НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.4, 004.75

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем» зі спеціальності 121 "Інженерія програмного забезпечення" на тему: «Система розпізнавання та оцінки калорійності їжі на базі штучного інтелекту»

Виконала:
студентка ІІ курсу, групи ІМ-32мп
Ганна БОНДАРЕНКО _____

Керівник:
доц. каф. ОТ, к.т.н., доц.,
Артем ВОЛОКИТА _____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н., професор,
Валерій ЖАБІН _____

Рецензент:
_____
_____ _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет/ННІ *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *121 «Інженерія програмного забезпечення»*
(код і назва)

Освітньо-професійна програма *Інженерія програмного забезпечення комп'ютерних систем*
(код і назва)

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«___»_____2024 р.

ЗАВДАННЯ

на магістерьку дисертацію студентці

Ганні Бондаренко
(прізвище, ім'я, по батькові)

1. Тема дисертації Система розпізнавання та оцінки калорійності їжі на базі штучного інтелекту

науковий керівник дисертації Волокита Артем Миколайович, доцент кафедри ОТ, кандидат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 08 » листопада 2024 р. № 5016-с

2. Строк подання студентом дисертації 29.11.2024

3. Об'єкт дослідження процес розпізнавання та оцінки калорійності їжі на базі штучного інтелекту

4. Предмет дослідження підхід та способи розробки користувацької системи для визначення продуктів та оцінки калорійності їжі

5. Перелік завдань, які потрібно розробити дослідити предметну область та проаналізувати існуючі рішення, на основі дослідження сформувати вимоги до системи розпізнавання їжі та визначення калорійності, обрати технології для

розробки, розробити архітектуру системи, розробити інтерфейс користувача, розробити стартап-проєкт

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Жабін В. І., проф. каф. ОТ, д.т.н., професор		
Розділ 1	Волокита А. М., доц. каф. ОТ, к.т.н., доц.		
Розділ 2	Волокита А. М., доц. каф. ОТ, к.т.н., доц.		
Розділ 3	Волокита А. М., доц. каф. ОТ, к.т.н., доц.		
Розділ 4	Волокита А. М., доц. каф. ОТ, к.т.н., доц.		

7. Дата видачі завдання: 02.09.2024

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	02.09.2024	
2	Огляд літератури	02.09.2024 – 15.09.2024	
3	Аналіз існуючих рішень	16.09.2024 – 30.09.2024	
4	Розробка підходів для реалізації системи розпізнавання та оцінки калорійності їжі	01.10.2024 – 27.10.2024	
5	Проєктування системи	01.10.2024 – 27.10.2024	
6	Програмна реалізація, тестування та виправлення помилок	01.10.2024 – 29.11.2024	
7	Розробка стартап-проєкту	10.11.2024 – 29.11.2024	
8	Оформлення магістерької дисертації	28.10.2024 – 29.11.2024	
9	Захист	19.12.2024	

Студентка Ганна БОНДАРЕНКО _____
(підпис)

Науковий керівник дисертації Артем ВОЛОКИТА _____
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: *Система розпізнавання та оцінки калорійності їжі на базі штучного інтелекту*

студенткою: Ганною Бондаренко

Магістерська дисертація складається із вступу та чотирьох розділів. Загальний обсяг роботи: _ аркуші основного тексту, _ ілюстрацій, _ таблиці. При підготовці використовувалась література із _ різних джерел.

Актуальність теми: У сучасному світі правильне харчування стає дедалі важливішим аспектом здорового способу життя. Автоматизація процесів відслідковування харчування, зокрема розпізнавання їжі, визначення калорійності та управління продуктами, дозволяє підвищити ефективність контролю за раціоном і зменшити навантаження на користувача. Існуючі рішення не враховують певні потреби, такі як інтеграція з вітчизняними (в даному випадку українськими) магазинами, ведення обліку запасів продуктів та пропонування рецептів на основі доступних інгредієнтів, що створює значну потребу у створенні більш адаптованих і комплексних систем.

Мета роботи: Розробка системи для автоматичного розпізнавання їжі, оцінки калорійності, управління запасами продуктів і рекомендацій рецептів, що дозволило б покращити харчові звички користувачів.

Об’єкт дослідження: Процеси автоматизованого розпізнавання їжі, визначення калорійності, управління продуктами та пропозиції рецептів.

Результати дослідження: Розроблено систему, яка забезпечує розпізнавання їжі, визначення калорійності, облік продуктів, рекомендацію рецептів та можливість інтеграції з онлайн-магазинами.

Предмет дослідження: Методи, моделі та алгоритми, що використовуються для автоматизації процесів розпізнавання їжі, оцінки калорійності.

Методи дослідження: Аналіз літератури та існуючих рішень, практична реалізація системи та її тестування.

Практичне значення отриманих результатів: Розроблена система може використовуватись як інструмент для автоматизації харчових звичок користувачів. Вона надає зручний спосіб управління продуктами, спрощує планування раціону та сприяє ефективному використанню харчових ресурсів.

Основні задачі дослідження випливають з його мети і полягають у наступному:

1. Вивчення існуючих систем автоматизованого розпізнавання їжі.
2. Аналіз технологій, які використовуються для розпізнавання їжі та оцінки калорійності.
3. Розробка системи: моделі бази даних для зберігання інформації про продукти, рецепти та харчові звички користувача, користувацького інтерфейсу, адміністративної панелі.
4. Реалізація алгоритмів розпізнавання їжі на основі комп’ютерного зору.
5. Створення можливості інтеграції з інтернет-магазинами для автоматичного формування кошику покупок.
6. Тестування системи на реальних даних для перевірки її ефективності.

Ключові слова: ВЕБ-ЗАСТОСУНОК, РОЗПІЗНАВАННЯ ЇЖІ, ВИЗНАЧЕННЯ КАЛОРІЙНОСТІ, УПРАВЛІННЯ ПРОДУКТАМИ, ПІДБІР РЕЦЕПТІВ, ШТУЧНИЙ ІНТЕЛЕКТ.

ABSTRACT

for a master's thesis

implemented on topic: *AI-based system for food recognition and calorie content estimation*

by student: Hanna Bondarenko

The master's thesis consists of an introduction and four chapters. Total volume of work: _ pages of the main text, _ illustrations, _ tables. For the preparation was used literature from _ different sources.

Relevance of the topic: In the modern world, proper nutrition is becoming an increasingly important aspect of a healthy lifestyle. Automation of nutrition tracking processes, including food recognition, calorie determination, and product management, can increase the efficiency of dietary control and reduce the user's workload. Existing solutions do not take into account certain needs, such as integration with local (in this case, Ukrainian) stores, keeping track of food inventory, and offering recipes based on available ingredients, which creates a significant need for more adaptable and comprehensive systems.

Objective of the research: To develop a system for automatic food recognition, calorie estimation, product inventory management, and recipe recommendation that would improve users' eating habits.

Object of research: The processes of automated food recognition, calorie estimation, product management and recipe suggestion.

Research results: A system has been developed that provides food recognition, calorie determination, product management, recipe recommendation, and the ability to integrate with online stores.

Subject of research: Methods, models and algorithms used to automate the processes of food recognition and calorie estimation.

Research methods: Analysis of literature and existing solutions, practical implementation of the system and its testing.

Practical significance of the results: The developed system can be used as a tool for automating users' eating habits. It provides a convenient way to manage products, simplifies diet planning and promotes the efficient use of food resources.

The main objectives of the study are based on its purpose and are as follows:

1. Study of existing automated food recognition systems.
2. Analysis of technologies used for food recognition and calorie estimation.
3. System development: database models for storing information about products, recipes and user's eating habits, user interface, administrative panel.
4. Implementation of food recognition algorithms based on computer vision.
5. Creating the ability to integrate with online stores to automatically create a shopping cart.
6. Testing the system on real data to verify its effectiveness.

Keywords: WEB APPLICATION, FOOD RECOGNITION, CALORIE CONTENT ESTIMATION, GROCERY MANAGEMENT, RECIPE SELECTION, ARTIFICIAL INTELLIGENCE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API	–	(Application Programming Interface) Набір функцій для взаємодії програмного забезпечення
CNN	–	(Convolution Neural Networks) Згорткові нейронні мережі
DB/БД	–	(Database) База даних
HTTP	–	(Hypertext Transfer Protocol) Протокол передачі даних
JSON	–	(JavaScript Object Notation) Формат обміну даними характерний для вебу
JWT	–	(JSON Web Token) Стандарт токена доступу на основі JSON
NPM	–	(Node Package Manager) Менеджер пакетів для лJavaScript
ORM	–	(Object-Relational Mapping) Об'єктно-реляційне відображення
PIP	–	(Pip Installs Packages) Система керування пакунками Python
REST	–	(Representational State Transfer) Підхід до архітектури мережевих протоколів
SQL	–	(Structured Query Language) Мова запитів до більшості баз даних
URI	–	(Universal Resource Identifier) Уніфікований ідентифікатор ресурсів
URL	–	(Uniform Resource Locator) Веб-адреса

ВСТУП

Зі зростанням обізнаності людей про важливість збалансованого харчування - підвищується попит на системи, які здатні автоматично розпізнавати харчові продукти. Такі технології не лише визначають вид продукту, але й оцінюють його поживну цінність, що є надзвичайно корисним для оптимізації раціону харчування. Вони можуть допомогти людям із специфічними дієтичними потребами, а також сприяти профілактиці захворювань пов'язаних із неправильним харчуванням.

Проєкт створений із використанням технології доповненої реальності, що значно спрощує процес збору інформації про поживність продуктів. Користувачеві більше не потрібно вручну вносити дані про калорійність та склад продуктів які він споживає. Натомість технологія забезпечує автоматичне розпізнавання за допомогою 3D-зображень, що дозволяє ідентифікувати продукт і зберегти інформацію про його калорії в базі даних. Це робить вибір здорового раціону швидким та простим, адже всі необхідні дані стають доступними автоматично. Основним завданням системи є точне визначення калорійності кожного продукту.

Як основний алгоритм - у моделі використовувалися Yolov3 та Yolov5. Алгоритми аналізують зображення, створюють граничні рамки для кожного об'єкта на зображенні та виключають зайві елементи. Для аналізу застосовувався попередньо підготовлений набір даних, на основі якого модель оцінювала зображення та виводила розрахункові дані про калорії. Завдяки цьому користувачі можуть використовувати систему у своєму повсякденному житті для моніторингу харчування.

Проєкт має як переваги, так і певні недоліки. Серед труднощів із якими зіткнулася модель, можна виділити залежність від якості зображень. Аналіз низькоякісних фотографій потребував декількох спроб для досягнення точного результату. Ще одним викликом стала робота з великими наборами даних, що вимагало часу на попередню обробку, хоча після декількох ітерацій модель

змогла адаптуватися до таких умов. Однак, основною перевагою є швидка обробка інформації: система здатна надати точні дані про калорійність на основі лише одного зображення за мінімальний час.

Ця система розроблена для людей, які стикаються з проблемами надмірної ваги, а також для тих, хто має недостатній рівень знань для організації здорового харчування.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Загальні відомості про оцінку калорійності їжі

Калорії є невід'ємною частиною нашого життя, адже вони визначають скільки енергії ми отримуємо з їжі та напоїв для підтримки нормальної життєдіяльності. Вони є мірою енергії, яка вивільняється під час перетравлення їжі, і забезпечують основні функції організму, такі як дихання, кровообіг та обмін речовин. Калорії зустрічаються всюди – на етикетках продуктів, у меню ресторанів, у додатках для відстеження харчування та в наукових статтях. Їхня важливість полягає не лише в розумінні енергетичної цінності їжі, але й у їхньому впливі на формування раціону, контроль ваги та загальне здоров'я людини.

Калорії надходять із трьох основних джерел: вуглеводів, жирів та білків. Кожен із цих макронутрієнтів виконує свою окрему функцію. Вуглеводи є основним джерелом енергії, особливо для мозку, нервової системи та кров'яних клітин. Жири – забезпечують тривале енергопостачання, допомагають у засвоєнні вітамінів і відіграють ключову роль у терморегуляції та захисті органів. Білки, в свою чергу, використовуються для відновлення тканин, синтезу ферментів і гормонів. Хоч білки і можуть слугувати джерелом енергії, організм намагається зберегти їх для будівельних та регуляторних функцій.

Калорійність їжі вимірюється в кілокалоріях (ккал), де одна ккал дорівнює енергії необхідній для підвищення температури 1 літра води на 1 градус Цельсія. У побутовому спілкуванні ми часто скорочуємо цей термін і говоримо просто "калорії". Наприклад, середній банан містить 105 ккал, що простіше сприймається, ніж 105 000 калорій. Для кожного макронутрієнта калорійність відрізняється: вуглеводи та білки забезпечують по 4 ккал на грам, а жири – 9 ккал на грам, що робить їх найенергетичнішим компонентом їжі [1].

Кількість калорій, необхідних для кожної людини, залежить від багатьох чинників: віку, статі, фізичної активності, маси тіла та загального стану здоров’я. Наприклад, середній дорослий чоловік потребує від 2000 до 3000 ккал на день, а жінка – від 1600 до 2400 ккал. Ці цифри можуть змінюватися залежно від рівня фізичної активності. Базовий рівень метаболізму, який відповідає за основні функції організму, становить близько 60-75% добової енергетичної витрати. Решта енергії використовується на фізичну активність та інші процеси.

Контроль ваги тісно пов’язаний із балансом калорій. Якщо людина споживає більше калорій, ніж витрачає, це призводить до накопичення жиру й збільшення ваги. У разі дефіциту калорій організм починає використовувати жирові запаси як джерело енергії, що сприяє схудненню. Важливо, щоб дефіцит калорій був помірним, адже надмірне його скорочення може уповільнити обмін речовин і викликати небажані наслідки для здоров’я.

Якість калорій також відіграє ключову роль. Калорійно щільна їжа, наприклад фастфуд або солодощі, містить багато калорій, але мало поживних речовин. Натомість продукти з високою поживною цінністю, такі як фрукти, овочі, горіхи, риба, забезпечують організм необхідними вітамінами, мінералами та іншими корисними речовинами. Раціон має базуватися на таких поживно щільних продуктах, які сприяють здоров’ю та підтримують оптимальний рівень енергії.

Таким чином, розуміння калорійності їжі та її впливу на організм дозволяє не лише ефективно контролювати вагу, але й підтримувати загальне здоров’я. Формування збалансованого раціону з урахуванням енергетичних потреб організму є важливим кроком до здорового способу життя.

1.2 Огляд існуючих застосунків

Сучасні технології штучного інтелекту та комп’ютерного зору активно розвиваються, і одним із ключових напрямів їх застосування є автоматичне розпізнавання їжі та оцінка її калорійності. Сьогодні вже існують численні

мобільні додатки, що працюють у цій сфері, зокрема Calorie Mama, Snap Calorie, Centa, CaloPal і MacrosAI. Кожен із цих додатків має свої унікальні риси, переваги та виклики, які важливо враховувати під час аналізу їхнього функціоналу та перспектив розвитку.

1.2.1 Calorie Mama

Одним із найпопулярніших інструментів у цій галузі є Calorie Mama, який працює на основі технологій глибокого навчання [2]. Суть роботи цього застосунку полягає в автоматичному розпізнаванні їжі за допомогою зображень. Користувачеві достатньо зробити фото страви, після чого система аналізує її, класифікує та оцінює калорійність і поживний склад (Рис.1.1.). Застосунок підтримує розпізнавання широкого спектра продуктів завдяки великій базі даних, що охоплює кухні різних країн світу.

Особливістю Calorie Mama - є його адаптивний алгоритм, який навчається на основі нових даних, що вводять користувачі. Проте його робота значною мірою залежить від інтернет-з’єднання, так як обробка зображень здійснюється на сервері. Це створює певні обмеження для використання в умовах низької доступності мережі. Крім того застосунок іноді демонструє труднощі у розпізнаванні багатокомпонентних страв, таких як салати або складні гарніри. Удосконаленням може стати локальне кешування бази даних та вдосконалення алгоритмів для більш точного аналізу змішаних продуктів.

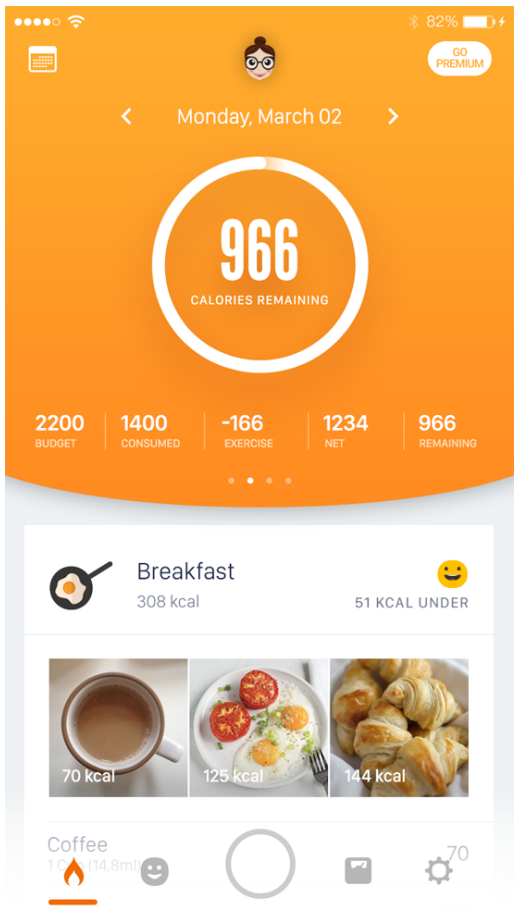


Рис.1.1. Інтерфейс застосунку Calorie Mama

1.2.2 Snap Calorie

Snap Calorie пропонує ще більш специфічний підхід, орієнтований на досягнення максимальної точності [3]. Він використовує сучасні нейронні мережі для аналізу текстури, кольору та інших візуальних характеристик їжі. І завдяки цьому система забезпечує високий рівень точності навіть для складних страв. SnapCalorie працює швидко: після завантаження зображення користувач отримує результати майже миттєво (Рис.1.2.).

Цікавим аспектом є інтеграція Snap Calorie із платформами для фітнесу. Це дозволяє користувачам не лише стежити за своїм харчуванням, але й зв’язувати його з рівнем фізичної активності. Система використовує дані датчиків глибини на смартфонах для оцінки розміру порції, і це є унікальною технологічною особливістю цього застосунку. Однак застосунок поки що не підтримує

можливість ручного коригування результатів, що може стати бар'єром для користувачів які бажають уточнювати свої дані або вводити власні параметри.

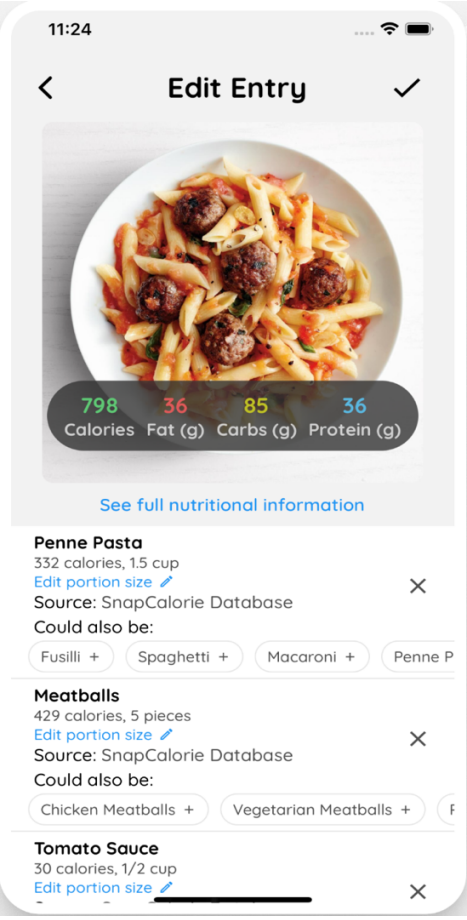


Рис.1.2. Інтерфейс застосунку Snap Calorie

1.2.3 Centa

Додаток вирізняється серед аналогів тим що орієнтований не лише на підрахунок калорійності, але й на глибший аналіз поживного складу страв [4]. Цей застосунок дозволяє автоматично визначати кількість білків, жирів та вуглеводів в кожній страві (Рис.1.3.). Також він інтегрується з різними пристроями, такими як розумні годинники та фітнес-трекери, і збирає дані про фізичну активність користувача. Це забезпечує комплексний підхід до оцінки здоров'я, поєднуючи аналіз харчування з рівнем активності.

Особливою функцією Centa є можливість створювати персоналізовані плани харчування на основі отриманих даних. Цей застосунок корисний для

людей, які прагнуть досягти конкретних цілей, наприклад, схуднути або підтримувати форму. Проте основним недоліком чи викликом для цієї системи залишається точність розпізнавання складних страв, таких як супи або змішані продукти. Для покращення результатів можна вдосконалити алгоритми розпізнавання та розширити можливості введення користувачем підредагованих даних.

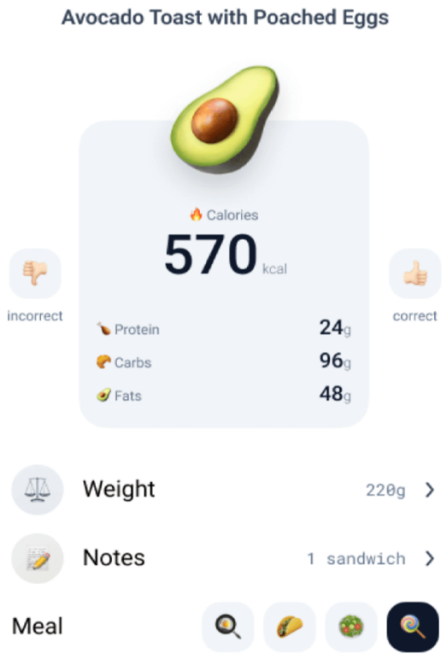


Рис.1.3. Інтерфейс застосунку Centa

1.2.4 CaloPal

CaloPal має унікальний функціонал, він поєднує аналіз зображень їжі та сканування штрих-кодів [5]. Такий підхід дозволяє користувачам отримувати інформацію не лише про страви, але й про конкретні продукти харчування, які продаються наприклад в супермаркетах. Система автоматично визначає калорійність, вміст поживних речовин і навіть рекомендує оптимальний час для прийомів їжі (Рис.1.4.).

Одним із нововведень CaloPal є інтеграція з календарем користувача, що дозволяє налаштовувати нагадування про прийоми їжі. Це робить його зручним для людей, які прагнуть дотримуватися регулярного режиму харчування. Але застосунок ще потребує технічних вдосконалень, так як є складнощі з розпізнаванням нестандартних штрих-кодів та залежність від інтернет-з'єднання. Оптимізація алгоритмів та додавання офлайн-режиму могли б значно розширити його аудиторію.

Рис.1.4. Інтерфейс застосунку CaloPal

1.2.5 MacrosAI

Цей додаток є одним із найбільш універсальних рішень у цій сфері. Його особливість полягає в мультиканальному підході до введення даних: тобто користувачі можуть завантажувати фото, вводити текстовий опис страви або сканувати штрих-коди [6]. Така різноманітність робить застосунок доволі зручним для людей з різними уподобаннями. MacrosAI використовує штучний

інтелект для аналізу інгредієнтів і пропорцій і автоматизує процес підрахунку калорій.

Цікавим моментом є можливість створення харчових щоденників, що дозволяє користувачам відстежувати свої звички протягом тривалого часу (Рис. 1.5.). Однак MacrosAI може демонструвати різницю в результатах при введенні даних різними способами і це викликає потребу в оптимізації обробки інформації, а також потребу у впровадженні функції самонавчання алгоритмів, щоб вони адаптувалися до індивідуальних звичок користувачів.

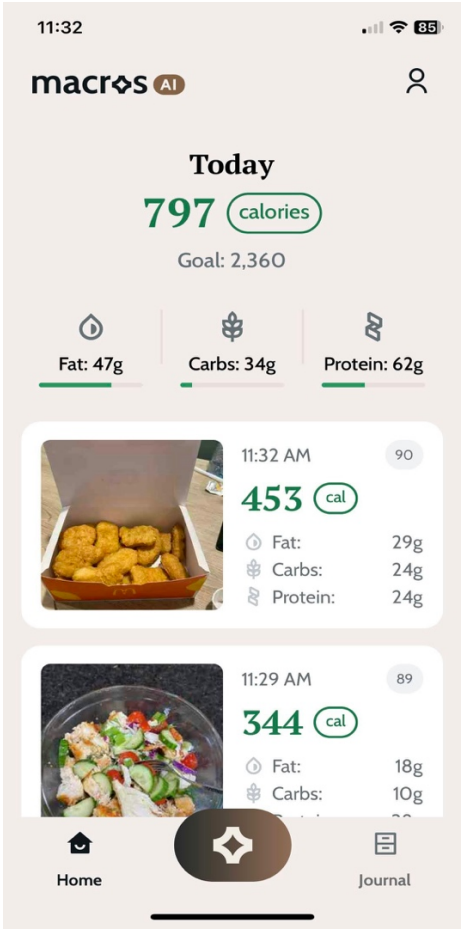


Рис.1.5. Інтерфейс застосунку MacrosAI

1.2.6 Висновки щодо існуючих застосунків

Загалом, аналіз сучасних додатків для розпізнавання їжі та оцінки її калорійності свідчить про те, що технології штучного інтелекту й комп’ютерного

зору мають значний потенціал для спрощення контролю за харчуванням і підвищення точності дієтичного аналізу. Такі системи автоматизують ключові процеси, зокрема розпізнавання страв, підрахунок калорій і визначення макронутрієнтів, а також надають практичні рекомендації щодо харчування. Це робить їх корисними як для пересічних користувачів, так і для тих, хто активно дбає про своє здоров'я або має особливі дієтичні потреби.

Серед основних тенденцій у цій сфері можна відзначити рух до більш комплексних і персоналізованих рішень. Багато сучасних додатків уже виходять за межі простого підрахунку калорій, додаючи такі функції, як аналіз поживного складу, інтеграція з різними пристроями для збору даних про фізичну активність, а також надання індивідуальних рекомендацій. І це відповідає запитам сучасних користувачів, які прагнуть отримувати інструменти для комплексного управління своїм здоров'ям.

Проте, попри очевидні досягнення, залишаються певні виклики, які ще потрібно вирішити. Наприклад, багато додатків сильно залежні від стабільного інтернет-з'єднання, що обмежує їхнє використання у зонах із поганим доступом до мережі. Також слабким місцем багатьох рішень є точність розпізнавання складних страв або багатокомпонентних продуктів, що вимагає вдосконалення алгоритмів. Крім того, для багатьох споживачів, які хочуть мати більш гнучкі й точні інструменти, проблемою є відсутність можливості вручну коригувати дані або адаптувати систему до індивідуальних потреб користувачів.

Оцінюючи перспективи розвитку цієї технології можна припустити, що майбутнє таких рішень полягає в покращенні точності алгоритмів, розширенні функціоналу, зокрема для роботи в офлайн-режимі, а також інтеграції з іншими платформами й пристроями. Такий розвиток зробить ці технології ще зручнішими та доступнішими для щоденного використання.

1.3 Характеристики існуючих рішень

У сучасних системах автоматичного аналізу харчування застосовуються інноваційні підходи на основі штучного інтелекту та комп'ютерного зору. Ці рішення мають ряд властивостей, які формують їхню ефективність, зручність використання та точність результатів. В даному підрозділі проаналізуємо властивості застосунків з метою виявлення їх сильних сторін, недоліків та особливостей реалізації. Це дозволить сформулювати вимоги до розробки власної системи.

Однією з ключових загальних характеристик є орієнтація на користувацький досвід. Усі розглянуті застосунки пропонують простий інтерфейс, що забезпечує швидкий доступ до основного функціоналу, такого як фотографування страви, підрахунок калорій або сканування штрих-кодів. Більшість рішень підтримують інтеграцію з іншими платформами, наприклад, фітнес-додатками, що розширює їхню функціональність.

Другою важливою властивістю є доступність документації. Наприклад, Calorie Mama та Snap Calorie мають детальні інструкції з використання, що спрощує взаємодію навіть для новачків. Однак деякі застосунки, такі як MacrosAI, надають обмежену кількість навчальних матеріалів, що може бути бар'єром для нових користувачів.

Третій аспект – це гнучкість роботи з даними. Snap Calorie, наприклад, використовує датчики глибини для оцінки розмірів порцій, а MacrosAI підтримує мультиканальний ввід, що дозволяє вводити дані у вигляді фото, тексту або штрих-коду. Така гнучкість підвищує точність результатів та зручність використання.

З технічної точки зору, одним із головних аспектів є точність розпізнавання. Calorie Mama використовує глибокі нейронні мережі для класифікації тисяч страв, але її точність знижується у випадку складних або багатокомпонентних страв. Snap Calorie демонструє вищу точність завдяки використанню сучасних алгоритмів та технології оцінки розмірів порцій. Centa

спеціалізується на визначенні макронутрієнтів, використовуючи дані про інгредієнти та інтегруючись із носимими пристроями.

Ще однією важливою властивістю є продуктивність та швидкість роботи. Усі аналізовані системи забезпечують майже миттєвий підрахунок калорій, що є важливим для користувачів. Проте це досягається за рахунок обробки даних у хмарі, що створює залежність від стабільного інтернет-з'єднання.

Особливості реалізації також включають використання штучного інтелекту для покращення точності. Наприклад, MacrosAI дозволяє системі навчатися на основі індивідуальних харчових звичок користувача, що підвищує персоналізацію результатів. CaloPal інтегрує функції нагадувань, які сприяють регулярності харчування.

1.4 Постановка задачі

Аналіз сучасних рішень для автоматичного розпізнавання їжі та оцінки калорійності виявив, що, попри значний прогрес у цій галузі, залишаються невирішені аспекти, які можна інтегрувати в нову систему. Зокрема, сучасні додатки зосереджені переважно на аналізі страв та розрахунку їх калорійності, однак не враховують потреби користувачів у зручному управлінні запасами продуктів, плануванні покупок та пошуку рецептів. Ці аспекти можна інтегрувати у майбутню систему, зробивши її більш комплексною та корисною.

1.4.1 Мета розробки

Розробити універсальну систему для автоматичного розпізнавання їжі, оцінки калорійності страв і управління харчовими звичками, яка доповнена функціями ведення обліку продуктів, інтеграції з вітчизняними магазинами для замовлення продуктів та рекомендаціями рецептів на основі наявних інгредієнтів.

1.4.2 Основні вимоги до системи

1. Точність розпізнавання:
   - Використання моделей глибокого навчання для класифікації страв та визначення їх калорійності.
   - Аналіз декількох продуктів на одному фото.
2. Гнучкість введення даних:
   - Можливість завантаження інформації про їжу через фото, текстовий опис або використовуючи готові рецепти.
   - Інтеграція з сенсорами для оцінки порцій (наприклад, через технології глибини на смартфонах).
3. Управління продуктами:
   - Ведення обліку наявних продуктів користувача.
   - Автоматичне відстеження закінчення запасів продуктів.
   - Генерація нагадувань про необхідність поповнення запасів.
   - Інтеграція з вітчизняними магазинами (наприклад, "Сільпо") для автоматичного формування кошика продуктів із посиланнями на сайт LOKO для швидкого замовлення.
4. Рекомендації рецептів:
   - Генерація списку рецептів на основі продуктів, що є в наявності у користувача.
   - Пропонування варіантів страв з продуктами на фото.
5. Автономність:
   - Забезпечення офлайн-функціональності для основних задач, таких як введення даних про продукти чи розпізнавання їжі.
6. Продуктивність:
   - Швидка обробка даних.
   - Оптимізовані алгоритми для ефективної роботи з великими базами даних продуктів та рецептів.
7. Персоналізація:

- Можливість адаптації системи до індивідуальних харчових звичок користувача.
- Запам’ятовування уподобань для автоматичного формування рекомендацій.

8. Розширена аналітика:
   - Надання користувачеві статистики щодо споживання калорій, використання продуктів та дотримання дієтичних цілей.
9. Інтуїтивний інтерфейс:
   - Забезпечення простого у використанні інтерфейсу для швидкого доступу до всіх функцій.
   - Наявність адміністративного запису для швидкого налаштування даних.

1.4.3 Додаткові функції

Система матиме можливість інтеграції з локальними магазинами для замовлення продуктів. Наприклад, при закінченні запасу конкретного інгредієнта система автоматично надаватиме користувачеві список відповідних магазинів, таких як "Сільпо", із посиланням на їхній сайт для швидкого додавання товарів у кошик. Це зручність може доповнюватися рекомендаціями щодо акцій або доступних альтернативних продуктів.

Ще одним нововведенням стане функція генерації рецептів. На основі аналізу продуктів, які є у користувача, система пропонуватиме варіанти страв, що можна приготувати. Такий підхід не лише сприятиме оптимізації використання продуктів, але й стане корисним для тих, хто хоче урізноманітнити своє меню або дотримуватися конкретного дієтичного режиму.

1.4.4 Очікувані результати

Запропонована система не лише спростить процес моніторингу харчування, але й забезпечить інтеграцію з екосистемою управління продуктами та покупками. Її унікальність полягає у поєднанні функцій розпізнавання їжі,

управління запасами продуктів та надання рекомендацій, що робить її корисною як для звичайних користувачів, так і для професіоналів у сфері харчування. У результаті користувачі отримають багатофункціональний інструмент, який не має аналогів на українському ринку.

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі було здійснено аналіз наявних рішень для автоматичного розпізнавання їжі та визначення її калорійності, таких як Calorie Mama, Snap Calorie, Centa, CaloPal і MacrosAI. Було детально розглянуто функціональні можливості цих додатків, їхні технічні характеристики, переваги та недоліки.

Проведений аналіз показав, що більшість сучасних додатків характеризуються зручними інтерфейсами, широкими можливостями інтеграції з іншими платформами та автоматизацією процесу підрахунку калорій. Однак також виявлено значні обмеження: залежність роботи від стабільного інтернет-з'єднання, низька точність розпізнавання складних або багатокомпонентних страв, відсутність функцій для управління запасами продуктів, а також обмежені можливості персоналізації рекомендацій.

Окрім базового підрахунку калорійності, жоден із розглянутих додатків не надає можливості ведення обліку продуктів, автоматизації поповнення запасів, інтеграції з українськими торговими мережами для формування кошика чи рекомендації рецептів на основі доступних інгредієнтів. Ці аспекти є важливими для забезпечення комплексного підходу до планування харчування.

На основі проведеного аналізу сформульовано ключові вимоги до розробки власного застосунку, який об'єднуватиме точне розпізнавання продуктів, управління запасами та персоналізацію рецептів. Виявлені обмеження та потреби користувачів стануть основою для створення унікального рішення, яке відповідатиме сучасним запитам ринку. Це підтверджує актуальність розробки системи Hroom (або Хрум), яка зможе запропонувати користувачам новий рівень функціоналу та зручності.

РОЗДІЛ 2
МЕТОДОЛОГІЯ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ ЇЖІ ТА ВИЗНАЧЕННЯ КАЛОРІЙНОСТІ

2.1 Основна концепція застосованої методології

Розробка системи розпізнавання їжі та визначення її калорійності базується на аналізі великого обсягу зображень продуктів харчування. Зібраний набір даних включає численні зображення їжі, а також фотографії, що представляють різні категорії, такі як яблука, банани, апельсини тощо. Модель налаштовується так, щоб могла розпізнавати об’єкти, що потрапляють у зону видимості веб-камери. Основні операції, які виконуються системою, включають обрізання та зміну розміру зображень, що дозволяє видаляти зайві частинки кадру та зосереджувати увагу на необхідних елементах.

Для анотації даних використовується спеціалізований графічний інструмент, що створює рамки або контури навколо об’єктів на зображеннях. Основним алгоритмом, що використовується для розпізнавання, є YOLO (You Only Look Once), який застосовує єдину нейронну мережу для аналізу зображення. Алгоритм розділяє зображення на кілька областей, прогнозує обмежувальні рамки для кожної з них та визначає ймовірності відповідних класів.

Головна перевага системи полягає в її здатності ідентифікувати кілька об’єктів на одному зображенні одночасно, надаючи точні обмежувальні рамки для кожного об’єкта. І додатково, після ідентифікації продуктів система здійснює розрахунок калорійності на основі бази даних харчової цінності.

2.2 Використані методи машинного навчання

У розробці системи використовуються сучасні методи машинного навчання, зокрема згорткові нейронні мережі (CNN) та алгоритми для виявлення

об’єктів, такі як YOLO (You Only Look Once). Ці технології забезпечують високу швидкість і точність розпізнавання зображень у реальному часі.

Метод YOLO базується на застосуванні єдиної згорткової нейронної мережі до всього зображення, що дозволяє одночасно передбачати ймовірність класів і координати обмежувальних рамок для кожного об’єкта. Такий підхід забезпечує швидку обробку даних і високу ефективність, оскільки вся картина аналізується як єдине ціле. CNN у цьому процесі відповідає за екстракцію ознак із зображення, які використовуються для класифікації об’єктів і прогнозування їхніх меж.

2.2.1 Intersection Over Union (IoU)

Одним із ключових методів оцінки точності моделі розпізнавання є показник Intersection Over Union (IoU), який використовується для вимірювання відповідності між прогнозованими та реальними (базовими) обмежувальними рамками об’єктів. IoU обчислюється за наступною формулою:

$$IoU = \frac{\text{Площа перекриття}}{\text{Площа об’єднання}}$$

Для застосування цього показника потрібні такі дані:

- Обмежувальні рамки базової істини (ground truth) – це рамки, які точно окреслюють об’єкт у зображенні відповідно до розмітки даних.
- Прогнозовані обмежувальні рамки (predicted bounding boxes) – це рамки, які модель формує під час аналізу зображення.

IoU дозволяє оцінити, наскільки добре модель визначає межі об’єкта. Його значення може змінюватися від 0 (повна відсутність перекриття) до 1 (ідеальне перекриття). Цей показник часто використовується для налаштування та вдосконалення моделей розпізнавання.

Застосування IoU є важливим етапом у процесі тренування й тестування системи, адже він допомагає виявити потенційні помилки у прогнозах моделі та коригувати параметри для досягнення більш високої точності. У нашій системі

цей метод забезпечує якісне розпізнавання їжі навіть за умови наявності складних фонових об’єктів або кількох продуктів на одному зображенні.

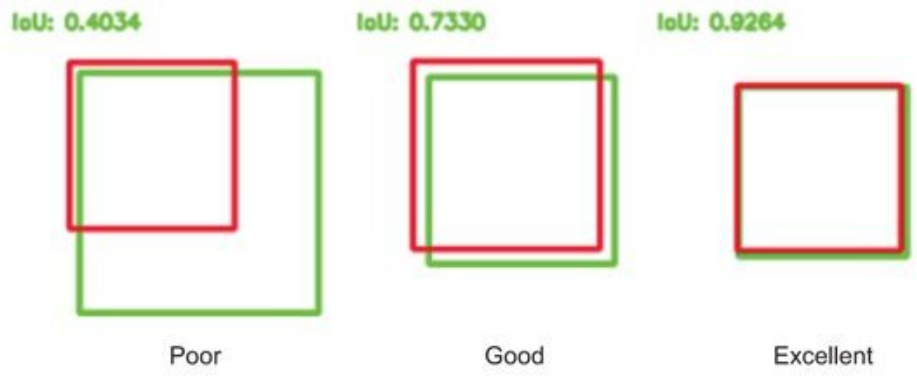


Рис.2.1. Приклад обчислення IoU [7]

2.2.2 Підхід YOLO

У розробці системи для розпізнавання їжі та визначення калорійності підхід YOLO (You Only Look Once) став ключовим методом завдяки своїй ефективності та можливості роботи в реальному часі. Цей алгоритм, який вперше запропонував Джозеф Редмон у 2016 році, дозволив зробити значний прорив у сфері виявлення об’єктів. На сьогодні існує кілька версій та архітектур YOLO які відрізняються між собою балансом між швидкістю та точністю. Ці особливості дозволяють адаптувати алгоритм до різних завдань і потреб.

YOLO використовує однопрохідний підхід для обробки зображень і розглядає виявлення об’єктів як задачу регресії. Цей підхід дозволяє одночасно прогнозувати обмежувальні області та ймовірності класів і це дає змогу моделі виділяти кілька об’єктів на одному зображенні. На відміну від інших алгоритмів, які поділяють процес виявлення на два етапи (генерація пропозицій і класифікація), YOLO виконує все за один крок, що значно прискорює обробку.

Основний принцип роботи YOLO полягає в тому, що він розділяє зображення на сітку комірок, кожна з яких відповідає за виявлення об’єкта, якщо той знаходиться в її межах. Для кожної комірки модель генерує прогноз

обмежувальних рамок, їхніх розмірів і оцінку довіри, яка визначає наявність або відсутність об’єкта. Оцінка довіри обчислюється за формулою:

$$\text{оцінка довіри} = p\ (\text{об'єкту}) \times IoU\ {}^{truth}_{pred}$$

де:

p(об’єкт) – ймовірність наявності об’єкта в комірці (1 – об’єкт є, 0 – відсутній);

IoU(Intersection Over Union) – метрика, що визначає перекриття між прогнозованою та реальною (істинною) обмежувальною рамкою.

Функція втрат у YOLO враховує:

1. Координати обмежувальних рамок – модель штрафується за похибки в прогнозах координат x, y.
2. Розміри рамок – втрата обчислюється за відхилення ширини та висоти рамки w, h.
3. Оцінку довіри – окремо для комірок, які містять об’єкти, і тих, які їх не містять.
4. Класифікацію – похибка в прогнозуванні ймовірностей класів для кожного об’єкта.

Загальна функція втрат включає всі ці компоненти, що дозволяє моделі максимально точно ідентифікувати об’єкти.

З моменту появи YOLO було розроблено кілька його версій, кожна з яких пропонувала вдосконалення:

- YOLOv1 – базова версія, що демонструвала швидкість 45 кадрів за секунду.
- YOLOv2 – оптимізована архітектура, яка підтримувала зображення з високою роздільною здатністю.
- YOLOv3 – впроваджено Darknet-53, що дозволило проводити багатомасштабне виявлення.
- YOLOv4 – покращення реальної продуктивності за рахунок використання CSPDarknet-53 і PANet для агрегації ознак.
- YOLOv5 – забезпечує автоматичне налаштування якірних блоків і є гнучкішим для користувацьких наборів даних.

У нашій системі розпізнавання їжі YOLO забезпечує високу точність і швидкість ідентифікації продуктів харчування на зображеннях. Це дозволяє визначати кілька видів їжі одночасно, аналізувати їхні характеристики та точно обчислювати калорійність на основі ідентифікованих об’єктів. Такий підхід робить систему універсальним інструментом для всіх типів користувачів.

2.3 Проєктування системи

Проєкт Hroom (або Хрум) — це система для передбачення калорійності їжі, що використовує технології машинного навчання та працює на архітектурі з розподіленими компонентами фронтенду та бекенду. Використання відповідних технологій визначало необхідність вибору інструментів, що забезпечують ефективну розробку, підтримку та масштабованість проєкту.

В наш час є багато різних підходів та платформ для додатків, хтось орієнтується на розробоку мобільних додатків, хтось робить повноцінні застосунки для операційних систем, проте найбільш універальними є наразі веб-додатки оскільки браузери є на усіх пристроях як мобільному телефоні так і на комп’ютері з будь-якою операційною системою, тому для розроки нашої системи клієнтська сторона буде саме веб-додатком.

Іншим важливим плюсом цього є те що існують такі речі як нативні веб-додатки – коли ми пишемо код ніби для веб-додатку, але в результаті маємо можливість розгортати наш додаток як для браузера так і для десктопу, телефонів, смарт пристроїв і так далі, наразі на цьому фокусу не буде, проте на перспективу клієнтську частину можна буде з легкістю модифікувати саме на такий підхід.

Головним аспектом при розробці веб-додатку є адаптивність, що згодом розглянемо як це будемо вирішувати, за рахунок адаптивності вміст веб-сторінки виглядає однаково добре, як на великому екрані комп’ютера так і на мобільному телефоні.

Будь-який веб-додаток не обійдеться без веб-серверу, який включає в себе саме логіку додатку, взаємодію з БД, синхронізацію стану та робить основні обчислення, тобто в нашому випадку клієнт – тонкий та основну роботу робить сервер, а клієнт просто відображає ті чи інші дані у зручному вигляді.

2.3.1 Архітектура додатку

В системі буде використано клієнт-серверну архітектуру, що випливає з попередніх тверджень. В рамках цієї архітектури користувачі системи – клієнти мають можливість виконувати ті чи інші запити, які оброблює сервер. В попередньому розділі було згадано про тонкий клієнт, який було покладено в основу цієї систему, було б дивно якби наприклад обробка фото з продуктами клієнта відбувалась на клієнті, оскільки для цього йому треба було б мати все необхідне для цього, знати щось про те як працюють нейронні мережі і т.д., саме тому всі складні обчислення лягають на плечі сервера.

Тобто маємо клієнт та сервер або як їх ще часто звуть frontend та backend частини системи, кожна з них ізольована і взаємодіє один з одним на основі певного API протоколу, який нам потрібно буде реалізувати згодом.

Для API було обрано HTTP протокол з використанням підходу REST API. REST – це підхід до архітектури мережевих протоколів та чітка структура того, як надається доступ до інформації та ресурсів системи. Згідно цього підходу дані будемо передавати у форматі JSON і дані системи будуть представлені як окремі ресурси, кожен такий ресурс ідентифікується унікальним початковим URI по типу /people, також є обмеження на використання HTTP методів та на їх відповідність діям, а саме:

- GET: Отримання ресурсу.
- POST: Створення нового ресурсу.
- PUT: Оновлення існуючого ресурсу.
- DELETE: Видалення ресурсу.

Інший важливий аспект, що для відповіді будемо використовувати HTTP статуси для опису характеру відповіді, наприклад 200 OK – для успішної дії, 201

Created – для успішного створення ресурсу, 404 Not Found – якщо ресурс не знайдено і так далі.

Також варто зазначити, що сервер не зберігає ніяку інформацію про стан клієнта між запитами та кожен запит має містити всю необхідну для нього інформацію.

При такому підході для авторизації та автентифікації зазвичай використовують або Cookie або HTTP хедери з певними даними для авторизації користувача, конкретно в нашій системі було обрано передавати цю інформацію в хедерах, а сам спосіб авторизації базується на JWT токені, який при авторизації генерується та шифрується секретним ключем або приватним ключем і за рахунок цього можемо визначити чи це валідний токен і чи видавала його наша система. На перспективу в JWT токені також можна зберігати додаткову інформацію про користувача в JSON форматі, поки що нам це не потрібно, але якщо будемо розвивати продукт це може стати в нагоді. Для простоти реалізації не будемо робити логіку renewal токену і у випадку коли час дії токену спливає користувачу потрібно буде заново ввійти в систему.

2.3.2 Вибір технологій для розробки ПЗ

Для розробки цієї системи в першу чергу нам потрібно обрати зручне середовище розробки з значним набором функціоналу, бажано, щоб там була підтримка Docker, інтерфейс для взаємодії з БД, можна було редагувати файли різних форматах з підсвіткою синтаксису, компілювати та запускати ті чи інші утиліти та інше. Враховуючи ці моменти, наразі одним з найбільш універсальних та зручних IDE є Intellij IDEA від компанії Jetbrains важливо зазначити, що для студентів вони надають безкоштовну версію Ultimate на час навчання, яка включає в себе доступ до величезної кількості додаткових плагінів, тож зможемо встановити плагіни для будь-яких потреб і зручно працювати над проєктом в рамках одного редактору коду.

Далі потрібно обрати систему контролю версій VCS тут доцільно буде взяти те що зараз є найпопулярнішим, а саме Git, де сам код буде на GitHub.

Саме тому для збереження коду проєкту використовується GitHub — одна з найпопулярніших платформ для керування вихідним кодом на основі Git. Вибір GitHub зумовлений його зручністю у роботі з командною розробкою, а також інтеграцією з багатьма іншими сервісами. Серед переваг GitHub варто виділити:

- Pull Requests – можливість створювати запити на внесення змін, обговорювати їх з командою і проводити рев'ю коду. Це підвищує якість коду і дозволяє уникнути конфліктів.
- GitHub Actions – автоматизація процесів CI/CD (Continuous Integration/Continuous Deployment), що дозволяє автоматично виконувати тести і деплоїти додаток у різні середовища. Незважаючи, що поки що нам цей функціонал не потрібен, проте коли час дійде до розробки стартап проєкту і систему буде розробляти вже ціла команда, безумовно це буде невід'ємною функціональністю
- Проєкти та Issues – для відстеження задач, їхньої пріоритизації та поділу на етапи. Це зручно для організації процесу розробки і забезпечує видимість для всієї команди. Аналогічно попередній фічі, хоч нам це зараз непотрібно далі без цього не обійтись

Також варто зазначити, що згодом можна буде настроїти купу ріхних правил автоматизації, правил для роботи з гілками в проєкті, розмежувати доступи до тих чи інших ресурів для кожного члена команди та навіть вести документацію та облік релізів в одному місці.

В якості бази даних системи було обрано PostgreSQL— це потужна, відкрита (open-source) реляційна система управління базами даних (РСУБД). Вона підтримує широкий набір функцій для зберігання, організації та запиту даних.

- Реляційна база даних: використовує Structured Query Language для маніпуляції даними. Дані зберігаються в таблицях з чітко визначеними полями та зв'язками.

- Підтримка ACID-транзакцій: забезпечує цілісність і надійність даних завдяки підтримці атомарності, узгодженості, ізоляції та довговічності транзакцій.
- Розширюваність: можливість створювати власні типи даних, функції, індекси та багато іншого.
- Підтримка JSON і NoSQL-функцій: дозволяє зберігати та обробляти JSON-дані, що зручно для сучасних додатків. В нашому випадку є дані які не хотілося б нормалізувати, а тримати прямо в колонці JSON типу в тій же таблиці до якої відносяться дані
- Безпека: підтримка автентифікації, шифрування, контроль доступу на рівні рядків та інші механізми захисту.
- Масштабованість: може обробляти великі обсяги даних і кількість запитів завдяки функціям паралельного виконання.

Для контенеризації та для локальної розробки використовуються Docker та Docker Compose для налаштування та запуску сервісів як єдиної системи, що включає бекенд, фронтенд та базу даних. Це забезпечує простоту в управлінні і масштабуванні, а також дозволяє тестувати систему на локальних машинах з аналогічними конфігураціями, як і в продакшн середовищі.

Для фронтенд частини було обрано мову програмування JavaScript – це динамічна мова з неявною типізацією, вона ідеально підходить для розробки клієнтських застосунків адже не потрібно витрачати час на опис сутностей, сигнатур методів і так далі, одразу можемо почати писати логіку без зайвих кроків. На перспективу можна буде розглянути Typescript якщо проєкт матиме потребу в типізації через те що буде набагато більше сутностей, модулей та логіки, але на поточному етапі це непотрібно.

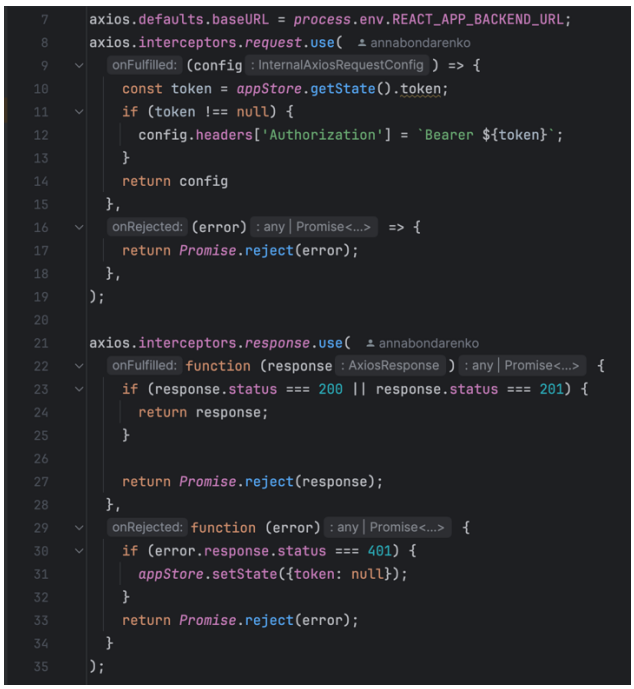
Для управління залежностями на фронтенді використовується npm (Node Package Manager). Це стандартний інструмент для роботи з JavaScript-проєктами, зокрема, проєктами на React, який використовується в "Hroom". npm дозволяє швидко додавати бібліотеки та плагіни, а також спрощує процес оновлення залежностей.

В наш час більшість додатків та систем не пишуть повністю з нуля, а використовують фреймворки та бібліотеки і даний проєкт не виключення, далі буде перелічено все що було використано в додатку.

- React – фреймворк для створення користувацького інтерфейсу. React забезпечує компонентний підхід, що дозволяє створювати масштабовані та підтримувані додатки. Він надає лаконічний та зрозумілий спосіб розробки інтерфейсу клієнтського застосунку а також дає готові рішення для контролю станів застосунку, коли робити рендер, а коли ні – це забезпечує швидкодію застосунку без втрати читабельності коду.
- MUI (Material-UI) – бібліотека UI-компонентів (@mui/material та @mui/icons-material). MUI надає зручний набір готових компонентів для створення естетично привабливих інтерфейсів, таких як кнопки, діалоги, меню тощо. Це значно зменшує час на розробку і робить додаток привабливим з точки зору користувачів. MUI базується на підході material design в якому вже є сталий набір правил для компонентів, також важливим аспектом цього підходу є адаптивність, завчасно пишемо так, щоб код виглядав однаково добре на всіх пристроях. Також на відміну від інших схожиш бібліотек маємо доступ міняти як хочемо ті чи інші аспекти компонентів за рахунок функціоналу тем. Тобто можна поміняти кольори, шрифти, стандартні аргументи компонент, заміняти певні компоненти нашими реалізаціями і так далі. Це дуже важливо оскільки на поточному етапі розробкою займаєьться тількии одна людина, немає дизайнера тому використати вже існуючий підхід виглядає раціонально.
- Recharts – бібліотека для побудови графіків і діаграм. У проєкті використовується для візуалізації статистики по калоріях, що допомагає користувачам відслідковувати свої харчові звички зручним і зрозумілим способом. Досить зріла бібліотека з зрозумілими набором

функціоналу та гарною документацією за потреби зможемо додати ще більше графіків та візуалізацій, коли прийде час.

- Zustand – менеджер станів додатку, який використовується для зберігання токенів аутентифікації та інших глобальних даних додатку. Ця бібліотека є легковажною і простою у використанні порівняно з більш складними рішеннями на кшталт Redux, що полегшує підтримку проєкту. Не зважаючи на те що React вже надає нам певні хуки по типу useState та має функціонал контекстів бібліотеки такого роду є не опцією, а обов'язковою складовою гарного рішення, адже дають можливість ділити глобальний стан між компонентами додатку та мають чималий набір додаткових middleware модулів для дебагінгу, збереження даних стану в localStorage чи sessionStorage, що значно спрощує роботу з станом додатку.
- React Router – для навігації всередині додатку. Цей модуль дозволяє визначати маршрути і створювати багатосторінкові додатки, що полегшує навігацію для користувача.
- Axios – бібліотека для виконання HTTP-запитів, яка використовується для взаємодії з бекендом, наприклад, для завантаження зображень їжі та отримання інформації про користувачів або рецепти. Тобто саме він використовується для взаємодії з REST HTTP API нашої системи, він має чітку і зрозумілу структуру та його можна зручно конфігурувати під потреби системи, наприклад автоматично брати JWT токен, який отримаємо після авторизації та підкладати в хедери запитів та розлогінювати користувача при зпливі часу роботи JWT токену. (Рис.2.2.).



```
axios.defaults.baseURL = process.env.REACT_APP_BACKEND_URL;
axios.interceptors.request.use(  annabondarenko
  onFulfilled: (config : InternalAxiosRequestConfig ) => {
    const token = appStore.getState().token;
    if (token !== null) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config
  },
  onRejected: (error) : any | Promise<...>  => {
    return Promise.reject(error);
  },
);

axios.interceptors.response.use(  annabondarenko
  onFulfilled: function (response : AxiosResponse ) : any | Promise<...>  {
    if (response.status === 200 || response.status === 201) {
      return response;
    }

    return Promise.reject(response);
  },
  onRejected: function (error) : any | Promise<...>  {
    if (error.response.status === 401) {
      appStore.setState({token: null});
    }
    return Promise.reject(error);
  }
);
```

Рис.2.2. Приклад конфігурації Axios для обробки авторизації

- react-query – бібліотека для управління серверними запитами, яка спрощує кешування та синхронізацію даних, забезпечуючи оптимальну продуктивність і зручність роботи з даними. Вона дає можливість зручно опрацьовувати як query запити так і мутації і дає інформацію про стан виконання дії за рахунок таких полів як loading, error, data відповідно спрощує логіку демонстрації стану для користувача. А найважливіше це те що після кожної мутації ця бібліотека автоматично робить запит на сервер, щоб отримати свіжі дані.
- react-scripts – набір скриптів для створення та управління проєктом на основі Create React App, що спрощує конфігурацію вебпакету, Babel та інших інструментів. За рахунок неї нам непотрібно буде вести налаштування всього пов’язаного з реакт, Javascript, форматуванням коду, версіями ES і так далі.
- Також було використано dayjs – легковажна бібліотека для роботи з датами, оскільки встроєний в Javascript функціонал для роботи з датами досить простий і не дає всіх потрібних нам речей.

- Для роботи з формами було використано react-form – бібліотека для створення та управління формами у React, яка забезпечує зручний API для роботи з формами, включаючи валідацію та обробку даних, що полегшує створення інтерактивних компонентів, таких як форми редагування, створення ресурсів. (Рис.2.3.)

```
<form onSubmit={(e : FormEvent<HTMLFormElement> ) : void  => {
  e.preventDefault()
  e.stopPropagation()
  form.handleSubmit()
}}>
  <Stack spacing={2} m={2}>
    <Stack direction="row" justifyContent="space-between" alignItems="center" spacing={2} mb={1}>
      <form.Field
        name="name"
        children={(field : FieldApi<any, `.${string}`, undefined, undefined, any> ) => (
          <TextField label="Назва" placeholder="Назва" size="small" fullWidth
                     defaultValue={field.state.value} onBlur={field.handleBlur}
                     onChange={e : ChangeEvent<...>  => field.handleChange(e.target.value)}/>
        )}
      />

      <Typography variant="caption">{new Date(meal.created).toLocaleTimeString()}</Typography>
    </Stack>
    <form.Field
      name="total_calories"
      children={(field : FieldApi<any, `.${string}`, undefined, undefined, any> ) => (
        <TextField label="Калорії" placeholder="Калорії" size="small" fullWidth
                   slotProps={{input: {endAdornment: <InputAdornment position="end">ккал</InputAdornment>}}}
                   defaultValue={field.state.value} onBlur={field.handleBlur}
                   onChange={e : ChangeEvent<...>  => field.handleChange(e.target.value)}/>
      )}
    />
    <form.Field
      name="description"
      children={(field : FieldApi<any, `.${string}`, undefined, undefined, any> ) => (
        <TextField label="Опис" placeholder="Опис" size="small" fullWidth multiline
                   defaultValue={field.state.value} onBlur={field.handleBlur}
                   onChange={e : ChangeEvent<...>  => field.handleChange(e.target.value)}/>
      )}
    />
```

Рис.2.3. Приклад використання react-form для реалізації редагування прийомів їжі

Тепер перейдемо до бекенду, в якості мови програмування було використано Python і основними причинами для цього було – наявність великої кількості пакетів для роботи з штучним інтелектом, data-science, machine-learning, а також за рахунок динамічної типізації подекуди буде простіше робити ті чи інші речі.

Для керування залежностями бекенду використовується pip разом із venv для створення віртуального середовища. Це дозволяє ізолювати залежності проєкту, щоб уникнути конфліктів між різними проєктами.

Серед ключових бібліотек та фреймворків варто згадати:

- FastAPI – веб-фреймворк, що використовується для створення REST API бекенду. Використання FastAPI дозволяє швидко створювати сучасні, асинхронні сервіси завдяки підтримці асинхронних функцій, її DI (Dependency Injection) функціональності, а також зручному декларативному опису моделей даних із використанням pydantic. Дуже зручно ще те що для описаних нами ендпоінтів автоматично генерується OpenAPI специфікація, що спрощує ведення документації доступних REST ендпоінтів та навіть даже можливість генерувати клієнтські методи взаємодії з сервером, це матиме значний вплив, коли команда розшириться і треба буде взаємодіяти між собою бекенд та фронтенд розробникам.
- Модуль fastapi-users дозволяє легко додати аутентифікацію та управління користувачами до проєкту. Саме він використовується для генерації JWT токенів та їх перевірки при зверненні до сервера, щоб тільки авторизовані користувачі мали змогу користуватись нашою системою. Також ця бібліотека забезпечує такі функції як реєстрація користувачів, скидання паролів та верифікація електронної пошти.
- SQLAlchemy та asyncpg – для роботи з базами даних. SQLAlchemy використовується як ORM для взаємодії з PostgreSQL, що дозволяє створювати складні запити до бази даних, використовуючи Python-код. Бібліотека asyncpg забезпечує асинхронний доступ до бази даних, що робить додаток більш продуктивним і зменшує час очікування навіть за значного навантаження. Реляційна структура дозволяє чітко визначити взаємозв’язки між різними сутностями проєкту.
- sqladmin – бібліотека для створення адміністративного інтерфейсу для роботи з моделями SQLAlchemy. Це забезпечує простий спосіб керування даними у базі без потреби писати SQL-запити вручну чи використовувати додаткові застосунку для підключення до БД, таким

чином адміністратор системи матиме змогу зручно наповнювати систему даними.

- OpenCV та NumPy – для обробки зображень. Бібліотека OpenCV використовується для сегментації і аналізу зображень їжі, що завантажуються користувачами. Крім того використовується підмодуль dnn з OpenCV для створення нейронної мережі, що використовує завчасно натреновану модель для ідентифікації їжі на зображеннях. NumPy допомагає в математичних операціях над зображеннями, зокрема при їхньому перетворенні та маніпуляції пікселями.
- Uvicorn — сервер ASGI для запуска FastAPI. Uvicorn забезпечує високу продуктивність та підтримує асинхронність, що робить його чудовим вибором для роботи з FastAPI.

Технологічний стек проєкту "Hroom" обрано з урахуванням продуктивності, зручності в розробці та підтримці, а також масштабованості системи. GitHub забезпечує зручний контроль версій та співпрацю над проєктом, npm та pip разом із віртуальними середовищами полегшують управління залежностями, а обрані бібліотеки та фреймворки дозволяють досягти високої продуктивності та зручності використання додатку як для кінцевих користувачів, так і для розробників.

2.4 Структура проєкту

Проєкт являєю собою monorepo – тобто весь код лежить в одній теці розмежований чіткою структурою. Ось короткий опис структури проєкту:

Frontend:

- api.js – файл з описом API взаємодії з бекендом, включаючи налаштування axios для роботи з токенами та обробкою запитів.
- /recipes – тека з функціоналом сторінок рецептів.
- Recipe.jsx – компонент для відображення інформації про окремий рецепт.

- Recipes.jsx – компонент для відображення списку рецептів.
- hooks.js – спільні React Hooks, такі як useAuth для перевірки аутентифікації користувача.
- store.js – конфігурація стану додатку з використанням Zustand, включає зберігання токену аутентифікації.
- Login.jsx – сторінка авторизації користувача.
- index.js – початковий файл додатку, що рендерить основний компонент App.
- mui.jsx – налаштування компонентів Material-UI для використання кастомної поведінки посилань.
- AppContainer.jsx – контейнер для основного оформлення додатку.
- dashboard
- MealsStats.jsx – компонент для візуалізації статистики калорій.
- MealUpload.jsx – компонент для завантаження зображень їжі.

Backend:

- routers
- users.py – API ендпоінти для роботи з користувачами, аутентифікація та керування користувачами.
- foods.py – API ендпоінти для роботи з Food сутністю.
- meals.py – API ендпоінти для роботи з прийомами їжі (Meals), включаючи функцію розпізнавання їжі.
- groceries.py – API ендпоінти для роботи зі списками покупок.
- admin.py – опис адміністраторського інтерфейсу для SQLAlchemy моделей за допомогою sqladmin.
- config.py – читання конфігурації бекенду зі змінних середовища
- app.py – основний файл FastAPI додатку, що налаштовує роутери, адміністративний інтерфейс та залежності.
- db.py – опис усіх сутностей бази даних (User, Food, Recipe, Meal) та налаштування з’єднання з базою даних.

- dependencies.py – визначення залежностей для FastAPI, таких як авторизований користувач або екземпляр FoodDetector.
- food_detector.py – клас для розпізнавання їжі на зображеннях з використанням OpenCV.
- food_detector_model.py – опис моделей даних, що використовуються в процесі сегментації зображень і розпізнавання їжі.
- image_helpers.py – функції для роботи із зображеннями, наприклад, обрізання зображень за заданими координатами.
- schemas.py – визначення Pydantic схем для валідації даних, що передаються в API (наприклад, MealCreate, RecipeUpdate).
- main.py – файл для запуску додатку з використанням Uvicorn.

На бекенді використовується структура з декількома модулями:

Роутери де кожен роутер (Рис.2.4.), такий як recipes.py, meals.py, groceries.py, відповідає за роботу з конкретним ресурсом. Наприклад, meals.py забезпечує маршрути для завантаження фотографій їжі та отримання прогнозів по калорійності. Ця модульність підвищує читабельність коду та полегшує підтримку.

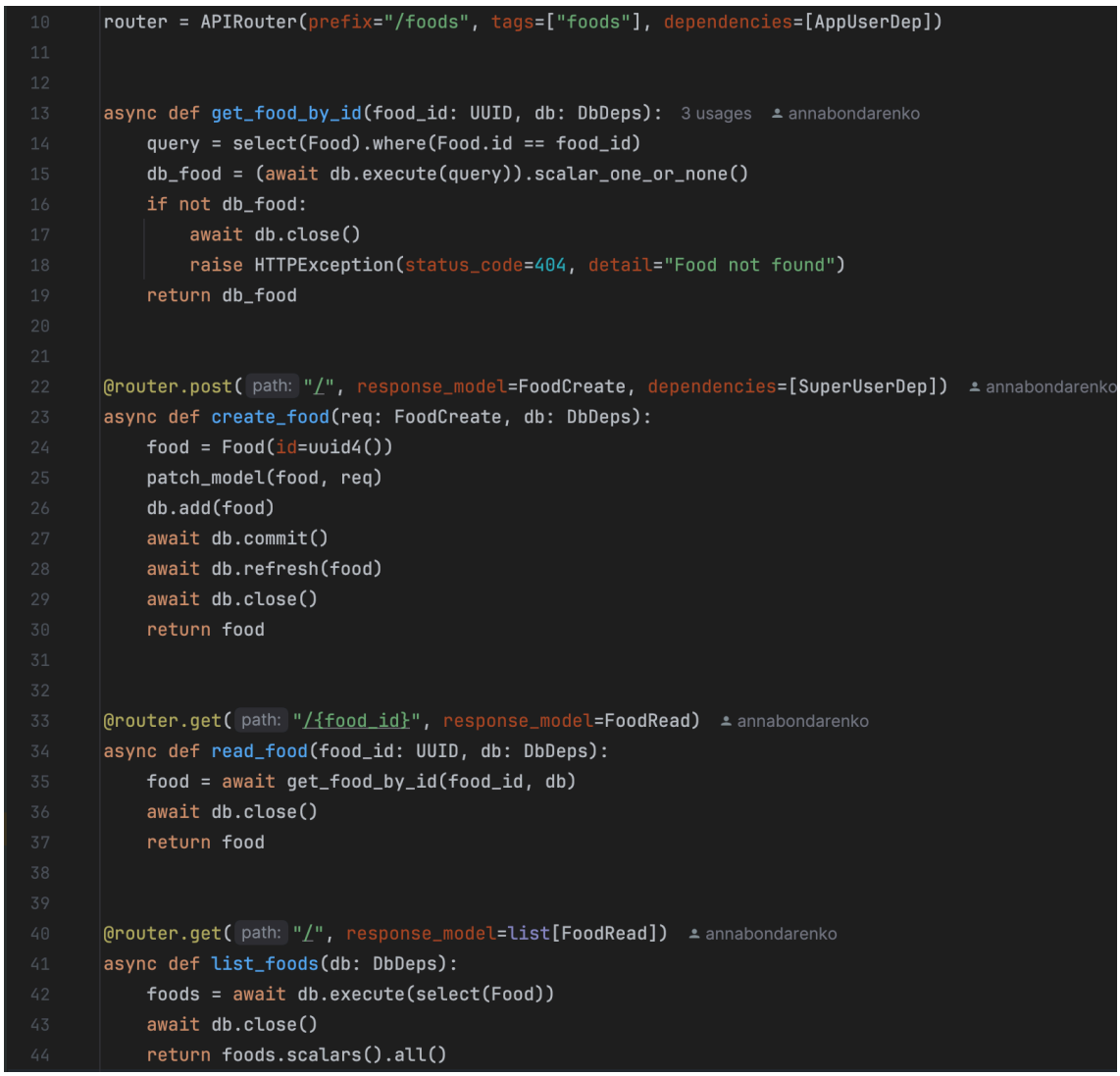


Рис.2.4. Приклад оголошення REST API ендпоінтів ресурсу foods за допомогою fastapi

Як видно з цього рисунку в залежностях роутера є залежність на користувача застосунку за рахунок чого всіми REST API можуть користуватись лише авторизовані користувачі, для випадків, коли певні ресурси мають бути доступні тільки для адміністраторів застосунку існує залежність на SuperUserDep, яка не дає працювати з ресурсом звичайним користувачам, тобто доступ є тільки у адміністраторів системи.

Моделі бази даних - це класи, такі як Recipe, Food, Grocery, представляють таблиці в базі даних. Завдяки використанню ORM SQLAlchemy, ці моделі дозволяють легко створювати, змінювати і видаляти записи з бази даних, використовуючи зручний API (Рис.2.5.).

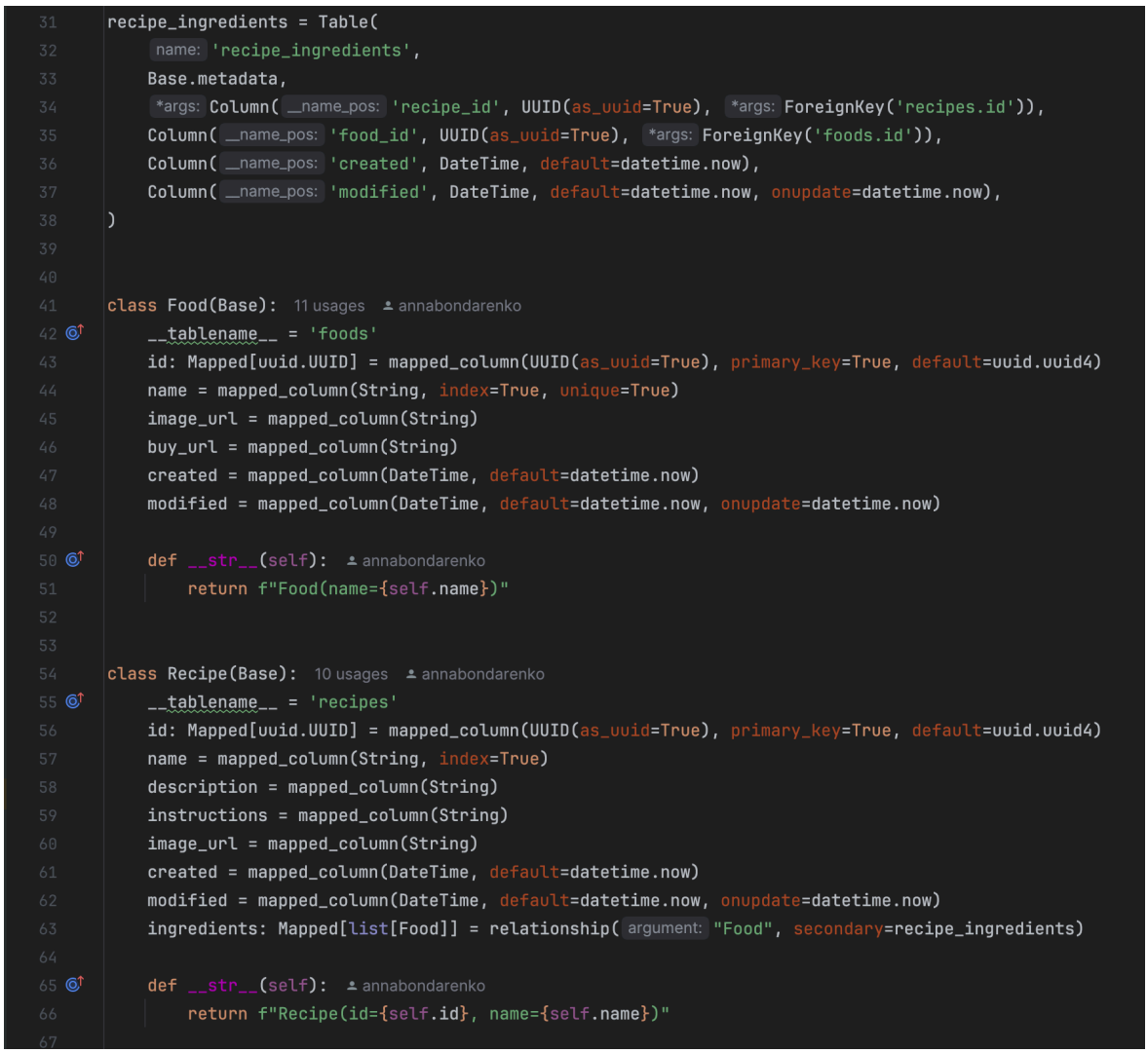


Рис.2.5. Приклад оголошення моделі БД за допомогою SQLAlchemy

На рисунку можна побачити приклад сутностей рецептів, які мають many-to-many реляцію реалізовану за рахунок проміжної таблички, за рахунок можливостей SQLAlchemy описати таку непросту реляцію стає досить тривіально.

На основі існуючих моделей бази даних формується адмінітративний інтерфейс додатку, що значно спрошує затрати на реалізацію адмініструванн додатку(Рис.2.6.).

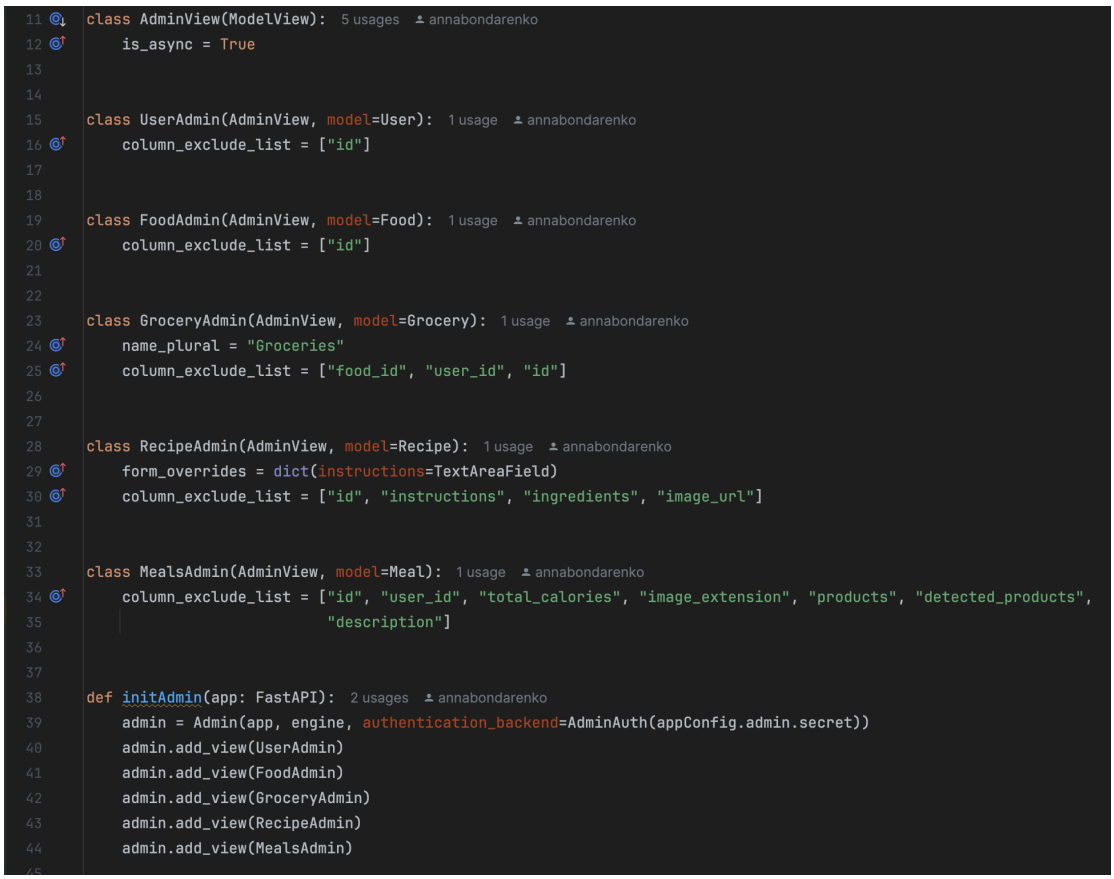


Рис.2.6. Конфігурування адміністративного інтерфейсу за допомогою sqladmin та моделей SQLAlchemy

Функції для обробки зображень, що знаходяться у модулі food_detector.py використовуються функції для сегментації та ідентифікації їжі на зображеннях. Обробка зображень включає сегментацію, знаходження контурів та розрахунок калорійності, що забезпечується моделлю, навченою для цього завдання. Крім того, у цьому модулі використовується підмодуль dnn з OpenCV для створення нейронної мережі, що використовує завчасно натреновану модель для ідентифікації їжі. OpenCV разом із кастомними функціями дозволяє гнучко працювати із зображеннями для досягнення потрібної точності.

На стороні сервера реалізовано ряд захисних процедур, по-перше всі секрети, паролі, логіни і посилання на БД не зберігаються в коді застосунку вони передаються через змінні середовища (Рис.2.7.).

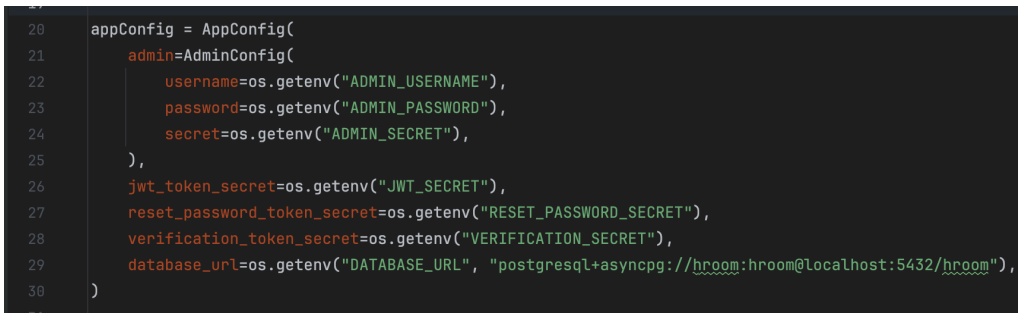


Рис.2.7. Читання секретів з змінних середовища

Далі ці секрети використовуються в fastapi-users та sqladmin модулях для засобів безпеки: в модулі users.py секрет для JWT токену шифрує токен, який користувач отримує при авторизації та присилає з кожним запитом всередині Authorization HTTP хедері, таким чином його не можна підробити адже цей секрет відомий лише застосунку, на додачу до цього секрет буде регулярно мінятись з кожним новим запуском застосунку, що зменшує шанси його витіку, аналогічним чином працює це і для sqladmin модулю.

REST API застосунку додатково захищено за допомогою політики CORS.

Також додатковим шаром захисту є те що користувачі не можуть доступатись до ресурсів, що їм не належать, це реалізовано за допомогою того, що у всіх ресурсах, які є специфічними для користувачів дані строго беруться з БД тільки якщо вони належать користувачеві (Рис.2.8.)

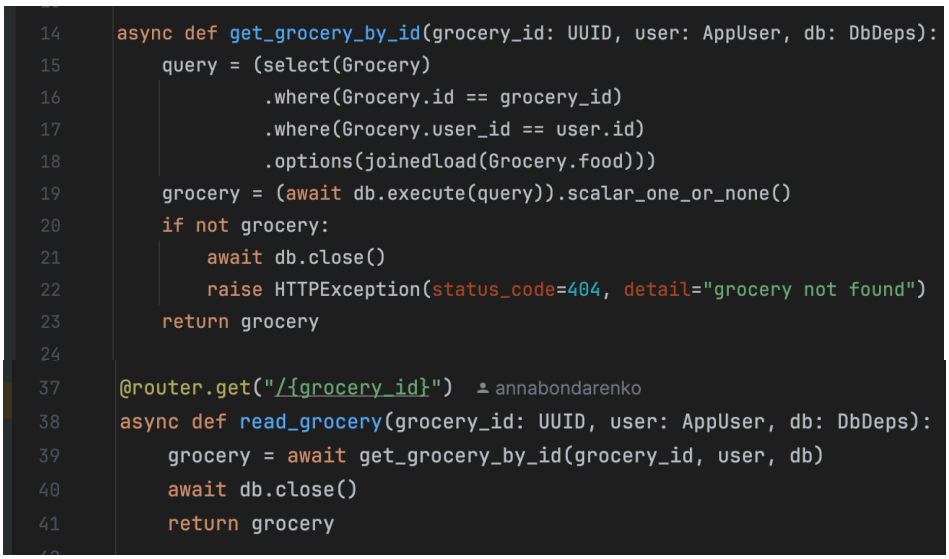


Рис. Приклад обмеження доступу до ресурсів, які належать користувачам

Фронтенд проєкту реалізовано за допомогою React і компонентна архітектура дозволяє розбити додаток на окремі блоки, такі як Dashboard, Meals, MealUpload, тощо, що значно спрощує розробку та тестування додатку.

React Hooks – у проєкті активно використовуються React Hooks, такі як useState, useEffect, useNavigate, useAppStore тощо. useState дозволяє керувати станом компонентів, наприклад, зберігати поточний вибір користувача або відстежувати стан завантаження. useEffect застосовується для виконання побічних ефектів, таких як завантаження даних при монтуванні компонента або реагування на зміну стану. useNavigate забезпечує навігацію між сторінками без перезавантаження, що створює зручніший досвід користування додатком. useAppStore використовується для доступу до глобального стану додатку за допомогою Zustand, що дозволяє спрощено ділитися інформацією між різними компонентами. Гарним прикладом використання хуків в застосунці є хук useAuth (Рис.2.9.) який реалізує перевірку, що користувач авторизований і має доступ до сторінки і якщо ні перенаправляє його на сторінку авторизації.

```
6   export function useAuth() : void {  Show usages  annabondarenko
7     const navigate : NavigateFunction  = useNavigate();
8     const token = useAppStore( selector: s => s.token);
9     useEffect( effect: () : void  => {
10      if (token == null) navigate("/login");
11    }, deps: [token, navigate]);
12  }
```

Рис.2.9. Приклад React hook для реалізації перевірки авторизації

Компонентний підхід, разом із використанням React Hooks, дозволяє створювати інтерактивний і легко підтримуваний інтерфейс користувача, який швидко реагує на зміни стану та події. Це робить додаток не тільки зручним для кінцевих користувачів, але й легким у розробці та тестуванні для розробників.

2.4.1 Проєктування бази даних

Оскільки в додатку використовується SqlAlchemy ORM то маємо підхід code-first коли спочатку описуємо наші сутності у вигляді класів з описом полів, які є у сутностей та описуємо зв'язки між ними, а самі таблиці генеруються автоматично з опису моделі даних з коду. Це один з найпопулярніших підходів в Python для роботи з БД, наразі немає потреби думати про міграції даних, проте на щастя вже є існуючі рішення для цього в контексті SqlAlchemy такі як Alembic, його на перспективу можна буде використати в майбутньому, коли проєкт перейде в активну фазу розробки стартап проєкту.

На рисунку зображено структуру БД застосунку, що була автоматично згенерована з нашої моделі, на цьому рисунку видно поля кожних сутностей, їх ключі, ключі для зв'язків, проміжні таблиці, які використовуються для Many to Many реляцій та всі інші види зв'язків між сутностями (Рис.2.10.).

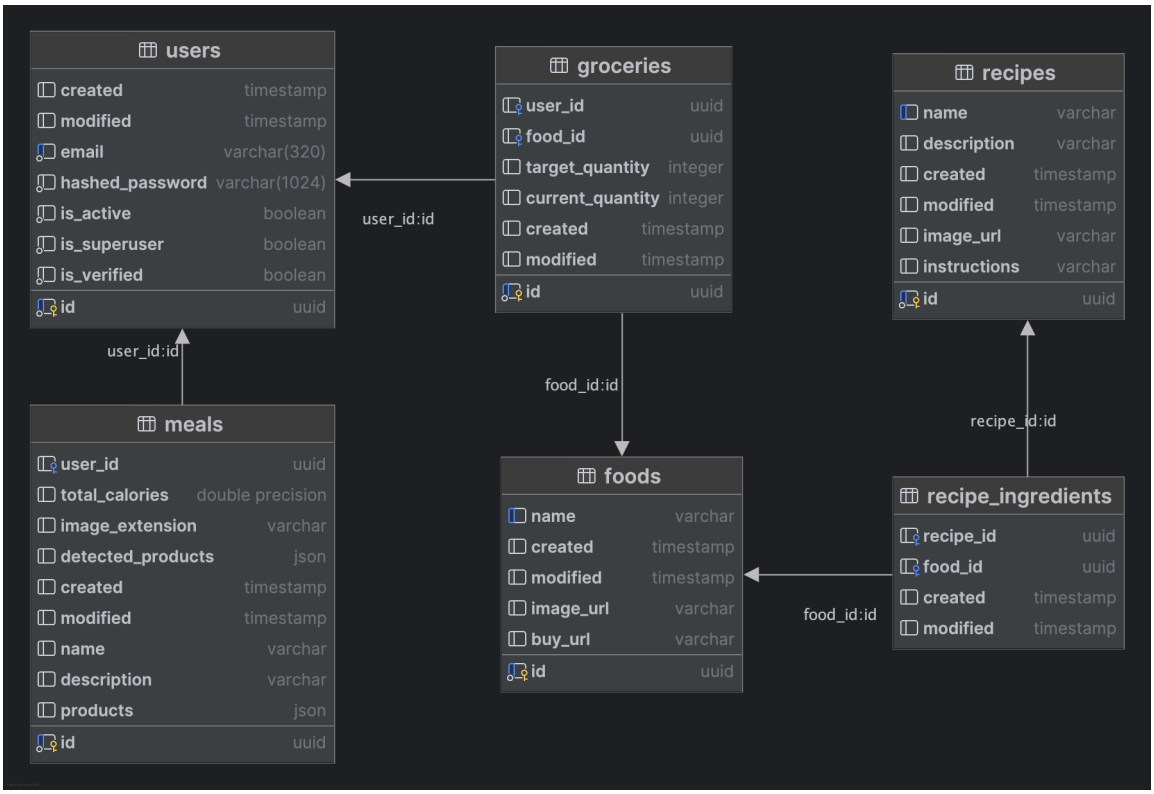


Рис.2.10. Діаграма сутностей

Далі наведено короткий опис усіх сутностей:

- Users – таблиця, що містить інформацію про користувачів системи. Поля включають:
    - id (uuid) – унікальний ідентифікатор користувача.
    - email (varchar) – адреса електронної пошти користувача, використовується для аутентифікації.
    - hashed_password (varchar) – зашифрований пароль користувача для безпеки.
    - is_active, is_superuser, is_verified (boolean) – відповідні статуси, які вказують на активність користувача, права адміністратора та підтвердження аккаунту.
    - created, modified (timestamp) – дати створення та останньої модифікації запису.
- foods – таблиця, що зберігає інформацію про продукти харчування. Поля включають:
    - id (uuid) – унікальний ідентифікатор продукту.
    - name (varchar) – назва продукту.
    - created, modified (timestamp) – дати створення та останньої модифікації запису.
    - image_url (varchar) – URL зображення продукту для візуального представлення.
    - buy_url (varchar) – посилання на покупку продукту, що може бути корисним для користувачів, які хочуть придбати продукт онлайн.
- recipes – таблиця з інформацією про рецепти. Поля включають:
    - id (uuid) – унікальний ідентифікатор рецепту.
    - name (varchar) – назва рецепту.
    - description (varchar) – опис рецепту, що допомагає користувачам зрозуміти його основні етапи.
    - created, modified (timestamp) – дати створення та останньої модифікації запису.
    - image_url (varchar) – URL зображення готової страви.

    - instructions (varchar) – детальні інструкції для приготування рецепту.
- groceries – таблиця, що зберігає інформацію про продукти в списках покупок користувачів. Поля включають:
    - id (uuid) – унікальний ідентифікатор запису.
    - user_id (uuid) – зв’язок з таблицею users, який вказує, якому користувачу належить цей список.
    - food_id (uuid) – зв’язок з таблицею foods, що вказує, який продукт входить у список покупок.
    - target_quantity (integer) – кількість продукту, яку користувач планує придбати.
    - current_quantity (integer) – поточна кількість продукту, що вже є у користувача.
    - created, modified (timestamp) – дати створення та останньої модифікації запису.
- meals – таблиця з інформацією про прийоми їжі користувачів. Поля включають:
    - id (uuid) – унікальний ідентифікатор прийому їжі.
    - user_id (uuid) – зв’язок з таблицею users, який вказує, кому належить цей прийом їжі.
    - total_calories (double precision) – загальна калорійність прийому їжі, що розраховується на основі складових продуктів.
    - image_extension (varchar) – тип файлу зображення, завантаженого користувачем.
    - detected_products (json) – інформація про продукти, розпізнані на зображенні їжі.
    - created, modified (timestamp) – дати створення та останньої модифікації запису.
    - name, description (varchar) – додаткова інформація про прийом їжі.
- products (json) – список продуктів, що входять до прийому їжі.

- recipe_ingredients – таблиця, що представляє зв’язок між рецептами та продуктами, які входять до рецепту. Поля включають:
- recipe_id (uuid) – зв’язок з таблицею recipes, який вказує, до якого рецепту належить інгредієнт.
- food_id (uuid) – зв’язок з таблицею foods, який вказує, який продукт є інгредієнтом.
- created, modified (timestamp) – дати створення та останньої модифікації запису.

Реляції між сутностями:

- users ↔ groceries: Один користувач може мати кілька записів у таблиці groceries, що представляють різні продукти, які він хоче придбати. Це зв’язок один-до-багатьох.
- users ↔ meals: Користувач може мати кілька прийомів їжі, записаних у системі, що зберігаються в таблиці meals. Це також зв’язок один-до-багатьох.
- foods ↔ groceries: Один продукт може входити до декількох списків покупок різних користувачів. Це зв’язок багато-до-багатьох, реалізований через поле food_id у таблиці groceries.
- recipes ↔ recipe_ingredients ↔ foods: Кожен рецепт може містити кілька інгредієнтів, представлених продуктами з таблиці foods. Цей зв’язок реалізовано через таблицю recipe_ingredients, яка містить посилання на рецепт та відповідні продукти.

Ці сутності та їхні зв’язки дозволяють організувати дані в системі для забезпечення зручної взаємодії користувачів із додатком, включаючи створення прийомів їжі, збереження рецептів, керування списками покупок та багато іншого.

2.5 Функції додатку

2.5.1 Користувацькі функції додатку

У системі Hroom користувачі можуть виконувати різні дії, пов’язані з управлінням своїм харчуванням, рецептам, продуктами та списками покупок. Нижче наведені основні дії користувача та їх детальний опис у вигляді активності:

Реєстрація нового користувача: Користувач заповнює форму реєстрації, вводячи адресу електронної пошти та пароль. Після цього йому надсилається лист для підтвердження реєстрації.

Вхід у систему: Після успішної реєстрації користувач може увійти в систему, вводячи свої дані для авторизації. При цьому генерується токен аутентифікації, який використовується для доступу до захищених ресурсів.

Відновлення паролю: Якщо користувач забув свій пароль, він може надіслати запит на відновлення паролю. Система надсилає лист з посиланням для встановлення нового паролю.

Додавання продукту до списку покупок: Користувач може переглянути список доступних продуктів і додати вибраний продукт до свого списку покупок. Після цього він може вказати кількість продукту, яку планує придбати.

Редагування списку покупок: Користувач може редагувати вже існуючий список покупок, змінюючи кількість продуктів або видаляючи їх зі списку.

Перегляд списку покупок: Система надає можливість переглядати поточний список покупок, який користувач може використовувати для організації покупок.

Додавання прийому їжі: Користувач може створити новий прийом їжі, завантажуючи фотографію їжі та додаючи опис. Система аналізує завантажене зображення за допомогою нейронної мережі, розпізнає продукти та розраховує калорійність.

Редагування прийому їжі: Після створення прийому їжі користувач може редагувати його, додаючи додаткову інформацію або змінюючи продукти, що входять до складу прийому їжі.

Перегляд історії прийомів їжі: Користувач може переглядати всі свої прийоми їжі за різні періоди часу, включаючи деталі про калорійність і склад продуктів.

Додавання нового рецепту: Користувач може створювати нові рецепти, додаючи інформацію про інгредієнти, інструкції з приготування та зображення готової страви.

Редагування рецептів: Якщо користувач хоче змінити вже створений рецепт, він може редагувати його, оновлюючи список інгредієнтів, інструкції або зображення.

Перегляд рецептів: Користувач може переглядати доступні рецепти, шукати їх за назвою або за інгредієнтами, щоб знайти підходящі страви.

Зміна паролю: Користувач може змінити свій поточний пароль, використовуючи форму зміни паролю в налаштуваннях профілю.

Аналіз споживаних калорій: Користувач може переглядати графіки, що показують калорійність його прийомів їжі за різні періоди часу, використовуючи компонент MealsStats. Це дозволяє користувачам аналізувати свої харчові звички та приймати обґрунтовані рішення щодо харчування.

Пошук продуктів та рецептів: Користувач може здійснювати пошук продуктів і рецептів у системі за ключовими словами, що допомагає швидко знаходити потрібні дані.

Ці активності показують, що користувачі можуть використовувати систему Hroom для управління своїм харчуванням, ведення записів про спожиту їжу, складання списків покупок та створення рецептів, що полегшує організацію харчування та допомагає стежити за калорійністю.

2.5.2 Адміністративні функції додатку

Адміністратор системи Hroom має доступ до спеціального інтерфейсу адміністрування, створеного за допомогою SQLAdmin. В цьому інтерфейсі адміністратор може виконувати наступні дії:

Перегляд і керування користувачами: Адміністратор може переглядати список користувачів системи, редагувати їхні дані, активувати або деактивувати облікові записи. Це корисно для підтримки безпеки системи та управління доступом.

Управління продуктами: Можливість перегляду, додавання, редагування або видалення записів про продукти в базі даних. Адміністратор може додавати нові продукти або оновлювати інформацію про наявні, щоб забезпечити актуальність бази даних продуктів.

Керування рецептами: Адміністратор має можливість додавати нові рецепти, редагувати наявні та видаляти їх, якщо вони більше не актуальні або не відповідають політиці системи.

Управління прийомами їжі користувачів: Адміністратор може переглядати всі прийоми їжі, створені користувачами, і в разі потреби редагувати або видаляти їх.

Моніторинг бази даних: SQLAdmin надає можливість адміністратору моніторити активність у базі даних, переглядати метадані таблиць, а також виконувати операції з резервного копіювання для захисту даних системи.

Підсумовуючи всі можливі функції можна їх побачити на діаграмі прецендентів системи (Рис.2.11)

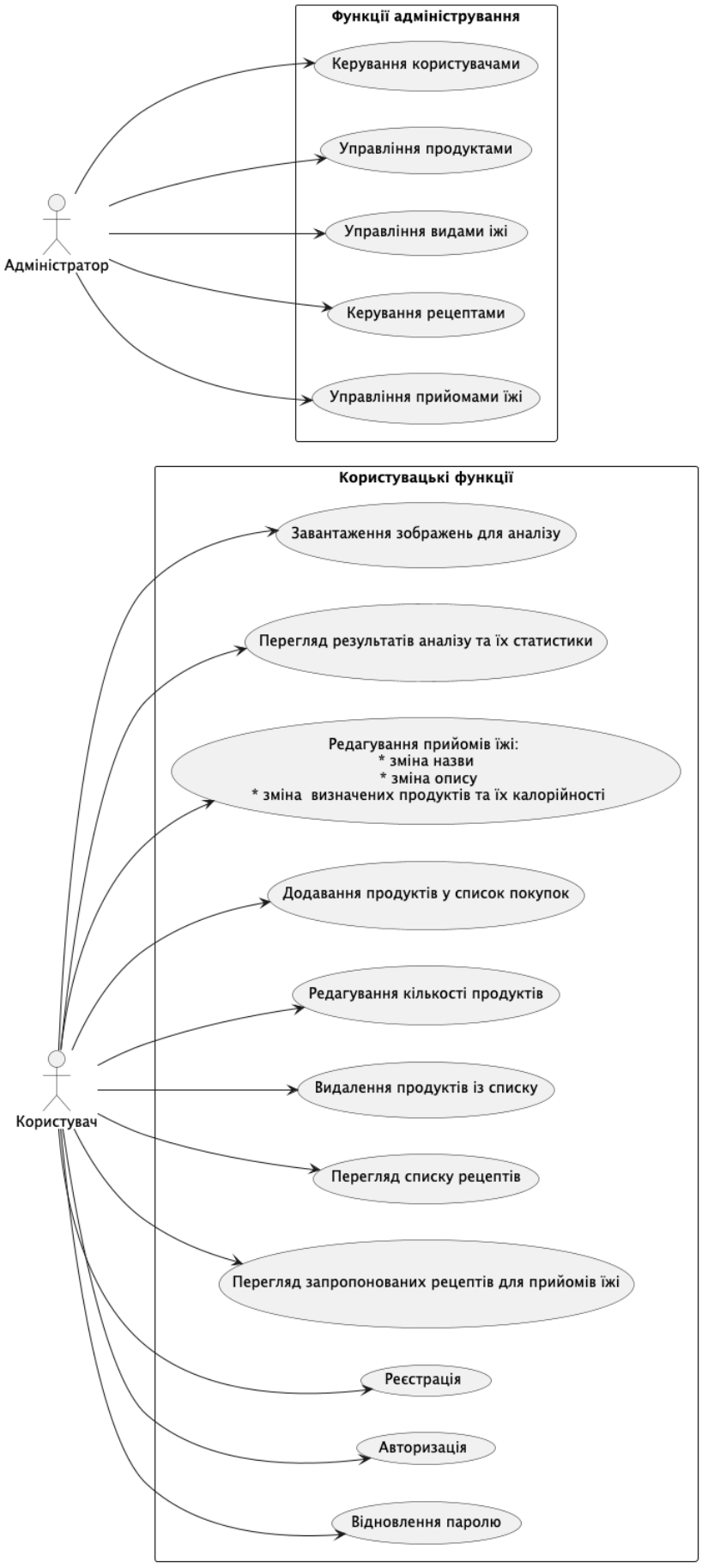


Рис.2.11. Діаграма прецендентів

2.6 Опис API додатку

Далі наведений детальний опис API:

Таблиця 2.1

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Auth:Jwt.Login	POST /auth/jwt/login	Надання облікових даних для входу	Токен доступу JWT	Аутентифікація користувача за допомогою JWT
Auth:Jwt.Logout	POST /auth/jwt/logout	Вихід з поточного сеансу	Підтвердження успішного виходу	Логаут користувача шляхом анулювання токена
Register:Register	POST /auth/register	Реєстрація нового користувача	Підтвердження створення облікового запису	Створення нового облікового запису користувача
Verify:Request-Token	POST /auth/request-verify-token	Запит на отримання токена верифікації	Токен верифікації	Генерація токена для верифікації користувача

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Verify:Verify	POST /auth/verify	Надання токена верифікації	Підтвердження верифікації	Перевірка токена для підтвердження облікового запису
Reset:Forgot Password	POST /auth/forgot-password	Запит на відновлення пароля	Посилання для відновлення пароля	Генерація посилання для відновлення пароля
Reset:Reset Password	POST /auth/reset-password	Надання нового пароля	Підтвердження зміни пароля	Встановлення нового пароля користувача
Users:Current User	GET /users/me	Запит на отримання даних поточного користувача	Інформація про поточного користувача	Отримання профілю поточного користувача
Users:Patch Current User	PATCH /users/me	Оновлення даних поточного користувача	Оновлений профіль користувача	Зміна профілю поточного користувача

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Users:User	GET /users/{id}	Запит на отримання даних конкретного користувача	Інформація про користувача	Отримання профілю іншого користувача за ID
Users:Patch User	PATCH /users/{id}	Оновлення даних конкретного користувача	Оновлений профіль користувача	Зміна профілю іншого користувача за ID
Users:Delete User	DELETE /users/{id}	Видалення облікового запису користувача	Підтвердження видалення	Видалення облікового запису користувача за ID
Recipes:List Recipes	GET /recipes/	Отримання списку рецептів	Список рецептів	Перегляд усіх доступних рецептів
Recipes:Create Recipe	POST /recipes/	Додавання нового рецепту	Інформація про створений рецепт	Створення нового рецепту

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Recipes:Read Recipe	GET /recipes/{recipe_id}	Запит на отримання даних конкретного рецепту	Інформація про рецепт	Отримання інформації про рецепт за ID
Recipes:Update Recipe	PUT /recipes/{recipe_id}	Оновлення даних рецепту	Оновлений рецепт	Зміна даних рецепту за ID
Recipes:Delete Recipe	DELETE /recipes/{recipe_id}	Видалення рецепту	Підтвердження видалення	Видалення рецепту за ID
Foods:List Foods	GET /foods/	Отримання списку продуктів	Список продуктів	Перегляд усіх доступних продуктів
Recipes:Create Food	POST /foods/	Додавання нового продукту	Інформація про створений продукт	Створення нового продукту

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Recipes:Read Food	GET /foods/{food_id}	Запит на отримання даних конкретного продукту	Інформація про продукт	Отримання інформації про продукт за ID
Recipes:Update Food	PUT /foods/{food_id}	Оновлення даних продукту	Оновлений продукт	Зміна даних продукту за ID
Recipes:Delete Food	DELETE /foods/{food_id}	Видалення продукту	Підтвердження видалення	Видалення продукту за ID
Groceries:List Groceries	GET /groceries/	Отримання списку покупок	Список покупок	Перегляд усіх доступних покупок
Groceries:Create Grocery	POST /groceries/	Додавання нової покупки	Інформація про створену покупку	Створення нового запису про покупку

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Groceries:Read Grocery	GET /groceries/{grocery_id}	Запит на отримання даних конкретної покупки	Інформація про покупку	Отримання інформації про покупку за ID
Groceries:Update Grocery	PUT /groceries/{grocery_id}	Оновлення даних покупки	Оновлена покупка	Зміна даних покупки за ID
Groceries:Delete Grocery	DELETE /groceries/{grocery_id}	Видалення покупки	Підтвердження видалення	Видалення покупки за ID
Meals:Predict Meal	POST /meals/predict	Надання даних для передбачення калорій	Результат передбачення	Передбачення калорійності їжі
Meals:Get Meal Result Image	GET /meals/{meal_id}/result-image	Отримання зображення результату аналізу	Зображення результату	Отримання візуального результату передбачення

Таблиця 2.1 (продовження)

Опис API додатку

Назва операції	**Шлях та HTTP метод**	**Короткий опис запиту**	**Короткий опис відповіді**	**Опис операції**
Meals:Read Meal	GET /meals/{meal_id}	Запит на отримання даних конкретної страви	Інформація про страву	Отримання даних про страву за ID
Meals:Update Meal	PUT /meals/{meal_id}	Оновлення даних страви	Оновлена страва	Зміна даних страви за ID
Meals:Delete Meal	DELETE /meals/{meal_id}	Видалення страви	Підтвердження видалення	Видалення страви за ID
Meals:List Meals	GET /meals/	Отримання списку страв	Список страв	Перегляд усіх доступних страв
Meals:Create Meal	POST /meals/	Додавання нової страви	Інформація про створену страву	Створення нового запису про страву

Таким чином було описано всі операції API додатку.

ВИСНОВКИ ДО РОЗДІЛУ 2

У розділі було детально описано підхід до розробки системи розпізнавання їжі та визначення її калорійності, з урахуванням використаних методологій і технологічного стеку. Розробка системи базується на інтеграції сучасних підходів до машинного навчання, таких як згорткові нейронні мережі (CNN) та алгоритм YOLO, які забезпечують високу точність і швидкість роботи в реальному часі. Алгоритм YOLO, зі своєю здатністю розпізнавати кілька об'єктів на одному зображенні, став ключовим елементом системи.

Методологія розробки також включає використання Intersection Over Union (IoU) як метрики для оцінки точності розпізнавання. Цей підхід дозволяє системі навчатися точному визначенню меж об'єктів, що є важливим для роботи з реальними даними. Завдяки використанню IoU система демонструє здатність до ідентифікації продуктів харчування навіть у складних умовах, таких як багатокомпонентні страви або фонові перешкоди.

У процесі проєктування системи було зроблено вибір на користь клієнт-серверної архітектури, що дозволяє забезпечити модульність і гнучкість у розробці. Тонкий клієнт, реалізований на базі React, відповідає за відображення даних та інтерактивність інтерфейсу, тоді як серверна частина, розроблена за допомогою FastAPI, виконує основні обчислення, обробку даних та взаємодію з базою даних. База даних PostgreSQL використовується для зберігання інформації про користувачів, продукти, рецепти та прийоми їжі, що забезпечує надійність і масштабованість системи.

Особливу увагу приділено безпеці системи. Використання JWT-токенів для авторизації, політики CORS для захисту REST API, а також збереження конфіденційних даних у змінних середовища гарантують захист інформації користувачів. Висока гнучкість інструментів, таких як SQLAlchemy для роботи з базою даних і Docker для контейнеризації, дозволяє легко адаптувати систему до змінних потреб.

Використання технологій, таких як React, FastAPI, OpenCV, а також бібліотек для управління станом та побудови графіків, сприяє створенню зручного, інтуїтивного та ефективного інтерфейсу для кінцевих користувачів. У поєднанні з аналітичними функціями, такими як візуалізація даних по калорійності, система Hroom пропонує унікальний функціонал, що задовольняє актуальні потреби ринку.

Таким чином, розроблена методологія і вибір технологій забезпечують основу для створення ефективного та масштабованого продукту, що спрощує управління харчуванням, розпізнавання їжі та визначення її калорійності.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ ЇЖІ ТА ВИЗНАЧЕННЯ КАЛОРІЙНОСТІ

У рамках цього проєкту було реалізовано систему оцінки калорійності їжі на основі методів глибокого навчання та комп'ютерного зору. Система спроєктована для роботи в локальних умовах, через веб-інтерфейс або у хмарному середовищі. Такий підхід забезпечує гнучкість у використанні та дозволяє адаптувати систему до різних потреб і технічних умов кінцевого користувача.

Для локального використання користувачеві необхідно мати пристрій із мінімальними технічними характеристиками: щонайменше 4 ГБ оперативної пам'яті та базову підтримку графічного процесора. У випадку використання у хмарі більша частина обчислень виконується на сервері, що мінімізує вимоги до апаратного забезпечення клієнта та забезпечує швидкість і зручність.

Запропонована система складається з наступних ключових етапів:

1. Отримання зображень як вхідних даних. Користувач завантажує зображення їжі через веб-інтерфейс або шляхом фотографування за допомогою мобільного пристрою. Зображення передається для подальшої обробки.
2. Виявлення об'єктів та їх ідентифікація. Використовуючи алгоритм YOLO, система виявляє об'єкти на зображенні, класифікує їх за категоріями та створює обмежувальні рамки для кожного продукту. Після цього здійснюється обрізання зайвих частин зображення.
3. Сегментація зображень. Система виконує сегментацію зображення для точного визначення меж кожного об'єкта, що покращує якість розпізнавання та підготовлює дані до подальшого аналізу.
4. Розрахунок об'єму, густини та калорійності. Якщо на зображенні присутній еталонний об'єкт (наприклад, стандартний посуд або інший відомий за розмірами предмет), система визначає об'єм кожного продукту.

На основі об’єму та густини, закладених у базу даних, розраховується калорійність продуктів. Якщо еталонного об’єкта немає, система лише визначає тип їжі та надає основну інформацію.

5. Відображення результатів. На екрані користувача виводиться інформація про назву виявлених продуктів та їх калорійність. Результати представлені у зручній формі з можливістю зберегти їх у власному профілі для подальшого аналізу.

Реалізація цієї системи забезпечує інтерактивний і простий спосіб відстеження харчових звичок, що робить її корисною як для звичайних користувачів, так і для спеціалістів у сфері дієтології. Подальші вдосконалення можуть включати інтеграцію з іншими сервісами для формування рекомендацій щодо харчування та управління запасами продуктів.

3.1 Керівництво по використанню системи

Для досягнення найкращих результатів при користуванні системою Hroom важливо дотримуватися кількох рекомендацій. Зображення мають бути чіткими, з хорошим освітленням, без надмірної яскравості чи темряви. Харчові об’єкти краще розташовувати на однорідному фоні, наприклад, на білому або нейтральному кольорі, використовуючи плоску тарілку для покращення точності розпізнавання.

На поточному етапі система налаштована на роботу з певними продуктами, такими як фрукти, і використовує стандартні еталонні об’єкти для визначення розмірів. Користувачі можуть завантажувати фотографії через веб-додаток, після чого система аналізує зображення, розпізнає об’єкти, визначає їх калорійність і відображає результати. У разі потреби користувач може внести корективи у назви чи кількість розпізнаних продуктів.

Щоб покращити результати, варто використовувати якісні камери та стежити за правильним розташуванням об’єктів у кадрі. У майбутньому система

буде розширена для роботи з більшою кількістю продуктів і новими функціями, що підвищать її зручність і ефективність.

Далі представлена блок-схема системи на Рисунку 3.1.

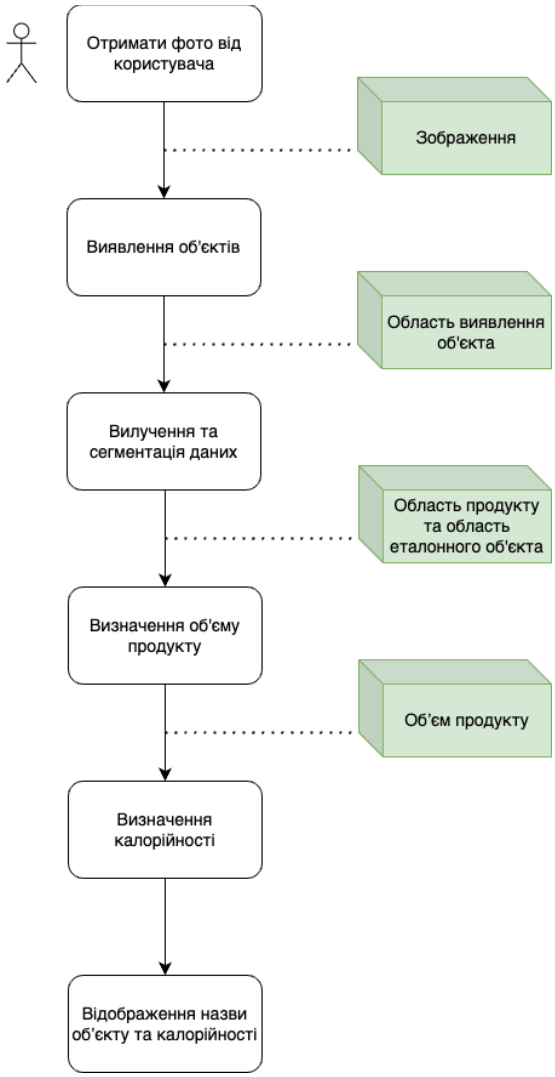


Рис.3.1. Блок-схема системи

3.2 Деталізація етапів системи

3.2.1 Отримання зображень

Робота системи починається з отримання зображення, яке користувач завантажує через веб-додаток. Це може бути фото, зроблене заздалегідь або створене безпосередньо на пристрої користувача (наприклад як на Рис.3.2.). Важливо, щоб зображення відповідало встановленим вимогам, наприклад,

чіткість та рівномірне освітлення, для забезпечення високої точності розпізнавання.

Рис.3.2. Приклад формату зображення

3.2.2 Виявлення об’єктів

Етап виявлення об’єктів у системі є одним із ключових, адже саме він забезпечує точність подальшої роботи з даними. Для цього використовується алгоритм YOLOv4 який є одним із найефективніших рішень у галузі комп’ютерного зору. Його перевага полягає в тому, що весь процес виявлення виконується за один прохід, що суттєво скорочує час обробки. Це особливо важливо для забезпечення роботи системи в реальному часі.

Під час навчання моделі було використано спеціалізовані набори даних, створені за допомогою платформи RoboFlow. Цей інструмент дозволяє легко додавати анотації до зображень, необхідних для роботи YOLO. Набір даних складався з приблизно 800 зображень семи різних категорій фруктів та еталонного об’єкта – великого пальця.

Для покращення якості моделі було застосовано аугментацію даних, що включала:

- Горизонтальні та вертикальні повороти зображень для збільшення їх варіативності.
- Зміни яскравості від -20% до +20%, що імітує різні умови освітлення.
- Повороти на 90° за годинниковою і проти годинникової стрілки та перевертання догори дном.
- Масштабування зображень до розміру 896x896, що відповідає стандартам навчання YOLO.

Система навчена розпізнавати кілька класів об’єктів, серед яких: яблуко, апельсин, банан, морква, помідор, ківі, цибуля, а також великий палець як еталонний об’єкт. Великий палець відіграє важливу роль у системі, адже він використовується для визначення масштабу зображення та розрахунку об’єму інших об’єктів.

3.2.3 Обрізання зображень

Після ідентифікації об’єкти на зображенні сегментуються, а відповідні області обрізаються, щоб виділити кожен об’єкт окремо (Рис.3.3.). Для цього використовуються алгоритми пошуку контурів і фільтрації. Об’єкти, включаючи еталонний об’єкт, зберігаються у вигляді окремих зображень для подальшого аналізу.

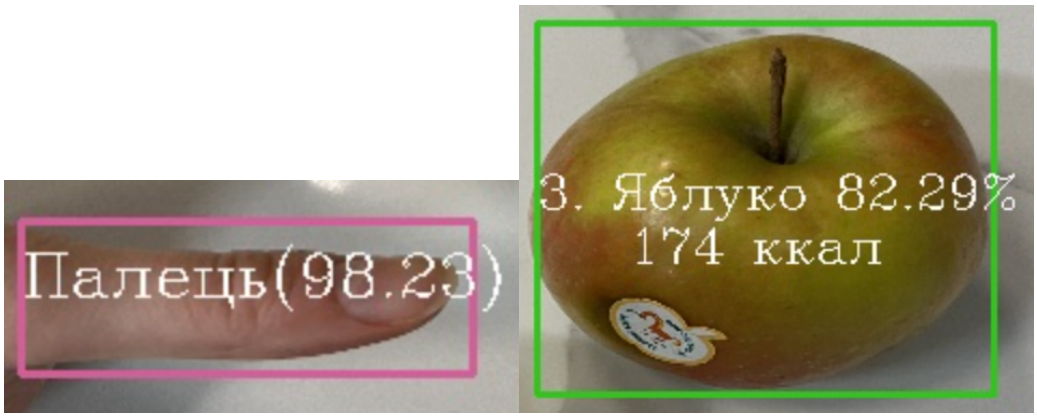


Рис.3.3. Приклад обрізаних зображень

3.2.4 Сегментація

На цьому етапі виконується додаткова обробка зображень, включаючи переведення їх у градації сірого, застосування порогових значень, виділення контурів та створення масок (Рис.3.4). Це дозволяє точно визначити контури об’єктів і підготувати дані для обчислення об’єму.

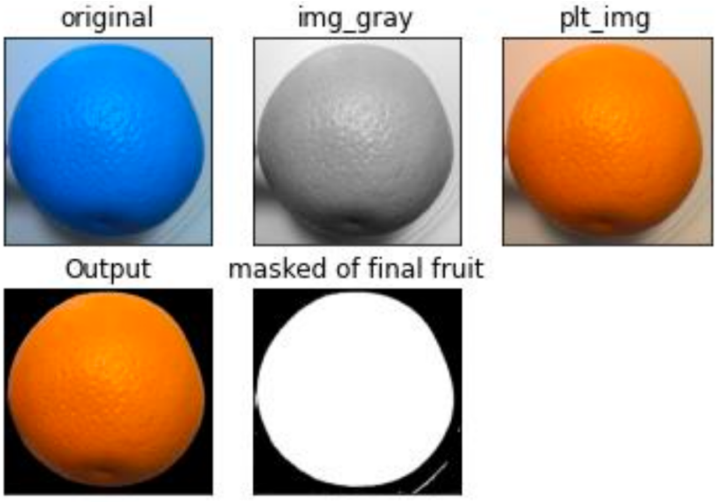


Рис.3.4. Поетапна сегментація апельсину

3.2.5 Визначення об’єму

На цьому етапі використовується інформація, отримана під час попередніх кроків: сегментація зображення, визначення площі об’єкта та його співвідношення до еталонного об’єкта (великого пальця). Основна мета – лоцінити об’єм продукту, використовуючи його форму, що дозволить у майбутньому точно розрахувати вагу та калорійність.

Для розрахунку об’єму застосовуються три ключові фактори:

1. Площа пікселів продукту – загальна кількість пікселів, що належать до контуру продукту після сегментації.
2. Площа пікселів еталонного об’єкта – кількість пікселів, яка визначає контур великого пальця.

3. Реальна площа еталонного об’єкта – фіксована величина, яка є відомою заздалегідь і використовується як мультиплікатор для визначення розмірів продукту.

На основі цих даних розраховується орієнтовна площа продукту за формулою:

$$\frac{\text{Площа пікселів продукту} \times \text{Реальна площа еталонного об’єкта}}{\text{Площа пікселів еталонного об’єкта}}$$

Після отримання площі продукту оцінюється його об’єм. Для цього використовуються математичні моделі, що враховують форму об’єкта:

1. Сферична форма (наприклад яблука, апельсини, помідори, цибуля):

   Обчислюється орієнтовний радіус:

$$ER = \sqrt{\frac{\text{Орієнтовна площа продукту}}{\pi}}$$

   Далі розраховується об’єм:

$$EV = \frac{4}{3} \times \pi \times ER^3$$

2. Циліндрична форма (банани, огірки, морква):

$$ER = \frac{\text{Ширина продукту}}{2}$$

$$EV = \pi \times ER^2 \times h$$

   де h – це довжина продукту

3.2.6 Визначення калорійності

Фінальним кроком є визначення калорійності. На основі об’єму, розрахованої щільності продукту та його калорійності на грам, система обчислює загальну калорійність їжі. Для цього використовується заздалегідь створена таблиця значень (Таблиця 3.1), що містить інформацію про щільність і калорійність кожного типу продукту. Результати, включаючи ідентифіковану їжу, її об’єм, вагу та калорійність, відображаються користувачу у зручному форматі.

Таблиця 3.1

Значення для визначення калорійності продукту

Продукт	Щільність (г\см^3)	Калорій (ккал\г)	Мітка	Форма
Яблуко	0.609	0.52	1	Сфера
Банан	0.94	0.89	2	Циліндр
Морква	0.641	0.41	3	Циліндр
Огірок	0.641	0.16	4	Циліндр
Цибуля	0.513	0.4	5	Сфера
Апельсин	0.482	0.47	6	Сфера
Помідор	0.481	0.18	7	Сфера

Калорійність продукту розраховується у два етапи.

1. Обчислення ваги продукту:

$$\text{Вага (г)} = \text{Щільність продукту } \left(\frac{\text{г}}{\text{см}^3}\right) \times \text{Об'єм продукту (см}^3\text{)}$$

Щільність продукту визначається на основі його типу та береться з таблиці щільності.

2. Розрахунок калорійності:

$$\text{Калорійність (ккал)} = \text{Вага } \left(\frac{\text{г}}{\text{см}^3}\right) \times \text{Калорійність на } 1\text{г } \left(\frac{\text{ккал}}{\text{г}}\right)$$

Після завершення розрахунків система відображає результат у зрозумілому для користувача вигляді, включаючи інформацію про ідентифікований продукт, його вагу та загальну калорійність (Рис. 3.5.).

Рис.3.5. Результат визначення калорійності продукту

3.3 Опис користувацького інтерфейсу системи

Інтерфейс розробленої системи побудований з акцентом на простоту у використанні, інтуїтивність і візуальну привабливість. Інтерфейс поділяється на дві основні частини: клієнтську, орієнтовану на кінцевих користувачів та адміністративну, яка забезпечує можливості управління даними. Такий підхід дозволяє одночасно задовольнити потреби користувачів і адміністраторів системи.

3.3.1 Авторизація користувачів

Сторінка авторизації забезпечує вхід користувачів до системи через введення електронної пошти та пароля (Рис.3.6.). Візуально вона виконана у мінімалістичному стилі: заголовок "Вхід в Хрум" і два текстові поля для введення даних. Для нових користувачів передбачено посилання для переходу на сторінку реєстрації.

У backend реалізовано шифрування паролів (за допомогою бібліотеки bcrypt), а авторизація здійснюється через автентифікацію з використанням JSON Web Token (JWT). У разі успішного входу користувач перенаправляється на головну сторінку.

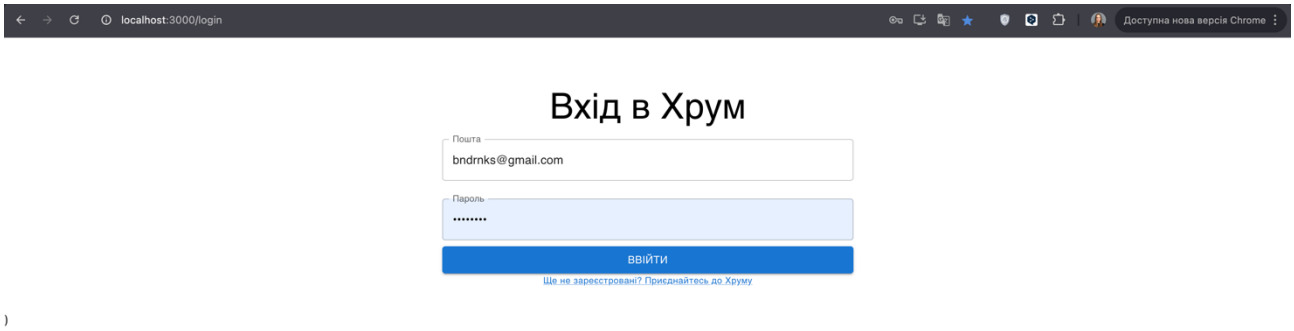


Рис. 3.6. Сторінка авторизації користувачів

3.3.2 Основна клієнтська частина

На головній сторінці користувач може завантажити зображення їжі для її автоматичного аналізу. Ключовим елементом є кнопка "Завантажити продукти", яка запускає процес обробки зображення. Після завантаження система розпізнає продукти, ідентифікує їх, визначає калорійність та кількість, що виводиться у вигляді списку. Також система підсумовує загальну кількість спожитих калорій, що видно на Рис.3.7.

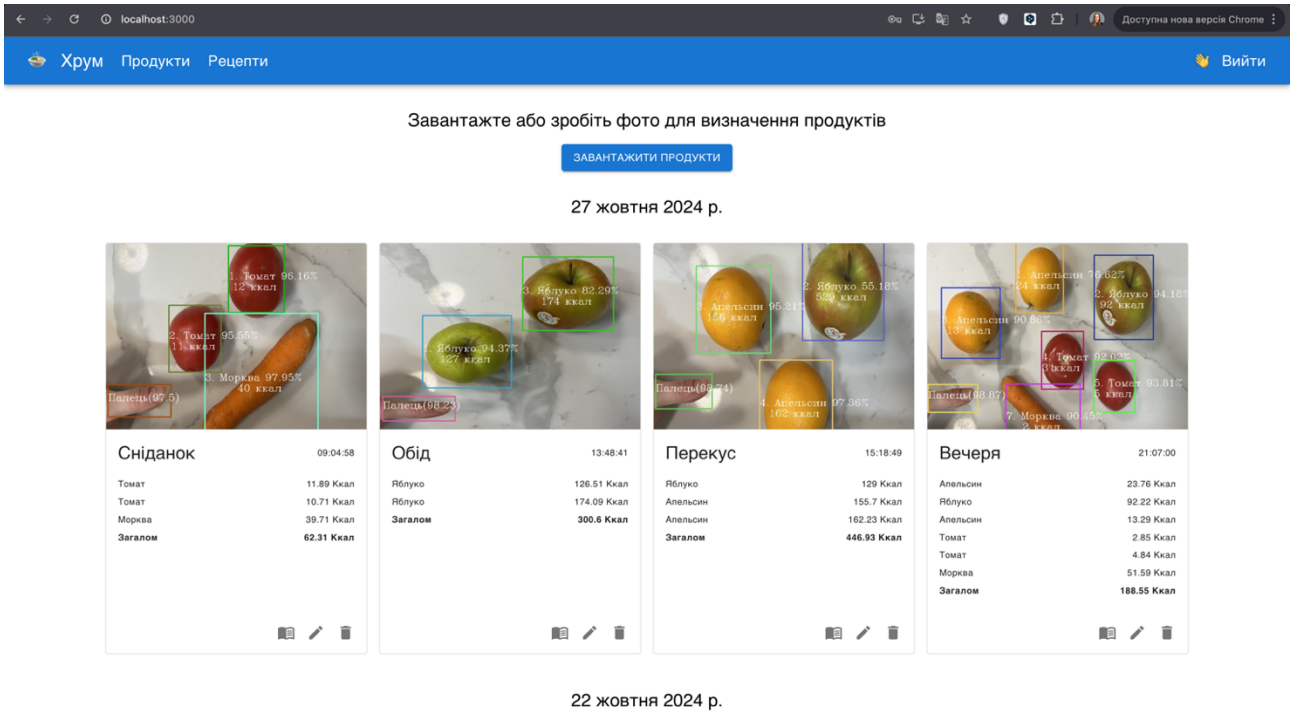


Рис. 3.7. Головна сторінка

Користувач має змогу редагувати прийоми їжі в формі редагування, яка відкривається при натисканні на піктограму редагування, можна змінити назву, опис, сумарну калорійність прийому, а також можна редагувати продукти, які були автоматично визначенні на фото та додавати свої за необхідності. При натисканні кнопки застосувати зміни зберігаються. Також на наступному Рис.3.8. показано адаптивність застосунку, а саме вигляд з мобільного пристрою:

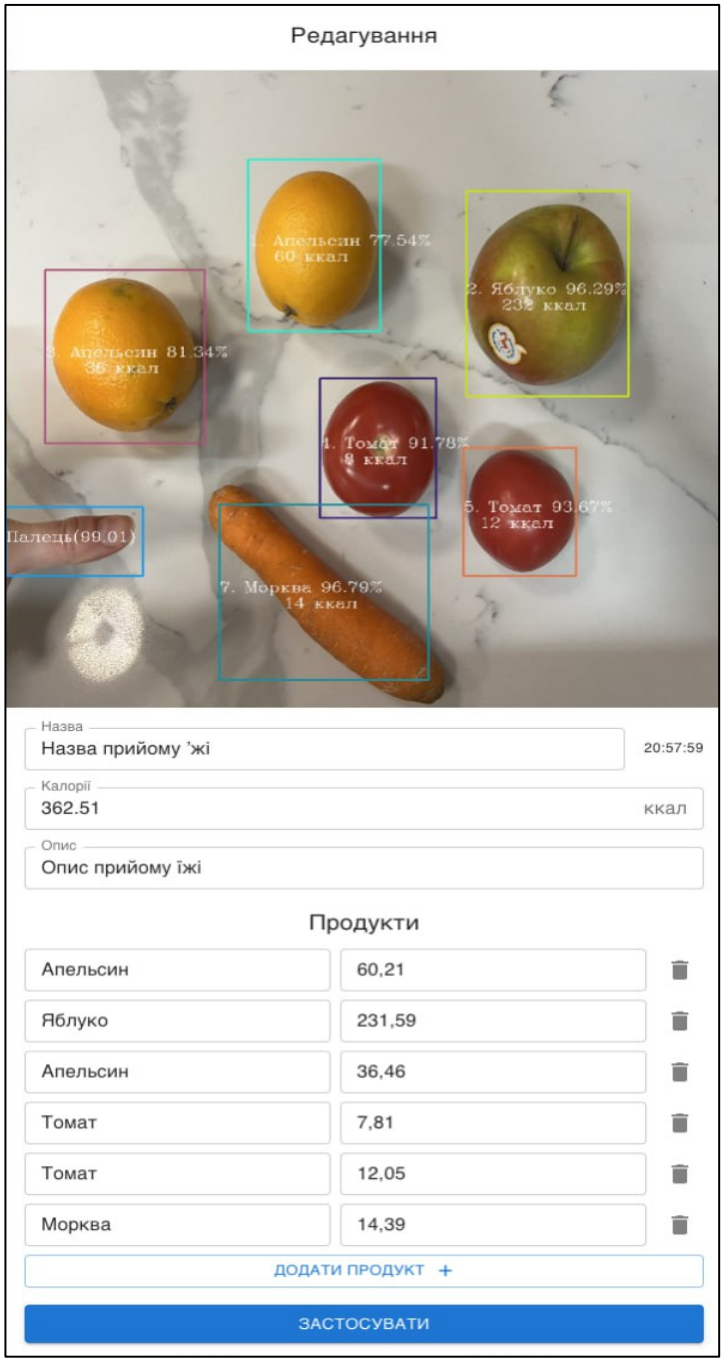


Рис.3.8. Редагування запису та вигляд на мобільному пристрої

Інтерфейс також має додатковий інтерактивний графік для відображення статистики споживання калорій за день, тиждень чи місяць. Це дозволяє користувачам аналізувати свій раціон і приймати обґрунтовані рішення щодо його коригування (Рис.3.9.).

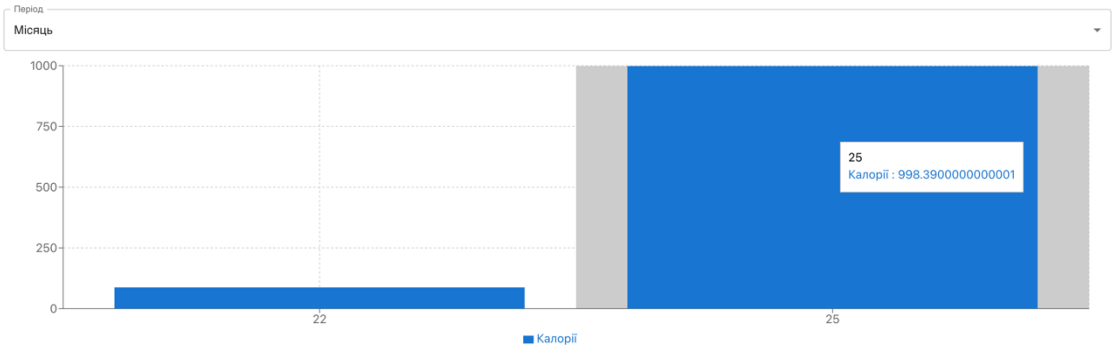


Рис.3.9. Статистика спожитих калорій

Розділ "Продукти" надає можливість керувати запасами продуктів. Цей модуль представлений у вигляді таблиці, де зазначено назви продуктів, їхню поточну кількість та цільові значення (Рис.3.4.). Користувач може редагувати ці дані, додавати нові продукти або видаляти існуючі. Також є інтеграція із сервісами інтернет-магазинів, наприклад Сільпо, що дозволяє зручно формувати кошики для покупок (Рис.3.10.).

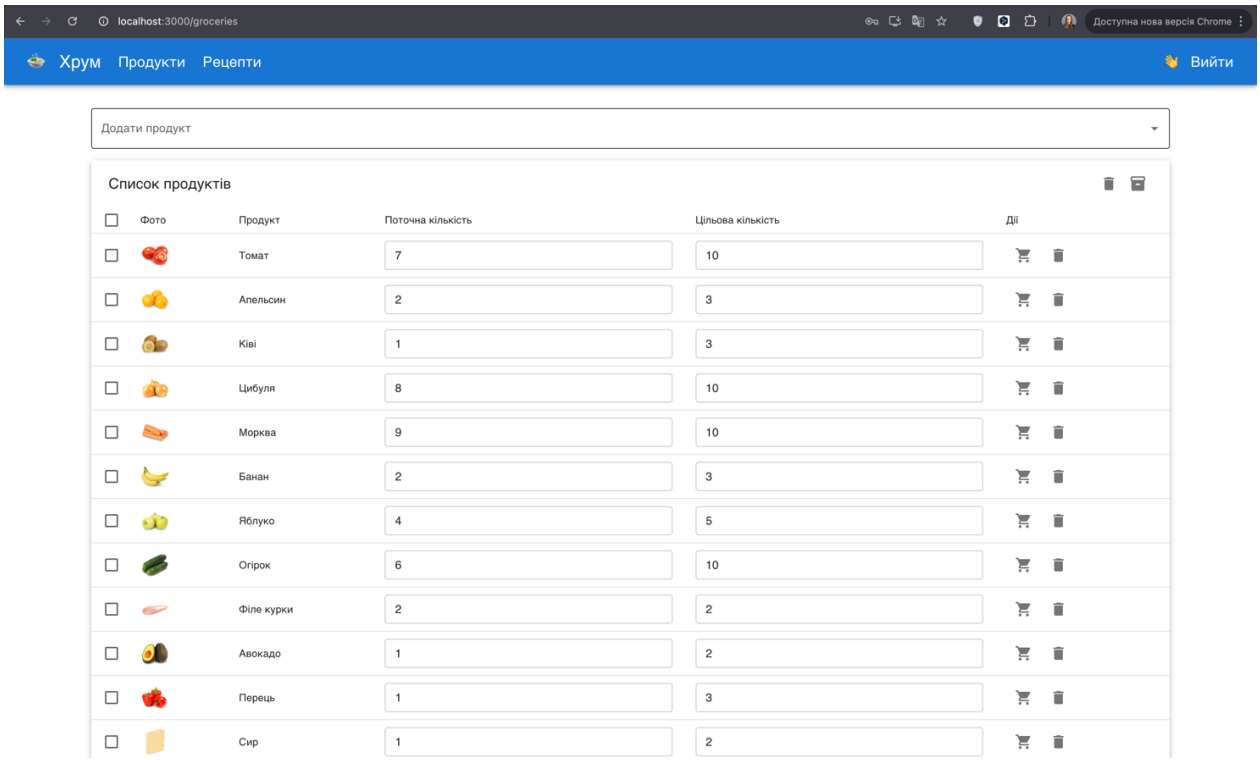


Рис.3.10. Сторінка списку продуктів

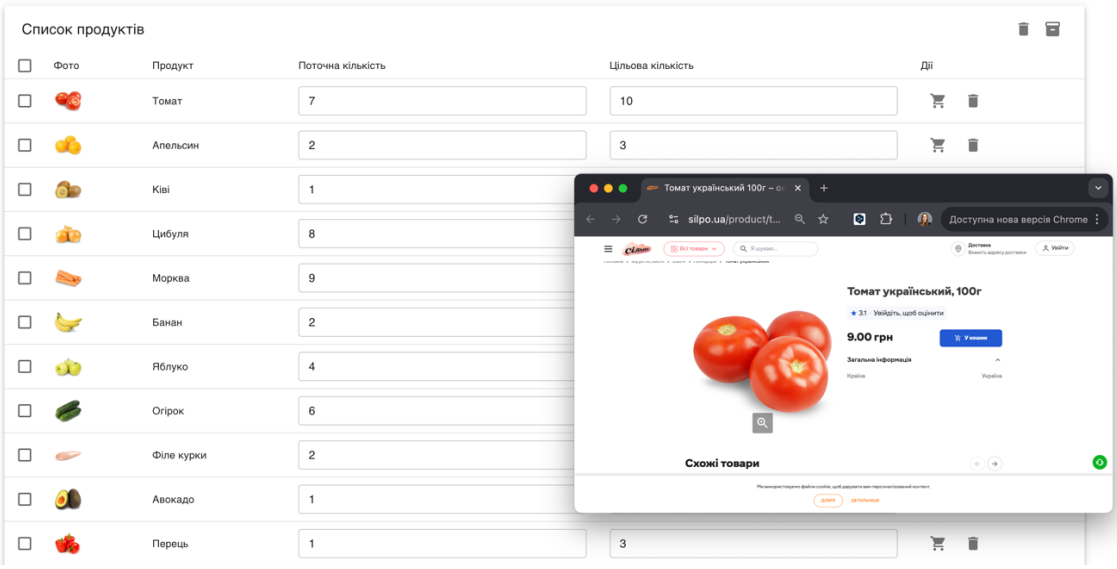


Рис.3.11. Приклад посилання на продукт з корзини в інтернет-магазині

Модуль "Рецепти" пропонує користувачам перелік страв, які можна приготувати з наявних продуктів. У цьому розділі представлені інтерактивні картки рецептів, які містять інформацію про страву, час приготування, кількість порцій, перелік необхідних інгредієнтів (Рис.3.12.).

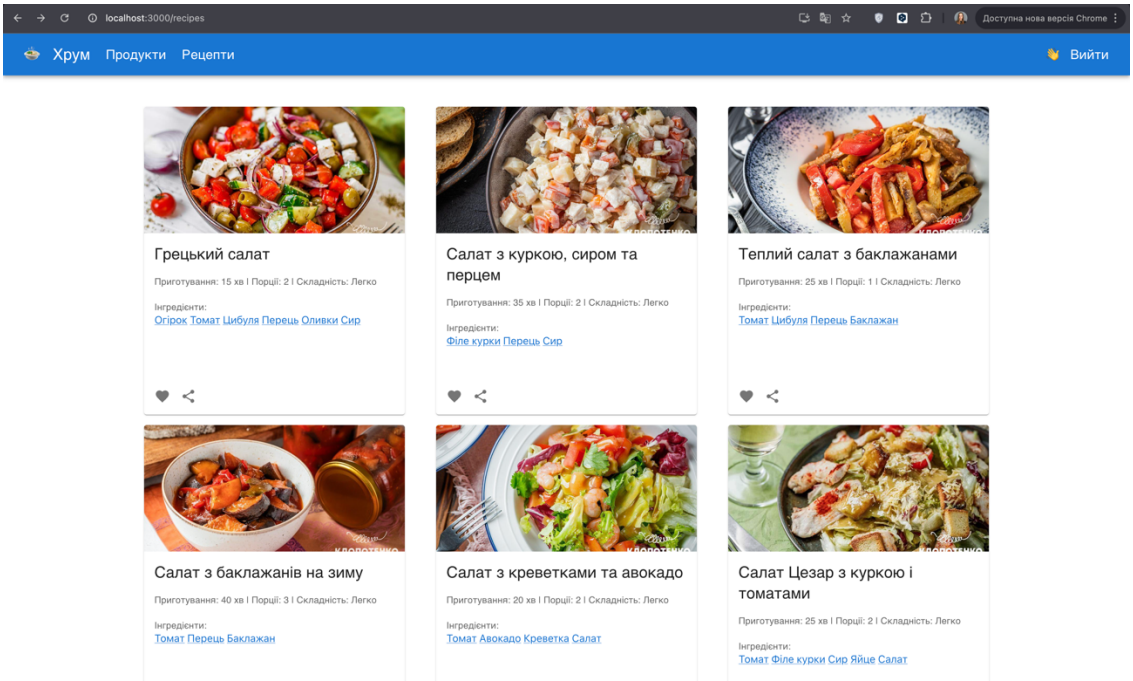


Рис.3.12. Сторінка рецептів

Також по кожному рецепту можна перейти на більш детальний опис, який відкривається в окремій сторінці (Рис.3.13.). Цей функціонал зручний для оптимального використання наявних продуктів і для планування раціону.

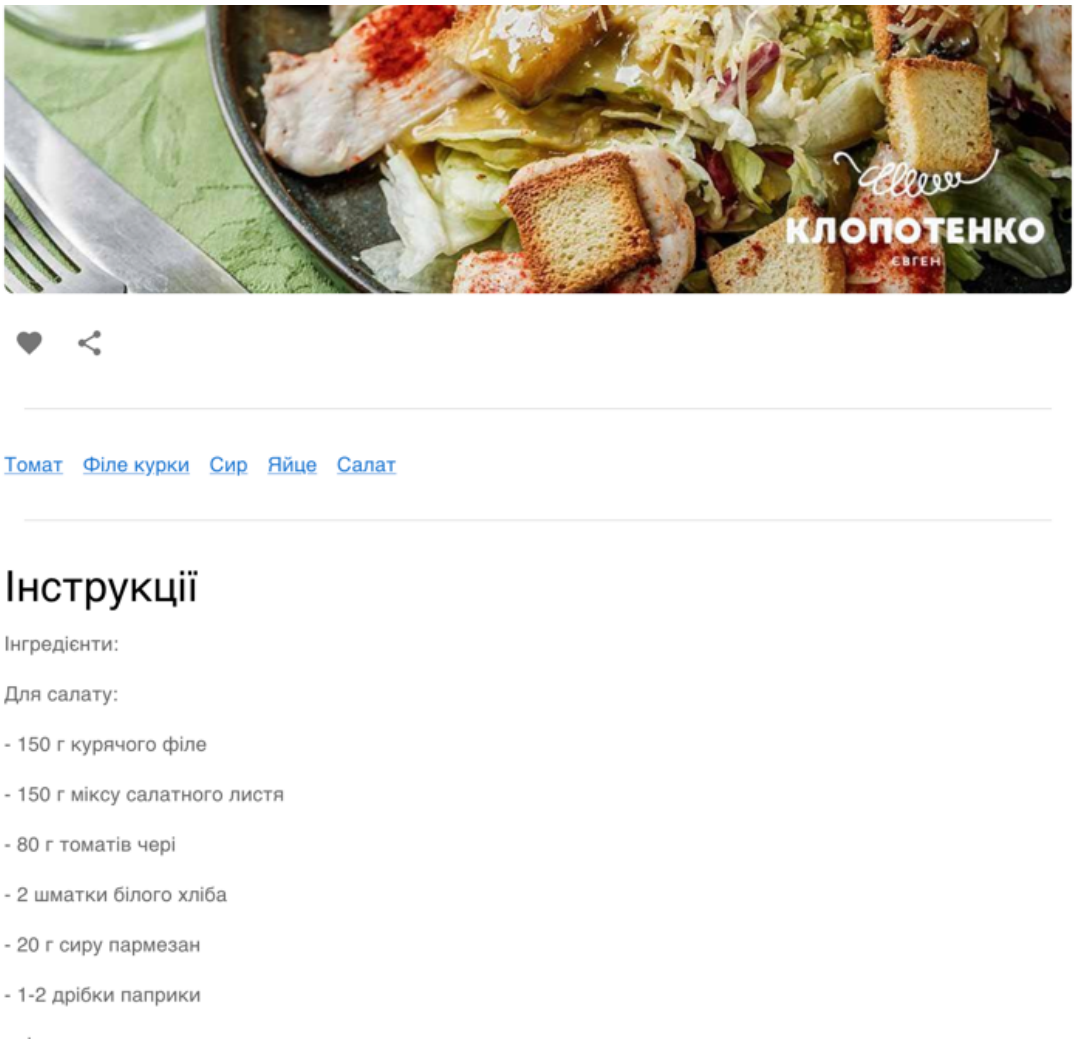


Рис.3.13. Детальна інформація рецепту

3.3.3 Адміністративна панель

Адміністративний модуль надає інструменти для управління системою, зокрема продуктами, рецептами та користувачами. В даному модулі також реалізована авторизація та керування розділами як в клієнтській частині.

Розділ "Foods" дозволяє переглядати та редагувати інформацію про всі продукти, включно з їхніми назвами, зображеннями та посиланнями на сторінки

в інтернет-магазинах. Це забезпечує актуальність і релевантність даних (Рис.3.14).

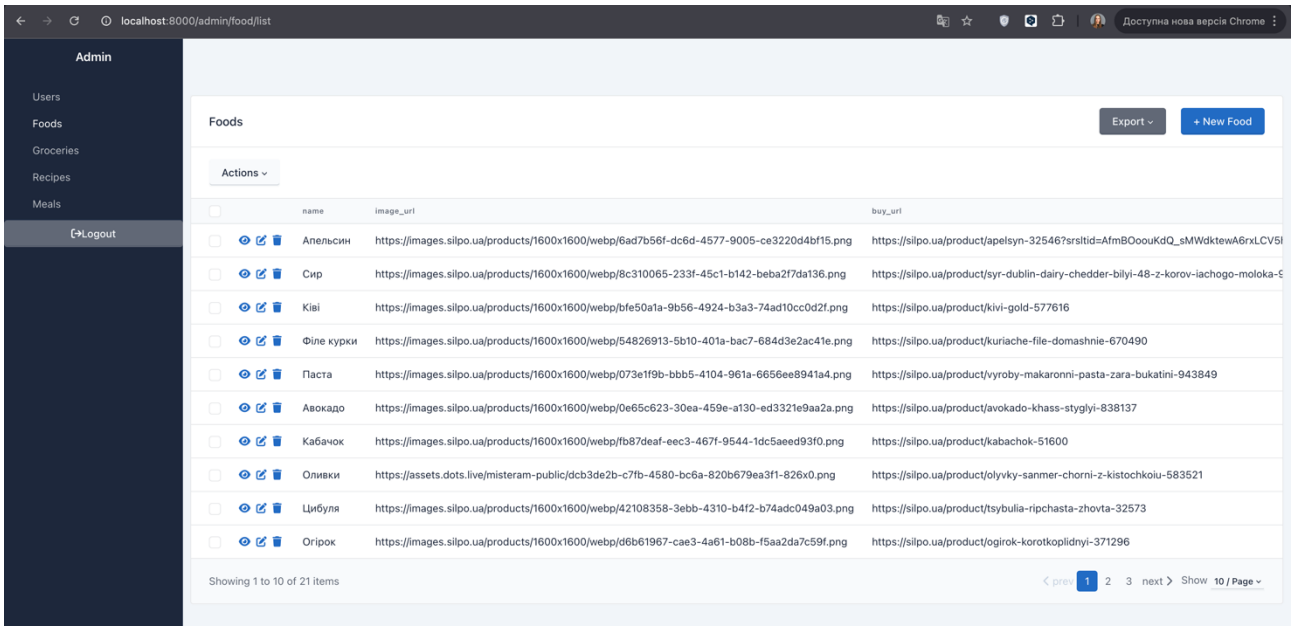


Рис.3.14. Сторінка керування продуктами в системі та посилань на них в інтернет-магазинах

Розділ "Recipes" дозволяє адміністраторам створювати та редагувати рецепти, додаючи деталі про склад інгредієнтів, час приготування, порції, опис та детальні інструкції. Як видно на Рис.3.15., ця функція забезпечує гнучкість у налаштуванні списку рецептів відповідно до потреб користувачів.

Крім цього, адміністративна панель має модуль аналітики, який дозволяє отримувати статистику щодо активності користувачів і популярності продуктів або рецептів. Ця інформація допомагає своєчасно реагувати на зміну потреб користувачів і покращувати функціональність системи.

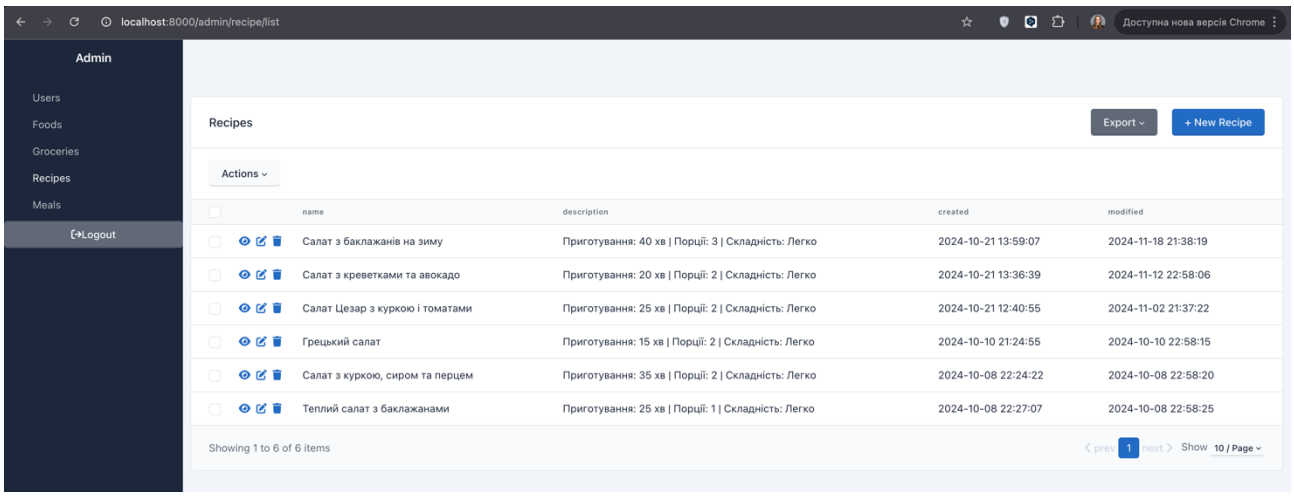


Рис.3.15. Сторінка логування та редагування рецептів

Система підтримує інтеграцію із зовнішніми сервісами, наприклад, із платформами інтернет-магазинів, що дозволяє автоматично формувати списки для покупок. Також реалізована можливість додавання нових функцій завдяки модульній архітектурі, це робить систему легко адаптованою до змін та потреб.

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі представлено реалізацію системи, яка автоматично розпізнає їжу та визначає її калорійність, поєднуючи сучасні методи комп'ютерного зору та глибокого навчання. Головними етапами роботи системи є завантаження зображень, ідентифікація об'єктів на них за допомогою алгоритму YOLOv4, сегментація продуктів, розрахунок їх об'єму з використанням еталонного об'єкта та фінальне обчислення калорійності на основі бази даних. Кожен із цих етапів був ретельно спроєктований і адаптований для забезпечення точності та зручності використання.

Особливістю системи є можливість точного визначення об'єму продуктів завдяки масштабуванню з використанням еталонного об'єкта, такого як великий палець. Це дозволяє проводити коректні розрахунки, навіть якщо користувачі надають зображення з різною перспективою чи розміром. Також варто зазначити, що система підтримує роботу з різними формами об'єктів, такими як сферичні та циліндричні, що значно розширює її можливості.

Розроблена система вже демонструє високу ефективність у розпізнаванні простих продуктів, таких як фрукти та овочі. Проте її архітектура дозволяє масштабувати функціонал, додаючи нові категорії продуктів, складні страви та інтеграцію з іншими платформами. Це відкриває перспективи для подальшого вдосконалення системи, що робить її корисним інструментом як для звичайних користувачів, так і для дієтологів чи фітнес-тренерів. Такий підхід сприяє формуванню зручного й сучасного сервісу для контролю харчування.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЄКТУ

Стартап-проєкт – це інноваційна ініціатива, яка спрямована на розробку і впровадження нового продукту, послуги чи технології з метою задоволення потреб ринку чи створення нових можливостей. Головна особливість стартапу полягає у використанні сучасних ідей та рішень, які мають потенціал для швидкого зростання і масштабування.

Стартап-проєкти відіграють важливу роль у розвитку економіки, адже вони сприяють впровадженню інновацій, створенню робочих місць та стимулюванню конкуренції. Основна мета стартапу – перевірити та реалізувати нову ідею, забезпечити її фінансову життєздатність і отримати ринкове визнання.

Цілі стартап-проєкту включають розробку унікального продукту, що вирішує конкретні проблеми споживачів, тестування його попиту на ринку, формування бізнес-моделі для монетизації та залучення інвестицій для подальшого розвитку. Завдяки цьому стартап може стати основою для довгострокового бізнесу або інтегруватися у вже існуючі галузі змінюючи правила гри.

4.1 Опис ідеї проєкту

Далі було проаналізовано і подано у вигляді таблиць:

- зміст ідеї, що пропонується;
- можливі напрямки застосування;
- основні вигоди, що може отримати користувач товару (за кожним напрямком застосування);
- відмінність від існуючих аналогів та замінників.

Перші три пункти подані у вигляді таблиці 4.1 і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 4.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Хрум – це розумна платформа для аналізу харчування, яка дозволяє автоматично розпізнавати страви за зображеннями, аналізувати їхній поживний склад і калорійність. Проєкт інтегрує функції відстеження запасів продуктів, створення списків покупок із прив'язкою до локальних магазинів та надання рецептів на основі наявних інгредієнтів. Основна мета платформи – допомогти користувачам дотримуватися збалансованого харчування та спростити контроль за раціоном.	Контроль калорійності та складу харчування для людей, які прагнуть підтримувати здоровий спосіб життя.	Економія часу завдяки автоматизації підрахунку калорій і ведення харчового обліку.
	Розробка індивідуальних планів харчування дієтологами та тренерами.	Полегшення контролю за здоровим харчуванням через персоналізовані рецепти.
	Оптимізація процесу купівлі продуктів та управління домашніми запасами їжі.	Мінімізація харчових відходів завдяки обліку продуктів і складанню списків покупок.

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;

- визначення попереднього кола конкурентів (проєктів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, та проводиться збір інформації щодо значень техніко-економічних показників для ідеї власного проєкту та проєктів-конкурентів відповідно до визначеного вище переліку;
- проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні) (таблиця 4.2).

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Конкурент 1 (Calorie Mama)	Конкурент 2 (Snap Calorie)	Конкурент 3 (MacrosAI)			
1	Автоматичне розпізнавання їжі	Так	Так	Так	Так		+	
2	Інтеграція з локальними магазинами	Так	Ні	Ні	Ні			+

Таблиця 4.2 (продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Конкурент 1 (Calorie Mama)	Конкурент 2 (Snap Calorie)	Конкурент 3 (MacrosAI)			
3	Облік запасів продуктів	Так	Ні	Ні	Так		+	
4	Підрахунок калорій	Так	Так	Так	Так		+	
5	Рекомендації рецептів	Так	Ні	Ні	Так			+

4.2 Технологічний аудит ідеї проєкту

Було проведено аудит технології, за допомогою якої можна реалізувати ідею проєкту (технології створення товару). Визначення технологічної здійсненності ідеї проєкту передбачає аналіз таких складових (Таблиця 4.3):

- за якою технологією буде виготовлено товар згідно ідеї проєкту;
- чи існують такі технології, чи їх потрібно розробити/доробити;
- чи доступні такі технології авторам проєкту.

За результатами аналізу таблиці зроблено висновок щодо можливості технологічної реалізації проєкту.

Таблиця 4.3

Технологічна здійсненність проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка інтерфейсу користувача	Створення клієнтської частини додатку із сучасним адаптивним дизайном для відображення основних функцій і даних.	React.js, Material-UI, Recharts	Так
2	Серверна частина з API	Реалізація бекенд-логіки із використанням асинхронних сервісів для взаємодії з базою даних і зовнішніми запитами.	FastAPI, SQLAlchemy, PostgreSQL	Так
3	Обробка зображень	Впровадження механізмів аналізу та сегментації зображень для розпізнавання їжі та підрахунку калорійності.	OpenCV, NumPy, YOLOv4	Так
4	Авторизація користувачів	Забезпечення безпеки даних користувачів через механізм токенів для аутентифікації і авторизації в системі.	JWT, FastAPI Users	Так
5	Управління базою даних	Зберігання структурованих і взаємопов’язаних даних для забезпечення їх швидкого доступу та безпечного адміністрування.	PostgreSQL, SQLAdmin	Так

Таблиця 4.3 (продовження)

Технологічна здійсненність проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
7	Інтеграція з API сторонніх сервісів	Забезпечення можливості отримання або відправки даних до сторонніх платформ через HTTP запити.	REST API, Axios	Так
8	Системи версійності та автоматизації	Управління версіями коду та автоматизація процесів тестування й розгортання системи в різних середовищах.	GitHub	Так
9	Візуалізація статистики	Побудова діаграм та графіків для наочного представлення даних про харчування та калорійність.	Recharts	Так
10	Контейнеризація для розробки	Створення ізольованих середовищ для всіх компонентів системи з можливістю їх одночасного запуску.	Docker, Docker Compose	Так
11	Кешування даних	Оптимізація взаємодії з сервером через збереження даних локально для зменшення затримок і повторних запитів.	React Query	Так

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проєкту, та ринкових загроз, які можуть перешкодити реалізації проєкту, дозволяє спланувати напрями розвитку проєкту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проєктів-конкурентів.

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (Таблиця 4.4).

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проєкту

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців	Близько 3-4 основних гравців (платформи для обліку харчування та аналізу калорійності).
2	Загальний обсяг продаж (у $/рік)	Ринок оцінюється в 2.5-3 млрд $ щорічно з постійним зростанням.
3	Динаміка ринку	Ринок активно розвивається, збільшення попиту через популяризацію здорового харчування та фітнесу.
4	Наявність обмежень для входу	Високі витрати на технології (розпізнавання зображень, нейронні мережі), необхідність у значному первинному капіталі.
5	Специфічні вимоги до стандартизації та сертифікації	Вимоги відсутні, але сертифікація обробки персональних даних може бути корисною для підвищення довіри споживачів.
6	Середня норма рентабельності в галузі (або по ринку)	25-35%, що свідчить про можливість отримання прибутку при якісному впровадженні продукту.

Висновок: враховуючи динаміку ринку, можна зробити висновок, що ринок для входження стартап-продукту є привабливим.

Надалі були визначені потенційні групи клієнтів, їх характеристики та зформовано орієнтовний перелік вимог до товару для кожної групи (Таблиця 4.5).

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги до споживачів продукту
1	Автоматичне розпізнавання їжі	Особи, які слідкують за харчуванням	Шукають простий та швидкий спосіб отримання інформації про продукти та їх калорійність	Простий інтерфейс, швидкість роботи, точність аналізу
2	Інтеграція зі списками покупок	Сім'ї, які планують покупки	Потребують засобів для легкого створення та управління списками покупок	Зручна навігація, можливість спільного використання
3	Рекомендації рецептів	Споживачі, які прагнуть здорового харчування і не хочуть вигадувати рецепти	Цінують рекомендації, засновані на їх індивідуальних потребах, таких як калорійність, алергії чи харчові переваги	Персоналізація, безпека даних

Таблиця 4.5 (продовження)

Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги до споживачів продукту
4	Візуалізація даних про харчування	Особи, що цікавляться статистикою	Зацікавлені у графіках та аналізі своїх харчових звичок, що допомагає їм краще зрозуміти свої дієтичні потреби	Гнучкість налаштувань, адаптивність для різних пристроїв

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проєкту, та факторів, що йому перешкоджають (Таблиці 4.6, 4.7).

Таблиця 4.6

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Високий рівень конкуренції	Наявність конкурентів, які вже займають частину ринку та пропонують подібний функціонал	Розробка маркетингової стратегії, акцент на унікальності продукту, створення програм лояльності

Таблиця 4.6 (продовження)

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
2	Технічна підтримка	Недостатність ресурсів для швидкого вирішення технічних проблем	Найняти команду технічної підтримки, забезпечити моніторинг та реагування на проблеми
3	Захист інформації	Можливі витоки даних користувачів або ненадійність системи захисту	Інтеграція сучасних технологій захисту даних, регулярний аудит безпеки
4	Обмеження фінансування	Високі витрати на розробку системи та її подальше обслуговування	Пошук додаткових інвесторів, оптимізація витрат, впровадження партнерських програм
5	Складність інтеграції	Труднощі з інтеграцією API з іншими сервісами або продуктами користувачів	Розробка універсального API з гарною документацією, консультації з користувачами

Таблиця 4.7

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Легкий вихід на ринок	Використання веб-застосунку для швидкого доступу до інформації та налаштування процесів.	Розробка стратегії виходу з акцентом на швидкість і зручність.

Таблиця 4.7 (продовження)

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
2	Розширення клієнтської бази	Орієнтація на нові сегменти споживачів у сфері, де попит на подібні рішення зростає.	Проведення маркетингових кампаній, підвищення обізнаності серед потенційних клієнтів.
3	Покращення функціоналу	Додаткові модулі для глибшого аналізу даних і автоматизації рутинних процесів.	Інвестиції у розробку нових функцій для покращення конкурентоспроможності.
4	Розширення ринку	Можливість роботи на національному та міжнародному ринку завдяки універсальності рішення.	Адаптація продукту під потреби нових ринків та пошук дистриб’юторів.
5	Інтеграція нових технологій	Впровадження штучного інтелекту для покращення аналізу даних і підвищення ефективності роботи системи.	Найм фахівців для інтеграції сучасних технологій у продукт.

Надалі було проведено аналіз пропозиції: визначено загальні риси конкуренції на ринку (Таблиця 4.8).

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції: монополістична	Ринок заповнений аналогами, що конкурують за функціонал та зручність використання.	Створення унікальних функцій та зручного інтерфейсу, орієнтованого на користувача.
За рівнем конкурентної боротьби: світовий	Існуючі аналоги розробляються скрізь у світі.	Одразу організувати вихід на міжнародну арену та запровадити стратегії відповідно до критеріїв світового ринку.
За галузевою ознакою: міжгалузева	Конкуренція охоплює різні галузі, які пропонують схожі послуги.	Позиціювання продукту як найбільш універсального та зручного рішення.
Конкуренція за видами товарів: товарно-родова	Конкуренти пропонують альтернативні послуги або продукти.	Покращення якості та впровадження функцій, яких немає в аналогів.
За характером конкурентних переваг: цінова	Значна частина конкурентів використовує демпінг цін.	Розробка економічних пакетів послуг, які відповідають потребам різних сегментів.
За інтенсивністю: марочна	Наявність унікальних ознак, що відрізнятиме продукт.	Розробка унікальних дизайнів та торг. марки.

Ступеневий аналіз конкуренції показав результати, що кажуть про те, що аналоги продукту на ринку відзначаються подібним функціоналом та мають аналогічні недоліки, причому їх відмінності виявляються малозначущими. Таким чином, введення нового продукту може спричинити залучення частини аудиторії від конкурентів.

Далі було проведено аналіз конкуренції у галузі за моделлю М. Портера (Таблиця 4.9). За результатами аналізу таблиці 4.9 було зроблено висновок про можливість роботи на ринку з огляду на конкурентну ситуацію. Також було зроблено висновок щодо характеристик, які повинен мати проєкт, щоб бути конкурентоспроможним на ринку. Цей висновок було враховано при формулюванні переліку факторів конкурентоспроможності далі. На основі аналізу конкуренції, наведеного в таблиці 4.9, а також з урахуванням характеристик ідеї проєкту (Таблиця 4.2), вимог споживачів до товару (Таблиця 4.5) та факторів маркетингового середовища (Таблиці 4.6, 4.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлено у таблиці 4.10.

Таблиця 4.9

Аналіз конкуренції в галузі за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Аналогічні системи у сфері харчування, що прогнозують калорії	Невеликі стартапи або індивідуальні розробники	Постачальники даних харчування (API для аналізу продуктів)	Ресторани, користувачі ЗСЖ	Онлайн-калькулятори, мобільні додатки для харчування

Таблиця 4.9 (продовження)

Аналіз конкуренції в галузі за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Висновки	Існує кілька міжнародних конкурентів, але вони не орієнтовані на локальні потреби	Потенційні конкуренти поки не активно присутні на ринку	Залежність від постачальників даних є важливим фактором	Клієнти зацікавлені в автоматизації та точності прогнозів	Додатки та сервіси, які мають подібний функціонал

Таблиця 4.10

Обгрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
1	Унікальність функціоналу	Розроблений продукт надає унікальні функції, які дозволяють автоматизувати процеси оцінки та аналізу, що недоступні в аналогічних рішеннях.
2	Надійність	Продукт забезпечує високу стабільність роботи навіть при пікових навантаженнях, завдяки використанню сучасних серверних рішень.
3	Масштабованість	Архітектура проєкту дозволяє легке додавання нових функцій та підтримку значної кількості користувачів без зниження продуктивності.

Таблиця 4.10 (продовження)

Обгрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
4	Гнучкість у налаштуванні	Система легко адаптується під потреби різних сегментів ринку завдяки модульному дизайну.
5	Інтеграція з іншими сервісами	Забезпечено можливість інтеграції з іншими популярними платформами, що підвищує зручність та розширює потенційний ринок.

За визначеними факторами конкурентоспроможності (Таблиця 4.10) проведено аналіз сильних та слабких сторін стартап-проєкту (Таблиця 4.11).

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін проєкту

№	Фактор конкурентоспроможності	Бали (1-20)	Рейтинг товарів-конкурентів у порівнянні з проєктом						
			-3	-2	-1	0	+1	+2	+3
1	Унікальність функціоналу	17						+	
2	Кроссплатформеність	15					+		
3	Надійність та масштабованість	16				+			

Фінальним етапом ринкового аналізу можливостей впровадження проєкту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (Таблиця 4.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (Таблиця

4.11). Перелік ринкових загроз та ринкових можливостей було складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 4.12

SWOT-аналіз стартап-проєкту

Сильні сторони	Слабкі сторони
1. Унікальність функціоналу. 2. Висока адаптивність і кроссплатформеність. 3. Сучасний та інтуїтивний інтерфейс. 4. Потенціал для інтеграції з іншими системами.	1. Високий рівень конкуренції. 2. Потреба у додаткових інвестиціях. 3. Недостатність ресурсів для масштабування. 4. Наявність технічних викликів на старті.
Можливості	**Загрози**
1. Залучення інвестицій. 2. Вихід на міжнародний ринок. 3. Розширення функціоналу на основі відгуків. 4. Партнерство з лідерами ринку.	1. Поява нового конкурента з кращим функціоналом. 2. Втрата інтересу клієнтів. 3. Зміна законодавства, яке ускладнює роботу. 4. Ризики кібербезпеки.

На основі SWOT-аналізу було розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проєкту на ринок та

орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проєкти конкурентів, що можуть бути виведені на ринок (див. Таблицю 4.9, аналіз потенційних конкурентів). Визначені альтернативи були проаналізовані з точки зору строків та ймовірності отримання ресурсів (Таблиця 4.13).

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Терміни реалізації
1	Пропозиція базової версії продукту безкоштовно для тестування протягом обмеженого періоду.	Висока, оскільки це дозволяє залучити широку аудиторію та отримати первинний зворотний зв’язок.	1-2 місяці
2	Залучення інвесторів через пітч-презентації та демонстрацію функціональних можливостей.	Середня, залежно від якості презентації та актуальності продукту для ринку.	3-4 місяці
3	Запуск рекламної кампанії з акцентом на унікальні функціональні можливості продукту.	Середня, потребує значних фінансових ресурсів для реалізації.	2-3 місяці
4	Публікація детальних статей про продукт на спеціалізованих платформах.	Висока, залежить від якості контенту та популярності обраних платформ.	2-5 місяців

4.4 Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: було проведено опис цільових груп потенційних споживачів (таблиця 4.14).

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачі в сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу в сегмент
1	Особи, що прагнуть контролювати вагу та вести здоровий спосіб життя	Висока	Високий	Середня	Висока
2	Спортсмени та фітнес-ентузіасти	Висока	Середній	Середня	Середня
3	Люди із медичними показаннями для контролю раціону	Середня	Низький	Низька	Середня
4	Сім’ї, що шукають корисні рецепти за наявними продуктами	Середня	Середній	Низька	Висока
Обрані цільові групи: Особи, що прагнуть контролювати вагу та вести здоровий спосіб життя, Спортсмени та фітнес-ентузіасти.					

За результатами аналізу потенційних груп споживачів було обрано цільові групи, для яких буде запропоновано товар та визначено стратегію охопленя ринку – стратегію диференційованого маркетингу (компанія працює з декількома сегментами).

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (Таблиця 4.15).

Таблиця 4.15

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Реалізація тестової версії з базовим функціоналом для залучення аудиторії	Вибірковий розподіл	Доступність для широкого кола користувачів, інтуїтивний інтерфейс, легкість у використанні	Стратегія диференціації
2	Додавання функцій для персоналізації раціону користувача	Масовий маркетинг	Розширення функціональності, адаптація до індивідуальних потреб, зручність налаштувань	Стратегія зростання
3	Розробка мобільного додатка для доповнення веб-функціоналу	Вихід на нові сегменти	Мобільність, інтеграція з іншими платформами, підтримка всіх пристроїв	Стратегія інновацій

Наступним кроком обрано стратегію конкурентної поведінки (Таблиця 4.16).

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№	Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні, схожі додатки вже існують, але вони мають обмежений функціонал.	Компанія буде орієнтуватись на залучення нових споживачів і частково переманювати існуючих через унікальний досвід.	Компанія не копіює конкурентів, але враховує зручність використання та базові функції, як-от калькулятор калорій та планування раціону.	Стратегія диференціації з покращенням споживацького досвіду.

На основі вимог споживачів з обраних сегментів до постачальника (стартап компанії) та до продукту (див. Таблицю 4.5), а також в залежності від обраної базової стратегії розвитку (Таблиця 4.15) та стратегії конкурентної поведінки (Таблиця 4.16) розроблено стратегію позиціонування (Таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проєкт.

Таблиця 4.17

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
1	Точність розрахунків калорійності, зручність інтерфейсу	Стратегія диференціації	Унікальна точність обчислень, легкість використання, автоматизація	Точність, Зручність, Інноваційність
2	Можливість персоналізованих рекомендацій	Наступальна стратегія	Персоналізовані підказки, інтеграція зі смарт-пристроями, адаптивність	Персоналізація, Інтеграція, Зручність
3	Доступність для широкої аудиторії за ціною	Стратегія зниження витрат	Мінімальна ціна, використання доступних технологій, легкий доступ	Економічність, Доступність, Масштабованість

В результаті отримано узгоджену систему рішень щодо ринкової поведінки стартап-компанії.

4.5 Розроблення маркетингової програми стартап-проєкту

Останнім та одним з найважливіших етапів підготовки стартап-проєкту для виходу на ринок є розробка стратегії маркетингу для нього (Таблиця 4.18). Спочатку необхідно сформувати концепцію товару, що отримує споживач. Для цього було зведено результати деяких попередніх аналізів в одну логічну групу і визначено головні відмінності від концепцій конкурентів, що можуть стати вирішальними для вибору потенційного товару серед всіх інших.

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Необхідність швидкого розрахунку калорійності продуктів.	Миттєвий розрахунок калорійності через зручний інтерфейс веб-застосунку.	1. Швидкість 2. Зручність 3. Інтуїтивність використання
2	Автоматичний моніторинг споживання калорій.	Користувач отримує аналітику про динаміку калорійності та рекомендації щодо оптимізації раціону.	1. Аналітичні дані 2. Рекомендації 3. Персоналізація
3	Потреба інтеграції з іншими трекерами здоров’я.	Наявність відкритого API для синхронізації з іншими додатками та пристроями.	1. Інтеграція 2. Гнучкість 3. Підтримка популярних платформ

Таблиця 4.18 (продовження)

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
4	Збереження історії раціону та порівняння результатів.	Доступ до детальної історії споживання калорій та можливість створення порівняльних звітів.	1. Архівування даних 2. Аналітика 3. Візуалізація прогресу

Даний продукт є новим на ринку і слідує стратегічній ідеї забирання клієнтів серед всіх способів отримання споживачів. Це є як недоліком, бо робить складніше весь процес, так і перевагою, бо в контрасті з іншими продуктами на ринку, він робить акцент на покращенні існуючого функціоналу аналогів.

Розроблено трирівневу маркетингову модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (Таблиця 4.19).

1-й рівень: При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і/або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов’язане з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень: Цей рівень являє собою рішення того, як буде реалізовано товар на ринку; включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень: Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості, доставка, умови оплати тощо).

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Веб-застосунок для аналізу харчового раціону та підрахунку калорій. Основна мета – покращення здорового харчування через зручний функціонал аналізу та рекомендацій.
II. Товар у реальному виконанні	1. Автоматизований аналіз калорійності: Розрахунок калорій на основі завантажених зображень та введених даних. 2. Рекомендації: Індивідуальні поради щодо харчування (рецепти). 3. Зручність: Інтуїтивний інтерфейс та кросплатформенність. 4. Швидкість обробки: Швидка реакція системи на дії користувача. 5. Безпека даних: Надійний захист персональної інформації.
III. Товар із підкріпленням	До продажу: Документація, відкритий API для інтеграції. Після продажу: Підтримка користувачів, оновлення системи, можливість персоналізації функціоналу.

Після формування маркетингової моделі товару слід відмітити, що проєкт буде захищено від копіювання шляхом патентування. Наступним кроком є визначення цінових меж (Таблиця 4.20), якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проєкту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Аналіз проведено експертним методом. В

результаті проведення аналізу встановлено верхню та нижню межі встановлення ціни на товар, що надається користувачам.

Таблиця 4.20

Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	0$ - 100$	10$ - 50$	Від 500$ на місяць і більше	5$ - 20$ на місяць

Наступним кроком є визначення оптимальної системи збуту (Таблиця 4.21).

Таблиця 4.21

Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Безкоштовне тестування додатку з подальшою підпискою	Розгортання сервісу, надання підтримки в процесі використання, консультації щодо функціоналу	Одноріневий канал	Розробник – кінцевий споживач через мобільний додаток

Таблиця 4.21 (продовження)

Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
2	Придбання додаткових функцій через підписку	Надання доступу до преміум-функцій, персоналізовані рекомендації, інтеграція з іншими сервісами	Глибока	Онлайн-канал через офіційний вебсайт
3	Інтеграція додатку з іншими системами клієнта	Налаштування API, підтримка інтеграції, надання технічної документації	Вертикальна	B2B-канал для бізнес-користувачів

Останнім етапом було розроблено концепцію маркетингових комунікацій, враховуючи обрану стратегію позиціонування та унікальність клієнтської поведінки. Нова концепція спрямована на інформування та нагадування споживачам про продукт, а також на підтримку його успішного впровадження (Таблиця 4.22).

Таблиця 4.22

Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Користувачі, які слідкують за калорійністю їжі	Соціальні мережі, мобільні додатки, електронна пошта	Легкість у використанні, точність, зручність	Підкреслити переваги додатку у швидкому підрахунку калорій	Пояснити зручність використання додатку для слідкування за раціоном.
2	Фітнес-ентузіасти та тренери	Блоги, відео на YouTube, спортивні форуми	Унікальний функціонал, індивідуальні рекомендації	Мотивувати до використання для досягнення спортивних цілей	Показати, як додаток може допомогти в плануванні харчування для тренувань.
3	Люди з проблемами здоров’я (дієти, алергії)	Медичні платформи, онлайн-спільноти	Безпечність, персоналізація, інтеграція рекомендацій	Залучити користувачів, які потребують специфічного меню	Підкреслити персоналізацію додатку для специфічних медичних потреб.

Таблиця 4.22 (продовження)

Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
4	Загальна аудиторія (ті, хто хоче слідкувати за раціоном)	Соціальні мережі, реклама в пошукових системах	Широка доступність, низька ціна, простота використання	Зацікавити великі групи користувачів	Презентувати як сучасний і доступний інструмент для покращення якості харчування.

Результатом аналізу стала ринкова (маркетингова) програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого впроваджено проєкт, та відповідну обрану альтернативу ринкової поведінки.

ВИСНОВКИ ДО РОЗДІЛУ 4

Розроблений стартап-проєкт орієнтований на вирішення актуальної проблеми оптимізації харчування та контролю калорійності. Враховуючи сучасні тенденції до здорового способу життя, проєкт відповідає потребам широкої аудиторії: від людей які прагнуть зберігати форму, до тих хто має специфічні медичні потреби в харчуванні. Використання інноваційних технологій дозволяє створити продукт, який поєднує зручність, точність та доступність.

Під час роботи над проєктом було проаналізовано конкурентне середовище, визначено основні сильні та слабкі сторони аналогічних продуктів, а також обрано стратегії позиціонування, ціноутворення та збуту. Проєкт пропонує унікальні переваги: інтеграцію з популярними платформами, персоналізовані рекомендації та простоту у використанні. Це дозволить ефективно зайняти конкурентну нішу на ринку.

Запропонована бізнес-модель передбачає використання доступних каналів збуту, орієнтованих як на індивідуальних користувачів, так і на організації. Завдяки цьому проєкт має значний потенціал для масштабування та залучення інвестицій. Окремо варто відзначити перспективи вдосконалення продукту, а саме через постійний зворотний зв’язок із користувачами та впровадження нових функцій.

Таким чином, стартап-проєкт є обґрунтованим як з технологічної, так і з економічної точки зору. Реалізація проєкту сприятиме задоволенню потреб цільової аудиторії, забезпечить стабільне зростання та стане гарним внеском у сферу цифрових рішень для здорового харчування.

ВИСНОВКИ

В процесі виконання магістерської роботи було успішно досягнуто поставленої мети, яка полягала у створенні системи для автоматизованого розпізнавання їжі та оцінки її калорійності. Система відповідає сучасним вимогам інтеграції технологій штучного інтелекту в повсякденне життя, орієнтована на зручність використання, персоналізацію та можливість інтеграції з локальними торговельними платформами.

Під час роботи було проведено ґрунтовний аналіз існуючих рішень у цій сфері, що дозволило виявити ключові недоліки та обмеження, які вдалося врахувати під час розробки власної системи. Обрані сучасні алгоритми машинного навчання, такі як YOLOv5, забезпечили ефективне розпізнавання об’єктів на зображеннях у реальному часі. Спроєктована архітектура веб-додатку забезпечує швидкий доступ до функціоналу, дозволяє масштабування та адаптацію до нових умов. У процесі реалізації було створено систему, що включає функції розпізнавання їжі, визначення калорійності, управління запасами продуктів, автоматичне формування кошика для покупок та рекомендації рецептів. Проведене тестування системи на реальних даних підтвердило її ефективність і точність.

Розроблена система Hroom (або Хрум) має значну практичну цінність. Вона стане корисним інструментом для користувачів, які прагнуть покращити свої харчові звички, ефективно управляти запасами продуктів та отримувати персоналізовані рекомендації щодо харчування. Інтеграція з локальними торговельними мережами робить систему зручною для використання і адаптованою до потреб ринку.

Подальший розвиток цієї системи може включати розширення функціоналу, вдосконалення алгоритмів для роботи з багатокомпонентними стравами, підтримку офлайн-режиму та впровадження додаткових аналітичних інструментів, що сприятиме підвищенню ефективності й популярності продукту. Загалом, виконана робота підтверджує актуальність і ефективність використання

технологій штучного інтелекту для автоматизації харчових процесів. Це рішення має сприяти покращенню якості життя користувачів та підвищенню інтересу до здорового харчування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. E.N. Whitney. Understanding Nutrition / E.N. Whitney, S.R. Rolfes., 2020.
2. Calorie Mama [Електронний ресурс] – Режим доступу до ресурсу: https://www.caloriemama.ai/.
3. CaloPal [Електронний ресурс] – Режим доступу до ресурсу: https://calopal.ai/.
4. Centa [Електронний ресурс] – Режим доступу до ресурсу: https://centa.world/.
5. Snap Calorie [Електронний ресурс] – Режим доступу до ресурсу: https://www.snapcalorie.com/.
6. MacrosAI [Електронний ресурс] – Режим доступу до ресурсу: https://macrosai.net/.
7. Deep Learning Bible - 4. Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://wikidocs.net/204455.
8. World Health Organization [Електронний ресурс] // World Health Statistics. – 2012. – Режим доступу до ресурсу: http://www.who.int/gho/publications/world_health_statistics/2012/en/index.html.
9. P. Pouladzadeh. Measuring Calorie and Nutrition from Food Image / P. Pouladzadeh, S. Shirmohammadi, R. Almaghrabi., 2014.
10. Introducing Hooks – React. [Електронний ресурс] // React – A JavaScript library for building user interfaces – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-intro.html .
11. Smith, Jon. Entity Framework core in action. Simon and Schuster, 2021.
12. Bloch, Joshua. "How to design a good API and why it matters." Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. 2006.

ДОДАТОК А
ПРОГРАМНИЙ КОД

```
// frontend/src/mui.jsx

import React from 'react';
import {Link as RouterLink} from 'react-router-dom';
import {createTheme} from "@mui/material";

const LinkBehavior = React.forwardRef(({href, ...other}, ref) => {
  return <RouterLink ref={ref} to={href} {...other} />;
});

export const theme = createTheme({
  components: {
    MuiLink: {
      defaultProps: {
        component: LinkBehavior,
      },
    },
    MuiButtonBase: {
      defaultProps: {
        LinkComponent: LinkBehavior,
      },
    },
  },
});
// frontend/src/reportWebVitals.js

const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB
}) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;

// frontend/src/hooks.js

import {useAppStore} from "./store";
import {useNavigate} from "react-router-dom";
import {useEffect} from "react";
```

```
export function useAuth() {
  const navigate = useNavigate();
  const token = useAppStore(s => s.token);
  useEffect(() => {
    if (token == null) navigate("/login");
  }, [token, navigate]);
}
// frontend/src/store.js

import {useStore} from 'zustand'
import {createStore} from 'zustand/vanilla'
import {persist} from 'zustand/middleware'

export const appStore = createStore(
  persist(
    (set, get) => ({
      token: null,
      setToken: (token) => set({token}),
    }),
    {
      name: 'hroom-storage',
    },
  ),
)

export const useAppStore = (selector) => useStore(appStore, selector)
// frontend/src/index.js

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-
vitals
reportWebVitals();

// frontend/src/index.css
```

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

// frontend/src/dashboard/MealsStats.jsx

import React, {useState} from "react";
import dayjs from "dayjs";
import {Box, FormControl, InputLabel, MenuItem, Select} from
"@mui/material";
import {Bar, BarChart, CartesianGrid, Legend, ResponsiveContainer,
Tooltip, XAxis, YAxis} from "recharts";

export function MealsStats({meals}) {
  //month | week | day
  const [statsPeriod, setStatsPeriod] = useState("month")
  const now = dayjs();
  const dateCalories = meals.map(m => ({date: dayjs(m.created), calories:
m.total_calories}))
  const filtered = dateCalories.filter(({date}) => date.isSame(now,
statsPeriod))
  const periodCalories = filtered.map(m => {
    let period = null;
    if (statsPeriod === "week") period = m.date.day()
    else if (statsPeriod === "day") period = m.date.format('HH')
    else if (statsPeriod === "month") period = m.date.format('DD')
    return ({...m, period})
  })
  const totalCaloriesPerPeriod = periodCalories.reduce((acc, {period,
calories}) => {
    if (!acc[period]) acc[period] = 0;
    acc[period] += calories;
    return acc;
  }, {});

  const preparedMeals =
Object.entries(totalCaloriesPerPeriod).map(([period, total_calories]) =>
({
    period,
```

```
    total_calories
  }));

  return (
    <Box my={4}>
      <FormControl fullWidth>
        <InputLabel id="period-select-label">Період</InputLabel>
        <Select label="Період" labelId="period-select-label" id="period-
select" variant="outlined"
                value={statsPeriod} onChange={(e) =>
setStatsPeriod(e.target.value)}>
          <MenuItem value="month">Місяць</MenuItem>
          <MenuItem value="week">Тиждень</MenuItem>
          <MenuItem value="day">День</MenuItem>
        </Select>
      </FormControl>

      {meals && (
        <ResponsiveContainer width="100%" height={400}>
          <BarChart data={preparedMeals} margin={{top: 20, right: 30,
left: 20, bottom: 5}}>
            <CartesianGrid strokeDasharray="3 3"/>
            <XAxis dataKey={(meal) => meal.period}/>
            <YAxis/>
            <Tooltip/>
            <Legend/>
            <Bar name="Калорії" dataKey="total_calories" fill="#1976d2"/>
          </BarChart>
        </ResponsiveContainer>
      )}
    </Box>
  )
}
// frontend/src/dashboard/Dashboard.jsx

import React from 'react';
import {useAuth} from "hooks";
import AppContainer from "AppContainer";
import {useMeals} from "api";
import {Meals} from "./Meals";
import {MealUpload} from "./MealUpload";
import {MealsStats} from "./MealsStats";

// Dashboard Component
export default function Dashboard() {
  useAuth()
  const mealsQuery = useMeals();
  const meals = mealsQuery.data

  return (
    <AppContainer>
```

```
      <MealUpload/>
      {meals && <Meals meals={meals}/>}
      {meals && <MealsStats meals={meals}/>}
    </AppContainer>
  );
}



// frontend/src/dashboard/Meals.jsx

import React, {useEffect, useState} from "react";
import axios from "axios";
import {v4 as uuid} from 'uuid';
import {
  Box,
  Button,
  Card,
  CardActions,
  CardContent,
  CardMedia,
  Dialog,
  DialogTitle,
  Grid2,
  IconButton,
  InputAdornment,
  Stack,
  TextField,
  Typography
} from "@mui/material";
import DeleteIcon from '@mui/icons-material/Delete';
import EditIcon from '@mui/icons-material/Edit';
import AddIcon from '@mui/icons-material/Add';
import RecipeIcon from '@mui/icons-material/MenuBook';
import {useDeleteMeal, useUpdateMeal} from "../api";
import {useForm} from "@tanstack/react-form";

export function Meals({meals}) {
  meals = meals.map(m => ({...m, products: m.products.map(p => ({id:
uuid(), ...p}))}))
  const mealsByDate = Object.entries(
    Object.groupBy(
      meals,
      ({created}) => new Date(created).toLocaleDateString('uk-UA', {day:
'2-digit', month: 'long', year: 'numeric'})
    )
  ).sort(([date1], [date2]) => date2.localeCompare(date1));

  return (
    <Stack spacing={2}>
      {mealsByDate.map(([date, dateMeals]) => (
```

```
        <Box key={date}>
          <Typography textAlign="center" variant="h5">{date}</Typography>
          <Grid2 my={4} container spacing={2} justifyContent="center">
            {dateMeals.map((meal) => (
              <Grid2 key={meal.id} md={4}>
                <MealView meal={meal}/>
              </Grid2>
            ))}
          </Grid2>
        </Box>
      ))}
    </Stack>
  )
}

function MealView({meal}) {
  const image = useImage(`/meals/${meal.id}/result-image`)
  const deleteMutation = useDeleteMeal()

  return (
    <Card variant="outlined"
          sx={{marginBottom: 2, width: 350, height: '100%', display:
'flex', flexDirection: 'column'}}>
      <CardMedia
        component="img"
        height="250"
        src={image}
        alt={meal.id + '-result-image'}
      />
      <CardContent sx={{pb: 0}}>
        <Stack spacing={2}>
          <Stack direction="row" justifyContent="space-between"
alignItems="center" spacing={2} mb={1}>
            <Typography variant="h5">{meal.name ?? 'Прийом
їжі'}</Typography>
            <Typography variant="caption">{new
Date(meal.created).toLocaleTimeString()}</Typography>
          </Stack>
          {meal.description && (
            <Typography variant="subtitle2">
              {meal.description}
            </Typography>
          )}
          <Stack component="span" spacing={0.5}>
            {meal.products && meal.products.map((it) =>
              <Stack key={it.id} direction="row" justifyContent="space-
between" alignItems="center" spacing={2}>
                <Typography variant="caption">
                  {it.title}
                </Typography>
                <Typography variant="caption">
```

```
                {it.calories} Ккал
              </Typography>
            </Stack>
          )}
          <Stack direction="row" justifyContent="space-between"
alignItems="center" spacing={2}>
            <Typography variant="caption" fontWeight="bold">
              Загалом
            </Typography>
            <Typography variant="caption" fontWeight="bold">
              {meal.total_calories} Ккал
            </Typography>
          </Stack>
        </Stack>
      </Stack>
    </CardContent>
    <Box flexGrow={1}/>
    <CardActions disableSpacing sx={{justifyContent: "flex-end"}}>
      <IconButton href={`/recipes${meal.products ?
`?filter=${meal.products.map(m => m.title).join(",")}` : ''}`}>
        <RecipeIcon/>
      </IconButton>
      <MealEditPopup meal={meal} image={image}/>
      <IconButton onClick={() => deleteMutation.mutate(meal.id)} aria-
label="delete">
        <DeleteIcon/>
      </IconButton>
    </CardActions>
  </Card>
 )
}


function useImage(url) {
  const [src, setSrc] = useState('');

  useEffect(() => {
    const fetchImage = async () => {
      try {
        const response = await axios.get(url, {responseType: 'blob'});
        setSrc(URL.createObjectURL(response.data));
      } catch (error) {
        console.error(`Error fetching the image ${url}`, error);
      }
    };

    fetchImage();
  }, [url]);

  return src
}
```

```
function MealEditPopup({meal, image}) {
  const updateMutation = useUpdateMeal()
  const [open, setOpen] = useState(false);
  const handleOpen = () => setOpen(true);
  const handleClose = () => setOpen(false);
  const form = useForm({
    defaultValues: meal,
    onSubmit({value}) {
      value.products.forEach((p, i) => {
        if (p.name === undefined) {
          p.name = p.title
        }
        if(p.id === undefined) {
          value.products.splice(i, 1)
        }
      })
      updateMutation.mutate(value);
      handleClose();
    }
  })

  return (
    <div>
      <IconButton type="button" onClick={handleOpen}>
        <EditIcon/>
      </IconButton>
      <Dialog onClose={handleClose} open={open} maxWidth="lg"
scroll="body">
        <DialogTitle textAlign="center">Редагування</DialogTitle>
        <img src={image} alt={meal.id + '-result-image'} width="600"/>
        <form onSubmit={(e) => {
          e.preventDefault()
          e.stopPropagation()
          form.handleSubmit()
        }}>
          <Stack spacing={2} m={2}>
            <Stack direction="row" justifyContent="space-between"
alignItems="center" spacing={2} mb={1}>
              <form.Field
                name="name"
                children={(field) => (
                  <TextField label="Назва" placeholder="Назва"
size="small" fullWidth
                             defaultValue={field.state.value}
onBlur={field.handleBlur}
                             onChange={e =>
field.handleChange(e.target.value)}/>
                )}
              />
```

```
              <Typography variant="caption">{new
Date(meal.created).toLocaleTimeString()}</Typography>
            </Stack>
            <form.Field
              name="total_calories"
              children={(field) => (
                <TextField label="Калорії" placeholder="Калорії"
size="small" fullWidth
                           slotProps={{input: {endAdornment:
<InputAdornment position="end">ккал</InputAdornment>}}}
                           defaultValue={field.state.value}
onBlur={field.handleBlur}
                           onChange={e =>
field.handleChange(e.target.value)}/>
              )}
            />
            <form.Field
              name="description"
              children={(field) => (
                <TextField label="Опис" placeholder="Опис" size="small"
fullWidth multiline
                           defaultValue={field.state.value}
onBlur={field.handleBlur}
                           onChange={e =>
field.handleChange(e.target.value)}/>
              )}
            />

            <Stack spacing={1}>
              <Typography variant="h6"
textAlign="center">Продукти</Typography>
              <form.Field name="products" mode="array">
                {(field) => (
                  <>
                    {field.state.value.filter(v => v.id !==
undefined).map((v, i) => (
                      <Stack key={v.id} direction="row" spacing={1}>
                        <form.Field name={`products[${i}].title`}>
                          {titleField => (
                            <TextField size="small" fullWidth

defaultValue={titleField.state.value} onBlur={titleField.handleBlur}
                                       onChange={e =>
titleField.handleChange(e.target.value)}/>
                          )}
                        </form.Field>
                        <form.Field name={`products[${i}].calories`}>
                          {caloriesField => (
                            <TextField
                              fullWidth
                              type="number"
```

```
                                slotProps={{htmlInput: {step: 0.01}}}
                                size="small"
                                defaultValue={caloriesField.state.value}
onBlur={caloriesField.handleBlur}
                                onChange={(e) =>
caloriesField.handleChange(e.target.value)}/>
                              )}
                            </form.Field>
                            <IconButton onClick={() =>
field.removeValue(i)}><DeleteIcon/></IconButton>
                          </Stack>
                        )
                      )}
                      <Button variant="outlined" size="small"
endIcon={<AddIcon/>}
                              onClick={() => field.pushValue({id: uuid(),
title: '', calories: 0})}>
                        Додати продукт
                      </Button>
                    </>
                  )}
                </form.Field>

              </Stack>

              <form.Subscribe
                selector={(state) => [state.canSubmit, state.isSubmitting]}
                children={([canSubmit, isSubmitting]) => (
                  <Button variant="contained" type="submit"
disabled={!canSubmit}>
                    {isSubmitting ? '...' : 'Застосувати'}
                  </Button>
                )}
              />
            </Stack>
          </form>
        </Dialog>
      </div>
    );
}
// frontend/src/dashboard/MealUpload.jsx

import {useCreateMeal} from "../api";
import React, {createRef} from "react";
import {Button, Stack, Typography} from "@mui/material";

export function MealUpload() {
  const createMeal = useCreateMeal()
  const fileInputRef = createRef();

  return (
```

```
    <Stack my={4} spacing={2} alignItems="center">
      <Typography variant="h5">Завантажте або зробіть фото для визначення
продуктів</Typography>
      <Button variant="contained" color="primary" onClick={() =>
fileInputRef.current?.click()}>
        Завантажити продукти
      </Button>
      <input
        ref={fileInputRef}
        hidden
        accept="image/*"
        onChange={e => {
          e.preventDefault()
          if (e.target.files) {
            const file = e.target.files[0];

            if (!file) {
              alert('Please select a file to upload');
              return;
            }

            createMeal.mutate({file})
          }
          e.target.value = null;
        }}
        type="file"
      />
    </Stack>
  )
}
// frontend/src/Login.jsx

import React, {useState} from 'react';
import axios from 'axios';
import {Button, Container, Link, TextField, Typography} from
'@mui/material';
import {useNavigate} from "react-router-dom";
import {useAppStore} from "./store"; // Import for visualization


// Login Component
export default function Login() {
  const setToken = useAppStore(s => s.setToken)
  const navigate = useNavigate();

  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = async () => {
    try {
      const response = await axios.postForm('/auth/jwt/login', {username:
```

```
email, password});
      setToken(response.data.access_token);
      navigate('/');
    } catch (error) {
      alert('Error logging in');
    }
  };

  return (
    <Container maxWidth="sm" sx={{p: 4, mt: 4, display: 'flex',
flexDirection: 'column', alignItems: 'stretch'}}>
      <Typography variant="h3" textAlign="center">Вхід в
Хрум</Typography>
      <TextField id="email" label="Пошта" type="email" fullWidth
margin="normal" value={email}
                 onChange={(e) => setEmail(e.target.value)}/>
      <TextField id="password" label="Пароль" type="password" fullWidth
margin="normal" value={password}
                 onChange={(e) => setPassword(e.target.value)}/>
      <Button variant="contained" fullWidth color="primary"
onClick={handleLogin}>Ввійти</Button>
      <Link href="/register" textAlign="center" variant="caption">Ще не
зареєстровані? Приєднайтесь до Хруму</Link>
    </Container>
  );
}
// frontend/src/recipes/Recipe.jsx

import {useAuth} from "hooks";
import AppContainer from "AppContainer";
import {useRecipe} from "api";
import {useParams} from "react-router-dom";
import React from "react";
import {Box, Container, Divider, Grid2, IconButton, Link, Stack,
Typography,} from "@mui/material";
import FavoriteIcon from "@mui/icons-material/Favorite";
import ShareIcon from "@mui/icons-material/Share";


export default function Recipe() {
  useAuth()
  const { id} = useParams()
  const recipe = useRecipe(id).data

  if (!recipe) {
    return (
        <AppContainer>
          <Typography variant="h2">Рецепт не знайдено</Typography>
        </AppContainer>
    );
  }
```

```
  return (
      <AppContainer>
        <Container maxWidth="md">
          <Box my={4}>
            <Typography variant="h3" component="h1" gutterBottom>
              {recipe.name}
            </Typography>
            <Typography variant="body1" color="text.secondary"
gutterBottom>
              {recipe.description}
            </Typography>
          </Box>
          <Box>
            <img
                src={recipe.image_url}
                alt={recipe.name}
                style={{ width: "100%", borderRadius: 8 }}
            />
          </Box>
          <Stack direction="row" spacing={1} mt={2} mb={4}>
            <IconButton aria-label="add to favorites">
              <FavoriteIcon />
            </IconButton>
            <IconButton aria-label="share">
              <ShareIcon />
            </IconButton>
          </Stack>
          <Divider variant="middle" />
          <Box my={4}>
            <Grid2 container spacing={2}>
              {recipe.ingredients.map((ingredient) => (
                  <Grid2 item xs={6} sm={4} key={ingredient.id}>
                    <Link href={ingredient.buy_url} target="_blank">
                      <Typography
variant="body1">{ingredient.name}</Typography>
                    </Link>
                  </Grid2>
              ))}
            </Grid2>
          </Box>
          <Divider variant="middle" />
          <Box my={4}>
            <Typography variant="h4" component="h2" gutterBottom>
              Інструкції
            </Typography>
            {recipe.instructions.split("\n").map((step, index) => (
                <Typography
                    key={index}
                    variant="body1"
                    color="text.secondary"
```

```
                    paragraph
                >
                  {step}
                </Typography>
            ))}
          </Box>
        </Container>
      </AppContainer>
  );
}
// frontend/src/recipes/Recipes.jsx

import React from "react";
import {useRecipes} from "api";
import {useAuth} from "hooks";
import AppContainer from "AppContainer";
import {
    Box,
    Card, CardActionArea,
    CardActions,
    CardContent,
    CardMedia,
    Grid2,
    IconButton,
    Link,
    Stack,
    Typography
} from '@mui/material';
import FavoriteIcon from '@mui/icons-material/Favorite';
import ShareIcon from '@mui/icons-material/Share';
import {useSearchParams} from "react-router-dom";

export default function Recipes() {
    useAuth()
    const [searchParams] = useSearchParams();
    const filter = searchParams.get('filter')?.split(',') ?? [];
    const recipes = (useRecipes().data ?? [])
    const filteredRecipes = filter.length === 0 ? recipes :
recipes.filter(recipe => recipe.ingredients.some(i =>
filter.includes(i.name)));
    return (
        <AppContainer>
            <Grid2 container spacing={2} justifyContent="center">
                {filteredRecipes.map((recipe) => (
                    <Grid2 key={recipe.id} xs={12} sm={6} md={4}>
                        <RecipeCard recipe={recipe}/>
                    </Grid2>
                ))}
                {filteredRecipes.length === 0 && (
                    <Grid2 xs={12} sm={6} md={4}>
                        <Typography variant="h2">Рецепти не
```

```
знайдено</Typography>
                    </Grid2>
                )}
            </Grid2>
        </AppContainer>
    )
}

function RecipeCard({recipe}) {
    return (
        <Card sx={{margin: 2, width: "400px", height: "100%"}}>
            <CardActionArea
                href={`/recipes/${recipe.id}`}
                sx={{height: "100%", display: 'flex', flexDirection:
'column', alignItems: "flex-start"}}>
                <CardMedia
                    component="img"
                    height="194"
                    image={recipe.image_url}
                    alt={recipe.name + '-image'}
                />
                <CardContent>
                    <Typography variant="h5" sx={{mb:
2}}>{recipe.name}</Typography>
                    <Typography variant="body2" color="text.secondary">
                        {recipe.description}
                    </Typography>
                    <br/>
                    <Typography variant="body2" color="text.secondary">
                        Інгредієнти:
                    </Typography>
                    <Stack direction="row" component="span"
spacing={0.5}>
                        {recipe.ingredients.map((ingredient) =>
                            <Link key={ingredient.id}
href={ingredient.buy_url} target="_blank">
                                {ingredient.name}
                            </Link>
                        )}
                    </Stack>
                </CardContent>
                <Box flexGrow={1}/>
                <CardActions disableSpacing>
                    <IconButton aria-label="add to favorites">
                        <FavoriteIcon/>
                    </IconButton>
                    <IconButton aria-label="share">
                        <ShareIcon/>
                    </IconButton>
                </CardActions>
            </CardActionArea>
```

```
        </Card>
    );
}
// frontend/src/groceries/Groceries.jsx

import {useAuth} from "hooks";
import React, {useEffect, useRef} from "react";
import DeleteIcon from '@mui/icons-material/Delete';
import BuyIcon from '@mui/icons-material/ShoppingCart';
import RestockIcon from '@mui/icons-material/Inventory';
import {
  Autocomplete,
  Avatar,
  Checkbox,
  IconButton,
  Paper,
  Stack,
  TextField,
  Toolbar,
  Tooltip,
  Typography
} from "@mui/material";
import Table from '@mui/material/Table';
import TableBody from '@mui/material/TableBody';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TableHead from '@mui/material/TableHead';
import TableRow from '@mui/material/TableRow';
import {useCreateGrocery, useDeleteGrocery, useFoods, useGroceries,
useUpdateGrocery} from "../api";
import AppContainer from "AppContainer";


export default function Groceries() {
  useAuth()
  const foods = useFoods().data ?? [];
  const groceries = useGroceries().data ?? [];
  const updateMutation = useUpdateGrocery();
  const createMutation = useCreateGrocery();
  const deleteMutation = useDeleteGrocery();
  const onDelete = (id) => () => deleteMutation.mutate(id)
  const onUpdate = (id, property) => (event) =>
updateMutation.mutate({id, [property]: event.target.value})

  const existingFoods = groceries.map(it => it.food.id)
  const foodOptions = foods
    .filter(it => existingFoods.indexOf(it.id) === -1)
    .map(it => ({label: it.name, id: it.id}))

  const [selected, setSelected] = React.useState([]);
```

```
  const onSelectAllClick = (event) => {
    if (event.target.checked) setSelected(groceries.map((g) => g.id));
    else setSelected([]);
  };

  const onRestockAction = () => {
    const restockGroceries = selected.length === 0 ? groceries :
groceries.filter(g => selected.includes(g.id))
    restockGroceries.forEach(g => updateMutation.mutate({id: g.id,
current_quantity: g.target_quantity}))
  }

  const onDeleteAction = () => {
    if (selected.length === 0) groceries.forEach(g =>
deleteMutation.mutate(g.id))
    else {
      selected.forEach(deleteMutation.mutate)
      setSelected([])
    }
  }

  const onSelect = id => () => {
    const selectedIndex = selected.indexOf(id);
    let newSelected = [];

    if (selectedIndex === -1) {
      newSelected = newSelected.concat(selected, id);
    } else if (selectedIndex === 0) {
      newSelected = newSelected.concat(selected.slice(1));
    } else if (selectedIndex === selected.length - 1) {
      newSelected = newSelected.concat(selected.slice(0, -1));
    } else if (selectedIndex > 0) {
      newSelected = newSelected.concat(
        selected.slice(0, selectedIndex),
        selected.slice(selectedIndex + 1),
      );
    }
    setSelected(newSelected);
  };

  return (
    <AppContainer>
      {/*<Typography variant="h3" textAlign="center">Ваш список
продуктів</Typography>*/}
      <Autocomplete
        disablePortal
        fullWidth
        options={foodOptions}
        value={null}
        blurOnSelect
        clearOnBlur
```

```
        sx={{mb: 2}}
        onChange={(event, value) => {
          if (value) {
            createMutation.mutate({food_id: value.id, current_quantity:
0, target_quantity: 0})
          }
        }}
        renderInput={(params) => <TextField {...params} label="Додати
продукт"/>}
      />

      <TableContainer component={Paper} elevation={4}>
        <Toolbar>
          {selected.length > 0 ? (
            <Typography
              sx={{flex: '1 1 100%'}}
              color="inherit"
              variant="subtitle1"
              component="div"
            >
              Обрано {selected.length} продуктів
            </Typography>
          ) : (
            <Typography
              sx={{flex: '1 1 100%'}}
              variant="h6"
              id="tableTitle"
              component="div"
            >
              Список продуктів
            </Typography>
          )}
          <Tooltip title="Видалити">
            <IconButton onClick={onDeleteAction}>
              <DeleteIcon/>
            </IconButton>
          </Tooltip>
          <Tooltip title="Оновити">
            <IconButton onClick={onRestockAction}>
              <RestockIcon/>
            </IconButton>
          </Tooltip>
        </Toolbar>
        <Table sx={{minWidth: 650}} stickyHeader size="small">
          <TableHead>
            <TableRow>
              <TableCell padding="checkbox">
                <Checkbox
                  color="primary"
                  indeterminate={selected.length > 0 && selected.length <
groceries.length}
```

```
                  checked={groceries.length > 0 && selected.length ===
groceries.length}
                  onChange={onSelectAllClick}
                />
              </TableCell>
              <TableCell>Фото</TableCell>
              <TableCell>Продукт</TableCell>
              <TableCell>Поточна кількість</TableCell>
              <TableCell>Цільова кількість</TableCell>
              <TableCell>Дії</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {groceries.map(it => (
              <TableRow
                key={it.id}
                sx={{'&:last-child td, &:last-child th': {border: 0}}}
              >
                <TableCell padding="checkbox">
                  <Checkbox color="primary"
checked={selected.includes(it.id)} onClick={onSelect(it.id)}/>
                </TableCell>
                <TableCell component="th" scope="row">
                  <Avatar alt="Food Image" src={it.food.image_url}/>
                </TableCell>
                <TableCell>
                  {it.food.name}
                </TableCell>
                <TableCell>
                  <QuantityInput grocery={it} onUpdate={onUpdate}
property="current_quantity"/>
                </TableCell>
                <TableCell>
                  <QuantityInput grocery={it} onUpdate={onUpdate}
property="target_quantity"/>
                </TableCell>
                <TableCell>
                  <Stack direction="row">
                    <IconButton target="_blank" href={it.food.buy_url}
size="large">
                      <BuyIcon/>
                    </IconButton>
                    <IconButton onClick={onDelete(it.id)} size="large">
                      <DeleteIcon/>
                    </IconButton>
                  </Stack>

                </TableCell>
              </TableRow>
            ))}
          </TableBody>
```

```
        </Table>
      </TableContainer>
    </AppContainer>
  )
}

function QuantityInput({grocery, onUpdate, property}) {
  const ref = useRef()
  const propertyValue = grocery[property];
  useEffect(() => {
    ref.current.value = propertyValue
  }, [propertyValue])

  return (
    <TextField onBlur={onUpdate(grocery.id, property)}
defaultValue={propertyValue} type="number" fullWidth size="small"
               inputRef={ref}/>
  )
}
// frontend/src/AppContainer.jsx

import {AppBar, Box, Button, Container, Toolbar, Typography} from
"@mui/material";
import {useAppStore} from "./store";

export default function AppContainer({children}) {
  const setToken = useAppStore(s => s.setToken)
  return (
    <>
      <AppBar position="static">
        <Toolbar>
          <Button color="inherit" href="/" sx={{textTransform: "none",
mr: 1}}>
            <Typography
variant="h5">🥗&amp;nbsp;&amp;nbsp;&amp;nbsp;Хрум</Typography>
          </Button>
          <Button color="inherit" href="/groceries" sx={{textTransform:
"none", mr: 1}}>
            <Typography variant="h6">Продукти</Typography>
          </Button>
          <Button color="inherit" href="/recipes" sx={{textTransform:
"none"}}>
            <Typography variant="h6">Рецепти</Typography>
          </Button>
          <Box sx={{flexGrow: 1}}/>
          <Button color="inherit" onClick={() => setToken(null)}
sx={{textTransform: "none"}}>
            <Typography
variant="h6">👋&amp;nbsp;&amp;nbsp;&amp;nbsp;Вийти</Typography>
          </Button>
        </Toolbar>
```

```
      </AppBar>
      <Container sx={{mt: 4, width: "100%"}} maxWidth="xl">
        {children}
      </Container>
    </>
  )
}
// frontend/src/api.js

import {QueryClient, useMutation, useQuery} from 'react-query'
import axios from "axios";
import {appStore} from "./store";

export const queryClient = new QueryClient()

axios.defaults.baseURL = process.env.REACT_APP_BACKEND_URL;
axios.interceptors.request.use(
  (config) => {
    const token = appStore.getState().token;
    if (token !== null) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config
  },
  (error) => {
    return Promise.reject(error);
  },
);

axios.interceptors.response.use(
  function (response) {
    if (response.status === 200 || response.status === 201) {
      return response;
    }

    return Promise.reject(response);
  },
  function (error) {
    if (error.response.status === 401) {
      appStore.setState({token: null});
    }
    return Promise.reject(error);
  }
);

//Recipes
export function useRecipes() {
  return useQuery('recipes', () => axios.get('/recipes/').then(r =>
r.data));
}
```

```
export function useRecipe(id) {
  return useQuery(['recipe', id], () =>
axios.get(`/recipes/${id}`).then(r => r.data));
}

//Foods
export function useFoods() {
  return useQuery('foods', () => axios.get('/foods/').then(r => r.data));
}

//Meals
export function useMeals() {
  return useQuery('meals', () => axios.get('/meals/').then(r => r.data));
}


export function useCreateMeal() {
  return useMutation(
    g => axios.post('/meals/predict', g, {headers: {'Content-Type':
'multipart/form-data'}}),
    {
      onSuccess: () => {
        queryClient.invalidateQueries('meals')
      },
    })
}


export function useUpdateMeal() {
  return useMutation(
    req => axios.put(`/meals/${req.id}`, req),
    {
      onSuccess: () => {
        queryClient.invalidateQueries('meals')
      },
    })
}

export function useDeleteMeal() {
  return useMutation(
    id => axios.delete(`/meals/${id}`),
    {
      onSuccess: () => {
        queryClient.invalidateQueries('meals')
      },
    })
}

//Groceries
export function useGroceries() {
  return useQuery('groceries', () => axios.get('/groceries/').then(r =>
```

```
r.data));
}

export function useUpdateGrocery() {
  return useMutation(g => axios.put(`/groceries/${g.id}`, g), {
    onSuccess: () => {
      queryClient.invalidateQueries('groceries')
    },
  })
}

export function useCreateGrocery() {
  return useMutation(g => axios.post(`/groceries/`, g), {
    onSuccess: () => {
      queryClient.invalidateQueries('groceries')
    },
  })
}

export function useDeleteGrocery() {
  return useMutation(id => axios.delete(`/groceries/${id}`), {
    onSuccess: () => {
      queryClient.invalidateQueries('groceries')
    },
  })
}


// frontend/src/Register.jsx

import React, {useState} from 'react';
import axios from 'axios';
import {Button, Container, Link, TextField, Typography} from
'@mui/material';
import {useNavigate} from "react-router-dom"; // Import for visualization

export default function Register() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleRegister = async () => {
    try {
      await axios.post('/auth/register', {email, password});
      navigate('/login');
      alert('Registration successful!');
    } catch (error) {
      alert('Error registering user');
    }
  };
```

```
  return (
    <Container maxWidth="sm" sx={{p: 4, mt: 4, display: 'flex',
flexDirection: 'column', alignItems: 'stretch'}}>
      <Typography variant="h3" textAlign="center">Реєстрація</Typography>
      <TextField label="Пошта" type="email" fullWidth margin="normal"
value={email}
                 onChange={(e) => setEmail(e.target.value)}/>
      <TextField label="Пароль" type="password" fullWidth margin="normal"
value={password}
                 onChange={(e) => setPassword(e.target.value)}/>
      <Button variant="contained" color="primary"
onClick={handleRegister}>Зареєструватись</Button>
      <Link href="/login" textAlign="center" variant="caption">Вже маєте
зареєстровані? Авторизуйтесь в Хрум</Link>
    </Container>
  );
}
// frontend/src/setupTests.js

// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';

// frontend/src/App.js

import React from "react";
import {createBrowserRouter, RouterProvider} from 'react-router-dom';
import Register from "./Register";
import Login from "./Login";
import Dashboard from "./dashboard/Dashboard";
import {ThemeProvider} from "@mui/material";
import {theme} from "./mui";
import Groceries from "./groceries/Groceries";
import {QueryClientProvider} from "react-query";
import {queryClient} from "./api";
import Recipes from "./recipes/Recipes";
import Recipe from "./recipes/Recipe";


const router = createBrowserRouter(
  [
    {path: "/register", element: <Register/>},
    {path: "/login", element: <Login/>},
    {path: "/", element: <Dashboard/>},
    {path: "/groceries", element: <Groceries/>},
    {path: "/recipes", element: <Recipes/>},
    {path: "/recipes/:id", element: <Recipe/>},
  ],
  {
```

```
    future: {
      v7_relativeSplatPath: true,
      v7_fetcherPersist: true,
      v7_normalizeFormMethod: true,
      v7_partialHydration: true,
      v7_skipActionErrorRevalidation: true,
    },
  }
);

function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <ThemeProvider theme={theme}>
        <RouterProvider router={router} future={{v7_startTransition:
true}}/>)
      </ThemeProvider>
    </QueryClientProvider>
  );
}

export default App;
```

```
// backend/routers/recipes.py

from uuid import uuid4, UUID

from fastapi import HTTPException, APIRouter
from sqlalchemy import select
from sqlalchemy.orm import joinedload

from backend.db import DbDeps, Recipe, Food
from backend.dependencies import SuperUserDep, AppUserDep
from backend.schemas import RecipeCreate, RecipeUpdate, patch_model

router = APIRouter(prefix="/recipes", tags=["recipes"],
dependencies=[AppUserDep])


async def get_recipe_by_id(recipe_id: UUID, db: DbDeps):
    query = select(Recipe).where(Recipe.id ==
recipe_id).options(joinedload(Recipe.ingredients))
    recipe = (await db.execute(query)).unique().scalar_one_or_none()
    if not recipe:
        await db.close()
        raise HTTPException(status_code=404, detail="recipe not found")
    return recipe
```

```
@router.post("/", dependencies=[SuperUserDep])
async def create_recipe(req: RecipeCreate, db: DbDeps):
    recipe = Recipe(id=uuid4())
    patch_model(recipe, req, exclude=["ingredients"])
    await set_recipe_ingredients(db, recipe, req.ingredients)
    db.add(recipe)
    await db.commit()
    await db.refresh(recipe)
    await db.close()
    return recipe


@router.get("/{recipe_id}")
async def read_recipe(recipe_id: UUID, db: DbDeps):
    recipe = await get_recipe_by_id(recipe_id, db)
    await db.close()
    return recipe


@router.get("/")
async def list_recipes(db: DbDeps):
    query = select(Recipe).options(joinedload(Recipe.ingredients))
    groceries = (await db.execute(query)).unique().scalars().all()
    await db.close()
    return groceries


@router.put("/{recipe_id}", dependencies=[SuperUserDep])
async def update_recipe(recipe_id: UUID, req: RecipeUpdate, db: DbDeps):
    recipe = await get_recipe_by_id(recipe_id, db)
    patch_model(recipe, req, exclude=["ingredients"])
    if req.ingredients is not None:
        await set_recipe_ingredients(db, recipe, req.ingredients)

    db.commit()
    db.refresh(recipe)
    db.close()
    return recipe


@router.delete("/{recipe_id}", dependencies=[SuperUserDep])
async def delete_recipe(recipe_id: UUID, db: DbDeps):
    recipe = await get_recipe_by_id(recipe_id, db)
    db.delete(recipe)
    db.commit()
    db.close()
    return {"detail": "Recipe deleted"}


async def set_recipe_ingredients(db: DbDeps, recipe: Recipe, ingredients:
```

```
list[UUID]):
    recipe.ingredients.clear()
    for ingredient_id in ingredients:
        query = select(Food).where(Food.id == ingredient_id)
        food = (await db.execute(query)).scalar_one_or_none()
        if not food:
            db.close()
            raise HTTPException(status_code=404, detail="Food not found")
        recipe.ingredients.append(food)

// backend/routers/groceries.py

from uuid import UUID, uuid4

from fastapi import HTTPException, APIRouter
from sqlalchemy import select
from sqlalchemy.orm import joinedload

from backend.db import DbDeps, Grocery
from backend.dependencies import AppUser, AppUserDep
from backend.schemas import GroceryUpdate, GroceryCreate, patch_model

router = APIRouter(prefix="/groceries", tags=["groceries"],
dependencies=[AppUserDep])


async def get_grocery_by_id(grocery_id: UUID, user: AppUser, db: DbDeps):
    query = (select(Grocery)
             .where(Grocery.id == grocery_id)
             .where(Grocery.user_id == user.id)
             .options(joinedload(Grocery.food)))
    grocery = (await db.execute(query)).scalar_one_or_none()
    if not grocery:
        await db.close()
        raise HTTPException(status_code=404, detail="grocery not found")
    return grocery


@router.post("/")
async def create_grocery(req: GroceryCreate, user: AppUser, db: DbDeps):
    grocery = Grocery(id=uuid4(), user_id=user.id)
    patch_model(grocery, req)
    db.add(grocery)
    await db.commit()
    await db.refresh(grocery)
    await db.close()
    return grocery


@router.get("/{grocery_id}")
async def read_grocery(grocery_id: UUID, user: AppUser, db: DbDeps):
```

```
    grocery = await get_grocery_by_id(grocery_id, user, db)
    await db.close()
    return grocery


@router.get("/")
async def list_groceries(db: DbDeps, user: AppUser):
    groceries = (await db.execute(

select(Grocery).options(joinedload(Grocery.food)).where(Grocery.user_id
== user.id))).scalars().all()
    await db.close()
    return groceries


@router.put("/{grocery_id}")
async def update_grocery(grocery_id: UUID, req: GroceryUpdate, user:
AppUser, db: DbDeps):
    grocery = await get_grocery_by_id(grocery_id, user, db)
    patch_model(grocery, req)
    await db.commit()
    await db.refresh(grocery)
    await db.close()
    return grocery


@router.delete("/{grocery_id}")
async def delete_grocery(grocery_id: UUID, user: AppUser, db: DbDeps):
    grocery = await get_grocery_by_id(grocery_id, user, db)
    await db.delete(grocery)
    await db.commit()
    await db.close()
    return {"detail": "Grocery list deleted"}

// backend/routers/meals.py

import logging
import os
from uuid import UUID, uuid4

import cv2
import numpy as np
from fastapi import HTTPException, APIRouter, UploadFile
from fastapi.responses import FileResponse
from sqlalchemy import select

from backend.db import Meal, DbDeps
from backend.dependencies import FoodDetectorDep, AppUser, AppUserDep
from backend.food_detector import RESULT_IMAGES_PATH
from backend.food_detector_model import DetectionResult,
ProductCaloriesList
```

```
from backend.schemas import MealCreate, MealUpdate, MealRead,
MealProducts, MealProduct, patch_model

logger = logging.getLogger(__name__)
router = APIRouter(prefix="/meals", tags=["meals"],
dependencies=[AppUserDep])


async def get_meal_by_id(meal_id: UUID, db: DbDeps, user: AppUser):
    query = select(Meal).where(Meal.user_id == user.id).where(Meal.id ==
meal_id)
    db_meal = (await db.execute(query)).scalar_one_or_none()
    if not db_meal:
        await db.close()
        raise HTTPException(status_code=404, detail="Meal not found")
    return db_meal


def writeResultImage(meal: Meal, result: DetectionResult):
    cv2.imwrite(os.path.join(RESULT_IMAGES_PATH, str(meal.id) +
meal.image_extension), result.image)


@router.post("/predict", response_model=MealRead)
async def predict_meal(file: UploadFile, food_detector: FoodDetectorDep,
db: DbDeps, user: AppUser):
    if not file.filename:
        raise HTTPException(status_code=400, detail="No selected file")
    _, image_extension = os.path.splitext(file.filename)

    try:
        image = cv2.imdecode(np.frombuffer(await file.read(), np.uint8),
cv2.IMREAD_COLOR)
        result = food_detector.detect(image)
        detected_products: ProductCaloriesList = ProductCaloriesList([])
        products: MealProducts = MealProducts([])
        total_calories = 0.0

        for v in result.final_calories_data.values():
            calories = v.calories
            if calories is not None:
                detected_products.root.append(calories)
                products.root.append(MealProduct(**calories.dict()))
                total_calories += calories.calories
        total_calories = round(total_calories, 2)

        meal = Meal(id=uuid4(), user_id=user.id,
total_calories=total_calories, image_extension=image_extension,
                    detected_products=detected_products,
products=products)
        writeResultImage(meal, result)
```

```
    except Exception as e:
        logger.exception(e)
        raise HTTPException(status_code=500, detail=str(e))

    db.add(meal)
    await db.commit()
    await db.refresh(meal)
    await db.close()
    return meal


@router.get("/{meal_id}/result-image")
async def get_meal_result_image(meal_id: UUID, db: DbDeps, user:
AppUser):
    meal = await get_meal_by_id(meal_id, db, user)
    image_path = os.path.join(RESULT_IMAGES_PATH, str(meal.id) +
meal.image_extension)
    if not os.path.exists(image_path):
        raise HTTPException(status_code=404, detail="Meal result image
not found")

    return FileResponse(image_path)


@router.get("/{meal_id}", response_model=MealRead)
async def read_meal(meal_id: UUID, db: DbDeps, user: AppUser):
    meal = await get_meal_by_id(meal_id, db, user)
    await db.close()
    return meal


@router.get("/", response_model=list[MealRead])
async def list_meals(db: DbDeps, user: AppUser):
    meals = await db.execute(select(Meal).where(Meal.user_id == user.id))
    await db.close()
    return meals.scalars().all()


@router.post("/", response_model=MealRead)
async def create_meal(req: MealCreate, db: DbDeps, user: AppUser):
    meal = Meal(id=uuid4(), user_id=user.id)
    patch_model(meal, req)
    db.add(meal)
    await db.commit()
    await db.refresh(meal)
    await db.close()
    return meal


@router.put("/{meal_id}", response_model=MealRead)
async def update_meal(meal_id: UUID, req: MealUpdate, db: DbDeps, user:
```

```
AppUser):
    meal = await get_meal_by_id(meal_id, db, user)
    patch_model(meal, req)
    await db.commit()
    await db.refresh(meal)
    await db.close()
    return meal


@router.delete("/{meal_id}")
async def delete_meal(meal_id: UUID, db: DbDeps, user: AppUser):
    meal = await get_meal_by_id(meal_id, db, user)
    await db.delete(meal)
    await db.commit()
    await db.close()
    return {"detail": "Meal deleted"}

// backend/routers/foods.py

from uuid import UUID, uuid4

from fastapi import HTTPException, APIRouter
from sqlalchemy import select

from backend.db import Food, DbDeps
from backend.dependencies import SuperUserDep, AppUserDep
from backend.schemas import FoodCreate, FoodUpdate, FoodRead, patch_model

router = APIRouter(prefix="/foods", tags=["foods"],
dependencies=[AppUserDep])


async def get_food_by_id(food_id: UUID, db: DbDeps):
    query = select(Food).where(Food.id == food_id)
    db_food = (await db.execute(query)).scalar_one_or_none()
    if not db_food:
        await db.close()
        raise HTTPException(status_code=404, detail="Food not found")
    return db_food


@router.post("/", response_model=FoodCreate, dependencies=[SuperUserDep])
async def create_food(req: FoodCreate, db: DbDeps):
    food = Food(id=uuid4())
    patch_model(food, req)
    db.add(food)
    await db.commit()
    await db.refresh(food)
    await db.close()
    return food
```

```
@router.get("/{food_id}", response_model=FoodRead)
async def read_food(food_id: UUID, db: DbDeps):
    food = await get_food_by_id(food_id, db)
    await db.close()
    return food


@router.get("/", response_model=list[FoodRead])
async def list_foods(db: DbDeps):
    foods = await db.execute(select(Food))
    await db.close()
    return foods.scalars().all()


@router.put("/{food_id}", response_model=FoodCreate,
dependencies=[SuperUserDep])
async def update_food(food_id: UUID, req: FoodUpdate, db: DbDeps):
    food = await get_food_by_id(food_id, db)
    patch_model(food, req)
    await db.commit()
    await db.refresh(food)
    await db.close()
    return food


@router.delete("/{food_id}", dependencies=[SuperUserDep])
async def delete_food(food_id: UUID, db: DbDeps):
    food = await get_food_by_id(food_id, db)
    await db.delete(food)
    await db.commit()
    await db.close()
    return {"detail": "Food deleted"}

// backend/db.py

import json
import uuid
from collections.abc import AsyncGenerator
from datetime import datetime
from typing import Annotated

import pydantic
from fastapi import Depends
from fastapi_users.db import SQLAlchemyBaseUserTableUUID,
SQLAlchemyUserDatabase
from sqlalchemy import Column, String, UUID, Table, ForeignKey, Integer,
JSON, DateTime, Numeric
from sqlalchemy.ext.asyncio import AsyncSession, async_sessionmaker,
create_async_engine
from sqlalchemy.orm import DeclarativeBase, relationship, mapped_column,
```

```
Mapped

from backend.config import appConfig


class Base(DeclarativeBase):
    pass


class User(SQLAlchemyBaseUserTableUUID, Base):
    __tablename__ = 'users'
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    created = mapped_column(DateTime, default=datetime.now)
    modified = mapped_column(DateTime, default=datetime.now,
onupdate=datetime.now)

    def __str__(self):
        return f"User(email={self.email})"


recipe_ingredients = Table(
    'recipe_ingredients',
    Base.metadata,
    Column('recipe_id', UUID(as_uuid=True), ForeignKey('recipes.id')),
    Column('food_id', UUID(as_uuid=True), ForeignKey('foods.id')),
    Column('created', DateTime, default=datetime.now),
    Column('modified', DateTime, default=datetime.now,
onupdate=datetime.now),
)


class Food(Base):
    __tablename__ = 'foods'
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    name = mapped_column(String, index=True, unique=True)
    image_url = mapped_column(String)
    buy_url = mapped_column(String)
    created = mapped_column(DateTime, default=datetime.now)
    modified = mapped_column(DateTime, default=datetime.now,
onupdate=datetime.now)

    def __str__(self):
        return f"Food(name={self.name})"


class Recipe(Base):
    __tablename__ = 'recipes'
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
```

```
    name = mapped_column(String, index=True)
    description = mapped_column(String)
    instructions = mapped_column(String)
    image_url = mapped_column(String)
    created = mapped_column(DateTime, default=datetime.now)
    modified = mapped_column(DateTime, default=datetime.now,
onupdate=datetime.now)
    ingredients: Mapped[list[Food]] = relationship("Food",
secondary=recipe_ingredients)

    def __str__(self):
        return f"Recipe(id={self.id}, name={self.name})"


class Grocery(Base):
    __tablename__ = 'groceries'
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    user_id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
ForeignKey('users.id'))
    food_id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
ForeignKey('foods.id'))
    target_quantity = mapped_column(Integer)
    current_quantity = mapped_column(Integer)
    created = mapped_column(DateTime, default=datetime.now)
    modified = mapped_column(DateTime, default=datetime.now,
onupdate=datetime.now)
    user = relationship("User")
    food = relationship("Food")

    def __str__(self):
        return f"Grocery(food={self.food.name}, user={self.user.email})"


class Meal(Base):
    __tablename__ = 'meals'
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    user_id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
ForeignKey('users.id'))
    name = mapped_column(String, index=True)
    description = mapped_column(String)
    total_calories = mapped_column(Numeric(53, 2))
    image_extension = mapped_column(String)
    detected_products = mapped_column(JSON)
    products = mapped_column(JSON)
    created = mapped_column(DateTime, default=datetime.now)
    modified = mapped_column(DateTime, default=datetime.now,
onupdate=datetime.now)
    user = relationship("User")
```

```
    def __str__(self):
        return f"Meal(id={self.id}, user={self.user.email})"


def pydantic_json_serializer(obj) -> str:
    if isinstance(obj, pydantic.BaseModel):
        return json.dumps(obj.model_dump())
    else:
        return json.dumps(obj)


engine = create_async_engine(appConfig.database_url,
json_serializer=pydantic_json_serializer)
session_maker = async_sessionmaker(autocommit=False, autoflush=False,
bind=engine)


async def create_db_and_tables():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)


async def get_session() -> AsyncGenerator[AsyncSession, None]:
    async with session_maker() as session:
        yield session


DbDeps = Annotated[AsyncSession, Depends(get_session)]


async def get_user_db(session: AsyncSession = Depends(get_session)):
    yield SQLAlchemyUserDatabase(session, User)

// backend/config.py

import os

from pydantic import BaseModel


class AdminConfig(BaseModel):
    username: str
    password: str
    secret: str


class AppConfig(BaseModel):
    admin: AdminConfig
    jwt_token_secret: str
    reset_password_token_secret: str
    verification_token_secret: str
```

```
    database_url: str


appConfig = AppConfig(
    admin=AdminConfig(
        username=os.getenv("ADMIN_USERNAME"),
        password=os.getenv("ADMIN_PASSWORD"),
        secret=os.getenv("ADMIN_SECRET"),
    ),
    jwt_token_secret=os.getenv("JWT_SECRET"),
    reset_password_token_secret=os.getenv("RESET_PASSWORD_SECRET"),
    verification_token_secret=os.getenv("VERIFICATION_SECRET"),
    database_url=os.getenv("DATABASE_URL",
"postgresql+asyncpg://hroom:hroom@localhost:5432/hroom"),
)

// backend/food_detector_model.py

from typing import Dict, Optional, List

import cv2
from pydantic import BaseModel, ConfigDict, RootModel

Cv2Image = cv2.typing.MatLike


class BoundingBox(BaseModel):
    x: int
    y: int
    w: int
    h: int

    def to_list(self) -> list[int]:
        return [self.x, self.y, self.w, self.h]

    def to_tuple(self) -> tuple[int, int, int, int]:
        return self.x, self.y, self.w, self.h


class DetectObjectResult(BaseModel):
    class_ids: list[int]
    confidences: list[float]
    boxes: list[BoundingBox]

    def boxes_to_list(self) -> list[list[float]]:
        return [[box.x, box.y, box.w, box.h] for box in self.boxes]


class SegmentedImagesData(BaseModel):
    model_config = ConfigDict(arbitrary_types_allowed=True)
```

```
    cv2_img: Cv2Image
    mask_product: Cv2Image
    img_contour: Cv2Image
    img_contour4: Cv2Image
    Thumb_img_min_rectangle: Optional[Cv2Image]


class SegmentedObjContourAreaPixel(BaseModel):
    model_config = ConfigDict(arbitrary_types_allowed=True)

    req_contour: Cv2Image
    req_object_area: float
    pix_to_cm_multiplier: Optional[float] = None


class CalculatedCroppedCoordinates(BaseModel):
    y_min: int
    y_max: int
    x_min: int
    x_max: int


class CropResult(BaseModel):
    model_config = ConfigDict(arbitrary_types_allowed=True)

    img_name: str
    cropped_image: Cv2Image
    pixel_margin: int
    input_img_shape: tuple[int, ...]
    bb_cordinate: BoundingBox
    calculated_cropped_coordinates: CalculatedCroppedCoordinates


class ProductCalories(BaseModel):
    name: str
    title: str
    volume: float
    calories_100grams: int
    mass: float
    calories: float


class CaloriesData(BaseModel):
    name: str
    bb: BoundingBox
    calories: Optional[ProductCalories]


class DetectionResult(BaseModel):
    model_config = ConfigDict(arbitrary_types_allowed=True)
```

```
    image: cv2.typing.MatLike
    final_calories_data: Dict[str, CaloriesData]


class ProductCaloriesList(RootModel):
    root: List[ProductCalories]

// backend/users.py

import uuid
from typing import Optional

from fastapi import APIRouter
from fastapi import Depends, Request
from fastapi_users import BaseUserManager, UUIDIDMixin
from fastapi_users import FastAPIUsers
from fastapi_users.authentication import AuthenticationBackend,
BearerTransport, JWTStrategy

from backend.config import appConfig
from backend.schemas import UserRead, UserCreate, UserUpdate
from db import User, get_user_db


class UserManager(UUIDIDMixin, BaseUserManager[User, uuid.UUID]):
    reset_password_token_secret = appConfig.reset_password_token_secret
    verification_token_secret = appConfig.verification_token_secret

    async def on_after_register(self, user: User, request:
Optional[Request] = None):
        print(f"User {user.id} has registered.")

    async def on_after_forgot_password(
            self, user: User, token: str, request: Optional[Request] =
None
    ):
        print(f"User {user.id} has forgot their password. Reset token:
{token}")

    async def on_after_request_verify(
            self, user: User, token: str, request: Optional[Request] =
None
    ):
        print(f"Verification requested for user {user.id}. Verification
token: {token}")


async def get_user_manager(user_db=Depends(get_user_db)):
    yield UserManager(user_db)
```

```
bearer_transport = BearerTransport(tokenUrl="auth/jwt/login")


def get_jwt_strategy() -> JWTStrategy:
    return JWTStrategy(secret=appConfig.jwt_token_secret,
lifetime_seconds=3600)


auth_backend = AuthenticationBackend(
    name="jwt",
    transport=bearer_transport,
    get_strategy=get_jwt_strategy,
)

# FastAPI Users
fastapi_users = FastAPIUsers[User, uuid.UUID](get_user_manager,
[auth_backend])
current_active_user = fastapi_users.current_user(active=True)
current_superuser = fastapi_users.current_user(active=True,
superuser=True)


# Create APIRouter for users
def get_users_router():
    router = APIRouter()
    router.include_router(
        fastapi_users.get_auth_router(auth_backend),
        prefix="/auth/jwt",
        tags=["auth"],
    )
    router.include_router(
        fastapi_users.get_register_router(UserRead, UserCreate),
        prefix="/auth",
        tags=["auth"],
    )
    router.include_router(
        fastapi_users.get_verify_router(UserRead),
        prefix="/auth",
        tags=["auth"],
    )
    router.include_router(
        fastapi_users.get_reset_password_router(),
        prefix="/auth",
        tags=["auth"],
    )
    router.include_router(
        fastapi_users.get_users_router(UserRead, UserUpdate),
        prefix="/users",
        tags=["users"],
    )
    return router
```

```
// backend/image_helpers.py

import os

import cv2

from backend.food_detector_model import Cv2Image, BoundingBox,
CalculatedCroppedCoordinates, CropResult


def crop_img(input_image: Cv2Image, input_image_name: str, bb:
BoundingBox, image_save_path: str = None,
             pixel_margin: int = 5):
    dh, dw, cha = input_image.shape

    for field in ['x', 'y', 'w', 'h']:
        attr = getattr(bb, field)
        if attr < 0:
            setattr(bb, field, abs(attr))

    x_min = max(bb.x - pixel_margin, 0)
    y_min = max(bb.y - pixel_margin, 0)
    x_max = min(bb.x + bb.w + pixel_margin, dw)
    y_max = min(bb.y + bb.h + pixel_margin, dh)

    cropped_image = input_image[y_min:y_max, x_min:x_max]

    # Для дебагінгу підрізки фото можна передати шлях куди зберегти
підрізане фото
    if image_save_path:
        cv2.imwrite(os.path.join(image_save_path, input_image_name),
cropped_image)

    return CropResult(
        img_name=input_image_name,
        cropped_image=cropped_image,
        pixel_margin=pixel_margin,
        input_img_shape=input_image.shape,
        bb_cordinate=bb,

calculated_cropped_coordinates=CalculatedCroppedCoordinates(y_min=y_min,
y_max=y_max, x_min=x_min, x_max=x_max)
    )

// backend/food_detector.py

import os

import cv2
import numpy as np
```

```
import image_helpers as ih
from backend.food_detector_model import DetectObjectResult, Cv2Image,
SegmentedObjContourAreaPixel, BoundingBox, \
    DetectionResult
from calories import Calories

MODEL_WORKDIR = os.getcwd() + r"/../model"
MODEL_CONFIG_PATH = os.path.join(MODEL_WORKDIR, r"config")
MODEL_PATH = os.path.join(MODEL_WORKDIR, r"weights")
CLASSES_FILE_PATH = os.path.join(MODEL_CONFIG_PATH, r"obj.names")
CONFIG_FILE_PATH = os.path.join(MODEL_CONFIG_PATH, r"yolo4-hroom.cfg")
RESULT_IMAGES_PATH = os.path.join(MODEL_WORKDIR, r"result")
PIXEL_TO_CM_MULTIPLIER_CONSTANT = 5.0


class FoodDetector:
    def __init__(self):
        self.classes = self.read_classes()
        self.net = cv2.dnn.readNet(MODEL_PATH, CONFIG_FILE_PATH)
        self.layer_names = self.net.getLayerNames()
        self.output_layers = [self.layer_names[i - 1] for i in
self.net.getUnconnectedOutLayers()]
        self.calories = Calories()
        self.colors = np.random.uniform(0, 255, size=(len(self.classes),
3))

    def detect(self, input_image: cv2.typing.MatLike, scaling="scaled"):
        image = self.scale_image(input_image, scaling)
        objects = self.detect_objects_from_image(image)

        if not objects.class_ids:
            raise ValueError("Can't detect objects inside image!")

        return self.find_required_detections(image, objects)

    def scale_image(self, image: cv2.typing.MatLike, scaling: str) ->
cv2.typing.MatLike:
        if scaling == "default":
            return cv2.resize(image, (608, 608))
        elif scaling == "scaled":
            return cv2.resize(image, None, fx=0.5, fy=0.5)
        elif scaling == "original":
            return image
        else:
            raise ValueError("Invalid scaling value")

    def read_classes(self):
        with open(CLASSES_FILE_PATH, "r") as f:
            return [line.strip() for line in f.readlines()]
```

```
    def get_colors(self):
        return self.colors

    def detect_objects_from_image(self, input_array_img,
baseline_confidence=0.5, input_img_size=(608, 608)):
        height, width, _ = input_array_img.shape

        self.net.setInput(cv2.dnn.blobFromImage(input_array_img, 0.00392,
input_img_size, (0, 0, 0), True, crop=False))
        outs = self.net.forward(self.output_layers)

        class_ids, confidences, boxes = [], [], []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]

                if confidence > baseline_confidence:
                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)

                    boxes.append(BoundingBox(x=x, y=y, w=w, h=h))
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        return DetectObjectResult(class_ids=class_ids,
confidences=confidences, boxes=boxes)

    def image_segmentation(self, cropped_img_name: str, cropped_img:
Cv2Image) -> SegmentedObjContourAreaPixel:
        image_gray = cv2.cvtColor(cropped_img, cv2.COLOR_BGR2GRAY)
        adaptive_threshold = cv2.adaptiveThreshold(image_gray, 100,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,
                                                   15, 2)

        contours, _ = cv2.findContours(adaptive_threshold, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
        largest_areas = sorted(contours, key=cv2.contourArea)

        mask = np.zeros(image_gray.shape, np.uint8)
        cv2.drawContours(mask, [largest_areas[-1]], 0, (255, 255, 255,
255), -1)

        image_bitcontour = cv2.bitwise_or(cropped_img, cropped_img,
mask=mask)
        hsv_img = cv2.cvtColor(image_bitcontour, cv2.COLOR_BGR2HSV)
```

```
        mask_plate = cv2.inRange(hsv_img, np.array([0, 0, 50]),
np.array([200, 90, 250]))
        mask_not_plate = cv2.bitwise_not(mask_plate)
        mask_product = cv2.bitwise_and(image_bitcontour,
image_bitcontour, mask=mask_not_plate)

        image_gray2 = cv2.cvtColor(mask_product.copy(),
cv2.COLOR_BGR2GRAY)
        adaptive_threshold2 = cv2.adaptiveThreshold(image_gray2, 100,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,
                                                    15, 2)

        contours2, _ = cv2.findContours(adaptive_threshold2,
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
        largest_areas2 = sorted(contours2, key=cv2.contourArea)

        result: SegmentedObjContourAreaPixel

        pix_to_cm_multiplier = None
        req_contour = largest_areas2[-1]

        if cropped_img_name.startswith("thumb"):
            pix_to_cm_multiplier = (PIXEL_TO_CM_MULTIPLIER_CONSTANT /
max(cv2.minAreaRect(req_contour)[1]))

        return SegmentedObjContourAreaPixel(
            req_contour=req_contour,
            req_object_area=cv2.contourArea(req_contour),
            pix_to_cm_multiplier=pix_to_cm_multiplier
        )

    def find_required_detections(self, input_image: cv2.typing.MatLike,
objects: DetectObjectResult):
        class_ids, confidences, boxes = objects.class_ids,
objects.confidences, objects.boxes_to_list()
        detections = {}

        for i in range(len(boxes)):
            if i in cv2.dnn.NMSBoxes(boxes, confidences, 0.4, 0.6):
                name =
f"{str(self.classes[class_ids[i]])}({round(confidences[i] * 100,
2)})_{i}.jpg"
                detections[name] = objects.boxes[i]

        segmentation_to_calorie = {}

        for image_name in detections:
            cropped = ih.crop_img(input_image=input_image,
input_image_name=image_name, bb=detections[image_name])
            segmentation = self.image_segmentation(cropped.img_name,
cropped.cropped_image)
```

```
            segmentation_to_calorie[cropped.img_name] = (segmentation,
detections[image_name])

        final_calories_data =
self.calories.getCalories(segmentation_to_calorie)
        result_image = self.draw_detections(input_image,
final_calories_data)

        return DetectionResult(image=result_image,
final_calories_data=final_calories_data)

    def draw_detections(self, input_image: cv2.typing.MatLike,
final_data: dict) -> cv2.typing.MatLike:
        font = cv2.FONT_HERSHEY_COMPLEX
        result_image = input_image.copy()
        for i, k in enumerate(final_data):
            name, bb, calories = final_data[k].name, final_data[k].bb,
final_data[k].calories
            x, y, w, h = bb.to_tuple()

            color = self.colors[i % len(self.colors)]
            cv2.rectangle(result_image, (x, y), (x + w, y + h), color, 2)
            half_height = int(h / 2)
            cv2.putText(result_image, name, (x, y + half_height), font,
0.75, (255, 255, 255), 1)
            if calories is not None:
                calories_text = f"{round(calories.calories)} ккал"
                cv2.putText(result_image, calories_text,
                            (x + int(w / 2) - len(calories_text) * 8, y +
half_height + 25), font, 0.75,
                            (255, 255, 255), 1)
        return result_image

// backend/calories.py

import re
from typing import Optional

import cv2
import numpy as np
from pydantic import BaseModel

from backend.food_detector_model import SegmentedObjContourAreaPixel,
BoundingBox, ProductCalories, CaloriesData

# We know area of thumb. It is 5*2.3 cm².
# So area per pixel is actual thumb area divided by pixel thumb area of
thumb.
SKIN_MULTIPLIER = 6 * 2.5
```

```
class Product(BaseModel):
    title: str
    calories: Optional[int]
    density: Optional[float]
    shape: Optional[str]


def product(title: str, shape: Optional[str], calories: Optional[int],
density: Optional[float]):
    return Product(title=title, shape=shape, calories=calories,
density=density)


class Calories:
    def __init__(self):
        self.products = {
            "thumb": product("Палець", None, None, 0.641),
            "Apple": product("Яблуко", "sphere", 52, 0.96),
            "Orange": product("Апельсин", "sphere", 47, 0.482),
            "Qiwi": product("Ківі", "sphere", 44, 0.575),
            "Tomato": product("Томат", "sphere", 18, 0.481),
            "Onion": product("Цибуля", "sphere", 40, 0.513),
            "Carrot": product("Морква", "cylinder", 41, 0.641),
            "Banana": product("Банан", "cylinder", 89, 0.94),
        }

    def getCalorie(self, label, volume) -> ProductCalories:  # # volume
in cm^3
        productConfig = self.products[label]
        calories_100grams = productConfig.calories
        density = productConfig.density
        title = productConfig.title
        mass = volume * density * 1.0
        calories = (calories_100grams / 100.0) * mass
        return ProductCalories(name=label, title=title,
volume=round(volume, 2), mass=round(mass, 2),
                               calories=round(calories, 2),
                               calories_100grams=calories_100grams)

    def getVolume(self, label, area, skin_area, pix_to_cm_multiplier,
product_contour):
        area_product = (area / skin_area) * SKIN_MULTIPLIER
        volume: float = 100

        shape = self.products[label].shape

        if shape == "sphere":
            radius = np.sqrt(area_product / np.pi)
            volume = (4 / 3) * np.pi * radius * radius * radius

        if shape == "cylinder":
```

```
            product_rect = cv2.minAreaRect(product_contour)
            height = max(product_rect[1]) * pix_to_cm_multiplier
            radius = area_product / (2.0 * height)
            volume = np.pi * radius * radius * height

        return volume

    def getCalories(self, segmentation_to_calorie: dict[str,
tuple[SegmentedObjContourAreaPixel, BoundingBox]]) -> dict[
        str, CaloriesData]:
        product_calories: dict[str, CaloriesData] = {}

        # Variables to store contour area and pixel-to-cm multiplier for
the reference object (thumb)
        skin_contour_Area = pix_cm = None

        for k in segmentation_to_calorie:
            if k.startswith("thumb"):
                skin_contour_Area =
segmentation_to_calorie[k][0].req_object_area
                pix_cm =
segmentation_to_calorie[k][0].pix_to_cm_multiplier
                thumb__title = self.products["thumb"].title
                confidence = self.parseSegmentationParams(k)[1]
                name = f"{thumb__title}({confidence})"
                product_calories[k] = CaloriesData(name=name,
bb=segmentation_to_calorie[k][1],
                                                   calories=None)

        for i, k in enumerate(segmentation_to_calorie):
            if not k.startswith("thumb"):
                name, confidence, bbNum = self.parseSegmentationParams(k)
                title = self.products[name].title
                product_contour =
segmentation_to_calorie[k][0].req_contour
                product_contour_area =
segmentation_to_calorie[k][0].req_object_area

                caloriesResult = None

                # Якщо є еталонний об'єкт, то рахуємо калорії
                if skin_contour_Area is not None and pix_cm is not None:
                    volume = self.getVolume(name, product_contour_area,
skin_contour_Area, pix_cm, product_contour)
                    caloriesResult = self.getCalorie(name, volume)

                product_calories[k] = CaloriesData(name=f"{i + 1}.
{title} {confidence}%",

bb=segmentation_to_calorie[k][1],
```

```
calories=caloriesResult)

        return product_calories

    def parseSegmentationParams(self, segmentation):
        return re.match(r"(.+?)\((.+)\)_(\d+)\..+",
segmentation).groups()

// backend/schemas.py

import uuid
from datetime import datetime
from typing import Optional, List

from fastapi_users import schemas
from pydantic import BaseModel, RootModel


class UserRead(schemas.BaseUser[uuid.UUID]):
    pass


class UserCreate(schemas.BaseUserCreate):
    pass


class UserUpdate(schemas.BaseUserUpdate):
    pass


class FoodRead(BaseModel):
    id: uuid.UUID
    name: str
    buy_url: str
    image_url: str


class FoodCreate(BaseModel):
    name: str
    buy_url: str
    image_url: str


class FoodUpdate(BaseModel):
    name: str = None
    buy_url: str = None
    image_url: str


class GroceryRead(BaseModel):
    id: uuid.UUID
```

```
    food_id: uuid.UUID
    target_quantity: int
    current_quantity: int


class GroceryCreate(BaseModel):
    food_id: uuid.UUID
    target_quantity: int
    current_quantity: int = 0


class GroceryUpdate(BaseModel):
    target_quantity: int = None
    current_quantity: int = None


class RecipeCreate(BaseModel):
    name: str
    description: str
    instructions: str
    ingredients: list[uuid.UUID]
    image_url: str


class RecipeUpdate(BaseModel):
    name: str = None
    description: str = None
    instructions: str = None
    ingredients: list[uuid.UUID] = None
    image_url: str


class MealProduct(BaseModel):
    name: str
    title: str
    calories: float


class MealProducts(RootModel):
    root: List[MealProduct]


class MealRead(BaseModel):
    id: uuid.UUID
    user_id: uuid.UUID
    name: Optional[str]
    description: Optional[str]
    total_calories: float
    products: Optional[MealProducts]
    created: datetime
    modified: datetime
```

```
class MealCreate(BaseModel):
    name: Optional[str]
    description: Optional[str] = None
    total_calories: float = None
    products: Optional[MealProducts]
    pass;


class MealUpdate(BaseModel):
    name: Optional[str] = None
    description: Optional[str] = None
    total_calories: float = None
    products: MealProducts = None
    pass;


def patch_model(model, patch: BaseModel, exclude=None):
    if exclude is None:
        exclude = []

    for attr in patch.model_fields:
        if getattr(patch, attr) is not None and attr not in exclude:
            setattr(model, attr, getattr(patch, attr))

// backend/admin.py

from fastapi import FastAPI
from sqladmin import Admin, ModelView
from sqladmin.authentication import AuthenticationBackend
from starlette.requests import Request
from wtforms.fields.simple import TextAreaField

from backend.config import appConfig
from backend.db import engine, User, Food, Recipe, Grocery, Meal


class AdminView(ModelView):
    is_async = True


class UserAdmin(AdminView, model=User):
    column_exclude_list = ["id"]


class FoodAdmin(AdminView, model=Food):
    column_exclude_list = ["id"]


class GroceryAdmin(AdminView, model=Grocery):
```

```
    name_plural = "Groceries"
    column_exclude_list = ["food_id", "user_id", "id"]


class RecipeAdmin(AdminView, model=Recipe):
    form_overrides = dict(instructions=TextAreaField)
    column_exclude_list = ["id", "instructions", "ingredients",
"image_url"]


class MealsAdmin(AdminView, model=Meal):
    column_exclude_list = ["id", "user_id", "total_calories",
"image_extension", "products", "detected_products",
                           "description"]


def initAdmin(app: FastAPI):
    admin = Admin(app, engine,
authentication_backend=AdminAuth(appConfig.admin.secret))
    admin.add_view(UserAdmin)
    admin.add_view(FoodAdmin)
    admin.add_view(GroceryAdmin)
    admin.add_view(RecipeAdmin)
    admin.add_view(MealsAdmin)


class AdminAuth(AuthenticationBackend):
    def __init__(self, secret_key: str) -> None:
        self.secret = secret_key
        super().__init__(secret_key)

    async def login(self, request: Request) -> bool:
        form = await request.form()
        username, password = form["username"], form["password"]

        if username != appConfig.admin.username or password !=
appConfig.admin.password:
            return False

        request.session.update({"token": self.secret})

        return True

    async def logout(self, request: Request) -> bool:
        request.session.clear()
        return True

    async def authenticate(self, request: Request) -> bool:
        token = request.session.get("token")

        if not token:
```

```
            return False

        return token == self.secret
```

```
// backend/app.py

from contextlib import asynccontextmanager

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from backend.admin import initAdmin
from backend.db import create_db_and_tables
from backend.routers import recipes, foods, groceries, meals
from food_detector import FoodDetector
from users import get_users_router


@asynccontextmanager
async def lifespan(app: FastAPI):
    await create_db_and_tables()
    yield


app = FastAPI(lifespan=lifespan)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
initAdmin(app)
food_detector = FoodDetector()

app.include_router(get_users_router())
app.include_router(recipes.router)
app.include_router(foods.router)
app.include_router(groceries.router)
app.include_router(meals.router)

// backend/main.py

import uvicorn

if __name__ == "__main__":
    uvicorn.run("app:app", host="0.0.0.0", log_level="info")

// backend/dependencies.py

from typing import Annotated
```

```
from fastapi import Depends

from backend.db import User
from backend.food_detector import FoodDetector
from backend.users import current_active_user, current_superuser


def get_food_detector():
    return FoodDetector()


FoodDetectorDep = Annotated[FoodDetector, Depends(get_food_detector)]
AppUserDep = Depends(current_active_user)
AppUser = Annotated[User, AppUserDep]
SuperUserDep = Depends(current_superuser)
SuperUser = Annotated[User, SuperUserDep]
```