </li>
      {% elif user.is_authenticated and can_edit %} <!-- Адмін або редактор -->
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'profile-page' %}">Управління</a>
        </li>
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'logout' %}">Вийти</a>
        </li>
      {% else %} <!-- Гість -->
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'login-page' %}">Вхід</a>
        </li>
        <li class="d-none d-lg-block">
          <a class="text-uppercase mx-3" href="{% url 'register-page' %}">Реєстрація</a>
        </li>
      {% endif %}
        <li class="search-box mx-2">
            <a href="#search" class="search-button">···
            </a>
        </li>
    </ul>
</div>
```

Рисунок 3.2 – Фрагмент HTML-розмітки для виведення навігаційного меню

Тут умовні блоки перевіряють чи користувач авторизований, якщо так і він не має прав на редагування системи, то надають йому доступ до профіля, в іншому випадку відображається елемент меню для доступу до панелі управління системою. Якщо ж користувач не виконав вхід у систему, йому доступні сторінки із формами реєстрації та авторизації. Подібна логіка реалізації дає змогу динамічно контролювати відображення і приховування елементів інтерфейсу залежно від ролі користувача, уникаючи створення окремих файлів для цього.

Усі сторінки вебзастосунку розроблені в єдиному стилі, на рисунку 3.3 відображено фрагмент файлу таблиць стилів, де це налаштовано.

```
body {
  --heading-font: "Merriweather", serif;
  --heading-font-weight: 400;
  --heading-color: #111;
  --heading-line-height: 1.24;

  --swiper-theme-color: #8C907E;

  --bs-body-font-family: "Inter", Roboto, sans-serif;
  --bs-body-font-size: 18px;
  --bs-body-font-weight: 400;
  --bs-body-line-height: 1.5;
  --bs-body-color: #8f8f8f;
  --bs-body-color-rgb: 143, 143, 143;

  --bs-primary: #8C907E;
  --bs-secondary: #6c757d;
  --bs-black: #111;
  --bs-light: #F1F1F0;
  --bs-dark: #212529;
  --bs-gray: #9aa1a7;
  --bs-gray-dark: #51565b;

  --bs-primary-rgb: 140, 144, 126;
  --bs-secondary-rgb: 108, 117, 125;
  --bs-black-rgb: 17, 17, 17;
  --bs-light-rgb: 241, 241, 240;
  --bs-dark-rgb: 33, 37, 41;

  --bs-link-color: #111;
  --bs-link-color-rgb: 17, 17, 17;
  --bs-link-decoration: underline;
  --bs-link-hover-color: #111;
  --bs-link-hover-color-rgb: 17, 17, 17;
}
```

Рисунок 3.4 – Загальні параметри стилю інтерфейсу

Колірна гама обрана не випадково, адже словосполучення «здорове харчування» викликає у людей асоціацію саме із зеленим кольором, а сірий і чорний та їхні відтінки тут використовуються для заголовків і текстів та розставлення акцентів на кнопках чи інших елементах дизайну. Створено універсальну шрифтову композицію, що добре працює як для заголовків, так і для основного тексту, а також підтримує кирилицю та латиницю і гармонійно поєднується з інтерфейсом. Усі ці аспекти підкреслюють послідовність у дизайні, забезпечують зручність читання текстів і роблять застосунок приємним для користувачів.

3.3.1 Створення інтерфейсу загальнодоступних сторінок

Гостям і абсолютно усім типам авторизованих користувачів доступі сторінки: Головна, Готові плани, Калькулятори, Рецепти, Блог, а також Про нас, Часті питання, Політика конфіденційності та Зворотний зв’язок, файли їх верстки розміщено у додатку А. Якщо користувач не авторизувався, він може виконати розрахунки у калькуляторах чи заповнити форму для отримання фідбеку, а всі інші сторінки будуть лише для перегляду контенту.

На рисунку 3.4 продемонстровано основні фрагменти головної сторінки вебзастосунку. За винятком зміни деяких пунктів, вона є однаковою для всіх користувачів системи. Її мета – представлення продукту, щоб зацікавити авдиторію і переконати чому варто використовувати саме його.

Сторінка поділена на декілька секцій, які включають: основну обкладинку, блок із коротким описом ключового функціоналу, який вирізняє продукт з-поміж аналогів, область із відгуками користувачів і фотогалерею деяких рецептів, які наявні у базі застосунку.

Подібна структура дозволяє швидко ознайомити гостей із усіма можливостями, які надає продукт, при цьому не потрібно переходити на інші сторінки і довго шукати те, що цікавить.

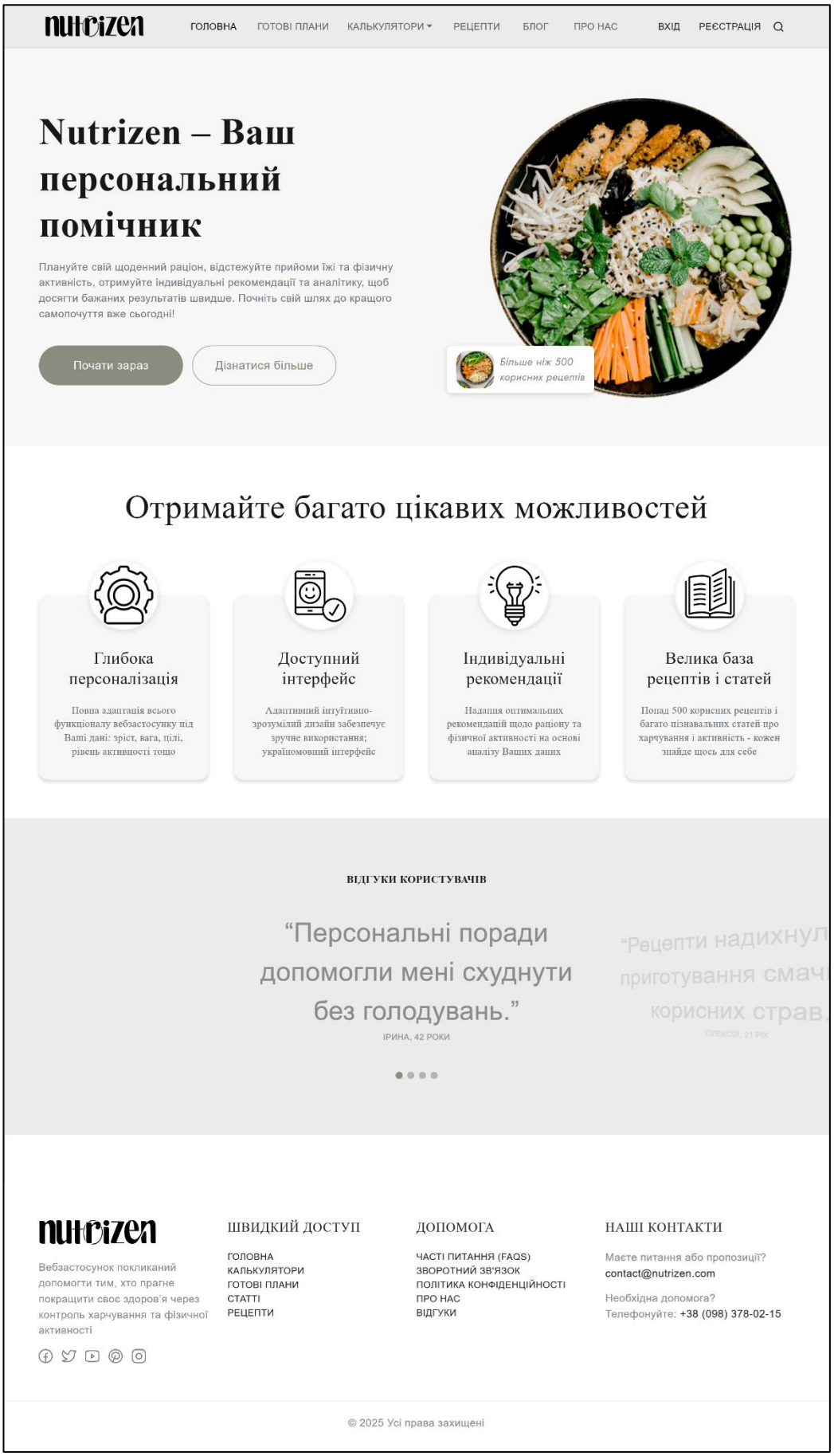

Рисунок 3.4 – Головна сторінка вебзастосунку «Nutrizen»

На рисунку 3.5 показано сторінку із калькулятором індексу маси тіла. Футер обрізано, адже він всюди однаковий.

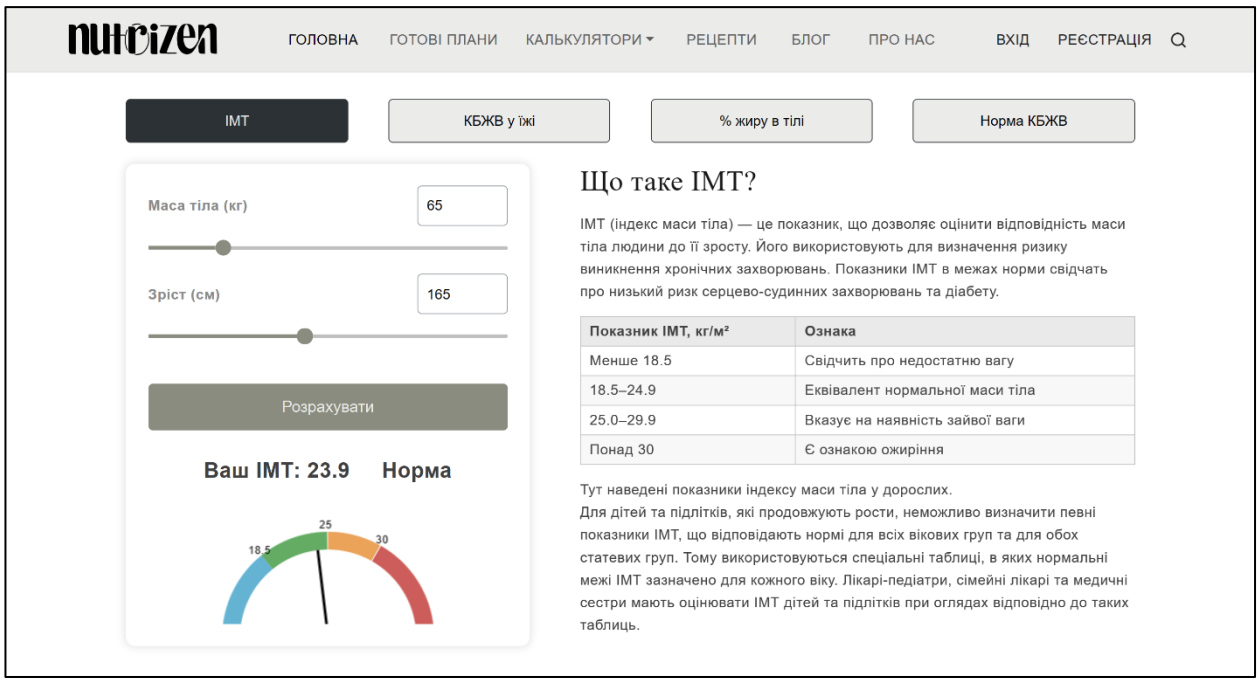

Рисунок 3.5 – Інтерфейс сторінки калькулятора ІМТ

Аналогічно на усіх інших вкладках форма калькулятора знаходиться ліворуч, а з правого боку розташована інформація про показник, що визначається. У мобільній версії калькулятор розташований вгорі під вкладками для вибору показника, а інформація подається нижче. Повна HTML-структура шаблону цієї сторінки міститься у додатку А.

Для того, щоб користувачу було простіше сприймати отримані результати, вони візуалізуються за допомогою діаграм, які реалізовані з використанням бібліотеки Charts.js. У додатку А знаходиться повний лістинг скриптів у файлі scripts.js.

Сторінки із рецептами та статтями візуально однакові, проте набір фільтрів у блоці ліворуч дещо відрізняється. На рисунку 3.6 показано сторінку рецептів на великому екрані, а на рисунку 3.7 – рецепти і статті у блозі в мобільному відображенні.

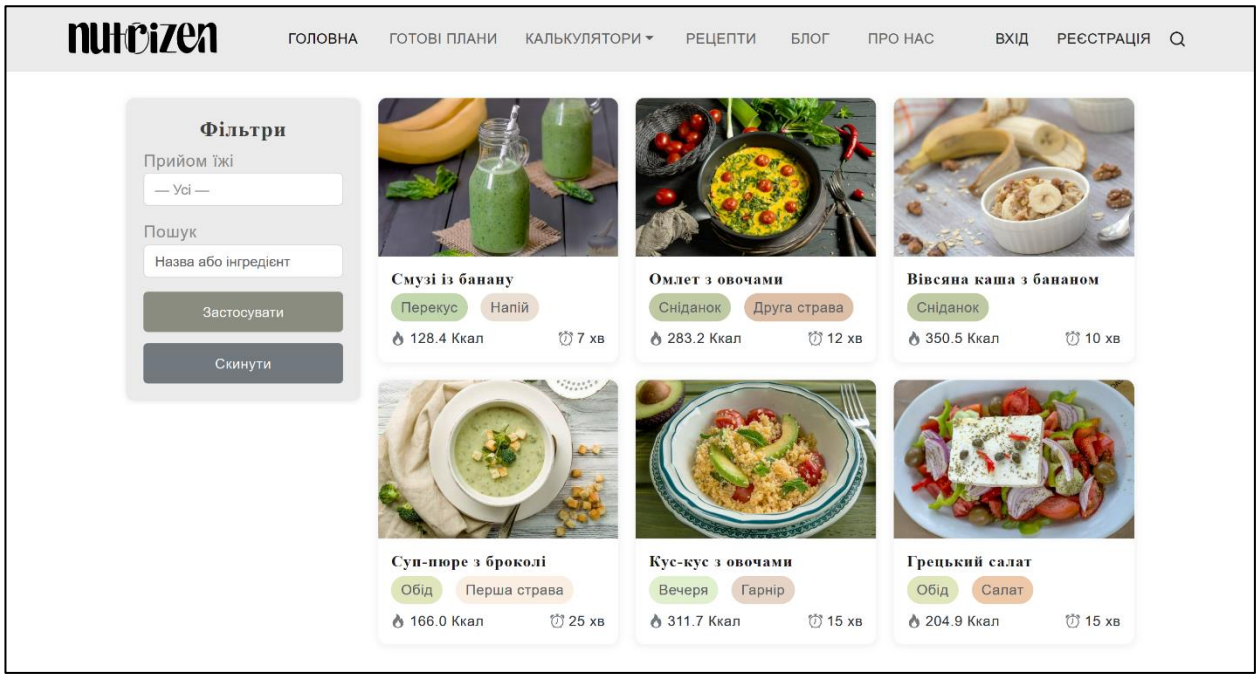

Рисунок 3.6 – Інтерфейс сторінки із рецептами у «Nutrizen»

Всі рецепти відображені у вигляді карток із фотографією страви, її назвою, міткою категорії та підходящого прийому їжу, унизу картки зазначена загальна калорійність і час приготування.

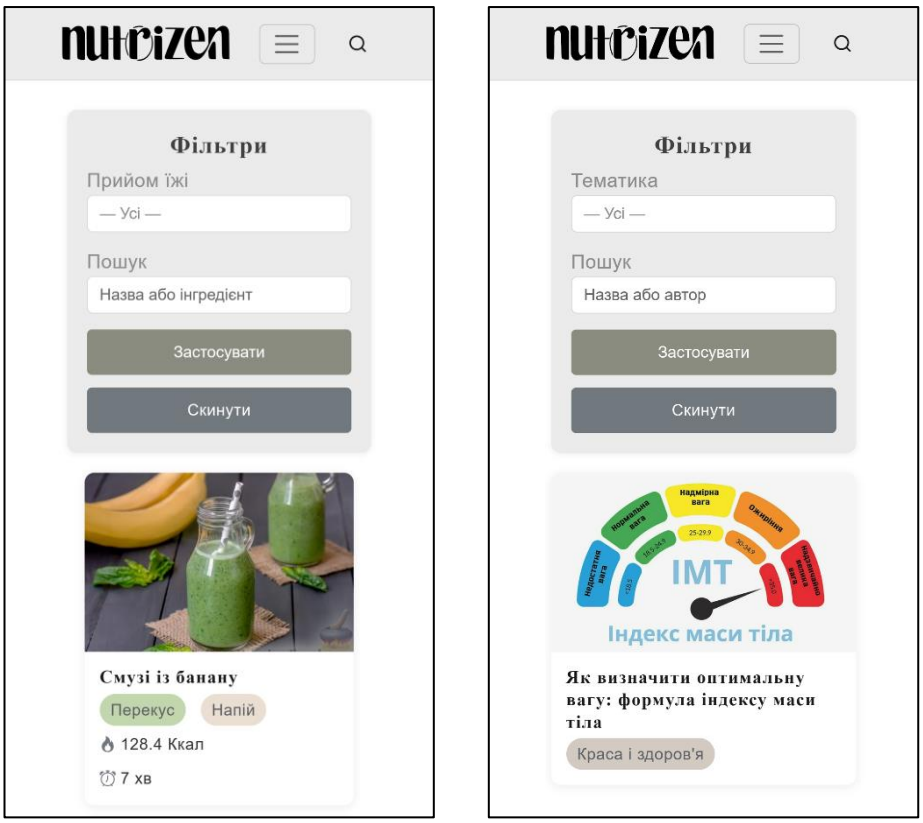

Рисунок 3.7 – Мобільна версія інтерфейсу сторінки із рецептами та блогу

При зменшенні ширини екрану блок із фільтрами розміщається зверху, а картки із контентом відповідної сторінки розташовуються внизу. При цьому вміст футера також відображається у стовбець один під одним. Навігаційне меню приховується, при натисненні на іконку, воно відображається на весь екран у вигляді шторки з правого боку, де всі пункти розміщені в одну колонку.

На сторінці Про нас розміщено інформацію про мету розробки застосунку, різні фотографії, а також рухомий рядок із гаслом «Робимо планування здорового харчування та активності простішим!».

Нижче на рисунку 3.8 продемонстровано сторінку із відповідями на часті питання. При натисненні на блок із питанням він розгортається і там відображається написана відповідь.

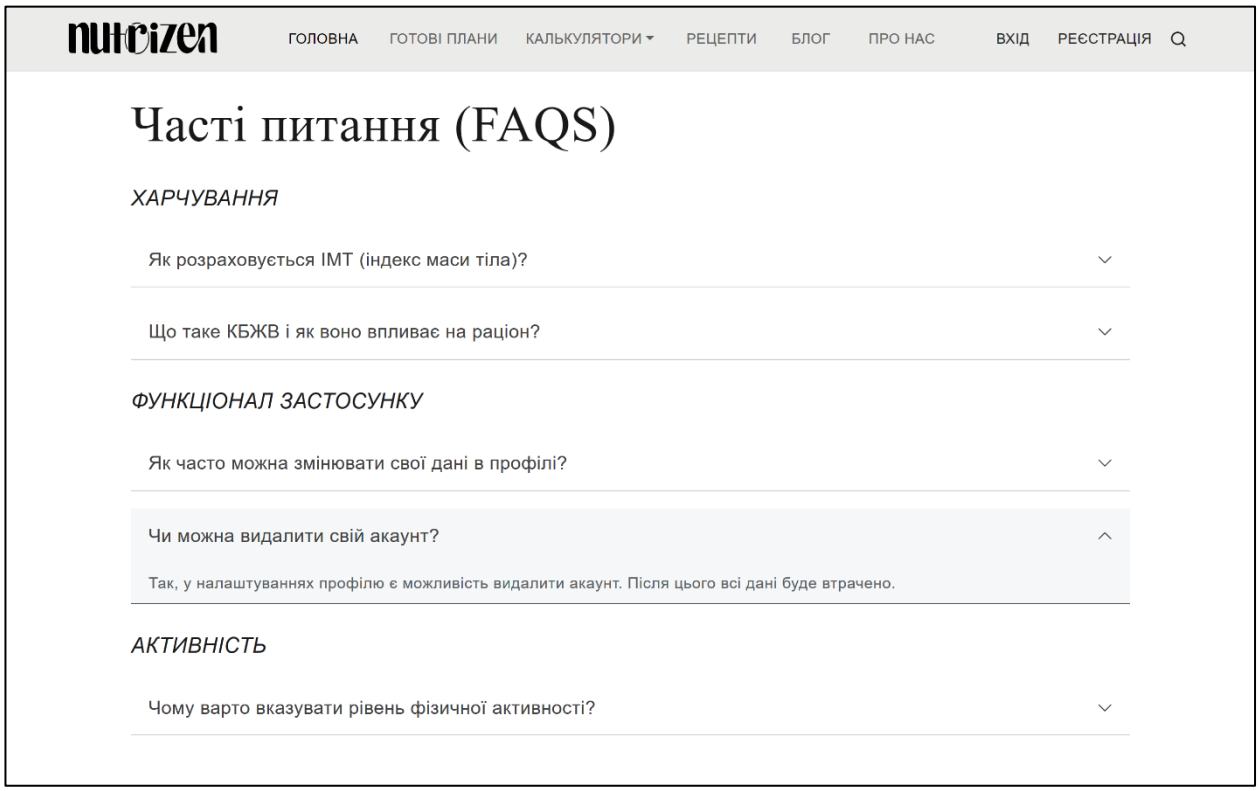

Рисунок 3.8 – Інтерфейс сторінки Часті питання

Розгортання вкладок реалізоване за допомогою стандартних елементів <details>/<summary>, що динамічно створюються за допомогою JavaScript, який також завантажує дані для відображення із json-файлу, групуючи питання за категоріями. Скрипт розміщено у додатку А.

При натисканні на пункт меню Реєстрація відкривається сторінка із відповідною формою, яка показана на рисунку 3.9.

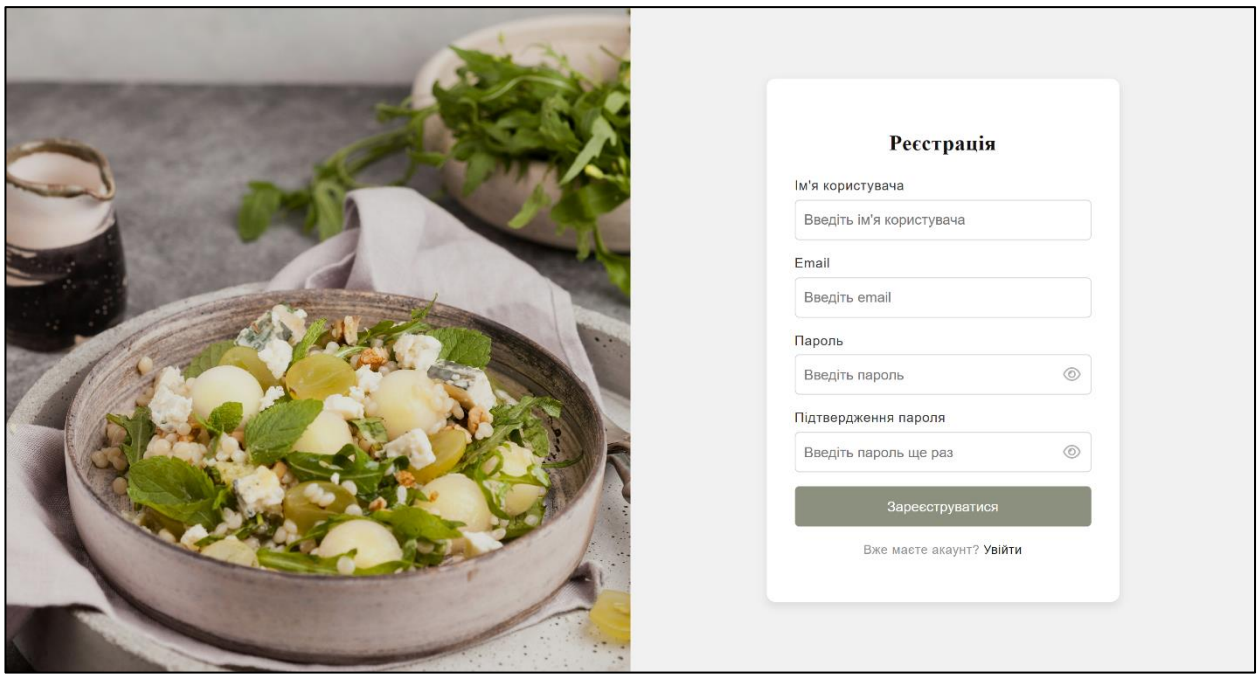

Рисунок 3.9 – Інтерфейс сторінки із формою реєстрації в «Nutrizen»

Праворуч розташована стандартна форма реєстрації, якщо користувач уже має акаунт можна перейти на сторінку авторизації за посиланням нижче. Інтерфейс сторінки входу в систему має аналогічний вигляд, але лише 2 поля у формі – ім’я користувача або електронна адреса та пароль.

3.3.2 Створення інтерфейсу профіля користувача

Авторизований користувач має змогу планувати і відстежувати свій раціон та фізичну активність. Для цього створено відповідні інтерфейси в особистому кабінеті, а також додано кнопки на інших сторінках, наприклад, на рисунку 3.10 видно, що при відкритті сторінки із конкретним рецептом відображається кнопка Додати в раціон, а для гостей вона не показується.

На сторінці рецепту прописані необхідні інгредієнти та їх кількість, а також кроки приготування страви, у таблиці наведено вміст макроелементів, який додатково відображається у вигляді діаграми нижче.

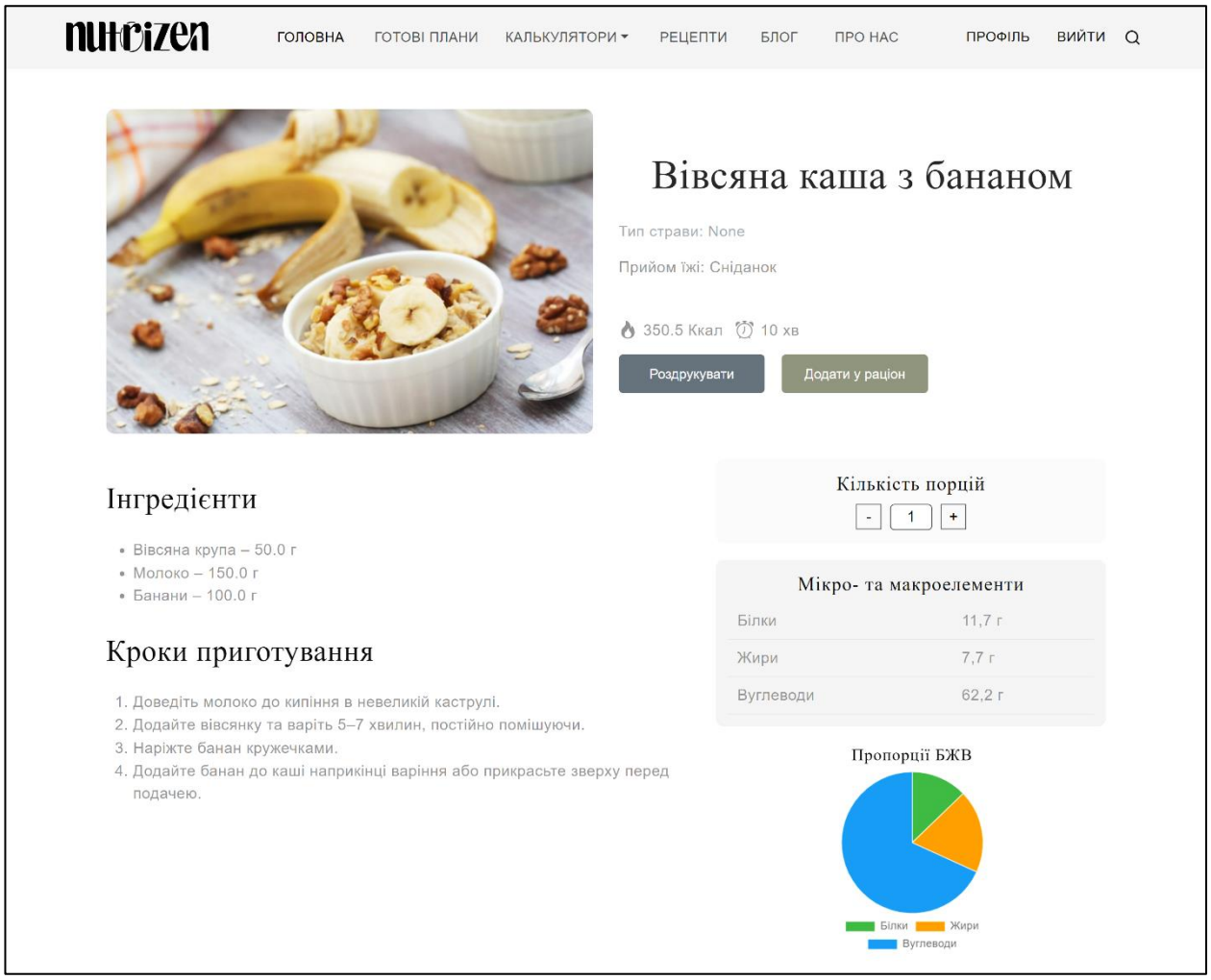

Рисунок 3.10 – Сторінка рецепту

Також можна задати більшу кількість порцій для зміни розрахунку інгредієнтів, що спростить процес приготування, адже всі пропорції будуть швидко і точно перераховані.

При перегляді сторінки із готовими планами, наведеної на рисунку 3.11, авторизованому користувачу, на відміну від гостя, доступна можливість обрання певної дієти шляхом натискання на відповідну кнопку внизу блока.

Тут розміщені найпопулярніші плани харчування, ефективність яких доведена науковцями. На кожній картці детально зазначено для кого підходить дієта, рекомендовані продукти і ті, вживання яких слід обмежити. Також наведено ризики і переваги планування раціону на основі цих програм, вказано, що спершу бажано проконсультуватися із лікарем, адже певні дієти є доволі суворими.

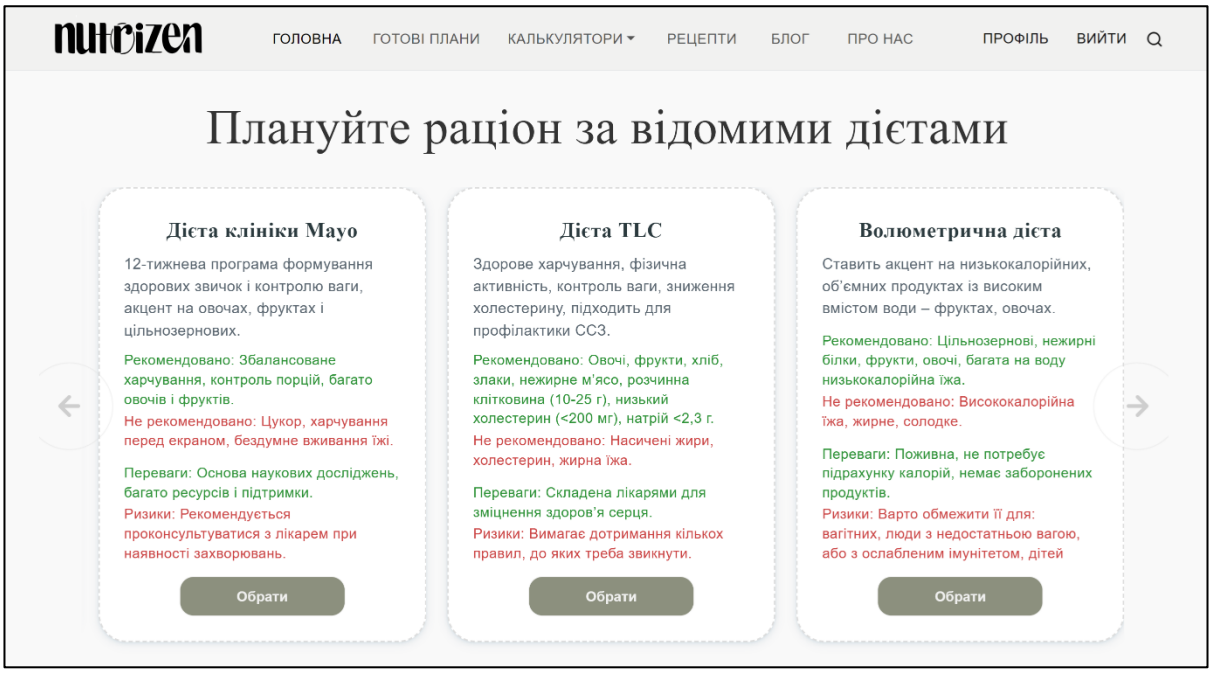

Рисунок 3.11 – Сторінка планів відомих дієт

Картки з інформацією зібрані у слайдер, який дає змогу переглянути і порівняти всі плани, щоб підібрати найбільш підходящий.

В особистому кабінеті на вкладці з історією раціону та активності, яка наведена на рисунку 3.12, відображається створений раціон за параметрами користувача, якщо він обрав певну дієту. В іншому випадку, він може самостійно записувати прийоми їжі за відповідну дату або ж обирати страви із бази автоматично. Кнопки для переходу до іншого дня знаходяться вгорі.

Дані про норми та спожиті калорії і макроелементи відображаються у вигляді діаграм, а також у текстовому форматі.

Нижче також реалізовано інтерфейс для додавання записів про фізичну активність. Для цього у модальному вікні є поля для введення активності і її тривалості для автоматичного розрахунку спалених калорій, або введення даних із інших пристроїв для врахування їх у застосунку і отримання коректних рекомендацій та аналітики. Праворуч відображено діаграму з нормою та фактичною активністю за тиждень.

Ліворуч під вкладками навігації по кабінету користувача розташовано аналіз активності за останні 7 днів та рекомендації фізичних навантажень, створені на основі віку та мети користувача.

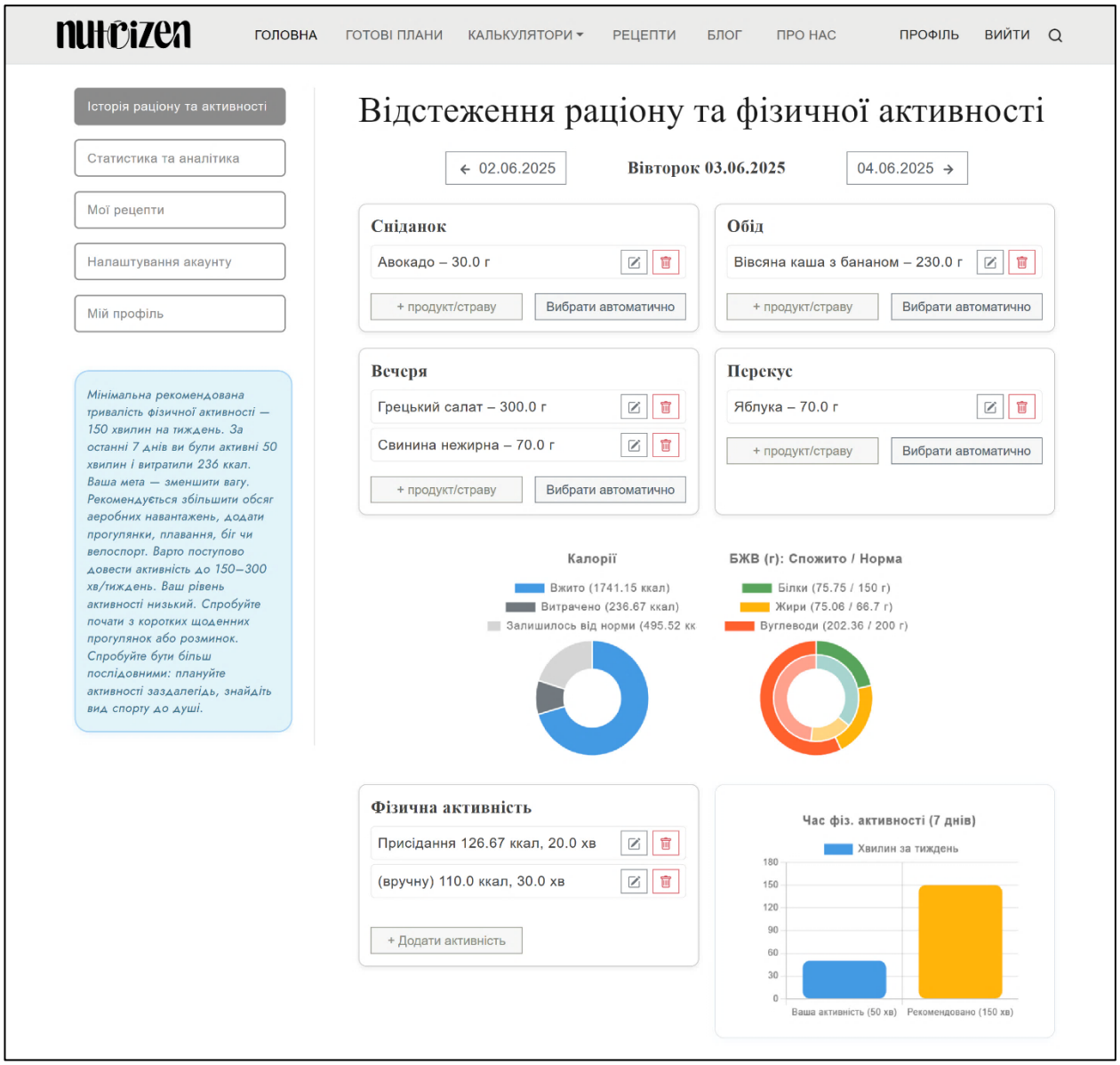

Рисунок 3.13 – Інтерфейс профіля для запису і відстеження раціону та фізичної активності

На вкладці зі статистикою та аналітикою, яка показана на рисунку 3.13, реалізовано зручний інтерфейс для перегляду графіків і діаграм зі статистичним аналізом норми активності, КБЖВ і фактичних показників користувача. Для вибору бажаного періоду аналізу над графіком є кнопки із відповідними написами.

Аналіз статистики фізичної активності відображено у вигляді графіків, а КБЖВ – за допомогою стовпчастих діаграм, щоб наочно показати норму та поточний результат.

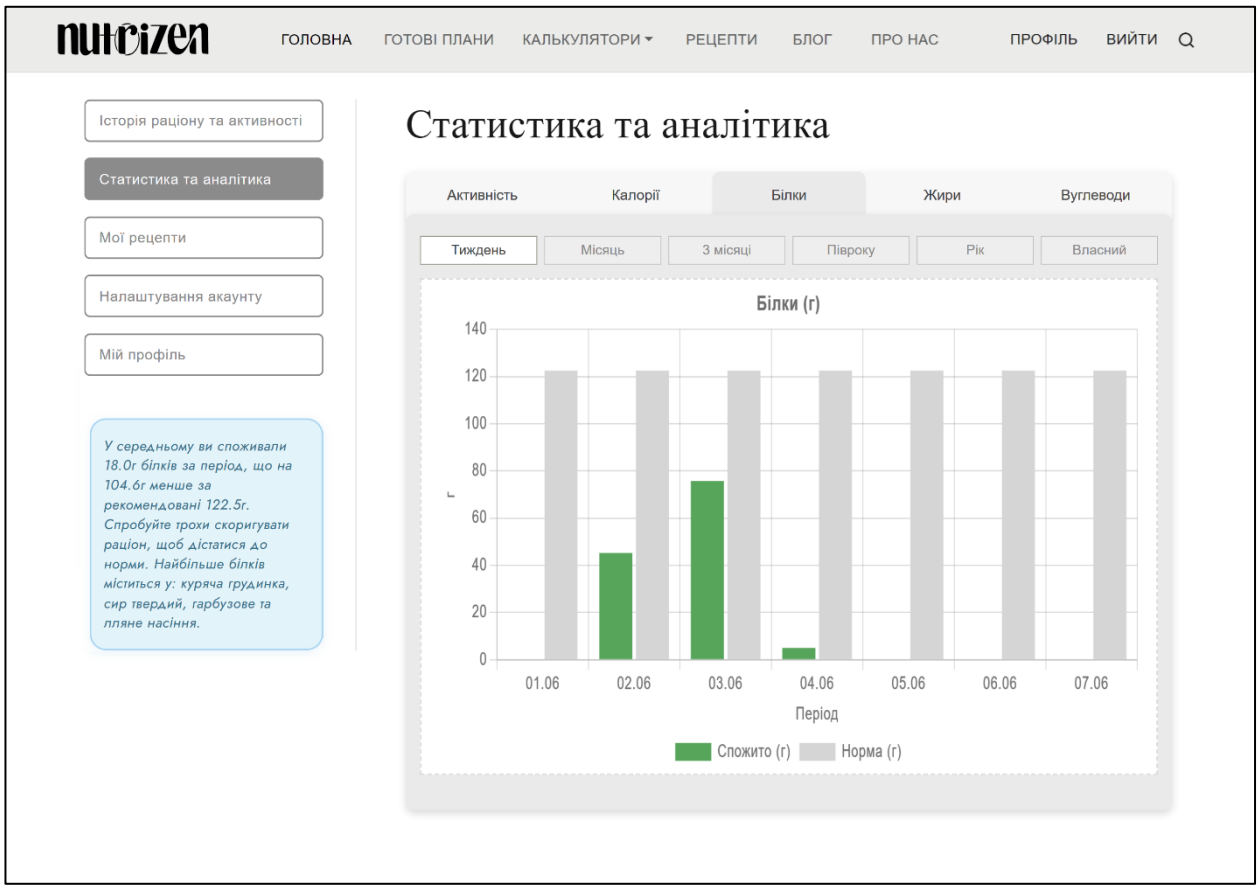

Рисунок 3.14 – Інтерфейс статистики та аналітики споживання білків за тиждень

Ліворуч під вкладками меню виводиться текстова аналітика по кожному з обраних показників за заданий період. Користувачеві, який самостійно планує своє харчування, також надаються рекомендації щодо збагачення раціону нутрієнтами, яких замало, або ж щодо зменшення їхнього надлишку.

На вкладці Мої рецепти відображено страви, які користувач додав у систему самостійно, її інтерфейс аналогічний зі сторінками Рецепти та Блог.

У налаштуваннях акаунту, які показані на рисунку 3.15, користувачі можуть змінити пароль чи повністю видалити свій акаунт.

Кнопка видалення має колір, який відрізняється від загального стилю, аби користувач підсвідомо розумів, що це спричинить втрату усіх даних і був обережним. Для видалення акаунту в модальному вікні, що відкриється, потрібно підтвердити свої наміри.

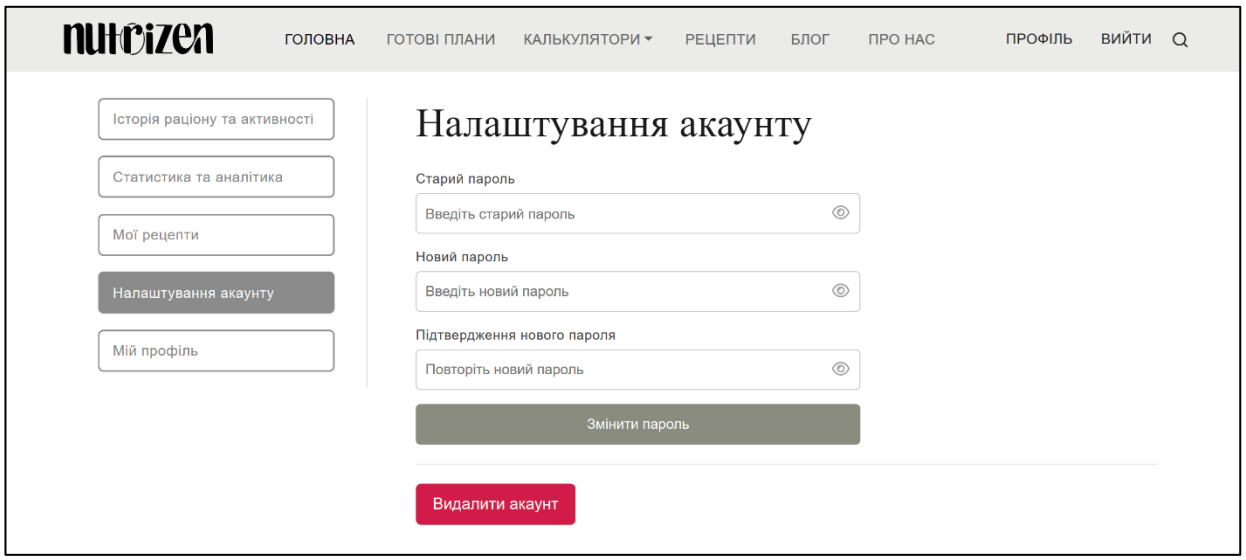

Рисунок 3.15 – Налаштування акаунту користувача «Nutrizen»

При виборі пункту Мій профіль, відображеного на рисунку 3.16, користувач може переглянути або змінити особисту інформацію, зокрема свою ціль, фізичні параметри, ім’я та прізвище, електронну адресу, вказати на що має алергію тощо.

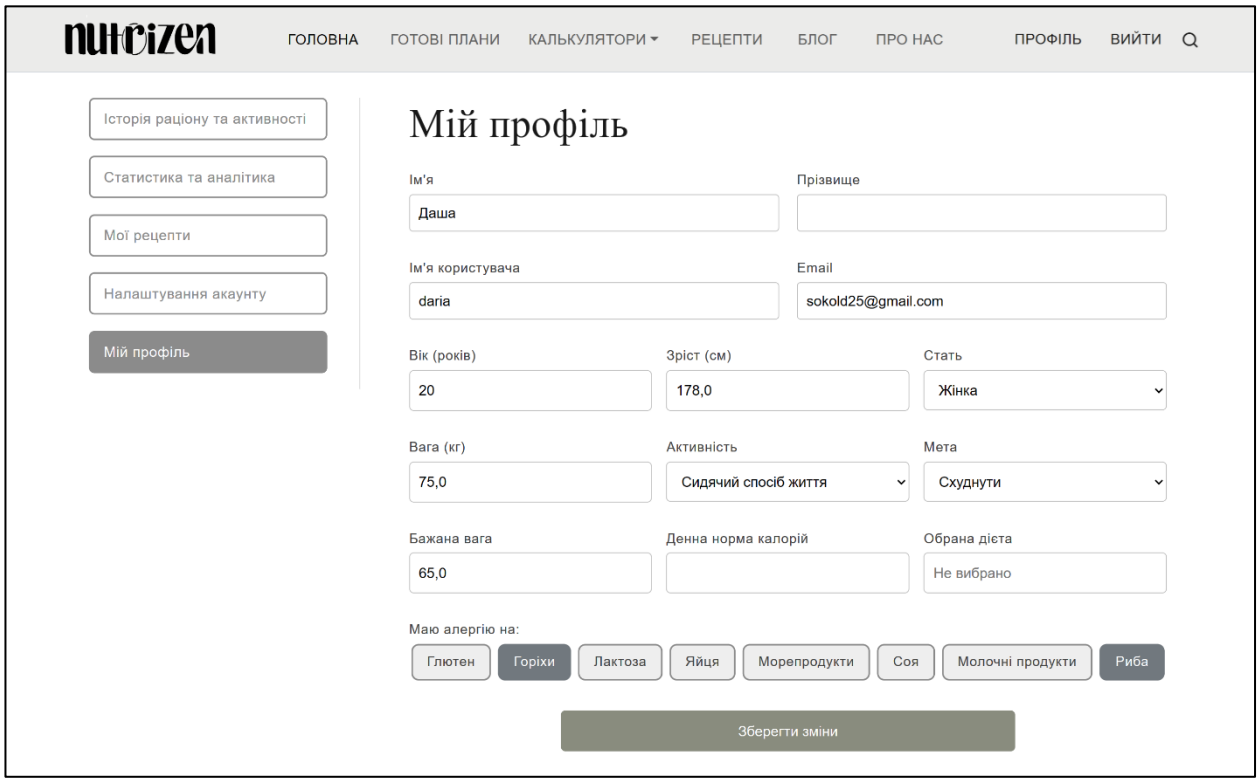

Рисунок 3.16 – Інтерфейс відображення особистих даних користувача

Інтерфейси вкладок профіля користувача спрямовані на надання найбільш глибокої аналітики щодо харчування та фізичної активності та зручного редагування особистих даних.

3.3.3 Створення інтерфейсу панелі управління системою

Користувачі із ролями адміністратора та редактора уповноважені додавати, редагувати та видаляти плани дієт, рецепти, статті. Для цього на відповідних сторінках біля кожного об’єкта з’являються кнопки, які надають можливість виконувати ці дії.

При натисканні на кнопку додавання або редагування даних відкриваються відповідні форми, а для видалення достатньо підтвердити дію у модальному вікні. На рисунку 3.17 показана форма додавання нової статті.

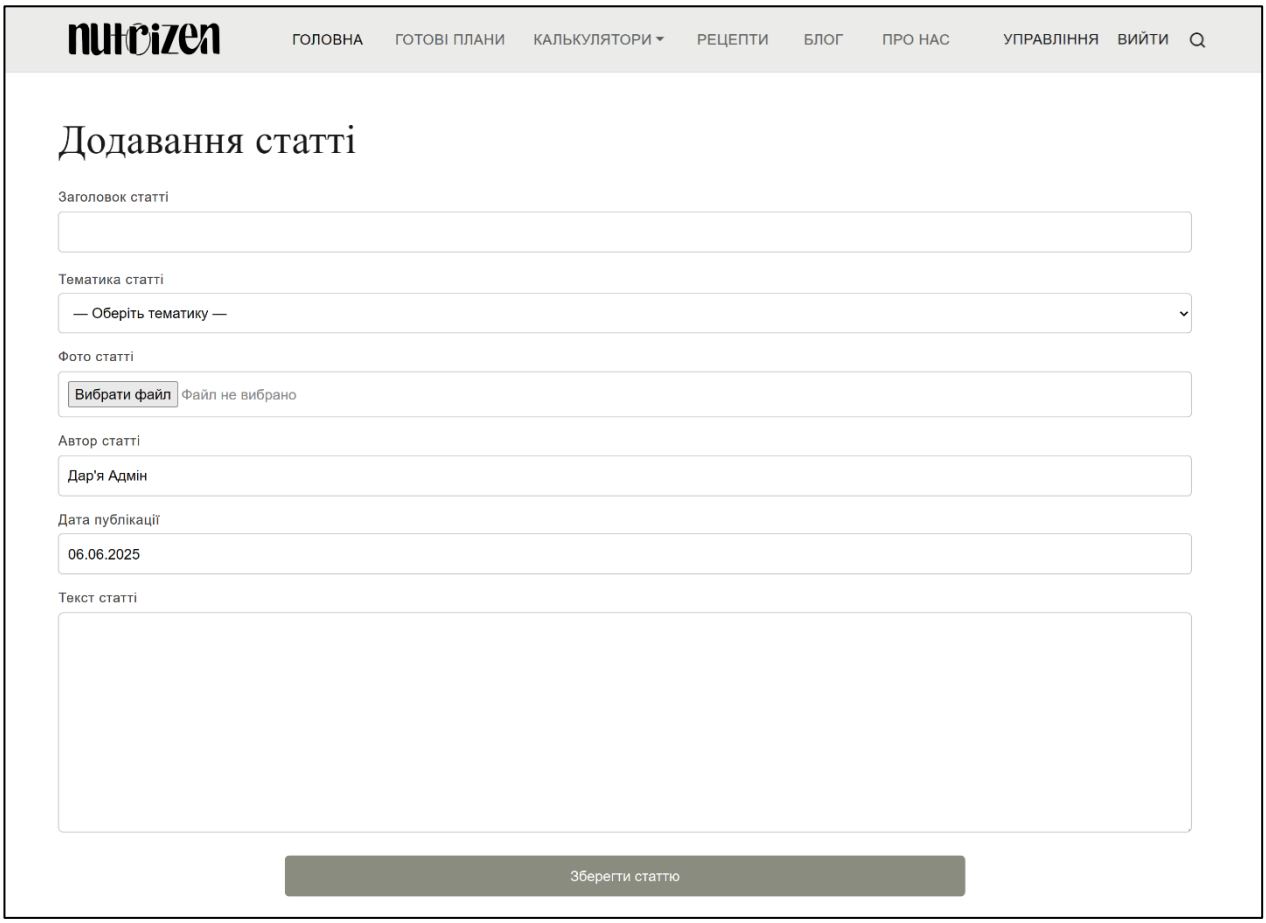

Рисунок 3.17 – Інтерфейс форми додавання статті у «Nutrizen»

Усі інші форми додавання та редагування мають аналогічний інтерфейс.

У профілі редактора доступні вкладки Продукти і Типи активності для додавання, редагування або видалення їх із бази, а також пункти меню Налаштування акаунту та Мій профіль, які мають такий же інтерфейс, як у авторизованих користувачів, тільки тут відсутні поля для фізичних показників, цілей тощо.

У адміністратора крім цих вкладок є ще одна – Користувачі, яка необхідна для додавання нових користувачів із заданою роллю, редагування ролей існуючих і видалення акаунтів. На рисунку 3.18 представлена панель керування користувачами вебзастосунку.

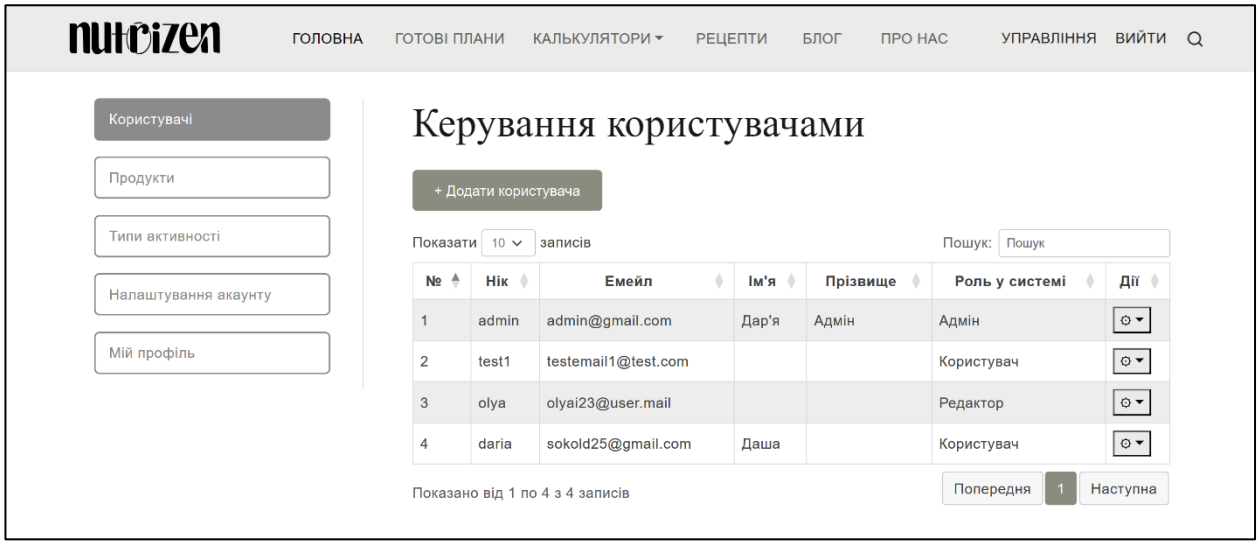

Рисунок 3.18 – Адміністративна панель керування користувачами «Nutrizen»

Кнопки редагування і видалення знаходяться в таблиці у випадному списку в стовпці Дії. Інтерфейс для додавання і редагування користувача подібний до інших форм, але реалізований у модальному вікні. Для видалення також потрібно підтвердити дію у діалоговому вікні.

Вищезгадані сторінки, які є спільними для редактора та адміністратора, мають такий же інтерфейс, тільки відображають інші дані з бази – про продукти та типи активностей відповідно.

3.4 Реалізація backend-частини

Бізнес-логіка вебзастосунку для планування раціону та фізичної активності реалізована на мові Python. В основі лежить ідея комплексного персоналізованого підходу до кожного користувача, відповідно до його особистих параметрів. Усі основні моделі, необхідні для реалізації подібної системи, розроблені за допомогою взаємопов'язаних моделей. Нижче на рисунку 3.19 наведена модель користувача вебзастосунку.

```
class User(AbstractUser):
    age = models.IntegerField(null=True, blank=True)
    weight = models.FloatField(null=True, blank=True)
    height = models.FloatField(null=True, blank=True)
    gender = models.CharField(choices=GENDER_CHOICES, null=True, blank=True)
    activity_level = models.CharField(choices=ACTIVITY_CHOICES, null=True, blank=True)
    goal = models.CharField(choices=GOAL_CHOICES, null=True, blank=True)
    allergic_to = models.JSONField(null=True, blank=True)
    target_weight = models.FloatField(null=True, blank=True)
    calories_intake = models.FloatField(null=True, blank=True)
    selected_diet = models.ForeignKey(DietPlan, blank=True, on_delete=models.SET_NULL)

    def calculate_calorie_needs(self):
        if not (self.gender and self.weight and self.height and self.age
                and self.activity_level and self.goal):
            return None
        ACTIVITY_FACTORS = {
            'sedentary': 1.2,
            'light': 1.375,
            'moderate': 1.55,
            'active': 1.725,
            'very_active': 1.9,
        }
        if self.gender == 'F':
            ree = (10 * self.weight) + (6.25 * self.height) - (5 * self.age) - 161
        else:
            ree = (10 * self.weight) + (6.25 * self.height) - (5 * self.age) - 5
        pa = ACTIVITY_FACTORS.get(self.activity_level, 1.2)
        tdee = ree * pa
        if self.goal == 'lose':
            calories = tdee * 0.85
        elif self.goal == 'gain':
            calories = tdee * 1.10
        else:
            calories = tdee
        return round(calories)
```

Рисунок 3.19 – Модель User

Стандартна модель користувача у Django була розширена додатковими полями, які необхідні для працездатності створюваного програмного продукту, а також дозволяють реалізувати індивідуальні особливості та потреби, автоматично виконувати розрахунки добової потреби в калоріях чи складати рекомендації щодо фізичної активності. Повний код файлу models.py із усіма моделями системи розміщено у додатку Б.

Авторизований користувач вебзастосунку не просто веде облік спожитих продуктів, а ще має можливість формувати денні прийоми їжі, обирати готові рецепти або додавати власні, а також вводити точну вагу порцій. Уся інформація про харчову цінність окремих продуктів та страв автоматично підсумовується та виводиться на екран. Це дозволяє точно планувати раціон, ретельно відстежувати баланс нутрієнтів і стежити за виконанням власних цілей. Нижче на рисунку 3.20 наведено код функції, яка реалізує додавання страви або продукту у прийом їжі обраного дня.

```
def ajax_add_meal_item(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        meal_type = data.get('meal_type')
        object_id = data.get('object_id')
        object_type = data.get('object_type')
        quantity = float(data.get('quantity', 0))
        date_str = data.get('meal_date')
        try:
            meal_date = datetime.strptime(date_str, '%Y-%m-%d').date()
        except (ValueError, TypeError):
            meal_date = date.today()
        meal, _ = Meal.objects.get_or_create(user=request.user, date=meal_date, meal_type=meal_type)
        if object_type == 'product':
            product = Product.objects.get(id=object_id)
            item = MealItem.objects.create(meal=meal, product=product, quantity=quantity)
            name = product.name
        elif object_type == 'recipe':
            recipe = Recipe.objects.get(id=object_id)
            item = MealItem.objects.create(meal=meal, recipe=recipe, quantity=quantity)
            name = recipe.name
        else:
            return JsonResponse({'success': False, 'error': 'Invalid object type'})
        meals = Meal.objects.filter(user=request.user,
                            date=meal_date).prefetch_related('items__product', 'items__recipe')
        activities = Activity.objects.filter(user=request.user, date=meal_date)
        summary = calculate_summary(meals, activities, request.user)
        return JsonResponse({'success': True, 'item_id': item.id, 'name': name,
                             'quantity': quantity, 'summary': summary, 'meal_type': meal_type
        })
    return JsonResponse({'success': False, 'error': 'Invalid request'})
```

Рисунок 3.20 – Додавання страви або продукту у прийом їжі

Ця функція обробляє AJAX-запит на додавання нового елемента до прийому їжі користувача. Вона приймає дані із фронтенду, визначає дату і кількість, а також додається цілий рецепт чи окремий продукт. Шукає або створює відповідний прийом їжі для поточного користувача і дня, додає новий продукт чи страву до цього прийому і повертає назад відповідь із інформацією для відображення або помилку.

Наявна можливість автоматичного обрання страви для обраного прийому їжі. Код функції, яка це реалізує наведений нижче на рисунку 3.21.

```
def generate_user_meal(user, meal_type):
    # Визначаємо список алергенів користувача
    user_allergens = user.allergic_to or []

    # Діапазон прийнятної калорійності страви
    calories_per_meal = (user.calories_intake or 1800) // 4

    calorie_min = calories_per_meal * 0.8
    calorie_max = calories_per_meal * 1.2

    print(calorie_min, calorie_max)

    # Вибираємо рецепти для цього типу прийому їжі
    qs = Recipe.objects.filter(
        meal_type=meal_type,
        #total_calories__gte=calorie_min,
        total_calories__lte=calorie_max,
    )

    # Відкидаємо ті, де є алергени користувача
    def is_safe(recipe):
        for item in recipe.recipe_items.select_related('product'):
            if hasattr(item.product, 'is_allergic') and item.product.is_allergic:
                # продукт є алергеном
                if item.product.name.lower() in user_allergens:
                    return False
        return True

    safe_recipes = [r for r in qs if is_safe(r)]

    if not safe_recipes:
        return None

    return random.choice(safe_recipes)
```

Рисунок 3.21 – Генерація страви відповідно до індивідуальних особливостей користувача

Дана функція підбирає для користувача випадковий рецепт певного типу прийому їжі, щоб страва відповідала рекомендованій калорійності на порцію відповідно до денної потреби і не містила інгредієнтів, на які у користувача є алергія. Якщо є підходящі страви, повертається одна випадкова, інакше – None.

Окрім харчування, у систему інтегровано облік фізичної активності. Для цього є модель різних типів активності, яка дозволяє автоматично підраховувати витрачені калорії залежно від тривалості активності та ваги користувача. Можна не лише вибирати активність зі списку, так і вводити витрачені калорії вручну. Користувач бачить не лише скільки калорій спожито, а й скільки витрачено, таким чином відбувається реальний аналіз прогресу.

Додання записів про фізичну активність реалізовано подібним чином до додавання їжі. Повний код цих функцій розміщено у views.py в додатку Б.

Щоб користувач отримував не лише цифри та графіки, реалізовано окрему логіку створення персоналізованих рекомендацій і мотиваційних повідомлень. На рисунку 3.22 наведено фрагмент функції, яка це реалізує.

```
if type_ == 'calories':
    consumed = bucket_sum['calories']
    burned   = bucket_sum['burned']
    avg_cons = sum(consumed) / num_buckets if num_buckets else 0
    avg_burn = sum(burned) / num_buckets if num_buckets else 0
    avg_targ = sum(target_arr) / num_buckets if num_buckets else 0
    analysis = ""
    if avg_cons > avg_targ:
        diff = round(avg_cons - avg_targ,1)
        analysis += (
            f"В середньому ви споживали {round(avg_cons,1)} ккал за період, "
            f"що на {diff} ккал більше за рекомендовані {round(avg_targ,1)}. ")
    else:
        diff = round(avg_targ - avg_cons,1)
        analysis += (
            f"В середньому ви споживали {round(avg_cons,1)} ккал за період, "
            f"що на {diff} ккал менше за рекомендовані {round(avg_targ,1)}. ")
    if avg_burn < avg_cons * 0.1:
        analysis += (
            f"Ви витрачали лише {round(avg_burn,1)} ккал за період, "
            "тому варто додати трохи більше фізичної активності."
            "Оптимальним варіантом буде, наприклад, 5000-6000 кроків/день та силові"
            "навантаження 1-2 рази на тиждень")
    else:
        analysis += (
            f"Ви витрачали в середньому {round(avg_burn,1)} ккал за період. "
            "Добре поєднуєте харчування з активністю!")
```

Рисунок 3.22 – Створення аналітики споживання і витрати калорій

Представлений фрагмент функції аналізує дані по калоріях за обраний період, порівнюючи з рекомендованою нормою скільки калорій було в середньому спожито та скільки витрачено. Далі формується текстовий аналіз і повертаються масиви даних для побудови графіків спожитих, витрачених і цільових калорій разом із текстовою порадою у форматі JSON.

Усі дії в кабінеті користувача виконуються через AJAX-запити, тобто додавання, редагування, видалення записів відбувається інтерактивно без перезавантаження сторінки. Це дає змогу одразу побачити оновлену статистику.

Для пошуку продуктів, страв і типів активностей при додаванні записів створено технологію підказок-автозаповнень зі знайденими у базі відповідними словами, що робить користування застосунком ще швидшим і зручнішим. На рисунку 3.23 наведено функцію, яка це реалізує.

```
def autocomplete_suggestions(request):
    query = request.GET.get('term', '')
    suggestions = []

    if query:
        # Продукти
        products = Product.objects.filter(name__icontains=query)[:5]
        suggestions += [
            {'type': 'product', 'id': p.id, 'name': p.name} for p in products
        ]

        # Рецепти
        recipes = Recipe.objects.filter(name__icontains=query)[:5]
        suggestions += [
            {'type': 'recipe', 'id': r.id, 'name': r.name} for r in recipes
        ]

    return JsonResponse(suggestions, safe=False)
```

Рисунок 3.23 – Реалізація підказок із автозаповненням

Функція приймає пошуковий запит із GET-параметра, знаходить 5 найбільш відповідних продуктів і рецептів, які містять цей текст у назві, і повертає їх у вигляді списку підказок для автозаповнення у форматі JSON.

Функціонал адміністраторської панелі управління системою включає можливість додавання нового користувача, на рисунку 3.24 наведений фрагмент функції, яка це реалізовує.

```
# Додавання нового користувача через модальне вікно
if request.user.is_superuser or request.user.groups.filter(name="Admin").exists():
    if request.method == 'POST' and 'adding_new_user' in request.POST:
        username = request.POST.get('username')
        email = request.POST.get('email')
        first_name = request.POST.get('first_name', '')
        last_name = request.POST.get('last_name', '')
        password = request.POST.get('password')
        role = request.POST.get('role', 'user')

        errors = {}
        if User.objects.filter(username=username).exists():
            errors['username'] = ['Користувач з таким іменем вже існує']
        if User.objects.filter(email=email).exists():
            errors['email'] = ['Користувач з таким email вже існує']
        if len(password) < 6:
            errors['password'] = ['Пароль має містити щонайменше 6 символів']

        if errors:
            context['user_form_errors'] = errors
            return render(request, 'profile.html', context)

        user = User.objects.create_user(
            username=username,
            email=email,
            password=password,
            first_name=first_name,
            last_name=last_name
        )
        # Призначення ролі
        group_name = {'admin': 'Admin', 'editor': 'Editor'}.get(role)
        if group_name:
            group = Group.objects.get(name=group_name)
            user.groups.add(group)
        messages.success(request, f"Користувача {user.username} створено успішно.")
        return redirect('profile-page')
```

Рисунок 3.24 – Додавання адміністратором нового користувача

У коді перевіряється чи поточний користувач має права адміністратора. Якщо надходить POST-запит із форми, дані валідуються, створюється новий користувач. Роль користувача визначається тим, до якої групи його додали.

Якщо є помилки, вони повертаються у шаблон для відображення на формі модального вікна, інакше – користувач додається і сторінка оновлюється.

Загалом додавання і редагування решти інформації через панель адміністратора або редактора відбуваються подібним чином. Повний код функцій наведено у додатку Б.

Розроблена бекенд-частина проєкту не лише обслуговує запити з інтерфейсу, а динамічно формує індивідуальний інформаційний простір для кожного користувача, забезпечує всі розрахунки та надання рекомендацій, і, зрештою, створює відчуття, що вебзастосунок здатен розуміти та підтримувати користувача на шляху до його мети.

3.5 Створення бази даних

Підключення до бази даних задано у файлі settings.py через відповідний блок. Структура бази даних була сформована автоматично на основі описаних моделей у файлі models.py, який наведено в додатку Б.

Для створення реальних таблиць у базі даних використовується система міграцій Django. Щоразу після визначення нових моделей або зміни існуючих запускалися команди для виконання міграцій, у результаті яких в базі з’являлися нові таблиці або додавалися чи видалялися поля в існуючих.

База даних створюваного вебзастосунку складається з декількох основних таблиць, які взаємопов’язані між собою. Кожна з таблиць відповідає окремому аспекту планування раціону та фізичної активності. Центральним об’єктом у базі є користувач для якого, крім стандартної інформації, зберігаються ще й особисті параметри, необхідні для персоналізованих розрахунків, аналітики, рекомендацій та складання планів.

Прийоми їжі зберігаються окремо для кожного дня і типу (сніданок, обід тощо), до кожного прийому додаються продукти або рецепти із вказаною кількістю спожитої їжі. Таблиця активностей користувача зберігає інформацію про тип, тривалість і витрачені калорії.

Усі ці таблиці пов’язані між собою через зовнішні ключі, що дозволяє легко отримувати та аналізувати інформацію про раціон і фізичну активність кожного користувача.

3.6 Тестування розробленого програмного продукту

Тестування створеного вебзастосунку проводилося вручну, без використання автоматизованих фреймворків. Весь функціонал перевірявся безпосередньо у браузері на локальному сервері.

Перевірялася на відповідність очікуваним сценаріям робота всіх модулів системи. Проходилися усі можливі варіанти використання з боку усіх визначених користувацьких ролей. Особлива увага приділялася перевірці логіки додавання, редагування та видалення даних, правильності розрахунків і реакції інтерфейсу на невалідні дані – помилкові або порожні значення. Також перевірялася робота і на валідних даних. Виявлені недоліки виправлялися одразу та перевірялися повторно.

У ході тестування кросбраузерності та адаптивності до різних розмірів екранів пристроїв було перевірено правильність відображення елементів інтерфейсу, шрифтів, доступність і клікабельність кнопок та пунктів меню. Знайдені проблеми із центруванням елементів було швидко усунуто.

Тестування – невід’ємний етап, важливий для виявлення та усунення можливих помилок у функціоналі програмного продукту або багів у його інтерфейсі. Проведене тестування не виявило серйозних проблем у працездатності вебзастосунку, а знайдені дрібні недоліки одразу були виправлені, тому система працює коректно і відповідає усім вимогам. При введенні невалідних даних не виникає екстрене завершення роботи, а виводяться необхідні попередження про помилки, успішне виконання дій супроводжується відповідними сповіщеннями для користувачів. У ході тестування верстки було перевірено кросбраузерність у трьох популярних браузерах та коректність відображення в десктопній і мобільній версіях шляхом зміни розмірів екрану в інструментах розробника в браузері.

ВИСНОВКИ

У кваліфікаційній роботі проведений аналіз наукових статей і досліджень у частині збалансованого харчування та визначення фізичних параметрів тіла. Здійснено комплексну оцінку переваг і недоліків впровадження програмних продуктів у предметній області, а також проаналізовано наявні аналоги.

Актуальність реалізації вебзастосунку для планування раціону та фізичної активності визначається тим, що у світі досі спостерігається зростання кількості людей із надлишковою вагою або ожирінням, нерідко таку проблему мають молоді люди та навіть підлітки і діти. А також на противагу цьому серед деяких груп населення поширюється загальний тренд на усвідомлене ставлення до здоров'я. Саме тому необхідно впроваджувати цифрові інструменти у процеси відстеження і планування харчування, боротьби з ожирінням, причинами його виникнення і можливими наслідками та для загального формування корисних звичок серед населення.

За допомогою UML-діаграм спроєктовано структуру, розподіл функціоналу між ролями користувачів і базу даних застосунку. Вибір технологій для реалізації бекенд-частини зумовлений необхідністю забезпечення високого рівня безпеки даних, а також гнучкості та можливості масштабування проєкту. Інтеграція стандартних фронтенд-технологій із AJAX зробила інтерфейс, дійсно, динамічним і зручним для користувачів.

До ключових особливостей розробленого вебзастосунку, які вирізняють його з-поміж існуючих рішень, належать:

– комплексний підхід до здорового способу життя через контроль харчування та активності;

– персоналізація застосунку для кожного користувача – розрахунки на основі індивідуальних показників, автоматичне визначення оптимального балансу нутрієнтів тощо;

– зрозумілий і приємний інтерфейс українською мовою;

– подання статистичних даних у вигляді графіків і діаграм – наочно та максимально спрощено для розуміння;

– аналітика харчування та рекомендації фізичної активності відповідно до потреб кожного користувача сприяють швидшому досягненню поставлених цілей;

– достовірні розрахунки на основі даних із наукових джерел.

Результатом є повнофункціональний вебзастосунок, який стане хорошим інструментом для відстеження способу життя і профілактики виникнення проблем зі здоров’ям, спричинених нераціональним харчуванням і малорухливістю. У майбутньому можливе розширення функціональних можливостей, наприклад, додавання модуля відстеження за дотриманням водного балансу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Здорове харчування: ВООЗ оновила рекомендації щодо вживання жирів і вуглеводів. Центр громадського здоров'я МОЗ України. [Електронний ресурс] / Режим доступу: https://www.phc.org.ua/news/zdorove-kharchuvannya-vooz-onovila-rekomendacii-schodo-vzhivannya-zhiriv-i-vuglevodiv

2. Що і як їсти, щоб жити довго: рекомендації МОЗ. Міністерство охорони здоров'я України. [Електронний ресурс] / Режим доступу: https://moz.gov.ua/uk/scho-i-jak-isti-schob-zhiti-dovgo-rekomendacii-moz

3. My Diet Meal Plan [Електронний ресурс] / Режим доступу до ресурсу: https://www.mydietmealplan.com

4. Таблиця калорійності [Електронний ресурс] / Режим доступу до ресурсу: https://www.tablycjakalorijnosti.com.ua

5. Eating Well [Електронний ресурс] / Режим доступу до ресурсу: https://www.eatingwell.com

6. Білецький В. С. Методологія наукових досліджень технічних об'єктів та їх оптимізація [Електронний ресурс] : навч. посібник / В. С. Білецький ; Нац. техн. ун-т «Харків. політехн. ін-т». – 2023. – 118 с. – Режим доступу до ресурсу: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/04399d6a-36c7-4bd6-949e-9a756e6908ef/content

7. Carpenter K., Truswell A. S, et al. BMR and REE: energy balance. Encyclopedia Britannica. URL: https://www.britannica.com/science/human-nutrition/BMR-and-REE-energy-balance

8. Mifflin MD, St Jeor ST, et al. A new predictive equation for resting energy expenditure in healthy individuals. Am J Clin Nutr. 1990;51(2):241–247

9. Плескач В. Л., Вакуленко Є. О., Сердюк А. А. Програмна система розрахунку калорій. Проблеми програмування. 2024. № 2-3. С. 199–206. – Режим доступу до ресурсу: https://doi.org/10.15407/pp2024.02-03.199

10. Cadeddu, L.; Meles, M. The diet problem, a mathematical approach, Progress in Nutrition vol. 25 (2) (2023), 1-8. URL: https://doi.org/10.23751/pn.v25i2.13266

11. Як визначити оптимальну вагу: формула індексу маси тіла. Міністерство охорони здоров'я України. [Електронний ресурс] / Режим доступу до ресурсу: https://moz.gov.ua/uk/jak-viznachti-optimalnu-vagu-formula-indeksu-masi-tila

12. Predicting Percent Body Fat from Circumference Measurements / C. L. Shake et al. Military Medicine. 1993. Vol. 158, no. 1. P. 26-31. URL: https://doi.org/10.1093/milmed/158.1.26

13. Django [Електронний ресурс] / Режим доступу до ресурсу: https://www.djangoproject.com

14. PostrgeSQL [Електронний ресурс] / Режим доступу до ресурсу: https://www.postgresql.org

15. Docker [Електронний ресурс] / Режим доступу до ресурсу: https://www.docker.com

16. HTML [Електронний ресурс] / Режим доступу до ресурсу: https://dev.w3.org/html5/spec-LC/

17. CSS [Електронний ресурс] / Режим доступу до ресурсу: https://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

18. JavaScript [Електронний ресурс] / Режим доступу до ресурсу: https://tc39.es/ecma262/

19. Bootstrap [Електронний ресурс] / Режим доступу до ресурсу: https://getbootstrap.com

20. Visual Studio Code [Електронний ресурс] / Режим доступу до ресурсу: https://code.visualstudio.com

21. Методичні вказівки до виконання кваліфікаційної роботи на здобуття ступеня вищої освіти бакалавра студентам усіх форм навчання за спеціальністю 121 – Інженерія програмного забезпечення / укладачі: І. О. Доценко, О. Г. Рибальченко, А. М. Стрюк, Н. Н. Шаповалова; рецензент А. І. Купін; відповідальний за випуск А. А. Азарян. – Кривий Ріг, 2023. – 39 с.

Додаток А – Реалізація візуального інтерфейсу

Файл «scripts.js»

```
// Відображення калькуляторів
document.addEventListener("DOMContentLoaded", function () {
   let urlParams = new URLSearchParams(window.location.search);
   let calculator = urlParams.get("calculator") || "bmi";

   showCalculator(calculator);
});

// Інформація про калькулятори
async function loadCalculatorInfo() {
   const response = await
fetch("/static/default/json/calculators.json");
   const data = await response.json();
   return data;
}

// Функція перемикання калькуляторів
async function showCalculator(calcId) {
   let sections = document.querySelectorAll(".calculator-
section");
   sections.forEach(section => section.style.display = "none");
   document.getElementById(calcId).style.display = "block";
   let buttons = document.querySelectorAll(".tab-button");
   buttons.forEach(btn => btn.classList.remove("active"));
   let activeButton = document.querySelector(`.tab-
button[onclick="showCalculator('${calcId}')"]`);
   if (activeButton) {
      activeButton.classList.add("active");
   }

// Видалення дієти
function deletePlan(planId) {
  window.location.href = `/recipes/${planId}/delete/`;
}

// Калькулятор ІМТ
document.addEventListener("DOMContentLoaded", function () {
      let calcId = "{{ calculator }}";
      showCalculator(calcId);
      drawBmiGauge();
      drawFatGauge(NaN, getSelectedGender());
      drawMacroChart(10, 10, 10);
});

function getSelectedGender() {
      const genderInput =
document.querySelector('input[name="gender-fat"]:checked');
      return genderInput ? genderInput.value : "male";
```

```
}

function calculateBMI() {
	const weight =
parseFloat(document.getElementById('weight').value);
	const height =
parseFloat(document.getElementById('height').value) / 100;
	const bmi = weight / (height * height);
	document.getElementById('bmi-result').innerText =
bmi.toFixed(1);
	let desc = "";
	if (bmi < 18.5) {
		desc = "Дефіцит ваги";
	} else if (bmi < 25) {
		desc = "Норма";
	} else if (bmi < 30) {
		desc = "Надлишкова вага";
	} else {
		desc = "Ожиріння";
	}
	document.getElementById('bmi-description').innerText =
desc;
	drawBmiGauge(bmi);
}

// Діаграма результату ІМТ
function drawBmiGauge(bmi = null) {
	const canvas = document.getElementById('bmiGauge');
	const ctx = canvas.getContext('2d');
	ctx.clearRect(0, 0, canvas.width, canvas.height);
	const cx = canvas.width / 2;
	const cy = canvas.height * 1.1;
	const radius = 110;
	const ranges = [
		{ min: 10, max: 18.5, color: "#5bc0de" },
		{ min: 18.5, max: 24.9, color: "#5cb85c" },
		{ min: 25.0, max: 29.9, color: "#f0ad4e" },
		{ min: 30.0, max: 40.0, color: "#d9534f" }
	];
	const minBMI = 10;
	const maxBMI = 40;
	// Кольорові дуги
	ranges.forEach(r => {
		const startAngle = Math.PI + Math.PI * (r.min -
minBMI) / (maxBMI - minBMI);
		const endAngle = Math.PI + Math.PI * (r.max -
minBMI) / (maxBMI - minBMI);
		ctx.beginPath();
		ctx.arc(cx, cy, radius, startAngle, endAngle,
false);
		ctx.lineWidth = 20;
		ctx.strokeStyle = r.color;
		ctx.stroke();
```

```
      });
      // Підписи діапазонів
      const labels = [10, 18.5, 25, 30, 40];
      ctx.fillStyle = "#000";
      ctx.font = "12px Arial";
      labels.forEach(val => {
            const angle = Math.PI + Math.PI * (val - minBMI) /
(maxBMI - minBMI);
            const x = cx + (radius + 18) * Math.cos(angle);
            const y = cy + (radius + 18) * Math.sin(angle);
            const label = val === 10 ? "<18.5" : val === 40 ?
">30" : val;
            ctx.fillText(label, x - 10, y + 4);
      });
      // Стрілка (після розрахунку)
      if (bmi !== null) {
            const clampedBMI = Math.min(maxBMI, Math.max(minBMI,
bmi));
            const angle = Math.PI + Math.PI * (clampedBMI -
minBMI) / (maxBMI - minBMI);
            const arrowLength = radius - 10;
            ctx.beginPath();
            ctx.moveTo(cx, cy);
            ctx.lineTo(cx + arrowLength * Math.cos(angle), cy +
arrowLength * Math.sin(angle));
            ctx.strokeStyle = "#000";
            ctx.lineWidth = 3;
            ctx.stroke();
            // Центр
            ctx.beginPath();
            ctx.arc(cx, cy, 5, 0, 2 * Math.PI);
            ctx.fillStyle = "#000";
            ctx.fill();
      }
}

// Калькулятор відсотка жиру
function calculateFat() {
      const gender = document.querySelector('input[name="gender-
fat"]:checked').value;
      const height = parseFloat(document.getElementById('fat-
height').value);
      const neck =
parseFloat(document.getElementById('neck').value);
      const waist =
parseFloat(document.getElementById('waist').value);
      const hips =
parseFloat(document.getElementById('hips').value);
      let fatPercent = 0;
      if (gender === "male") {
            fatPercent = 495 / (1.0324 - 0.19077 *
Math.log10(waist - neck) + 0.15456 * Math.log10(height)) - 450;
      } else {
```

```
            fatPercent = 495 / (1.29579 - 0.35004 *
Math.log10(waist + hips - neck) + 0.22100 * Math.log10(height))
- 450;
      }

      document.getElementById('fat-result').innerText =
fatPercent.toFixed(1) + "%";
      let fatDesc = "";
      const fp = parseFloat(fatPercent);
      if (gender === "male") {
            if (fp >= 0 && fp < 3) fatDesc = "Небезпечно для
здоров'я";
            else if (fp >= 4 && fp <= 6) fatDesc = "Мінімальний
відсоток жиру";
            else if (fp > 6 && fp <= 12) fatDesc = "Спортивна
фігура";
            else if (fp > 12 && fp <= 15) fatDesc = " Невелика
кількість жиру";
            else if (fp >= 16 && fp <= 19) fatDesc = " Середній
відсоток вмісту жиру";
            else if (fp > 19 && fp <= 25) fatDesc = "Прийнятний
вміст жиру";
            else if (fp > 25 && fp <= 40) fatDesc = "Наявність
зайвої ваги";
            else fatDesc = "Виявлено надлишок жиру в тілі";
      } else {
            if (fp >= 0 && fp <= 10) fatDesc = "Критично низький
відсоток жиру, є ризик припинення менструації";
            else if (fp >= 11 && fp <= 14) fatDesc =
"Мінімальний відсоток жиру для нормального самопочуття";
            else if (fp >= 15 && fp <= 16) fatDesc = "Спортивна
фігура";
            else if (fp >= 17 && fp <= 20) fatDesc = "Невелика
кількість жиру";
            else if (fp >= 21 && fp <= 24) fatDesc = "Відсоток
вмісту жиру прийнятний";
            else if (fp >= 25 && fp <= 30) fatDesc = "Середній
відсоток вмісту жиру";
            else if (fp >= 31 && fp <= 45) fatDesc = " Наявність
зайвої ваги";
            else fatDesc = " Виявлено надлишок жиру в тілі";
      }
      document.getElementById('fat-description').innerText =
fatDesc

      drawFatGauge(fatPercent, gender);
}

// Діаграма результату
function drawFatGauge(fatPercent = NaN, gender) {
      const canvas = document.getElementById("fatGauge");
      if (!canvas) return;
      const ctx = canvas.getContext("2d");
```

```
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      const width = canvas.width;
      const height = 30;
      const xStart = 20;
      const y = 20;
      const maxRange = 50;
      const ranges = gender === "female" ? rangesFemale :
rangesMale;
      // Секції шкали
      ranges.forEach(range => {
            const rangeStartX = xStart + (range.start /
maxRange) * (width - 2 * xStart);
            const rangeEndX = xStart + (range.end / maxRange) *
(width - 2 * xStart);
            const rangeWidth = rangeEndX - rangeStartX;
            ctx.fillStyle = range.color;
            ctx.fillRect(rangeStartX, y, rangeWidth, height);
            ctx.save();
            ctx.translate(rangeStartX + rangeWidth / 2, y +
height + 15);
            ctx.rotate(-Math.PI / 2);
            ctx.fillStyle = "#000";
            ctx.font = "11px Arial";
            ctx.textAlign = "right";
            ctx.textBaseline = "middle";
            ctx.fillText(range.label, 0, 0);
            ctx.restore();
      });
      // Стрілка / індикатор
      if (!isNaN(fatPercent)) {
            const percent = Math.min(fatPercent, maxRange);
            const indicatorX = xStart + (percent / maxRange) *
(width - 2 * xStart);
            ctx.beginPath();
            ctx.moveTo(indicatorX, y - 5);
            ctx.lineTo(indicatorX - 5, y - 15);
            ctx.lineTo(indicatorX + 5, y - 15);
            ctx.closePath();
            ctx.fillStyle = "#000";
            ctx.fill();
      }
}

// Калькулятор калорій
function calculateCalories() {
      const gender = document.querySelector('input[name="gender-
calories"]:checked').value;
      const age = parseFloat(document.getElementById('calories-
age').value);
      const weight =
parseFloat(document.getElementById('calories-weight').value);
      const height =
parseFloat(document.getElementById('calories-height').value);
```

```
      const activity =
parseFloat(document.getElementById('activity-level').value);
      const goal = document.getElementById('goal').value;
      let ree;
      if (gender === "male") {
            ree = 10 * weight + 6.25 * height - 5 * age + 5;
      } else {
            ree = 10 * weight + 6.25 * height - 5 * age - 161;
      }
      let calories = ree * activity;
      if (goal === "lose") {
            calories *= 0.85;
      } else if (goal === "gain") {
            calories *= 1.1;
      }
      document.getElementById('calories-result').innerText =
Math.round(calories);
      // Розподіл макроелементів
      const proteinCalories = 0.3 * calories;
      const fatCalories = 0.3 * calories;
      const carbCalories = 0.4 * calories;
      const proteinGrams = Math.round(proteinCalories / 4);
      const fatGrams = Math.round(fatCalories / 9);
      const carbGrams = Math.round(carbCalories / 4);
      const caloriesDesc = `Б: ${proteinGrams} г, Ж: ${fatGrams}
г, В: ${carbGrams} г`;
      document.getElementById('calories-description').innerText
= caloriesDesc;
      drawMacroChart(proteinGrams, fatGrams, carbGrams);
}

// Діаграма БЖВ
function drawMacroChart(proteinGrams, fatGrams, carbGrams) {
      const ctx =
document.getElementById('macroChart').getContext('2d');
      if (window.macroChartInstance) {
            window.macroChartInstance.destroy();
      }
      window.macroChartInstance = new Chart(ctx, {
            type: 'pie',
            data: {
                  labels: ['Білки', 'Жири', 'Вуглеводи'],
                  datasets: [{
                        data: [proteinGrams * 4, fatGrams * 9,
carbGrams * 4],
                        backgroundColor: ['#4CAF50', '#FF9800',
'#2196F3'],
                        borderWidth: 1
                  }]
            },
            options: {
                  responsive: true,
                  plugins: {
```

```
                        legend: {
                              position: 'bottom',
                              labels: {
                                    font: {
                                          size: 14
                                    }
                              }
                        },
                        tooltip: {
                              callbacks: {
                                    label: function(context) {
                                          const label =
context.label || '';
                                          const value =
context.raw || 0;
                                          return `${label}:
${value.toFixed(0)} ккал`;
                                    }
                              }
                        }
                  }
            }
      });
}

//Видалення рецепту
function deleteRecipe(recipeId) {
    window.location.href = `/recipes/${recipeId}/delete/`;
}

//Зміна кількості порцій страви
function changePortion(delta) {
    let input = document.getElementById('portion-count');
    let current = parseInt(input.value);
    if (current + delta >= 1) input.value = current + delta;
}

//Видалення інгредієнту
function removeIngredient(button) {
    button.parentElement.remove();
}

//Видалення статті
function deleteArticle(articleId) {
    window.location.href = `/blog/${articleId}/delete/`;
}

//Часті питання
document.querySelectorAll('.faq-question').forEach(q => {
    q.addEventListener('click', () => {
        q.parentElement.classList.toggle('active');
    });
});
```

```
//Перемикач секцій профілю, збереження активної секції
document.addEventListener("DOMContentLoaded", function () {
    //Пріоритет: hash -> localStorage -> default
    let hash = window.location.hash ||
localStorage.getItem("activeSection") || "#account-settings";
    //Активація потрібної секції
    activateSection(hash);
    //Відслідковування кліків
    document.querySelectorAll(".nav-link").forEach(link => {
        link.addEventListener("click", function (e) {
            const targetHash = this.getAttribute("href");
            //Збереження обраний хеш у localStorage
            localStorage.setItem("activeSection", targetHash);
            //Активація
            activateSection(targetHash);
        });
    });

    function activateSection(hash) {
        document.querySelectorAll(".profile-
section").forEach(section =>
section.classList.remove("active"));
        const target = document.querySelector(hash);
        if (target) target.classList.add("active");
        document.querySelectorAll(".nav-link").forEach(link =>
link.classList.remove("active"));
        const activeLink = document.querySelector(`.nav-
link[href="${hash}"]`);
        if (activeLink) activeLink.classList.add("active");
    }
});

//Показує/ховає пораду залежно від активної секції
function toggleAdviceSticker() {
    const sticker = document.querySelector('.activity-advice-
sticker');
    if (!sticker) return;
    const isActive = document.getElementById("meal-and-
activity")?.classList.contains('active');
    if (isActive) {
        sticker.style.display = '';
    } else {
        sticker.style.display = 'none';
    }
}

document.addEventListener('DOMContentLoaded', function () {
    toggleAdviceSticker();
    document.querySelectorAll(".nav-link").forEach(link => {
        link.addEventListener("click", function () {
            setTimeout(toggleAdviceSticker, 0);
        });
```

```
    });
});

document.querySelectorAll(".tab-button").forEach(btn => {
    btn.addEventListener("click", () => {
        document.querySelectorAll(".tab-button").forEach(b =>
b.classList.remove("active"));
        document.querySelectorAll(".tab-content").forEach(tc =>
tc.style.display = 'none');
        btn.classList.add("active");
        document.querySelector(btn.getAttribute("data-
tab")).style.display = 'block';
    });
});

// Підказки для поля activity_type
document.addEventListener('DOMContentLoaded', function() {
    const input =
document.querySelector('input[name="activity_type"]');
    const suggestionsBox = document.getElementById('activity-
suggestions');
    if (!input) return;
    input.addEventListener('input', function() {
        const term = input.value;
        if (term.length < 2) {
            suggestionsBox.innerHTML = '';
            suggestionsBox.style.display = 'none';
            return;
        }
        fetch(`/activitytype-
autocomplete/?term=${encodeURIComponent(term)}`)
            .then(response => response.json())
            .then(data => {
                suggestionsBox.innerHTML = '';
                if (data.length) {
                    suggestionsBox.style.display = 'block';
                    data.forEach(item => {
                        const div =
document.createElement('div');
                        div.classList.add('suggestion-item');
                        div.textContent = item.name;
                        div.dataset.name = item.name;
                        div.onclick = function() {
                            input.value = this.dataset.name;
                            suggestionsBox.innerHTML = '';
                            suggestionsBox.style.display =
'none';
                        }
                        suggestionsBox.appendChild(div);
                    });
                } else {
                    suggestionsBox.style.display = 'none';
                }
```

```
            });
    });
    input.addEventListener('blur', function() {
        setTimeout(() => {
            suggestionsBox.innerHTML = '';
            suggestionsBox.style.display = 'none';
        }, 200);
    });
});

// Функція відкриття модального вікна для додавання їжі
document.querySelectorAll('.add-meal-item-btn').forEach(button
=> {
    button.addEventListener('click', function () {
        const form = document.getElementById('mealItemForm');
        form.reset();
        form.dataset.selectedId = '';
        form.dataset.selectedType = '';
        form['item_id'].value = '';
        form.querySelector('[name="meal_type"]').value =
this.dataset.mealType;
        form.querySelector('[name="meal_date"]').value =
document.getElementById("meal-date-input")?.value;
        document.getElementById('autocomplete-input').disabled =
false;
        document.getElementById('mealItemModal').style.display =
'flex';
        document.body.style.overflow = 'hidden';
    });
});

// Автоматичний вибір страви
document.querySelectorAll('.automatic-meal-item-
btn').forEach(btn => {
    btn.addEventListener('click', function() {
        const mealType = btn.dataset.mealType;
        const date = document.getElementById('meal-date-
input').value;
        fetch(`/profile/meals/api/random-
recipe/?meal_type=${mealType}&date=${date}`)
        .then(res => res.json())
        .then(data => {
            const block = document.getElementById('random-
recipe-block-' + mealType);
            if (!block) return;
            if (data.error) {
                block.style.display = "none";
                alert(data.error || "Немає страв за вашими
параметрами");
            } else {
                block.innerHTML = `
                    <p>Випадкова страва: ${data.name}
(${data.calories} ккал)</p>
```

```
                    <button class="add-random-item-btn" data-
id="{{ item.id }}" onclick="addRecipeToMeal('${data.id}',
'${mealType}', '${date}')">
                    </button>
                    <button
onclick="removeRandomRecipeBlock(this)">
                    </button>
                `;
                block.style.display = "block";
            }
        });
    });
});

function removeRandomRecipeBlock(btn) {
    const block = btn.closest('.random-recipe-block');
    block.innerHTML = '';
    block.style.display = "none";
}

// Отримання CSRF токена з cookies
function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        const cookies = document.cookie.split(';');
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
            if (cookie.substring(0, name.length + 1) === (name +
'=')) {
                cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

const csrftoken = getCookie('csrftoken');

// Відправка форми додавання/редагування
document.getElementById('mealItemForm').addEventListener('submit
', function (e) {
    e.preventDefault();
    const form = e.target;
    const data = {
        meal_date: form.meal_date.value,
        meal_type: form.meal_type.value,
        object_id: form.dataset.selectedId,
        object_type: form.dataset.selectedType,
        quantity: form.quantity.value
    };
    const itemId = form.item_id.value;
```

```
    if (itemId) {
        data.item_id = itemId;
    }
    const url = itemId ? '/profile/meals/ajax-edit/' :
'/profile/meals/ajax-add/';
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrftoken
        },
        body: JSON.stringify(data)
    })
    .then(res => res.json())
    .then(data => {
        if (data.success) {
            updateCharts(data.summary);
            const mealItems = document.querySelector(`#meal-
items-${data.meal_type}`);
            if (mealItems) {
                const existingDiv =
document.querySelector(`[data-id="${data.item_id}"]`);
                const div = document.createElement('div');
                div.dataset.id = data.item_id;
                div.innerHTML = `${data.name} — ${data.quantity}
                    <button class="edit-meal-item-btn" data-
id="${data.item_id}">
                    </button>
                    <button class="delete-meal-item-btn" data-
id="${data.item_id}">
                    </button>`;
                if (existingDiv) {
                    existingDiv.replaceWith(div);
                } else {
                    mealItems.appendChild(div);
                }
                bindMealItemButtons();
            }

document.getElementById('mealItemModal').style.display = 'none';
            document.body.style.overflow = '';
            form.reset();
            form.dataset.selectedId = '';
            form.dataset.selectedType = '';
            form.item_id.value = '';
        }
    });
});

// Видалення їжі з прийому
document.querySelectorAll('.delete-meal-item-
btn').forEach(button => {
    button.addEventListener('click', function () {
```

```
        const itemId = this.dataset.id;
        fetch('/profile/meals/ajax-delete/', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken': csrftoken,
            },
            body: JSON.stringify({ item_id: itemId })
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                const el = document.querySelector(`[data-
id="${data.item_id}"]`);
                if (el) {
                    el.remove();
                    updateCharts(data.summary);
                }
            }else {
                console.error('Помилка видалення:', data.error);
                }
        });
    });
});

    document.querySelectorAll('.edit-meal-item-
btn').forEach(button => {
        button.onclick = function () {
            const itemId = this.dataset.id;
            const div = this.closest('div');
            const text = div.textContent.trim();
            const match = text.match(/(.+)\s—\s([\d.]+)\sг/);
            if (!match) return;
            const name = match[1].trim();
            const quantity = match[2].trim();
            const form =
document.getElementById('mealItemForm');
            form.quantity.value = quantity;
            form.item_id.value = itemId;
            form.dataset.selectedId = button.dataset.id;
            form.dataset.selectedType =
div.innerHTML.includes('(Рецепт)') ? 'recipe' : 'product';
            document.getElementById('autocomplete-input').value
= name;

document.getElementById('mealItemModal').style.display = 'flex';
            document.body.style.overflow = 'hidden';
        };
    });
}

// КНОПКА ДОДАТИ АКТИВНІСТЬ
```

```
document.querySelector('.add-activity-btn').onclick = function()
{
    const form = document.getElementById('activityForm');
    form.reset();
    form.activity_id.value = '';
    document.getElementById('activity-modal-title').textContent
= 'Додати активність';
    document.getElementById('activityModal').style.display =
'flex';
    document.body.style.overflow = 'hidden';
};

// КНОПКА РЕДАГУВАТИ
function bindActivityItemButtons() {
    document.querySelectorAll('.edit-activity-
btn').forEach(button => {
        button.onclick = function () {
            const itemId = this.dataset.id;
            fetch(`/profile/activities/item/${itemId}/get/`)
                .then(response => response.json())
                .then(data => {
                    const form =
document.getElementById('activityForm');
                    form.activity_id.value = data.id;
                    form.activity_type.value =
data.activity_type;
                    form.duration.value = data.duration || '';
                    form.manual_calories.value =
data.manual_calories || '';
                    document.getElementById('activity-modal-
title').textContent = 'Редагувати активність';

document.getElementById('activityModal').style.display = 'flex';
                    document.body.style.overflow = 'hidden';
                });
        };
    });

    document.querySelectorAll('.delete-activity-
btn').forEach(button => {
        button.onclick = function () {
            const itemId = this.dataset.id;
            fetch('/profile/activities/ajax-delete/', {
                method: 'POST',
                headers: {'Content-Type': 'application/json',
'X-CSRFToken': csrftoken},
                body: JSON.stringify({ id: itemId })
            })
            .then(res => res.json())
            .then(data => {
                if (data.success) {
                    const el =
document.querySelector(`.activity-item[data-id="${itemId}"]`);
```

```
                    if (el) el.remove();
                    updateCharts(data.summary);
                }
            });
        };
    });
}

bindActivityItemButtons();

// ВІДПРАВКА ФОРМИ ДОДАВАННЯ/РЕДАГУВАННЯ
document.getElementById('activityForm').addEventListener('submit
', function (e) {
    e.preventDefault();
    const form = e.target;
    const data = {
        id: form.activity_id.value,
        activity_type: form.activity_type.value,
        duration: form.duration.value,
        manual_calories: form.manual_calories.value,
        date: form.date.value
    };
    const url = data.id ? '/profile/activities/ajax-edit/' :
'/profile/activities/ajax-add/';
    fetch(url, {
        method: 'POST',
        headers: {'Content-Type': 'application/json', 'X-
CSRFToken': csrftoken},
        body: JSON.stringify(data)
    })
    .then(res => res.json())
    .then(data => {
        if (data.success) {
            const items = document.getElementById('activity-
items');
            let el = document.querySelector(`.activity-
item[data-id="${data.id}"]`);
            let html = `<div class="activity-item" data-
id="${data.id}">
                <span class="activity-
title">${data.activity_type}${data.manual_calories ? ' (вручну)'
: ''}</span>
                <span class="activity-info">${
                data.manual_calories
                    ? data.manual_calories + ' ккал' +
(data.duration ? ', ' + data.duration + ' хв' : '')
                    : data.calories_burned + ' ккал, ' +
data.duration + ' хв'
                }</span>
                <button class="edit-activity-btn" data-
id="${data.id}">
                </button>
```

```
                <button class="delete-activity-btn" data-
id="${data.id}">
                </button>
            </div>`;
            if (el) {
                el.outerHTML = html; // оновлення
            } else {
                items.insertAdjacentHTML('beforeend', html); //
додавання
            }
            updateCharts(data.summary);
            bindActivityItemButtons();

document.getElementById('activityModal').style.display = 'none';
            document.body.style.overflow = '';
            form.reset();
        }
    });
});

// запуск при завантаженні
document.addEventListener('DOMContentLoaded',
bindMealItemButtons);

function updateCharts(summary) {
    if (!summary) return;
    if (!caloriesChart || !macrosChart || !activityBarChart) {
        initCharts(summary);
    } else {
        // Калорії
        caloriesChart.data.datasets[0].data = [
            summary.calories_consumed,
            summary.calories_burned,
            Math.max(summary.calories_target -
summary.calories_consumed + summary.calories_burned, 0)
        ];
        caloriesChart.update();
        // Макроелементи
        macrosChart.data.datasets[0].data = [summary.protein,
summary.fat, summary.carbs];
        macrosChart.update();
        // Активність
        activityBarChart.data.datasets[0].data = [
            summary.activity_minutes,
            summary.activity_minutes_target
        ];
        activityBarChart.data.datasets[0].backgroundColor = [
            summary.activity_minutes >=
summary.activity_minutes_target ? '#4caf50' : '#36A2EB',
            '#FFC107'
        ];
        activityBarChart.options.scales.y.suggestedMax =
Math.max(
```

```
            summary.activity_minutes,
            summary.activity_minutes_target
        ) + 30;
        activityBarChart.update();
        }
}

//Статистика в кабінеті
document.addEventListener("DOMContentLoaded", function () {
    //Переключення вкладок «Статистика та аналітика»
    const tabButtons =
Array.from(document.querySelectorAll('.stat-tabs button[data-
tab]'));
    const statPanels = document.querySelectorAll('.stat-panel');
     const indicator = document.querySelector('.stat-tabs-
indicator');
     const tabContainer = document.querySelector('.stat-tabs');

//Індикатор під активну кнопку
function moveIndicator(activeBtn) {
    const left = activeBtn.offsetLeft;
    const width = activeBtn.offsetWidth;
    indicator.style.left = `${left}px`;
    indicator.style.width = `${width}px`;
}

//Показати/сховати панель діаграми
function showPanel(keyTab) {
    statPanels.forEach(panel => {
    panel.classList.toggle('d-none', panel.dataset.content !==
keyTab);
    });
}
```

Файл «profile.html»

```
<div class="container profile-container">
  <div class="profile-wrapper">
    <aside class="profile-sidebar">
      <ul class="profile-nav">
        {% if is_admin %}
        <li><a href="#users-management" class="nav-link
active">Користувачі</a></li>
        <li><a href="#products-management" class="nav-
link">Продукти</a></li>
        <li><a href="#activities-management" class="nav-
link">Типи активності</a></li>
        {% elif is_editor %}
        <li><a href="#products-management" class="nav-link
active">Продукти</a></li>
        <li><a href="#activities-management" class="nav-
link">Типи активності</a></li>
        {% else %}
```

```
        <li><a href="#meal-and-activity" class="nav-link
active">Історія раціону та активності</a></li>
        <li><a href="#statistics-and-analytics" class="nav-
link">Статистика та аналітика</a></li>
        {% endif %}
        <li><a href="#account-settings" class="nav-
link">Налаштування акаунту</a></li>
        <li><a href="#my-profile" class="nav-link">Мій
профіль</a></li>
      </ul>

        <!-- Рекомендації фіз. активності -->
        {% if not is_admin and not is_editor %}
        <div id="advice-sticker-anchor">
            <div class="activity-advice-sticker"
style="display:none;">
            {{ user.activity_advice }}
            </div>
        </div>
        {% endif %}

        <!—Аналітика статистики -->
        {% if not is_admin and not is_editor %}
        <div class="stats-analysis-container" id="stats-
analysis-container" style="display: none;"></div>
        {% endif %}
    </aside>

    <!-- Основний вміст -->
    <section class="profile-content">
      <div id="account-settings" class="profile-section active">
        <h2 class="profile">Налаштування акаунту</h2>
        <!-- Форма зміни пароля -->
      </div>

        <div id="my-profile" class="profile-section">
            <h2 class="profile">Мій профіль</h2>
            <!-- Форма відображення/зміни даних користувача -->
        </div>

        <div id="meal-and-activity" class="profile-section">
            <h2 class="profile">Відстеження раціону та фізичної
активності</h2>
            <!-- Навігація по датах -->
            <div class="date-navigation d-flex justify-content-
between align-items-center mb-4">
                    <a href="?date={{ prev_date|date:"Y-m-d"
}}#meal-and-activity" class=" meal-active-date-btn">
                        {{ prev_date|date:"d.m.Y" }}
                    </a>
                    <span class="h5 mb-0 meal-active-date-
text">{{ current_date|date:"l d.m.Y" }}</span>
```

```
                    <a href="?date={{ next_date|date:"Y-m-d"
}}#meal-and-activity" class="meal-active-date-btn">
                        {{ next_date|date:"d.m.Y" }}
                    </a>
            </div>

            <div class="tab-content" id="tab-add-meal">
                {% for meal_type, label in meal_type_choices %}
                    {% with meals|get_meal_by_type:meal_type as
meal %}
                        <div class="meal-block" data-meal-
type="{{ meal_type }}">
                            <h4>{{ label }}</h4>

                            <div class="meal-items" id="meal-
items-{{ meal_type }}">
                                {% if meal %}
                                    {% for item in
meal.items.all %}
                                        <div data-id="{{ item.id
}}">
                                            <div class="meal-
item-info">
                                                {% if
item.product %}
                                                    {{
item.product.name }}
                                                {% elif
item.recipe %}
                                                    {{
item.recipe.name }}
                                                {% endif %}
                                                - {{
item.quantity }} г
                                            </div>
                                            <div class="meal-
item-actions">
                                                <button
class="edit-meal-item-btn" data-id="{{ item.id }}"></button>
                                                <button
class="delete-meal-item-btn" data-id="{{ item.id }}"></button>
                                            </div>
                                        </div>
                                    {% endfor %}
                                {% endif %}
                                <div class="random-recipe-block"
id="random-recipe-block-{{ meal_type }}"
style="display:none;"></div>
                            </div>
                            <div class="meal-buttons">
                                <button class="add-meal-item-
btn" data-meal-type="{{ meal_type }}"> +продукт/страву </button>
```

```
                                    <button class="automatic-meal-
item-btn" data-meal-type="{{ meal_type }}"> Вибрати автоматично
</button>
                                </div>
                            </div>
                        {% endwith %}
                    {% endfor %}
                </div>
                <!-- Діаграми КБЖВ -->
                <div id="nutrition-stats">
                    <canvas id="caloriesChart"></canvas>
                    <canvas id="macrosChart"></canvas>
                    {{ summary|json_script:"summary-data" }}
                </div>

                <!-- Активність -->
                <div class="tab-content" id="tab-add-activity">
                    <div class="activity-block">
                        <h4>Фізична активність</h4>
                        <div class="activity-items" id="activity-
items">
                            {% for activity in activities %}
                            <div class="activity-item" data-id="{{
activity.id }}">
                                <div class="activity-item-info">
                                    <span class="activity-title">
                                        {% if activity.activity_type
%}{{ activity.activity_type.name }}{% endif %}
                                        {% if
activity.manual_calories %} (вручну){% endif %}
                                    </span>
                                    <span class="activity-info">
                                        {% if
activity.manual_calories and activity.duration %}
                                            {{
activity.manual_calories }} ккал, {{ activity.duration }} хв
                                        {% elif
activity.manual_calories %}
                                            {{
activity.manual_calories }} ккал
                                        {% elif
activity.calories_burned and activity.duration %}
                                            {{
activity.calories_burned }} ккал, {{ activity.duration }} хв
                                        {% endif %}
                                    </span>
                                </div>
                                <div class="activity-item-actions">
                                    <button class="edit-activity-
btn" data-id="{{ activity.id }}"></button>
                                    <button class="delete-activity-
btn" data-id="{{ activity.id }}"></button>
                                </div>
```

```
                        </div>
                        {% empty %}
                        <div class="text-muted">Дані про
активність відсутні</div>
                        {% endfor %}
                    </div>
                    <button class="add-activity-btn mt-3"> +
Додати активність </button>
                </div>
                <!-- Діаграма фіз. активності -->
                <div id="activityBarChart-container">
                    <canvas id="activityBarChart"></canvas>
                    {{ summary|json_script:"summary-data" }}
                </div>
            </div>
        </div>

        <div id="statistics-and-analytics" class="profile-
section">
            <h2 class="profile">Статистика та аналітика</h2>
            <div class="stat-tabs position-relative">
                <div class="stat-tabs-indicator"></div>
            </div>
            <div class="stat-content"><!-- Діаграми --></div>
        </div>

        <div id="my-recipes" class="profile-section">
            <h2 class="profile">Мої рецепти</h2>
            <!-- Сітка із рецептами, які створив користувач -->
        </div>

        <div id="users-management" class="profile-section">
            <h2 class="profile">Керування користувачами</h2>
            <!-- Кнопка додавання користувача -->
            <button id="add-user-btn" onclick="openModal()">
+Додати користувача</button>
            <!-- Таблиця користувачів -->
            <div>
                <table id="usersTable">
                    <thead>
                    <tr>
                        <th>№</th>
                        <th>Нік</th>
                        <th>Емейл</th>
                        <th>Ім'я</th>
                        <th>Прізвище</th>
                        <th>Роль у системі</th>
                        <th>Дії</th>
                    </tr>
                    </thead>
                    <tbody>
                    {% for user in users %}
                        <tr>
```

```
                        <td>{{ forloop.counter }}</td>
                        <td>{{ user.username }}</td>
                        <td>{{ user.email }}</td>
                        <td>{{ user.first_name }}</td>
                        <td>{{ user.last_name }}</td>
                        <td>{{ user.role }}</td>
                        <td>
                            <div class="dropdown">
                            <button class="dropdown-
toggle">Дії</button>
                            <ul class="dropdown-menu"
type="button">
                                <li><a href="#"
onclick="editUser('{{ user.id }}')">Змінити</a></li>
                                <li><a href="#"
onclick="deleteUser('{{ user.id }}')">Видалити</a></li>
                            </ul>
                            </div>
                        </td>
                        </tr>
                    {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
</div>
```

Додаток Б – Реалізація бізнес-логіки вебзастосунку

Файл «views.py»

```
def login_page(request):
    if request.user.is_authenticated:
        return redirect('home')
    form = LoginForm(request.POST or None)
        if request.method == 'POST':
        if form.is_valid():
            identifier = form.cleaned_data['username']
            password = form.cleaned_data['password']
            try:
                user = User.objects.get(email=identifier)
                username = user.username
            except User.DoesNotExist:
                username = identifier
            user = authenticate(request, username=username,
password=password)
            if user is not None:
                login(request, user)
                return redirect('home')
            else:
                messages.error(request, "Невірне ім'я
користувача або пароль.")
    context['form'] = form
    return render(request, 'login.html', context)

def register_user(request):
    if request.user.is_authenticated:
        return redirect("home")
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('home')
        else:
            context['form'] = form
    else:
        form = UserRegistrationForm()
        context['form'] = form
    context['form'] = form
    return render(request, 'register.html', context)

@login_required
def logout_user(request):
    logout(request)
    return redirect('home')

@login_required
def profile_page(request):
```

```
    #Секція "Налаштування акаунту"
    #Зміна пароля
    if request.method == 'POST'and 'change_password' in
request.POST:
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, request.user)
            messages.success(request, 'Пароль успішно змінено.',
extra_tags="change_password")
            return redirect('profile-page')
        else:
            context['form'] = form
    else:
        form = PasswordChangeForm(request.user)
    context['form'] = form

    #Видалення акаунта
    if request.method == 'POST' and 'delete_account' in
request.POST:
        user = request.user
        logout(request)
        user.delete()
        messages.success(request, "Ваш акаунт було успішно
видалено.", extra_tags="delete_account")
        return redirect('home')

    #Перевірка ролей користувача
    user = request.user
    context['is_admin'] = user.is_superuser or
user.groups.filter(name="Admin").exists()
    context['is_editor'] =
user.groups.filter(name="Editor").exists()

    #Секція "Управління користувачами"
    #Заповнення таблиці користувачів
    if request.user.is_superuser or
request.user.groups.filter(name="Admin").exists():
        users = User.objects.all().order_by('id')
        for user in users:
            groups = list(user.groups.values_list('name',
flat=True))
            if 'Admin' in groups:
                user.role = 'Адмін'
            elif 'Editor' in groups:
                user.role = 'Редактор'
            else:
                user.role = 'Користувач'
        context['users'] = users

    #Додавання нового користувача через модальне вікно
    if request.user.is_superuser or
request.user.groups.filter(name="Admin").exists():
```

```
        if request.method == 'POST' and 'adding_new_user' in
request.POST:
            username = request.POST.get('username')
            email = request.POST.get('email')
            first_name = request.POST.get('first_name', '')
            last_name = request.POST.get('last_name', '')
            password = request.POST.get('password')
            role = request.POST.get('role', 'user')
            errors = {}
            if User.objects.filter(username=username).exists():
                errors['username'] = ['Користувач з таким іменем
вже існує']
            if User.objects.filter(email=email).exists():
                errors['email'] = ['Користувач з таким email вже
існує']
            if len(password) < 6:
                errors['password'] = ['Пароль має містити
щонайменше 6 символів']
            if errors:
                context['user_form_errors'] = errors
                return render(request, 'profile.html', context)
            user = User.objects.create_user(
                username=username,
                email=email,
                password=password,
                first_name=first_name,
                last_name=last_name
            )
            # Призначення ролі
            group_name = {'admin': 'Admin', 'editor':
'Editor'}.get(role)
            if group_name:
                group = Group.objects.get(name=group_name)
                user.groups.add(group)
            messages.success(request, f"Користувача
{user.username} створено успішно.")

    #Секція "Управління продуктами"
    #Заповнення таблиці продуктів
    if request.user.is_superuser or
request.user.groups.filter(name="Admin").exists() or
request.user.groups.filter(name="Editor").exists():
        context['products'] =
Product.objects.all().order_by('id')

    #Секція "Управління типами активностей"
    #Заповнення таблиці типів активностей
    if request.user.is_superuser or
request.user.groups.filter(name="Admin").exists() or
request.user.groups.filter(name="Editor").exists():
        context['activities_types'] =
ActivityType.objects.all().order_by('id')
```

```
    #Секція "Мій профіль"
    if request.method == 'POST' and 'save_profile_changes' in
request.POST:
        user = request.user
        user.first_name = request.POST.get('user_first_name', '')
        user.last_name = request.POST.get('user_last_name', '')
        user.username = request.POST.get('user_username', '')
        user.email = request.POST.get('user_email', '')
        if not user.is_superuser or
request.user.groups.filter(name__in=["Admin",
"Editor"]).exists():
            user.age = request.POST.get('user_age') or None
            user.height = request.POST.get('user_height') or None
            user.weight = request.POST.get('user_weight') or None
            goal = request.POST.get('user_goal')
            user.goal = goal if goal else None
            gender = request.POST.get('user_gender')
            user.gender = gender if gender else None
            activity = request.POST.get('user_activity')
            user.activity_level = activity if activity else None
            user.calories_intake =
request.POST.get('user_calories_intake') or None
            user.target_weight =
request.POST.get('user_target_weight') or None
            allergens = request.POST.getlist('user_allergic_to')
            user.allergic_to = allergens if allergens else []
        user.save()
        messages.success(request, 'Профіль оновлено успішно.',
extra_tags="save_profile_changes")
        return redirect('profile-page')

    #Секція "Раціон та активність"
    #Прийоми їжі
    user = request.user
    # Отримання дати з параметра GET або сьогоднішньої
    date_str = request.GET.get('date')
    if date_str:
        try:
            current_date = datetime.strptime(date_str, '%Y-%m-
%d').date()
        except ValueError:
            current_date = date.today()
    else:
        current_date = date.today()
    prev_date = current_date - timedelta(days=1)
    next_date = current_date + timedelta(days=1)
    meals = Meal.objects.filter(user=user,
date=current_date).prefetch_related('items__product',
'items__recipe')
    activities = Activity.objects.filter(user=user,
date=current_date)
    summary = calculate_summary(meals, activities, user)
    context.update({
```

```
        'meals': meals,
        'activities': activities,
        'meal_type_choices': MEAL_TYPE_CHOICES,
        'summary': summary,
        'current_date': current_date,
        'prev_date': prev_date,
        'next_date': next_date,
    })
    #Активності
    activity_form = ActivityForm(initial={'date': current_date})
    activity = None
    if request.method == "POST" and 'add_activity' in
request.POST:
        activity_form = ActivityForm(request.POST)
        if activity_form.is_valid():
            cd = activity_form.cleaned_data
            activity = Activity(user=request.user,
date=cd['date'])
            if cd['enter_manual']:
                activity.activity_type = None
                activity.duration = None
                activity.calories_burned = cd['manual_calories']
            else:
                # autocomplete: знайти ActivityType
                try:
                    act_type =
ActivityType.objects.get(name=cd['activity_type'])
                except ActivityType.DoesNotExist:
                    act_type =
ActivityType.objects.create(name=cd['activity_type'],
met_value=1)
                activity.activity_type = act_type
                activity.duration = cd['duration']
                # calories_burned по моделі (calculate_calories)
            activity.save()
            return redirect(request.path +
f'?date={cd["date"]}#meal-and-activity')
    context.update({
        'activity_form': activity_form,
        'activity': activity,
        'current_date': current_date,
    })

    #Секція Мої рецепти
    #Заповнення сітки рецептів
    context['recipes'] =
Recipe.objects.filter(user=request.user).order_by('id')
    return render(request, 'profile.html', context)

def autocomplete_suggestions(request):
    query = request.GET.get('term', '')
    suggestions = []
    if query:
```

```
        # Продукти
        products =
Product.objects.filter(name__icontains=query)[:5]
        suggestions += [
            {'type': 'product', 'id': p.id, 'name': p.name} for
p in products
        ]
        #Рецепти
        recipes =
Recipe.objects.filter(name__icontains=query)[:5]
        suggestions += [
            {'type': 'recipe', 'id': r.id, 'name': r.name} for r
in recipes
        ]
    return JsonResponse(suggestions, safe=False)

@login_required
def ajax_add_meal_item(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        meal_type = data.get('meal_type')
        object_id = data.get('object_id')
        object_type = data.get('object_type')
        quantity = float(data.get('quantity', 0))
        date_str = data.get('meal_date')
        try:
            meal_date = datetime.strptime(date_str, '%Y-%m-
%d').date()
        except (ValueError, TypeError):
            meal_date = date.today()
        #отримуємо або створюємо Meal для поточного дня і типу
        meal, _ = Meal.objects.get_or_create(user=request.user,
date=meal_date, meal_type=meal_type)

        if object_type == 'product':
            product = Product.objects.get(id=object_id)
            item = MealItem.objects.create(meal=meal,
product=product, quantity=quantity)
            name = product.name
        elif object_type == 'recipe':
            recipe = Recipe.objects.get(id=object_id)
            item = MealItem.objects.create(meal=meal,
recipe=recipe, quantity=quantity)
            name = recipe.name
        else:
            return JsonResponse({'success': False, 'error':
'Invalid object type'})
        meals = Meal.objects.filter(user=request.user,
date=meal_date).prefetch_related('items__product',
'items__recipe')
        activities = Activity.objects.filter(user=request.user,
date=meal_date)
```

```
        summary = calculate_summary(meals, activities,
request.user)
        return JsonResponse({
            'success': True,
            'item_id': item.id,
            'name': name,
            'quantity': quantity,
            'summary': summary,
            'meal_type': meal_type
        })
    return JsonResponse({'success': False, 'error': 'Invalid
request'})

@login_required
def ajax_edit_meal_item(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        item_id = data.get('item_id')
        quantity = float(data.get('quantity', 0))
        try:
            item = MealItem.objects.get(id=item_id,
meal__user=request.user)
            item.quantity = quantity
            item.save()
            #Визначаємо назву продукту або страви
            if item.product:
                name = item.product.name
            elif item.recipe:
                name = item.recipe.name
            else:
                name = "Невідомо"
            meal_date = item.meal.date
            meals = Meal.objects.filter(user=request.user,
date=meal_date).prefetch_related('items__product',
'items__recipe')
            activities =
Activity.objects.filter(user=request.user, date=meal_date)
            summary = calculate_summary(meals, activities,
request.user)
            return JsonResponse({'success': True, 'item_id':
item.id, 'name': name, 'quantity': item.quantity, 'summary':
summary, 'meal_type': item.meal.meal_type})
        except MealItem.DoesNotExist:
            return JsonResponse({'success': False, 'error':
'Item not found'})

@login_required
def ajax_delete_meal_item(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            item_id = data.get('item_id')
            if not item_id:
```

```
                return JsonResponse({'success': False, 'error':
'Не передано item_id'})
            item = MealItem.objects.get(id=item_id,
meal__user=request.user)
            meal = item.meal  #Зберігаємо meal перед видаленням
            item.delete()
            # Після видалення – перерахунок
            meals = Meal.objects.filter(user=request.user,
date=meal.date).prefetch_related('items__product',
'items__recipe')
            activities =
Activity.objects.filter(user=request.user, date=meal.date)
            summary = calculate_summary(meals, activities,
request.user)
            return JsonResponse({
                'success': True,
                'item_id': item_id,
                'summary': summary,
                'meal_type': meal.meal_type
            })
        except MealItem.DoesNotExist:
            return JsonResponse({'success': False, 'error':
'Елемент не знайдено'})

@login_required
def get_meal_item(request, item_id):
    try:
        item = MealItem.objects.select_related('product',
'recipe', 'meal').get(id=item_id, meal__user=request.user)
        return JsonResponse({
            'success': True,
            'item_id': item.id,
            'meal_type': item.meal.meal_type,
            'meal_date': item.meal.date.isoformat(),
            'quantity': item.quantity,
            'name': item.product.name if item.product else
item.recipe.name,
            'object_type': 'product' if item.product else
'recipe',
            'object_id': item.product.id if item.product else
item.recipe.id
        })
    except MealItem.DoesNotExist:
        return JsonResponse({'success': False, 'error': 'Item
not found'})

@login_required
def api_random_recipe(request):
    meal_type = request.GET.get('meal_type')
    recipe = generate_user_meal(request.user, meal_type)
    if recipe:
        return JsonResponse({'id': recipe.id, 'name':
recipe.name, 'calories': recipe.total_calories})
```

```
    return JsonResponse({'error': "Немає страв за вашими
параметрами"}, status=404)

def generate_user_meal(user, meal_type):
    #Список алергенів користувача
    user_allergens = user.allergic_to or []
    #Діапазон прийнятної калорійності страви
    calories_per_meal = (user.calories_intake or 1800) // 4
    calorie_min = calories_per_meal * 0.8
    calorie_max = calories_per_meal * 1.2
    #Вибір рецептів для цього типу прийому їжі
    qs = Recipe.objects.filter(
        meal_type=meal_type,
        #total_calories__gte=calorie_min,
        total_calories__lte=calorie_max,)
    #Відкинути ті, де є алергени користувача
    def is_safe(recipe):
        for item in
recipe.recipe_items.select_related('product'):
            if hasattr(item.product, 'is_allergic') and
item.product.is_allergic:
                # продукт є алергеном
                if item.product.name.lower() in user_allergens:
                    return False
        return True

    safe_recipes = [r for r in qs if is_safe(r)]
    if not safe_recipes:
        return None  # Немає підходящих страв
    return random.choice(safe_recipes)

def calculate_summary(meals, activities, user):
    summary = {
        'calories_consumed': 0,
        'calories_burned': 0,
        'protein': 0,
        'fat': 0,
        'carbs': 0
    }
    for meal in meals:
        for item in meal.items.all():
            if item.product:
                summary['calories_consumed'] +=
item.product.calories * item.quantity / 100
                summary['protein'] += item.product.protein *
item.quantity / 100
                summary['fat'] += item.product.fat *
item.quantity / 100
                summary['carbs'] += item.product.carbs *
item.quantity / 100
            elif item.recipe:
                summary['calories_consumed'] +=
item.recipe.total_calories * item.quantity / 100
```

```
                summary['protein'] += item.recipe.total_protein
* item.quantity / 100
                summary['fat'] += item.recipe.total_fat *
item.quantity / 100
                summary['carbs'] += item.recipe.total_carbs *
item.quantity / 100
    for activity in activities:
        summary['calories_burned']+=activity.calories_burned or 0
    calories_intake = user.calories_intake if
user.calories_intake else 2000
    percent = {'protein': 0.3, 'fat': 0.3, 'carbs': 0.4}
    calories_per_g = {'protein': 4, 'fat': 9, 'carbs': 4}
    macros_target = {
        'protein': round((percent['protein'] * calories_intake)
/ calories_per_g['protein'], 1),
        'fat': round((percent['fat'] * calories_intake) /
calories_per_g['fat'], 1),
        'carbs': round((percent['carbs'] * calories_intake) /
calories_per_g['carbs'], 1),
    }
    today = date.today()
    week_ago = today - timedelta(days=6)
    weekly_activities =
user.activities.filter(date__gte=week_ago, date__lte=today)
    activity_minutes = sum(act.duration or 0 for act in
weekly_activities)
    activity_minutes_target = 150

    return {
        'calories_consumed': round(summary['calories_consumed'],
2),
        'calories_burned': round(summary['calories_burned'], 2),
        'protein': round(summary['protein'], 2),
        'fat': round(summary['fat'], 2),
        'carbs': round(summary['carbs'], 2),
        'calories_target': round(calories_intake, 2),
        'macros_target': macros_target,
        'activity_minutes': round(activity_minutes, 1),
        'activity_minutes_target': activity_minutes_target,
    }

@login_required
def activitytype_autocomplete(request):
    if request.method == 'GET':
        q = request.GET.get('term', '')
        results = []
        if q:
            results =
list(ActivityType.objects.filter(name__icontains=q)[:10].values(
'id', 'name'))
        return JsonResponse(results, safe=False)

@login_required
```

```
def get_activity_item(request, item_id):
    try:
        activity =
Activity.objects.select_related('activity_type').get(id=item_id,
user=request.user)
        return JsonResponse({
            'success': True,
            'id': activity.id,
            'activity_type': activity.activity_type.name if
activity.activity_type else '',
            'duration': activity.duration or '',
            'manual_calories': activity.manual_calories or '',
            'calories_burned': activity.calories_burned or '',
            'date': activity.date.isoformat(),
        })
    except Activity.DoesNotExist:
        return JsonResponse({'success': False})

@login_required
def ajax_add_activity(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        activity_type_name = data.get('activity_type')
        duration = float(data.get('duration') or 0)
        manual_calories = float(data.get('manual_calories') or 0)
        date_val = datetime.strptime(data.get('date'), '%Y-%m-
%d').date()

        if manual_calories:
            activity = Activity.objects.create(
                user=request.user,
                activity_type=None,
                duration=duration if duration else None,
                manual_calories=manual_calories,
                date=date_val)
        else:
            if not activity_type_name or not duration:
                return JsonResponse({'success': False, 'error':
'Введіть тип активності та тривалість або калорії вручну.'})
            activity_type, _ =
ActivityType.objects.get_or_create(name=activity_type_name)
            activity = Activity.objects.create(
                user=request.user,
                activity_type=activity_type,
                duration=duration,
                manual_calories=None,
                date=date_val)
        summary = calculate_summary(
            Meal.objects.filter(user=request.user,
date=date_val),
            Activity.objects.filter(user=request.user,
date=date_val),
            request.user)
```

```
        return JsonResponse({
            'success': True,
            'id': activity.id,
            'activity_type': activity.activity_type.name if
activity.activity_type else '',
            'duration': activity.duration or '',
            'manual_calories': activity.manual_calories or '',
            'calories_burned': activity.calories_burned or '',
            'summary': summary,
        })

@login_required
def ajax_edit_activity(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        activity_id = data.get('id')
        activity_type_name = data.get('activity_type')
        duration = float(data.get('duration') or 0)
        manual_calories = float(data.get('manual_calories') or 0)
        activity = Activity.objects.get(id=activity_id,
user=request.user)
        if manual_calories:
            activity.activity_type = None
            activity.duration = duration if duration else None
            activity.manual_calories = manual_calories
        else:
            if not activity_type_name or not duration:
                return JsonResponse({'success': False, 'error':
'Введіть тип активності та тривалість або калорії вручну.'})
            activity_type, _ =
ActivityType.objects.get_or_create(name=activity_type_name)
            activity.activity_type = activity_type
            activity.duration = duration
            activity.manual_calories = None
        activity.save()
        summary = calculate_summary(
            Meal.objects.filter(user=request.user,
date=activity.date),
            Activity.objects.filter(user=request.user,
date=activity.date),
            request.user
        )
        return JsonResponse({
            'success': True,
            'id': activity.id,
            'activity_type': activity.activity_type.name if
activity.activity_type else '',
            'duration': activity.duration or '',
            'manual_calories': activity.manual_calories or '',
            'calories_burned': activity.calories_burned or '',
            'summary': summary,
        })
```

```
@login_required
def ajax_delete_activity(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        activity_id = data.get('id')
        try:
            activity = Activity.objects.get(id=activity_id,
user=request.user)
            date_val = activity.date
            activity.delete()
            summary = calculate_summary(
                Meal.objects.filter(user=request.user,
date=date_val),
                Activity.objects.filter(user=request.user,
date=date_val),
                request.user
            )
            return JsonResponse({'success': True, 'id':
activity_id, 'summary': summary})
        except Activity.DoesNotExist:
            return JsonResponse({'success': False})
@login_required
def stats_data(request):
    user = request.user
    type_ = request.GET.get('type')
    period = request.GET.get('period')
    if period == 'custom':
        try:
            start =
datetime.strptime(request.GET.get('start',''), '%Y-%m-
%d').date()
            end   = datetime.strptime(request.GET.get('end',''),
'%Y-%m-%d').date()
        except:
            end = date.today()
            start = end - timedelta(days=6)
    else:
        try:
            days = int(period)
        except:
            days = 7
        end = date.today()
        start = end - timedelta(days=days-1)
    total_days = (end - start).days + 1
    if total_days <= 0:
        return JsonResponse({'error': 'Діапазон дат
некоректний'}, status=400)
    if total_days <= 30:
        bucket_size = 1
    elif total_days <= 180:
        bucket_size = 7
    else:
        bucket_size = 14
```

```
    buckets = []
    labels = []
    cur = start
    while cur <= end:
        bucket_start = cur
        bucket_end = cur + timedelta(days=bucket_size - 1)
        if bucket_end > end:
            bucket_end = end
        buckets.append((bucket_start, bucket_end))
        if bucket_size == 1:
            labels.append(bucket_start.strftime('%d.%m'))
        else:
            labels.append(f"{bucket_start.strftime('%d.%m')}-
{bucket_end.strftime('%d.%m')}")
        cur = bucket_end + timedelta(days=1)
    num_buckets = len(buckets)
    age = user.age or 30
    if age < 18:
        daily_activity_target = 60
    else:
        daily_activity_target = 20
    calories_intake = user.calories_intake or
user.calculate_calorie_needs() or 2000
    perc = {'protein': 0.3, 'fat': 0.3, 'carbs': 0.4}
    cpg = {'protein': 4, 'fat': 9, 'carbs': 4}
    target_daily = {
        'calories': round(calories_intake, 2),
        'protein':  round((perc['protein'] * calories_intake) /
cpg['protein'], 2),
        'fat':      round((perc['fat']     * calories_intake) /
cpg['fat'],     2),
        'carbs':    round((perc['carbs']   * calories_intake) /
cpg['carbs'],   2),
    }
    if type_ == 'activity':
        activities = Activity.objects.filter(user=user,
date__range=(start, end))
        user_arr = [0.0] * num_buckets
        target_arr = [0.0] * num_buckets
        for act in activities:
            d = act.date
            idx = (d - start).days // bucket_size
            if 0 <= idx < num_buckets:
                user_arr[idx] += (act.duration or 0.0)
        for i, (bstart, bend) in enumerate(buckets):
            length = (bend - bstart).days + 1
            target_arr[i] = daily_activity_target * length
        user_arr = [round(x, 2) for x in user_arr]
        target_arr = [round(x, 2) for x in target_arr]
        #Аналітика для активності
        avg_user = sum(user_arr) / len(user_arr) if num_buckets
else 0
```

```
        avg_target = sum(target_arr) / len(target_arr) if
num_buckets else 0
        return JsonResponse({
            'labels': labels,
            'user':   user_arr,
            'target': target_arr,
            'analysis': analysis
        })
    bucket_sum = {
        'calories': [0.0]*num_buckets,
        'protein':  [0.0]*num_buckets,
        'fat':      [0.0]*num_buckets,
        'carbs':    [0.0]*num_buckets,
        'burned':   [0.0]*num_buckets,
    }
    acts = Activity.objects.filter(user=user,
date__range=(start, end))
    for act in acts:
        d = act.date
        idx = (d - start).days // bucket_size
        if 0 <= idx < num_buckets:
            bucket_sum['burned'][idx] += (act.calories_burned or
0.0)
    meals_qs = Meal.objects.filter(user=user,
date__range=(start, end)).prefetch_related('items__product',
'items__recipe')
    for meal in meals_qs:
        d = meal.date
        idx = (d - start).days // bucket_size
        if not (0 <= idx < num_buckets):
            continue
        for item in meal.items.all():
            qty = item.quantity or 0.0
            if item.product:
                p = item.product
                bucket_sum['calories'][idx] += p.calories*qty/100
                bucket_sum['protein'][idx] += p.protein*qty/100
                bucket_sum['fat'][idx] += p.fat * qty / 100
                bucket_sum['carbs'][idx] += p.carbs * qty / 100
            elif item.recipe:
                r = item.recipe
                bucket_sum['calories'][idx] += r.total_calories
* qty / 100
                bucket_sum['protein'][idx] += r.total_protein *
qty / 100
                bucket_sum['fat'][idx] += r.total_fat * qty /100
                bucket_sum['carbs'][idx] += r.total_carbs*qty/100
    for key in bucket_sum:
        bucket_sum[key] = [round(x,2) for x in bucket_sum[key]]
    target_arr = []
    for i, (bstart, bend) in enumerate(buckets):
        length = (bend - bstart).days + 1
        if type_ == 'calories':
```

```
            target_arr.append(round(target_daily['calories'] *
length, 2))
        else:
            target_arr.append(round(target_daily[type_] *
length, 2))
    if type_ == 'calories':
        consumed = bucket_sum['calories']
        burned   = bucket_sum['burned']
        avg_cons = sum(consumed) / num_buckets if num_buckets
else 0
        avg_burn = sum(burned) / num_buckets if num_buckets else
0
        avg_targ = sum(target_arr) / num_buckets if num_buckets
else 0
        return JsonResponse({
            'labels':   labels,
            'consumed': consumed,
            'target':   targ,
            'analysis': analysis
        })
    return JsonResponse({'error': 'Invalid type parameter'},
status=400)

def diet_page(request):
    context['can_edit'] = request.user.is_authenticated and (
        request.user.is_superuser or
        request.user.groups.filter(name__in=["Admin",
"Editor"]).exists())
    context['plans'] = DietPlan.objects.all().order_by('id')
    return render(request, 'diet_plans.html', context)

@login_required
def add_plan(request):
    if request.method == 'POST':
        return save_plan(request)
    return render(request, 'manage_plan.html', context)

@login_required
def edit_plan(request, plan_id):
    plan = get_object_or_404(DietPlan, id=plan_id)
    if request.method == 'POST':
        return save_plan(request, plan)
    context.update({
        'plan': plan,
    })
    return render(request, 'manage_plan.html', context)

@login_required
def delete_plan(request, plan_id):
    plan = get_object_or_404(DietPlan, id=plan_id,
user=request.user)
    plan.delete()
    return redirect('diet-plans-page')
```

```
def save_plan(request, plan=None):
    name = request.POST['name']
    description = request.POST['description']
    recommendations = request.POST['recommendations']
    not_recommended = request.POST['not_recommended']
    benefits = request.POST['benefits']
    risks = request.POST['risks']
    icon = request.FILES.get('icon')
    if not plan:
        plan = DietPlan(user=request.user)
    plan.name = name
    plan.description = description
    plan.recommendations = recommendations
    plan.not_recommended = not_recommended
    plan.benefits = benefits
    plan.risks = risks
    if icon:
        plan.icon = icon
    plan.save()
    return redirect('diet-plans-page')

@login_required
def set_diet_plan(request):
    if request.method == 'POST':
        diet_id = request.POST.get('diet_id')
        diet = get_object_or_404(DietPlan, pk=diet_id)
        request.user.selected_diet = diet
        request.user.save()
        return JsonResponse({'status': 'ok'})

@login_required
def generate_daily_meal_plan(user, date):
    recipes = get_appropriate_recipes(user)
    meal_types = ['breakfast', 'lunch', 'dinner', 'snack']
    meals = []
    for meal_type in meal_types:
        recipe = select_best_recipe(recipes, meal_type, user)
        if recipe:
            meal = Meal.objects.create(user=user, date=date,
meal_type=meal_type)
            MealItem.objects.create(meal=meal, recipe=recipe,
quantity=recipe.servings or 1)
            meals.append(meal)
    return meals

def calc_page(request):
    context['calculator'] = request.GET.get('calculator', 'bmi')
    return render(request, 'calculators.html', context)

def recipes_page(request):
    context['can_edit'] = request.user.is_authenticated and (
        request.user.is_superuser or
```

```
        request.user.groups.filter(name__in=["Admin",
"Editor"]).exists())
    context['recipes'] = Recipe.objects.all().order_by('id')
    #Фільтрація, пошук
    recipes = Recipe.objects.all().select_related('user')
    type = request.GET.get('type')
    search = request.GET.get('search')
    if type:
        recipes = recipes.filter(meal_type=type)
    if search:
        recipes = recipes.filter(
            Q(name__icontains=search) |
            Q(recipe_items__product__name__icontains=search)
        ).distinct()
    context.update({
        'recipes': recipes,
        'types': MEAL_TYPE_CHOICES
    })
    return render(request, 'recipes.html', context)

def recipe_detail(request, recipe_id):
    recipe = get_object_or_404(Recipe, id=recipe_id)
    ingredients = RecipeItem.objects.filter(recipe=recipe)
    steps =
RecipeStep.objects.filter(recipe=recipe).order_by('step_number')
    context.update({
        'recipe': recipe,
        'ingredients': ingredients,
        'steps': steps,
    })
    return render(request, 'recipe.html', context)

def autocomplete_products(request):
    q = request.GET.get('q', '')
    results =
Product.objects.filter(name__icontains=q).values_list('name',
flat=True)[:10]
    return JsonResponse(list(results), safe=False)

@login_required
def add_recipe(request):
    if request.method == 'POST':
        return save_recipe(request)
    return render(request, 'manage_recipe.html', context)

@login_required
def edit_recipe(request, recipe_id):
    recipe = get_object_or_404(Recipe, id=recipe_id,
user=request.user)
    if request.method == 'POST':
        return save_recipe(request, recipe)
    ingredients = RecipeItem.objects.filter(recipe=recipe)
```

```
    steps =
RecipeStep.objects.filter(recipe=recipe).order_by('step_number')
    context.update({
        'recipe': recipe,
        'ingredients': ingredients,
        'steps_text': "\n".join(s.instruction for s in steps)
    })
    return render(request, 'manage_recipe.html', context)

@login_required
def delete_recipe(request, recipe_id):
    recipe = get_object_or_404(Recipe, id=recipe_id,
user=request.user)
    recipe.delete()
    return redirect('recipes-page')

def save_recipe(request, recipe=None):
    name = request.POST['name']
    description = request.POST['description']
    meal_type = request.POST['meal_type']
    category = request.POST['category']
    prep_time = request.POST['prep_time']
    servings = request.POST['servings']
    steps = request.POST.get('steps', '')
    image = request.FILES.get('image')
    if not recipe:
        recipe = Recipe(user=request.user)
    recipe.name = name
    recipe.description = description
    recipe.meal_type = meal_type
    recipe.category = category
    recipe.prep_time = prep_time
    recipe.servings = servings
    if image:
        recipe.photo = image
    recipe.save()
    ingredient_names = request.POST.getlist('ingredient_name[]')
    ingredient_quantities =
request.POST.getlist('ingredient_quantity[]')
    existing_items = list(recipe.recipe_items.all())
    for i, (name, qty) in enumerate(zip(ingredient_names,
ingredient_quantities)):
        try:
            product =
Product.objects.get(name__iexact=name.strip())
            if i < len(existing_items):
                item = existing_items[i]
                item.product = product
                item.quantity = float(qty)
                item.save()
            else:
                RecipeItem.objects.create(recipe=recipe,
product=product, quantity=float(qty))
```

```
            except Product.DoesNotExist:
                continue
        if len(existing_items) > len(ingredient_names):
            for extra_item in
existing_items[len(ingredient_names):]:
                extra_item.delete()
        new_steps = steps.strip().split('\n')
        existing_steps = list(recipe.steps.all())
        for i, step_text in enumerate(new_steps):
            if i < len(existing_steps):
                step = existing_steps[i]
                step.step_number = i + 1
                step.instruction = step_text.strip()
                step.save()
            else:
                RecipeStep.objects.create(recipe=recipe,
step_number=i + 1, instruction=step_text.strip())
        if len(existing_steps) > len(new_steps):
            for extra_step in existing_steps[len(new_steps):]:
                extra_step.delete()
        recipe.calculate_nutrition()
        return redirect('recipes-page')

    def blog_page(request):
        context['can_edit'] = request.user.is_authenticated and (
            request.user.is_superuser or
            request.user.groups.filter(name__in=["Admin",
"Editor"]).exists())
        context['articles'] = Article.objects.all().order_by('id')
        #Фільтрація, пошук
        articles = Article.objects.all().select_related('author')

        article_category = request.GET.get('article_category')
        search = request.GET.get('search')
        if article_category:
            articles = articles.filter(category=article_category)
        if search:
            articles = articles.annotate(
                full_name=Concat('author__first_name', V(' '),
'author__last_name')
            ).filter(
                Q(title__icontains=search) |
                Q(author__username__icontains=search) |
                Q(full_name__icontains=search)
            ).distinct()
        context.update({
            'articles': articles,
            'article_categories': Article.ARTICLE_CATEGORY_CHOICES
        })
        return render(request, 'blog.html', context)

    def article_detail(request, article_id):
        article = get_object_or_404(Article, id=article_id)
```

```
    paragraphs = article.content.strip().split('\n')
    intro = paragraphs[0] if paragraphs else ''
    body = paragraphs[1:] if len(paragraphs) > 1 else []
    context.update({
        'article': article,
        'intro': intro,
        'body': body,
    })
    return render(request, 'article.html', context)

@login_required
def add_article(request):
    context = context_data()
    context['page'] = 'article-add'
    context['page_title'] = 'Додавання статті'
    categories = Article.ARTICLE_CATEGORY_CHOICES
    if request.method == 'POST':
        return save_article(request)
    context.update({
        'now': now(),
        'categories': categories,
    })
    return render(request, 'manage_article.html', context)

@login_required
def edit_article(request, article_id):
    article = get_object_or_404(Article, id=article_id,
author=request.user)
    categories = Article.ARTICLE_CATEGORY_CHOICES
    if request.method == 'POST':
        return save_article(request, article)
    context.update({
        'article': article,
        'categories': categories,
    })
    return render(request, 'manage_article.html', context)

@login_required
def delete_article(request, article_id):
    article = get_object_or_404(Article, id=article_id,
author=request.user)
    article.delete()
    return redirect('blog-page')

def save_article(request, article=None):
    if article:
        if 'title' in request.POST:
            article.title = request.POST['title'].strip()
        if 'content' in request.POST:
            article.content = request.POST['content'].strip()
        if 'category' in request.POST:
            article.category = request.POST['category'].strip()
        if 'image' in request.FILES:
```

```
            article.image = request.FILES['image']
    else:
        article = Article(
            author=request.user,
            title=request.POST.get('title', '').strip(),
            content=request.POST.get('content', '').strip(),
            category=request.POST.get('category', '').strip(),
            image=request.FILES.get('image'))
    article.save()
    return redirect('blog-page')
```

Файл «models.py»

```
from .choices import GENDER_CHOICES, ACTIVITY_CHOICES,
GOAL_CHOICES, ALLERGEN_CHOICES, MEAL_TYPE_CHOICES

class DietPlan(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    recommendations = models.TextField(blank=True)
    not_recommended = models.TextField(blank=True)
    benefits = models.TextField(blank=True)
    risks = models.TextField(blank=True)
    icon = models.ImageField(upload_to='diet_icons/',
blank=True, null=True)


class User(AbstractUser):
    age = models.IntegerField(null=True, blank=True)
    weight = models.FloatField(null=True, blank=True)
    height = models.FloatField(null=True, blank=True)
    gender = models.CharField(max_length=20,
choices=GENDER_CHOICES, null=True, blank=True)
    activity_level = models.CharField(max_length=100,
choices=ACTIVITY_CHOICES, null=True, blank=True)
    goal = models.CharField(max_length=150,
choices=GOAL_CHOICES, null=True, blank=True)
    allergic_to = models.JSONField(null=True, blank=True)
    target_weight = models.FloatField(null=True, blank=True)
    calories_intake = models.FloatField(null=True, blank=True)
    selected_diet = models.ForeignKey(DietPlan, null=True,
blank=True, on_delete=models.SET_NULL)

    def calculate_calorie_needs(self):
        if not (self.gender and self.weight and self.height and
self.age and self.activity_level and self.goal):
            return None
        if self.gender == 'F':
            ree = (10 * self.weight) + (6.25 * self.height) - (5
* self.age) - 161
        else:
            ree = (10 * self.weight) + (6.25 * self.height) - (5
* self.age) - 5
```

```
        pa = ACTIVITY_FACTORS.get(self.activity_level, 1.2)
        tdee = ree * pa
        if self.goal == 'lose':
            calories = tdee * 0.85
        elif self.goal == 'gain':
            calories = tdee * 1.10
        else:
            calories = tdee
        return round(calories)

    def activity_advice(self):
        advice = []
        today = date.today()
        age = self.age or 30
        gender = self.gender or 'M'
        weight = self.weight or 70
        height = self.height or 170
        goal = self.goal
        level = self.activity_level
        target_weight = self.target_weight or self.weight
        week_ago = today - timedelta(days=6)
        recent_activities =
self.activities.filter(date__gte=week_ago, date__lte=today)
        week_minutes = sum((act.duration or 0) for act in
recent_activities)
        week_calories = sum((act.calories_burned or 0) for act
in recent_activities)
        if age < 18:
            min_minutes = 420
        else:
            min_minutes = 150


class Product(models.Model):
    name = models.CharField(max_length=255)
    calories = models.FloatField()
    protein = models.FloatField()
    fat = models.FloatField()
    carbs = models.FloatField()
    unit = models.CharField(max_length=50)
    is_allergic = models.BooleanField(default=False)


class Recipe(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)
    photo = models.ImageField(upload_to='recipes/', null=True,
blank=True)
    prep_time = models.PositiveIntegerField(null=True,
blank=True)
    servings = models.PositiveIntegerField(null=True, blank=True)
```

```
    meal_type = models.CharField(max_length=20,
choices=MEAL_TYPE_CHOICES, null=True, blank=True)
    category = models.CharField(max_length=20,
choices=CATEGORY_CHOICES, null=True, blank=True)
    total_calories = models.FloatField(default=0)
    total_protein = models.FloatField(default=0)
    total_fat = models.FloatField(default=0)
    total_carbs = models.FloatField(default=0)

    def calculate_nutrition(self):
        items = self.recipe_items.all()
        self.total_calories = sum((item.product.calories *
item.quantity) / 100 for item in items)
        self.total_protein = sum((item.product.protein *
item.quantity) / 100 for item in items)
        self.total_fat     = sum((item.product.fat *
item.quantity) / 100 for item in items)
        self.total_carbs   = sum((item.product.carbs *
item.quantity) / 100 for item in items)
        self.save()


class RecipeItem(models.Model):
    recipe = models.ForeignKey(Recipe,
related_name='recipe_items', on_delete=models.CASCADE)
    product = models.ForeignKey(Product,
on_delete=models.CASCADE)
    quantity = models.FloatField()

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        self.recipe.calculate_nutrition()


class RecipeStep(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='steps')
    step_number = models.IntegerField()
    instruction = models.TextField()


class Meal(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='meals')
    date = models.DateField()
    meal_type = models.CharField(max_length=50,
choices=MEAL_TYPE_CHOICES, null=True, blank=True)


class MealItem(models.Model):
    meal = models.ForeignKey(Meal, on_delete=models.CASCADE,
related_name='items')
```

```
    product = models.ForeignKey(Product,
on_delete=models.SET_NULL, null=True, blank=True)
    recipe = models.ForeignKey(Recipe,
on_delete=models.SET_NULL, null=True, blank=True)
    quantity = models.FloatField()


class ActivityType(models.Model):
    name = models.CharField(max_length=500)
    met_value = models.FloatField()
    description = models.TextField(blank=True, null=True)


class Activity(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='activities')
    activity_type = models.ForeignKey(ActivityType,
on_delete=models.RESTRICT, null=True, blank=True)
    duration = models.FloatField(help_text="Тривалість у
хвилинах", null=True, blank=True)
    calories_burned = models.FloatField(blank=True, null=True)
    manual_calories = models.FloatField(blank=True, null=True)
    date = models.DateField()

    def calculate_calories(self):
        if self.manual_calories:
            return self.manual_calories
        if self.user.weight and self.activity_type and
self.duration:
            return round((self.activity_type.met_value *
self.user.weight * self.duration) / 60, 2)
        return 0

    def save(self, *args, **kwargs):
        self.calories_burned = self.calculate_calories()
        super().save(*args, **kwargs)


class Article(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    category = models.CharField(max_length=100,
choices=ARTICLE_CATEGORY_CHOICES)
    created_at = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='articles')
    image = models.ImageField(upload_to='articles/', null=True,
blank=True)
```