МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

завідувач кафедри ІПЗ, канд. техн. наук, доцент

_____ Є. О. Давиденко

підпис

« __ » _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

«ВЕБЗАСТОСУНОК ОБЛІКУ ТА КОНТРОЛЮ ВЖИТИХ КАЛОРІЙ НА ОСНОВІ ОБРАНОГО ПЛАНУ ХАРЧУВАННЯ»

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.21810827

Студент _____ Р. Є. Цяпко

підпис

«__» _____ 2022 р.

Керівник старший викладач _____ С. В. Дворецька

підпис

«__» _____ 2022 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AMDR – Acceptable Macronutrient Distribution Range

API – Application Programming Interface

BMR – Basal Metabolic Rate

BSON – Binary JavaScript Object Notation

CRUD – Create Read Update Delete

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

MVC – Model View Controller

NPM – Node Package Manager

ORM – Object–relational mapping

PAL – Physical Activity Level

REST – Representational State Transfer

SPA – Single Page Application

TDEE – Total Daily Energy Expenditure

UI – User Interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

XML – Extensible Markup Language

ВСТУП

Людство пройшло довгий шлях, коли справа доходить до розробки Інтернету та сили зробити його доступним зручним способом за допомогою веброзробки. З розвитком технологій він пережив експоненційну еволюцію від свого скромного початку розробки статичного вебсайту, розміщеного у фізичній системі, до багатофункціональних вебзастосунків, які зберігаються на віддалених серверах та переважно нині використовуються.

Сьогодні вебзастосунки використовуються щодня: Інтернет-банкінг, соціальні мережі, онлайн сервіси та служби для бронювання, навчання, опитувань, застосунки для електронної комерції, інтерактивні ігри, блоги, форуми, системи керування вмістом тощо. З'являються складні інструменти графічного дизайну або відеоредактори, які працюють прямо в браузері.

Вебзастосунок має багато переваг, зокрема [1]:

– адаптивний дизайн може забезпечити гнучкість доступу до користувачів на будь-якому пристрої;

– не потребують місця на пристрої оскільки вихідні файли знаходяться на віддаленому сервері;

– вимагає менше підтримки у розробці та обслуговуванні з боку бізнесу та нижчих технічних вимог до пристрою користувача;

– усі користувачі мають доступ до однієї версії, тому це усуває будь-які проблеми з сумісністю;

– доступ до вебзастосунку будь-де за допомогою сумісного браузеру на кількох платформах незалежно від операційної системи чи пристрою;

– нові технології, такі як прогресивні вебзастосунки, дозволяють наближатися до рідних мобільних застосунків.

Багато людей починають новий рік із рішучістю покращити своє здоров’я. Це поліпшення часто починається зі зміни того, що вони їдять.

Корисним інструментом у цьому процесі може стати харчовий щоденник. Це може допомогти зрозуміти харчові звички та моделі харчування, а також визначити

корисні та не дуже продукти харчування які вживаються на регулярній основі. Дослідження показують, що для людей, зацікавлених у схудненні, ведення щоденника може бути дуже ефективним інструментом, який допоможе змінити поведінку. В одному дослідженні втрати ваги, в якому взяли участь майже 1700 учасників, ті, хто вів щоденний облік їжі, втратили вдвічі більше ваги, ніж ті, хто не робив жодних записів [2].

Деякі люди вважають це стомлюючим процесом, однак ведення обліку їжі:

– **підвищує самосвідомість.** З'являється потреба думати, що саме і як часто вживалося протягом дня. Це також підвищує усвідомлення розміру порції;

– **підвищує обізнаність про кількість калорій та макронутрієнтів у певному продукті харчування.** Деякі люди можуть випивати рідкі калорії і не усвідомлювати, скільки грамів цукру або калорій споживають. Відстеження споживання за допомогою вебзастосунку допоможе зрозуміти, що саме вживається в організм.

– **допомагає зрозуміти загальний розподіл калорій між категоріями макронутрієнтів.** Якщо просто подивитися на калорії, можна не усвідомлювати, скільки їх надходить з білків, вуглеводів і жирів. Рекомендації щодо харчування США рекомендують споживати 10-35% калорій з білків, 20-35% калорій з жирів і 45-65% калорій з вуглеводів [3]. Коли починається відстеження їжі можна зрозуміти, що одна з типових дієт не відповідає необхідним рекомендаціям. Ця інформація може допомогти зрозуміти, що потрібно налаштувати.

– **допомагає зрозуміти контроль порцій.** Коли фіксується своє споживання, з'являється бажання знати точну частину кожного продукту, який споживається. Це допоможе навчити контролювати порції.

Спираючись на вищевказані дані, вибір припав на тему «Вебзастосунок обліку та контролю вжитих калорій на основі обраного плану харчування» оскільки з'являється потреба в щоденному відстеженні кожного прийому їжі з урахуванням кількості калорій та макронутрієнтів відповідно до обраного плану харчування.

Актуальність обраної теми полягає в тому, щоб надавати можливість відслідковувати щоденне харчування та підтримувати енергетичний баланс, при якому калорійність раціону харчування відповідатиме енергетичній потребі організму.

Об’єкт кваліфікаційної роботи – процес обліку та контролю вжитих калорій та макронутрієнтів на основі обраного плану харчування.

Предметом кваліфікаційної роботи є технології, методи та рекомендації для визначення та формування загальної добової витрати енергії та кількості макронутрієнтів.

Метою кваліфікаційної роботи є популяризація контролю та підтримки щоденного балансу вжитих калорій та макронутрієнтів за рахунок розробленого вебзастосунку.

Для досягнення поставленої мети необхідно розв’язати наступні **завдання**:

– аналіз предметної області та огляд сучасних застосунків для ведення щоденного обліку та контролю вжитих калорій на основі обраного плану харчування;

– розробка специфікації вимог до вебзастосунку;

– вибір технологій та програмних рішень для реалізації вебзастосунку;

– проєктування бази даних, структури серверної та клієнтської частини;

– програмна реалізація серверної та клієнтської частини, а також налаштування серверу бази даних та створення користувацького інтерфейсу.

Метою спеціального розділу з охорони праці є дослідження питань охорони праці, які безпосередньо пов’язані з діяльністю під час роботи з персональним комп'ютером.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Не можна заперечувати вплив їжі на здоров'я. Вживання високоякісної їжі має важливе значення для живлення організму та захисту його від запалення та окисного стресу [4]. Недостатнє споживання поживних речовин у щоденному раціоні харчування може призвести до будь-чого, від низького рівня енергії до хронічних захворювань та серйозних станів здоров'я [5]. Обізнаність і розуміння про різні типи поживних речовин у продуктах, вплив, який вони мають на тіло, і як включити їх у щоденний раціон, полегшує здоровий, збалансований спосіб життя [6].

Поживні речовини необхідні організму для сприяння росту і розвитку, а також для регулювання процесів в організмі, можна розділити на дві групи: макронутрієнти та мікронутрієнти (рис. 1.1).

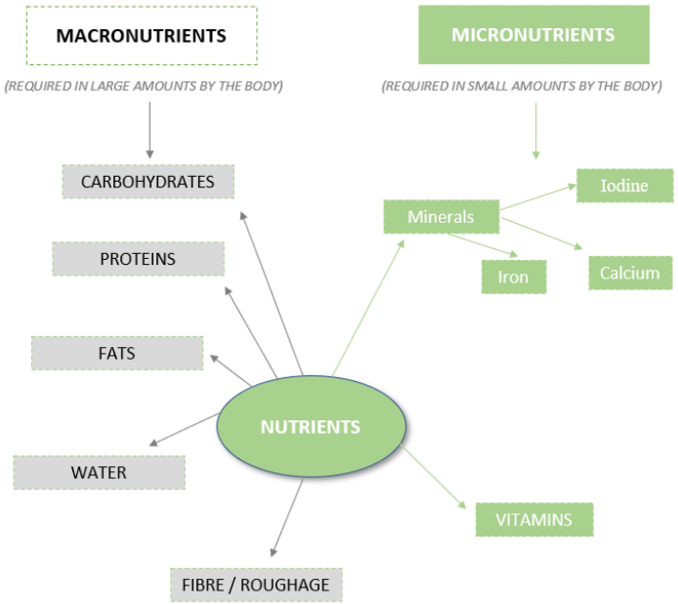

Рисунок 1.1 – Основні поживні речовини в їжі [8]

Макронутрієнти – це поживні речовини, необхідні організму у більшій кількості, а саме вуглеводи, білки та жири. Вони забезпечують тіло енергією або калоріями [7].

Мікронутрієнти – це поживні речовини, необхідні організму в менших кількостях, які зазвичай називають вітамінами та мінералами [7].

1.1 Метод підрахунку калорій та макронутрієнтів

Більшість людей не усвідомлюють, скільки вони їдять. Дослідження показують, що люди часто недооцінюють споживання їжі, іноді на 30-50 відсотків [9].

Дві ймовірні причини:

– **відсутність поінформованості, наскільки калорійною може бути багато продуктів.** Так, легко зрозуміти, що переповнена тарілка – це вірний спосіб набрати кілограмів, але дуже складно зрозуміти, наскільки калорійною може бути їжа в маленьких порціях.

– **неправильне оцінювання порцій.** Без зручної контрольної точки легко випадково спожити набагато більше калорій, ніж планувалося. Як наслідок, багато людей намагаються зрозуміти, скільки калорій містить вжита їжа, і не вживають їжу порціями відповідного розміру.

Звичайно, є добре відоме рішення – відстеження їжі, а саме: підрахунок калорій та макронутрієнтів.

Причини використання даного методу:

– **дослідження показують, що це працює.** Дослідження показують, що відстеження калорій і макронутрієнтів – навіть без будь-яких інших дієтичних консультацій допомагає людям втратити до п’яти відсотків ваги тіла [10]. Для людини, яка важить 100 кілограм, це втрата ваги на 5 кілограм.

– **забезпечує максимальну точність.** Відстеження калорій і макронутрієнтів не є на 100 відсотків точними, але це найточніші методи, доступні поза лабораторією. Важлива примітка: якщо ви вирішите оцінити розмір порції замість зважування та вимірювання їжі, цей метод стає менш точним;

– **інформування про калорії.** Відстежуючи макроси або калорії, краще усвідомлюєте, скільки калорій міститься у всьому, що з'їсти або випити.

Метод переважно призначений для:

– **короткочасного використання.** Відстеження ваших калорій або макросів протягом кількох тижнів може допомогти вам дізнатися більше про ваші

поточні харчові звички. Це також дає вам краще розуміння відповідних порцій. Після того, як ви впораєтеся з цим, ви можете перейти до ручних порцій і, зрештою, до саморегуляції;

– **людей з розширеними потребами.** Для більш точних цілей потрібна більша точність. Наприклад, припустимо, що комусь потрібно важити рівно 75 кілограм, щоб досягти своєї вагової категорії, або мати рівно 8% жиру для своєї професії. Відстеження калорій та/або макросів, як правило, є найефективнішим способом досягнення.

– **людей орієнтованих на цифри.** Деяким людям дійсно подобається процес збору даних про калорії та макронутрієнти, а потім моніторинг змін ваги, розміру тіла та показників здоров'я, таких як кров'яний тиск і холестерин. Вони також зазвичай емоційно відсторонені від чисел, тобто бачать їх як інформацію, а не приписують їм «хороші» чи «погані» значення. Для цих людей відстеження калорій або макросів може розширити можливості.

1.2 Аналіз та значення калорій у житті людини

Калорії є одиницею виміру. Вони є способом висловити певну кількість енергії. Люди, як правило, найбільш знайомі з «великими» калоріями, які передають кількість енергії в продуктах і напоях. Енергія калорій життєва необхідна для підтримки життя та здоров'я. Це допомагає підтримувати ключові функції організму, такі як функціонування клітин. Він використовує цю енергію негайно або зберігає її для подальшого використання, залежно від поточних потреб [11].

Існує багато причин рахувати калорії. Загалом, це дозволяє людині виміряти, скільки енергії вона споживає на день. Якщо людина приймає більше, ніж витрачає її організм, вона зазвичай починає набирати вагу. Якщо людина вживає менше, ніж вимагає її організм, вона зазвичай починає худнути [11].

Не всім потрібна однакова кількість калорій щодня. Люди мають різний метаболізм, який спалює енергію з різною швидкістю, і деякі люди ведуть більш активний спосіб життя, ніж інші [11].

Рекомендована добова норма калорій залежить від таких факторів як [11]:

- загальний стан здоров'я;
- рівень фізичної активності;
- стать;
- вага;
- зріст;
- вік;
- форма тіла.

Якби люди споживали лише необхідну кількість калорій щодня, вони, ймовірно, мали б здорове життя. Занадто низьке або занадто високе споживання калорій в кінцевому підсумку призведе до проблем зі здоров’ям. Кількість калорій у їжі говорить нам про те, скільки потенційної енергії вони містять. Важливі не тільки калорії, а й речовина, з якої калорії беруться.

Для визначення та обчислення необхідної щоденної кількості макронутрієнтів використовуються загальні енергетичні коефіцієнти Atwater [12], відомі як метод 4:4:9, для обчислення кількості калорій на грам кожного макронутрієнта:

- **вуглеводи**: 4 калорії на грам;
- **білки**: 4 калорії на грам;
- **жири**: 9 калорій на грам.

Здається, що калорії пов’язані лише зі збільшенням ваги та ожирінням, але вони життєво важливі для здоров’я. Вони становлять небезпеку для здоров’я лише тоді, коли люди споживають більше рекомендованої кількості.

Розмірковуючи про калорії, потрібно враховувати не лише раціон, а й рівень фізичної активності. Високому споживанню калорій можна подолати регулярними високоінтенсивними фізичними вправами.

1.3 Етапи обчислення загальної добової кількості макронутрієнтів

Для підрахунку «макросів» необхідно оцінити цільове співвідношення макронутрієнтів – іншими словами, відсоток щоденних калорій, які надходять з вуглеводів, білків і жирів. Для розуміння можна уявити персоналізовану кругову діаграму, яка враховує потреби тіла в енергії, цілі щодо ваги та загальну картину здоров'я.

Обчислення основного метаболічного показника

Основний метаболічний показник (BMR) – це відображення того, скільки калорій на день необхідно організму для функціонування залежно від зросту, ваги, віку та статі. Формула була вперше розроблена в 1918 році [13] і за останні десятиліття зазнала різноманітних уточнень [14]. Треба зауважити, що подібні розрахунки метаболізму своїми руками є приблизними оцінками.

Щоб отримати загальне уявлення про BMR, треба надати свою інформацію до одного з найновіших рівнянь Міффліна-Сент-Джора [14], наведеного нижче (формула 1, 2), а потім округлити відповідь до найближчого цілого числа.

$$BMR_{men} = 10 * W + 6.25 * H - 5 * A + 5 \quad (1)$$

$$BMR_{women} = 10 * W + 6.25 * H - 5 * A - 161 \quad (2)$$

де W – вага;

H – зріст;

A – вік у повних роках.

Обчислення загальної добової витрати енергії

Загальна добова витрата енергії (TDEE) – це енергія, яка в середньому спалюється за цілий день. Вона відображає середню кількість енергії, витраченої протягом звичайного дня. Єдиної універсальної формули TDEE для оцінки загальних добових витрат енергії не існує. Оцінка TDEE зазвичай проводиться шляхом оцінки рівня основного метаболізму (BMR), а потім його множення на відповідний рівень фізичної активності (PAL). Дослідники розробили кілька рівнянь, які передбачають основний метаболізм. Різні формули вимагають

використання різних змінних, і, незважаючи на деякі обмеження, вони використовуються як простий і доступний метод розрахунку потреби в енергії [15].

Рівень фізичної активності (PAL) – це спосіб вираження щоденної фізичної активності людини. Різні значення PAL були отримані дослідниками за допомогою прямих методів оцінки TDEE, і вони були підтверджені в багатьох дослідженнях [15].

Для визначення TDEE необхідно помножити BMR на коефіцієнт, який найкраще відповідає рівню активності за середній тиждень та знову округлити відповідь до найближчого цілого числа [15].

Класифікація рівнів фізичної активності [15]:

– сидячий спосіб життя (мало або відсутність фізичних навантажень): **1,2**;

– легка активність (легкі вправи один-три дні на тиждень): **1,375**;

– помірна активність (помірні фізичні навантаження три-п’ять днів на тиждень): **1,55**;

– дуже активний (інтенсивні фізичні вправи шість-сім днів на тиждень): **1,725**;

– надзвичайно активний (дуже інтенсивні фізичні вправи шість або сім днів на тиждень, або фізично важка професія): **1,9**.

Врахування цілі щодо ваги

Втрата ваги відбувається, коли використовується більше енергії, ніж споживаєте та навпаки. Рекомендації громадського здоров’я показують [16], що поступове, постійне зниження ваги приблизно на 0,5 – 1 кілограм на тиждень найкраще для довгострокового успіху. Щоб скинути 1 кілограм на тиждень, знадобиться щоденний дефіцит у 500 калорій – іншими словами, TDEE мінус 500.

Визначення індивідуального співвідношення макронутрієнтів

Після того як розрахована загальна добова потреба в калоріях, настав час оцінити, який відсоток має виходити з кожної з трьох категорій макронутрієнтів. Як і для більшості речей у світі харчування, тут немає універсальних рекомендацій.

Це ґрунтується на особистих потребах з урахуванням способу життя, уподобань в їжі, ліків, які, можливо, приймаються, історії хвороби і навіть стадії життя.

Для більшості здорових дорослих корисною відправною точкою для розгляду є три діапазони допустимого розподілу макронутрієнтів (AMDR). Розроблені Національною академією наук [17], ці діапазони макронутрієнтів, як показано, пов'язані зі зниженим ризиком хронічних захворювань. Для дорослої людини AMDR це:

- **вуглеводи:** 45-65% калорій;
- **жири:** 20-35% калорій;
- **білки:** 10-35% калорій.

1.4 Огляд та аналіз сучасних конкурентних вебзастосунків

Сучасні застосунки для підрахунку калорій дозволяють реєструвати та відстежувати щоденне споживання калорій, часто витягуючи інформацію про калорії з бази даних продуктів або завантажуючи фотографії відсканованих етикеток продуктів, також вони надають індивідуальний план щоденного споживання калорій, щоб допомогти досягти персональних цілей щодо здоров'я.

Застосунок «MyFitnessPal» (рис. 1.2) [18]

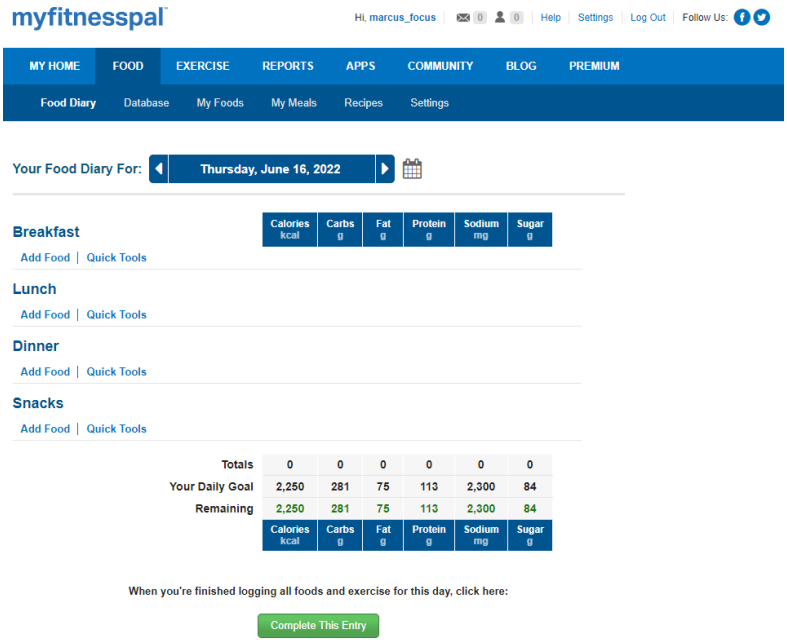

Рисунок 1.2 – Інтерфейс вебзастосунку «MyFitnessPal» [18]

Основні функції:

– відстежує вагу та розраховує рекомендоване щоденне споживання калорій;

– показує залишок рекомендованого споживання та кількість спалених калорій під час фізичних вправ;

– синхронізується з пристроєм для відстеження фітнесу, для додавання даних в журнал вправ;

– відстежує прогрес у досягненні цілей і пропонує чат-форуми з іншими користувачами;

– завантажує рецепти з Інтернету або створює власні;

– зберігає улюблені страви для зручного ведення журналу;

– дозволяє миттєво вводити інформацію про поживність деяких упакованих продуктів за допомогою сканеру штрих-кодів;

– надає можливість робити нотатки для кожного дня щодо самопочуття.

Застосунок Nutritionix (рис. 1.3) [19]

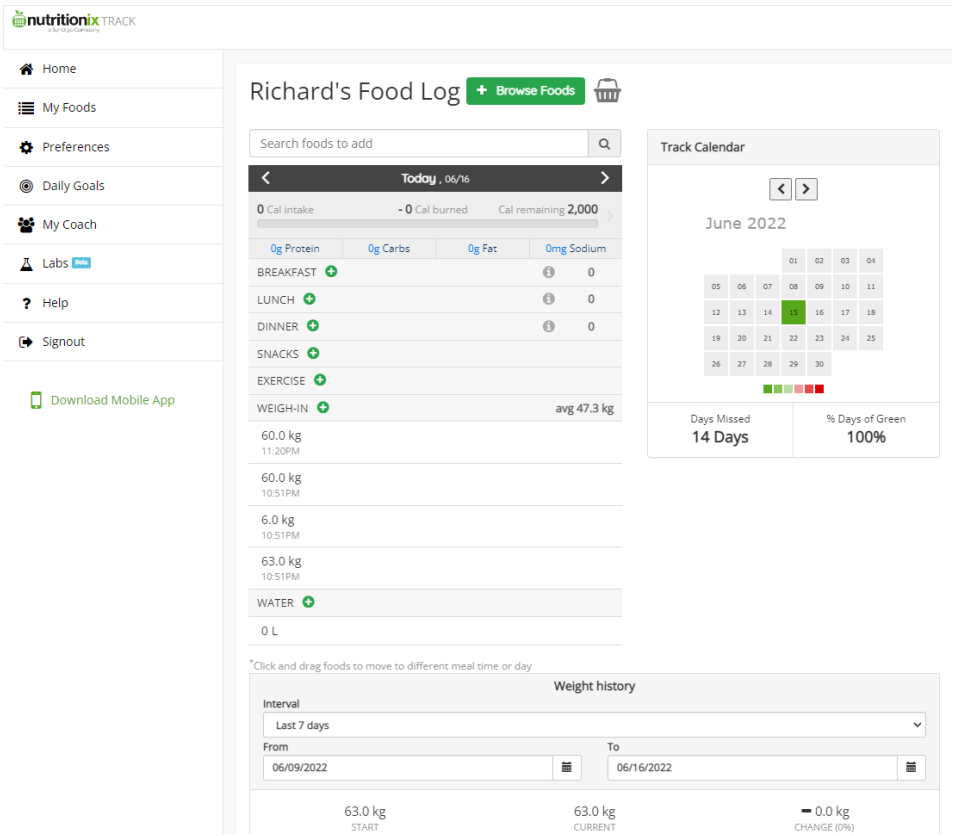

Рисунок 1.3 – Інтерфейс вебзастосунку «Nutritionix» [19]

Основні функції:

– відстежує прийом їжі, випитої води, макронутрієнтів;

– відстежує зміни ваги;

– встановлює цілі щодо калорій та макронутрієнтів;

– має базу даних із більш ніж 800 тисяч унікальних продуктів харчування;

– надає можливість створювати кастомні продукти та рецепти;

– надає можливість експортувати дані як .csv у файл.

Застосунок «Cronometer» (рис. 1.4) [20]

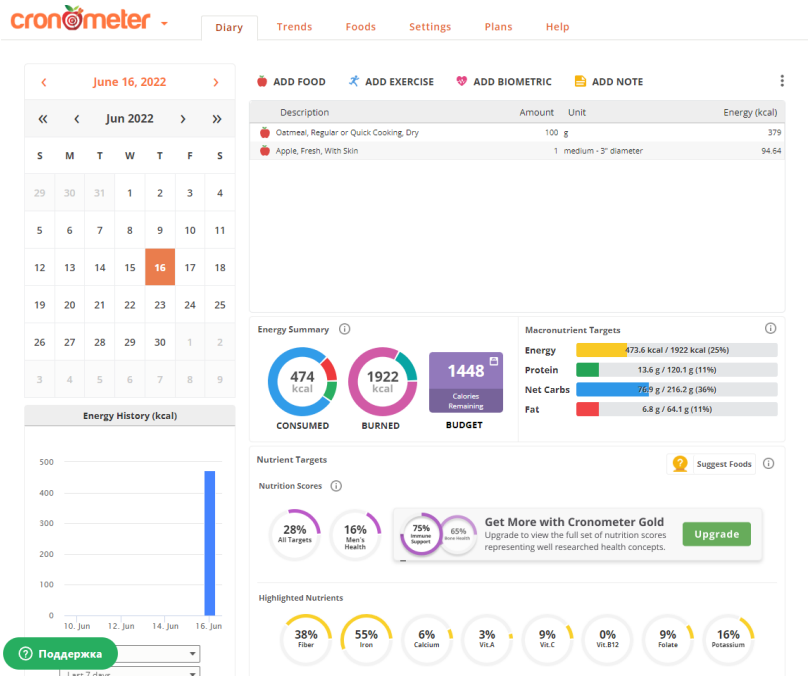

Рисунок 1.4 – Інтерфейс мобільного застосунку «Cronometer» [20]

Основні функції:

– відстежує раціон харчування, вправи та вагу тіла;

– пропонує точні розміри порцій і корисну базу даних вправ;

– має індивідуальний профіль на основі більшої потреби в калоріях;

– показує розподіл вуглеводів, жирів і білків за день разом із загальними спожитими калоріями.

На підставі розглянутих застосунків для обліку та контролю щоденного харчування варто виділити та порівняти наявність деяких важливих функцій (табл. 1.1):

– використання SPA архітектури для швидкої та зручної навігації;

– наявність сучасного UI, який буде інтуїтивно простим та зрозумілим для користувача;

– адаптивність до пристроїв з різними дисплеями та максимальними дозволами екранів;

– додавання продуктів до певного прийому їжі;

– можливість виробляти кастомний продукт;

– використання пошукових фільтрів для більш гнучкого та швидкого пошуку та форматування необхідних продуктів харчування.

Таблиця 1.1 – Порівняння наявності важливих функцій серед розглянутих аналогів

Аналог / Функція	MyFitnessPal	Nutritionix	Cronometer
SPA архітектура	–	+	+
Сучасний UI	–	–	+
Адаптивний до різних пристроїв	+	+	–
Додавання продуктів харчування до певного прийому їжі	+	+	–
Кастомні продукти	–	+	+
Пошукові фільтри	–	–	+

Виходячи з отриманих результатів при порівнянні аналогів, можна зробити висновок, не всі аналоги є ідеальними. Саме тому є доцільним створити такий вебзастосунок, який буде забезпечений всіма порівнянними функціональними вимогами.

1.5 Специфікація вимог до вебзастосунку

У цьому підрозділі детально описується план проєкту з розробки вебзастосунку обліку та контролю вжитих калорій на основі обраного плану харчування.

Загальний опис

Майбутній вебзастосунок забезпечуватиме користувачам інтуїтивно простий і доступний спосіб визначення добової кількості калорій залежно від бажаного результату ваги тіла та встановлення відповідних пропорцій макронутрієнтів із запропонованих дієт на основі отриманих персональних даних про користувача після проходження своєрідної анкети.

Сфера застосування

Користувачі, у яких є потреба в щоденному відстеженні кожного прийому їжі з урахуванням кількості калорій та макронутрієнтів відповідно до обраного плану харчування або бажання дізнатися більше про продукти, які щодня вживаються.

Характеристики користувачів

– **користувач (user):** має доступ до облікового запису, продуктів та плану харчування та взаємодії з ними в межах свого облікового запису, а також налаштувань інтерфейсу;

– **адміністратор (admin):** успадковує весь функціонал користувача, а також має доступ до керування обліковими записами всіх користувачів та планами харчування.

Загальна структура та використовувані технології для розробки вебзастосунку

Вебзастосунок повинен використовувати 3-х рівневу архітектуру та складатися з трьох фундаментальних компонентів: клієнтська та серверна частина застосунку, а також бази даних.

– мовою програмування для розробки вебзасосунку є **JavaScript** використовуючи крос-платформне середовище виконання **Node.js**;

– для реалізації серверної частини вебзастосунку використовується серверний фреймворк **Express.js**;

– для реалізації клієнтської частини вебзастосунку використовується клієнтський фреймворк **Vue.js**;

– програмним забезпеченням для бази даних використовується хмарне сховище **MongoDB Atlas Cluster**;

– для реалізації користувацького інтерфейсу використовуються UI компоненти, які надає бібліотека **Vuetify**;

– для надання та взаємодії з даними про продукти харчування та аналітику використовується API сервіс **Edamam**.

Функціональні вимоги

– реєстрація та авторизація облікового запису;

– верифікація та відновлення доступу до облікового запису за допомогою електронної пошти;

– редагування налаштувань інтерфейсу вебзастосунку;

– проходження анкети для формування плану харчування;

– встановлення щоденної кількості вжитих калорій та макронутрієнтів на підставі отриманих даних про користувача;

– редагування облікового запису (зміна даних профілю, додавання та оновлення світлини користувача, видалення всіх доданих продуктів харчування, видалення облікового запису та всіх наявних даних про користувача);

– навігація по щоденнику харчування за допомогою календаря та відображення доданих продуктів харчування;

– пошук продуктів харчування за допомогою стороннього API сервісу, а також можливість використовувати пошукові фільтри;

– керування продуктами у щоденнику харчування (CRUD);

– створення кастомного продукту харчування;

– відстеження та оновлення вжитих калорій та макронутрієнтів за допомогою діаграм та трекерів;

– керування обліковими записами та планами харчування за допомогою адмін-панелі (CRUD);

– вихід із системи.

Вимоги надійності

Вебзастосунок має використовувати безперервну інтеграцію, щоб функції та виправлення помилок можна було швидко розгорнути без простоїв.

Вимоги зручності

– вебзастосунок має на меті максимально зручне використання функціональності системи;

– вебзастосунок повинен використовувати SPA архітектуру;

– вебзастосунок повинна підтримувати адаптивність до пристроїв з різною роздільною здатністю екрану (desktop, laptop, netbook, tablet, smartphone);

– користувацький інтерфейс повинен бути легким для розуміння і дозволяти користувачам досягати своїх цілей без помилок.

Вимоги продуктивності

– вебзастосунок має завантажитися та бути доступним протягом доступу до мережі Інтернет;

– вебзастосунок повинен моментально оновлювати інтерфейс при взаємодії з користувачем.

Вимоги безпеки

– аутентифікація повинна здійснюватися за допомогою JWT та cookie-даних;

– паролі користувачів мають використовувати безпечне шифрування;

– дані повинні бути захищені від спроб змінити їх іншими користувачами або без наявних прав адміністратора;

– ключі та паролі, що використовуються, для доступу до сторонніх сервісів повинні надійно зберігатися.

Висновки до розділу 1

У першому розділі було проведено аналітичний аналіз предметної області, на підставі яких були сформовані наступні стратегії, які можуть допомогти у реалізації вебзастосунку:

– створення певної анкети для отримання інформації про користувача;

– формування плану харчування за допомогою математичних обчислень використовуючи формули BMR та TDEE ґрунтуючись на отриманих даних про користувача;

– використання різних планів харчування, що ґрунтуються на пропорційному співвідношенні кількості макронутрієнтів до калорій.

Під час пошуку сучасних застосунків, щодо існуючих методів визначення та контролю спожитих калорій, а також загальної добової кількості макронутрієнтів, було розглянуто вже готові проєкти. Спираючись на аналіз та порівняння розглянутих аналогів було визначено важливі вимоги, якими має бути оснащений вебзастосунок, який розробляється.

Підрахунок щоденної кількості калорій та макронутрієнтів може допомогти людям досягти та підтримувати цілі щодо ваги та фізичної підготовки, оскільки він включає відстеження того, скільки енергії надходить в організм щодня. Для тих, хто хоче змінити свою вагу, важливо також враховувати рівень активності та дефіцит калорій. Всі ці фактори відіграють важливу роль.

На підставі інформації про розглянуті аналоги можна зробити висновок, що не всі застосунки є ідеальними. Виходячи з цього, доцільно було б реалізувати вебзастосунок, який має SPA архітектуру, зрозумілий і сучасний UI, а також, важливою є адаптивність до екранів з різною роздільною здатністю. Безумовно, на відміну від багатьох інших застосунків користувач повинен мати доступ до всієї інформації про продукти харчування, що надає йому API сервіс, у тому числі з гнучким пошуковими фільтрами та налаштуванням щодо вибору планів харчування та макронутрієнтів та створення своїх кастомних продуктів харчування.

2 ВИБІР ТЕХНОЛОГІЙ ТА ПРОГРАМНИХ РІШЕНЬ

2.1 Визначення архітектури та основних компонентів вебзастосунку

Вебзастосунок – це клієнт-серверний застосунок, де є браузер (клієнт) і вебсервер. Логіка вебзастосунку розподілена між сервером і клієнтом, є канал для обміну інформацією та зберігання даних локально або в хмарному сховищі [21].

Архітектура вебзастосунків – це структура високого рівня, яка визначає спосіб функціонування, продуктивності та масштабування продукту або бізнесу. Сьогодні на етапі вибору архітектури вебзастосунків можна легко загубитеся в різноманітних варіантах, доступних на ринку розробки програмного забезпечення. Чим більше з'являється нових імен і тенденцій, тим важче стає визначитися, яка сучасна архітектура вебзастосунків найкраща і які критерії використовувати для оцінки [21].

Сучасні вебзастосунки все ще використовують концепцію 3-рівневої архітектури, яка поділяє застосунок на рівень презентації, програми та даних (рис. 2.1) [21].

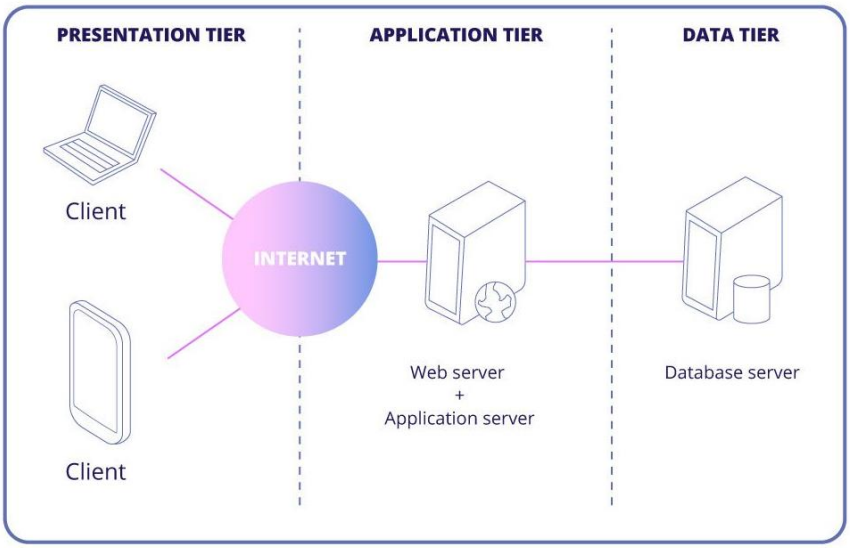

Рисунок 2.1 – 3-рівнева архітектура вебзастосунку [21]

У 3-рівневій архітектурі вебзастосунків кожен рівень працює на власній інфраструктурі і може паралельно розроблятися різними командами. Така

структура дозволяє оновлювати та масштабувати кожен рівень за потреби, не впливаючи на інші рівні [21].

Веброзробка – це безперервний життєвий цикл, із кількох етапів та ітерацій, що включає масштабні технічні рішення щодо фреймворків і технологій, які підтримуватимуть застосунок, а також фактичного кодування вебзастосунку та підтримуваного вебсервера для розміщення застосунка.

Гарний вебзастосунок, який відображається у браузері має багато роботи за лаштунками. Хоча кожен вебзастосунок може мати різні атрибути, три основні компоненти здійснюють кожну взаємодію між користувачем і вебзастосунком:

– **клієнтська частина (front-end):** включає функціональну взаємодію з користувачем та макет вебзастосунку з використанням HTML, CSS і JavaScript. Велика проблема в цьому випадку полягає в тому, щоб переконатися, що вебсайт функціонує однаково в усіх браузерах та дисплеях;

– **серверна частина (back-end):** обробляє отримані клієнтські запити за допомогою доступу до бази даних вебзастосунку, після чого формує і повертає бажану відповідь назад клієнту, простими словами виступає в ролі посередника між клієнтською частиною та базою даних.

– **база даних:** сховище, яке відповідає за збір, зберігання, обробку та керування зібраними даними та забезпечення безпечного доступу до цих даних.

Для успішної розробки клієнтської частини будь-якого вебзастосунку, як правило необхідно на впевненому рівні розуміти і вміти взаємодіяти з HTML, CSS і JavaScript, або як альтернатива, використовувати один з клієнтських фреймворків, який надає весь перерахований вище функціональний набір технологій. Найбільш популярні нині такі фреймворки як React, Vue.js, Angular, Svelte.

Розробка серверної частини зобов'язує використання однієї з мов програмування для взаємодії та обробки запитів, що надходять і надсилаються між клієнтом і базою даних вебзастосунка. Також як і у випадку з клієнтською частиною у зв'язку з стрімким рівнем розвитку технологій з'являється все більше і більше сучасних і функціональних серверних фреймворків, які надають

можливість для реалізації всіх необхідних вимог для коректної роботи вебзастосунку, що розробляється.

Вебфреймворк – це бібліотека коду, яка робить веброзробку швидшою та простішою, надаючи основні шаблони для створення надійних, масштабованих і підтримуваних вебзастосунків. Вебфреймворки існують, щоб полегшити розробнику створення та контролювання процесу розробки програмного забезпечення або більшу його частину з однієї платформи. Вебфреймворки уникають необхідність переписувати код з нуля кожного разу, коли створюється вебзастосунок або вебсервіс, так як за все це відповідає безпосередньо вебфреймворк [22].

Фреймворки надають функціональність у своєму коді або через розширення для виконання повсякденних операцій, необхідних для запуску вебзастосунків. Ці операції **включають** [22]:

– маршрутизація URL-адрес;

– керування формою введення та перевірка;

– HTML, XML, JSON та інші налаштування продукту з механізмом шаблонування;

– конфігурація з'єднання з базою даних і чітке маніпулювання даними через ORM;

– веббезпека від CSRF (Cross-site request forgery), XSS (Cross-site scripting), ін'єкції SQL та інших частих зловмисних атак;

– репозиторій сесії та відновлення.

Переваги використання вебфреймворку [23]:

– допомагає у впровадженні кращих практик програмування та відповідності використанню шаблонів проєктування;

– код стає більш безпечний;

– можна уникнути дублювання та надлишкового коду;

– допомагає послідовній розробці коду з меншою кількістю помилок;

– полегшує роботу зі складними технологіями;

– функціональність постійно вдосконалюється за рахунок внесення змін до фреймворків з відкритим кодом;

– сегменти коду та функціональні можливості попередньо створені та протестовані, що робить застосунки більш надійними;

– тестування та налагодження коду набагато простіше і може бути виконано навіть розробниками, які не володіють кодом;

– час, необхідний для розробки вебзастосунку, значно скорочується.

З розвитком вебстандартів логіка застосунків перемістилася в бік клієнта, що забезпечує розумнішу комунікацію між користувачем і вебзастосунком. Завдяки логікі на стороні клієнта можливо швидко реагувати на дії користувача. Це робить вебзастосунки більш чуйними, доступними для навігації на будь-якому пристрої. Таким чином, є дві **функції** фреймворків:

a) для роботи на стороні сервера, яка допомагає налаштувати логіку вебзастосунку;
b) для роботи на стороні клієнта для створення UI та взаємодії з даними.

2.2 Вибір фреймворку для клієнтської частини

Фреймворки на стороні клієнта функціонують всередині браузера та є потужним інструментом для розробки складних інтерфейсів користувача. Вони заохочують створювати модульну, автономну архітектуру, яка підтримується в обслуговуванні, що полегшує створення застосунку та співпрацю з іншими розробниками. Кожен фреймворк на стороні клієнта відрізняється функціональністю та використанням, але всі вони використовують JavaScript як мову програмування. Популярні фреймворки підтримуються спільнотами підтримки, багатьма документацією та навчальними посібниками, а також пропонують перевірений у боях код, який вирішує звичайні проблеми, які ставлять передні панелі під час їхнього масштабування. Дані фреймворки дають змогу використовувати найсучасніші функції JavaScript та пропонують інструменти, які

полегшують створення прототипів програм. Нарешті, вони дають можливість обговорювати архітектуру та проблеми спільною мовою [24].

Найбільш популярні клієнтські фреймворки

React – це один із найпопулярніших клієнтських фреймворків. Його бібліотека основана на компонентах JavaScript із синтаксисом JSX, яка була розроблена Facebook. React дуже швидко завоював популярність. Він створює інтерактивний і динамічний UI для вебсайтів і мобільних застосунків з високим вхідним трафіком. Декларативні уявлення роблять код читабельним і легким для налагодження. Він використовує віртуальний DOM, і, отже, інтеграція з будь-яким застосунком є більш простою. У React створена повністю сумісна система подій об’єктної моделі W3C. Він також забезпечує міжбраузерний інтерфейс для рідної події. React використовує одностороннє прив’язування даних з архітектурою застосунку, яка називається Flux controls. ReactJS допомагає оновлювати View для користувача за допомогою Flux, який контролює робочий процес застосунка.

Vue.js – це один з найпростіших фреймворків на сьогодні. Він має віртуальний DOM, архітектуру на основі компонентів і двостороннє прив’язування, що лежить в основі його високошвидкісної продуктивності: все це полегшує оновлення пов’язаних компонентів і відстеження змін даних, що є бажаним для будь-якого застосунку, в якій оновлення в реальному часі є обов'язковими. Розробники, які вибирають Vue.js, можуть отримати переваги від його невеликого розміру в порівнянні з React або іншими фреймворками (файл ZIP з фреймворком важить лише 18 КБ) та можливість створювати прогресивні вебзастосунки (PWA).

Основна ідея розробки Vue полягає в тому, щоб надати набагато простішу концепцію, тому Vue вважається однією з найбільш зручних для початківців фреймворків, що мають добре розроблену документацію та підтримку спільноти. Vue має широкий вибір інструментів, таких як інструменти наскрізного тестування, системи встановлення плагінів, інструменти налагодження браузера, засіб візуалізації серверів, менеджер станів та інші. Незважаючи на те, що Vue створений для вирішення складнощів і підвищення продуктивності застосунків, він поки не

дуже популярний серед гігантів галузі. Vue.js продовжує поширюватися з точки зору прийняття, незважаючи на меншу кількість користувачів із Кремнієвої долини.

Angular – це один з найпотужніших та найефективніших фреймворків на основі TypeScript. Він використовується для розробки односторінкового застосунку (SPA) та володіє можливостями сучасної вебплатформи для забезпечення високопродуктивних, офлайнових та нульових кроків установки. Angular перетворює шаблони на код, який дуже оптимізований для віртуальних машин JavaScript, надаючи переваги рукописного коду. За допомогою нового Component Router angular-застосунки швидко завантажуються, забезпечуючи автоматичне розділення коду. Angular створює UI представлення за допомогою простого та потужного синтаксису шаблонів, а також має Angular CLI, які дозволяють швидко розпочати створення, додавання компонентів та тестів, а потім миттєво розгортати.

Svelte – це не фреймворк і не бібліотека, а компілятор. Svelte відомий як легкий варіант розробки інтерфейсу та дозволяє розробникам виконувати свої проєкти з набагато меншою кількістю кодування, ніж під іншими фреймворками. Крім того, він вважається одним із найшвидших інтерфейсних фреймворків.

Кожен розглянутий клієнтський фреймворк має як сильні, так і слабкі сторони, тому доцільно згадати їх для більш об'єктивного оцінювання (табл. 2.1).

Таблиця 2.1 – Порівняльна характеристика клієнтських фреймворків

Назва	Переваги	Недоліки
React	– часто оновлюється; – віртуальний DOM забезпечує високу швидкість; – поєднується з багатьма іншими бібліотеками JavaScript; – дозволяє писати компоненти без класів; – легкість переходу між різними версіями; – багаторазові компоненти коду; – легко вчитися.	– відсутність чітко розробленої документації; – складнощі вивчення синтаксису JSX.

Кінець таблиці 2.1

Vue.js	– має високу швидкість; – простий для навчання синтаксис; – детальна документація; – реактивне двостороннє прив’язування даних; – прогресивні застосунки; – позитивно впливає на SEO.	– порівняно менше компонентів; – надмірна гнучкість; – обмежена спільнота розробників.
Angular	– архітектура на основі компонентів; – директиви та функції ін'єкції залежностей; – двостороння прив’язка даних; – дуже сильно протестований; – покращена продуктивність сервера; – простіше маніпулювання DOM; – односторінкові застосунки; – велика екосистема та висока продуктивність.	– складним для масштабних сценаріїв; – складний для навчання; – можливі проблеми з продуктивністю; – обмежені можливості SEO; – роздутий код і великий розмір.
Svelte	– масштабована структура; – швидка реактивність; – компонентна архітектура з мінімальним кодом; – легкий, спрощений, працює з бібліотеками; – немає віртуального DOM; – найновіший з усіх; – SEO-оптимізований.	– обмежена екосистема; – відсутність допоміжних матеріалів; – обмежений інструментарій; – не масштабований.

Використовуючи інформацію та порівняльні характеристики в якості клієнтського фреймворку був обраний саме **Vue.js** за рахунок того, що він має високу швидкість та легкість налаштування, а також використовує реактивне двостороннє зв'язування даних і віртуальний DOM та надає можливість створювати SPA і PWA. Усі ці переваги закріплюються легким для розуміння синтаксисом і докладною документацією, що створює дуже низький поріг входу для вибору саме даного фреймворку, натомість отримуючи розробку з використанням фреймворку, що стрімко розвивається і охоплює все більшу і більшу спільноту.

2.3 Вибір фреймворку для серверної частини

Для створення вебзастосунку з добре розробленим інтерфейсом потрібен більш широкий спектр функціональних можливостей. Фреймворки на стороні сервера обробляють HTTP-запити, контроль та керування базою даних, відображення URL-адрес тощо. Ці фреймворки можуть покращити безпеку та сформувати вихідні дані, спрощуючи процес розробки [24].

Найбільш популярні серверні фреймворки

Django – це провідний серверний фреймворк з відкритим вихідним кодом, заснований на мові програмування Python та відповідає архітектурі MVC. Django підходить для розробки складних і багатофункціональних вебсайтів, керованих базами даних, і є одним із найпростіших серверних фреймворків. Цей серверний фреймворк сприяє оптимальній підключеності, зменшенню кодування, більшій можливості повторного використання та швидшій розробці. Він надає додатковий інтерфейс адміністратора, який допомагає створювати, читати, оновлювати та видаляти операції. Django також має власний вебсервер для модульного тестування Python і компоненти, необхідні для вирішення деяких випадків.

Lavarel – це вебфреймворк PHP на основі Symfony, які відповідають архітектурі MVC. Він пропонує модульну систему пакування, оснащену спеціальним менеджером залежностей. Laravel також надає своїм користувачам кілька способів доступу до реляційних баз даних разом із утилітами обслуговування та розгортання застосунків.

Ruby on Rails – це фреймворк MVC для створення серверних застосунків на основі Ruby зі структурами баз даних за замовчуванням, вебсторінками та службами. Завдяки використанню форматів XML і JSON для передачі даних і HTML/CSS і JavaScript для інтерфейсів, Rails забезпечує швидку та поглиблену розробку вебзастосунків за допомогою Ruby. Найбільш помітними функціями RoR є шаблон активного запису, підхід «DRY» та «CoC».

Express.js – вважається стандартною серверною структурою Node.js. Він пишається як мінімальна, швидка та безпідставна структура. Він надає деякі

основні функціональні можливості фреймворку, не приховуючи особливостей Node, і використовує надійну продуктивність асинхронного Node.js. Він також досить гнучкий і підтримує повноцінні застосунки, а також REST API.

Spring Boot – це фреймворк з відкритим вихідним кодом та інверсією контейнеру керування на платформі Java створення високопродуктивних і надійних вебзастосунків. Він забезпечує легке керування залежностями та властивостями, що залежать від профілю. Spring допомагає створювати прості, портативні, швидкі та гнучкі системи та застосунки на основі JVM.

Кожен розглянутий серверний фреймворк має як сильні, так і слабкі сторони, тому доцільно порівняти їх для більш об'єктивного оцінювання (табл. 2.2).

Таблиця 2.2 – Порівняльна характеристика серверних фреймворків

Назва	Переваги	Недоліки
Django	– має низький рівень навчання; – поглиблена безпека; – висока масштабованість та універсальність.	– потрібні глибокі знання системи; – обмеження у використанні ORM; – монолітне розгортання.
Laravel	– Eloquent ORM та архітектура MVC; – легка аутентифікація; – відправка пошти на основі API; – спрощена конфігурація маршрутизації URL-адрес; – просте кешування даних; – зручна обробка логів; – широке тестування.	– необхідність у глибокому навчанні; – деякі оновлення викликають проблеми; – немає вбудованої функції підтримки.
Express.js	– швидке кешування та висока асинхронна продуктивність; – економічно ефективний з full-stack JavaScript розробкою; – легка масштабованість та навчання; – підтримка спільноти для спрощення розробки;	– знижує продуктивність під час виконання важких обчислювальних завдань.

Кінець таблиці 2.1

Ruby on Rails	– швидкість розвитку; – прості процеси автоматизації тестування; – послідовність у розробці; – безліч готових модулів і плагінів; – масштабованість та якісна розробка.	– обмежена гнучкість під час реалізації унікальних функцій; – сильні залежності між різними компонентами; – продуктивність під час виконання.
Spring Boot	– модель POJO для класів; – декларативне програмування; – усуває необхідність самостійно створювати фабричні та одиночні класи; – різні методи налаштування; – автоматична перевірка справності.	– відсутність контролю; – не підходить для масштабних проєктів; – складний для навчання; – паралельні механізми.

При реалізації вебзастосунку доцільно було б використовувати одну мовну базу для більш швидкої та ефективної розробки. Оскільки **Express.js** надає таку можливість за рахунок реалізації на **Node.js** і тим самим покриває всю розробку клієнт-серверного вебзастосунку однією мовою **JavaScript**. Крім цього, він має безлічі методів утиліти HTTP та middleware для створення надійного API, що вкрай важливо у цій реалізації.

2.4 Вибір бази даних

Те як керуються дані у вебзастосунку, відіграє вирішальну роль у забезпеченні позитивного досвіду користувачів. Зрештою, не має значення, наскільки добре розроблений інтерфейс застосунку та наскільки чистий код, якщо застосунок не здатний швидко отримувати, обробляти та передавати інформацію. Більше того, всі ці дані повинні бути захищені, щоб зловмисники не змогли отримати їх у руки. На щастя, цього можна досягти за допомогою грамотної обраної системи керування базами даних [25].

SQL – це зріла технологія: вони добре задокументовані, мають чудову підтримку та добре працюють з більшістю сучасних фреймворків і бібліотек. Найяскравішими прикладами баз даних SQL є PostgreSQL і MySQL. Обидва виявилися стабільними та безпечними [25].

Реляційна база даних – це набір таблиць, які мають заздалегідь визначені відносини між ними. Для підтримки та запитів до реляційної бази даних система керування базою даних використовує мову структурованих запитів (SQL), звичайну програму для користувача, яка забезпечує простий інтерфейс програмування для взаємодії з базою даних. Реляційні бази даних складаються з рядків, які називаються кортежами, і стовпців, які називаються атрибутами [25].

Бази даних NoSQL, також звані нереляційними або розподіленими базами даних, служать альтернативою реляційним базам даних. Вони можуть зберігати та обробляти неструктуровані дані (дані із соціальних мереж, фотографії, MP3-файли тощо) [25].

Найбільш популярним типом нереляційної бази даних є документно-орієнтовані бази даних, які зберігають всю інформацію, пов’язану з даним об’єктом, в одному файлі BSON, JSON або XML. Однотипні документи можна згрупувати в так звані колекції або списки. Ці бази даних дозволяють розробникам не турбуватися про типи даних і міцні зв’язки [25].

Коли справа доходить до вибору бази даних, однією з найбільших проблем є вибір між структурою даних SQL (реляційною) і NoSQL (нереляційною). Хоча обидві мають гарну продуктивність, є певні ключові відмінності, які треба пам’ятати (табл. 2.3) [25].

Таблиця 2.3 – Порівняльна характеристика SQL та NoSQL баз даних

Бази даних	Переваги	Недоліки
SQL	– ідеально підходить для зберігання структурованих даних; – підтримують права доступу; – захищає дані від втрати та пошкодження.	– бракує гнучкості та важкий обмін інформацією між застосунками при складних структурах; – запускаються лише на одному сервері;

Кінець таблиці 2.3

NoSQL	– мають більшу гнучкість, масштабованість та відмовостійкість; – запускаються на кількох серверах.	– менш зрілі; – спільнота не так чітко визначена; – жертвують відповідністю ACID заради доступності та гнучкості.

Вибираючи інструмент керування базою даних, слід звернути увагу на кілька аспектів [25]:

– **тип даних.** Бази даних SQL ідеально підходять для зберігання та обробки структурованих даних, а бази даних NoSQL є найкращим рішенням для роботи з неструктурованими або напівструктурованими даними. Якщо ви будете керувати як структурованими, так і неструктурованими даними, ви можете вибрати змішування баз даних SQL і NoSQL.

– **масштабованість.** Зі зростанням вебзасосунку одночасно має збільшуватися і його база даних. На вибір бази даних може вплинути тип масштабування: горизонтальне чи вертикальне. Нереляційні бази даних з їх сховищами ключ-значення оптимізовані для горизонтального масштабування, тоді як реляційні бази даних оптимізовані для вертикального масштабування.

– **безпека.** Оскільки в ній зберігаються всі дані користувача, база даних має бути добре захищеною. Реляційні бази даних, сумісні з ACID (Atomicity, Consistency, Isolation, Durability), є більш безпечними, ніж нереляційні бази даних, які обмінюють узгодженість і безпеку на продуктивність і масштабованість.

– **інтеграція.** Треба переконатися в тому, що система керування базами даних може бути інтегрована з іншими інструментами та сервісами у проєкті. У більшості випадків погана інтеграція з іншими рішеннями може зупинити розвиток.

Найбільш популярні бази даних

OracleDB – це СКБД, розроблена в 1977 році, залишається найбільш відомим і надійним рішенням. Він займає перше місце в рейтингу DB-Engines [25].

MySQL – це одна з найпопулярніших систем керування реляційними базами даних, створеної в 1995 році, а тепер керується Oracle. Uber, Tesla, YouTube, Netflix, Spotify, Airbnb і багато хто використовує MySQL для своїх послуг [25].

PostgreSQL – це об’єктно-реляційна база даних, що означає, що вона схожа на реляційні бази даних, тільки всі дані представлені у вигляді об’єктів замість стовпців і рядків [25].

MongoDB – це база даних NoSQL №1. У MongoDB всі дані зберігаються в документах BSON (двійковий JSON). Автоматичне розподілення означає, що можна легко розповсюджувати дані на сервери, підключені до вебзастосунку [25].

Кожна розглянута база даних має як сильні, так і слабкі сторони, тому доцільно порівняти для більш об'єктивного оцінювання (табл. 2.2).

Таблиця 2.4 – Порівняльна характеристика розглянутих баз даних

Назва	Переваги	Недоліки
OracleDB	– ідеальне рішення для великих проєктів, яким потрібно зберігати великий обсяг даних; – має останні інновації та функції	– малі та середні проєкти повинні шукати більш економічно ефективні альтернативи.
MySQL	– має відкритий вихідний код; – величезна база користувачів і відмінна підтримка; – добре працює з більшістю бібліотек і фреймворків; – швидке налаштування; – більшість завдань можна виконати в командному рядку; – структурована база даних з регулярними оновленнями.	– додаткова функціональність за фіксовану ціну; – потрібно багато часу для створення додаткових резервних копій або зміни архітектури даних;
PostgreSQL	– масштабований і призначений для обробки терабайту даних; – має розширений захист; – абсолютно безкоштовна база даних.	– потрібні експертні навички для вирішення проблем.

Кінець таблиці 2.4

MongoDB	– дані можна легко передавати між вебзастосунками та серверами у зрозумілому форматі; – високу масштабованість і доступність; – автоматичне розподілення; – чудове рішення для роботи з масивними неструктурованими наборами даних.	– зберігає імена ключів для кожної пари значень, збільшуючи використання пам’яті; – немає обмежень зовнішнього ключа для забезпечення узгодженості; – вкладення не більше ніж для 100 рівнів.

Саме використання **MongoDB** є найбільш оптимальним рішенням для вибору бази даних, оскільки вона легко інтегрується в даний технологічний стек і дозволяє маніпулювати даними у зрозумілому форматі з використанням гнучкого функціоналу, який надає хмарне сховище MongoDB Atlas Cluster.

Висновки до розділу 2

У другому розділі було визначено архітектуру сучасних вебзастосунків та причини використання вебфреймворків для їх реалізації.

Наступним кроком було проведення аналізу існуючих популярних фреймворків як для серверної, так і клієнтської частини вебзастосунку. Крім цього, не мало важливо було розглянути існуючі популярні бази даних. На підставі наявних даних, було проведено порівняння кожного розглянутого фреймворку та бази даних для виявлення переваг та недоліків. Виходячи з цього, було сформовано наступний технологічний стек, який повністю забезпечує реалізацію всіх необхідних функціональних та нефункціональних вимог до вебзастосунку:

– для реалізації серверної частини вебзастосунку був обраний серверний фреймворк **Express.js**;

– для реалізації клієнтської частини вебзасосунку був обраний клієнтський фреймворк **Vue.js**;

– для зберігання та взаємодії з даними була обрана база даних **MongoDB**.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Моделювання діаграми варіантів використання

Перш ніж перейти безпосередньо до самої програмної реалізації вебзастосунку, необхідно мати повне уявлення про роботу кожної з функції. Найбільш зручним способом моделювання та проєктування функціональних елементів вебзастосунку є побудова відповідних UML діаграм.

UML – це уніфікована мова моделювання, що складається з інтегрованого набору діаграм, розроблених для допомоги розробникам систем і програмного забезпечення для визначення, візуалізації, конструювання та документування артефактів програмних систем, а також для бізнес-моделювання та інші непрограмні системи. UML являє собою набір найкращих інженерних практик, які виявилися успішними в моделюванні великих і складних систем та використовує переважно графічні позначення для вираження дизайну програмних проєктів [26].

Діаграма варіантів використання описує функціональність системи з точки зору акторів, цілей як варіантів використання та залежностей (рис. 3.1) [26].

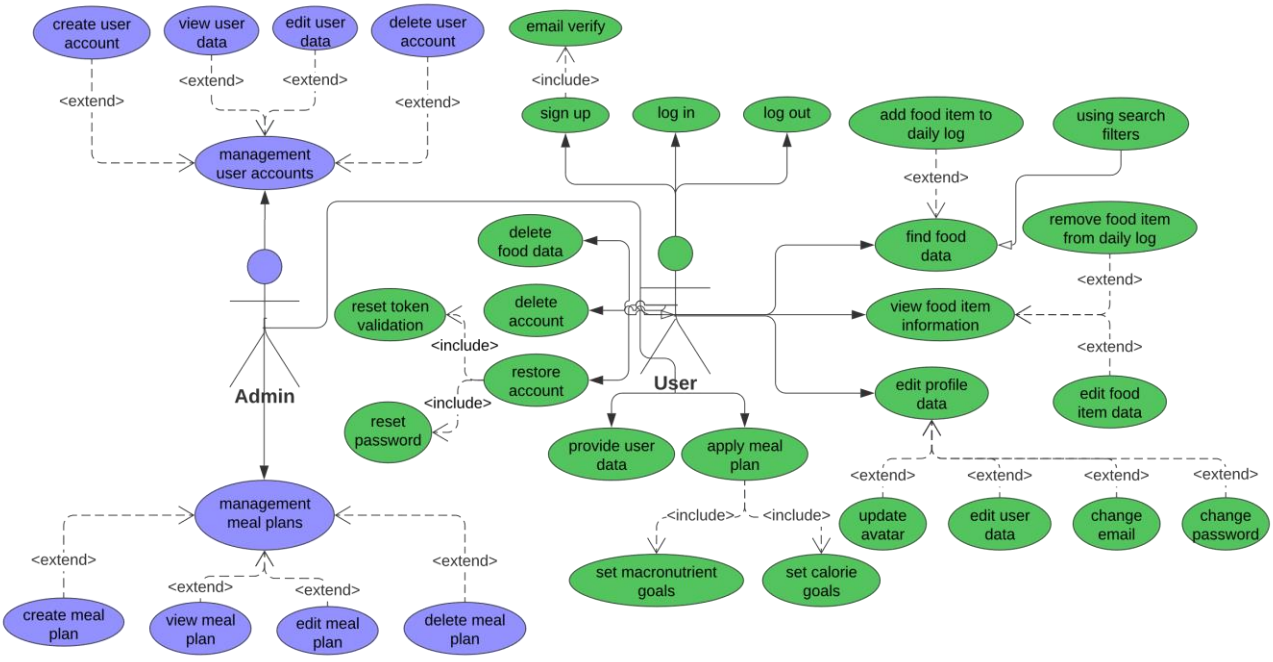

Рисунок 3.1 – Діаграма варіантів використання відповідно до функціональних вимог вебзастосунку

Через те, що діаграма варіантів використання показує лише взаємозв'язок між усіма функціональними вимогами вебзастосунку, є доцільним створити список усіх сценаріїв та докладно описати їх деталі наскільки це можливо.

3.2 Складання та опис сценаріїв роботи вебзастосунку

Сценарій – це формальний опис потоку подій, які відбуваються під час виконання екземпляра варіанту використання. Він визначає конкретну послідовність подій між системою та зовнішніми акторами. Зазвичай він описується в тексті і відповідає текстовому представленню діаграми послідовності.

У таблиці 3.1 описується призначення всіх сценаріїв згідно з створеною діаграмою варіантів використання.

Таблиця 3.1 – Сценарії вебзастосунку

№	Назва	Призначення
U-1	sign up	Реєстрація облікового запису
U-1.1	email verify	Підтвердження облікового запису за допомогою адреси електронної пошти
U-2	log in	Авторизація у системі
U-3	logout	Вихід з облікового запису
U-4	restore account	відновлення доступу до облікового запису
U-4.1	reset token validation	валідація токена для скидання пароля
U-4.2	reset password	скидання пароля облікового запису
U-5	delete account	видалення облікового запису користувача
U-10	edit user account data	редагування інформації про обліковий запис користувача
U-10.1	update avatar	оновлення світлини облікового запису
U-10.2	edit user data	редагування інформації про користувача
U-10.3	change email	зміна адреси електронної пошти
U-10.4	change password	зміна пароля
U-6	provide user data	проходження анкети для надання даних про користувача
U-7	apply meal plan	формування та встановлення плану харчування
U-7.1	set macronutrient goals	встановлення цілей щодо кількості та пропорцій макронутрієнтів

Кінець таблиці 3.1

U-7.2	set calorie goals	встановлення цілей щодо щоденної кількості калорій
U-8	find food data	пошук даних про продукт харчування за допомогою стороннього API сервісу
U-8.1	search filters	пошукові фільтри для атрибутів продуктів харчування
U-8.2	add food item to daily log	додавання продукту харчування до щоденника харчування
U-9	view food item information	перегляд інформації про продукт харчування
U-9.1	edit food item data	редагування інформації про продукт харчування
U-9.2	remove food item from daily log	видалення продукту харчування із щоденника харчування
A-1	management user accounts	керування обліковими записами користувачів
A-1.1	create user account	створення нового облікового запису
A-1.2	view user account	перегляд інформації про обліковий запис
A-1.3	edit user account	редагування облікового запису
A-1.4	remove user account	видалення облікового запису
A-2	management meal plans	керування планами харчування
A-2.1	create meal plan	створення плану харчування
A-2.2	view meal plan	перегляд планів харчування
A-2.3	edit meal plan	редагування плану харчування
A-2.4	delete meal plan	видалення плану харчування

У таблицях 3.2, 3.3 докладно описується робота двох найважливіших сценаріїв даного вебзастосунку.

Таблиця 3.2 – Детальний опис роботи сценарію U-7

Діючі особи	Користувач
Мета	Вибрати індивідуальний план харчування
Передумова	Користувач авторизувався в системі та переходить на сторінку вибору плану харчування
Успішний сценарій: 1) При завантаженні сторінки клієнтська частина надсилає HTTP запит на сервер, використовуючи параметр ID користувача; 2) На серверній частині відбувається пошуку інформації про користувача в базі даних;	

Кінець таблиці 3.2

2) У базі даних знаходиться потрібний документ з інформацією про користувача; 3) На клієнтську частину повертається HTTP відповідь знайдених даних; 4) На клієнтській частині за допомогою математичних формул обчислюється TDEE та AMDR; 5) Користувач може змінити ціль щодо його бажаної ваги або встановити власне значення TDEE; 6) Користувач може змінити план дієти вибравши зі списку, що випадає; 7) Користувач може встановити нові пропорції макронутрієнтів або змінити їх щоденну кількість; 8) На клієнтській частині інтерфейс вебзастосунку моментально реагує на кожну зміну користувача та коригує TDEE та встановлює нові пропорційні співвідношення для макронутрієнтів та їх кількості; 9) Користувач натискає кнопку застосувати план живлення; 10) На клієнтській частині виконується певна валідація відповідно кількості макронутрієнтів зі значенням TDEE; 11) На клієнтській частині надсилається запит HTTP для оновлення даних про користувача; 12) На сервері виконується повторна валідація надісланих даних; 13) У базі даних оновлюється інформація про користувача; 14) На клієнтську частину надходить HTTP відповідь з оновленими даними про користувача та відповідне повідомлення; 15) Клієнтська частина перенаправляє користувача на сторінку щоденника харчування та виводить повідомлення про успішне формування нового плану харчування.	
Успішний результат	Користувач створив новий план харчування
Альтернативні сценарії: – введені дані користувачем не відповідають правилам та вимогам валідації (як на клієнті, так і на серверній частині); – в базі даних не було знайдено інформації про користувача; – при надсиланні форми реєстрації на сервері або в базі даних виникла непередбачена помилка.	
Альтернативний результат	Користувач не зміг застосувати вибраний план харчування та отримав відповідне повідомлення про помилку на одному з етапів

Таблиця 3.3 – Детальний опис роботи сценарію U-8

Діючі особи	Користувач
Мета	Знайти продукти харчування за допомогою певного запиту та використовуючи задані фільтри пошуку
Передумова	Користувач авторизувався в системі та знаходиться на головній сторінці
Успішний сценарій: 1) Користувач натискає кнопку «Add daily food»; 2) Діалогове вікно для пошуку їжі стає активним; 3) Користувач вводить назву продукту харчування у поле для пошуку; 4) Користувач натискає на іконку «Search filters»; 5) Діалогове вікно для вибору фільтрів для пошуку стає активним; 6) Користувач встановлює всі необхідні фільтри пошуку, які задовольняють запит; 7) Користувач натискає кнопку «Search»; 8) На стороні клієнта формується та надсилається HTTP запит до стороннього API сервісу, який надає дані про їжу; 9) Використовуваний API сервіс повертає у вигляді HTTP відповіді дані із знайденими продуктами харчування які відповідають запиту користувача; 10) На стороні клієнта формується масив з отриманих даних; 11) Інтерфейс вебзастосунку відображає дані про продукт харчування у вигляді списку.	
Успішний результат	Користувач отримав відповідний список продуктів харчування, які задовольняють його умовам пошуку
Альтернативні сценарії: API сервіс, що використовується, не знайшов жодного збігу за запитом користувача серед усіх наявних продуктів у базі даних	
Альтернативний результат	Користувач отримує повідомлення про відсутність знайдених продуктів

Опис сценаріїв є важливим аспектом для розуміння роботи функціоналу вебзастосунку та необхідних кроків при подальшій його реалізації.

3.3 Проєктування бази даних та опис призначення колекцій

Діаграма зв’язків між об’єктами (ER-diagram) – це тип блок-схеми, яка ілюструє, як «суб’єкти», такі як люди, об’єкти або поняття, пов’язані один з одним

у системі. Діаграми ER використовуються для моделювання та проєктування реляційних баз даних з точки зору логіки та бізнес-правил (в логічній моделі даних) і з точки зору конкретної технології, яка має бути реалізована (у фізичній моделі даних). У програмній інженерії діаграма ER часто є початковим кроком у визначенні вимог до проєкту інформаційних систем. Також використовується для моделювання певної бази даних. Реляційна база даних має еквівалентну реляційну таблицю і потенційно може бути виражена таким чином за потреби [27].

Незважаючи на те, що в якості бази даних обрана нереляційна база даних, доцільно змоделювати її фізичну модель (рис. 3.2) для демонстрації зв'язків між колекціями.

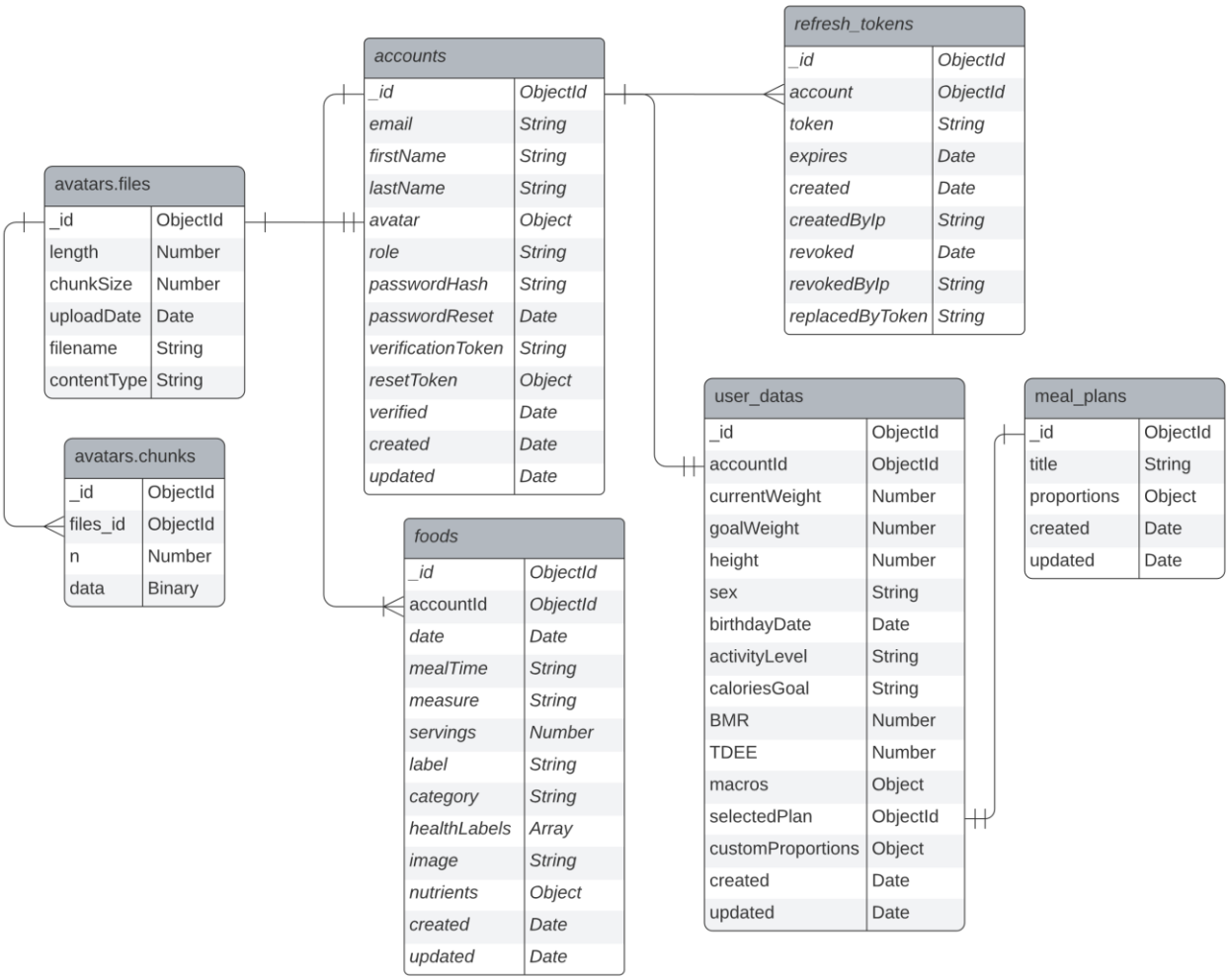

Рисунок 3.2 – Фізична модель бази даних

Оскільки фізична модель бази даних показує лише назви використовуваних колекцій та їх атрибути доцільно описати призначення кожної з них (табл. 3.4).

Таблиця 3.4 – MongoDB колекції, що використовуються та їх призначення

Назва	Призначення
accounts	зберігає інформацію про облікові записи користувачів
avatars.files	зберігає метадані для світлин користувачів
avatars.chunks	зберігає розбиті на шматки бінарні дані світлин користувачів
foods	зберігає інформацію про додані за допомогою API сервісу або створені вручну продукти харчування
meals_plans	зберігає інформацію про наявні дієтичні плани харчування та пропорційні співвідношення макронутрієнтів
user_datas	зберігає персональну інформацію користувача для створення та управлінням індивідуального плану харчування
refresh_tokens	зберігає дані про токени, що оновлюється

При проектуванні бази даних використовувалося 7 колекцій, проте при реалізації моделей буде використовуватися лише 5, так як колекції з назвою avatars створюються автоматично і не потребують моделей.

3.4 Визначення архітектури та концепції роботи вебзастосунку

Наявність відправної точки, коли справа доходить до архітектури проєкту, є життєво важливою для реалізації проєкту та того, як можна задовольняти мінливі потреби в майбутньому. Погана, брудна архітектура проєкту часто **призводить до**:

– нечитабельного і брудного коду, який подовжує та ускладнює процес розробки;

– безкорисне повторення, яке ускладнює підтримку та керування кодом;

– труднощі із впровадженням нових функцій без зіпсування існуючого коду;

Основна мета будь-якої структури проєкту **допомогти**:

– написати чистий і читабельний код;

– написати багаторазові фрагменти коду;

– уникнути повторень;

– додавати нові функції, не порушуючи існуючого коду.

Розділення проблем (Separation of Concerns) є однією з найважливіших концепцій, яку розробник програмного забезпечення повинен засвоїти, також це одне з найважчих для опису, оскільки це абстракція абстракції [28].

У вебпрограмуванні існує принцип дизайну, який говорить, що реалізацію вебзастосунку слід **розділити на три домени** [28]:

– **Стиль і презентація** – візуальний вигляд сайту;

– **Бізнес-логіка** – спосіб поведінки сайту у відповідь на дії користувача;

– **Вміст** – фактичні дані, які представлені, як-от дописи в блозі чи статті.

Дотримуючись цих правил, можна було б намагатися уникнути змішування бізнес-логіки та презентації, і те саме стосується контенту. Якщо можливо, ці елементи містилися б у різних джерелах або файлах даних, інакше вони будуть у окремих і легко ідентифікованих розділах того самого файлу.

Для цього є ряд вагомих **причин** [28]:

– елементи часто змінюються самостійно;

– з різними елементами часто пов’язані різні командні ролі.

– реалізації на чітко визначені частини полегшує пошук речей.

Загалом, усі ці причини відповідають загальному принципу **«зробити код легким для розуміння»**.

Виходячи з принципу поділу проблем, мета полягає в тому, щоб повністю виділити і відокремити бізнес-логіку від API. Зокрема, зробити так, щоб бізнес-логіка ніколи не була присутня в маршрутах або контролерах (рис. 3.3).

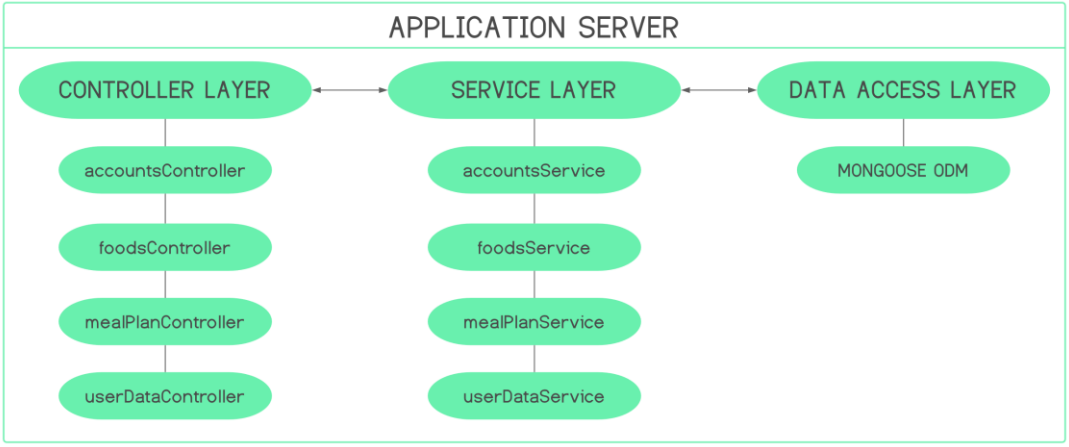

Рисунок 3.3 – Архітектура роботи серверної частини проєкту

При моделюванні архітектури серверної частини проєкту на підставі розглянутого принципу були визначені наступні домовленості:

– **рівень контролеру (Controller Layer)** – відповідає за прийняття вхідних HTTP запитів, перевірку їх належної аутентифікації та форматування, а потім передачу їх на рівень обслуговування для обробки (результати можуть передатися назад клієнту).

– **рівень обслуговування (Service Layer)** – містять всю бізнес-логіку, а також можуть здійснювати виклики на рівень доступу до даних (може потім передати все назад на рівень контролеру);

– **рівень доступу до даних (Data Access Layer)** – бере на себе відповідальність за спілкування з базою даних – отримання, запис, оновлення та видалення (результати передаються назад на рівень сервісу).

Робота клієнтської частини застосунку побудована на реалізації **Vue компонентів** та використанні концепції **«State Management Pattern».**

Кожен екземпляр компонента Vue під час створення проходить серію кроків ініціалізації – наприклад, йому потрібно налаштувати спостереження за даними, зібрати шаблон, підключити екземпляр до DOM та оновити DOM при зміні даних. Попутно він також запускає функції, які називаються хуками життєвого циклу (рис. 3.4), що дає користувачам можливість додавати власний код на певних етапах [29].

Наприклад, mounted хук можна використовувати для запуску коду після того, як компонент завершив початковий рендеринг і створив вузли DOM. Існують також інші хуки, які будуть викликані на різних етапах життєвого циклу екземпляра, причому найбільш часто використовувані з них mounted, updated та unmounted.

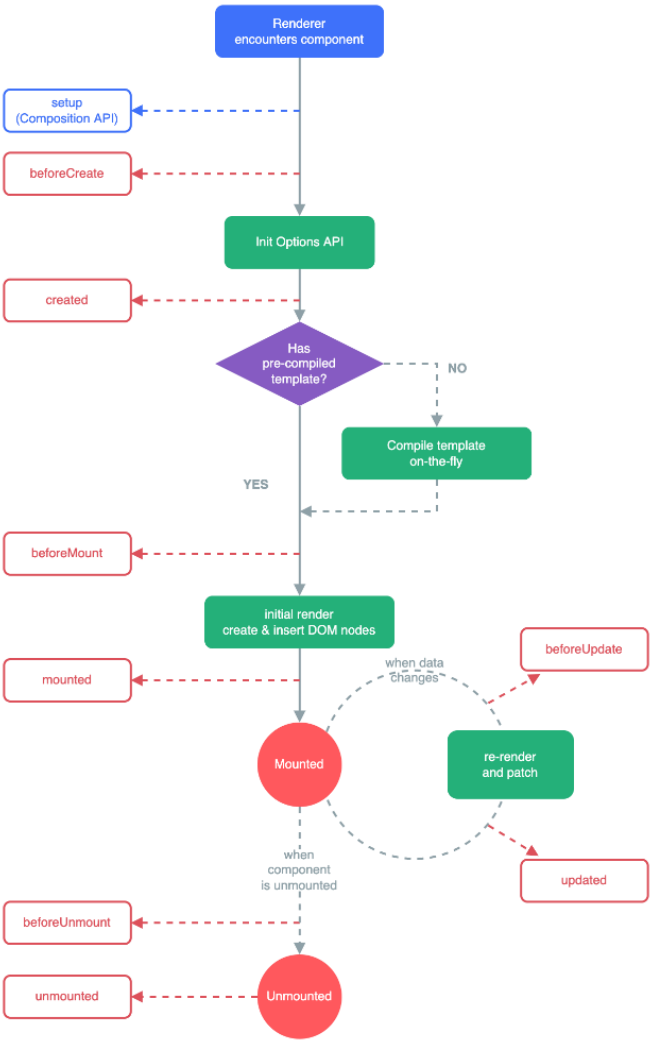

Рисунок 3.4 – Діаграма хуків життєвого циклу фреймворку Vue.js

Для реалізації «State Management Pattern» використовується Vuex [30].

Vuex – це шаблон керування станом і бібліотека для застосунків Vue.js. Він служить централізованим сховищем для всіх компонентів застосунка, з правилами, які гарантують, що стан можна змінювати лише передбачуваним чином [30].

Це автономний застосунок із такими частинами [30]:

– **стан (state):** джерело стану, що керує застосунком;

– **представлення (view):** декларативне відображення стану;

– **дії (actions):** можливі способи зміни стану у відповідь на введення користувача з представлення.

На рисунку 3.5 наведено просте представлення концепції «одностороннього потоку даних».

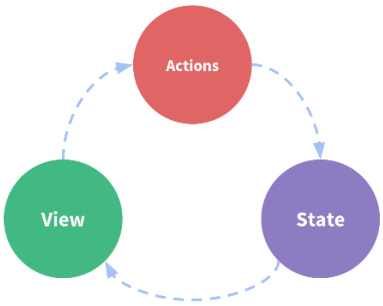

Рисунок 3.5 – Представлення концепції «одностороннього потоку даних»

Однак простота швидко руйнується, коли **є кілька компонентів, які мають загальний стан** [30]:

– кілька представлень можуть залежати від одного і того ж стану;

– дії з різних поглядів можуть вимагати мутації одного і того ж стану.

Для першої проблеми передача props може бути стомлюючою для глибоко вкладених компонентів і просто не працює для компонентів-побратимів. Для другої проблеми приходимо до таких рішень, як прямі посилання на батьківський/дочірній екземпляр або намагаємося змінити та синхронізувати декілька копій стану за допомогою подій. Обидва ці моделі є крихкими і швидко призводять до непридатного для обслуговування коду [30].

Тому є потреба витягнути спільний стан з компонентів і не керувати ним у глобальному синглтоні. Завдяки цьому дерево компонентів стає великим «представленням», і будь-який компонент може отримати доступ до стану або ініціювати дії, незалежно від того, де вони знаходяться в дереві [30].

Визначаючи та відокремлюючи концепції, пов’язані з управлінням станом, і дотримуючись правил, які підтримують незалежність між представленнями та станами, надаємо коду більшу структурність і можливість підтримки [30].

На рисунку 3.6 візуалізовано повну концепцію роботи «State Management Pattern», яка є основою ідеєю Vuex, натхненною Flux, Redux та The Elm Architecture. На відміну від інших шаблонів, Vuex також є реалізацією бібліотеки, розробленою спеціально для Vue.js, щоб скористатися перевагами його детальної системи реактивності для ефективних оновлень [30].

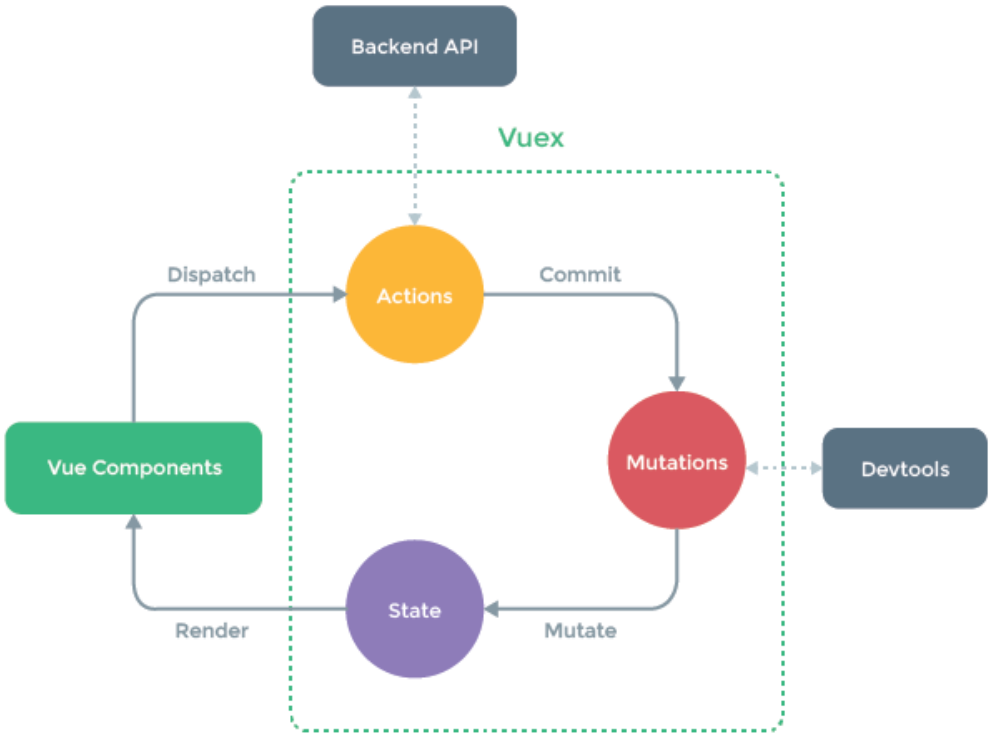

Рисунок 3.6 – Повна концепція роботи «State Management Pattern»

Маючи повне розуміння роботи вебзастосунку з'являється можливість реалізувати діаграму розгортання (рис. 3.7).

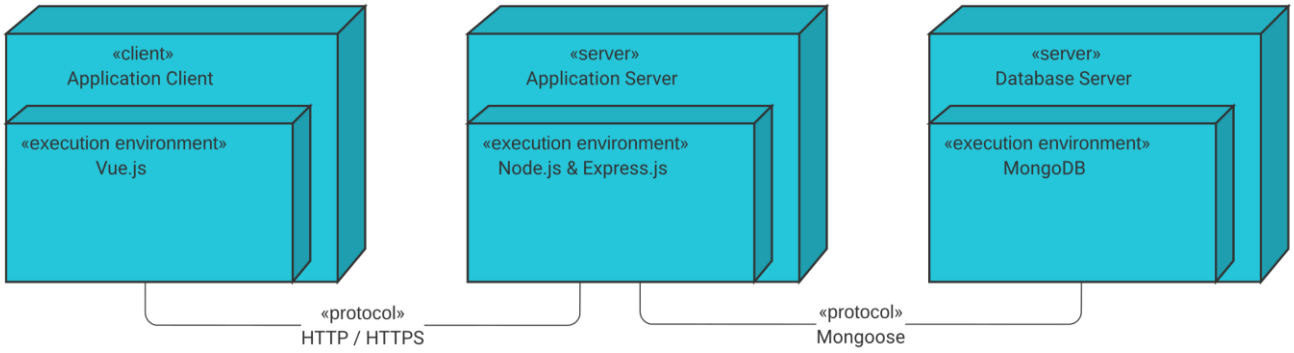

Рисунок 3.7 – Діаграма розгортання вебзастосунку

Змодельована діаграма розгортання візуалізує роботу програмного забезпечення вебзастосунку, що розробляється і допомагає зрозуміти як дана система може бути фізично розміщена на хостингу.

3.5 Проєктування файлової структури вебзастосунку

Реалізувавши файлову структуру проєкту доцільно описати призначення кореневих директорій та файлів (табл. 3.5, 3.6).

Таблиця 3.5 – Опис файлової структури для серверної частини

Назва	Призначення
controllers	містить відповідні контролери, методи якого отримують запити від маршрутів і перетворюють їх у відповіді HTTP з використанням middleware, якщо необхідно
logs	містить лог файли для збереження історії оброблених запитів та помилок
middlewares	відокремлює middleware, необхідне для коректної роботи застосунку (аутентифікація, перехоплення та обробка помилок, валідація запитів) в одному місці
models	моделі даних, визначені відповідно до вимог відображення об'єктного документа (ODM)
routes	логічний набір маршрутів
services	включає всю бізнес-логіку, яка виконує запити до бази даних
utils	містить усі утиліти та помічники
.env	зберігає секретні ключі або інші дані для налаштування сервера
package.json	ключовий файл на основі екосистеми Node.js та NPM
server.js	головний файл, з якого починається виконання під час запуску сервера

Таблиця 3.6 – Опис файлової структури для клієнтської частини

Назва	Призначення
public	містить базовий файл index.html та інші публічні дані
assets	зберігає допоміжні для реалізації файли (шрифти, зображення, іконки)
components	містить багаторазові компоненти, такі як: діаграми, діалогові вікна, форми, повідомлення, списки
layouts	містить шаблони, на основі якого працює UI
plugins	містить налаштовані плагіни, наприклад Vuetify
router	місце, де налаштовуються всі маршрути, а також навігаційна безпека
services	сервіси, які взаємодіють з різними API
store	централізоване сховище для всіх компонентів застосунку
utils	містить усі утиліти та помічники
views	UI представлення для всіх встановлених маршрутів (admin, auth, daily-log, onboarding, settings)
App.vue	головний Vue-компонент, з якого починається відображення інтерфейсу

Кінець таблиці 3.6

main.js	головний файл клієнтської частини, де ініціалізується новий об'єкт Vue
.env	зберігає секретні ключі або інші дані для налаштування клієнта
package.json	ключовий файл на основі екосистеми Node.js та NPM
vue-config.js	конфігураційний файл для налаштування Vue застосунку

Реалізована файлова структура є інтуїтивно простою та дозволяє швидко перейти до необхідного шматка коду, що робить проєкт добре масштабованим, підтримуваним та розширюваним у міру його збільшення.

Висновки до розділу 3

У третьому розділі була розроблена діаграма варіантів використання, яка наочно показує як повинні використовуватися функціональні вимоги до вебзастосунку, що розробляється. На підставі даної діаграми були складені та описані деякі сценарії використання, які детально показують етапи роботи за успішних та альтернативних умов.

Крім цього, у даному розділі також була змодельована ER-діаграма, яка показує фізичний зв'язок моделей між колекціями у базі даних MongoDB. До кожної колекції було коротко описано її призначення.

Для більш візуального розуміння роботи вебзастосунку була змодельована архітектура роботи серверної частини, яка ґрунтується на концепції «Розділення проблем». Робота клієнтської частини застосунку побудована на реалізації Vue компонентів та використанні концепції «State Management Pattern».

Саме після того як була реалізована файлова структура проєкту, як головний висновок можна сказати, що вже все готове для того, щоб приступити до програмної реалізації проєкту.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Програмна реалізація вебзастосунку поділяється на створення серверної та клієнтської частини. Серверна частина включає створення API і обробку HTTP запитів, що надходять шляхом взаємодії з базою даних. Клієнтська частина відповідає за створення UI для візуального представлення та керування використовуваних даних як зі свого, так і зі стороннього API сервісу.

4.1 Реалізація серверної частини

Створення серверної частини починається з підключення всіх зовнішніх бібліотек та залежностей, необхідних для подальшої розробки та правильної роботи сервера за допомогою NPM (табл. 4.1).

Таблиця 4.1 – Використовувані залежності, їх версії та призначення

Залежність	Версія	Призначення
bcryptjs	2.4.3	Шифрування пароля
body-parser	1.20.0	Аналіз тіла вхідних запитів
cookie-parser	1.4.6	Аналіз заголовків Cookie і заповнення req.cookies об’єктом, ключем якого є імена файлів cookie
cors	2.8.5	Надає middleware Connect/Express, яке можна використовувати для ввімкнення CORS з різними параметрами
dotenv	16.0.1	Завантажує змінні середовища з файлу .env у process.env
express	4.18.1	Вебфреймворк для Node.js
express-jwt	7.7.5	Забезпечує middleware Express для перевірки JWT через модуль jsonwebtoken
joi	17.6.0	Мова опису схем і перевірка даних для JavaScript
jsonwebtoken	8.5.1	Реалізація вебтокенів JSON
mongodb	4.7.0	Офіційний драйвер MongoDB для Node.js
mongoose	6.3.6	Інструмент моделювання об’єктів MongoDB, призначений для роботи в асинхронному середовищі
multer	1.4.4	Обробка даних із multipart/form-data
multer-gridfs-storage	5.0.2	Механізм зберігання GridFS для Multer для зберігання завантажених файлів безпосередньо в MongoDB

Кінець таблиці 4.1

multer-gridfs-storage	5.0.2	Механізм зберігання GridFS для Multer для зберігання завантажених файлів безпосередньо в MongoDB
nodemailer	6.7.5	Надсилання електронних листів прямо з коду Node.js
rootpath	0.1.2	Робить всі імпорти абсолютними до кореневого каталогу проєкту
swagger-ui-express	4.4.0	Дозволяє обслуговувати автоматично згенеровані swagger-ui документи API з express на основі файлу swagger.json
winston	3.7.2	Логер майже для всього
yamljs	0.3.0	Автономний аналізатор і кодер JavaScript YAML 1.2, містить інструменти перетворення YAML/JSON командного рядка
nodemon	2.0.16	допомагає розробляти застосунки на основі Node.js шляхом автоматичного перезапуску, коли виявлено зміни файлів у каталозі

Всі залежності, включаючи їх версії, а також інші метадані зберігаються у файлі package.json у вигляді об'єкта JSON.

Налаштування middleware-функцій

Middleware можна додати до будь-якого маршруту, щоб обмежити доступ до маршруту автентифікованим користувачам із зазначеними ролями. Якщо параметр roles опущено, маршрут буде доступним для всіх аутентифікованих користувачів незалежно від ролі. Він використовується контролером для обмеження доступу до захищених маршрутів (рис. 4.1).

```
function authorize(roles : any[] = []) {
  const secret = process.env.SECRET
  if (typeof roles === 'string') roles = [roles]
  return [
    expressjwt( options: { secret, algorithms: ['HS256'] }),
    async (req, res, next) => {
      const account = await db.Account.findById(req.auth.id);
      const refreshTokens = await db.RefreshToken.find( filter: { account: account.id });
      if (!account) {
        return res.status(401).json({ message: 'Account no longer exists' });
      } else if (roles.length && !roles.includes(account.role)) {
        return res.status(401).json({ message: 'Denied access' });
      }
      req.auth.role = account.role;
      req.auth.ownsToken = token => !!refreshTokens.find( filter: x => x.token === token);
      next();
    }
  ];
}
```

Рисунок 4.1 – Middleware-функція authorize для захисту маршрутів

Функція authorize повертає масив, що містить дві middleware-функції:

– перша (expressjwt) аутентифікує запит, перевіряючи JWT у заголовку «Authorization» запиту HTTP. Після успішної аутентифікації об’єкт auth додається до об’єкта req, який містить дані з токена JWT, який у цьому прикладі включає ідентифікатор користувача (req.auth.id);

– друга авторизує запит, перевіряючи, що аутентифікований обліковий запис все ще існує та має дозвіл на доступ до запитуваного маршруту на основі його ролі. Друга middleware-функція також додає властивість role та метод ownsToken до об’єкта req.auth, щоб до них могли отримати доступ методи контролерів.

Якщо аутентифікація або авторизація не вдалися, повертається відповідь 401 з відповідним повідомленням.

Глобальний обробник помилок (errorHandler) використовується для лову всіх помилок і усуває потребу в дубльованому коді обробки помилок. Він налаштований як middleware в головному файлі server.js (рис. 4.2).

```
function errorHandler(err, req, res, next) {
  switch (true) {
    case typeof err === 'string':
      const is404 = err.toLowerCase().endsWith('not found');
      const statusCode = is404 ? 404 : 400;
      return res.status(statusCode).json({ message: err });
    case err.name === 'ValidationError':
      return res.status(400).json({ message: err.message });
    case err.name === 'UnauthorizedError':
      return res.status(401).json({ message: 'Unauthorized' });
    default:
      return res.status(500).json({ message: err.message });
  }
}
```

Рисунок 4.2 – Middleware-функція errorHandler для обробки помилок

Перехоплюються такі типи помилок:

– **користувацькі:** визначені у сервісі або контролері та мають тип string. Якщо користувальницька помилка закінчується словами "не знайдено", повертається код відповіді 404, інакше повертається стандартна відповідь 400;

– **валідації:** відноситься до mongoose та joi та повертає код відповіді 400;

– **авторизації:** відноситься до неавторизованих користувачів та повертає код відповіді 401;

– **серверні:** за замовчуванням повертає код відповіді 500.

Помилки фіксуються в контролерах для кожного маршруту та передаються до next(err), який передає їх цьому глобальному обробнику помилок.

Middleware-функція validateRequest перевіряє тіло запиту щодо об’єкта схеми Joi (рис. 4.3).

```
function validateRequest(req, next, schema) {
  const options = {
    abortEarly: false,
    allowUnknown: true,
    stripUnknown: true
  };
  const { error, value } = schema.validate(req.body, options);
  if (error) {
    next(error.details.map(x => x.message).join(', '));
  } else {
    req.body = value;
    next();
  }
}
```

Рисунок 4.3 – Middleware-функція validateRequest для перевірки валідації запиту

Він використовується middleware-функціями валідації схеми в контролерах для перевірки запиту щодо схеми для певного маршруту (наприклад, authenticateSchema в контролері облікових записів).

Налаштування бази даних

Підключення до бази даних MongoDB відбувається за допомогою інструменту mongoose. Використовується допоміжна функція isValidId(), щоб увімкнути перевірку, чи є ідентифікатор дійсним MongoDB ObjectId, перш ніж спробувати виконати запит (рис. 4.4).

```
mongoose.connect(process.env.MONGODB_URI).then(() => {
  mongoose.Promise = global.Promise
  logger.info( message: 'Connected to MongoDB successfully')
}).catch(error => logger.error(Promise.reject(error)));

const isValidId = id => mongoose.Types.ObjectId.isValid(id)
```

Рисунок 4.4 – Використання методів інструменту mongoose

За допомогою об'єкта GridFsStorage зберігаються світлини профілів користувачів і експортуються у вигляді параметра для multer, який обробляє дані форм з маршруту (рис. 4.5).

```
const storage = new GridFsStorage({
  url: process.env.MONGODB_URI,
  file: (req, file) => {
    return new Promise( executor: (resolve, reject) => {
      crypto.randomBytes( size: 16, callback: (err : Error | null , buf : Buffer ) => {
        if (err) return reject(err);
        const filename = buf.toString( encoding: 'hex') + path.extname(file.originalname);
        const fileInfo = {
          filename,
          bucketName: 'avatars'
        };
        resolve(fileInfo);
      });
    });
  }
});
module.exports = multer( options: { storage });
```

Рисунок 4.5 – Обробка збереження світлин користувачів у базі даних

Налаштована база даних забезпечує простий спосіб отримати доступ до будь-якої частини бази даних з однієї точки.

Створення моделей для взаємодій із базою даних

Для опису схеми моделей використовується об'єкт Schema, що надає інструмент mongoose. Схема визначає властивості в MongoDB для записів певних моделей (рис. 4.6).

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const schema = new Schema({
  email: { type: String, unique: true, required: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  avatar: { id: { type: Schema.Types.ObjectId,  ref: 'avatars.files' }, filename: String },
  role: { type: String, required: true },
  passwordHash: { type: String, required: true },
  passwordReset: Date,
  verificationToken: String,
  resetToken: { token: String, expires: Date },
  verified: Date,
  created: { type: Date, default: Date.now },
  updated: Date,
});
```

Рисунок 4.6 – Опис схеми моделі Account

Віртуальні властивості є зручними властивостями, доступними в моделі mongoose, які не зберігаються в MongoDB (рис. 4.7).

schema.set('toJSON', { ... }) налаштовує, які властивості облікового запису включаються під час перетворення записів MongoDB в об’єкти JSON:

– **virtuals:** true включає властивість віртуального ідентифікатора Mongoose, яка є копією властивості MongoDB _id;

– **versionKey:** false виключає ключ версії Mongoose (__v);

– **transform:** видаляє властивості MongoDB під час перетворення записів у JSON.

```
schema.virtual( name: 'isVerified').get(function () {
  return !!(this.verified || this.passwordReset);
});

schema.set( key: 'toJSON', value: {
  virtuals: true,
  versionKey: false,
  transform: function (doc, ret) {
    delete ret._id;
    delete ret.passwordHash;
  }
});
```

Рисунок 4.7 – Налаштування схеми моделі Account

За цим принципом створюються всі інші необхідні моделі: Refresh_Token, User_Data, Food, Meal_Plan.

Налаштування маршрутизації для створення кінцевих точок

Для створення системи маршрутизації використовується клас express.Router(), який створює модульний обробник маршрутів та використовує middleware- функцію authorize(). Об'єкт класу Router використовує один із типів HTTP запиту (в даному випадку використовуються GET, POST, PUT, DELETE) як метод і встановлює шлях маршруту та метод, який обробляє цей запит (рис. 4.8).

```
const express = require('express');
const router = express.Router();
const accountController = require("controllers/accounts.controller");
const Role = require("utils/role");
const uploadAvatar = require('utils/avatars-storage');
const authorize = require("middlewares/authorize");

router.post( path: '/authenticate', accountController.authenticateSchema, accountController.authenticate);
router.post( path: '/refresh-token', accountController.refreshToken);
router.post( path: '/register', accountController.registerSchema, accountController.register);
router.post( path: '/verify-email', accountController.verifyEmailSchema, accountController.verifyEmail);
router.post( path: '/forgot-password', accountController.forgotPasswordSchema, accountController.forgotPassword);
router.post( path: '/validate-reset-token', accountController.validateResetTokenSchema, accountController.validateResetToken);
router.post( path: '/reset-password', accountController.resetPasswordSchema, accountController.resetPassword);
router.post( path: '/revoke-token', authorize(), accountController.revokeTokenSchema, accountController.revokeToken);
router.get( path: '/', authorize(Role.Admin), accountController.getAll);
router.post( path: '/', authorize(Role.Admin), accountController.createSchema, accountController.create);
router.get( path: '/:id', authorize(), accountController.getById);
router.put( path: '/:id', authorize(), accountController.updateSchema, accountController.update);
router.delete( path: '/:id', authorize(), accountController.delete);
router.post( path: '/upload/:id', authorize(), uploadAvatar.single( fieldName: 'file'), accountController.uploadAvatar);
router.get( path: '/avatar/:id', accountController.getAvatar);
router.get( path: '/avatar/:id/:filename', accountController.updatedAvatar);
router.delete( path: '/avatar/:id', authorize(), accountController.deleteAvatar);
```

Рисунок 4.8 – Маршрутизація кінцевих точок входу для моделі Account

Маршрути, які вимагають авторизації, виконують middleware-функцію authorize() і, за потребою, вказують роль (наприклад, authorize(Role.Admin), якщо вказана роль, то маршрут обмежується користувачами в цій ролі, інакше маршрут обмежено всіма автентифікованими користувачами незалежно від ролі.

Маршрути, які потребують перевірки схеми, включають middleware-функцію з умовою найменування <route>Schema (наприклад, authenticateSchema). Кожна функція перевірки схеми визначає схему для тіла запиту за допомогою бібліотеки Joi і викликає validateRequest(req, next, schema), щоб переконатися, що тіло запиту дійсне. Якщо перевірка успішна, запит продовжується до наступної функції маршруту, інакше повертається помилка з подробицями про те, чому не вдалося виконати перевірку.

Створення контролерів для обробки запитів

Контролер складається з методу, який прив'язаний до конкретного маршруту, обробляє отриманий запит, після чого передає сервісу ті дані, які необхідні для взаємодії з моделями базою даних.

Кожен метод контролера складається з трьох наступних аргументів:

– **req:** аргумент HTTP запиту;

– **res:** аргумент HTTP відповіді;

– **next:** аргумент зворотного виклику.

У деяких ситуаціях (рис. 4.9) перед обробкою запиту методом контролера може відбуватися певна валідація даних за допомогою бібліотеки joi та її інструментів.

```
const registerSchema = (req, res, next) => {
  const schema = Joi.object( schema: {
    firstName: Joi.string().required(),
    lastName: Joi.string().required(),
    email: Joi.string().email().required(),
    password: Joi.string().min( limit: 6).required(),
    passwordConfirm: Joi.string().valid(Joi.ref( key: 'password')).required()
  });
  validateRequest(req, next, schema);
}

const register = (req, res, next) => {
  accountService.register(req.body, req.get('origin')).then(response => res.json(response)).catch(next);
}
```

Рисунок 4.9 – Обробка запиту реєстрації облікового запису

Також, якщо певний маршрут включають додаткову спеціальну перевірку авторизації (рис. 4.10), щоб запобігти доступу користувачів без прав адміністратора до інших облікових записів, крім власних.

```
const update = (req, res, next) => {
  if (req.params.id !== req.auth.id && req.auth.role !== Role.Admin) {
    return res.status(401).json({ message: 'Denied access' });
  }
  userDataService.update(req.params.id, req.body).then(userData => res.json(userData)).catch(next);
}
```

Рисунок 4.10 – Обробка запиту оновлення даних користувача

Таким чином, звичайні облікові записи користувачів (Role.User) мають доступ CRUD до свого власного облікового запису, але не до інших, а облікові записи адміністратора (Role.Admin) мають повний доступ CRUD до всіх облікових записів.

Реалізація сервісів для взаємодії з базою даних

Сервіси містять основну бізнес-логіку застосунку, що складається з методів. Методи сервісів приймають параметри з контролерів і застосовуються для взаємодії з даними у базі даних (рис. 4.11).

```
const _delete = async (id) => {
  const account = await getAccount(id)
  await db.Food.deleteMany( filter: { accountId: id })
  await db.RefreshToken.deleteMany( filter: { account: id })
  await db.UserData.findOneAndDelete( filter: { accountId: id })
  await account.remove()
  const message = 'Account deleted successfully'
  logger.info(message)

  return { message }
}
```

Рисунок 4.11 – Метод _delete для видалення облікового запису з бази даних

На рис. 4.11 продемонстрована робота методу _delete, яка асинхронно (за допомогою async/await) знаходить обліковий запис з відповідним параметром id, після чого видаляє всі наявні дані про нього з бази даних, додає в лог повідомлення про успішне видалення і повертає його у вигляді Promise, який обробляє метод контролера.

Налаштування надсилання листів на електронну пошту

Для надсилання шаблонів електронних листів використовується встановлений інструмент nodemailer та його методи createTransport, sendMail (4.12).

```
const smtpOptions = {
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS
  }
}

async function sendEmail({ to, subject, html, from = process.env.EMAIL_FROM }) {
  const transporter = createTransport(smtpOptions);
  await transporter.sendMail( data: { from, to, subject, html });
}
```

Рисунок 4.12 – Налаштування надсилання електронних листів

Для надсилання використовуються SMTP параметри та інформація про лист (від кого, кому, тема, HTML шаблон).

4.2 Реалізація клієнтської частини

Для реалізації клієнтської частини необхідно виконати низку наступних етапів:

– підключити NPM залежності та бібліотеки;
– налаштувати маршрутизацію та навігацію;
– налаштувати роботу HTTP запитів;
– створити сервіси для надсилання HTTP запитів;
– реалізувати управління станом застосунка;
– налаштувати UI та створити всі необхідні компоненти.

Як і у випадку з реалізацією серверної частини, всі необхідні залежності та бібліотеки для реалізації клієнтської частини використовується NPM та його інструменти.

Таблиця 4.2 – Використовувані залежності, їх версії та призначення

Залежність	**Версія**	**Призначення**
axios	0.27.2	відправлення та обробка HTTP запитів на основі Promise
chart.js	3.7.1	налаштування компонентів для створення діаграм
moment	2.29.3	аналізування, перевірка, маніпуляція та форматування дат.
qs	6.10.3	аналізування рядків запитів і визначення рядків із додатковим захистом
rxjs	7.5.5	використання BehaviorSubject, щоб полегшати складання асинхронного коду або коду на основі зворотного виклику
uuid	8.3.2	створення універсального унікального ідентифікатора
vue	2.6.14	прогресивний фреймворк JavaScript для створення вебінтерфейсу
vue-chartjs	4.1.0	обгортка Vue.js для chart.js для створення красивих діаграм
vue-round-slider	1.0.1	створення пів кружного повзунка для Vue.js з підтримкою діапазону
vue-meta	2.4.0	керування метаданими HTML у компонентах Vue.js
vue-router	3.5.3	створення маршрутизації та навігаційної безпеки
vuelidate	0.7.7	валідація даних на основі моделей
vuetify	2.6.5	створення UI використовуючи Material Components
vuex	3.6.2	керування станом застосунку Vue.js

У наведеній таблиці описані залежності, що використовуються тільки для production (знаходяться в об'єкті dependencies).

Налаштування маршрутизації та навігації

Перед тим як перейти до розробки необхідного функціоналу, спочатку потрібно створити кастомні функції для зручної перевірки терміну життя JWT, авторизації та отримання даних про cookie, користувача та налаштувань застосунку з localStorage (рис. 4.13).

```
export const isAuthorized = () => {
  const cookie = getCookie( name: 'refreshToken')
  const user = getUser()
  const token = getToken()
  return !!(cookie && user && token)
}

export const getCookie = (name) => {
  let matches = document.cookie.match(new RegExp( pattern: "(?:^|; )" + name.replace(/([\\.$?*|{}()[\]/+^])/g, '\\$1') + "=([^;]*)"))
  return matches ? decodeURIComponent(matches[1]) : null
}

export const getApplication = () => JSON.parse(localStorage.getItem( key: 'application')) || null

export const getUser = () => {
  const user = JSON.parse(localStorage.getItem( key: 'user'))
  return user?.id && (user?.role === Role.Admin || user?.role === Role.User) ? user : null
}

export const getToken = () => {
  const jwtToken = JSON.parse(localStorage.getItem( key: 'jwtToken'))
  return jwtToken?.token && jwtToken?.expires ? jwtToken : null
}

export const isTokenExpired = expires => expires < new Date().getTime()

export const tokenExpirationTime = ({ jwtToken }) => {
  const token = JSON.parse(atob(jwtToken.split('.')[1]))
  return new Date( value: token.exp * 1000).getTime()
}
```

Рисунок 4.13 – Реалізація кастомних функцій

Для встановлення створених маршрутів використовується об'єкт інструменту VueRouter, де встановлюємо необхідні параметри (рис. 4.14).

```
const router = new VueRouter( options: {
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})
```

Рисунок 4.14 – Встановлення створених маршрутів

Для захисту навігації шляхом переспрямування створюється глобальний навігаційний guard, яка викликається в порядку створення, щоразу, коли виконується навігація за застосунку (рис. 4.15).

```
router.beforeEach( guard: (to : Route , from : Route , next : NavigationGuardNext<Vue> ) => {
  const { authorized } = to.meta
  const user = getUser()
  const token = getToken()
  if (authorized && !isAuthorized()) {
    if (user || token) store.commit( type: 'account/LOGOUT_USER')
    return next( to: { name: 'Login', query: { returnUrl: to.path } })
  } else if (authorized?.length && !authorized.includes(user.role)) {
    return next( to: { name: 'DailyLog' })
  } else {
    return next()
  }
})
```

Рисунок 4.15 – Глобальний навігаційний guard

Глобальний навігаційний guard має три аргументи:

– **to:** цільове розташування маршруту в нормалізованому форматі, до якого здійснюється навігація;

– **from:** поточне місцезнаходження маршруту в нормалізованому форматі, від якого потрібно перейти;

– **next:** викликається рівно один раз під час будь-якого проходження через навігаційну охорону.

При виконанні навігації перевіряється, чи захищений маршрут до якого виконується навігація, інакше викликається next() і користувач успішно переходить на потрібний маршрут. У випадку, якщо маршрут захищений, а користувач не авторизований, виконується перехід на сторінку авторизації (Login), інакше якщо маршрут має будь-яку роль (Admin), а у поточного користувача вона відсутня, то користувач потрапляє на сторінку щоденника харчування (DailyLog).

Налаштування axios для роботи із запитами

У зв'язку з тим, що в даному проєкті використовуються два різні API (серверна частина та сторонній сервіс) доцільно створити також 2 екземпляра axios з різними налаштуваннями для зручнішого використання при надсиланні HTTP запитів (рис. 4.16).

```
const foodDatabaseConfig = {
  baseURL: process.env.VUE_APP_FOOD_DATABASE_API_URL,
  params: {
    app_id: process.env.VUE_APP_FOOD_DATABASE_API_APP_ID,
    app_key: process.env.VUE_APP_FOOD_DATABASE_API_APP_KEY
  }
}

export const FoodDatabaseAPI = axios.create(foodDatabaseConfig)

const defaultConfig = {
  baseURL: process.env.VUE_APP_BASE_API_URL,
  withCredentials: true,
  headers: {
    'Content-Type': 'application/json',
    'Access-Control-Allow-Private-Network': true
  }
}

export const DefaultAPI = axios.create(defaultConfig)
```

Рисунок 4.16 – Створення екземплярів axios для роботи з 2 різними API

Для запобігання проблемам при роботі з JWT (закінчення терміну життя через вказаний час) бібліотека axios має перехоплювачі (interceptors), які будуть викликатись перед або після виконання запиту та повертати Promise (рис. 4.17).

```
DefaultAPI.interceptors.request.use( onFulfilled: async (config : AxiosRequestConfig ) => {
  const token = getToken()
  const cookie = getCookie( name: 'refreshToken')
  token ? config.headers.Authorization = `Bearer ${token.token}` : delete config.headers.Authorization
  if (cookie && token && isTokenExpired(token?.expires)) {
    const refreshInstance = axios.create(defaultConfig)
    await refreshInstance.post( url: '/accounts/refresh-token').then(response => {
      const expires = tokenExpirationTime(response.data)
      const jwtToken = {
        token: response.data.jwtToken,
        expires
      }
      store.commit( type: 'account/SET_USER', response.data)
      store.commit( type: 'account/SET_TOKEN', jwtToken)
      config.headers['Authorization'] = `Bearer ${response.data.jwtToken}`
    }).catch(async error => {
      console.log(error.response)
      await store.commit( type: 'account/LOGOUT_USER')
      await this.$router.push({ name: 'Login' })
      await store.dispatch( type: 'notification/setSnackbar',  payload: { type: 'error', text: error.response })
    })
  }
  return config
}, onRejected: error => {
  return Promise.reject(error);
})
```

Рисунок 4.17 – Використання перехоплювача перед виконанням запиту

В даному випадку, перевіряється наявність cookie, токена та його терміну життя. Якщо є cookie і JWT токен, у якого вже минув термін життя створюється новий екземпляр axios і надсилається запит на оновлення токена. У разі неуспішного оновлення токена відбувається вихід з облікового запису поточного користувача і перенаправлення на сторінку авторизації при цьому виводиться відповідне повідомлення.

Створення сервісів для надсилання запитів

Сервіси на стороні клієнта виконують дуже важливу бізнес-логіку застосунку, оскільки вони є набором методів, які за допомогою створених екземплярів axios і використовуючи потрібний тип HTTP запиту повертають Promise з отриманими даними. (наприклад, рис. 4.18 - accountService і рис. 4.19 – foodService відповідно).

```
const baseURL = '/accounts'

const login = async (email, password) => {
  return await DefaultAPI.post( url: `${baseURL}/authenticate`, data: { email, password }).then(response => {
    DefaultAPI.interceptors.request.use( onFulfilled: config => {
      config.headers['Authorization'] = `Bearer ${response.data.jwtToken}`
      return config
    }, onRejected: error => {
      return Promise.reject(error)
    })
    return response
  }).catch(error => {
    return Promise.reject(error)
  })
}
```

Рисунок 4.18 – Реалізація методу login для авторизації користувача

```
const baseURL = '/foods'

const searchParser = async params => {
  return await FoodDatabaseAPI.get( url: '/api/food-database/v2/parser', config: {
    params,
    paramsSerializer: params => qs.stringify(params, options: { arrayFormat: 'repeat' })
  })
}
```

Рисунок 4.19 – Реалізація методу searchParser для пошуку продуктів харчування

Таким чином, можна створювати різні послуги та методи в них для роботи з потрібним екземпляром axios.

Реалізація керування станом застосунку

Для реалізації керування станом застосунку використовується бібліотека Vuex та її концепція «State Management Pattern».

Так як у проєкті використовується багато різних компонентів, що не взаємодіють між собою, доцільно розділити їх на різні частини використовуючи модулі (рис. 4.20).

```
Vue.use(Vuex)

export default new Vuex.Store( options: {
  state: {},
  mutations: {},
  actions: {},
  modules: {
    application,
    notification,
    account,
    userData,
    mealPlan,
    food,
  }
})
```

Рисунок 4.20 – Ініціалізація Vuex сховища та встановлення модулів

Кожен модуль експортується з властивостями, що застосовуються (рис. 4.21).

```
export const account = {
  namespaced: true,
  state,
  getters,
  actions,
  mutations,
};
```

Рисунок 4.21 – Експортування модуля account

У стані зберігаються дані про користувача та його JWT використовуючи концепт BehaviourSubject, який надає бібліотека rxjs, а також масив з усіма користувачами для панелі адміністратора (рис. 4.22).

```
const user = new BehaviorSubject(getUser())
const token = new BehaviorSubject(getToken())

const state = () => ({
  user,
  token,
  users: []
})
```

Рисунок 4.22 – Встановлення та зміст стану модуля account

Для отримання відформатованих даних без змін стану використовуються гетери (рис. 4.23).

```
const getters = {
  getUser: state => state.user.asObservable(),
  getUserValue: state => state.user.value,
  getUsers: state => state.users
}
```

Рисунок 4.23 – Використання гетерів для модуля account

Дії здійснюють виклик мутацій та можуть містити довільні асинхронні операції. Наприклад, такі як виклик методів сервісів для відправки HTTP запитів (рис. 4.24).

```
async login ({ commit }, { email, password }) {
  return await accountService.login(email, password).then(response => {
    const expires = tokenExpirationTime(response.data)
    const jwtToken = {
      token: response.data.jwtToken,
      expires
    }
    commit('SET_USER', response.data)
    commit('SET_TOKEN', jwtToken)
    return response
  })
},
```

Рисунок 4.24 – Обробка дії авторизації користувача

Єдиний спосіб фактично змінити стан у Vuex – здійснити мутацію. Мутації Vuex дуже схожі на події: кожна мутація має рядковий тип і обробник. Функція обробника – це місце, де виконуються фактичні модифікації стану, і вона отримає стан як перший аргумент (рис. 4.25).

```
SET_USER: (state, response) => {
  const excludedKeys = ['message', 'jwtToken']
  const user = excludedFilter(response, excludedKeys)
  localStorage.setItem('user', JSON.stringify(user))
  state.user.next(user)
},
SET_TOKEN: (state, jwtToken) => {
  localStorage.setItem('jwtToken', JSON.stringify(jwtToken))
  state.token.next(jwtToken)
},
```

Рисунок 4.25 – Мутації для зміни стану поточного користувача та JWT

Тепер досить просто можна звернутися до обробника дій прямо як до методу прямо з компонента Vue використовуючи необхідні параметри (рис. 4.26).

```
methods: {
  ...mapActions('account', ['login']),
  ...mapActions('notification', ['setAlert', 'setSnackbar']),
```

Рисунок 4.26 – Використання обробників дій у Vue-компоненті

Також, можна отримувати значення зі стану та гетерів використовуючи методи ...mapState() та ...mapGetters() відповідно.

Створення інтерфейсу користувача

Для створення інтерфейсу користувача використовуються UI компоненти, які надає бібліотека Vuetify. На рисунку 4.27 представлено налаштування параметрів, що включає вибір теми за замовчуванням та встановлення використовуваних кольорів.

```
export default new Vuetify({
  theme: {
    dark: true,
    themes: {
      dark: {
        primary: colors.green.accent3,
        accent: colors.grey.darken3,
        secondary: colors.amber.darken3,
        info: colors.teal.lighten1,
        warning: colors.amber.base,
        error: colors.deepOrange.accent4,
        success: colors.green.accent3,
        blue: '#0057b8',
        yellow: '#ffd800'
      }
    }
  }
});
```

Рисунок 4.27 – Налаштування теми та кольорів Vuetify

Після цього дана бібліотека разом з усіма іншими ініціалізуються в об'єкті Vue у файлі main.js (рис. 4.28).

```
new Vue({
  router,
  store,
  vuetify,
  render: h => h(App)
}).$mount( elementOrSelector: '#app')
```

Рисунок 4.28 – Ініціалізація застосунку Vue

На рисунках 4.29 – 4.39 продемонстровано кілька фрагментів реалізованого UI після створення всіх необхідних компонентів відповідно до функціональних та нефункціональних вимог вебзастосунку.

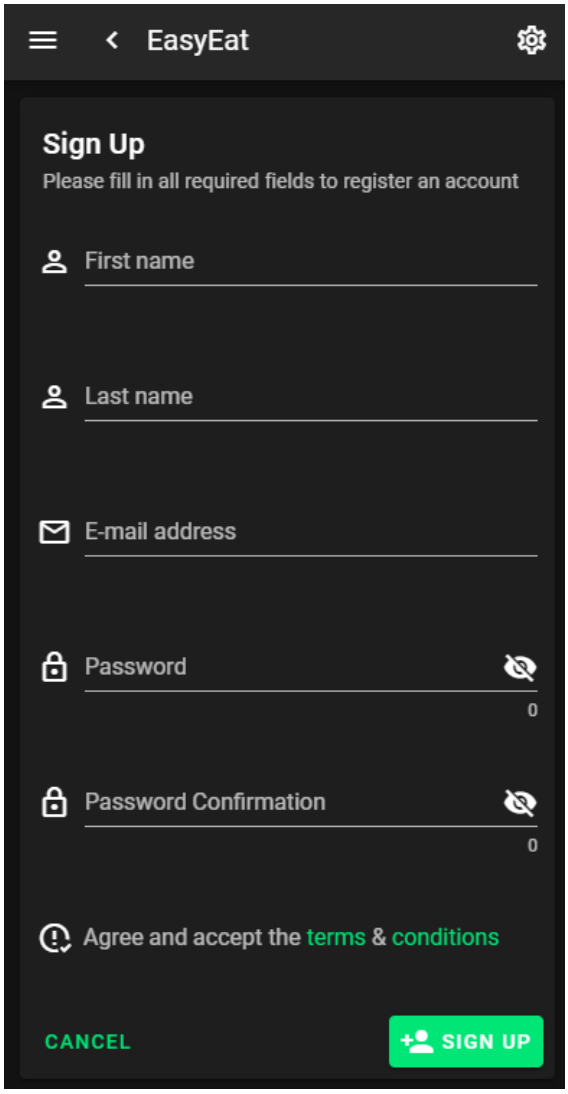

Рисунок 4.29 – Реалізований UI для реєстрації нового облікового запису

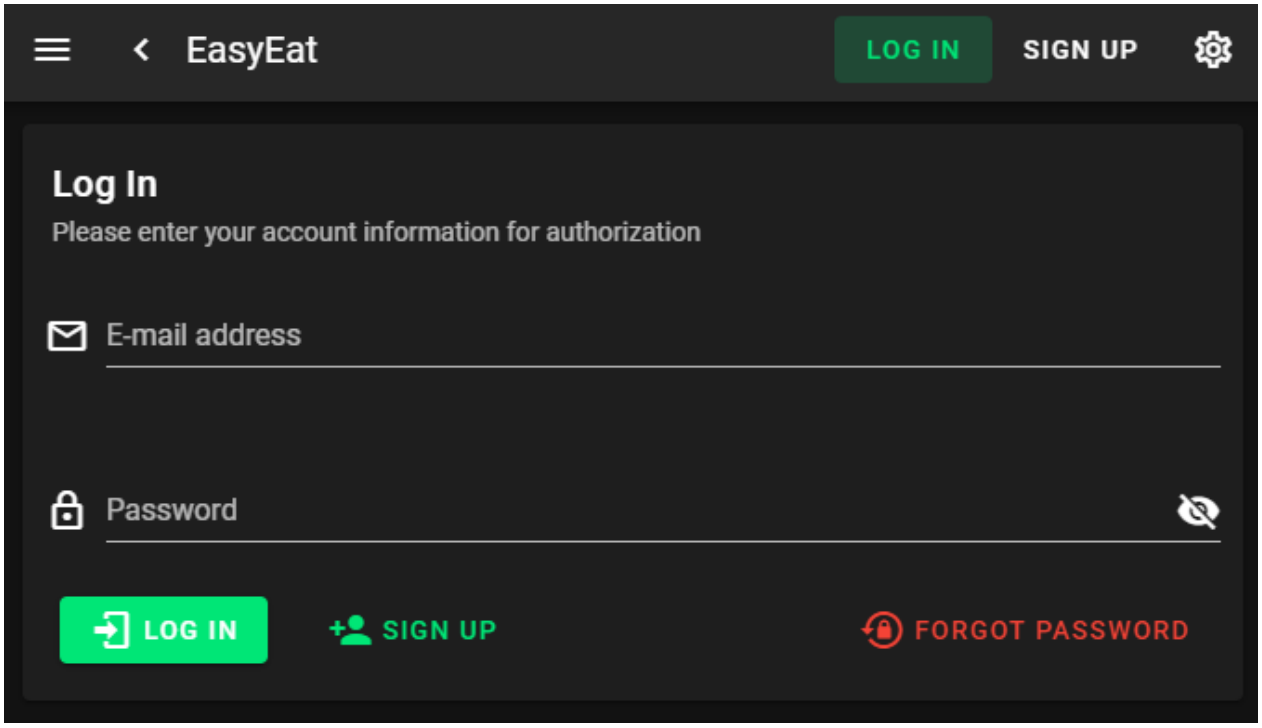

Рисунок 4.30 – Реалізований UI для авторизації користувача

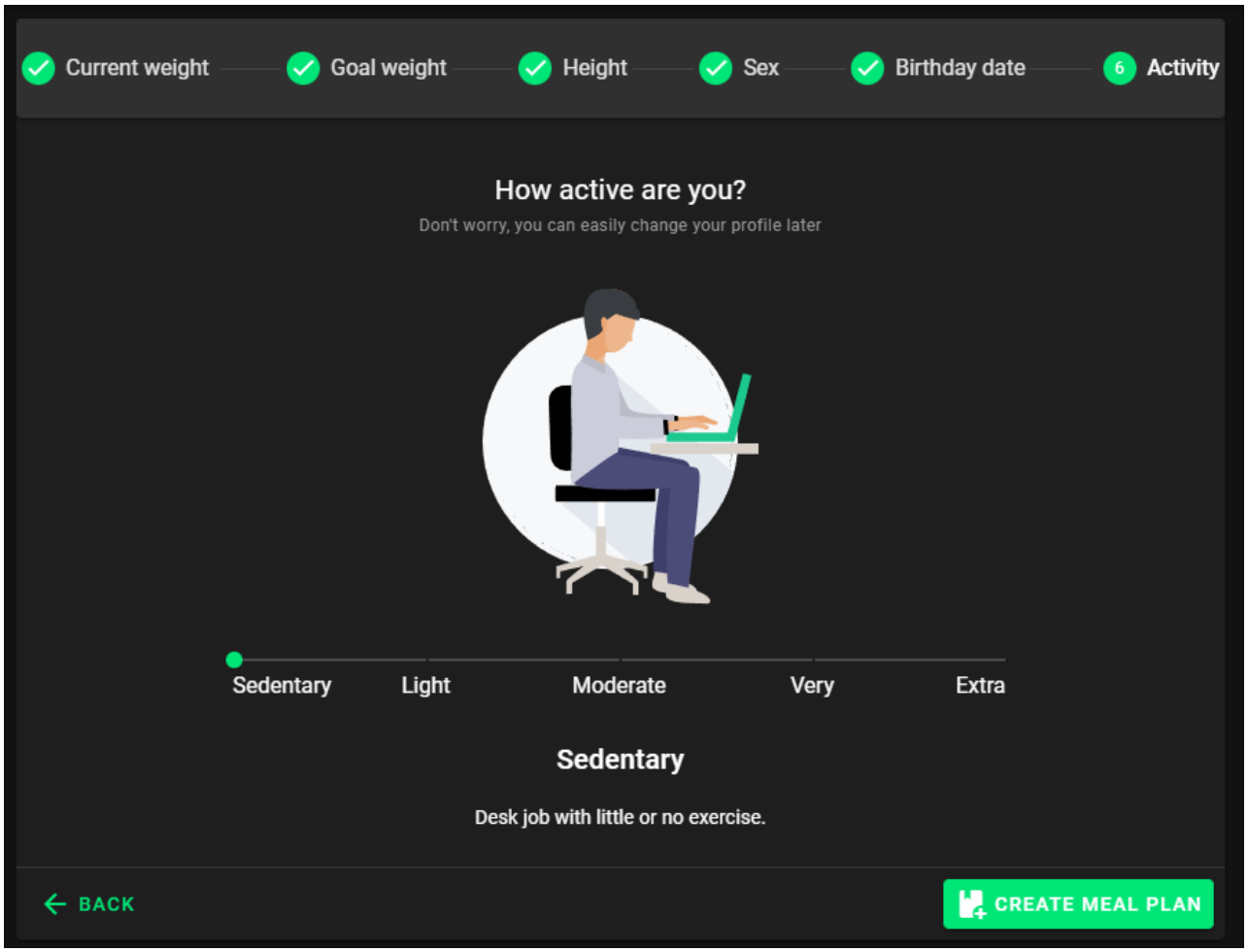

Рисунок 4.31 - Реалізований UI для проходження анкети користувача

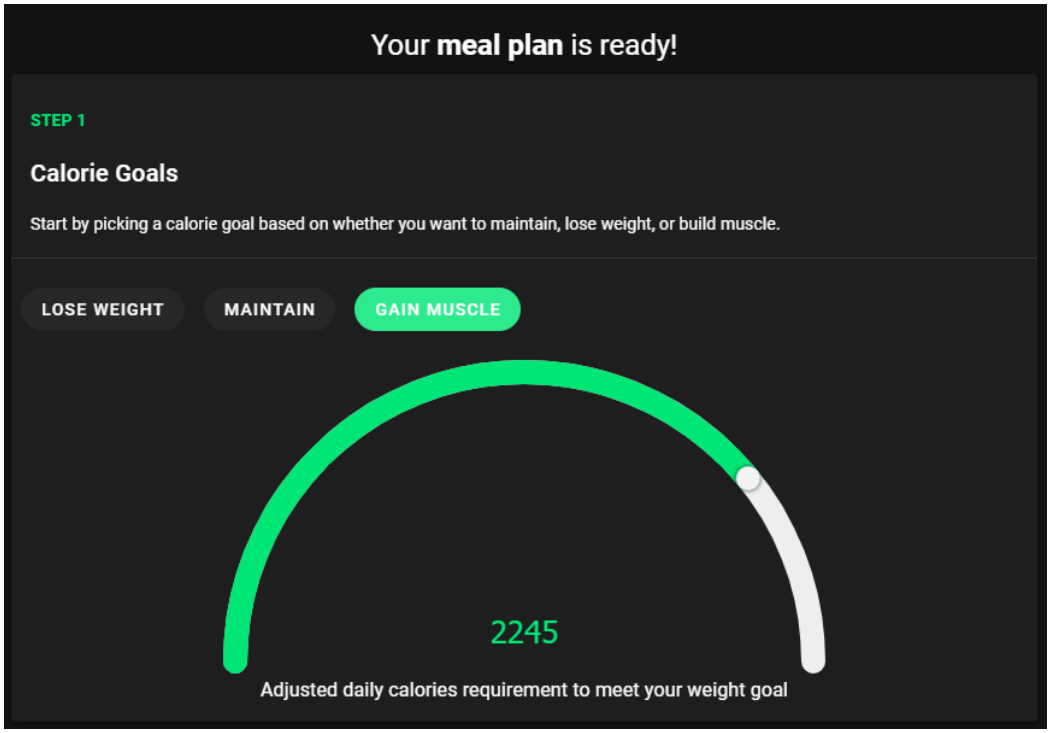

Рисунок 4.32 – Реалізований UI для визначення та встановлення плану харчування

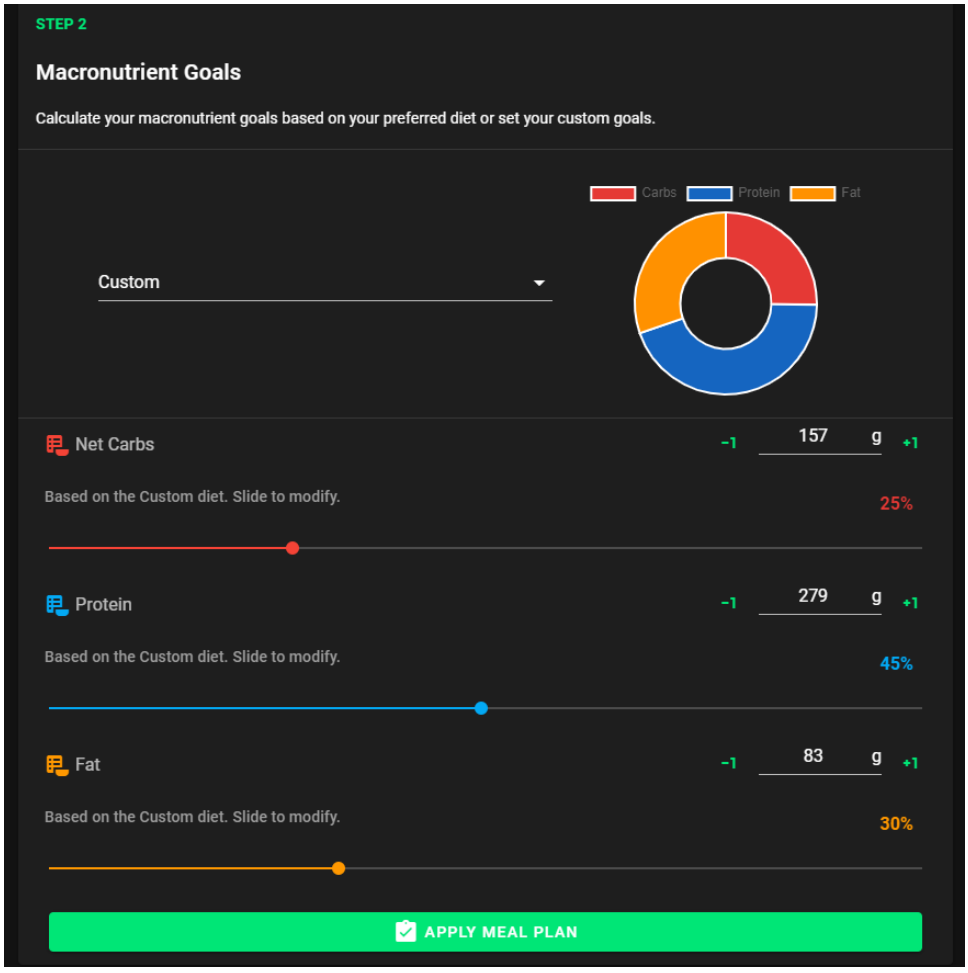

Рисунок 4.33 – Реалізований UI для визначення та встановлення плану харчування

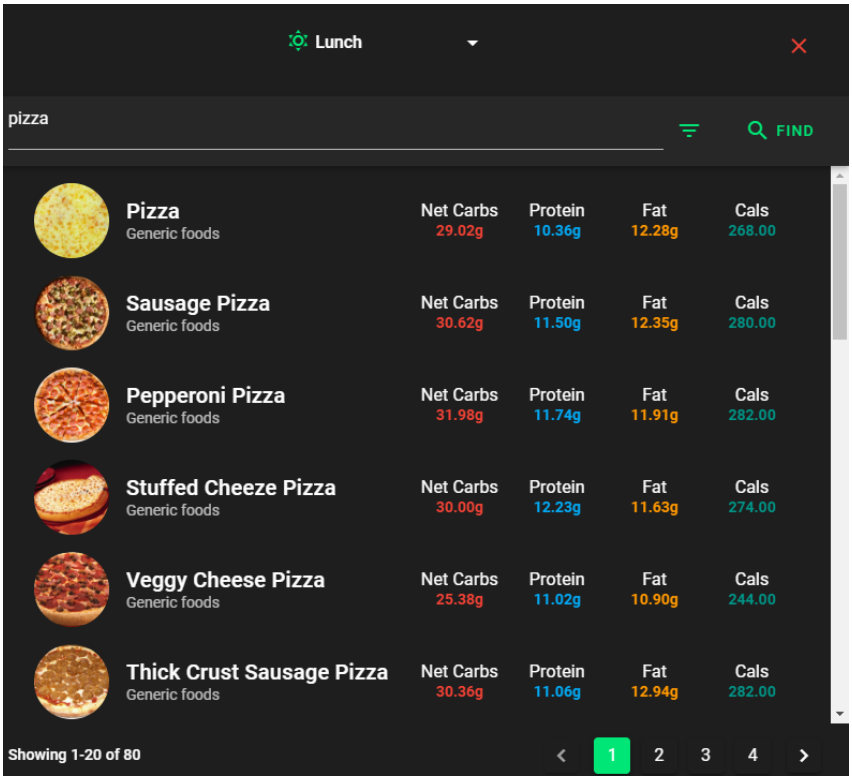

Рисунок 4.34 – Реалізований UI для пошуку продуктів харчування

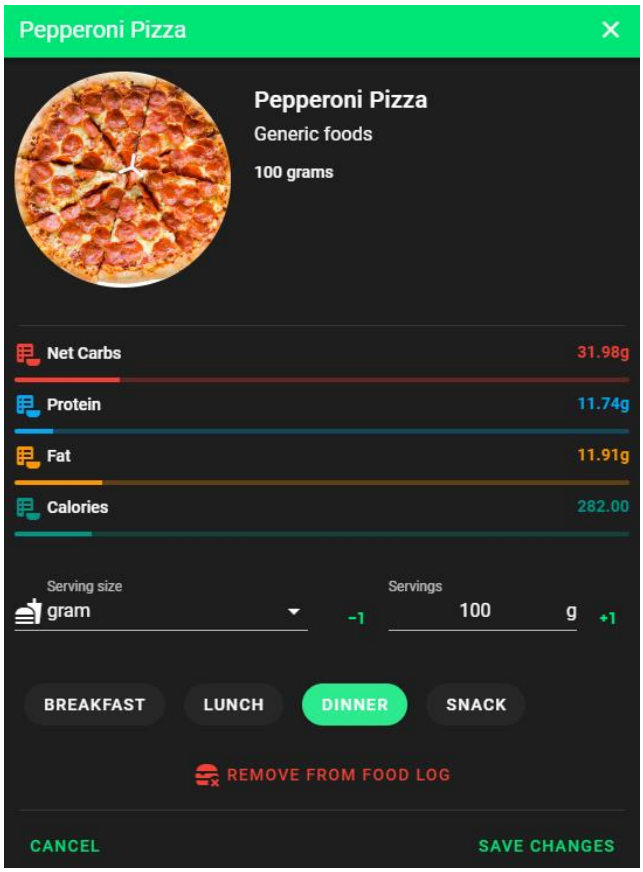

Рисунок 4.35 – Реалізований UI для редагування інформації про продукт харчування

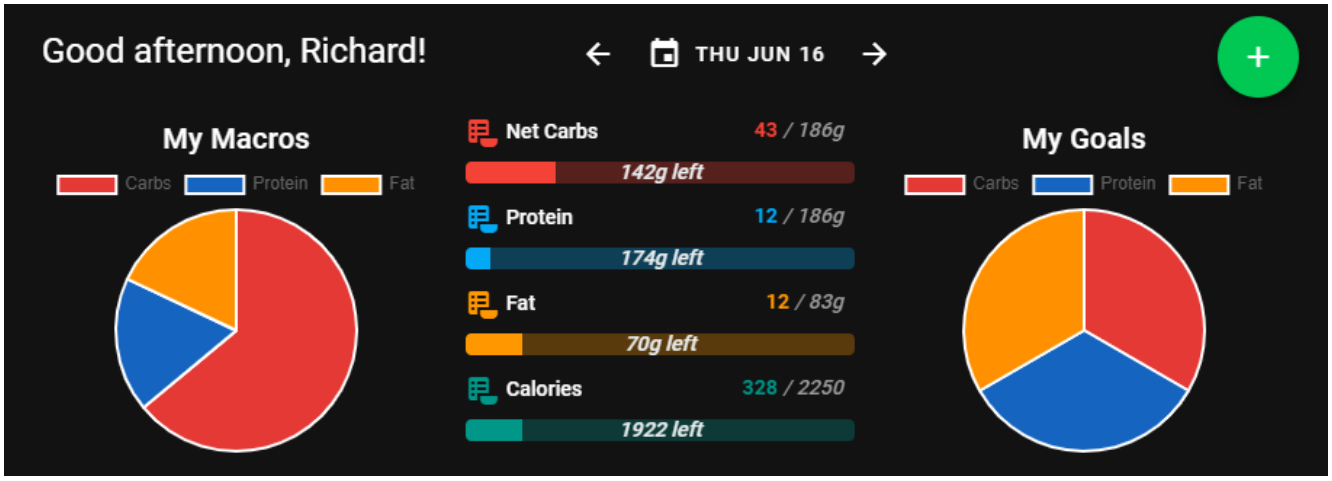

Рисунок 4.36 – Реалізований UI для діаграм, трекерів та необхідної навігації для користувача

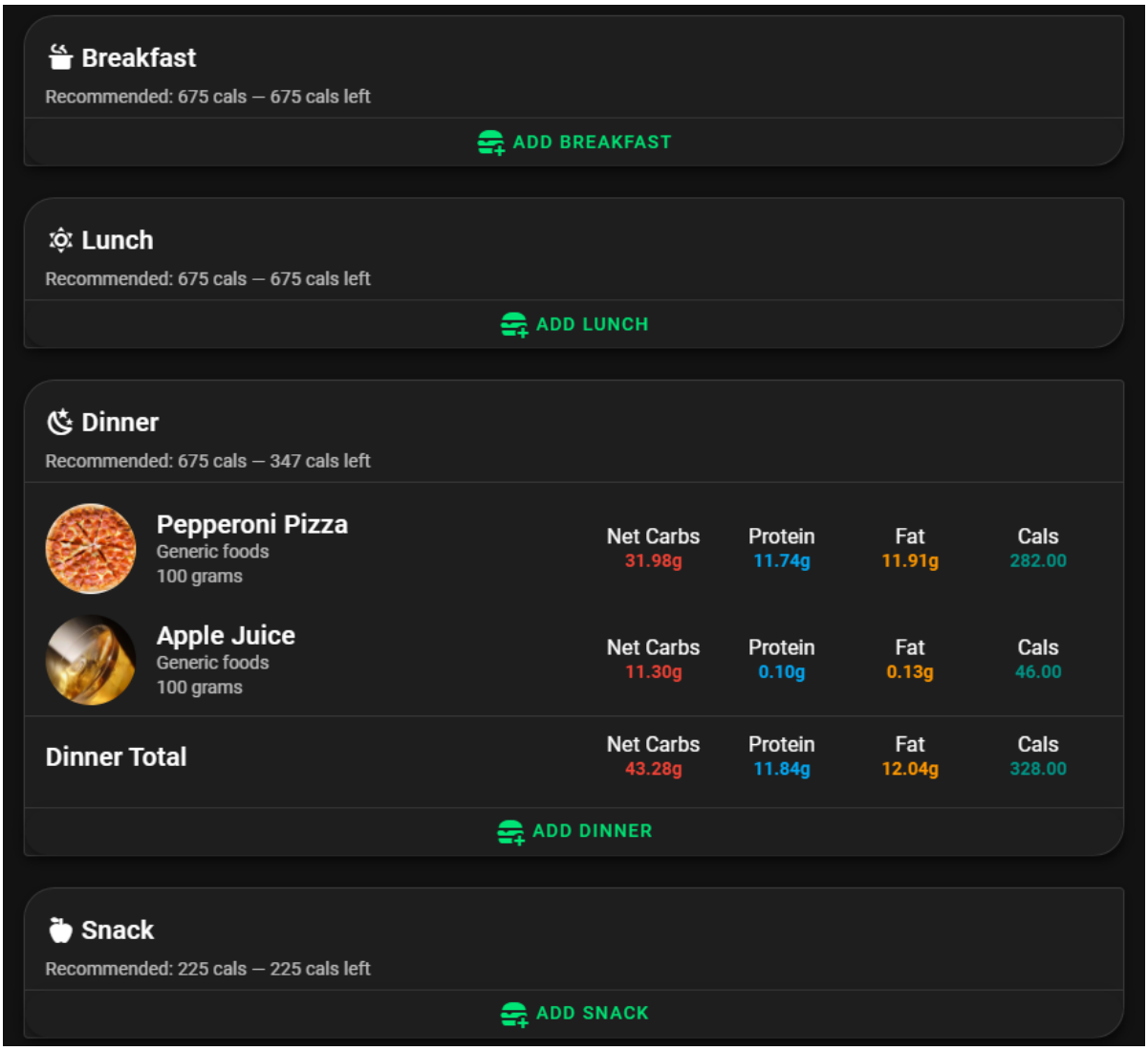

Рисунок 4.37 – Реалізований UI для виведення доданих продуктів харчування до кожного денного прийому їжі

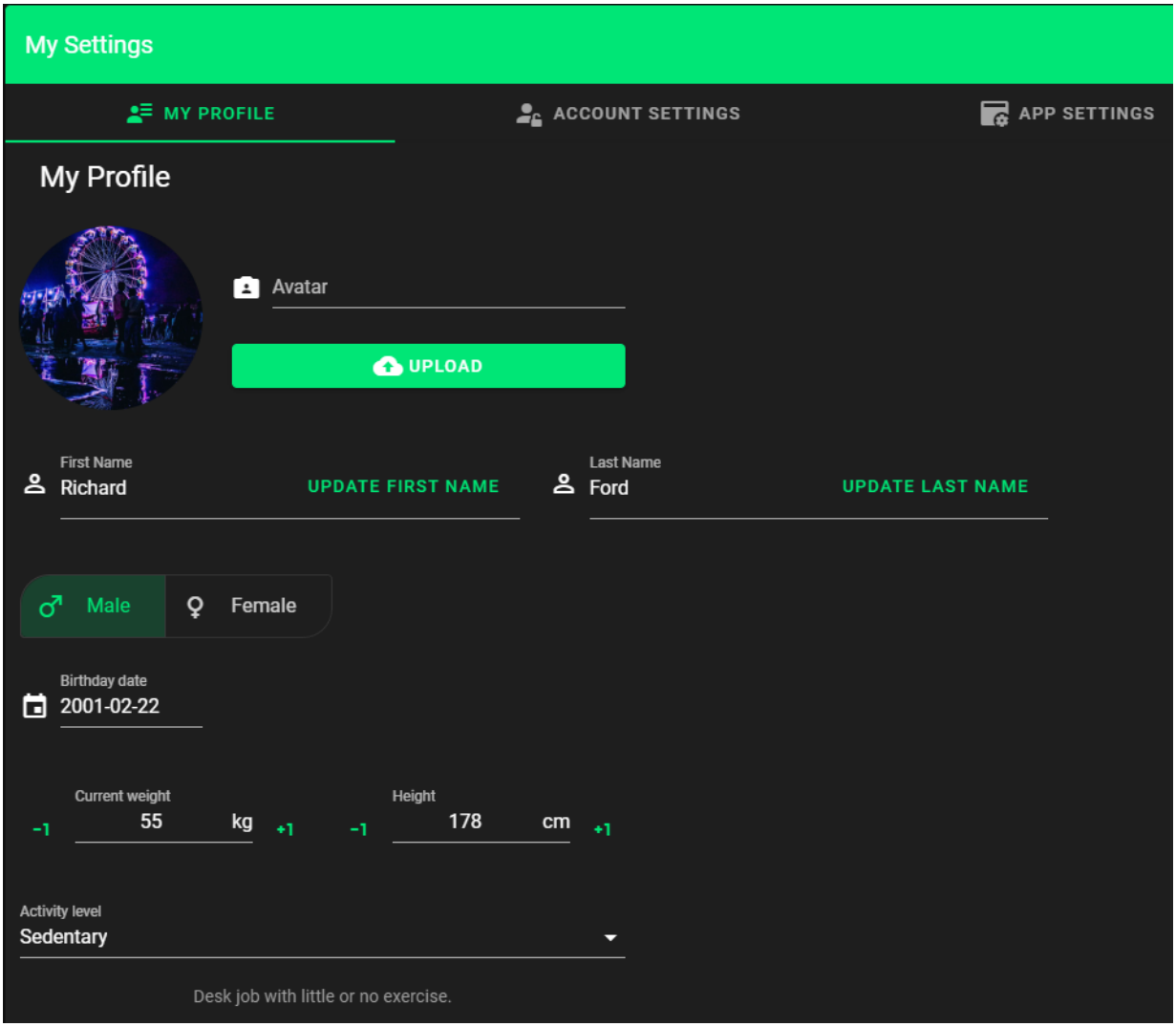

Рисунок 4.38 – Реалізований UI для виведення та редагування профілю користувача

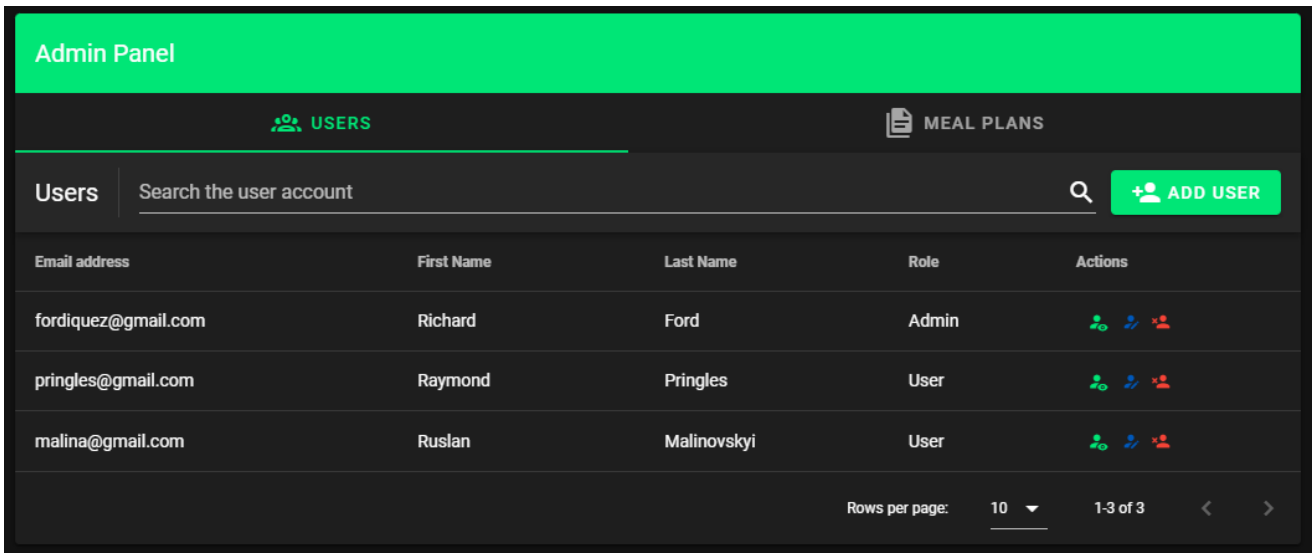

Рисунок 4.39 – Реалізований UI адмін-панелі для керування користувачами

Реалізований UI за допомогою використання компонентів UI бібліотеки Vuetify є досить простим і водночас сучасним на вигляд. Важливо відзначити, що він також має адаптивність до екранів з різною роздільною здатністю, що повністю забезпечує весь наявний функціонал, як для користувача, так і для адміністратора вебзастосунку.

Висновки до розділу 4

У четвертому розділі була виконана програмна реалізація вебзастосунку, яка включає серверну і клієнтську частину, а також базу даних.

У ході реалізації серверної частини за допомогою фреймворку Express.js та за допомогою платформи Node.js було створено middleware-функції, налаштовано підключення до бази даних, створено всі необхідні моделі, маршрути, контролери та сервіси.

На клієнтській частині було проведено низку таких дій:

– налаштування маршрутизації для реалізації SPA та створення глобального навігаційного guard для захисту маршрутів;

– створення двох axios екземплярів та request-перехоплювача для роботи з HTTP запитами;

– реалізація керування станом вебзастосунку за допомогою бібліотеки Vuex та її концепції «State Management Pattern»;

– створено та візуалізовано інтерфейс користувача за допомогою Vue-компонентів та UI-компонентів, які надає бібліотека Vuetify.

Як підсумковий висновок можна констатувати, що в ході програмної реалізації вебзастосунку всі необхідні функціональні та нефункціональні вимоги були виконані.

ВИСНОВКИ

Для виконання кваліфікаційної роботи бакалавра була поставлена мета популяризація контролю та підтримки щоденного балансу вжитих калорій та макронутрієнтів за рахунок розробленого вебзастосунку. Для реалізації якого було виконано низку таких завдань:

- проведено аналіз предметної області та складено специфікацію вимог;
- обрано технології та програмні рішення;
- змодельовано та спроєктувано вебзастосунок;
- реалізовано програмну частину вебзастосунку.

Реалізований вебзастосунок враховує всі недоліки конкурентних аналогів та використовує описаний метод та етапи для формування цілей щодо вибору плану харчування. Важливою особливістю є те, що цей вебзастосунок можна назвати досить гнучким як для кінцевого використання, так і для розробника. Це легко пояснити двома наступними факторами.

По-перше, UI є адаптивним до пристроїв з різною роздільною здатністю екрану. Крім цього, функціонал вебзастосунку забезпечує створення кастомних продуктів харчування, а адмін-панель у свою чергу оснащена можливістю додавати плани харчування з певними пропорційними співвідношеннями макронутрієнтів.

По-друге, при програмної реалізації вебзастосунку використовувалася концепція «Seperation of Concerns» та створення Vue-компонентів з використанням концепції «State Management Pattern», що робить код чистішим для розуміння і дозволяє легко в ньому розібратися. Також це розділяє вебзастосунок на різні зони відповідальності, що дозволяє зручно модифікувати реалізований функціонал або впроваджувати нові ідеї для того, щоб йти за крок із сучасними трендами веброзробки.

Результатом завдання кваліфікаційної роботи є функціонуючий вебзастосунку обліку та контролю вжитих калорій на основі обраного плану харчування, тому всі цілі кваліфікаційної роботи бакалавра вважаються виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What Is a Web Application? How It Works, Benefits and Examples. https://www.indeed.com/career-advice/career-development/what-is-web-application. (дата звернення: 24.01.2022);

2. Weight loss during the intensive intervention phase of the weight-loss maintenance trial URL: https://pubmed.ncbi.nlm.nih.gov/18617080/. (дата звернення: 24.01.2022);

3. Guide to AMDR: Macronutrient Ranges and Recommendations. https://www.weightwatchers.com/us/blog/food/acceptable-macronutrient-distribution-range. (дата звернення: 24.01.2022);

4. Selhub, E. Nutritional psychiatry: Your brain on food. URL: https://www.health.harvard.edu/blog/nutritional-psychiatry-your-brain-on-food-201511168626. (дата звернення: 25.01.2022);

5. Satrazemis, E. Macro & Micronutrients. URL: https://www.trifectanutrition.com/blog/macro-micro-nutrients. (дата звернення: 25.01.2022);

6. Diet and food supplements. URL: https://www.macmillan.org.uk/cancer-information-and-support/treatment/coping-with-treatment/complementary-therapies/diet-and-food-supplements. (дата звернення: 25.01.2022);

7. Nutrition basics. URL: https://mynutrition.wsu.edu/nutrition-basics. (дата звернення: 25.01.2022);

8. Food, Nutrition, Health and Fitness. URL: https://ncert.nic.in/textbook/pdf/kehe103.pdf. (дата звернення: 26.01.2022);

9. Trabulsi, J., Schoeller, D. 2001. Evaluation of dietary assessment instruments against double labeled water, a biomarker of habitual intake. American Journal of Physiology: Endocrinology and Metabolism, T. 281. C. 891 – 899. URL: https://journals.physiology.org/doi/full/10.1152/ajpendo.2001.281.5.E891 (дата звернення: 26.01.2022);

10. Patel, M.L., Hopkins, C.M., Brooks, T.L., Bennett, G. G. 2019. Comparing self-monitoring strategies for weight loss in a smartphone app: randomized controlled trial. Journal of Medical Internet Research, URL: https://mhealth.jmir.org/2019/2/e12209/ (дата звернення: 26.01.2022);

11. Calories: Requirements, health needs, and function. URL: https://www.medicalnewstoday.com/articles/263028. (дата звернення: 26.01.2022);

12. How to Calculate the Energy Value of Food. URL: https://huel.com/pages/how-to-calculate-the-energy-value-of-food. (дата звернення: 26.01.2022);

13. J. Arthur Harris and Francis G. Benedict. 1918. A Biometric Study of Human Basal Metabolism. с. 370-373. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1091498/. (дата звернення: 26.01.2022);

14. M. D. Mifflin, S. T. St Jeor, L. A. Hill, B. J. Scott, S. A. Daugherty, Y. O. Koh. 1990. A new predictive equation for resting energy expenditure in healthy individuals. С. 241 – 247. URL: https://academic.oup.com/ajcn/article-abstract/51/2/241/4695104 (дата звернення: 26.01.2022);

15. TDEE Calculator – Total Daily Energy Expenditure. URL: https://www.omnicalculator.com/health/tdee. (дата звернення: 26.01.2022);

16. Losing Weight | Healthy Weight, Nutrition, and Physical Activity. URL: https://www.cdc.gov/healthyweight/losing_weight/index.html. (дата звернення: 26.01.2022);

17. AMDR: Macronutrient Ranges & Recommendations | WW USA. URL: https://www.weightwatchers.com/us/blog/food/acceptable-macronutrient-distribution-range. (дата звернення: 26.01.2022);

18. Застосунок «MyFitnessPal». URL: https://www.myfitnesspal.com/ru. (дата звернення: 27.01.2022);

19. Застосунок «Nutritionix». URL: https://www.nutritionix.com/. (дата звернення: 28.01.2022);

20. Застосунок «Cronometer». URL: https://cronometer.com/. (дата звернення: 29.01.2022);

21. Web Application Architecture in 2022: Moving in the Right Direction. URL: https://mobidev.biz/blog/web-application-architecture-types. (дата звернення: 31.01.2022);

22. What is a Web Framework?. URL: https://www.goodfirms.co/glossary/web-framework/. (дата звернення: 31.01.2022);

23. What is a Framework? [Definition] Types of Frameworks. URL: https://hackr.io/blog/what-is-frameworks. (дата звернення: 31.01.2022);

24. What are web frameworks and why you need them? URL: https://www.intelegain.com/what-are-web-frameworks-and-why-you-need-them/. (дата звернення: 31.01.2022);

25. Detailed Analysis of the Top Modern Database Solutions. URL: https://yalantis.com/blog/how-to-choose-a-database/. (дата звернення: 02.02.2022);

26. What is Unified Modeling Language (UML)? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. (дата звернення: 07.02.2022);

27. ER Diagram (ERD) – Definition & Overview | Lucidchart. URL: https://www.lucidchart.com/pages/er-diagrams. (дата звернення: 10.02.2022);

28. Separation of Concerns. URL: https://medium.com/machine-words/separation-of-concerns-1d735b703a60. (дата звернення: 14.02.2022);

29. Lifecycle Hooks | Vue.js. URL: https://vuejs.org/guide/essentials/lifecycle.html. (дата звернення: 15.02.2022);

30. What is Vuex? | Vuex. URL: https://vuex.vuejs.org/. (дата звернення: 15.02.2022).