НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Виконав студент IV курсу, групи ІП-91
(шифр групи)

Шейко Максим Олексійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл. Халус О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. викл. Халус О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент проф. каф ІСТ, д.т.н., проф., Теленик С.Ф. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Шейку Максиму Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів
керівник проєкту Халус Олена Андріївна, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту «17» червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: визначення актуальності, аналіз та опис предментої області, огляд ринку програмних продуктів, постановка задачі.
2) Моделювання програмного забезпечення, опис бізнес-процесів
3) Інформаційне забезпечення: опис структури бази даних, опис конструювання програмного забезпечення, опис аналізу безпеки даних
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
5) Технологічний розділ: керівництво користувача, програма та методика тестування програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна класів класів програмного забезпечення
2) Схема структурна класів компонентів програмного забезпечення
3) Креслення вигляду екранних форм
4)
5)
6)

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	03.03.2023	
3	Постановка та формалізація задачі	19.03.2023	
4	Розробка інформаційного забезпечення	30.03.2023	
5	Алгоритмізація задачі	05.04.2023	
6	Обґрунтування вибору використаних технічних засобів	10.04.2023	
7	Розробка програмного забезпечення	14.04.2023	
8	Налагодження програми	12.05.2023	
9	Виконання графічних документів	20.05.2023	
10	Оформлення пояснювальної записки	25.05.2023	
11	Подання ДП на попередній захист	06.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Максим ШЕЙКО
(підпис) (ініціали, прізвище)

Керівник _____ Олена ХАЛУС
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 48 таблиць, 37 рисунків та 16 джерел – загалом 87 сторінки.

Дипломний проєкт присвячений розробці веб-застосунку для відстежування запасів продуктів у користувача та підбору рецептів з наявних або випадкових інгредієнтів з урахуванням кулінарних сподобань та дієтичних обмежень користувача.

Метою розробки даного веб-застосунку є сприяння формуванню більш усвідомленої культури споживання у середньостатистичного користувача задля зменшення кількості харчових відходів через вчасне споживання продуктів.

У розділі першому проаналізовано та досліджено предметну область, визначено актуальність теми, знайдені та порівняні аналоги програмного забезпечення, розглянуті технічні рішення, розроблені та описані функціональні та нефункціональні вимоги до ПЗ.

Розділ другий присвячений моделюванню програмного забезпечення, опису бізнес-процесів, архітектури продукту, що розроблюється.

У третьому розділі розглядається аналіз розробленого програмного забезпечення та проведено тестування веб-застосунку.

Четвертий розділ присвячений процесу розгортання веб-застосунку та випуску програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ANGULAR, .NET, РЕЦЕПТ, БАЗА ДАНИХ, СТАТИСТИКА, AZURE

ABSTRACT

The explanatory note of the diploma project consists of four chapters, 48 tables, 37 figures, and 16 sources - 87 pages in total.

The diploma project is devoted to the development of a web application for tracking the user's food stocks and selecting recipes from available or random ingredients, considering the user's culinary preferences and dietary restrictions.

The purpose of developing this web application is to promote a more conscious consumption culture among the average user to reduce food waste through timely consumption of food.

Chapter One analyzes and explores the subject area, determines the relevance of the topic, finds and compares software analogues, considers technical solutions, and develops and describes functional and non-functional requirements for the software.

Section two is devoted to software modeling, description of business processes, and architecture of the product under development.

The third section deals with the analysis of the developed software and the testing of the web application.

The fourth section is devoted to the process of web application deployment and software release.

KEYWORDS: WEB APPLICATION, ANGULAR, .NET, RECIPE, DATABASE, STATISTICS, AZURE

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-9128.045440.01.91	Технічне завдання	13	
3	А4	КПІ.ІП-9128.045440.02.81	Пояснювальна записка	87	
4	А4	КПІ.ІП-9128.045440.03.12	Текст програми	56	
5	А4	КПІ.ІП-9128.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-9128.045440.05.34	Керівництво користувача	20	
7	А3	КПІ.ІП-9128.045440.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-9128.045440.00.90		
Розроб.		Шейко М.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Халус О.А.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Технічне завдання

КПІ.ІП-9128.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Максим ШЕЙКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів.

Галузь застосування: дане програмне забезпечення має загальну галузь застосування.

Наведене технічне завдання поширюється на розробку веб-застосунку для відстеження запасів продуктів та підбір рецептів кулінарних блюд із наявних продуктів. Додаток використовується для відстеження стану продуктів, що придбаваються, знаходження кулінарних рецептів з наявних або випадкових інгредієнтів, аналізу кількісті витрачених грошей на продути. Застосунок призначений для користування у повсякденному житті.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосунку для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для допомоги людям слідкувати за продуктами, що купуються щодня, не витрачати їх марно та аналізувати кількість грошей, що витрачаються на них.

Метою розробки є створення веб-застосунку, який має функціонал як генерації кулінарних рецептів, так і трекінгу продуктів, які купуються в магазинах.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу:

– всі графічні компоненти користувацького інтерфейсу мають бути розділені порожнім місцем, не накладатися один на одного;

– заголовки мають мати розмір шрифту у 24мм;

– основний текст має мати розмір шрифту у 16мм;

– основний текст має мати розмір шрифту у 16мм;

– кнопки, що знаходяться на користувацькому інтерфейсі, повинні спрацьовувати при натисканні або надсилати повідомлення про помилку;

– компонент навігації та низу сторінки мають бути спільними для всіх інших компонентів.

4.1.2 Для анонімного користувача:

а) можливість створення облікового запису:

1) форма реєстрації повинна мати поле вводу «Ім’я»;

2) форма реєстрації повинна мати поле вводу «Електронна пошта»;

3) форма реєстрації повинна мати поле вводу «Пароль»;

4) форма реєстрації повинна мати поле вводу «Підтвердження паролю»;

б) можливість авторизації в вже існуючий обліковий запис:

1) форма авторизації повинна мати поле вводу «Електронна пошта»;

2) форма авторизації повинна мати поле вводу «Пароль»;

в) можливість генерації рецептів з певних інгредієнтів:

1) вибрати з перелічених інгредієнтів за списком або пошуком;

2) кількість вибраних інгредієнтів є необмеженою;

3) сторінка пошуку має компонент додаткових критеріїв пошуку;

4.1.3 Для зареєстрованого користувача:

а) можливість додавати в інвентар продукти:

1) форма додавання продукту повинна мати поле вводу назву інгредієнта;

2) форма додавання продукту повинна мати поле вводу інформації про термін придатності;

3) форма додавання продукту повинна мати поле вводу ціни продукту;

б) за 7 днів до закінчення терміну придатності продукту, додаток сповіщає користувача на сторінці інвентаря про це шляхом підсвічування жовтим кольором кінцевої дати придатності на картці. За 3 дні – дана інформація повинна підсвічуватися червоним кольором;

в) додаток повинен мати окрему сторінку статистики, де зображена інформація про витрачені на продукти гроші у розрізі року, місяця та дня.

4.1.4 Додаткові вимоги:

– інтерфейс додатку має зручний для користування дизайн у мінімалістичному стилі;

– веб-застосунок повинен бути простим для використання.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

Введена інформація, перед відправкою на сервер має бути спочатку перевірена на клієнтській стороні додатку. В випадку якщо дані не можуть бути відправлені на сервер – показувати відповідне повідомлення.

4.2.2 Передбачити захист від некоректних дій користувача.

Дані які вводить користувач, мають спочатку перевірятись на відповідність на клієнтській стороні. Якщо до прикладу введена інформація не задовольняє умову перевірки на коректність – вивести відповідне повідомлення. Забезпечити додаткову перевірку на серверній стороні додатку, для уникнення запису некоректних даних. Результат перевірки сервера також потрібно виводити в повідомленні для користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 16 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в форматі полів вводу.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в форматі повідомлень.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# для серверної частини та TypeScript для клієнтської.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Visual Studio 2022 та Webstorm IDE.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на платформі GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Розгорнути клієнтську частину, серверну частину веб-застосунку, сховище даних з використанням хмарної платформи Microsoft Azure.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна компонентів програмного забезпечення;

– схема структурна класів програмного забезпечення;

– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки застосування наведено у таблиці 6.1.

Таблиця 6.1 – Стадії та етапи розробки

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.04	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.04	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування.

Пояснювальна записка
до дипломного проєкту

на тему: Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

КПІ.ІП-9128.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SDK	–	Software Development Kit, набір для розробки програмного забезпечення.
BPMN	–	Business Process Model and Notation, система умовних позначень та моделювання бізнес-процесів.
ООП	–	Об’єктно-орієнтоване програмування, одна з головних парадигм програмування.
ORM	–	Object-Relational Mapping, технологія в програмуванні, що допомагає об’єднати ООП та реляційні бази даних.
.NET	–	Платформа для розробки програмного забезпечення від Microsoft.
EF	–	Entity Framework, пакет ORM на платформі .NET.
API	–	Application Programming Interface, прикладний програмний Інтерфейс.
UX/UI	–	User Experience, User Interface – проектування користувацьких інтерфейсів, в яких зручність користування мають таку ж важливість, як і приємний зовнішній вигляд.
MVC	–	Model-View-Controller, схема розбиття даних застосунку.

ВСТУП

Темою дипломної роботи є проєктування та розробка веб-застосунку для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних інгредієнтів. Головною причиною, що посприяла вибору даної теми для дипломного проєкту стала її актуальність. Нераціональне використання ресурсів, в тому числі й харчових, призводить до збільшення кількості відходів, які постійно забруднюють нашу планету. Також існує проблема з відсутністю різноманіття харчування в значної частини людей. Все більше людей зараз готують однакову їжу через нестачу часу або ідей щодо приготування нових кулінарних блюд із вже наявних інгредієнтів.

Розвиток технологій не є кінцевою метою в цілому, будь-яка технологія є інструментом для покращення якості життя людини або засобом для досягнення цілі, поставленою людиною. Тому головна ціль даного проєкту – це розробити програмне забезпечення, що здатне зацікавити сучасного користувача та може слугувати як невеликий елемент розв’язання глобальних викликів.

Застосунок повинен мати функціонал з відстеження харчових продуктів, що придбаваються, їх стану (терміну придатності тощо); функцію підбору різноманітних кулінарних рецептів з вже наявних або випадкових інгредієнтів для створення більш корисного та збалансованого раціону харчування й одночасного зменшення кількості харчових відходів через споживання продуктів до закінчення їх терміну придатності.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Сьогодні людство стикається з низкою екологічних, економічних та соціальних проблем. Питання харчування має важливе місце в кожній з них. Зайве використання харчових ресурсів призводить до збільшення відходів. Це відбувається в той же час, коли в світі 828 мільйонів людей знаходяться на межі голоду[1] через економічні проблеми або які знаходяться в тих регіонах світу, що потерпають від війн та конфліктів. Також за статистикою щороку викидається понад 1.3 мільярда тон їжі, що в еквіваленті дорівнює 940 мільярдів доларів у світовій економіці[2]. Проте в світі виробляється достатньо продуктів, щоб прогодувати всіх та майже половина з овочів і фруктів попадає у відходи. Харчові відходи сприяють глобальному потеплінню, оскільки на світову продовольчу систему припадає близько 30% парникових викидів. Це відбувається через те, що коли їжу викидають на смітник, вона починає виділяти в процесі розкладання метан та вуглекислий газ, що сприяють процесу зміни клімату. Продовольча та сільськогосподарська організація Об'єднаних Націй у 2013 році підрахувала, що щороку витрачається 1,3 гігатонни їстівної їжі[3]. Це в свою чергу виділяє 3,3 гігатонни CO2-еквівалента. Але слід додати, що дана оцінка не включає зміну землекористування. При вираховуванні даного чиннику – оцінка буде значно більше.

Дослідження, що було проведено Our World In Data демонструє, що на харчові відходи припадає близько 6% загальних викидів в цілому у світі[4]. З них близько 2/3 припадає на харчові втрати, а решта 1/3 — на харчові відходи.

Залишки з продуктів харчування є глобальною проблемою, оскільки вони впливають не лише на навколишнє середовище, а й на суспільство та економіку. Багато ресурсів витрачаються на виробництво їжі, її транспортування, створення упаковок для неї тощо. Дані ресурси включають в себе робочу силу, використання земельних ділянок та плодючих земель,

енергію, добрива та паливо для виробництва й транспорту продовольчих продуктів. Тому, коли їжа не споживається, а даремно викидається – втрачається багато ресурсів, що були задіяні для її виробництва. Мінімізація надлишкового та даремного використання продуктів харчування допоможе знизити негативний вплив та навантаження на навколишнє середовище. Зменшення даного впливу дасть змогу суспільству більш усвідомлено та ефективно використовувати природні ресурси в сфері виробництва харчових продуктів.

Як було зазначено вище, незважаючи на кількість їжі, що виробляється в світі – існує нагальна проблема голоду в певних куточках нашого світу. Тоді, коли мільйони людей страждають від голоду, майже половина окремих категорій продовольства псуються та марнуються. Вирішення проблеми більш осмисленого використання продуктів може допомогти також у проблемі подолання голоду шляхом перенаправлення надлишкової частини їжі тим, кому це необхідно. Використання продуктів до закінчення їх терміну придатності є одним з ключових компонентів даного рішення.

За даними Ukrinform.ua, в 2021 році середньостатистичний українець витрачав на продукти харчування 41.6% від своєї заробітної плати[5]. В свою чергу, це майже на 2% більше, аніж у 2019 році. Ця тенденція очевидно є наростаючою з причини повномасштабного вторгнення росії в Україну в 2022 році. Як наслідок, подальше падіння економіки нашої країни через бойові дії, викрадення агротехніки, замінованість плодючих земель на Півдні та Сході, блокування морських шляхів експорту продовольчих товарів і розірвання логістичних ланцюжків призвели до зростання цін на продукти. Тому, можна сказати, що економічне питання в проблемі нераціонального використання продовольства зараз також посідає важливе місце. Через викидання та даремного споживання продуктів люди також витрачаюсь свої гроші. Відповідно, ефективність в управлінні запасами холодильника може оптимізувати покупки та зменшити грошові витрати.

Враховуючи наслідки в екологічній, соціальній та економічних сферах, дана проблема потребує комплексного та дієвого способу вирішення. Це вимагає залучення зацікавлених осіб на різних рівнях діяльності суспільства, бізнесу та держав. Проблема має широко висвітлюватися в соціальних мережах та медіа. Необхідне впровадження ефективної логістики, проведення оптимізаційних заходів для покращення процесів транспортування, зберігання, виробництва з використанням зеленої енергії для зменшення кількості відходів, налагоджування місць переробки відходів та повторного використання ресурсів.

Проте дуже багато залежить від культури споживання та інформованості суспільства. Необхідний також інший підхід до даних процесів, оскільки людям треба довести, що надмірне споживання харчових продуктів, які призводять до їх подальшого псування та викиду – це погано як для навколишнього середовища, так і для економічного стану кожної люди, оскільки вони викидають гроші, що були витрачені на ці продукти. Сприяння цьому культурному переходу до більш відповідального споживання харчових продуктів дасть змогу значно зменшити кількість відходів. Відповідно, це допоможе зменшити тиск на навколишнє середовище, перерозподіляти продовольство для вирішення проблеми голоду, підвищувати економічну стабільність через налагодження стійких та стабільних економічних процесів.

Покращити культуру споживання можуть допомогти інноваційні технологічні рішення, що легко інтегруються в повсякденне життя пересічного користувача та дають переваги й користь при використанні, паралельно змінюючи підхід споживання до більш відповідального.

Такі застосунки, маючи й поєднуючи зручний програмний інтерфейс та функціонал, сприяють досягненню мети щодо мінімізації харчових відходів та формуванню відповідального ставлення та підходу до споживання їжі. Як наслідок, такі інформаційні рішення через зменшення витрат на харчові продукти, можуть допомагати контролювати економіку користувача та заощаджувати гроші.

Також перевагою зменшення харчових відходів є можливість вирішення проблеми продовольчої безпеки. Це відбувається через те, що надмірне споживання продуктів забирає ті ресурси, які могли б бути використаними для годування людей, які цього потребують.

Таким чином, проблема харчових відходів в сучасному світі є глобальною. Вона має значні екологічні, соціальні та економічні наслідки. Але якщо купувати лише ту їжу, що дійсно необхідна, можна зменшити кількість відходів та зберегти ресурси, паралельно заощадивши на цьому гроші.

Відповідно веб-застосунок, що розроблюється в даній дипломній роботі, має на меті стати практичним інструментом для користувача в питанні вирішення цієї глобальної проблеми. Цей програмний продукт повинен дати змогу людям керувати запасами продуктів, що придбаваються в магазинах, надавати їм ідеї рецептів на основі випадкових інгредієнтів або тих, які вже наявні в них дома. Це допоможе змінити культуру споживання та, відповідно, зменшити кількість харчових відходів.

1.2 Змістовний опис і аналіз предметної області

Веб-застосунок, який розроблюється в даному дипломному проєкті, має спростити планування меню користувача та сформувати комплексний й більш відповідальний підхід у нього до придбання харчових продуктів. Перш за все, програмне забезпечення повинне надавати персоналізовані рецепти на основі інформації користувача про дієтичні уподобання, можливі обмеження, такі як, наприклад, алергії та на основі інгредієнтів, що наявні під рукою. Це буде заохочувати користувачів в першу чергу використовувати їжу, яка вже є у них, до того моменту, ніж вона зіпсується, тим самим зменшуючи харчові відходи та заощаджуючи гроші в цьому процесі.

В даний час через актуальність проблеми збільшення кількості харчових відходів, надзвичайно важливо мати зручну можливість для контролю запасів

їжі та намагатися максимально використовувати харчові інгредієнти, що вже наявні. Такі програми та застосунки для відстеження запасів їжі та створення рецептів стають все більш актуальними в нашому повсякденному житті.

Програмне забезпечення повинно мати сторінку інвентаря з функцією додавання до нього продуктів, що були придбані, з зазначенням ціни, терміну придатності. Також на даній сторінці можна видаляти ці продукти зі свого інвентаря або редагувати ці записи. Це дозволить користувачеві бути більш уважними до того, що він має холодильнику та морозильнику. Таким чином, маючи точний інвентар, він може уникнути надмірних закупівель продуктів та дублювання. Дана функція призводить до значної економії коштів, оскільки ми можна краще спланувати використання продуктів наперед, а й до зменшення витрат часу на дорогу в магазин. Додатково до функції додання в інвентар, користувачеві було б корисно мати статистику витрат коштів на харчові продукти за певний період часу. Веб-застосунок повинен мати для цього додаткову сторінку, на якій користувач може побачити свої витрати. Також застосунок може відстежувати термін придатності продукту та повідомляти користувача про його наближення, нагадуючи про те, що цей інгредієнт краще використати для приготування певного кулінарного блюда за рецептом.

Програмне забезпечення передбачає наявність фунціоналу підбору рецептів з наявних та випадкових інгредієнтів. Даний функціонал дає змогу як максимально використовувати за призначенням вже наявні у користувача інгредієнти, так і надихати на кулінарну творчість, пропонуючи низку варіантів страв з різних кулінарних культур. Використання наявних інгредієнтів до того, як вони зіпсуються, дасть змогу зменшувати харчові відходи. Функція підбору рецептів має враховувати різні персональні дієтичні вподобання та обмеження (наприклад, шукати лише веганські або безглютенові страви), давати змогу фільтрувати рецепти на основі конкретних потреб. Даний підхід сприяє формуванню здорових харчових звичок та допомагає користувачам досягати своїх дієтичних цілей.

Крім того, цей функціонал може дати платформу для вивчення нових смаків, різних кулінарних підходів, заохочуючи користувачів виходити із зони свого кулінарного комфорту та більше експериментувати з новими стравами, знаходячи їх рецепти.

Користувач може зареєструватися в даному програмному забезпеченні для використання функціоналу відстежування запасів продуктів або використовувати лише функцію підбору рецептів з повного переліку інгредієнтів. Після реєстрації користувач в даному застосунку може зберегти інформацію про вподобання та особливості прийому їжі. При необхідності, персональні дані користувач може змінювати.

Даний веб-застосунок може допомогти користувачам ефективно відстежувати своє придбання їжі з урахуванням коштів, що на них витрачається та ставати більш свідомими в підході до споживання харчових продуктів. Програма надає можливість аналізувати витрати на продукти, підбирати рецепти на основі вже наявних інгредієнтів та повідомляти користувача про наближення закінчення терміну придатності певних продуктів. Всі функції даного застосунку сприяють формуванню правильної культури споживання та зменшенню кількості харчових відходів шляхом рекомендації щодо використання їжі перед її псуванням, надаючи можливість користувачеві знайти рецепт з урахуванням персональних вподобань. Більш того, цей застосунок може бути корисним для середнього бізнесу в гастроіндустрії у вигляді колаборації магазин-ресторан. Мережі зможуть не тільки відслідковувати терміни придатності товарів, а й пропонувати ресторану-партнеру різноманітні страви, що можна додати до меню із наявних надлишкових продуктів.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відомі на сьогодні алгоритмічні та технічні рішення у даній області та вже наявні програмні продукти, що допоможуть у визначенні

переваг і реалізації веб-застосунку для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Сьогодні в ІТ індустрії виділяють 4 найпоширеніших види архітектури програмного забезпечення: клієнт-серверну, багатошарову, сервіс-орієнтовану та мікро-сервісну. Кожний з цих видів має свої переваги та недоліки й використовується як індивідуально, так і комплексно в залежності від задачі. Багатошарова архітектура є досить простою в реалізації, має декілька рівнів абстракції, ізолює шари один від одного. Однак програмне забезпечення, створене на основі такого шаблону, матиме монолітну структуру та не передбачає легкого масштабування. Клієнт-серверна архітектура передбачає існування декількох рівнів взаємодії та комунікація між ними відбувається за допомогою Request Response шаблону. Така архітектура дає змогу масштабувати застосунок як горизонтально, так і вертикально. Сервісно-орієнтована архітектура складається з компонентів, що взаємодіють між собою за допомогою сервісів. Коли запит від клієнта надходить до головного сервісу – то він делегує його виконання відповідальному сервісу. Мікро-сервісна архітектура передбачає існування та взаємодію між багатьма атомарними сервісами, що мають незалежну логіку. Такий підхід пропонує високу гнучкість та масштабованість й підвищує модульність. Але для коректної роботи як сервісно-орієнтованої, так і мікро-сервісної архітектури необхідна високопродуктивна інфраструктура для підтримки роботи й швидкодії.

Для реалізація веб-застосунку за темою відстеження запасів продуктів та підбору кулінарних рецептів було обрано клієнт-серверну архітектуру з наявністю окремого рівня клієнтської частини та серверної частини з базою

даних, відповідно. Серверна частина спроектована з використанням шаблону Onion-Architecture (архітектура луковиці) та цей шаблон використовує певні концепції багатошарової архітектури. Клієнтська частина має шар представлення та шар бізнес-логіки, оскільки для функціоналу, що відповідає за підбор рецептів на основі вподобань користувача, використовує сторонні API для отримання великої кількості рецептів й відповідає за їх фільтрацію з урахуванням всіх параметрів та вподобань.

Веб-застосунок за цією предметною областю може використовувати візуалізацію на стороні сервера «Server Side Rendering» (SSR) або односторінковий інтерфейс «Single Page Application» (SPA). Головною перевагою візуалізації на стороні сервера є швидкодія та цей підхід є чудовим вибором для чутливих до часу відповіді застосунків, що покладаються на велику кількість взаємодії з користувачем. SSR дозволяє створювати динамічний персоналізований контент швидше за SPA при використанні та правильному налаштуванні Service Worker. Використання SSR для отримання продуктивних результатів потребує швидку та стійку інфраструктуру, що може обробляти велику кількість запитів до сервера з урахуванням легкої масштабованості додатку, й відповідно, зростання їх кількості. Проте для розробки даного програмного забезпечення обрано технологію SPA, оскільки кількість взаємодії з застосунком на стороні клієнта є невеликою та навантаження є помірним. Для розробки веб-застосунку на основі технології SPA можливо використання низки сучасних веб-фрейморків та бібліотек, що мають актуальну підтримку та забезпечують контроль над архітектурою застосунку.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Для розробки клієнтської частини застосунку, з урахуванням обраної технології SPA, необхідно обрати мову програмування та фреймворк, на якій вона буде реалізована. В наш час найбільш популярними технологіями є

Angular[6], React та Vue.js. Дані рішення є фреймворками та бібліотеками мови програмування JavaScript та її надмножини - TypeScript. Кожна технологія має свої переваги та сильні сторони. React є добре відомою бібліотекою через свою гнучкість, наявність оптимізації віртуального DOM, об'ємною екосистемою повторно використовуваних та додаткових компонентів. Vue.js пропонує простоту в розробці, швидкодію, легкість у навчанні та ознайомленні з внутрішніми компонентами. Однак для написання клієнтської частини застосунку даного проєкту було обрано фреймворк Angular, оскільки це повний фреймворк, що надає весь повний набір функцій після його завантаження, включаючи інші вбудовані інструменти та бібліотеки. Рішення пропонує використання чітко визначеної, структурованої архітектури, яка заохочує дотримування модульності створених компонентів. Angular розроблений за допомогою TypeScript, статично типізованої надмножини JavaScript. TS пропонує розширені актуальні інструменти, строгу типізацію та покращену підтримку коду. Також Angular має актуальну та добре підтримувану документацію й активну спільноту розробників, що надають численні ресурси, навчальні посібники.

Вибір мови програмування та додаткових програмних засобів для розробки серверної частини застосунку складався з мови програмування C# та фреймворку .NET[7], Java і фреймворку Spring та платформи Node.js з використанням мови TypeScript й фреймворку NestJS. .NET є комплексним фреймворком з вже вбудованою великою кількістю додаткових класів й пакетів, що дозволяють легко створювати масштабовані додатки в екосистемі Microsoft, при наявності досвіду роботи з мовою програмування C#. При використанні .NET Core додаток є крос-платформним й .NET має безкоштовну інтеграцію з іншими технологіями Microsoft, наприклад, з хмарним середовищем Azure.

В свою чергу Node.js та фреймворк NestJS використовує мову програмування TypeScript, що може полегшити написання як серверної, так і клієнтської частин через використання спільної мови програмування. Nest

надає детальну актуальну документацію, чітко визначену архітектуру та можливість підключення до всіх поширених сховищ даних: PostgreSQL, Microsoft SQL Server, MongoDB. В додаток до цього, платформа Node.js та менеджер пакетів NPM дають змогу використовувати додаткові бібліотеки та модулі від інших розробників.

Java має стабільну екосистему, що перевірена роками та є крос-платформенною за замовчуванням. Фреймворк Spring пропонує найкращу з перелічених рішень швидкодію та може забезпечувати високу продуктивність роботи системи під великим навантаженням. Швидкодія Spring хоч і виділяється на фоні всіх фреймворків, проте для розробки серверної частини веб-застосунку для дипломного проєкту було обрано .NET через його інтеграцію з системою Windows й іншими продуктами Microsoft. .NET посідає друге місце в порівняннях за швидкістю роботи після Spring та на вибір також вплинув чинник кращого знання мови програмування C#, аніж Java.

Як додаткові бібліотеки для створення адаптивного веб-дизайну були обрані Bootstrap[8] та Angular Material[9]. Вони є найбільш розповсюдженими рішеннями в індустрії та дані технології легко інтегруються з Angular, це забезпечує більш ефективну розробку. Надбудова над мовою CSS – SCSS була обрана для написання правил стилів компонентів.

Для Back-End додатково був обраний Entity Framework[10] – фреймворк об'єктно-реляційного відображення (ORM) для .NET, який спрощує доступ та взаємодію з базами даних. Як сховище даних було обрано Microsoft SQL Server через використання Windows платформи для розробки та платформи .NET.

Як інтегроване середовище розробки клієнтської частини використовувався WebStorm від компанії JetBrains через розширені можливості для веб-розробки. Дане IDE пропонує широкий спектр функцій, розроблених спеціально для розробки веб-додатків, а саме підтримку та інтелектуальне завершення та статичний аналізатор коду HTML й SCSS, повну підтримку TypeScript та Angular проектів з урахуванням відлагодження, тестів. Для серверної частини було обрано Visual Studio – інтегроване середовище

розробки (IDE) від компанії Microsoft через підтримку .NET та мови програмування C# за замовчуванням і підключення до інших продуктів Microsoft: Microsoft SQL Server, Azure тощо. Обрані IDE надають широкий функціонал при розробці ПЗ, в тому числі для його тестування й відлагодження.

Отже, для дипломного проєктування та розробки програмного забезпечення було обрано мови програмування TypeScript та C#, базу даних SQL Server. Для клієнтської частини був обраний фреймворк мови TS – Angular, для серверної – платформа .NET з використанням Entity Framework. У якості IDE – Webstorm та Visual Studio. Для розгортання програмного забезпечення було взято Microsoft Azure через використання інших продуктів Microsoft.

1.3.3 Аналіз відомих програмних продуктів

В рамках дослідження предметної області в мережі було знайдено декілька аналогів веб-застосунку, що розроблюється. Вони включають в себе інші веб-застосунки та мобільні додатки. Серед них можна виділити 3 основних аналоги: "MealBoard"[11], "Cookpad"[12] та "Fridge Pal".

"MealBoard" – це мобільний додаток, який доступний для пристроїв Apple, має веб-версію, пропонує функції управління запасами продуктів, планування меню харчування на певний період часу та підбору рецептів. Проте, для використання рецептів, їх потрібно спочатку імпортувати до застосунку, тобто додаток не передбачає існування внутрішньої бази рецептів та, незважаючи на присутність модуля з додавання продуктів у свій інвентар, функція аналітики або збору статистики відсутня.

"Cookpad" – мобільний додаток для пристроїв на базі операційних систем iOS та Android. Він пропонує функції підбору рецептів за різними інгредієнтами та планування меню харчування. Даний додаток також дозволяє користувачам шукати та зберігати рецепти від інших клієнтів, та включає в

додатковий функціонал можливість формування списку продуктів. Однак, Cookpad не пропонує функцій управління запасами продуктів, збору статистики та більш деталізованого підбору рецептів на основі кулінарних вподобань користувача.

"Fridge Pal" - це мобільний додаток, який доступний як для пристроїв на iOS, так і для Android. Він пропонує функції відстеження запасів продуктів з відображенням термінів їх придатності. Цей додаток може надсилати користувачам повідомлення тоді, коли термін придатності продуктів починає закінчуватися. Проте цей додаток не пропонує функцію статистки або підбору рецептів.

MealBoard і Cookpad надають функції пошуку рецептів і планування харчування, тоді як Fridge Pal пропонує функцію управління запасами і відстеження термінів придатності. Cookpad, в свою чергу, такий функціонал не пропонує та Fridge Pal не пропонує функціоналу пошуку рецептів.

Веб-застосунок, що розроблюється в рамках дипломного проєкту, має назву називається FridgeHelper. Додаток має свої переваги над розглянутими аналогами. Наприклад, дозволяючи користувачам вводити ціни на придбані продукти, він збирає та формує статистику грошових витрат на харчові продукти для подальшого перегляду. FridgeHelper надає уявлення про загальні витрати на харчові продукти клієнта. Дана функція має в собі економічну ефективність та можливість додаткового управління бюджетом. Для споживання продуктів до закінчення їх терміну придатності з урахуванням специфічних вподобань користувача, додаток дозволяє клієнтам встановлювати та використовувати за замовчуванням свої кулінарні смаки та можливі дієтичні обмеження задля створення персоналізованих рекомендацій рецептів на основі даних параметрів. Така функція значно покращує користувацький досвід та сприяє кулінарним експериментам.

Завдяки наявності функції збору статистики та можливостям персоналізації, FridgeHelper вирізняється серед існуючих аналогів у кращу сторону. Через поєднання широкого спектру функціональних можливостей та

зручного дизайну, додаток є цінним інструментом для середньостатистичного користувача, який дозволяє їм керувати запасами продуктів, використовувати їжу до закінчення термінів її придатності. В результаті, кількість харчових відходів зменшується та відбувається економія коштів, що може витрачатися на продукти.

Для порівняння дипломної роботи з головними аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Веб-застосунок для відстеження запасів продуктів та підбору рецептів з них – FridgeHelper (Дипломний проєкт)	MealBoard	CookPad	Пояснення
Авторизація	+	+	+	Можливість реєстрації в застосунку для збереження інформації та доступу до повного функціоналу.
Пошук рецептів за інгредієнтами	+	+	+	Можливість пошуку рецептів на основі

				наявних інгредієнтів.
Пошук рецептів за кулінарними вподобаннями	+	+	-	Можливість пошуку рецептів на основі кулінарних вподобань.
Відстеження запасів продуктів	+	+	-	Можливість відстежування інвентарю продуктів, додавати в нього продукти, видаляти їх.
Збір та демонстрація статистики витрати коштів	+	-	-	Можливість переглядати статистику про кількість грошей, що були витрачені на їжу.
Мобільна версія додатку	-	+	+	Можливість користування додатком на мобільному телефоні.

1.4 Аналіз вимог до програмного забезпечення

Головними функціями програмного забезпечення є відстеження запасів продуктів, додавання, редагування продуктів в інвентарі, збір статистики та пошук рецептів на основі як випадкових інгредієнтів, так і на основі наявних з урахуванням кулінарних вподобань і можливих дієтичних обмежень. Більше функцій даного застосунку можна побачити на рисунку 1.1 – діаграмі варіантів використання.

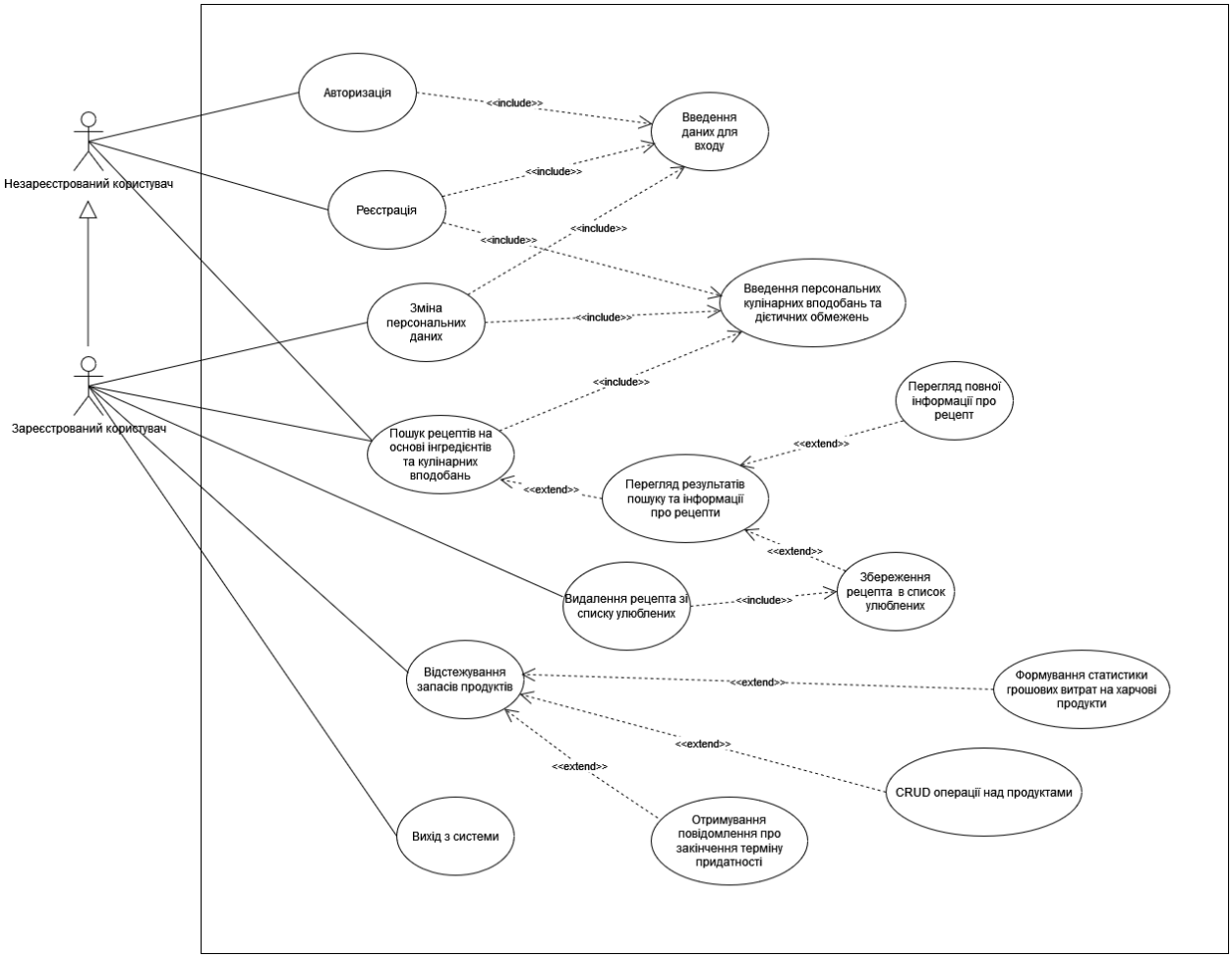


Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 – 1.15 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Створення облікового запису користувача в веб-застосунку для відстежування запасів продуктів та підбору кулінарних рецептів із наявних інгредієнтів.
Actors	Гість (незареєстрований користувач)
Trigger	Анонімний користувач бажає зареєструватися у веб-застосунку для того, щоб отримати доступ до повного функціоналу.
Pre-conditions	Користувач не має наявного облікового запису в даному застосунку.
Flow of Events	Анонімний користувач відкриває діалогове вікно авторизації та потім – діалогове вікно реєстрації через перехід за посиланням. У поля вводу він вписує необхідні персональні дані: ім’я, адресу електронної пошти пароль та підтвердження паролю. Після перевірки введених даних на коректність стає активною кнопка “Sign up”. Користувач натискає кнопку для завершення реєстрації.
Extension	У випадку, якщо були введені дані в неправильному форматі - кнопка реєстрації залишається неактивною. При існуванні іншого користувача з такою же електронною поштою в системі – повідомляється про таку помилку.
Post-Condition	Створення сторінки користувача, відкриття діалогового вікна авторизації.

Таблиця 1.3 - Варіант використання UC-2

Use case name	Авторизація
Use case ID	UC-02
Goals	Провести авторизацію користувача у веб-застосунку, якщо в нього вже є створений обліковий запис в системі.
Actors	Гість (анонімний користувач)
Trigger	Користувач бажає увійти до свого облікового запису у веб-застосунку.
Pre-conditions	Користувач має обліковий запис в системі.
Flow of Events	Анонімний користувач знаходиться на головній сторінці застосунку. Він натискає на посилання у навігаційній панелі та після натискання відкривається діалогове вікно з формою для заповнення даних входу у веб-застосунок. У відповідні поля вводу користувач вносить свою електронну пошту та пароль. Після введення цих даних та перевірки їх на коректність стає активною кнопка “Sign in”.
Extension	У випадку введення пустих або некоректних даних кнопка залишається неактивною. При введенні неправильного паролю від облікового запису або введення неіснуючої електронної пошти – система повідомляє про це користувача шляхом зображення на інтерфейсі форми інформації про помилку.
Post-Condition	Користувач авторизується у веб-застосунку та перенаправляється на головну сторінку.

Таблиця 1.4 - Варіант використання UC-3

Use case name	Підбір рецептів кулінарних блюд із інгредієнтів
Use case ID	UC-03
Goals	Підібрати рецепти за заданими інгредієнтами та кулінарними вподобаннями
Actors	Анонімний користувач, Зареєстрований користувач
Trigger	Користувач бажає отримати рецепти для приготування кулінарних блюд із наявних у нього або випадкових інгредієнтів з урахуванням його кулінарних вподобань.
Pre-conditions	-
Flow of Events	Користувач знаходиться на сторінці підбору рецептів кулінарних блюд, обирає з низки інгредієнтів ті, що наявні у нього або ті, що мають бути присутні в рецепті. Також користувач може обирати певні фільтри для урахування його кулінарних вподобань або дієтичних обмежень. Форма має поля вибору можливих алергій, регіональної приналежності страви, вид страви.. Після цього користувач натискає на кнопку «Знайти рецепти» й чекає завершення пошуку.
Extension	Користувач не може почати пошук рецептів, якщо немає обраних інгредієнтів.
Post-Condition	Користувач бачить на сторінці результат свого пошуку. При наявності рецептів показаний перелік карток з інформацією про рецепти. В іншому випадку – виведено повідомлення про не існування рецептів за заданими параметрами.

Таблиця 1.5 - Варіант використання UC-4

Use case name	Вибір інгредієнтів та фільтрів, на яких засновувати пошук рецептів
Use case ID	UC-04
Goals	Обрати інгредієнти та вподобання зі списку для пошуку кулінарних рецептів на їх основі.
Actors	Анонімний користувач, Зареєстрований користувач
Trigger	Користувач бажає обрати інгредієнти та додаткові фільтри, на яких засновувати пошук рецептів.
Pre-conditions	Користувач знаходиться на сторінці пошуку рецептів та бажає обрати критерії.
Flow of Events	Користувач знаходиться на сторінці пошуку рецептів кулінарних блюд. З переліку всіх інгредієнтів в базі даних користувач може обрати будь-яку кількість продуктів з різних категорій, знайти продукт за назвою. Також користувач може додати до пошуку параметри можливих дієтичних обмежень, виду страв за регіональною приналежністю, дієтичний вид, вид за основною категорією інгредієнтів у страві.
Extension	-
Post-Condition	Обрані критерії з’являються в переліку для пошуку рецептів.

Таблиця 1.6 - Варіант використання UC-5

Use case name	Перегляд результату пошуку
Use case ID	UC-05
Goals	Переглянути результат пошуку кулінарних рецептів за обраними критеріями.
Actors	Анонімний користувач, Зареєстрований користувач
Trigger	Користувач отримав результат пошуку, список з рецептів з’явився на сторінці.
Pre-conditions	Користувач обрав критерії для пошуку рецептів та зробив запит.
Flow of Events	Користувач знаходиться на сторінці результату пошуку кулінарних рецептів. На ній представлений список рецептів з коротким описом та наступною інформацією: скільки з інгредієнтів наявно у користувача, кількість калорій, час приготування.
Extension	-
Post-Condition	Користувач може натиснути на картку рецепту для відкриття вікна з більш детальним описом.

Таблиця 1.7 - Варіант використання UC-6

Use case name	Перегляд повної інформації про рецепт
Use case ID	UC-06
Goals	Переглянути повну інформацію про кулінарний рецепт, що був в результуючому списку.
Actors	Анонімний користувач, Зареєстрований користувач
Trigger	Користувач отримав результат пошуку, список з рецептів з’явився на сторінці та бажає переглянути більш детальну інформацію про конкретний рецепт.
Pre-conditions	Користувач отримав результат пошуку, натискає на картку рецепту.
Flow of Events	Користувач знаходиться на сторінці результату пошуку кулінарних рецептів. Натискає на картку певного рецепту. Після кліку з’являється діалогове вікно з детальним описом рецепту, переліком інгредієнтів, посилання на першоджерело рецепту, короткий опис.
Extension	-
Post-Condition	Користувач може закрити діалогове вікно, перейти на сторонній сайт для перегляду інструкції приготування чи додати рецепт в улюблені, якщо користувач є зареєстрованим.

Таблиця 1.8 - Варіант використання UC-7

Use case name	Зберегти рецепт як улюблений
Use case ID	UC-07
Goals	Дати можливість користувачеві зберігати рецепт для того, щоб була можливість потім до нього повернутися.
Actors	Зареєстрований користувач
Trigger	Користувач бажає зберегти рецепт в список улюблених.
Pre-conditions	Користувач є авторизованим, користувач знаходиться на сторінці рецепту або сторінці результату пошуку.
Flow of Events	Користувач знаходиться на сторінці повною інформації про рецепт. Він натискає на відповідну кнопку. Рецепт додається до списку збережених.
Extension	-
Post-Condition	Рецепт є збереженим. Список улюблених рецептів можна побачити в особистому кабінеті.

Таблиця 1.9 - Варіант використання UC-8

Use case name	Видалити рецепт зі списку улюблених
Use case ID	UC-08
Goals	Дати можливість користувачеві видаляти рецепт зі списку улюблених
Actors	Зареєстрований користувач
Trigger	Користувач бажає видалити рецепт зі списку улюблених через неактуальність.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач знаходиться на сторінці особистого кабінету та бачить список улюблених рецептів. Він натискає на кнопку видалення рецепту зі списку. Рецепт видаляється.
Extension	-
Post-Condition	Рецепт є видаленим зі списку улюблених рецептів користувача.

Таблиця 1.10 - Варіант використання UC-9

Use case name	Відстеження запасів продуктів
Use case ID	UC-09
Goals	Користувач має змогу відстежувати свій запас продуктів
Actors	Зареєстрований користувач
Trigger	Користувач хоче знати про свій запас продуктів: які продукти наявні та який в них термін придатності.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач находиться на головній сторінці. Він переходить на сторінку інвентаря, натискаючи на значок на навігаційній панелі.
Extension	-
Post-Condition	Користувач бачить свій інвентар з інформацією по кожному продукту.

Таблиця 1.11 - Варіант використання UC-10

Use case name	Управління запасами продуктів (CRUD операція)
Use case ID	UC-10
Goals	Користувач може додавати, змінювати та видаляти продукти зі свого інвентаря.
Actors	Зареєстрований користувач
Trigger	Користувач хоче додати або змінити інвентар своїх продуктів.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач находиться сторінці інвентаря. Він натискає на кнопку додавання, з'являється форма, в яку вписується інформація про інгредієнт, що додається, його ціну, термін придатності. Натискається кнопка додати продукт. Цей продукт з'являється в переліку, всіх продуктів на сторінці. При необхідності, користувач може змінити інформацію про доданий продукти, чи видалити цей продукт зі списку.
Extension	Користувач не може додати продукт, якщо всі поля у формі не є заповненими.
Post-Condition	Користувач додає, змінює та видаляє продукти зі свого інвентаря.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Отримання повідомлення про закінчення терміну придатності інгредієнта.
Use case ID	UC-11
Goals	Користувачу необхідно знати про наближення закінчення терміну придатності продукту.
Actors	Зареєстрований користувач
Trigger	Користувач хоче отримувати інформаційне повідомлення при закінченні терміну придатності продукту, що знаходиться у нього в інвентарі.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач находиться на сторінці інвентаря. Ті продукти, в яких термін придатності починає закінчуватися, підсвічуються помаранчевим кольором, якщо до завершення залишилося від 7 до 4 днів та червоним, якщо залишилося менше 4 днів.
Extension	-
Post-Condition	Наближення закінчення терміну придатності продуктів продемонстровано у вигляді змін кольору в інтерфейсі.

Таблиця 1.13 - Варіант використання UC-12

Use case name	Перегляд статистики грошових витрат на харчові продукти за певний період
Use case ID	UC-12
Goals	Користувач може переглядати статистику своїх грошових витрат за певний час для керування особистої економіки.
Actors	Зареєстрований користувач
Trigger	Користувач хоче отримувати повну статистику щодо кількості грошей, які він витрачає на їжу, таким чином більш усвідомлено керувати своїми витратами.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач находиться на головній сторінці застосунку та переходить на сторінку аналітики. Там користувач обирає період, за який він хоче переглянути статистику та бачить графічне представлення даної інформації.
Extension	-
Post-Condition	Користувач може змінити період та отримати додаткову інформацію або перейти до іншої сторінки.

Таблиця 1.14 - Варіант використання UC-13

Use case name	Зміна персональних даних користувача
Use case ID	UC-13
Goals	Дати змогу користувачеві змінювати певні персональні дані.
Actors	Зареєстрований користувач
Trigger	Користувач хоче змінити свої вподобання в їжі, що обираються автоматично.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач находиться на сторінці особистого кабінету. Користувач переходить на сторінку редагування інформації про себе та змінює кулінарні вподобання при такої необхідності. Далі користувач натискає на кнопку «Зберегти зміни».
Extension	-
Post-Condition	Після натискання на кнопку зміни в профілі зберігаються, користувач перенаправляється на сторінку особистого кабінету.

Таблиця 1.15 - Варіант використання UC-14

Use case name	Вихід з системи
Use case ID	UC-14
Goals	Завершити сесію користувача в системі
Actors	Зареєстрований користувач
Trigger	Користувач бажає вийти з системи.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач натискає на відповідний значок на навігаційній панелі.
Extension	-
Post-Condition	Користувач перенаправляється на головну сторінку, в якості анонімного користувача.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На рисунку 1.2 наведено загальну модель вимог, а в таблицях 1.16 – 1.34 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.3.

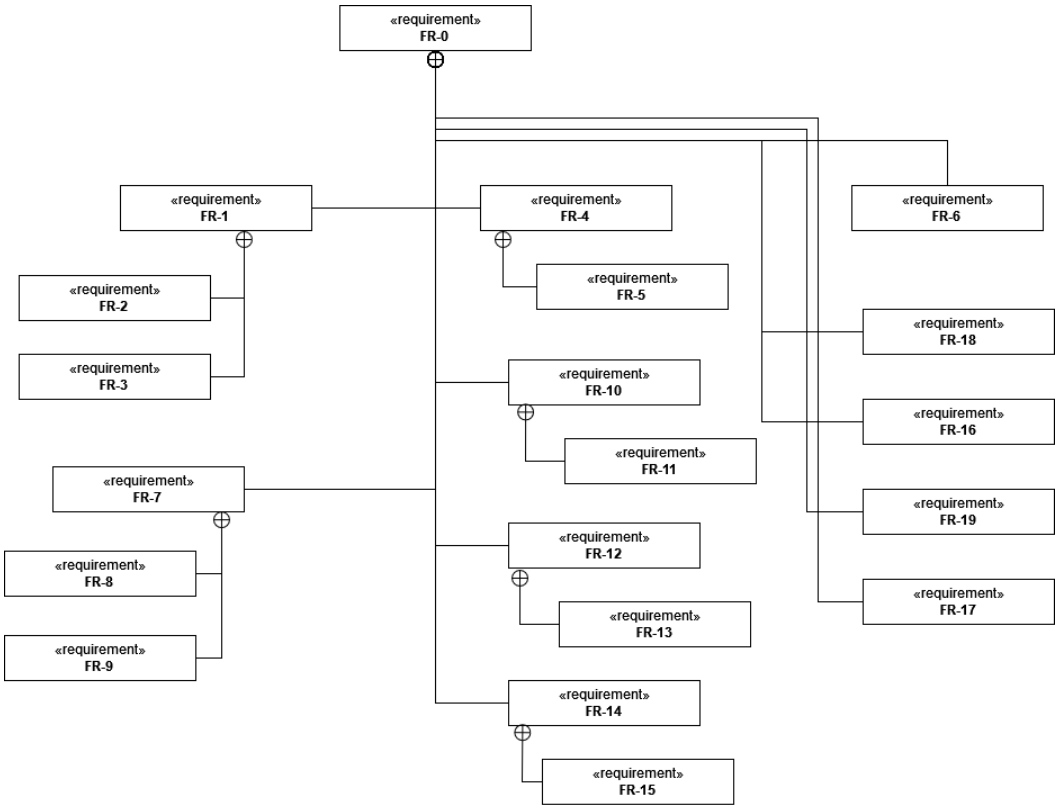


Рисунок 1.2 – Модель вимог у загальному вигляді

Таблиця 1.16 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Система повинна надавати можливість реєстрації користувача за допомогою введення пошти, паролю у відповідну форму.

Таблиця 1.17 – Функціональна вимога FR-2

Назва	Форма для реєстрації користувача має поля «Електронна пошта», «пароль».
Опис	Форма, яка з’являється на сторінці реєстрації користувача повинна мати наступні поля вводу «Електронна пошта», «пароль», «підтвердження пароля», «ім’я». Мінімальна кількість символів для вводу пароля – 8. Пароль має містити латинські букви, цифри та спеціальні знаки. Також форма має кнопку «Зареєструватися».

Таблиця 1.18 – Функціональна вимога FR-3

Назва	Вибір кулінарних вподобань користувача
Опис	Система повинна надавати можливість користувачеві обирати персональні кулінарні вподобання та дієтичні обмеження після реєстрації в застосунку. Це відбувається шляхом редагування інформації на сторінці особистого профіля.

Таблиця 1.19 – Функціональна вимога FR-4

Назва	Авторизація користувача
Опис	Система повинна надавати можливість авторизації користувачеві шляхом введення електронної пошти та паролю.

Таблиця 1.20 – Функціональна вимога FR-5

Назва	Форма для авторизації користувача має поля «Електронна пошта», «пароль».
Опис	Форма, яка з’являється на сторінці реєстрації користувача повинна мати наступні поля вводу «Електронна пошта», «пароль». Також форма має кнопку «Авторизуватися».

Таблиця 1.21 – Функціональна вимога FR-6

Назва	Повідомлення про неправильний ввід даних
Опис	Якщо користувач ввів неправильні дані при авторизації або реєстрації (неправильний формат даних або існування вже зареєстрованого запису при намаганні створити новий), то система має надавати графічне повідомлення про помилку.

Таблиця 1.22 – Функціональна вимога FR-7

Назва	Пошук рецептів за обраними інгредієнтами та фільтрами
Опис	Система повинна надавати користувачеві можливість пошуку рецептів за введеними інгредієнтами та фільтрами на основі його персональних вподобань. Також на сторінці пошуку рецептів має знаходитися кнопка «Почати пошук», яка відповідає за початок пошуку рецептів за введеними параметрами.

Таблиця 1.23 – Функціональна вимога FR-8

Назва	Вибір інгредієнтів для пошуку
Опис	На сторінці пошуку рецептів має бути представлений перелік інгредієнтів, з якого користувач може обрати необмежену кількість позицій. Інгредієнти розбиті на категорії.

Таблиця 1.24 – Функціональна вимога FR-9

Назва	Вибір додаткових параметрів для пошуку
Опис	На сторінці пошуку рецептів має бути представлений перелік додаткових параметрів пошуку. Це вподобання та можливі дієтичні обмеження: тип блюда, тип кухні, стиль блюда.

Таблиця 1.25 – Функціональна вимога FR-10

Назва	Результат пошуку рецептів за параметрами
Опис	Система повинна надавати користувачеві результат пошуку рецептів за введеними параметрами у вигляді переліку карток рецептів з коротким описом про них: час приготування, кількість калорій, скільки інгредієнтів наявно у користувача.

Таблиця 1.26 – Функціональна вимога FR-11

Назва	Перегляд повного опису рецепту
Опис	Система повинна надавати користувачеві можливість переглядати повний опис рецепту шляхом натискання на картку рецепту з переліку. У діалоговому вікні має бути повна інформація: всі інгредієнти, список з їх кількістю, ключові слова для цього рецепту.

Таблиця 1.27 – Функціональна вимога FR-12

Назва	Збереження рецепту в список улюблених
Опис	Система повинна надавати можливість користувачеві зберігати рецепти у список улюблених для їх подальшого використання. Це має відбувається шляхом натискання користувачем на відповідну кнопку та сповіщенням про успішне додавання рецепту до списку.

Таблиця 1.28 – Функціональна вимога FR-13

Назва	Управління списком збережених рецептів
Опис	Система повинна надавати можливість користувачеві управляти списком збережених рецептів: видаляти їх при необхідності або повертатися до сторінки повного описку рецепту.

Таблиця 1.29 – Функціональна вимога FR-14

Назва	Відстеження запасів продуктів
Опис	Система повинна надавати користувачеві можливість відстежувати свій запас продуктів через маніпуляції з даними, що в ньому знаходяться та збором статистики грошових витрат на харчові продукти.

Таблиця 1.30 – Функціональна вимога FR-15

Назва	Додавання, видалення та редагування продуктів
Опис	Система повинна надавати користувачеві можливість додавати продукти в свій інвентар, видаляти та редагувати їх при такій необхідності.

Таблиця 1.31 – Функціональна вимога FR-16

Назва	Повідомлення користувача про закінчення терміну придатності продукту
Опис	Система має сповіщати користувача про закінчення терміну придатності певного продукту шляхом зміни кольору картки продукту у веб-інтерфейсі застосунку на сторінці інвентаря.

Таблиця 1.32 – Функціональна вимога FR-17

Назва	Збір та демонстрація статистики грошових витрат на продовольство
Опис	Система повинна надавати можливість користувачеві переглядати статистику своїх грошових витрат на продукти за певний період часу.

Таблиця 1.33 – Функціональна вимога FR-18

Назва	Зміна персональних даних користувача
Опис	Система має надавати можливість користувачеві змінювати персональні дані в особистому кабінети: ім’я, кулінарні вподобання, використовування кулінарних вподобань за замовчуванням.

Таблиця 1.34 – Функціональна вимога FR-19

Назва	Вихід з системи
Опис	Система має надавати можливість користувачеві виходити з веб-застосунку, змінюючи його статус на неавторизований. Це відбувається після натискання користувачем на відповідну кнопку на навігаційній панелі.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19
UC-01	X	X	X			X													
UC-02				X	X	X													
UC-03							X	X	X										
UC-04								X	X										
UC-05										X	X								
UC-06											X								
UC-07												X							
UC-08													X						
UC-09														X	X				
UC-10															X				
UC-11																X			
UC-12																	X		
UC-13																		X	
UC-14																			X

Рисунок 1.3 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення має бути:

- зрозумілим для користувача;
- легким в використанні;
- підтримуватись всіма сучасними браузерами;
- мати адаптивний дизайн;
- привабливим естетично, мати приємний мінімалістичний дизайн.

1.5 Постановка задачі

Веб-застосунок, що розроблюється в рамках дипломного проєктування призначений для покращення культури споживання харчових продуктів середньостатистичного користувача, мета розробки – посприяти формуванню більш усвідомленої культури споживання та через це зменшити кількість харчових відходів через споживання продуктів до закінчення їх терміну придатності.

Процес розробки веб-додатку для відстеження запасів харчових продуктів, збору статистики та підбору рецептів на основі наявних інгредієнтів та персональних кулінарних вподобань ставить перед собою розв’язання наступних задач:

– надання можливості реєстрації та авторизації користувачу;

– реалізація можливості додавання, видалення та редагування продуктів, що знаходяться у запасі користувача;

– передбачення зміни стану продукту при наближенні закінчення його терміну придатності та повідомляти про це користувача шляхом виділення даної інформації на користувацькому інтерфейсі;

– створення системи пошуку рецептів на основі випадкових та наявних інгредієнтів з урахуванням кулінарних вподобань користувача;

– надання можливості зберігання рецептів у список улюблених;

– формування та демонстрація зібраної статистики придбаних продуктів за певний проміжок часу.

Висновки до розділу

В першому розділі дипломної роботи було описано загальні положення щодо обраної теми дипломного проєктування, викладено актуальність даної теми та наголошено на ключових проблемах чому важливо почати вирішувати

проблему збільшення кількості харчових відходів на інформаційно-технічному полі.

Розглянуто та описано предметну область проєкту, виділено основний стек технологій для розробки програмного забезпечення: Angular та мову програмування TypeScript для реалізації клієнтської частини застосунку, .NET і C# – для серверної частини. Як сховище даних було обрано Microsoft SQL Server. Проаналізовано ринок IT індустрії та знайдені аналоги програмного забезпечення, які надають схожий функціонал. Проведено порівняння веб-застосунку, що розроблюється з його конкурентами та в результаті виділені сильні сторону FridgeHelper – збір та формування статистики витрат власних коштів користувача на харчові продукти. Представлена UML діаграма варіантів використання та їх табличний опис. Було розроблено та описано основні функціональні вимоги, що висуваються для реалізації цього продукту. На основі них було сформовано модель функціональних вимог. Вищезазначені варіанти використання та функціональні вимоги було об’єднано за допомогою матриці трасування. Також були наведені нефункціональні вимоги для розробки цього веб-застосунку та описана постановка задачі. В ній розглянуто для чого ця розробка призначена та яку мету ставить перед собою ця розробка.

Метою розробки даного веб-застосунку є сприяння формуванню більш усвідомленої культури споживання у середньостатистичного користувача задля зменшення кількості харчових відходів через вчасне споживання продуктів.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

В даному розділі дипломного проєктування опишемо бізнес-процеси за допомогою BPMN – нотації моделювання бізнес-процесів. В роботі застосунку можна виділити п’ять основних бізнес процесів: реєстрація у веб-застосунку, авторизація, пошук рецептів за інгредієнтами та параметрами, збереження знайдених рецептів в список улюблених, додавання продуктів в інвентар та отримання повідомлення про закінчення терміну придатності продукту, збір статистики грошових витрат на харчові продукти.

Для опису бізнес процесів програмного забезпечення використовуються BPMN моделі. Схема бізнес-процесу реєстрації у веб-застосунку продемонстрована на рисунку 2.1.

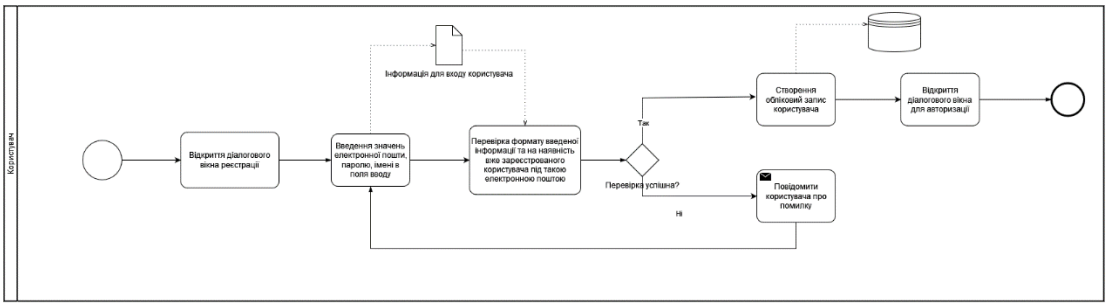


Рисунок 2.1 – Схема бізнес процесу реєстрації у веб-застосунку

Опис послідовності реєстрації користувача у веб-застосунку:

– знаходячись на головній сторінці застосунку, користувач відкриває діалогове вікно реєстрації з формою для вводу інформації;

– у поля вводу вводить необхідні дані: ім’я, електронну пошту пароль;

– відбувається перевірка коректності вводу даних. Якщо помилки немає, кнопка «Зареєструватися» стає активною, користувач натискає на неї та дані відправляються в базу даних. Якщо помилка є – то відображається повідомлення про це, кнопка залишається неактивною;

– після успішного створення облікового запису в системі – користувач перенаправляється на сторінку авторизації.

Схема бізнес-процесу авторизації у веб-застосунку зображена на рисунку 2.2.

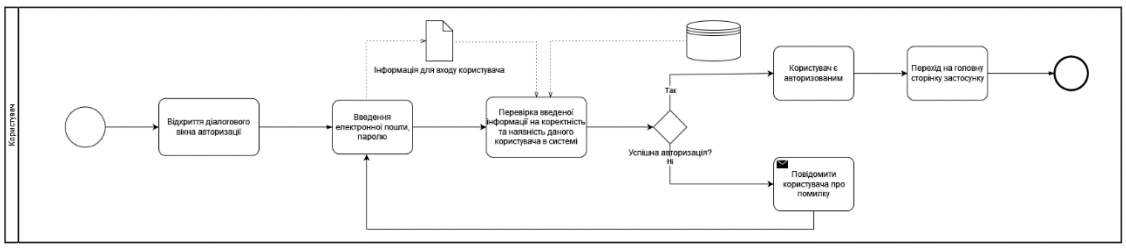


Рисунок 2.2 – Схема бізнес процесу авторизації у веб-застосунку

Опис послідовності авторизації користувача у веб-застосунку:

– користувач відкриває діалогове вікно авторизації з формою для введення інформації;

– у поля вводу вводить необхідні дані: електронну пошту та пароль;

– введена інформація перевіряється на коректність та наявність користувача в системі;

– якщо помилки немає – то користувач успішно авторизується. В іншому випадку – користувачеві повідомляється про наявність помилки;

– якщо авторизація успішна – користувач перенаправляється на головну сторінку застосунку.

Схема бізнес-процесу пошуку рецептів та перегляду результатів у веб-застосунку зображена на рисунку 2.3.

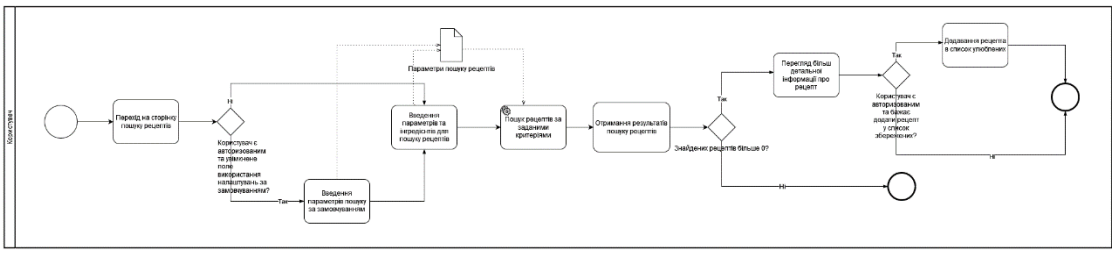


Рисунок 2.3 – Схема бізнес процесу пошуку рецептів та перегляду результатів

Опис послідовності пошуку рецептів та перегляду результатів у веб-застосунку:

– користувач переходить на сторінку пошуку рецептів;

– якщо користувач є авторизованим у системі та використовує налаштування кулінарних вподобань за замовчуванням – то в полях вводу параметрів вони відображаються;

– на сторінці користувач задає додаткові параметри пошуку рецептів: наявні або випадкові інгредієнти та фільтри по кулінарним вподобанням або обмеженням;

– після цього користувач натискає на кнопку «Почати пошук» та з’являється результат запиту. Якщо рецептів за такими параметрами немає – то користувачеві про це повідомляється. В іншому випадку з’являється перелік рецептів з короткою інформацією про кожного з них;

– далі користувач натискає на картку рецепту щоб переглянути повну інформацію про нього та додає цей рецепт в список улюблених.

Схема бізнес-процесу відстежування терміну придатності доданих продуктів в інвентар зображена на рисунку 2.4.

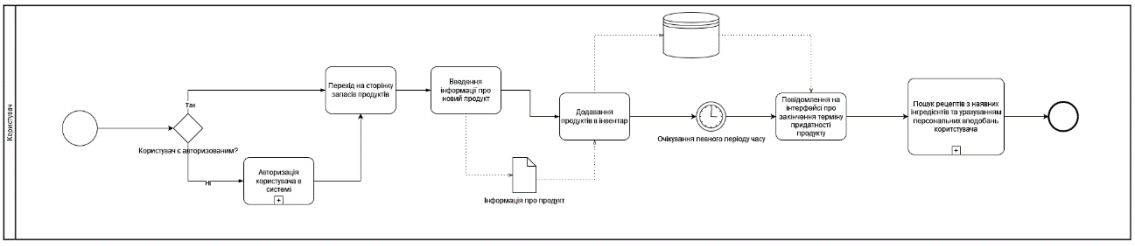


Рисунок 2.4 – Схема бізнес процесу відстежування терміну придатності доданих продуктів в інвентар

Опис послідовності відстежування терміну придатності доданих продуктів в інвентар у веб-застосунку:

– користувач є авторизованим в системі;

– користувач переходить на сторінку інвентаря та додає продукти в нього з вказуванням інформації про термін придатності продукту;

– через певний період часу термін придатності продукту починає наближуватися до кінця та з’являється на картці продукту на сторінці інвентаря повідомлення про дане наближення;

– далі користувач може обрати інгредієнт та сформувати список рецептів, де він використовується з урахуванням персональних вподобань користувача.

Схема бізнес-процесу формування статистики зображена на рисунку 2.5.

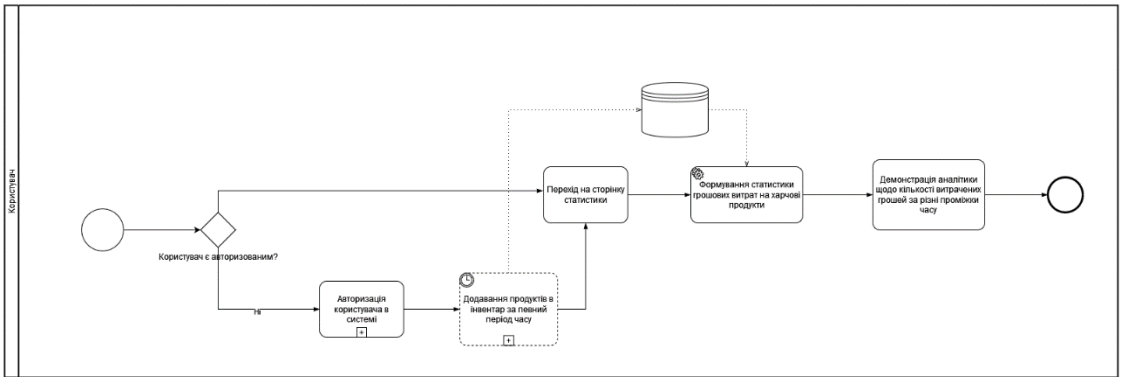


Рисунок 2.5 – Схема бізнес процесу формування статистики

Опис послідовності формування статистики у веб-застосунку:

– користувач є авторизованим в системі;

– користувач додає в інвентар продукти протягом певного періоду часу;

– інформація про додані продукти записується в базу даних та, відповідно, оброблюється. Далі користувач переходить на сторінку статистики та обирає період. За цим періодом формується та демонструється інформація про грошові витрати на продовольчі товари.

2.2 Архітектура програмного забезпечення

Для проектування програмного забезпечення, як вже було зазначено у попередньому розділі, було обрано клієнт-сервісну архітектуру, а саме її різновид – багаторівневу клієнт-серверну архітектуру. Клієнтська частина

застосунку є рівнем представлення та має невелику кількість бізнес-логіки, вона взаємодіє зі стороннім API від Edamam. Серверна частина є прикладним рівнем, в ній реалізовано бізнес-логіку та взаємодію з даними. Як рівень даних – представлене сховище даних SQL Server, що використовується. Загальне представлення архітектури програмного забезпечення можна побачити на рисунку 2.6.

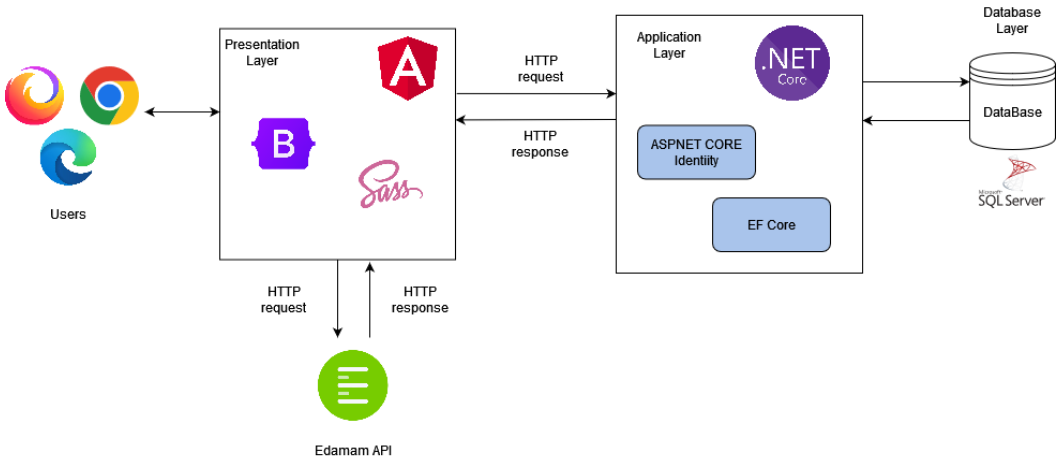


Рисунок 2.6 – Загальне представлення архітектури застосунку

В якості шаблону проєктування серверної частини застосунку було обрано «Архітектуру цибулі» або «Onion Architecture»[13]. Даний архітектурний паттрен має модульний дизайн через забезпечення чітко визначеного поділу між шарами та залежностями у додатку. Цей шаблон сприяє зручності тестування програмного продукту та масштабованості, оскільки внутрішні шари «не знають» про зовнішні – через це він і називається «цибулинним». На рисунку 2.7 зображена схема архітектурного шаблону[14].

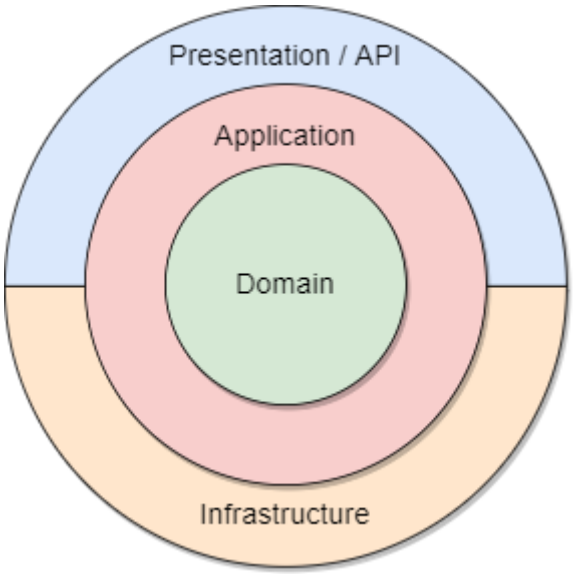


Рисунок 2.7 – Архітектурний шаблон «Onion Architecture»

Головними перевагами використання даного шаблону є імплементування принципу інверсії залежностей (Dependency Inversion Principle), яка стверджує, що високорівневі модулі не мають залежати від нізькорівневих, проте обидві мають залежати від абстракцій. Даний принципе допомагає послабити зв’язок між компонентами та використовувати принцип впровадження залежностей.

В даному архітектурному шаблоні можна виділити чотири основних рівня: доменний, прикладний, інфраструктурний та рівень представлення. Внутрішній або доменний рівень представляє основну бізнес-логіку застосунку. Прикладний рівень слугує посередником між внутрішнім рівнем та зовнішніми інтерфейсами. Рівень інфраструктури реалізовує компоненти доступу та взаємодії з базою даних. Рівень представлення складається з реалізацій API контролерів та моделей представлення.

Такий архітектурний шаблон забезпечує модульність, а отже це полегшує підтримку коду й тестування. До того через наявність чіткого розподілу завдань між рівнями, додаток легко масштабується.

Рішення серверної частини застосунку складається з 4 проєктів: Core, DataAccess, Services та API. В проєкті Core знаходяться моделі даних для їх подальшого використання, інтерфейси сервісів: IProductService – сервіс для взаємодоії з продуктами, IRecipeService – рецептами, IUserInfoService –

інформацією про користувача, IUserSettingsService – налаштуваннями користувача, IStatisticsService – статистикою витрат за певний період. Та в даному проєкті знаходиться інтерфейс базового класу, що реалізує паттерн репозиторію IBaseRepository<T> для взаємодії з сутностями в базі даних за допомогою Entity Framework. У DataAccess знаходяться сутності, налаштування підключення до бази даних, створені міграції даних. Також тут розташовуються інтерфейси та реалізації репозиторіїв для кожної сутності: Product, Recipe, Settings, Category, Ingredient, AppUser. ER діаграма сутностей представлена на рисунку 2.8.

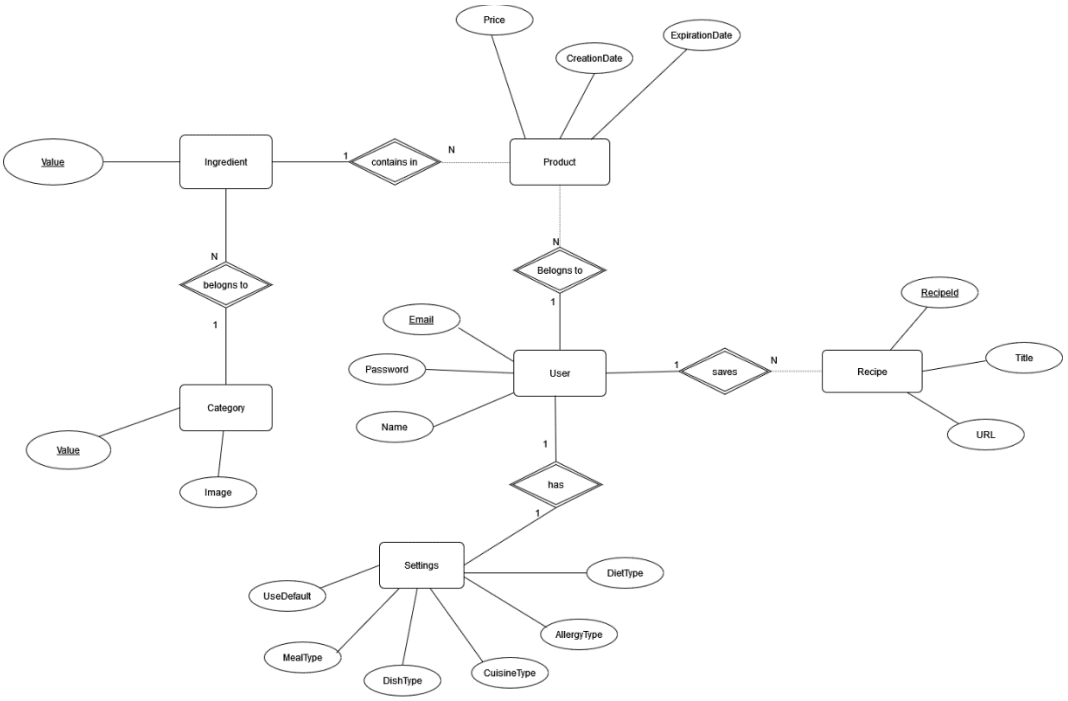


Рисунок 2.8 – ER діаграма сутностей

В проєкті Services знаходяться реалізації сервісів для взаємодії з визначеними сутностями та їх репозиторіями. В проєкті API – реалізовані MVC контролери для відправки моделей представлення даних, створений додатковий сервіс для регістрації, авторизації та аутентифікації користувачів, зберігаються налаштування серверної частини застосунку. Можна виділити основні API контролери: AccountController, ProductBaseController, ProductsController, RecipeController, UserSettingsController та

UserStatisticsContoller. Кожен з них взаємодіє з реалізованими відповідними сервісами.

При розробки клієнтської частини застосунку дотримувався паттерн впровадження залежностей при використанні компонентів ПЗ. Даний шаблон забезпечує виконання одного з принципів SOLID.

Головними компонентами клієнтської частини застосунку є:

– home-page – головна сторінка застосунку з привітанням;

– profile-page – сторінка профілю користувача, на якій відображається його персональна інформація, налаштування, список збережених рецептів;

– profile-edit – компонент для редагування профілю користувача і змінити його налаштувань;

– inventory-page – компонент для відображення запасів продуктів, додавання, редагування та видалення продуктів;

– statistics-page – сторінка для відображення сформованої статистики витрат на харчові продукти;

– recipe-generator-page – сторінка для налаштування параметрів пошуку рецептів;

– recipe-search-result-page – сторінка виводу результатів пошуку рецептів за заданими параметрами;

– login-pop-up – діалогове вікно з формою авторизації користувача;

– signup-pop-up -діалогове вікно з формою реєстрації користувача.

Інші компоненти, допоміжні функції, типи знаходяться в теці shared-module. Додаткові діалогові вікна – в теці modals. Також можна виділити реалізовані сервіси для взаємодії з серверною частиною та компонентами для представлення отриманих даних. Були створені ProductService, RecipeService, StatisticsService, AuthService, SettingsService. Дані сервіси відповідають за виконання HTTP запитів для комунікації з відповідними контролерами на серверній частині додатку і отримання відповідей.

2.3 Конструювання програмного забезпечення

Для реалізації серверної частини, як було зазначено в попередньому розділі дипломного проєкту, було обрано мову програмування C# та платформу .NET, а саме .NET Core 6.0. Для взаємодії з базою даних SQL Server додатково використовувався Entity Framework. Як допоміжний засіб у налаштуванні реєстрації, авторизації користувачів був використаний пакет Identity з ASP.NET Core[15]. У якості автоматичного перетворювача сутностей та моделей представлення обраний пакет AutoMapper. Під час розробки для його коректної роботи були створені додаткові профілі обгортання.

Клієнтська частина додатку написана на мові програмування TypeScript з використанням фреймворку Angular 15. Для допоміжного засобу стилістики застосунку взято фреймворк Bootstrap. Додатково використовувалися Angular Material, ng-select.

У додатку «Структурна схема класів серверної частини застосунку» представлена схема класів серверної частини додатку та у додатку «Структурна схема класів клієнтської частини застосунку» представлена схема класів клієнтської частини.

В якості системи управління базами даних використовується Microsoft SQL Management Studio, як вже було зазначено раніше як база даних використовується Microsoft SQL Server. База даних серверу призначена для зберігання інформації про користувачів, їх запаси продуктів, налаштування улюблені рецепти, для зберігання списку інгредієнтів та категорій. Опис таблиць бази даних наведено у таблицях 2.1 - 2.5. Фізична модель бази даних наведена на рисунку 2.8.

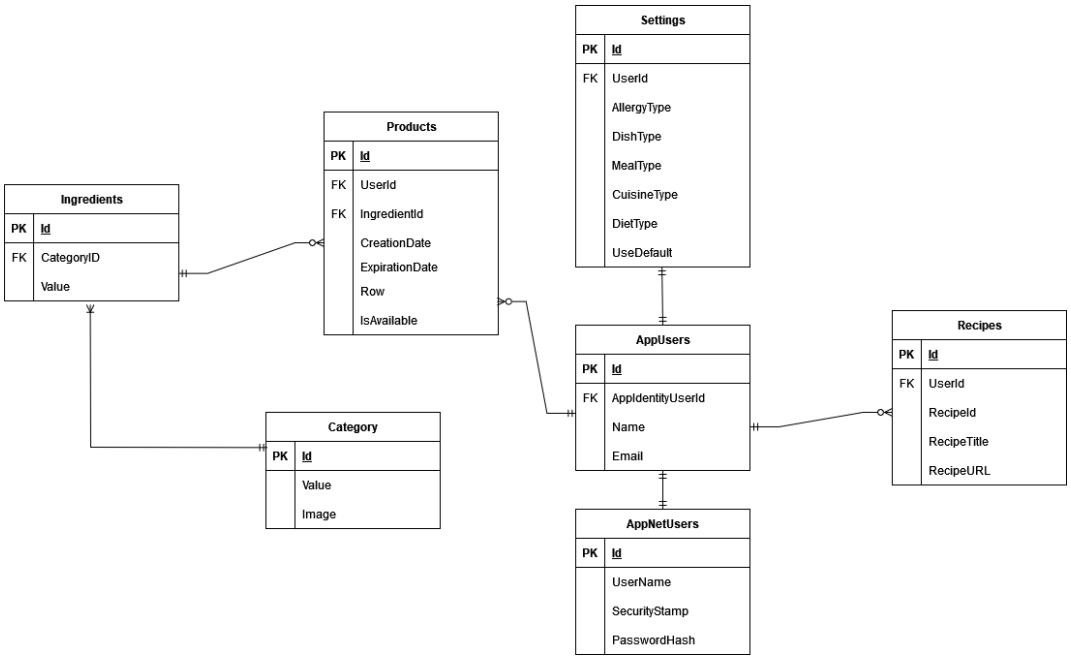


Рисунок 2.8 – Фізична модель бази даних

Таблиця 2.1 – Опис таблиці User

Таблиця	Назва поля	Тип даних	Опис
AppUsers	id	uniqueidentifier	Ідентифікаційний номер користувача в системі. Первинний ключ
	Name	nvarchar(50)	Ім’я користувача
	Email	nvarchar(50	Електронна пошта користувача

Продовження таблиці 2.1

	AppIdentityUserId	uniqueidentifier	Ідентифікаційний номер персональних даних користувача в системі. Зовнішній ключ

Таблиця 2.2 – Опис таблиці Category

Таблиця	Назва поля	Тип даних	Опис
Category	id	uniqueidentifier	Ідентифікаційний номер категорії. Первинний ключ
	Value	nvarchar(255)	Назва категорії
	Image	nvarchar(255)	Посилання на зобаження категорії

Таблиця 2.3 – Опис таблиці Ingredient

Таблиця	Назва поля	Тип даних	Опис
Ingredient	id	uniqueidentifier	Ідентифікаційний номер інгредієнта. Первинний ключ
	CategorytId	uniqueidentifier	Ідентифікаційний номер категорії Зовнішній ключ
	Value	nvarchar(255)	Назва інгредієнта

Таблиця 2.4 – Опис таблиці Products

Таблиця	Назва поля	Тип даних	Опис
Products	id	uniqueidentifier	Ідентифікаційний номер продукту з інвентаря користувача. Первинний ключ
	IngredientId	uniqueidentifier	Ідентифікаційний номер інгредієнта. Зовнішній ключ
	UserId	uniqueidentifier	Ідентифікаційний номер користувача. Зовнішній ключ
	CreationDate	datetime	Дата створення продукту
	ExpirationDate	datetime	Дата закінчення терміну придатності продукту
	Price	float	Ціна продукту
	IsAvailable	bool	Поле, що вказує чи є продукт доступним

Таблиця 2.5 – Опис таблиці Settings

Таблиця	Назва поля	Тип даних	Опис
Settings	id	uniqueidentifier	Ідентифікаційний номер налаштувань користувача. Первинний ключ
	UserId	uniqueidentifier	Ідентифікаційний номер користувача. Зовнішній ключ
	AllergyType	nvarchar(255)	Налаштування користувача щодо наявних дієтичних обмежень
	MealType	nvarchar(255)	Налаштування користувача щодо наявних кулінарних вподобань за видом страви
	DietType	nvarchar(255)	Налаштування користувача щодо наявних кулінарних вподобань за видом дієти
	CuisineType	nvarchar(255)	Налаштування користувача щодо наявних кулінарних вподобань за видом кухні

Продовження таблиці 2.5

	DishType	nvarchar(255)	Налаштування користувача щодо наявних кулінарних вподобань за видом блюда
	UseDefault	bool	Поле, що вказує чи використовувати налаштування за замовчуванням

Таблиця 2.6 – Опис таблиці Recipes

Таблиця	Назва поля	Тип даних	Опис
Recipes	id	uniqueidentifier	Ідентифікаційний номер запису збереженого рецепту користувача. Первинний ключ
	UserId	uniqueidentifier	Ідентифікаційний номер користувача. Зовнішній ключ
	RecipeId	nvarchar(255)	Ідентифікаційний номер рецепту
	RecipeTitile	nvarchar(255)	Назва рецепту
	RecipeURL	nvarchar(255)	Посилання на першоджерело рецепту

Таблиця 2.7 – Опис таблиці AppNetUsers

Таблиця	Назва поля	Тип даних	Опис
AppNetUsers	id	unique identifier	Ідентифікаційний номер персональних даних користувача. Первинний ключ
	UserName	nvarchar(255)	Ім’я користувача в системі
	SecurityStamp	nvarchar(255)	Значення, що змінюється коли змінюється пароль або логін.
	PasswordHash	nvarchar(255)	Запис паролю користувача після хешування.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.8.

Таблиця 2.8 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Webstorm IDE	Головне інтегроване середовище розробки програмного забезпечення клієнтської частини дипломної роботи.

Продовження таблиці 2.8

2	Postman	Програмне забезпечення, що використовується для тестування REST запитів. Використовувалось для тестування API інтерфейсів, клієнтських запитів.
3	Visual Studio 2022	Головне інтегроване середовище розробки програмного забезпечення серверної частини дипломної роботи.
4	AutoMapper	Пакет для перетворення між DTO й сутностями для платформи .NET.

2.4 Аналіз безпеки даних

Безпека даних є ключовою проблемою при зберіганні особистих даних користувача. Програмне забезпечення передбачає наявність реалізації та авторизації користувача, а тому безпека такої чутливої інформації є важливою. Для підвищення захищеності та безпеки даних в дипломній роботі на серверній частині додатку використовувалися унікальні ідентифікатори – GUID. Їх застосування дуже ускладнює підбір ідентифікаційного номеру для отримання даних. Для авторизації та реєстрації було використано методи з бібліотеки Identity. Вона пропонує зберігання паролів коритсувачів у вигляді хешованих даних за алгоритмом HMACSHA256. Також в даному програмному забезпеченні передбачена авторизація за допомогою Google акаунта. Використання такого підходу гарантує безпеку даних, оскільки особисті дані користувачів зберігаються на серверах Google. Для авторизації користувача був обраний токен JWT. Всі методи API є непублічними, а отже до їх доступу необхідно передача даного токену при запиті, а отже це унеможливлює отримання даних про інших користувачів.

Висновки до розділу

У другому розділі дипломної роботи було виконано моделювання та конструювання програмного забезпечення, що розроблюється, розглянуто та проаналізовано наступне:

– були виділені та описані бізнес процеси програмного забезпечення з використанням BPMN моделей;

– визначено та описано будову архітектури програмного забезпечення, побудовано структурну схему класів, зображено архітектурний патерн, що був використаний для розробки серверної частини застосунку;

– побудовано схему архітектурного шаблону, що використовується;

– описано сутності, що існують в системі та побудовано ER діаграму бази даних на основі них;

– описано утиліти та бібліотеки, що використовувалися під час реалізації дипломної роботи.

Серверна частина веб-застосунку реалізовано на платформі .NET з використанням пакетів Entity Framework, ASP .NET Core Identity, AutoMapper, клієнтська частина – за допомогою фреймворку Angular. Як сховище даних було обрано SQL Server. Під час проєктування архітектури програмного забезпечення було виділено основні паттерни, які мають використовуватися.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Протягом процесу розробки веб-застосунку були задіяні різноманітні інструменти для підтримки якості програмного коду програмного забезпечення. Як засоби для статичного аналізу коду та форматування під час його написання на клієнтській частині застосунку використовувалися ESLint і Prettier, відповідно. Це бібліотеки з відкритим кодом для JavaScript та TypeScript. На серверній частині застосунку використовується StyleCop – статичний аналізатор коду C# від Microsoft.

Для оцінки та аналізу якості програмного забезпечення було обрано SonarQube – платформу для аналізу та вимірювання якості програмного коду з урахуванням різних метрик. Предметна область дипломного проєкту передбачає наявність веб-застосунку, який має мінімум вразливостей і «гарячих точок» у безпеці при користуванні та зберіганні клієнтських даних. Ще однією важливою метрикою оцінки якості написаного продукту є можливість його подальшого підтримування. Розроблене програмне забезпечення має неперервний цикл підтримки та розвитку через додавання нового функціоналу, покращення існуючого, а тому написаний код повинен бути читабельним та підтримуваним через деякий час. Додатковою метрикою, за якою SonarQube вимірює якість ПЗ, є “Code smell” – це невеликі проблемі в написанні коду, що краще було би виправити при подальшій розробці. До них відносяться великі класи, дублювання коду, занадто довгі іменування функцій, методів або змінних тощо. SonarQube був встановлений на локальну машину та розгорнутий. Проведений аналіз проєкту, результати оцінки зображені на рисунку 3.1.

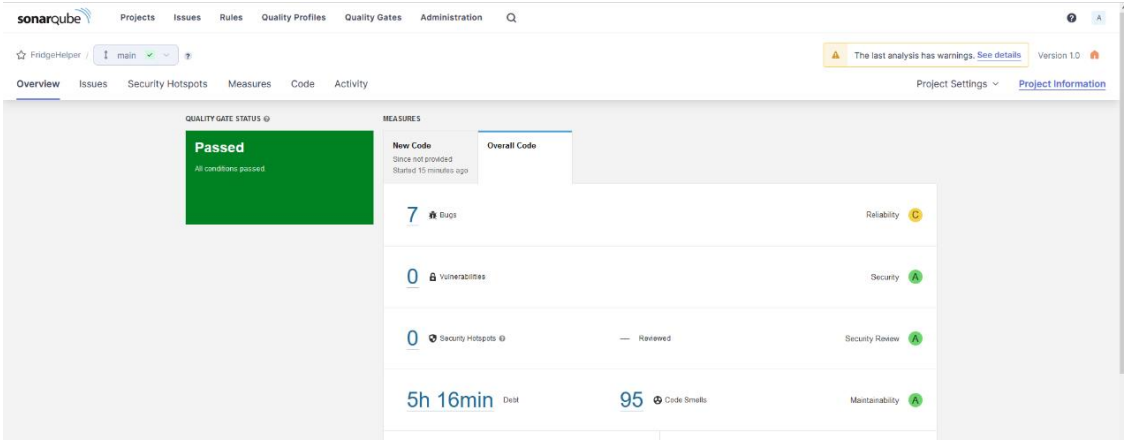


Рисунок 3.1 – Результат аналізу якості програмного забезпечення за допомогою засобу SonarQube

Інструмент для аналізу знайшов 7 багів та дав оцінку C за метрикою надійність, в той же час він дав оцінку A за метриками обслуговування та безпеки.

3.2 Опис процесів тестування

Для перевірки правильності реалізації написаних компонентів, контролерів, сервісів, було проведено тестування програмного забезпечення за допомогою написання модульних тестів та виконання мануального тестування. Наскрізне ручне тестування роботи програмного забезпечення перевірило наявну функціональність системи шляхом імітації бізнес-процесів та реальних сценаріїв взаємодії з застосунком. Така техніка тестування має на меті знайти помилки при взаємодії різних компонентів або частин програми. Методика наскрізного тестування проводилася протягом всього циклу розробки програмного забезпечення та допомогла виявити проблеми при інтеграції сторонньої API, клієнтської та серверної частин застосунку і вчасно виправити недоліки.

На серверній частині застосунку проводилося димове тестування з використанням застосунку Postman для перевірки з'єднання з базою даних. Модульне тестування відповідало за взаємодію зі сховищем даних, правильність виводу моделі представлення. Як засіб для написання модульних

тестів на платформі .NET використовувався фреймворк xUnit[16]. Результат проведення тестування зображений на рисунку 3.2.

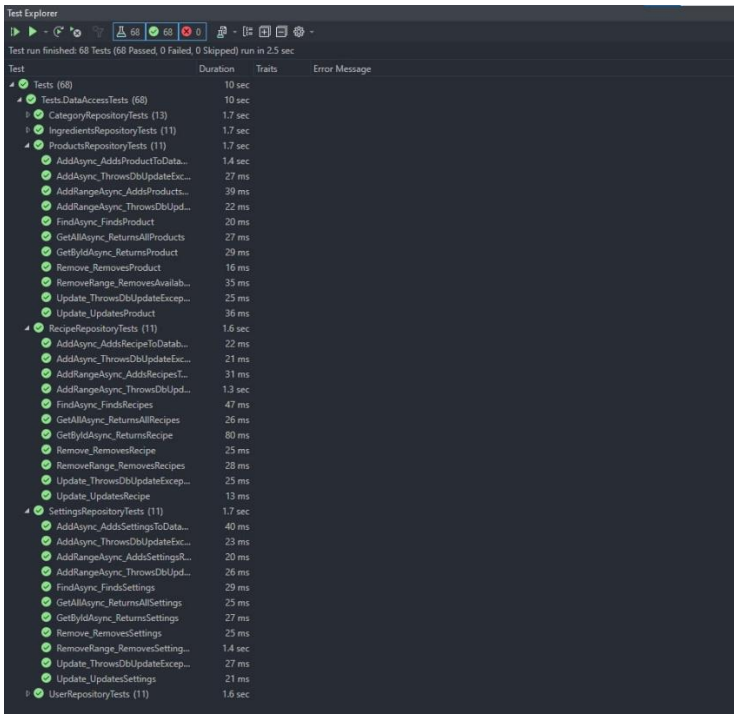


Рисунок 3.2 – Результат виконання модульного тестування на серверній частині застосунку

Для проведення юніт-тестування клієнтської частини додатку були задіяні пакети Jasmine та Karma. Написані тести для перевірки базового функціоналу головних компонентів та правильного їх відображення на сторінці користувача. Результат проведення тестування клієнтської частини можна побачити на рисунку 3.3.

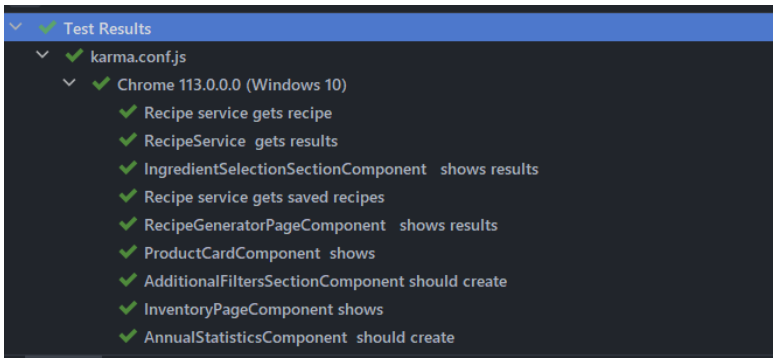


Рисунок 3.3 – Результат виконання модульного тестування на клієнтській частині застосунку

Як вже було зазначено, крім модульного тестування, було проведено мануальне тестування розробленого програмного забезпечення, опис відповідних тестів наведено у таблицях 3.3 – 3.30. Таблиці супроводжуються скріншотами результатів тестів.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача в системі
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку
Вхідні дані	Ім’я, електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	Користувач знаходиться на головній сторінці застосунку, натискає на кнопку “Login”, що розташована на навігаційній панелі. Відкривається діалогове вікно для авторизації в застосунку. Користувач натискає на посилання “sign-up” та переходить до вікна реєстрації. З’являється форма введення даних, у відповідні поля вводяться: ім’я користувача, що не має містити цифр або спеціальних символів, коректна електронна пошта, яка ще не існує в системі, пароль від 8 до 40 символів. Пароль має містити лише латинські букви, мінімум одне число і

Продовження таблицы 3.1

	один спеціальний символ. Далі користувач вводить пароль ще один раз для його підтвердження та погоджується з умовами користування. Після введення всіх даних стає активною кнопка реєстрації. Користувач натискає на неї.
Очікуваний результат	Реєстрація відбувається успішно, обліковий запис користувача створюється в системі та користувач перенаправляється на активне вікно форми авторизації.
Фактичний результат	Реєстрація відбувається успішно, обліковий запис користувача створюється в системі та користувач перенаправляється на активне вікно форми авторизації.

Процес проведення тесту та його результат зображені на рисунках 3.4 – 3.5.

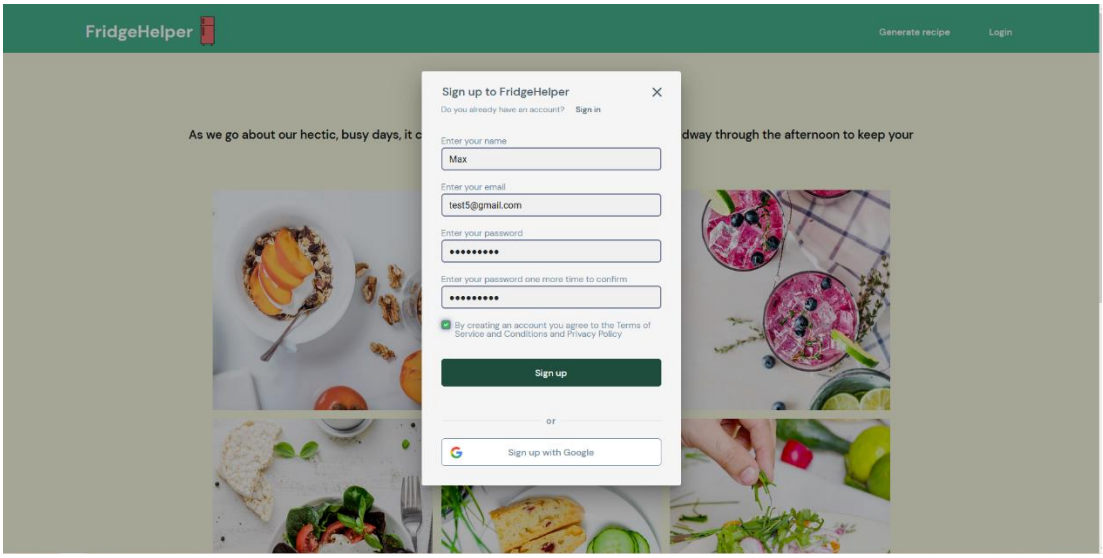


Рисунок 3.4 – Введення даних користувача при реєстрації

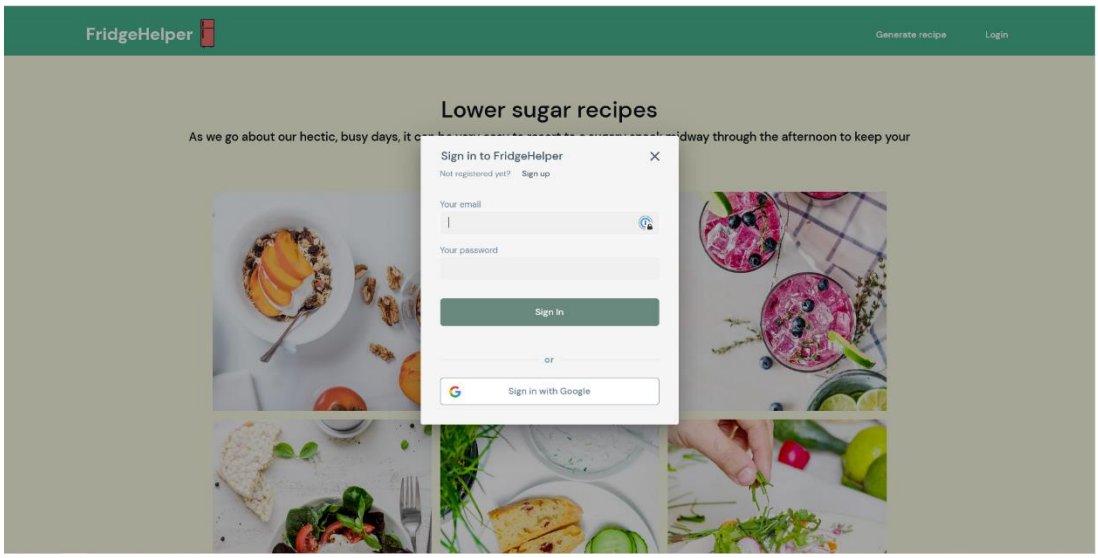


Рисунок 3.5 – Перенаправлення користувача на діалогове вікно авторизацію після успішної реєстрації

Таблиця 3.2 – Тест 1.2

Тест	Некоректне введення даних під час реєстрації
Модуль	Реєстрація користувача
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку
Вхідні дані	Ім’я, електронна пошта, пароль, підтвердження паролю в некоректному форматі.
Опис проведення тесту	Користувач перебуває на головній сторінці застосунку, натискає на посилання “Login”, що знаходиться на навігаційній панелі. З’являється діалогове вікно для авторизації в додатку. Користувач натискає на посилання “sign-up” та переходить до вікна реєстрації. З’являється форма введення даних, у відповідні поля вводяться: ім’я користувача електронна пошта, пароль, його підтвердження та натискання на прапорець біля погодження з умовами користування. Користувач вводить інформацію в некоректному форматі.

Продовження таблицы 3.2

Очікуваний результат	Повідомлення про помилку підсвічується червоним кольором, реєстрація користувача не відбувається.
Фактичний результат	Повідомлення з текстом помилки, що з’явилася, підсвічується червоним кольором, реєстрація користувача не відбувається.

Стан системи після введення неправильних даних зображений на рисунку 3.6.

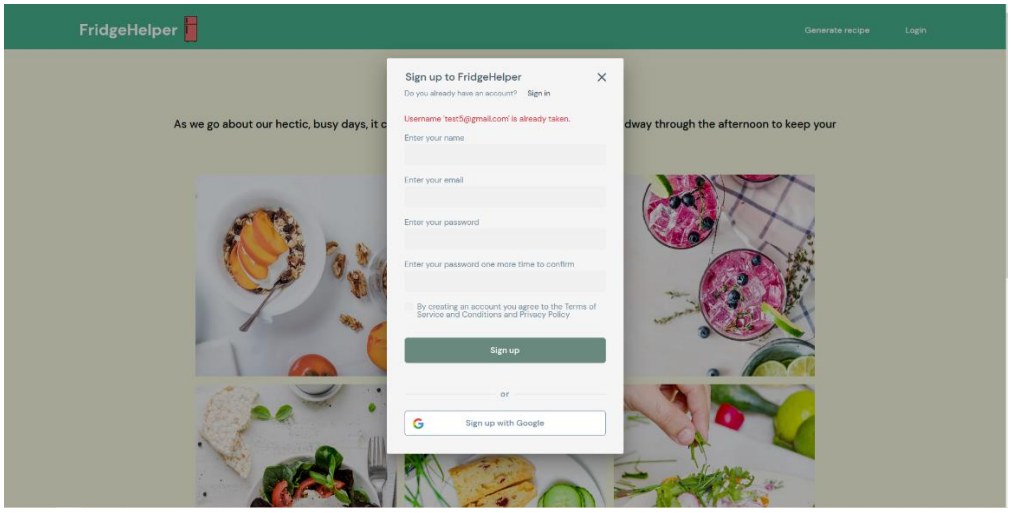


Рисунок 3.6 – Повідомлення, що з’являється при спробі створити користувача з поштою, що вже існує в системі

Таблиця 3.3 – Тест 1.3

Тест	Проведення авторизації користувача
Модуль	Авторизація користувача
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку
Вхідні дані	Електронна пошта, пароль.

Продовження таблиці 3.3

Опис проведення тесту	Користувач перебуває на головній сторінці застосунку, натискає на посилання “Login”, що знаходиться на навігаційній панелі. З’являється діалогове вікно для авторизації в додатку та форма в яку необхідно ввести дані. У форму вводяться електронна пошта, що вже існує в системі та пароль від облікового запису довжиною не більше 40 символів. пароль. Після введення всіх даних, стає активною кнопка «Авторизуватися». Користувач натискає на неї.
Очікуваний результат	Користувач успішно авторизується в системі, перенаправляється на головну сторінку. На навігаціній панелі додаються сторінки, що були недоступні анонімному користувачеві.
Фактичний результат	Користувач успішно авторизується додатку, він перенаправляється на головну сторінку застосунку. На навігаціній панелі додаються сторінки, що були недоступні анонімному користувачеві.

Процес проведення тесту та результат його проведення зображені на рисунках 3.7 – 3.8.

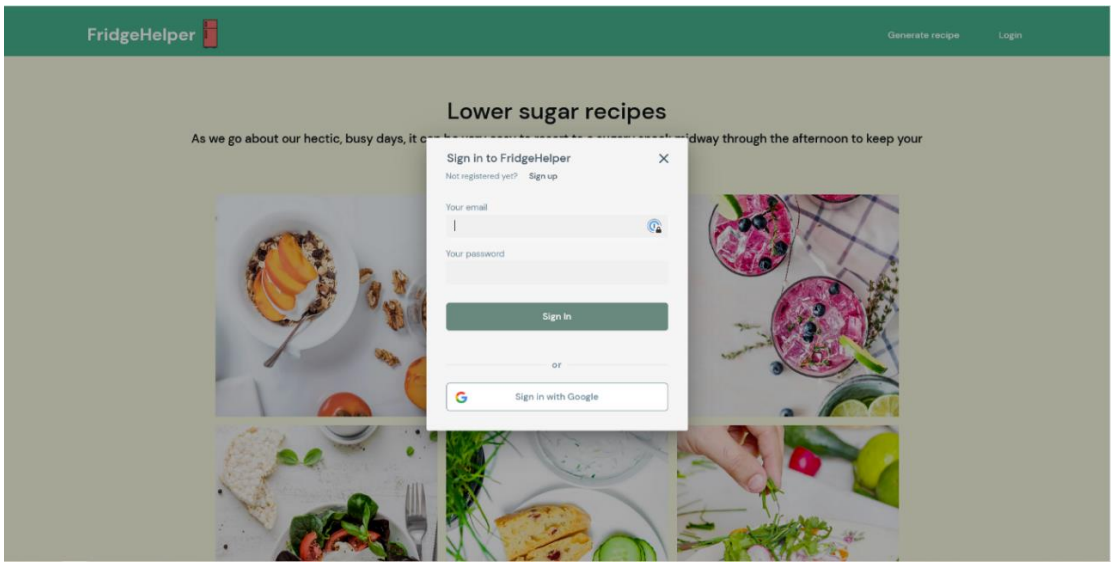


Рисунок 3.7 – Вид діалогового вікна з формою для введення даних для авторизації

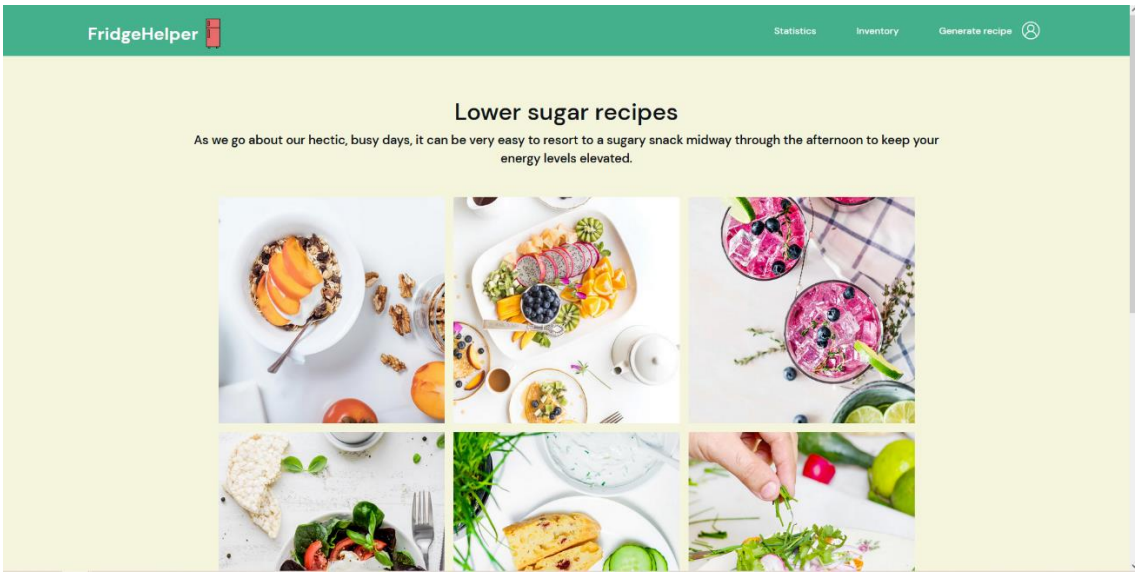


Рисунок 3.8 – Вид головної сторінки після успішної авторизації

Таблиця 3.4 – Тест 1.4

Тест	Введення неіснуючої пошти під час авторизації
Модуль	Авторизація користувача
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку

Продовження таблиці 3.4

Вхідні дані	Електронна пошта, пароль
Опис проведення тесту	Користувач знаходиться на головній сторінці застосунку, натискає на посилання “Login” в навігаційній панелі. З’являється діалогове вікно з формою для введення даних авторизації в додатку. Користувач вводить пошту, обліковий запис до якої не існує в системі. Вводить пароль у відповідне поле вводу. Стає активною кнопка «Авторизуватися». Користувач натискає на неї погодження з умовами користування.
Очікуваний результат	З’являється повідомлення про те, що даного облікового запису не існує в системі.
Фактичний результат	В формі вводу з’являється повідомлення про те, що користувача з такою електронною поштою не існує.

Стан системи після введення електронної пошти, яка не існує в системі під час авторизації зображений на рисунку 3.9.

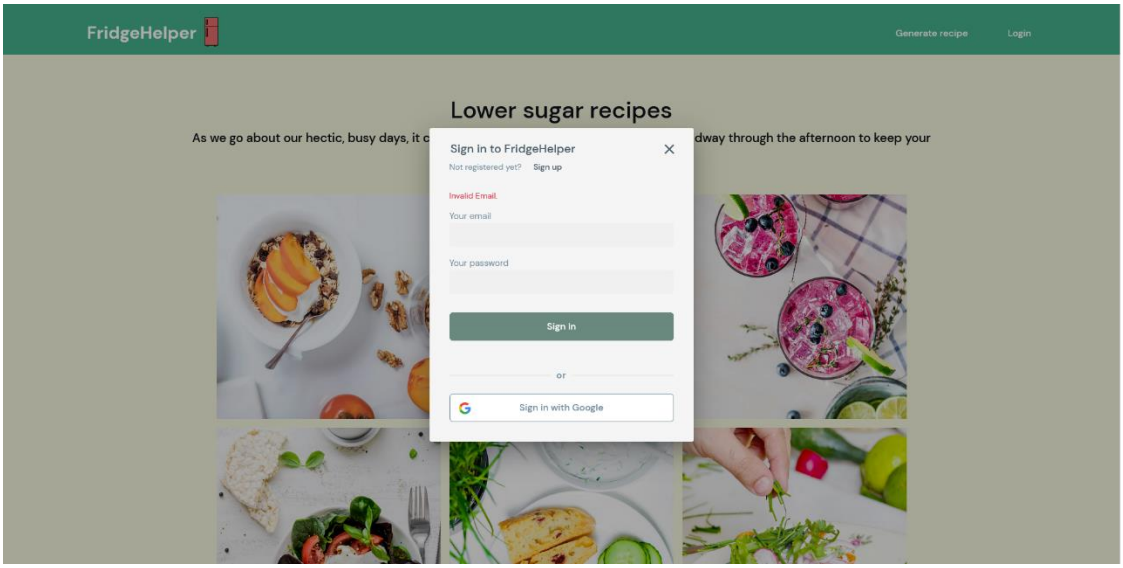


Рисунок 3.9 – Вид форми авторизації після спроби авторизуватися за допомогою пошти, якою не існує в системі

Таблиця 3.5 – Тест 1.5

Тест	Спроба пошуку рецепту без вибору інгредієнтів
Модуль	Пошук рецептів
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку
Вхідні дані	-
Опис проведення тесту	Користувач знаходиться на головній сторінці веб-застосунку та натискає на посилання “generate recipe” з навігаційної панелі. Він перенаправляється на сторінку пошуку рецептів на основі інгредієнтів і вподобань. Обираються певні вподобання та натискається кнопка пошуку, при цьому користувач не обирає ніяких інгредієнтів.
Очікуваний результат	З’являється повідомлення про неможливість пошуку без вибору хоча б одного інгредієнту.
Фактичний результат	Біля кнопки пошуку з’являється повідомлення червоним кольором про неможливість здійснення пошуку без обраних інгредієнтів.

Стан системи після спроби виконання пошуку без вибору інгредієнтів зображена на рисунку 3.10.

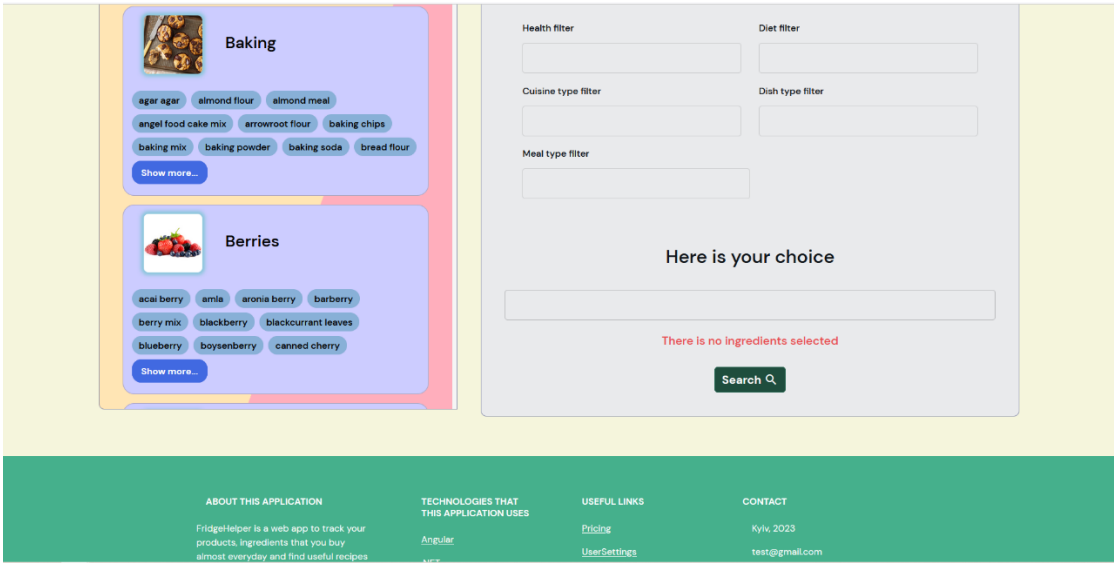


Рисунок 3.10 – Повідомлення, що з’являється при спробі пошуку рецептів без вибору інгредієнтів

Таблиця 3.6 – Тест 1.6

Тест	Відображення результатів пошуку рецептів на основі інгредієнтів та вподобань.
Модуль	Пошук рецептів на основі інгредієнтів та вподобань
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на головній сторінці застосунку
Вхідні дані	Інгредієнти, кулінарні вподобання та дієтичні обмеження
Опис проведення тесту	Користувач перебуває на головній сторінці застосунку, переходить на сторінку генерації рецептів за допомогою посилання з навігаціною панелі. В меню вибору інгредієнтів користувач обирає за своїм бажанням певну кількість продуктів. Далі обирає додаткові фільтри за кулінарними вподобаннями і натискає на кнопку «Знайти рецепти».

Продовження таблиці 3.6

Очікуваний результат	Відображається результат пошуку рецептів за обраними параметрами. Перелік рецептів може бути відсутнім, якщо таких не існує.
Фактичний результат	З’являється результат пошуку у вигляді переліку карток рецептів з короткою інформацією про них. Даний перелік відсутній, якщо рецепти не існують за такими параметрами, з’являється повідомлення замість цього.

Процес проведення описаного тесту та стани системи після виконання пошуку рецептів за обраними параметрами зображені на рисунках 3.11 – 3.13.

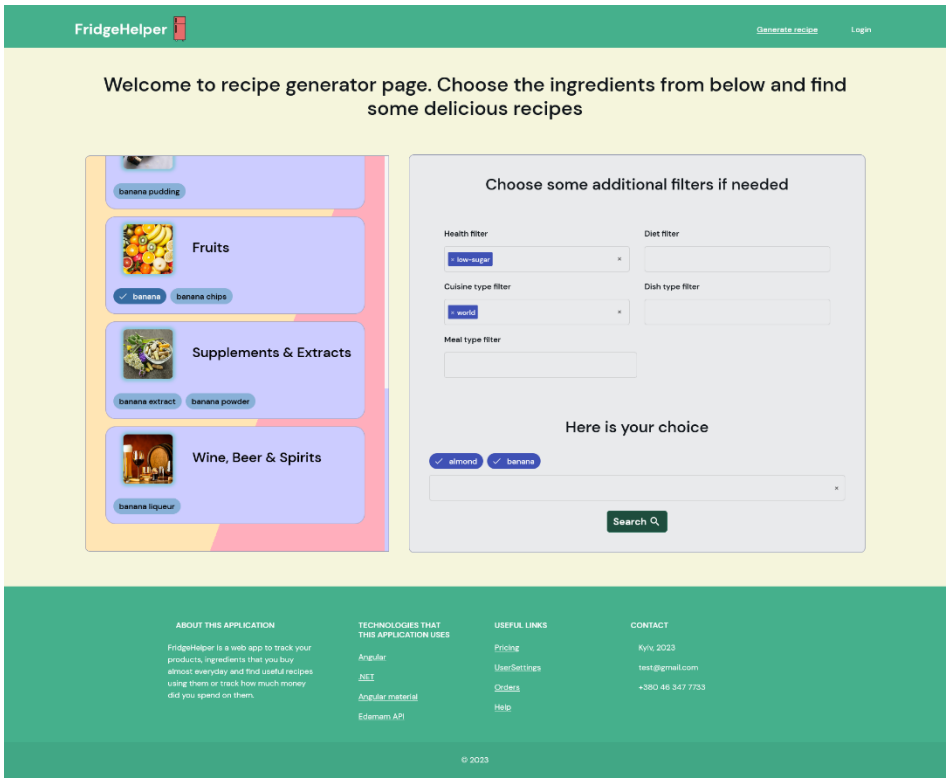


Рисунок 3.11 – Вигляд меню вибору інгредієнтів після вибору параметрів для пошуку рецептів

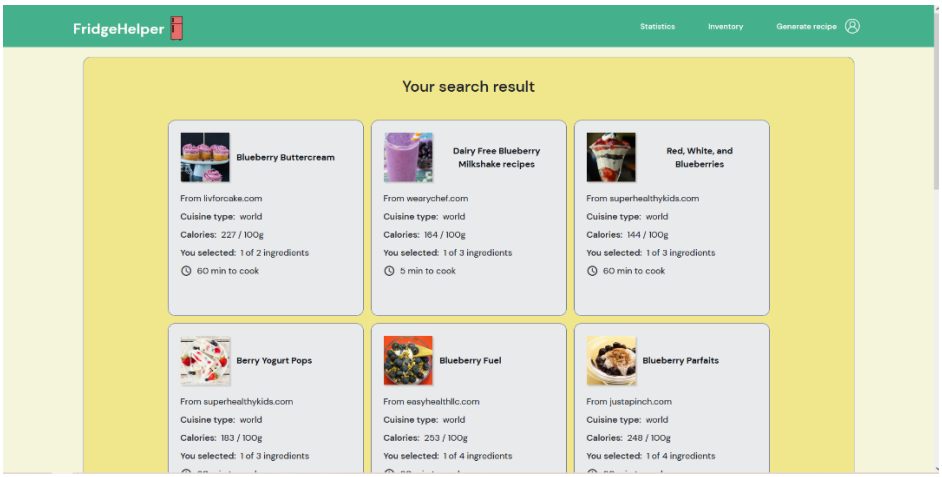


Рисунок 3.12 – Результат пошуку рецептів за заданими параметрами

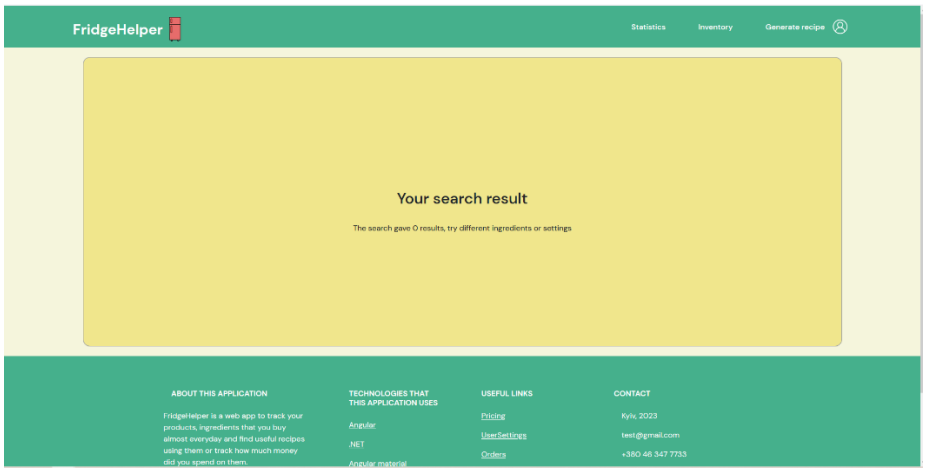


Рисунок 3.13 – Вигляд системи якщо пошук рецептів не дав результатів

3.3 Опис контрольного прикладу

Контрольний приклад, що був розроблений під час написання дипломної роботи має наступний функціонал: реєстрація, авторизація користувачів та пошук рецептів були протестовані та продемонстровані в попередньому підрозділі. Для здійснення додавання, редагування та видалення продуктів в інвентарі користувачеві необхідно бути зареєстрованим в додатку, потім перейти до вкладки «Інвентар», посилання знаходиться на навігаційній панелі застосунку. Сторінка інвентаря зображена на рисунку 3.14.

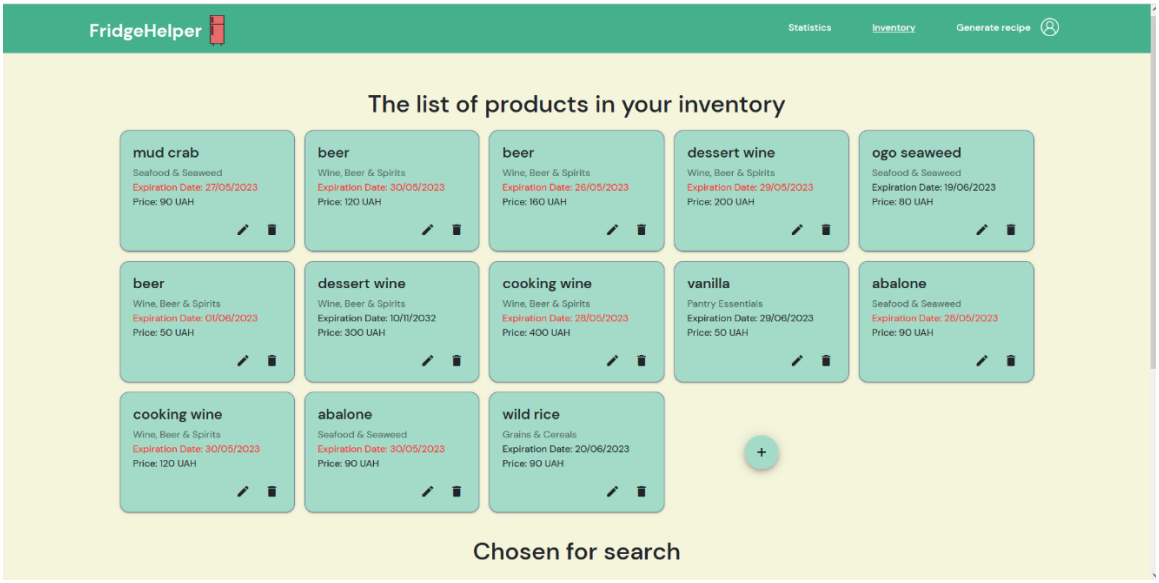


Рисунок 3.14 – Сторінка інвентаря користувача

Для додавання нового продукту до інвентаря, користувач натискає на відповідну кнопку, з'являється форма введення даних, куди вноситься назва інгредієнта, ціна та термін придатності. Вигляд даної форми представлений на рисунку 3.15.

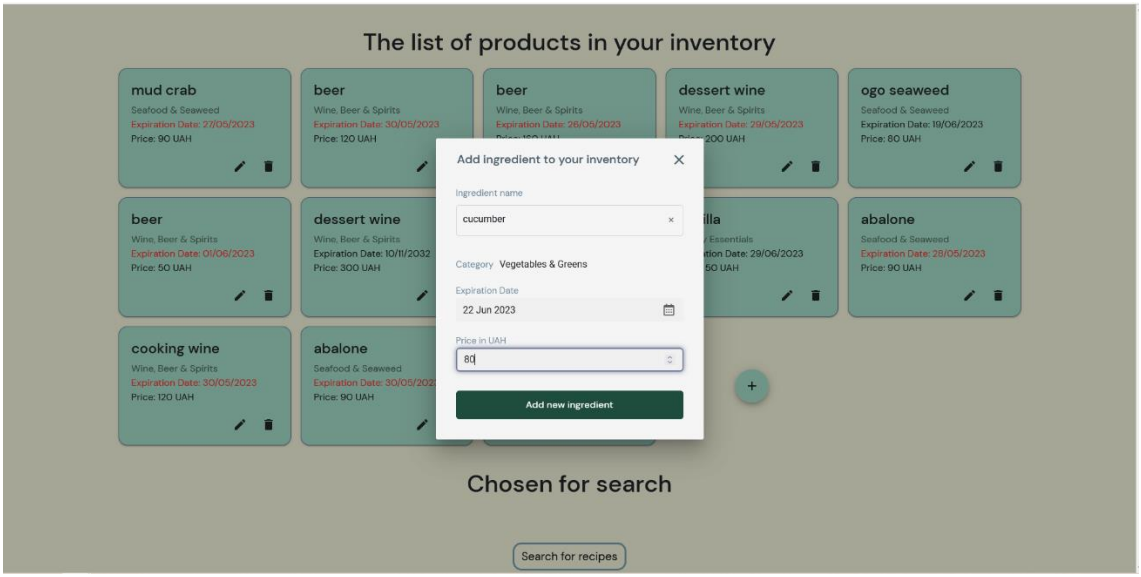


Рисунок 3.15 – Додавання нового продукту до інвентаря

Після введення необхідних даних у коректному форматі, стає активною кнопка «Додати до інвентаря», користувач натискає на неї, продукт додається до запасів. Результат проведення операції додавання зображений на рисунку 3.16.

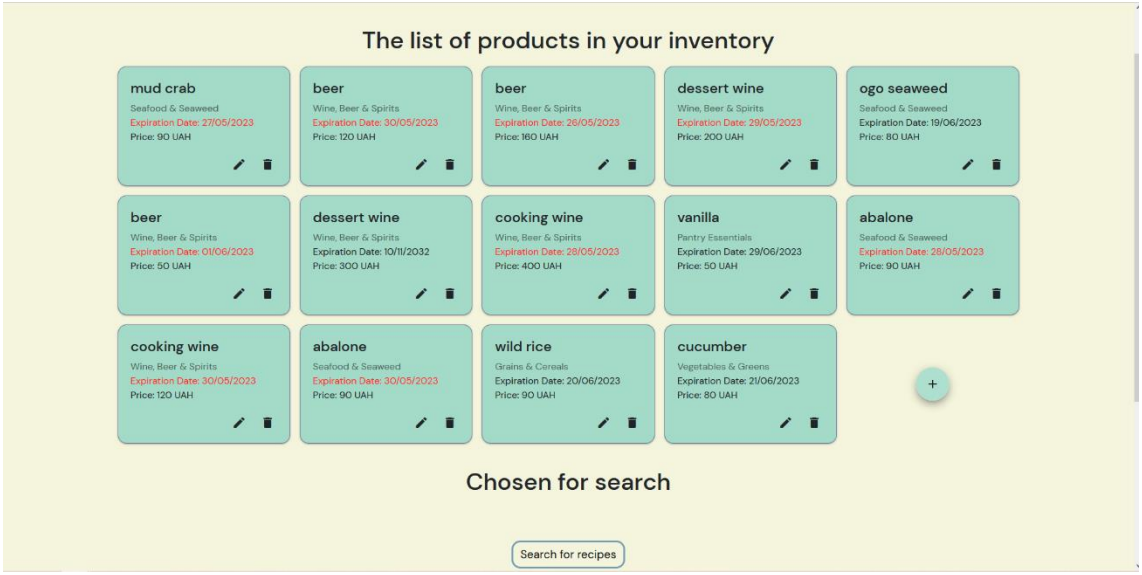


Рисунок 3.16 – Результат додавання нового продукту до інвентаря

Користувач може змінювати інформацію про продукт або видаляти його з інвентаря, наприклад, після його споживання. Результати проведення даних операцій зображені на рисунках 3.17 – 3.19.

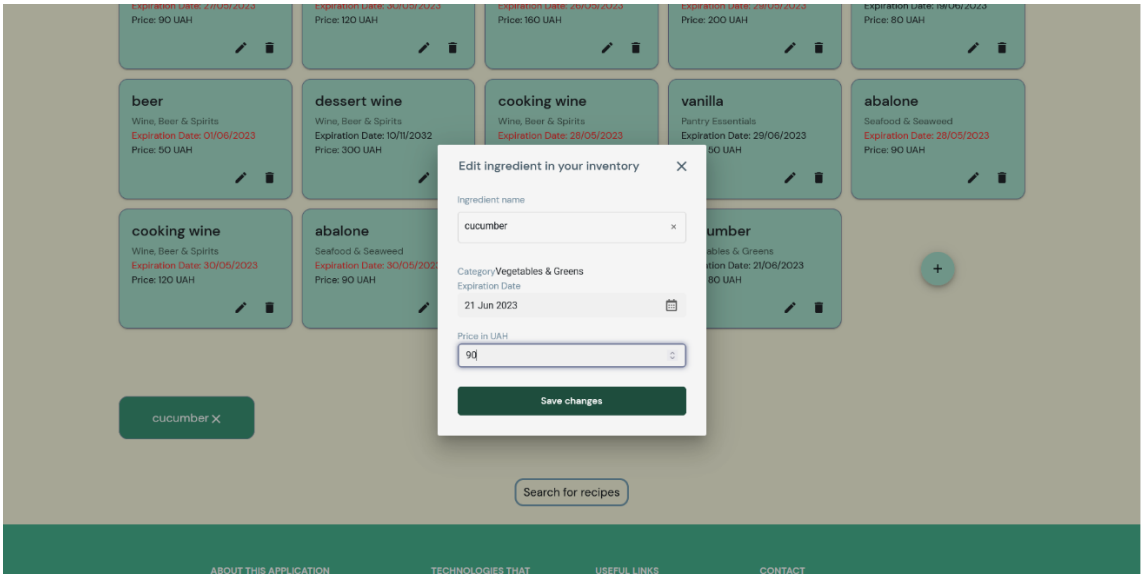


Рисунок 3.17 – Форма для редагування інформації про продукт

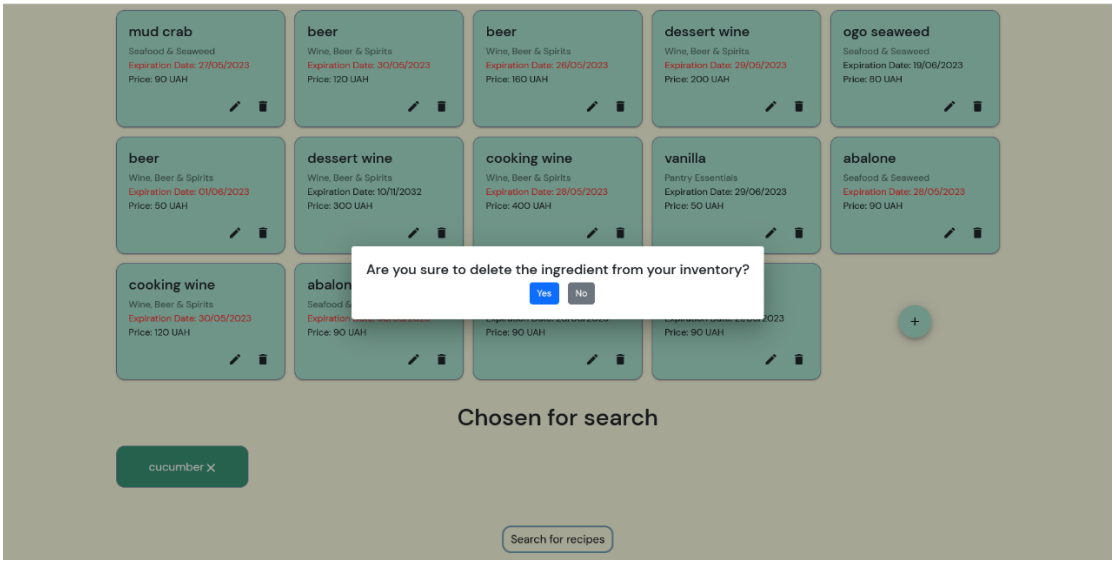


Рисунок 3.18 – Діалогове вікно підтвердження видалення продукту з інвентаря

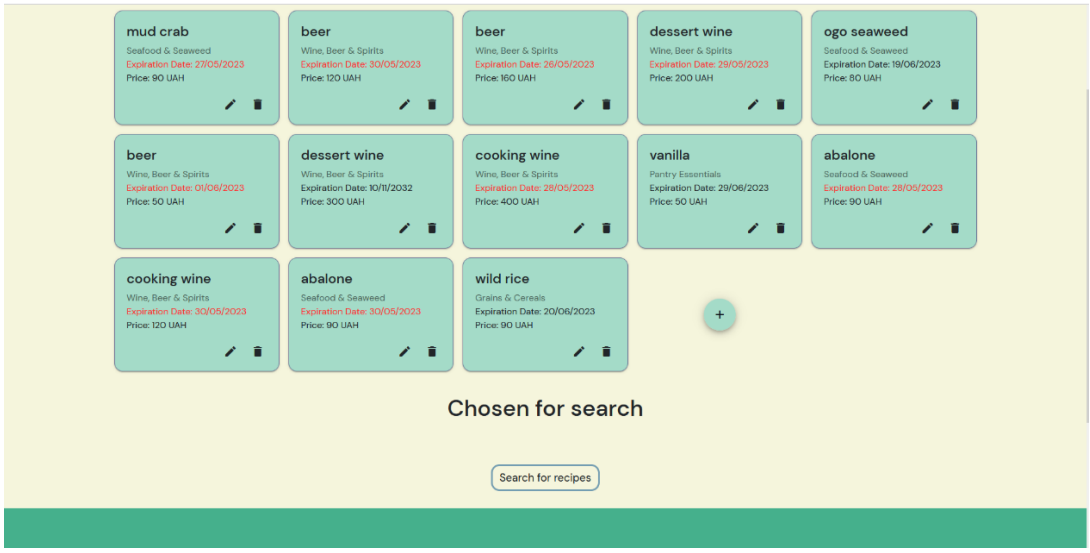


Рисунок 3.19 – Результат видалення продукту з інвентаря

Користувач може перейти за допомогою навігаційної панелі до свого профілю. Там зображене ім’я, яке користувач вводив при реєстрації в застосунку, електронна пошта та налаштування кулінарних вподобань і дієтичних обмежень, що можуть використовуватися за замовчуванням при переході на сторінку пошуку рецептів. Також користувач при необхідності може змінювати дану всю інформацію та налаштування, окрім адреси електронної пошти. Це він може зробити за допомогою натискання на кнопку «Редагування профілю» і перенаправлення на сторінку редагування. На сторінці профіля зображений список збережених рецептів, користувач може

відкрити та подивитися повну інформацію про рецепт, який знаходиться в списку улюблених та видаляти його зі списку. Вигляд користувацького інтерфейсу, що забезпечує описану функціональність, зображено на рисунках 3.20 – 3.26.

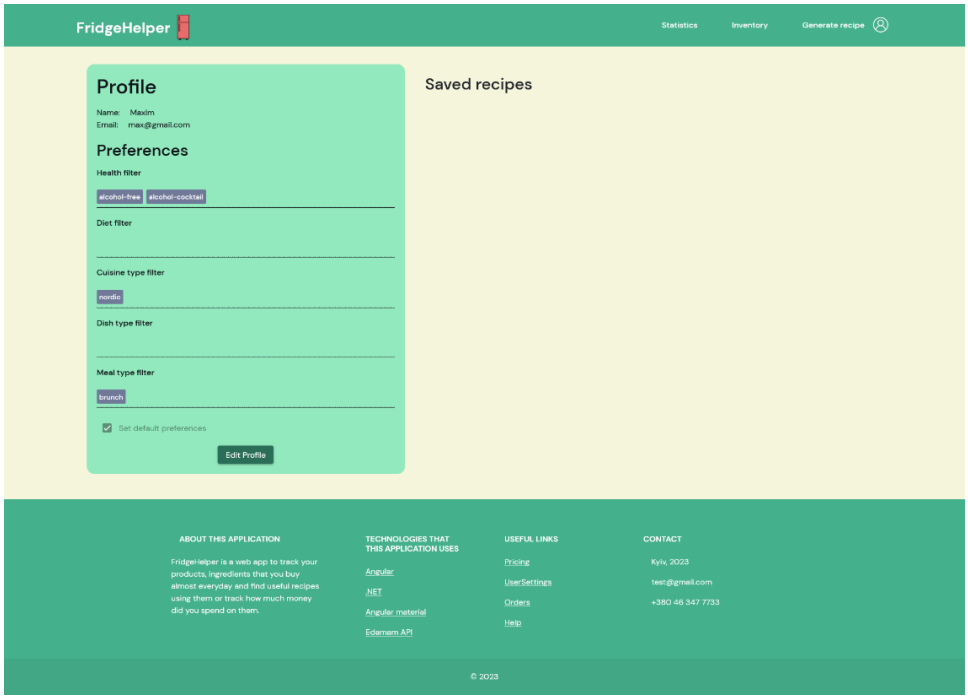


Рисунок 3.20 – Сторінка особистого профілю користувача

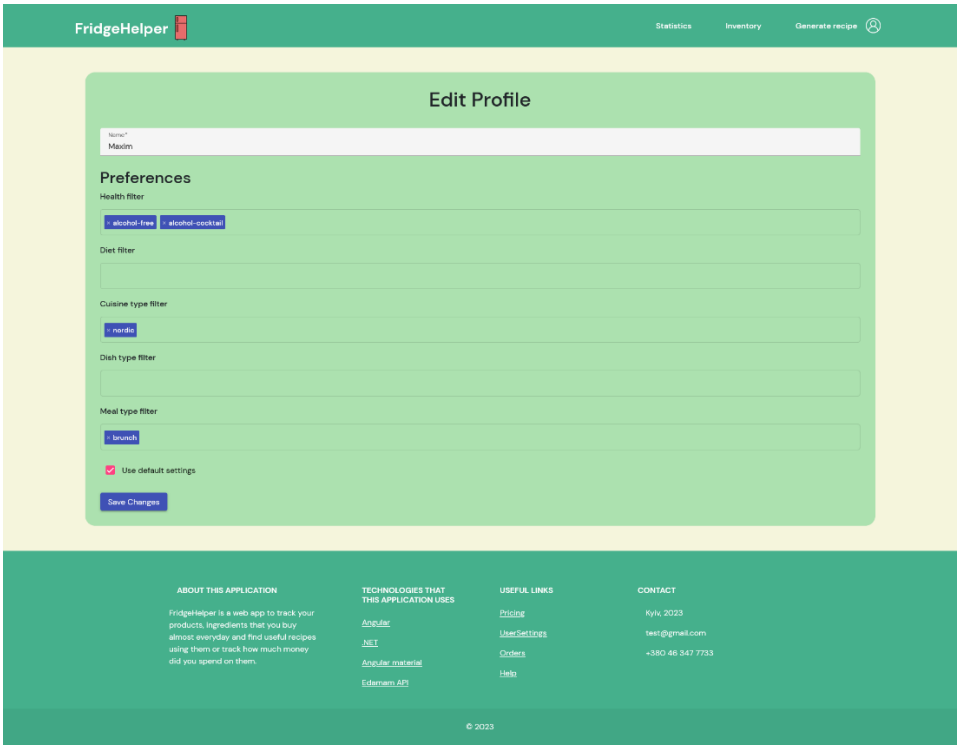


Рисунок 3.21 – Сторінка редагування профілю користувача

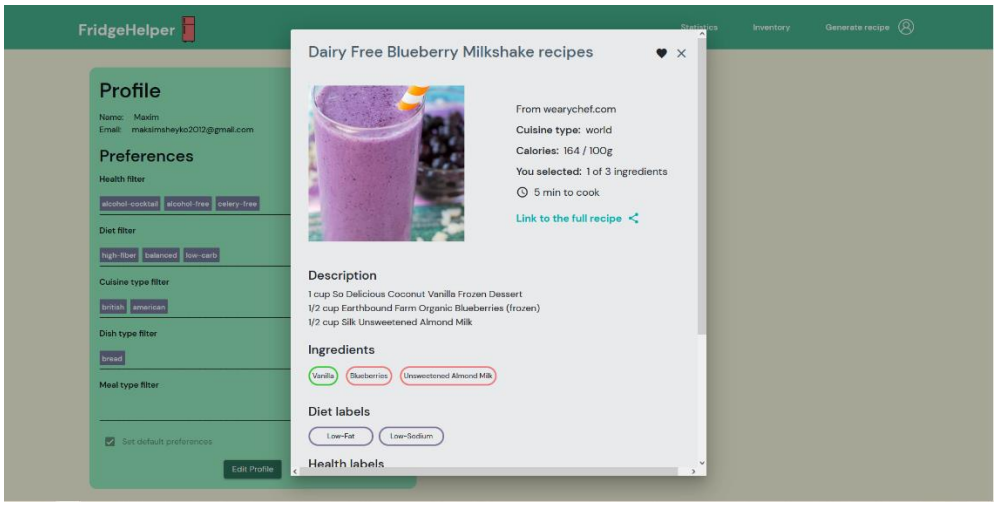


Рисунок 3.22 – Перегляд повної інформації про рецепт зі списку збережених

Висновки до розділу

Важливою складовою розробки програмного забезпечення є його тестування. Знаходження та виправлення недоліків або багів, що з’являються під час написання коду та реалізацій функціоналу є невід’ємною частиною процесу створення таких технологічних продуктів.

Програмне забезпечення, що було розроблено в рамках дипломного проєктування, було протестовано за допомогою різних підходів. З самого початку процесу розробки були підключені та налаштовані допоміжні засоби для написання коду, який має правильне форматування та використовує загальноприйняті підходи. В якості таких інструментів використовувалися ESLint та Prettier. Для проведено статичного аналізу якості написаного програмного забезпечення було обрано SonarQube. ПЗ показало хороші результати за метриками безпеки та обслуговування. Тестування правильності реалізацій компонентів застосунку проводилося за допомогою використання модульні тести та наскрізних мануальних тестів.

Розроблене програмне забезпечення розв’язує всі поставлені задачі, дотримується описаних функціональних та нефункціональних вимог. Головні функції, які були закладені в основу даного проєкту, а саме: авторизація та реєстрація користувачів, пошук рецептів за інгредієнтами та кулінарними вподобаннями, відстеження стану продуктів в інвентарі користувача,

додавання, видалення й редагування продуктів в інвентарі, збір та відображення статистики грошових витрат на продукти були реалізовані та перевірені на коректність своєї роботи.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання реалізованого програмного забезпечення було обрано платформу хмарного середовища від компанії Microsoft – Azure. Дану технологію було обрано через надійність та широкий спектр можливостей через інтеграції з іншими продуктами Microsoft. А, як вже було зазначено в розділі моделювання ПЗ, як сховище даних використовувалося SQL Server.

Спочатку в Azure було створено ресурсну групу під назвою ResourceGroupFridgeHelper. Далі, на основі даної ресурсної групи, розгорнуто екземпляр SQL Server. Обрані налаштування зображені на рисунку 4.1.

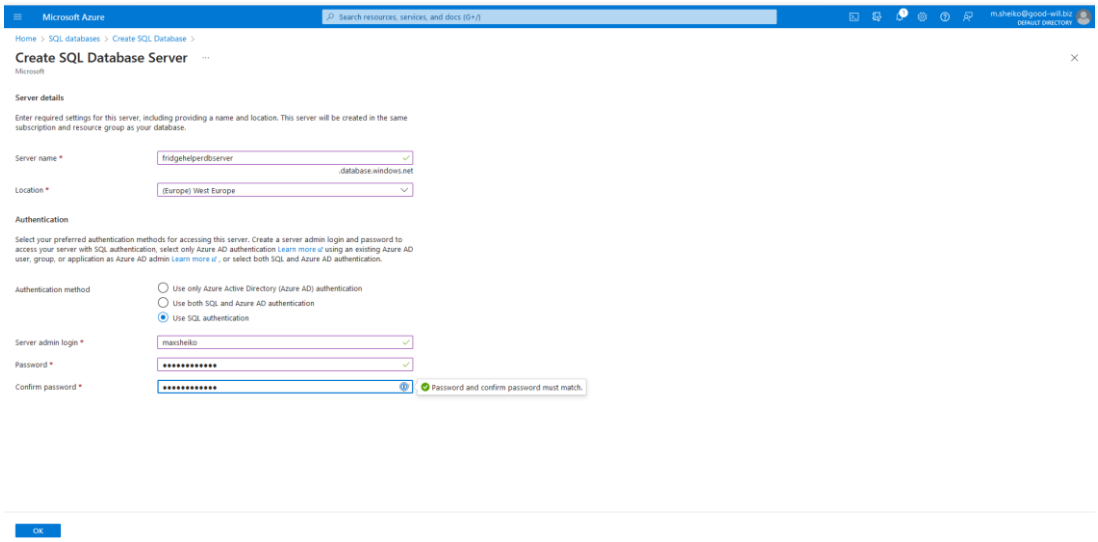


Рисунок 4.1 – Процес розгортання серверу бази даних

Після успішного налаштування серверу, до нього під’єднали SQL Database. Процес встановлення зображений на рисунку 4.2.

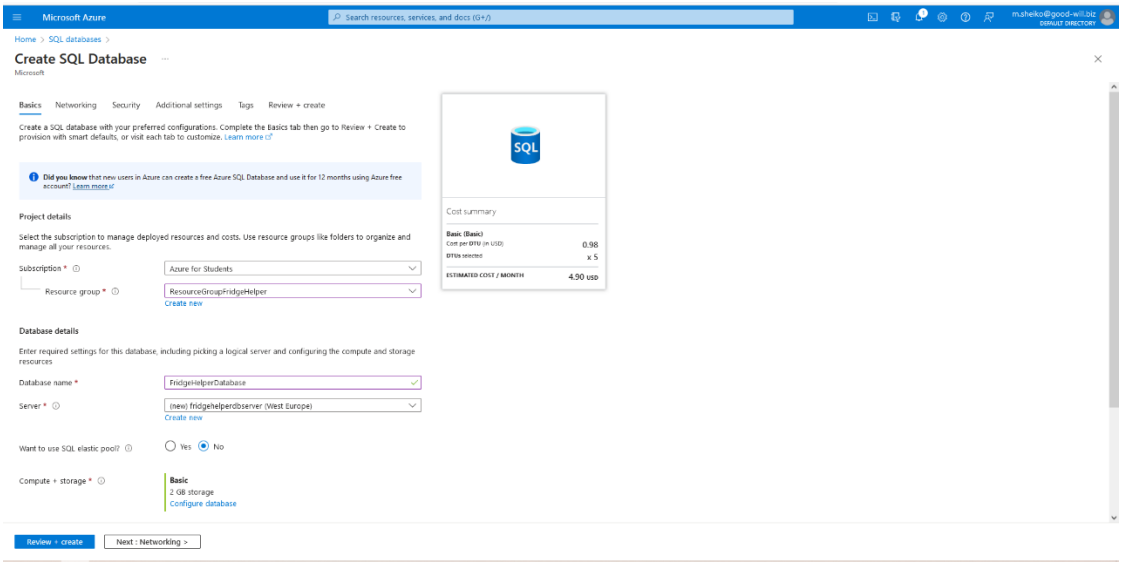


Рисунок 4.2 – Створення SQL Database

Далі було взято строку підключення до створеної бази даних та змінено конфігураційний файл серверної частини додатку. Створено екземпляр веб-застосунку в ресурній групі Azure. Завантажений конфігураційний файл зі створеного екземпляру та на основі нього опубліковано серверну частину за допомогою внутрішньої інтеграції Visual Studio та Azure. На рисунку 4.3 зображено процес розгортання серверної частини застосунку.

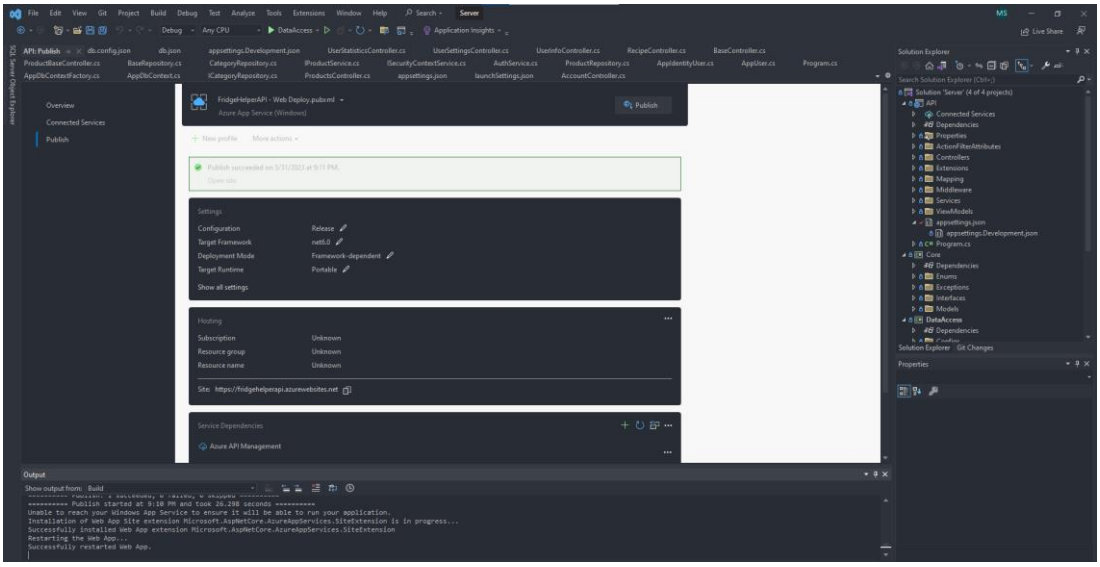


Рисунок 4.3 – Публікація серверної частини додатку до створеного екземпляру Web Application у Azure

Ще один екземпяр веб-додатку в ресурній групі хмарного середовища було створено для розгортання клієнтської частини застосунку. Налаштовані конфігураційні файлі на клієнтській частині додатку та за допомогою розширення до Visual Studio Code вона була опублікована на другому екземплярі Web Application. Проведено налаштування мереж, додані до білих списків IP адреси веб-додатків, встановлений фаєрвол та налаштовано політику CORS.

4.2 Підтримка програмного забезпечення

Розроблений програмний продукт має надалі підтримуватися та клієнти мають завжди отримувати нову версію веб-застосунку. Процес публікації нових версій додатку відбувається через взаємодію з хмарним середовищем Azure та оновленням збірок додатку, що там зберігаються.

За допомогою використання такої системи розгортання, користувачі завжди матимуть доступ до найбільш актуальної версії продукту.

Висновки до розділу

В якості платформи для розгортання програмного забезпечення було обрано хмарне середовище Microsoft Azure. Створено ресурсну групу та до неї були додані екземпляри веб-застосунків і бази даних. На створені веб-застосунки опубліковано клієнтську та серверну частину веб-додатку й налаштовано коректний зв’язок між ними. Також були змінені конфігураційні файлі частин додатку на продуктове середовище. Налаштовано підключення до бази даних, яка розгорнута у хмарі. Проведено аналіз та визначено принцип за яким розроблений програмний продукт буде надалі підтримуватися.

ВИСНОВКИ

Проблема збільшення кількості харчових відходів є дуже актуальною сьогодні, її вирішення потребує впровадження комплексних дій в різних сферах. Одним з найбільш важливих питань – є зміна культури споживання користувачів, а саме формування більш усвідомленого підходу до придбання та використання продуктових товарів. Програмне забезпечення, що було розроблено в рамках даної дипломної роботи, спрямоване на покращення культури споживання середньостатистичного користувача шляхом надання різноманітного функціоналу. Головними функціями додатку є:

– відстеження запасів продуктів в інвентарі. Повідомлення користувача про наближення кінцевого терміну придатності продукту;

– пошук рецептів з наявних інгредієнтів для їх споживання до звершення терміну придатності;

– урахування персональних кулінарних вподобань та дієтичних обмежень користувача;

– збір, формування та представлення статистики грошових витрат на продути, яка дає змогу переглянути пріоритетність придбання певних продуктів та зекономити гроші.

Під час дипломного проектування було покращено навички з розробки клієнтської та серверної частин веб-застосунків, хмарними середовищами, базами даних, ознайомився з низкою сучасних технологій розробки. Серверна частина застосунку написана за допомогою мови програмування C# та платформи .NET, клієнтська частина – з використанням мови програмування TypeScript й фреймворку Angular. Як інтегровані середовища розробки були обрані Visual Studio та Webstorm. Розгортання програмного забезпечення проводилося на платформі хмарного середовища Azure, як сховище даних використовувалося SQL Server. Додаток має чітку клієнт-серверну архітектуру. Використовувалися різноманітні шаблони архітектури під час проєктування та розробки серверної частини додатку.

Розроблений веб-застосунок вирішує всі поставлені задачі, коректність роботи протестована за допомогою написання модульних тестів, проведення ручних та наскрізних тестувань. В подальшому планується збільшити функціонал застостосунку та імплементувати інтеграції з різниим інтернет-магазинами України. Розроблене програмне забезпечення в даний час направлене на загальне користування, проте також може стати в нагоді колаборацiям магазин-ресторан для використання продуктів до їх псування першими контрагентами та через можливість зробити меню більш різноманітним закладам харчування.

Незважаючи на актуальність та важливість теми збільшення харчових відходів в світі, дане технологічне рішення – це лише допоміжний інструмент для вирішення цієї проблеми, проте найбільше все одно залежить від користувачів та нас з вами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) World Hunger Facts [Електронний ресурс]. 2023. URL: https://www.actionagainsthunger.org/the-hunger-crisis/world-hunger-facts/

2) Food Waste Facts [Електронний ресурс]. 2023. URL: https://www.ozharvest.org/food-waste-facts/

3) S. Safdie. Global Food Waste in 2023. *Greenly*. 2023. URL: https://greenly.earth/en-us/blog/ecology-news/global-food-waste-in-2022

4) H. Ritchie. Food waste is responsible for 6% of global greenhouse gas emissions. *Our World in Data*. 2020. URL: https://ourworldindata.org/food-waste-emissions

5) В Україні за три роки зросли витрати населення на продукти – IAE [Електронний ресурс]. 2023. URL: https://www.ukrinform.ua/rubric-economy/3672449-v-ukraini-za-tri-roki-zrosli-vitrati-naselenna-na-produkti-iae.html

6) Angular [Електронний ресурс]. 2023. URL: https://angular.io/

7) NET | Build. Test. Deploy [Електронний ресурс]. 2023. URL: https://dotnet.microsoft.com/en-us/

8) Build fast, responsive sites with Bootstrap. [Електронний ресурс]. 2023. https://getbootstrap.com/

9) Angular Material UI component library. [Електронний ресурс]. 2023. https://material.angular.io/

10) Overview of Entity Framework Core - EF Core - Microsoft Learn [Електронний ресурс]. 2023. URL: https://learn.microsoft.com/en-us/ef/core/

11) MealBoard - Meal Planner and Recipe Manager [Електронний ресурс]. 2023. URL: https://www.mealboard.com/

12) Cookpad - зробіть щоденне приготування страв цікавішим [Електронний ресурс]. – 2023. – Режим доступу: https://cookpad.com/ua/home

13) G. Ziemoński. Onion Architecture Is Interesting. *DZone*. 2017. URL: https://dzone.com/articles/onion-architecture-is-interesting

14) Onion Architecture / Clean Architecture. [Електронний ресурс]. 2021. URL: https://github.com/NilavPatel/dotnet-onion-architecture

15) Introduction to Identity on ASP.NET Core - Microsoft Learn [Електронний ресурс]. 2023. URL: https://learn.microsoft.com/enus/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio

16) About xUnit.net. [Електронний ресурс]. 2023. URL: https://xunit.net/

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015369126

Дата перевірки:
01.06.2023 15:01:30 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2023 15:39:20 EEST

ID користувача:
76913

Назва документа: **ІП-91_Шейко_ПЗ**

Кількість сторінок: **81** Кількість слів: **11992** Кількість символів: **96391** Розмір файлу: **8.04 MB** ID файлу: **1015035555**

6.99%
Схожість

Найбільша схожість: **2.28%** з джерелом з Бібліотеки (ID файлу: **1015035353**)

2.05% Джерела з Інтернету	104	Сторінка 83
6.9% Джерела з Бібліотеки	171	Сторінка 85

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0% Вилучення з Інтернету	4	Сторінка 86
0% Вилученого тексту з Бібліотеки	1	Сторінка 86

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Текст програми

КПІ.ІП-9128.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Максим ШЕЙКО

Київ – 2023

Файл ProductBaseController.cs

```
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class ProductBaseController : Controller
   {
      private readonly IProductService productsService;
      private readonly IMapper mapper;

      public ProductBaseController(
         IProductService productsService,
         IMapper mapper)
      {
         this.productsService = productsService;
         this.mapper = mapper;
      }

      [HttpGet("categories")]
      public async Task<ActionResult<IEnumerable<CategoryModel>>> GetAllCategories()
      {
         var data = await productsService.GetCategoriesAsync();
         return Ok(data);
      }

      [HttpGet("ingredients")]
      public async Task<ActionResult<IEnumerable<IngredientModel>>> GetAllIngredients()
      {
         var data = await productsService.GetIngredientsAsync();
         return Ok(data);
      }
   }
}
```

Файл ProductsController.cs

```
using API.Controllers.Base;
using API.ViewModels;
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using DataAccess.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Services.Realizations;
using Swashbuckle.AspNetCore.Annotations;

namespace API.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   [Authorize]
   public class ProductsController : BaseController
   {
```

```
private readonly IProductService productsService;
private readonly IMapper mapper;

public ProductsController(
   IProductService productsService,
   IMapper mapper)
{
   this.productsService = productsService;
   this.mapper = mapper;
}


[HttpGet("getById/{id}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
[SwaggerResponse(200, "Returns object containing a single product.")]
[SwaggerResponse(400, "The model state is invalid.")]
[SwaggerResponse(401, "An unauthorized request cannot be processed.")]
[SwaggerResponse(404, "No product has been found by the provided id.")]
[SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
public async Task<ActionResult<ProductViewModel>> GetProductById(Guid id)
{
   if (id == Guid.Empty)
   {
      return BadRequest("Product id is empty.");
   }

   var product = await this.productsService.GetProductByIdAsync(id);

   if (product == null)
   {
      return NotFound("A product wasn`t found.");
   }

   return Ok(mapper.Map<ProductViewModel>(product));
}


[HttpGet("getByUser")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
[SwaggerResponse(200, "Returns all producs in user inventory.")]
[SwaggerResponse(400, "The model state is invalid.")]
[SwaggerResponse(401, "An unauthorized request cannot be processed.")]
[SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
public async Task<ActionResult<IEnumerable<ProductViewModel>>> GetProductByUser()
{
   var userId = this.UserId;

   Console.WriteLine(userId);

   if (userId == Guid.Empty)
   {
      return BadRequest("Task id is empty.");
   }

   var products = await this.productsService.GetAvailableProductsByUserAsync(userId);
```

```
            return Ok(mapper.Map<IEnumerable<ProductViewModel>>(products));
        }


        [HttpPost]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(201, "Returns created object with product id.")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<ProductViewModel>> CreateProduct([FromBody] ProductViewModel
product)
        {
            Console.WriteLine(product.Price);
            var productModel = mapper.Map<ProductModel>(product);

            productModel.UserId = UserId;

            productModel = await productsService.CreateProductAsync(productModel);

            if (productModel == null)
            {
                return BadRequest();
            }

            var result = mapper.Map<ProductViewModel>(productModel);

            return CreatedAtAction(nameof(this.GetProductById), new { id = result.Id }, result);
        }

        [HttpDelete("{id}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns success result of product deletion.")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(403, "User cannot delete the product of another person.")]
        [SwaggerResponse(404, "No product found by the provided id.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult> DeleteProduct(Guid id)
        {
            var product = await productsService.GetProductByIdAsync(id);

            if (product == null)
            {
                return NotFound("Product wasn`t found.");
            }

            if (product.UserId != UserId)
            {
                return Forbid("Can`t delete the product that doesn`t related to current user.");
            }

            await productsService.DeleteProductAsync(product);
```

```
            return Ok();
        }

        [HttpPut("{id}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns success result of product updation.")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(403, "User cannot update product of another person.")]
        [SwaggerResponse(404, "No product found by the provided id.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult> UpdateProduct(
            Guid id,
            [FromBody] ProductViewModel product)
        {
            var productModel = await productsService.GetProductByIdAsync(id);

            if (productModel == null)
            {
                return NotFound("Product wasn`t found.");
            }

            if (productModel.UserId != UserId)
            {
                return Forbid("Can`t update the product that doesn`t related to current user.");
            }

            productModel = mapper.Map<ProductModel>(product);

            productModel = await productsService.UpdateProductAsync(productModel);

            if (productModel == null)
            {
                return BadRequest();
            }

            var result = mapper.Map<ProductViewModel>(productModel);

            return AcceptedAtAction(nameof(this.GetProductById), new { id = result.Id }, result);
        }
    }
}
```

Файл RecipeController.cs

```
using API.Controllers.Base;
using API.ViewModels;
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Services.Realizations;
using Swashbuckle.AspNetCore.Annotations;

namespace API.Controllers
{
```

```
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class RecipeController : BaseController
    {
        private readonly IRecipeService recipeService;
        private readonly IMapper mapper;
        public RecipeController(
            IRecipeService recipeService,
            IMapper mapper)
        {
            this.recipeService = recipeService;
            this.mapper = mapper;
        }

        [HttpPost]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(201, "Add recipe to saved recipes")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<RecipeViewModel>> AddRecipeToUserFavourites([FromBody]
RecipeViewModel recipe)
        {
            var recipeModel = mapper.Map<RecipeModel>(recipe);

            recipeModel = await recipeService.CreateRecipeAsync(recipeModel);

            if (recipeModel == null)
            {
                return BadRequest();
            }

            var result = mapper.Map<RecipeViewModel>(recipeModel);

            return CreatedAtAction(nameof(this.GetRecipeById), new { id = result.Id }, result);
        }

        [HttpGet("getById/{id}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns a recipe with specific id.")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<RecipeViewModel>> GetRecipeById(Guid recipeId)
        {
            if (recipeId == Guid.Empty)
            {
                return BadRequest("Recipe id is empty.");
            }

            var recipe = await recipeService.GetRecipeByIdAsync(recipeId);

            if (recipe == null)
            {
                return NotFound("A recipe wasn`t found.");
            }
```

```
            return Ok(mapper.Map<RecipeViewModel>(recipe));
        }


        [HttpGet("getSavedRecipes")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns all saved recipes for user")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<IEnumerable<RecipeViewModel>>> GetSavedRecipes()
        {
            var userId = this.UserId;

            if (userId == Guid.Empty)
            {
                return BadRequest("User id is empty.");
            }

            var recipes = await this.recipeService.GetSavedRecipesByUserId(UserId);

            return Ok(recipes);
        }


        [HttpDelete("{id}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns success result of recipe removal.")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(403, "User cannot delete the product of another person.")]
        [SwaggerResponse(404, "No product found by the provided id.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult> RemoveRecipe(Guid id)
        {
            await recipeService.DeleteRecipeAsync(id);
            return Ok();
        }
    }
}
```

Файл UserInfoController.cs

```
using API.Controllers.Base;
using API.ViewModels;
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace API.Controllers
```

```
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class UserInfoController : BaseController
    {
        private readonly IUserInfoService userService;
        private readonly IMapper mapper;

        public UserInfoController(
            IUserInfoService userService,
            IMapper mapper)
        {
            this.userService = userService;
            this.mapper = mapper;
        }

        [HttpGet("getUserInfo")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns info about user")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<UserInfoModel>> GetInfo()
        {
            var userId = this.UserId;

            if (userId == Guid.Empty)
            {
                return BadRequest("User id is empty.");
            }

            var info = await this.userService.GetUserInfo(userId);

            return Ok(mapper.Map<UserInfoModel>(info));
        }


        [HttpPut("getUserInfo")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns info about user")]
        [SwaggerResponse(400, "The model state is invalid.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<UserInfoModel>> ChangeInfo(UserInfoModel newUser)
        {
            var userId = this.UserId;

            if (userId == Guid.Empty)
            {
                return BadRequest("User id is empty.");
            }

            var info = await this.userService.UpdateUser(newUser);

            return Ok(mapper.Map<UserInfoModel>(info));
        }
```

```
    }
}
```

Файл UserSettingsController.cs

```
using API.Controllers.Base;
using API.ViewModels;
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class UserSettingsController : BaseController
    {
        private readonly IMapper mapper;
        private readonly IUserSettingsService settingsService;

        public UserSettingsController(IMapper mapper, IUserSettingsService settingsService)
        {
            this.mapper = mapper;
            this.settingsService = settingsService;
        }


        [HttpGet]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns an object containing user settings.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(404, "No settings found for the current user.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<SettingsViewModel>> GetUserSettings()
        {
            var settingsModel = await settingsService
                .GetUserSettingsAsync(UserId);

            if (settingsModel is null)
            {
                return NotFound("No settings found for the current user.");
            }

            var settingsViewModel = mapper.Map<SettingsViewModel>(settingsModel);
            return Ok(settingsViewModel);
        }



        [HttpPut("{id}")]
        [ProducesResponseType(StatusCodes.Status202Accepted)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
```

```
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(202, "Returns updated object containing user settings.")]
        [SwaggerResponse(400, "The model state is invalid or there is a mismatch.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(403, "User cannot update settings of another person.")]
        [SwaggerResponse(404, "No settings found by the provided id.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<SettingsViewModel>> UpdateSettings(
            Guid id,
            [FromBody] SettingsViewModel settingsViewModel)
        {
            if (UserId != settingsViewModel.UserId)
            {
                return Forbid();
            }

            if (id != settingsViewModel.Id)
            {
                return BadRequest("SettingsId mismatch.");
            }

            var settingsModel = mapper.Map<UserSettingsModel>(settingsViewModel);
            settingsModel = await settingsService.UpdateSettingsAsync(settingsModel);
            if (settingsModel is null)
            {
                return NotFound("No settings found by the provided id.");
            }

            var updatedViewModel = mapper.Map<SettingsViewModel>(settingsModel);
            return AcceptedAtAction(nameof(this.GetUserSettings), updatedViewModel);
        }
    }
}
```

Файл UserStatisticsController.cs

```
using API.Controllers.Base;
using AutoMapper;
using Core.Interfaces.IServices;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using API.ActionFilterAttributes;
using Swashbuckle.AspNetCore.Annotations;
using Core.Models.UserStatistics;

namespace API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class UserStatisticsController : BaseController
    {
        private readonly IMapper mapper;
        private readonly IStatisticsService statisticsService;

        public UserStatisticsController(
            IMapper mapper,
            IStatisticsService statisticsService)
        {
            this.mapper = mapper;
            this.statisticsService = statisticsService;
        }
```

```
        [HttpGet("daily/{day}")]
        [ValidateDate]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns an object containing daily user statistics.")]
        [SwaggerResponse(400, "The model state is invalid or validation error occured.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<UserStatisticsDay>> GetDailyStatistics(DateTime day)
        {
            var result = await statisticsService
                .GetDailyStatisticsAsync(UserId, day);

            return Ok(result);
        }

        [HttpGet("monthly/{year}/{month}")]
        [ValidateYear]
        [ValidateMonth]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns an object containing monthly user statistics.")]
        [SwaggerResponse(400, "The model state is invalid or validation error occured.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<UserStatisticsMonth>> GetMonthlyStatistics(int year, int month)
        {
            var result = await statisticsService
                .GetMonthlyStatisticsAsync(UserId, year, month);

            return Ok(result);
        }

        [HttpGet("annual/{year}")]
        [ValidateYear]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [SwaggerResponse(200, "Returns an object containing annual user statistics.")]
        [SwaggerResponse(400, "The model state is invalid or validation error occured.")]
        [SwaggerResponse(401, "An unauthorized request cannot be processed.")]
        [SwaggerResponse(500, "An unhandled exception occurred on the server while executing the request.")]
        public async Task<ActionResult<UserStatisticsYear>> GetAnnualStatistics(int year)
        {
            var result = await statisticsService
                .GetAnnualStatisticsAsync(UserId, year);

            return Ok(result);
        }
    }
}
```

Файл IProductService.cs

```
using Core.Models;
```

```
namespace Core.Interfaces.IServices
{
    public interface IProductService
    {
        public Task<ProductModel?> GetProductByIdAsync(Guid productId);
        public Task<IEnumerable<ProductModel>> GetAvailableProductsByUserAsync(Guid userId);
        public Task<ProductModel?> CreateProductAsync(ProductModel productModel);
        public Task DeleteProductAsync(ProductModel productModel);
        public Task<ProductModel?> UpdateProductAsync(ProductModel productModel);
        public Task<IEnumerable<CategoryModel>> GetCategoriesAsync();
        public Task<IEnumerable<IngredientModel>> GetIngredientsAsync();
    }
}
```

Файл RecipeModel.cs

```
using Core.Models.Base;

namespace Core.Models
{
    public class RecipeModel : BaseUserOrientedModel
    {
        public Guid Id { get; set; }
        public string? RecipeId { get; set; }
        public string? RecipeTitle { get; set; }
        public string? RecipeURL { get; set; }
    }
}
```

Файл ProductRepository.cs

```
using DataAccess.EF;
using DataAccess.Entities;
using DataAccess.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;

namespace DataAccess.Repositories.Realizations
{
    public class ProductRepository : BaseRepository<Product>, IProductRepository
    {
        public ProductRepository(AppDbContext context) : base(context)
        {

        }

        public async Task<Product?> FindOneTaskAsync(Guid id)
        {
            return await context.Set<Product>()
                .Where(t => t.Id == id)
                .FirstOrDefaultAsync();
        }

        public async Task<IEnumerable<Product>> FindAllAsync(Expression<Func<Product, bool>> predicate)
        {
            return await context.Set<Product>()
                .Where(predicate)
                .ToListAsync();
        }
    }
```

```
}
```

Файл ProductService.cs

```
using AutoMapper;
using Core.Enums;
using Core.Interfaces.IServices;
using Core.Models;
using Core.Models.Base;
using DataAccess.Entities;
using DataAccess.Repositories.Interfaces;

namespace Services.Realizations
{
    public class ProductService : IProductService
    {
        private readonly IProductRepository _productRepo;
        private readonly ICategoryRepository _categoryRepo;
        private readonly IIngredientRepository _ingredientRepo;
        private readonly IMapper _mapper;

        public ProductService(
            IProductRepository productRepo,
            ICategoryRepository categoryRepo,
            IIngredientRepository ingredientRepo,
            IMapper mapper)
        {
            _productRepo = productRepo;
            _categoryRepo = categoryRepo;
            _ingredientRepo = ingredientRepo;
            _mapper = mapper;
        }

        public async Task<IEnumerable<ProductModel>> GetAvailableProductsByUserAsync(Guid userId)
        {
            var userProducts = await _productRepo.FindAllAsync(t => t.UserId == userId && t.IsAvailable == true);
            return _mapper.Map<IEnumerable<ProductModel>>(userProducts);
        }

        public async Task<ProductModel?> GetProductByIdAsync(Guid productId)
        {
            var product = await _productRepo.FindOneTaskAsync(productId);

            if (product == null)
            {
                return null;
            }

            return _mapper.Map<ProductModel>(product);
        }


        public async Task<ProductModel?> CreateProductAsync(ProductModel productModel)
        {
            var product = _mapper.Map<Product>(productModel);

            await _productRepo.AddAsync(product);
            await _productRepo.SaveChangesAsync();

            return _mapper.Map<ProductModel>(product);
```

```
        }


        public async Task DeleteProductAsync(ProductModel productModel)
        {
            if (productModel == null)
            {
                throw new ArgumentNullException(nameof(productModel), "Can`t be Null.");
            }

            var product = await _productRepo.GetByIdAsync(productModel.Id);

            if (product == null)
            {
                throw new InvalidOperationException("Can`t find product in db.");
            }

            product.IsAvailable = false;

            _productRepo.Update(product);
            await _productRepo.SaveChangesAsync();
        }

        public async Task<ProductModel?> UpdateProductAsync(ProductModel productModel)
        {
            if (productModel == null)
            {
                throw new ArgumentNullException(nameof(productModel), "Can`t be Null.");
            }

            var product = await _productRepo.GetByIdAsync(productModel.Id);

            if (product == null)
            {
                return null;
            }

            product.IngredientId = productModel.IngredientId;
            product.ExpirationDate = productModel.ExpirationDate;
            product.Price = productModel.Price;
            product.IsAvailable = productModel.IsAvailable;

            _productRepo.Update(product);
            await _productRepo.SaveChangesAsync();

            return _mapper.Map<ProductModel>(product);
        }


        public async Task<IEnumerable<CategoryModel>> GetCategoriesAsync()
        {
            var result = await _categoryRepo.GetAllAsync();
            return _mapper.Map<IEnumerable<CategoryModel>>(result);
        }

        public async Task<IEnumerable<IngredientModel>> GetIngredientsAsync()
        {
            var result = await _ingredientRepo.GetAllAsync();
            return _mapper.Map<IEnumerable<IngredientModel>>(result);
        }

    }
}
```

Файл RecipeService.cs

```
using AutoMapper;
using Core.Enums;
using Core.Interfaces.IServices;
using Core.Models;
using DataAccess.Entities;
using DataAccess.Repositories.Interfaces;

namespace Services.Realizations
{
    public class RecipeService : IRecipeService
    {
        private readonly IRecipeRepository _recipeRepo;
        private readonly IMapper _mapper;

        public RecipeService(
            IRecipeRepository recipeRepo,
            IMapper mapper)
        {
            _mapper = mapper;
            _recipeRepo = recipeRepo;
        }


        public async Task<RecipeModel?> CreateRecipeAsync(RecipeModel recipeModel)
        {
            if (recipeModel == null)
            {
                throw new ArgumentNullException(nameof(recipeModel), "Model can`t be null.");
            }

            var checkRecipe = await _recipeRepo.FindAsync(r => r.RecipeId == recipeModel.RecipeId
                && r.UserId == recipeModel.UserId);

            if (checkRecipe.FirstOrDefault() != null)
            {
                throw new ArgumentNullException(nameof(recipeModel), "This recipe is already saved");
            }

            var recipe = _mapper.Map<Recipe>(recipeModel);

            await _recipeRepo.AddAsync(recipe);
            await _recipeRepo.SaveChangesAsync();

            return _mapper.Map<RecipeModel>(recipe);
        }


        public async Task<RecipeModel?> UpdateRecipeAsync(RecipeModel recipeModel)
        {
            if (recipeModel == null)
            {
                throw new ArgumentNullException(nameof(recipeModel), "Model can`t be null.");
            }

            var recipe = await _recipeRepo.GetByIdAsync(recipeModel.Id);

            if (recipe == null)
            {
```

```
            return null;
        }

        _recipeRepo.Update(_mapper.Map<Recipe>(recipeModel));
        await _recipeRepo.SaveChangesAsync();

        return recipeModel;

    }

    public async Task DeleteRecipeAsync(Guid id)
    {
        if (id == Guid.Empty)
        {
            throw new ArgumentNullException("Id can not be empty");
        }

        var recipe = await _recipeRepo.GetByIdAsync(id);

        if (recipe == null)
        {
            throw new InvalidOperationException("Can`t find recipe in db.");
        }

        _recipeRepo.Remove(recipe);
        await _recipeRepo.SaveChangesAsync();

        return;
    }

    public async Task<RecipeModel?> GetRecipeByIdAsync(Guid id)
    {
        var recipe = await this._recipeRepo.FindAsync(r => r.Id == id);
        return _mapper.Map<RecipeModel>(recipe);
    }

    public async Task<IEnumerable<RecipeModel?>> GetSavedRecipesByUserId(Guid userId)
    {
        var savedRecipes = await this._recipeRepo.FindAllAsync(x => x.UserId == userId);
        return _mapper.Map<IEnumerable<RecipeModel?>>(savedRecipes);
    }
  }
}
```

Файл SettingsService.cs

```
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using DataAccess.Entities;
using DataAccess.Repositories.Interfaces;

namespace Services.Realizations
{
  public class SettingsService : IUserSettingsService
  {
    private readonly IMapper mapper;
    private readonly ISettingsRepository settingsRepository;

    public SettingsService(
```

```
    IMapper mapper,
    ISettingsRepository settingsRepository)
{
    this.mapper = mapper;
    this.settingsRepository = settingsRepository;
}

public async Task<UserSettingsModel> CreateSettingsAsync(UserSettingsModel settingsModel)
{
    if (settingsModel is null)
    {
        throw new ArgumentNullException(nameof(settingsModel));
    }

    var settings = mapper.Map<Settings>(settingsModel);

    await RemoveExistingSettings(settings.UserId);
    await settingsRepository.AddAsync(settings);
    await settingsRepository.SaveChangesAsync();

    return mapper.Map<UserSettingsModel>(settings);
}

public async Task<UserSettingsModel?> GetSettingsAsync(Guid id)
{
    var settings = await settingsRepository.GetByIdAsync(id);
    return mapper.Map<UserSettingsModel>(settings);
}

public async Task<UserSettingsModel?> GetUserSettingsAsync(Guid userId)
{
    if (userId == Guid.Empty)
    {
        throw new ArgumentException("Argument value is all zeroes.", nameof(userId));
    }

    var settings = await settingsRepository
        .FindAsync(s => s.UserId == userId);

    return mapper.Map<UserSettingsModel>(settings?.FirstOrDefault());
}

public async Task<UserSettingsModel?> UpdateSettingsAsync(UserSettingsModel settingsModel)
{
    if (settingsModel is null)
    {
        throw new ArgumentNullException(nameof(settingsModel));
    }

    if (!await settingsRepository
        .HasByIdAsync(settingsModel.Id))
    {
        return null;
    }

    var settings = mapper.Map<Settings>(settingsModel);

    settingsRepository.Update(settings);
    await settingsRepository.SaveChangesAsync();

    return mapper.Map<UserSettingsModel>(settings);
}
```

```
        private async Task RemoveExistingSettings(Guid userId)
        {
            var settings = await settingsRepository
                .FindAsync(s => s.UserId == userId);

            if (settings?.Count() > 0)
            {
                settingsRepository.RemoveRange(settings);
                await settingsRepository.SaveChangesAsync();
            }
        }
    }
}
```

Файл StatisticsService.cs

```
using AutoMapper;
using Core.Enums;
using Core.Interfaces.IServices;
using Core.Models;
using Core.Models.UserStatistics;
using DataAccess.Repositories.Interfaces;
using System;

namespace Services.Realizations
{
    public class StatisticsService : IStatisticsService
    {
        private readonly IProductRepository _productRepo;
        private readonly IMapper _mapper;

        public StatisticsService(
            IProductRepository productRepository,
            IMapper mapper
        )  {
            this._productRepo = productRepository;
            this._mapper = mapper;
        }


        public async Task<UserStatisticsDay> GetDailyStatisticsAsync(Guid userId, DateTime date)
        {
            var dailyStatistics = new UserStatisticsDay
            {
                UserId = userId,
                Day = date
            };

            var nextDay = date.AddDays(1);

            var userProducts = await _productRepo.FindAllAsync(t => t.UserId == userId
            && t.CreationDate >= date && t.CreationDate < nextDay );

            var result = _mapper.Map<IEnumerable<ProductModel>>(userProducts).ToList();
            dailyStatistics.ProductBought = result;
            dailyStatistics.MoneySpent = result.Select(x => x.Price).ToList().Sum();

            return dailyStatistics;
        }
```

```
        public async Task<UserStatisticsMonth> GetMonthlyStatisticsAsync(Guid userId, int year, int month)
        {
            var monthlyStatistics = new UserStatisticsMonth
            {
                UserId = userId,
                Month = (Month) month,

            };

            var userProducts = await _productRepo.FindAllAsync(t => t.UserId == userId
            && t.CreationDate.Year == year && t.CreationDate.Month >= month && t.CreationDate.Month < month +
1);

            var result = _mapper.Map<IEnumerable<ProductModel>>(userProducts).ToList();
            monthlyStatistics.ProductBought = result;
            monthlyStatistics.MoneySpent = result.Select(x => x.Price).ToList().Sum();

            return monthlyStatistics;
        }

        public async Task<UserStatisticsYear> GetAnnualStatisticsAsync(Guid userId, int year)
        {
            var yearStatistics = new UserStatisticsYear
            {
                UserId = userId,
                Year = year,
                AnalyticsPerMonths = new List<UserStatisticsMonth> { }
            };

            foreach (int month in Enum.GetValues(typeof(Month)))
            {
                var analyticsPerMonth = await this.GetMonthlyStatisticsAsync(userId, year, month);
                yearStatistics.AnalyticsPerMonths.Add(analyticsPerMonth);
            }

            return yearStatistics;
        }

    }
}
```

Файл UserInfoService.cs

```
using AutoMapper;
using Core.Interfaces.IServices;
using Core.Models;
using DataAccess.Entities;
using DataAccess.Repositories.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Services.Realizations
{
    public class UserInfoService : IUserInfoService
    {

        private readonly IMapper mapper;
        private readonly IUserRepository userRepository;
```

```
        public UserInfoService(
            IMapper mapper,
            IUserRepository userRepository)
        {
            this.mapper = mapper;
            this.userRepository = userRepository;
        }
        public async Task<UserInfoModel> GetUserInfo(Guid userId)
        {
            var result = await userRepository.GetByIdAsync(userId);
            return mapper.Map<UserInfoModel>(result);
        }

        public async Task<UserModel?> GetUser(Guid userId)
        {
            var result = await userRepository.GetByIdAsync(userId);
            return mapper.Map<UserModel>(result);
        }

        public async Task<UserModel?> UpdateUser(UserInfoModel user)
        {
            var updatedUser = await userRepository.FindAsync(u => u.Email == user.Email);
            var newUser = updatedUser.FirstOrDefault();
            Console.WriteLine(user.Name);
            newUser.Name = user.Name;
            userRepository.Update(newUser);
            await userRepository.SaveChangesAsync();
            return mapper.Map<UserModel>(newUser); ;

        }
    }
}
```

Файл recipe-search-result-page.component.ts

```
import {Component, OnInit} from '@angular/core';
import test from "../../testResult.json";
import {RecipePopUpComponent} from "./recipe-pop-up/recipe-pop-up.component";
import {MatDialog} from "@angular/material/dialog";
import {similarity} from "../shared-module/validation/similiarity";
import {count} from "rxjs";
import {ProductService} from "../shared-module/services/product.service";
import {AuthService} from "../shared-module/auth/auth.service";
import {IngredientFullInfo} from "../shared-module/types/ingredient-full-info";


interface Food {
  value: string;
  viewValue: string;
}


@Component({
  selector: 'app-recipes-search-result-page',
```

```
  templateUrl: './recipes-search-result-page.component.html',
  styleUrls: ['./recipes-search-result-page.component.scss']
})

export class RecipesSearchResultPageComponent implements OnInit {

  selectedIngredients;

  requestResult: any = [];
  count: number;
  pageIndex: number = 0;
  pageSize: number = 12;
  recipes: any[] = [];
  sortedRecipes: any;

  allIngredients: IngredientFullInfo[] = [];

  constructor(
    private dialogRef: MatDialog,
    private productService: ProductService,
    private authService: AuthService
  ) {
    this.selectedIngredients = history.state["request"].q.split(',');
    this.requestResult = history.state["recipes"];
    console.log(this.selectedIngredients);



    //this.from = this.requestResult.from;
    //this.to = this.requestResult.to;
    this.count = 20;//this.requestResult.count;
    //this.nextPageUrl = this.requestResult._links.nextPageUrl;
    console.log(this.requestResult);
    this.requestResult.forEach((x: any) => this.recipes.push({
      recipe: x.recipe,
      matchNumber: this.checkMatchIngredients(x.recipe),
      matchRate: this.checkMatchIngredients(x.recipe) / x.recipe.ingredients.length,
    }));
    this.recipes = this.recipes.filter((r: any) => r.matchNumber > 0);
    const uniqueRecipes = [...new Set(this.recipes.map((item: any) => [item.recipe.label, item])).values()];
    console.log(uniqueRecipes);
    this.recipes.sort(this.compareRecipes);
```

```
  this.count = this.recipes.length;

  console.log(this.count);
}

ngOnInit(): void {
   /*if (this.authService.isAuthenticated()) {
     this.productService.allIngredientsChanged.subscribe(
       (ingredients) => (this.allIngredients = ingredients)
     );
     this.productService.changeAllIngredients();
   }*/
}

handlePageEvent($event: any) {
  this.pageSize = $event.pageSize;
  this.pageIndex = $event.pageIndex;
}

openFull(recipe: any) {
  const body = {
    "selectedIngredients": this.selectedIngredients,
    "recipe": recipe,
  }
  this.dialogRef.open(RecipePopUpComponent, {
    data: body,
    autoFocus: false,
    maxHeight: '90vh'
  });
}

checkMatchIngredients(recipeInfo: any) {
  const ingredients: any[] = [];
  let ingredientsMatch = 0;
  recipeInfo.ingredients.forEach((x: any) => ingredients.push(x.food));

  //console.log(recipeInfo.label);

  for (let ingredient of this.selectedIngredients) {
    for (let item of ingredients) {
      //console.log("ingredient: " + item + "  " + similarity(item, ingredient))
```

```
        if (similarity(item, ingredient) > 0.6) {
          ingredientsMatch += 1;
        }
      }
    }
    return ingredientsMatch;
  }

  compareRecipes(a: any, b: any) {
    if (a.matchRate > b.matchRate) return -1;
    if (a.matchRate < b.matchRate) return 1;
    return 0;
  }

}
```

Файл recipe-pop-up.component.ts

```
import {Component, Inject} from '@angular/core';
import {MAT_DIALOG_DATA, MatDialog} from "@angular/material/dialog";
import {AuthService} from "../../shared-module/auth/auth.service";
import {ConnectionService} from "../../shared-module/services/connection.service";
import {RecipeService} from "../../shared-module/services/recipe.service";
import {RecipeInfo} from "../../shared-module/types/recipe-info";
import {Guid} from "../../shared-module/types/guid";
import {Product} from "../../shared-module/types/product";
import {SavedRecipe} from "../../shared-module/types/saved-recipe";
import {ProductService} from "../../shared-module/services/product.service";
import {IngredientFullInfo} from "../../shared-module/types/ingredient-full-info";
import {similarity} from "../../shared-module/validation/similiarity";

@Component({
  selector: 'app-recipe-pop-up',
  templateUrl: './recipe-pop-up.component.html',
  styleUrls: ['./recipe-pop-up.component.scss']
})
export class RecipePopUpComponent {

  public recipeInfo;
  public selectedItems;
  public caloriesToShow;
```

```
public matchNumber;
public recipeId: string;
public ingredientsInInventory: string[] = [];

public isLoggedIn = false;
public connected = true;
public isInFavourites = false;

private savedRecipes: SavedRecipe[] = [];
private allIngredients: IngredientFullInfo[] = [];
private products: Product[] = [];

private uniqueProductNames: string[] = [];

constructor(
  private authService: AuthService,
  private recipeService: RecipeService,
  private connection: ConnectionService,
  private dialogRef: MatDialog,
  private productService: ProductService,
  @Inject(MAT_DIALOG_DATA) private data: any
) {
  this.recipeInfo = data.recipe.recipe;
  this.selectedItems = data.selectedIngredients;
  this.matchNumber = data.recipe.matchNumber;
  this.caloriesToShow = Math.round(100 * this.recipeInfo.calories / this.recipeInfo.totalWeight);

  const recipeUri: string = this.recipeInfo.uri.toString();
  this.recipeId = recipeUri.substring(recipeUri.indexOf("_") + 1);

  this.authService.authStatus.subscribe({
     next: (result) => {
       console.log(result);
     },
     error: (error) => {
       console.log(error);
     }
   })
  this.connected = this.connection.connected;
  connection.Changes.subscribe((state) => (this.connected = state));

  this.isLoggedIn = this.authService.isAuthenticated();
```

```
    if (this.isLoggedIn) {
      this.recipeService.savedRecipesChanged.subscribe({
        next: (recipes: SavedRecipe[]) => {
          this.savedRecipes = recipes;
        },
        error: (error: Error) => {
          console.log('Error occurred while adding recipe to saved: ' + error.message);
        },
      });
    }

    if (this.isLoggedIn) {
      this.productService.allIngredientsChanged.subscribe(
        (ingredients) => {
          this.allIngredients = ingredients;
        });

      this.productService.userInventoryChanged.subscribe({
        next: (products: Product[]) => {
          this.products = products;
        },
        error: (error: Error) => {
          console.log('Error occurred while getting existing ingredients in recipe pop-up' + error.message);
        },
      });
    }
  }

  ngOnInit() {
    const recipesIds = this.savedRecipes.map(x => x.recipeId);
    if (recipesIds.includes(this.recipeId)) {
      this.isInFavourites = true;
    }

    const ingredientsInRecipe = this.recipeInfo.ingredients.map((x: any) => x.food);
    this.ingredientsInInventory.push(...this.products.map(x => this.allIngredients.find(r => r.id ===
x.ingredientId)!.value));
    this.uniqueProductNames = [...new Set(this.ingredientsInInventory)];
  }

  checkForPresence(name: string) {
```

```
  for (let ingredient of this.uniqueProductNames) {
    if (similarity(ingredient, name) > 0.5) return true
  }
  return false;
}

addToFavourites() {
  const body = <RecipeInfo> {};
  body.id = Guid.empty;
  body.recipeId = this.recipeId;
  body.userId = this.authService.getUserId();
  body.recipeUrl = this.recipeInfo.url;
  body.recipeTitle = this.recipeInfo.label;

  this.recipeService.addRecipeToFavourites(body).subscribe({
    next: (data) => {
      this.recipeService.getRecipesByUser().subscribe({
        next: (recipes: SavedRecipe[]) => {
          this.recipeService.savedRecipes = recipes;
          this.recipeService.changeSavedRecipes();
          this.isInFavourites = true;
        },
        error: (error: Error) => {
          console.log('Error occurred while adding recipe to saved: ' + error.message);
        },
      });
    },
    error: (error) => {
      console.log(error);
    },
  });
}

removeFromFavourites() {
  const recipeId = this.savedRecipes.find(r => r.recipeId === this.recipeId)!.id;
  this.recipeService.removeRecipe(recipeId).subscribe({
    next: () => {
      this.recipeService.getRecipesByUser().subscribe({
        next: (recipes: SavedRecipe[]) => {
          this.recipeService.savedRecipes = recipes;
          this.recipeService.changeSavedRecipes();
          this.isInFavourites = false;
```

```
        },
        error: (error: Error) => {
          console.log('Error occurred while removing recipe from favourites: ' + error.message);
        },
       });
     }
   });
 }
}
```

Файл ingredient-selection-section.component.ts

```
import {Component, OnInit, ViewChild, AfterViewInit, Input} from '@angular/core';
import { FormControl } from "@angular/forms";
import db from "../../../db.json";
import { Ingredient } from "../../shared-module/types/ingredient-info";
import { Category } from "../../shared-module/types/category-info";
import { AdditionalFiltersSectionComponent } from "../additional-filters-section/additional-filters-
section.component";
import { RecipeService } from "../../shared-module/services/recipe.service";
import {RecipeRequest} from "../../shared-module/types/recipe-request";
import {NavigationExtras, Router} from "@angular/router";
import {ProductService} from "../../shared-module/services/product.service";
import {IngredientFullInfo} from "../../shared-module/types/ingredient-full-info";
import {environment} from "../../../environments/environment";
import {UserSettings} from "../../shared-module/types/user-settings";
import {concatAll, EMPTY, expand, forkJoin, map, mergeAll, Observable, reduce, switchMap, take} from "rxjs";
import * as http from "http";
import {HttpClient} from "@angular/common/http";

@Component({
  selector: 'app-ingredient-selection-section',
  templateUrl: './ingredient-selection-section.component.html',
  styleUrls: ['./ingredient-selection-section.component.scss']
})

export class IngredientSelectionSectionComponent implements OnInit {

  @Input() isLoggedIn!: boolean;
  @Input() selectedIngredients!: string[];
  categories: Category[] = [];
```

```
allFilteredCategories: Category[] = [];
allIngredients: IngredientFullInfo[] = [];
result = new FormControl();
public showCountOfPost = 10;
searchText: string = '';
allergies: string[] = [];
diets: string[] = [];
cuisineTypes: string[] = [];
mealTypes: string[] = [];
dishTypes: string[] = [];

selectedAllergies: string[] = [];
selectedDiets: string[] = [];
selectedMeals: string[] = [];
selectedDishes: string[] = [];
selectedCuisine: string[] = [];

recipes: any = [];

selectedError: boolean = false;


constructor(
  private recipeService: RecipeService,
  private productService: ProductService,
  private router: Router,
  private http: HttpClient
) {
  this.allergies = productService.getAllergies();
  this.cuisineTypes = productService.getCuisineTypes();
  this.mealTypes = productService.getMeals();
  this.dishTypes = productService.getDishes();
  this.diets = productService.getDiets();

}

ngOnInit() {
  this.productService.allCategoriesChanged.subscribe(
    (categories) => {
      this.categories = categories;
      this.allFilteredCategories = this.categories;
    }
```

```
    );
    this.productService.allIngredientsChanged.subscribe(
      (ingredients) => (this.allIngredients = ingredients)
    );
    this.productService.changeAllIngredients();
    this.productService.changeAllCategories();
  }


  toggleSelection(ingredient: string) {
    if (this.selectedIngredients.includes(ingredient)) {
      const index = this.selectedIngredients.indexOf(ingredient);
      this.selectedIngredients.splice(index, 1);
      this.selectedIngredients = [...this.selectedIngredients];
    } else {
      if (this.selectedError) {
        this.selectedError = false;
      }
      this.selectedIngredients = [...this.selectedIngredients, ingredient];
    }
  }

  filterIngredients() {
    //this.searchText = this.searchText.toLocaleLowerCase();
    const filteredIngredients = this.allIngredients.filter(i => i.value.includes(this.searchText.toLocaleLowerCase()));
    const uniqueCategories = [...new Set(filteredIngredients.map(item => item.categoryId))];

    this.allFilteredCategories = this.categories.filter(c => uniqueCategories.includes(c.id));
  }


  findIngedientsByCategory(category: string, searchText: string) {
    return this.allIngredients.filter(x => x.categoryId === this.categories.find(c => c.value === category)?.id &&
x.value.includes(searchText));
  }

  groupByFn = (item: IngredientFullInfo) => item.categoryId;
  groupValueFn = (_: string, children: any[]) => ({ value: children[0].categoryId });


  findCategoryValueName(categoryId: string) {
    return this.categories.find(c => c.id === categoryId)?.value;
```

```
}

findCategoryById(categoryId: string) {
 return this.categories.find(c => c.id === categoryId);
}


removeFromSelected(ingredient: string) {
 const index = this.selectedIngredients.indexOf(ingredient);
 this.selectedIngredients.splice(index, 1);
 this.selectedIngredients = [...this.selectedIngredients];
}

sendRequest() {
 if (this.selectedIngredients.length === 0) {
  this.selectedError = true;
  return;
 }
 const request: RecipeRequest = {
  type: "public",
  q: this.selectedIngredients.join(","),
  app_id: environment.appId,
  app_key: environment.appKey,
  cuisineType: this.selectedCuisine,
  diet: this.selectedDiets,
  dishType: this.selectedDishes,
  health: this.selectedAllergies,
  mealType: this.selectedMeals,
 }

 console.log(request);

 /*this.recipeService.getAllRecipes(request).subscribe((data: any) => {
  console.log(data);
  this.recipes = data;
  const navigationExtras: NavigationExtras = {
   state: { recipes: this.recipes, request },
  };
  this.router.navigate(["recipe-search-result"], navigationExtras);
 });*/

 this.recipeService.getAllRecipes(request).pipe(
```

```
    expand((response) => {
      return response._links.next ? this.http.get(response._links.next.href) : EMPTY
    }),
    take(3),
    reduce((acc, current: any) => acc.concat(current.hits), [])
  ).subscribe((data) => {
    console.log(data);
    this.recipes = data;
    const navigationExtras: NavigationExtras = {
      state: { recipes: this.recipes, request },
    };
    this.router.navigate(["recipe-search-result"], navigationExtras);
  });
 }


 selectAllergies($event: string[]) {
  this.selectedAllergies = $event;
 }

 selectDiets($event: string[]) {
  this.selectedDiets = $event;
 }

 selectDishes($event: string[]) {
  this.selectedDishes = $event;
 }

 selectMeals($event: string[]) {
  this.selectedMeals = $event;
 }

 selectCuisine($event: string[]) {
  this.selectedCuisine = $event;
 }
}
```

Файл additional-filters-section.component.ts

```
import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
import {UserSettings} from "../../shared-module/types/user-settings";
```

```
import {SettingsService} from "../../shared-module/services/settings.service";

@Component({
  selector: 'app-additional-filters-section',
  templateUrl: './additional-filters-section.component.html',
  styleUrls: ['./additional-filters-section.component.scss']
})
export class AdditionalFiltersSectionComponent implements OnInit {
  @Input() allergies!: string[];
  @Input() diets!: string[];
  @Input() mealTypes!: string[];
  @Input() dishType!: string[];
  @Input() cuisineType!: string[]
  @Input() isLogged!: boolean;
  userSettings: UserSettings = <UserSettings> {};

  @Input() selectedAllergies: string[] = [];
  @Output() selectedAllergiesChange = new EventEmitter<string[]>();

  @Input() selectedDiets: string[] = [];
  @Output() selectedDietsChange = new EventEmitter<string[]>();

  @Input() selectedMeals: string[] = [];
  @Output() selectedMealsChange = new EventEmitter<string[]>();

  @Input() selectedDishes: string[] = [];
  @Output() selectedDishesChange = new EventEmitter<string[]>();

  @Input() selectedCuisine: string[] = [];
  @Output() selectedCuisineChange = new EventEmitter<string[]>();

  constructor(
    private settingsService: SettingsService
  ) {
  }

  ngOnInit(): void {
    if (this.isLogged) {
      this.settingsService.getFromServer().subscribe({
        next: (settings) => {
          this.userSettings = settings;
```

```
        if (settings.useDefault) {
          if (this.userSettings.allergyType === "") {
            this.selectedAllergies = [];
          } else {
            this.selectedAllergies = this.userSettings.allergyType.split(';');
            this.selectedAllergiesChange.emit(this.selectedAllergies);
          }

          if (this.userSettings.dietType === "") {
            this.selectedDiets = [];
          } else {
            this.selectedDiets = this.userSettings.dietType.split(';');
            this.selectedDietsChange.emit(this.selectedDiets);
          }

          if (this.userSettings.mealType === "") {
            this.selectedMeals = [];
          } else {
            this.selectedMeals = this.userSettings.mealType.split(';');
            this.selectedMealsChange.emit(this.selectedMeals);
          }

          if (this.userSettings.dishType === "") {
            this.selectedDishes = [];
          } else {
            this.selectedDishes = this.userSettings.dishType.split(';');
            this.selectedDishesChange.emit(this.selectedDishes);
          }

          if (this.userSettings.cuisineType === "") {
            this.selectedCuisine = [];
          } else {
            this.selectedCuisine = this.userSettings.cuisineType.split(';');
            this.selectedCuisineChange.emit(this.selectedCuisine);
          }
        }
      }
    })
  }
}

changeAllergies(event: any) {
```

```
    this.selectedAllergiesChange.emit(event);
  }

  changeDiets(event: any) {
    this.selectedDietsChange.emit(event);
  }

  changeDishes(event: any) {
    this.selectedDishesChange.emit(event);
  }

  changeMeals(event: any) {
    this.selectedMealsChange.emit(event);
  }

  changeCuisine(event: any) {
    this.selectedCuisineChange.emit(event);
  }
}
```

Файл profile-page-section.component.ts

```
import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
import {NavigationExtras, Router} from "@angular/router";
import {AuthService} from "../shared-module/auth/auth.service";
import {SettingsService} from "../shared-module/services/settings.service";
import {UserInfo} from "../shared-module/types/user-info";
import {UserSettings} from "../shared-module/types/user-settings";
import {RecipeService} from "../shared-module/services/recipe.service";
import {SavedRecipe} from "../shared-module/types/saved-recipe";
import {MatDialog} from "@angular/material/dialog";
import {DeleteProductPopUpComponent} from "../modals/delete-product-pop-up/delete-product-pop-
up.component";
import {DeleteConfirmPopUpComponent} from "../modals/delete-confirm-pop-up/delete-confirm-pop-
up.component";
import {RecipePopUpComponent} from "../recipes-search-result-page/recipe-pop-up/recipe-pop-up.component";


@Component({
  selector: 'app-profile-page',
  templateUrl: './profile-page.component.html',
  styleUrls: ['./profile-page.component.scss']
```

```
})
export class ProfilePageComponent implements OnInit {

  user: UserInfo = <UserInfo> {};
  settings: UserSettings = <UserSettings> {};

  selectedAllergies: string[] = [];
  selectedDiets: string[] = [];
  selectedMeals: string[] = [];
  selectedDishes: string[] = [];
  selectedCuisine: string[] = [];

  savedRecipes: SavedRecipe[] = [];

  constructor(
    private router: Router,
    private authService: AuthService,
    private settingsService: SettingsService,
    private recipeService: RecipeService,
    private matDialog: MatDialog
  ) {

  }

  ngOnInit(): void {
    this.authService.getUserInfo().subscribe({
      next: (data) => {
        this.user.name = data.name;
        this.user.email = data.email;
      }
    });

    this.recipeService.savedRecipesChanged.subscribe({
      next: (recipes) => {
        console.log("here: ", recipes);
        this.savedRecipes = recipes;
      }, error: (error) => {
        console.log("Error while getting saved recipes from server" + error)
      }
    });

    this.settingsService.userSettingsChanged.subscribe(
```

```
    (settings) => {
      this.settings = settings;

      if (settings.allergyType === undefined) {
        return;
      }

      if (settings.allergyType === "") {
        this.selectedAllergies = [];
      } else {
        this.selectedAllergies = settings.allergyType.split(';');
      }

      if (settings.dietType === "") {
        this.selectedDiets = [];
      } else {
        this.selectedDiets = settings.dietType.split(';');
      }

      if (settings.mealType === "") {
        this.selectedMeals = [];
      } else {
        this.selectedMeals = settings.mealType.split(';');
      }

      if (settings.dishType === "") {
        this.selectedDishes = [];
      } else {
        this.selectedDishes = settings.dishType.split(';');
      }

      if (settings.cuisineType === "") {
        this.selectedCuisine = [];
      } else {
        this.selectedCuisine = settings.cuisineType.split(';');
      }
    }
  );
}


editProfile() {
```

```
    const navigationExtras: NavigationExtras = {
      state: { user: this.user, settings: this.settings},
    };
    this.router.navigate(['/edit-profile'], navigationExtras);
  }

  openFullRecipe(recipe: SavedRecipe) {
    this.recipeService.getRecipeById(recipe.recipeId).subscribe(
      (data) => {
        const info = {
          recipe: data.recipe,
          matchNumber: 1,
          matchRate: 0.5,
        }
        const body = {
          "selectedIngredients": [],
          "recipe": info,
        }
        this.matDialog.open(RecipePopUpComponent, {
          data: body,
          autoFocus: false,
          maxHeight: '90vh'
        });
      }
    );
  }

  openConfirmDelete(recipe: SavedRecipe) {
    this.matDialog.open(DeleteConfirmPopUpComponent, {
      data: recipe
    });
  }
}
```

Файл create-product-pop-up.component.ts

```
import {Component, Inject, OnInit, ViewChild} from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatDatepicker } from '@angular/material/datepicker';
import {MAT_DIALOG_DATA, MatDialog} from '@angular/material/dialog';
import { TaskFrequenciesEnum } from 'src/app/shared-module/enums/task-frequencies.enum';
import { Guid } from 'src/app/shared-module/types/guid';
import {ProductService} from "../../shared-module/services/product.service";
```

```
import {Product} from "../../shared-module/types/product";
import {IngredientFullInfo} from "../../shared-module/types/ingredient-full-info";
import {Category} from "../../shared-module/types/category-info";

@Component({
  selector: 'app-create-product-pop-up',
  templateUrl: './create-product-pop-up.component.html',
  styleUrls: [
    './create-product-pop-up.component.scss',
    '../login-pop-up/login-pop-up.component.scss',
  ],
})
export class CreateProductPopUpComponent implements OnInit {
  @ViewChild('picker') datePicker?: MatDatepicker<Date>;
  createProductForm = <FormGroup>{};
  minDate = new Date();
  createProductError?: string;
  selectedCategory: string = "No selected category";

  allIngredients: IngredientFullInfo[] = [];
  allCategories: Category[] = [];

  constructor(
    private productService: ProductService,
    private fb: FormBuilder,
    private dialogRef: MatDialog,
    @Inject(MAT_DIALOG_DATA) private data: any
  ) {
    this.allIngredients = data.allIngredients;
    this.allCategories = data.allCategories;
  }

  ngOnInit(): void {
    this.createProductForm = this.buildFormGroup();
  }

  buildFormGroup(): FormGroup {
    return this.fb.group({
      ingredient: ['', [Validators.required, Validators.minLength(3)]],
      expirationDate: [new Date(), Validators.required],
      price: [0, [Validators.required, Validators.min(1)]],
      category: ""
```

```
  });
}

findCategoryValueName(categoryId: string | undefined) {
  return this.allCategories.find(c => c.id === categoryId)?.value;
}

groupByFn = (item: IngredientFullInfo) => item.categoryId;
groupValueFn = (_: string, children: any[]) => ({ value: children[0].categoryId });

onSubmit() {
  if (this.createProductForm.valid) {
    const newProduct: Product = this.getProductFromForm();
    this.productService.createProduct(newProduct).subscribe({
      next: () => {
        this.productService.getAvailableProductsByUser().subscribe({
          next: (products: Product[]) => {
            this.productService.userInventory = products;
            this.productService.changeUserInventory();
          },
          error: (error: Error) => {
            console.log('Error occurred while creating product: ' + error.message);
          },
        });
        this.dialogRef.closeAll();
      },
      error: (error) => {
        this.createProductError = error;
      },
    });
  } else {
    this.dialogRef.closeAll();
  }
}

openDatePicker() {
  this.datePicker?.open();
}

onSelect() {
  const result = this.findCategoryValueName(
    this.allIngredients.find(x => x.value === this.createProductForm.value.ingredient)?.categoryId
```

```
    );

    if (result !== undefined) {
      this.selectedCategory = result;
    }

    console.log(this.selectedCategory);
  }

  private getProductFromForm(): Product {
    const ingredient = this.allIngredients.find(x => x.value === this.createProductForm.value.ingredient);
    const date = new Date(this.createProductForm.value.expirationDate);
    date.setHours(date.getHours() + 3);
    return {
      id: Guid.empty,
      ingredientId: ingredient!.id,
      userId: "",
      expirationDate: date.toISOString(),
      price: this.createProductForm.value.price,
      isAvailable: true,
      creationDate: new Date().toISOString(),
    };
  }

}
```

Файл inventory-page.component.ts

```
import {Component, OnInit} from '@angular/core';
import { Product}  from "../shared-module/types/product";
import {EditProductPopUpComponent} from "../modals/edit-product-pop-up/edit-product-pop-up.component";
import {MatDialog} from "@angular/material/dialog";
import {CreateProductPopUpComponent} from "../modals/create-product-pop-up/create-product-pop-
up.component";
import {ProductService} from "../shared-module/services/product.service";
import {IngredientFullInfo} from "../shared-module/types/ingredient-full-info";
import {Category} from "../shared-module/types/category-info";
import {RecipePopUpComponent} from "../recipes-search-result-page/recipe-pop-up/recipe-pop-up.component";
import {NavigationExtras, Router} from "@angular/router";
```

```
@Component({
  selector: 'app-inventory-page',
  templateUrl: './inventory-page.component.html',
  styleUrls: ['./inventory-page.component.scss']
})

export class InventoryPageComponent implements OnInit {
  date = new Date();
  products: Product[] = [];

  allIngredients: IngredientFullInfo[] = [];
  allCategories: Category[] = [];

  chooseForSearchList: string[] = [];

  constructor(
    private dialog: MatDialog,
    private productService: ProductService,
    private router: Router
  ) {
  }

  ngOnInit() {
    this.productService.userInventoryChanged.subscribe(
      (data) => {
        this.products = data;
      }
    );
    this.productService.allCategoriesChanged.subscribe(
      (categories) => (this.allCategories = categories)
    );
    this.productService.allIngredientsChanged.subscribe(
      (ingredients) => (this.allIngredients = ingredients)
    );
    this.productService.changeAllIngredients();
    this.productService.changeAllCategories();
  }

  addProduct() {
    const body = {
      "allIngredients": this.allIngredients,
```

```
      "allCategories": this.allCategories,
    }
    this.dialog.open(CreateProductPopUpComponent, {
      data: body
    });
  }

  chooseForSearch(product: Product) {
    const ingredientName = this.allIngredients.find(i => i.id === product.ingredientId)!.value;
    if (!this.chooseForSearchList.includes(ingredientName)) {
      this.chooseForSearchList.push(ingredientName);
    }
  }

  removeFromList(ingredient: string) {
    const index = this.chooseForSearchList.indexOf(ingredient);
    if (index > -1) {
      this.chooseForSearchList.splice(index, 1);
    }
  }

  searchFromExistingIngredients() {
    const navigationExtras: NavigationExtras = {
      state: { existingIngredients: this.chooseForSearchList },
    };
    this.router.navigate(['/recipe-generator'], navigationExtras);
  }
}
```

Файл nav-menu-component.html

```
<nav class="navbar navbar-expand-lg">
  <a class="navbar-brand btn btn-outline-danger" [routerLink]="['/']"
    >FridgeHelper
    <img class="google" src="./assets/img/fridge.png" style="width: 50px; height: 50px" alt="fridgeIcon"/>
  </a>
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="fridgeNavbar"
    aria-controls="fridgeNavbar"
    [attr.aria-expanded]="!isCollapsed"
```

```
  aria-label="Toogle navigation"
  (click)="isCollapsed = !isCollapsed"
>
  <span class="navbar-toggler-icon"></span>
</button>
<div
  class="collapse navbar-collapse"
  id="fridgeNavbar"
  [collapse]="isCollapsed"
>
  <div class="navbar-nav mr-auto mt-2 mt-lg-0">
    <li class="nav-item" *ngIf="isLoggedIn && connected">
      <a
        class="nav-link btn btn-outline-danger"
        [routerLink]="['/statistics']"
        routerLinkActive="active-route"
        >Statistics</a>
    </li>
    <li class="nav-item" *ngIf="isLoggedIn && connected">
      <a
        class="nav-link btn btn-outline-danger"
        [routerLink]="['/inventory']"
        routerLinkActive="active-route"
      >Inventory</a>
    </li>
    <li class="nav-item" *ngIf="connected">
      <a
        class="nav-link btn btn-outline-danger"
        [routerLink]="['/recipe-generator']"
        routerLinkActive="active-route"
      >Generate recipe</a>
    </li>
    <li class="nav-item" *ngIf="!isLoggedIn && connected">
      <a class="nav-link btn btn-outline-danger" (click)="openLogin()"
        >Login</a
      >
    </li>
    <li class="nav-item nav-item-profile dropdown" *ngIf="isLoggedIn && connected">
      <button id="dropdownMenuButton" mat-icon-button [matMenuTriggerFor]="menu" aria-label="Example
icon-button with a menu">
        <img
          src="assets/img/user.png"
```

```
          alt="user" style="
         height: 30px;
         width: 30px;
         filter: brightness(0) invert(1);"
         ></button>
        <mat-menu #menu="matMenu" class="dropdown-menu">
         <a
          mat-menu-item
          [routerLink]="['/profile']"
          routerLinkActive="active-route"
         >Profile</a>
         <a
          mat-menu-item
          (click)="onLogout()"
         >Logout</a>
        </mat-menu>
       </li>
     </div>
   </div>
</nav>
```

Файл recipe-service.ts

```
import { Inject, Injectable } from '@angular/core';
import {HttpClient, HttpHeaders, HttpParams} from '@angular/common/http';
import {BehaviorSubject, catchError, concatAll, map, Observable, Subject, takeUntil, tap, throwError} from 'rxjs';
import jwt_decode from 'jwt-decode';
import { DOCUMENT } from '@angular/common';
import {RecipeRequest} from "../types/recipe-request";
import {environment} from "../../../environments/environment";
import {RecipeInfo} from "../types/recipe-info";
import {SavedRecipe} from "../types/saved-recipe";
import {Product} from "../types/product";
import {AuthService} from "../auth/auth.service";
import * as http from "http";


@Injectable({
  providedIn: 'root',
})
export class RecipeService {
```

```
  private url: string = "https://api.edamam.com/api/recipes/v2";

  public savedRecipes: SavedRecipe[] = [];
  private savedRecipesSource = new BehaviorSubject<SavedRecipe[]>(this.savedRecipes);
  savedRecipesChanged = this.savedRecipesSource.asObservable();

  constructor(
    private http: HttpClient,
    private AuthService: AuthService,
    @Inject(DOCUMENT) private document: Document
  ) {
    console.log("initial of recipe service");
    console.log("Authenticated: " + AuthService.isAuthenticated());
    if (AuthService.isAuthenticated()) {
      this.getRecipesByUser().subscribe({
        next: (recipes: SavedRecipe[]) => {
          console.log(recipes)
          this.savedRecipes = recipes;
          this.changeSavedRecipes();
        },
        error: (error: Error) => {
          console.log('Error occurred while getting recipes: ' + error.message);
        },
      });
    }
  }

  getAllRecipes(body: RecipeRequest): Observable<any> {

    let dietString = "";
    let healthString = "";
    let cuisineString = "";
    let mealString = "";
    let dishString = "";

    let requestUrl = this.url + "?type=" + body.type + "&q=" + body.q + "&app_id=" + body.app_id + "&app_key="
+ body.app_key;

    if (body.diet.length > 0) {
      const loweredDiet = body.diet.map(x => x.toLocaleLowerCase());
      dietString = "&diet=" + loweredDiet.join("&diet=");
    }
```

```
    if (body.health.length > 0) {
      const loweredHealth = body.health.map(x => x.toLocaleLowerCase());
      healthString = "&health=" + loweredHealth.join("&health=");
    }

    if (body.cuisineType.length > 0) {
      const loweredCuisine = body.cuisineType.map(x => x.toLocaleLowerCase());
      cuisineString = "&cuisineType=" + loweredCuisine.join("&cuisineType=");
    }

    if (body.mealType.length > 0) {
      const loweredMealType = body.mealType.map(x => x.toLocaleLowerCase());
      mealString = "&mealType=" + loweredMealType.join("&mealType=");
    }

    if (body.dishType.length > 0) {
      const loweredDishes = body.dishType.map(x => x.toLocaleLowerCase());
      dishString = "&dishType=" + loweredDishes.join("&dishType=");
    }

    requestUrl = requestUrl + dietString + healthString + cuisineString + mealString + dishString;

    return this.http.get<any>(requestUrl);
  }


  getRecipeById(id: string) {
    let requestUrl = this.url + "/recipe_" + id + "?type=public" + "&app_id=" + environment.appId + "&app_key=" +
environment.appKey;
    return this.http.get<any>(requestUrl);
  }

  addRecipeToFavourites(body: RecipeInfo) {
    const url = environment.baseUrl + 'recipe';
    return this.http.post<RecipeInfo>(url, body);
  }

  getRecipesByUser() {
    const url = environment.baseUrl + 'Recipe/getSavedRecipes';
    return this.http.get<SavedRecipe[]>(url);
  }
```

```
  removeRecipe(recipeId: string) {
    const url = environment.baseUrl + `recipe/${recipeId}`;
    return this.http.delete<RecipeInfo>(url);
  }

  changeSavedRecipes() {
    this.savedRecipesSource.next(this.savedRecipes);
  }
}
```

Файл product-service.ts

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpParams} from '@angular/common/http';
import db from "../../../db.json";
import {environment} from "../../../environments/environment";

import { AuthService } from "../auth/auth.service";
import { BehaviorSubject } from "rxjs";
import { Ingredient } from "../types/ingredient-info";
import { Category } from "../types/category-info";
import { Product } from "../types/product";
import { IngredientFullInfo } from "../types/ingredient-full-info";
import {addUnitAlias} from "ngx-bootstrap/chronos/units/aliases";


@Injectable({ providedIn: 'root' })
export class ProductService {
  private url: string = environment.baseUrl;
  private allergies: string[] = [];
  private diets: string[] = [];
  private cuisineTypes: string[] = [];
  private mealTypes: string[] = [];
  private dishTypes: string[] = [];

  public userInventory: Product[] = [];
  private userInventorySource = new BehaviorSubject<Product[]>(this.userInventory);
  userInventoryChanged = this.userInventorySource.asObservable();

  public allIngredients: IngredientFullInfo[] = [];
  private allIngredientsSource = new BehaviorSubject<IngredientFullInfo[]>(this.allIngredients);
  allIngredientsChanged = this.allIngredientsSource.asObservable();
```

```
public allCategories: Category[] = [];
private allCategoriesSource = new BehaviorSubject<Category[]>(this.allCategories);
allCategoriesChanged = this.allCategoriesSource.asObservable();

constructor(
  private http: HttpClient,
  private AuthService: AuthService
) {
  db.Allergies.forEach(x => this.allergies.push(x));
  db.Diets.forEach(x => this.diets.push(x));
  db.CuisineType.forEach(x => this.cuisineTypes.push(x));
  db.MealType.forEach(x => this.mealTypes.push(x));
  db.DishType.forEach(x => this.dishTypes.push(x));

  if (AuthService.isAuthenticated()) {
    this.getAvailableProductsByUser().subscribe({
      next: (products: Product[]) => {
        console.log("id: " + this.AuthService.getUserId() + "products" + products);
        this.userInventory = products;
        this.changeUserInventory();
      },
      error: (error: Error) => {
        console.log('Error occurred while getting products: ' + error.message);
      },
    });
  }

  this.getCategories().subscribe({
    next: (data) => {
      this.allCategories = data;
      this.changeAllCategories();
    },
    error: (err) => {
      console.log(err);
    }
  });

  this.getIngredients().subscribe({
    next: (data) => {
      const tempArray = [];
      for (let item of data) {
```

```
        tempArray.push({
          categoryId: item.categoryId,
          id: item.id,
          selected: false,
          value: item.value
        });
      }
      this.allIngredients = tempArray;
      this.changeAllIngredients();
    },
    error: (err) => {
      console.log(err);
    }
  })
}


changeUserInventory() {
  this.userInventorySource.next(this.userInventory);
}

changeAllIngredients() {
  this.allIngredientsSource.next(this.allIngredients);
}

changeAllCategories() {
  this.allCategoriesSource.next(this.allCategories);
}

getIngredients() {
  return this.http.get<Ingredient[]>(`${this.url}ProductBase/ingredients`);
}

getCategories() {
  return this.http.get<Category[]>(`${this.url}ProductBase/categories`);
}

getCategoryById(categoryId: string) {
  return this.http.get<Category>(`${this.url}/${categoryId}`)
}

getIngredientsByCategories() {
```

```
  return this.http.get<Product[]>(this.url);
}

getIngredientByName(name: string) {
  return this.http.get<Ingredient>(`${this.url}ProductBase/ingredients/name/${name}`);
}

getAvailableProductsByUser() {
  return this.http.get<Product[]>(`${this.url}Products/getByUser`);
}

createProduct(product: Product) {
  product.userId = this.AuthService.getUserId();
  return this.http.post<Product>(`${this.url}Products`, product);
}

updateProduct(product: Product) {
  return this.http.put<Product>(`${this.url}Products/${product.id}`, product);
}

deleteProduct(id: string) {
  return this.http.delete(`${this.url}Products/${id}`);
}

getAllergies() {
  return this.allergies;
}

getDiets() {
  return this.diets;
}

getMeals() {
  return this.mealTypes;
}

getDishes() {
  return this.dishTypes;
}

getCuisineTypes() {
  return this.cuisineTypes;
```

```
  }

  getBoughtProductsByDate(date: Date) {
    return this.http.get<Product[]>(`${this.url}Products/getByUser`);
  }
}
```

Файл profile-page.component.html

```
<div class="profile-container">
  <div class="profile-info-wrapper">
    <h1>Profile</h1>
    <div class="personal-info">
      <span class="personal-info-title">Name:</span>
      <span>{{ user.name }}</span>
    </div>
    <div class="personal-info">
      <span class="personal-info-title">Email:</span>
      <span>{{ user.email }}</span>
    </div>
    <div class="user-preferences">
      <h2>Preferences</h2>
      <div>
        <div class="filter-row">
          <div class="allergies">
            <h6>Health filter</h6>
            <ng-select
              [items]="selectedAllergies"
              bindLabel="name"
              [disabled]="true"
              [multiple]="true"
              [(ngModel)]="selectedAllergies"
            >
            </ng-select>
          </div>
          <div class="diets">
            <h6>Diet filter</h6>
            <ng-select
              [items]="selectedDiets"
              bindLabel="name"
              [disabled]="true"
              [multiple]="true"
              [(ngModel)]="selectedDiets"
```

```
      >
      </ng-select>
    </div>
  </div>
  <div class="filter-row">
    <div class="cuisine">
      <h6>Cuisine type filter</h6>
      <ng-select
        [items]="selectedCuisine"
        bindLabel="name"
        [disabled]="true"
        [multiple]="true"
        [(ngModel)]="selectedCuisine"
      >
      </ng-select>
    </div>
    <div class="dish">
      <h6>Dish type filter</h6>
      <ng-select
        [items]="selectedDishes"
        bindLabel="name"
        [disabled]="true"
        [multiple]="true"
        [(ngModel)]="selectedDishes"
      >
      </ng-select>
    </div>
  </div>
  <div class="filter-row">
    <div class="meals">
      <h6>Meal type filter</h6>
      <ng-select
        [items]="selectedMeals"
        bindLabel="name"
        [disabled]="true"
        [multiple]="true"
        [(ngModel)]="selectedMeals"
      >
      </ng-select>
    </div>
  </div>
</div>
```

```
      <mat-checkbox [disabled]="true" [(ngModel)]="settings.useDefault"
        >Set default preferences</mat-checkbox
      >
    </div>
    <div class="button-wrapper">
      <button mat-raised-button color="primary" (click)="editProfile()">
        Edit Profile
      </button>
    </div>
  </div>
  <div class="saved-recipes">
    <h2>Saved recipes</h2>
    <div
      *ngFor="let recipe of savedRecipes"
      class="saved-recipe-item"
    >
      <span (click)="openFullRecipe(recipe)">{{ recipe.recipeTitle }}</span>
      <mat-icon (click)="openConfirmDelete(recipe)">delete</mat-icon>
    </div>
  </div>
</div>
```

Файл edit-product-pop-up.component.ts

```
import {Component, Inject, Input, OnInit, ViewChild} from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatDatepicker } from '@angular/material/datepicker';
import {MAT_DIALOG_DATA, MatDialog} from '@angular/material/dialog';
import {IngredientFullInfo} from "../../shared-module/types/ingredient-full-info";
import {Category} from "../../shared-module/types/category-info";
import {ProductService} from "../../shared-module/services/product.service";
import {Product} from "../../shared-module/types/product";
import {Guid} from "../../shared-module/types/guid";

@Component({
  selector: 'app-edit-product-pop-up',
  templateUrl: './edit-product-pop-up.component.html',
  styleUrls: [
    './edit-product-pop-up.component.scss',
    '../login-pop-up/login-pop-up.component.scss',
  ],
})
export class EditProductPopUpComponent implements OnInit {
```

```
@ViewChild('picker') datePicker?: MatDatepicker<Date>;
editProductForm = <FormGroup>{};
currentProduct: Product = <Product>{};
minDate = new Date();
editProductError?: string;
selectedCategory: string = "No selected category";

allIngredients: IngredientFullInfo[] = [];
allCategories: Category[] = [];

constructor(
  private productService: ProductService,
  private fb: FormBuilder,
  private dialogRef: MatDialog,
  @Inject(MAT_DIALOG_DATA) private data: any
) {
  this.allIngredients = data.allIngredients;
  this.allCategories = data.allCategories;
  this.currentProduct = data.product;

  console.log(this.currentProduct);
}

ngOnInit(): void {
  this.editProductForm = this.buildFormGroup();
  this.onSelect();
  this.minDate = new Date(this.currentProduct.expirationDate);
}

private productIsChanged(): boolean {
  const ingredient = this.findIngredientNameById(this.currentProduct.ingredientId);
  return (
    ingredient!.value !== this.editProductForm.value.ingredient ||
    this.currentProduct.expirationDate !== this.editProductForm.value.expirationDate||
    this.currentProduct.price !== this.editProductForm.value.price
  );
}

findIngredientNameById(id: string) {
  return this.allIngredients.find(x => x.id === id);
}
```

```
  buildFormGroup(): FormGroup {
    return this.fb.group({
      ingredient: [this.findIngredientNameById(this.currentProduct.ingredientId)!.value, [Validators.required,
Validators.minLength(3)]],
      expirationDate: [this.currentProduct.expirationDate, Validators.required],
      price: [this.currentProduct.price, [Validators.required, Validators.min(1)]],
      category:
this.findCategoryValueName(this.findIngredientNameById(this.currentProduct.ingredientId)!.categoryId)
    });
  }

  findCategoryValueName(categoryId: string | undefined) {
    return this.allCategories.find(c => c.id === categoryId)?.value;
  }

  groupByFn = (item: IngredientFullInfo) => item.categoryId;
  groupValueFn = (_: string, children: any[]) => ({ value: children[0].categoryId });

  onSubmit() {
    if (this.editProductForm && this.productIsChanged()) {
      const updatedProduct: Product = this.getUpdatedProductFromForm();
      this.productService.updateProduct(updatedProduct).subscribe({
        next: () => {
          this.productService.getAvailableProductsByUser().subscribe({
            next: (products: Product[]) => {
              this.productService.userInventory = products;
              this.productService.changeUserInventory();
            },
            error: (error: Error) => {
              console.log('Error occurred while updating product: ' + error.message);
            },
          });
          this.dialogRef.closeAll();
        },
        error: (error) => {
          this.editProductError = error;
        },
      });
    } else {
      this.dialogRef.closeAll();
```

```
    }
  }

  openDatePicker() {
    this.datePicker?.open();
  }

  onSelect() {
    const result = this.findCategoryValueName(
      this.allIngredients.find(x => x.value === this.editProductForm.value.ingredient)?.categoryId
    );

    if (result !== undefined) {
      this.selectedCategory = result;
    }

    console.log(this.selectedCategory);
  }

  private getUpdatedProductFromForm(): Product {
    const ingredient = this.allIngredients.find(x => x.value === this.editProductForm.value.ingredient);
    const date = new Date(this.editProductForm.value.expirationDate);
    date.setHours(date.getHours() + 3);
    return {
      id: this.currentProduct.id,
      ingredientId: ingredient!.id,
      userId: this.currentProduct.userId,
      expirationDate: date.toISOString(),
      price: this.editProductForm.value.price,
      isAvailable: this.currentProduct.isAvailable,
      creationDate: this.currentProduct.creationDate
    };
  }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Програма та методика тестування

КПІ.ІП-9128.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Максим ШЕЙКО

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об'єктом випробування є розроблений протягом дипломного проєктування веб-застосунок призначений для відстеження запасів продуктів, рецептів з наявних та випадкових інгредієнтів з урахуванням кулінарних вподобань користувача та дієтичних обмежень, аналізу статистики грошових витрат на придбавання харчових продуктів.

Розроблене програмне забезпечення з використанням фреймворку Angular для клієнтської частини та .NET для серверної частини тестувалося за допомогою використання різних веб-браузерів, включаючи Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Також була проведена перевірка коректності роботи застосунку на різних операційних системах, таких як Windows, macOS і Linux.

Під час випробування програмного забезпечення, відбувалася перевірка його функціональності, коректності взаємодії з користувачем при операціях авторизації та реєстрації, правильності пошуку рецептів на основі різних параметрів та аналізу статистики витрат користувача. Тестування було проведено на різних пристроях зі своїми роздільними здатностями екранів для того, щоб переконатися, що користувацький інтерфейс коректно відображається на різноманітних пристроях.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– забезпечення високої якості та надійності розробленого програмного забезпечення перед його використанням користувачами в продуктивному середовищі.

– перевірка сумісності веб-застосунку з сучасними браузерами: веб-додаток має перевірятися на сумісність з використанням різних актуальних веб-браузерів, такими як Chrome, Opera, Firefox, Edge, Safari. Це виконується шляхом перевірки функціональності додатку в кожному з перерахованих браузерів.

– перевірка правильності роботи додатку: проведення тестів забезпечує перевірку того, чи відповідає розроблене програмне забезпечення поставленим функціональним вимогам.

– перевірка збереження даних: проведення тестування передбачає перевірку збереження даних. Дані користувача, які вводяться у програму, мають бути коректно збережені та доступні для подальшого використання.

– перевірка сумісності з операційними системами: тестування передбачаэ перевірку сумісності додатку з різними операційними системами, (Windows, Linux, MacOS). Це потрібно для того, щоб переконатися, що додаток коректно працює на різних платформах і операційних системах.

– виявлення помилок і недоліків: тестування допомагає виявити проблеми та недоліки в роботі програмного забезпечення. Тестування дозволяє вчасно виправити знайдені помилки перед розгортанням продукту, забезпечуючи надійність роботи програми.

– перевірка зручності графічного інтерфейсу: розроблений додаток має виконувати поставлені нефункціональні вимоги, бути зручним у використанні та графічний інтерфейс повинен мати естетичний вигляд. Тестування передбачає перевірку розміщення компонентів, навігації, додаткового візуального оформлення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – включає перевірку програми та супровідної документації для перевірки дотримання стандартів програмування та виявлення потенційних помилок;

– динамічне тестування використовується під час розробки програми. Виконується певна кількість тестів для перевірки правильності роботи програмного забезпечення. Під час кожного тесту збираються та аналізуються дані про помилки та недоліки в роботі додатку;

– функціональне тестування передбачає перевірку наскільки реальна поведінка програмного забезпечення відповідає описаним функціональним вимогам;

– системне тестування перевіряє все реалізоване програмне забезпечення як цілий продукт. Даний тип тестування орієнтований на перевірку взаємодії між різними компонентами та модулями;

– мануальне тестування – передбачає тестування без використання автоматизації, де тести виконуються вручну відповідальною особою;

– тестування "чорної скриньки" - об'єктом тестування є функціональність програми. Перевіряється правильність вихідних даних при заданих вхідних параметрах;

– тестування "білої скриньки" – об’єктом тестування є внутрішня поведінка програми. Перевіряється архітектура програмного забезпечення та взаємодія компонентів між собою;

– тестування безпеки – даний вид тестування зосереджений на перевірці безпеки реалізованого програмного забезпечення. Передбачається виконання тестування з урахуванням потенційних загроз та вразливості веб-застосунку. Метою цього тестування є виявлення можливих вразливостей та розробка відповідних заходів безпеки.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування розробленого програмного забезпечення виконується мануально, за допомогою написання модульних тестів та завдяки проведенню наскрізного тестування. Наскрізне тестування (E2E) є важливою складовою тестування додатку. Воно спрямоване на перевірку поведінки системи в цілому, симулюючи реальні сценарії взаємодії з користувачем Мета тестування – знайти можливі недоліки або помилки під час роботи програми та їх вчасно виправити для того, щоб зробити користувацький досвід використання більш приємним. Для написання модульних тестів використовувалися xUnit та Jasmine. Тестування одиниць сприяє перевірці правильності роботи окремих функцій, класів або методів та забезпечує перевірку того, що вони поводяться відповідно до очікувань. Jasmine, що є фреймворком для тестування TypeScript коду допоміг у написанні тестів одиниць для перевірки правильності обробки подій та компонентів Angular. Для того, щоб перевірити працездатність та відмовостійкість розробленого застосунку, рекомендується виконати такі тести:

– динамічне тестування на відповідність функціональним вимогам розробленого програмного забезпечення;

– тестування на виведення повідомлень про помилку при реєстрації та авторизації, коли це необхідно - застосунок має відображати повідомлення про помилки та попередження користувачеві він намагається вводити неправильні дані під час реєстрації або авторизації в застосунку;

– тестування зміни орієнтації екрану для того, щоб переконатися, що застосунок та його компоненти правильно адаптуються до зміни орієнтації екрану;

– тестування інтерфейсу користувача – перевірка того, що інтерфейс додатку є зрозумілим для користувача;

– тестування зручності використання;

– тестування безпеки збереження персональних даних користувача.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Керівництво користувача

КПІ.ІП-9128.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Максим ШЕЙКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Програмне забезпечення, що було розроблене в рамках дипломного проєктування призначене для сприяння покращенню культури споживання середньостатистичних користувачів. Завдяки таким функціям, як відстеження запасів, відображення повідомлення про закінчення терміну придатності доданих продуктів до інвентаря, пошук рецептів на різній основі та збір і відображення статистики витрат, додаток має на меті надання користувачам можливість приймати більш обґрунтовані рішення щодо придбавання продуктів харчування та в результаті мінімізувати кількість харчових відходів.

За допомогою розробленому функціоналу додатку користувачі можуть краще розуміти стан своїх запасів продуктів, використовувати наявні в них інгредієнти та споживати їх до закінчення терміну придатності. Це заохочує користувачів оптимізувати свої покупки, планувати приготування страв на основі наявних у них продуктів харчування. Такий підхід до відстеження та використання продуктів харчування сприяє розвитку культури усвідомленого споживання та має значний вплив на зменшення кількості утворюваних харчових відходів окремими особами.

Функція збору статистики витрат на харчові продукти ставить на меті допомогти користувачу більш усвідомлено та відповідально підходити до придбавання товарів або перегляду власної економіки. Це дозволяє користувачам мати контроль над своїми витратами на продукти та заохочує їх проведення свідомих покупок, відмовляючись від надмірних, які можуть в результаті призвести до збільшення відходів.

Персоналізований пошук рецептів надає користувачам креативні ідеї рецептів на основі інгредієнтів, які вони вже мають дома, додаток сприяє використанню наявних продуктів замість того, щоб купувати нові без потреби.

Таким чином, створений додаток покликаний сприяти підвищенню культури споживання через надання користувачам можливості відстежувати свої запаси, ефективно планувати покупки продуктів й приймати обґрунтовані рішення щодо їх вибору. Сприяючи відповідальному та усвідомленому

споживанню, додаток має потенціал зробити значний внесок у зменшення харчових відходів на індивідуальному рівні.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи веб-застосунку, необхідно виконання наступних мінімальних вимог до локальної машини користувача:

– обладнання повинно мати двоядерний процесор (Intec Core i5 або аналог від компанії AMD чи Apple);

– 4 гб оперативної пам’яті;

– наявність доступу до Інтернету.

Рекомендовані вимоги до апаратного забезпечення для використання веб-додатку наступні:

– обладнання повинно мати чотириядерний процесор (Intec Core i5 або аналог від компанії AMD чи Apple);

– 8 гб оперативної пам’яті;

– наявність доступу до Інтернету.

Додаток працює на всіх сучасних версіях браузерів Google Chrome, Mozilla Firefox, Microsoft Edge і Safari.

2.2 Завантаження застосунку

В результаті розробки програмного забезпечення, воно було розгорнуте на хмарному середовищі Azure. Користувач для доступу до створеного додатку вводить в адресну строку свого браузеру адресу додатку та переходить на сторінку. Веб-застосунок є готовим для користування. Клієнт може зайти на сторінку пошуку рецептів за різними параметрами або створити обліковий запис в системі й авторизуватися.

Для отримання актуальної версії веб-застосунку, користувачі можуть періодично оновлювати веб-сторінку додатку. Це можна зробити, натиснувши кнопку "Оновити/перезавантажити" на панелі інструментів браузера або за допомогою комбінації клавіш (наприклад, Ctrl + R або F5).

Слідкувати за наявністю сповіщень та оголошень про оновлення застосунку. Провести перевірку, чи відображається нова версія додатку в інтерфейсі користувача за допомогою певних візуальних індикаторів,

2.3 Перевірка коректної роботи

Для перевірки коректної роботи застосунку, користувачу необхідно відкрити браузер, вставити URL-адресу у рядок пошуку на натиснути ENTER. Відкривається головна сторінка застосунку. Користувач може перейти до сторінки пошуку рецептів за наявними або випадковими інгредієнтами та персональними кулінарними вподобаннями, або авторизуватися в системі, якщо він має обліковий запис чи зареєструватися.

Користувач авторизується в системі, проводить пошук рецептів а отримує результат. Також він перевіряє можливість додавання, видалення продуктів в інвентар. Застосунок працює належним чином.

3 ВИКОНАННЯ ПРОГРАМИ

Користувач переходить за посиланням у браузері, відкривається головна сторінка розробленого веб-застосунку. Користувач знаходиться на головній сторінці, вона складається з навігаційної панелі, короткого опису та кнопки «Почати». Вид головної сторінки застосунку представлений на рисунку 3.1

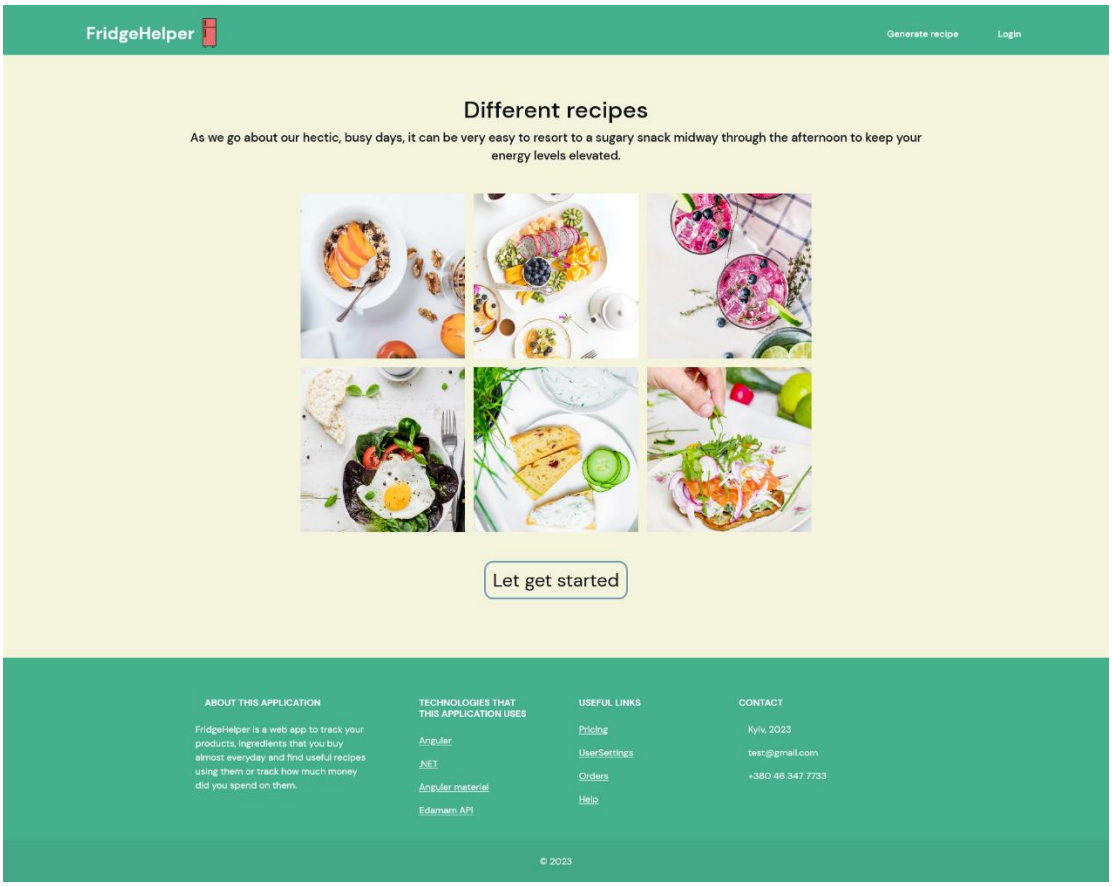


Рисунок 3.1 – Головна сторінка застосунку

Користувач в неавторизованому стані переходить на сторінку пошуку рецептів за випадковими або наявними інгредієнтами з урахуванням персональних вподобань в кулінарії або можливих дієтичних обмежень. Сторінка складається з двох основних частин. В першій користувач може обрати з переліку всіх інгредієнтів ті, за якими необхідно здійснити пошук, вони розбиті за категоріями товарів. З правої сторони екрану – меню вибору параметрів для пошуку. Зображення графічного інтерфейсу сторінки представлене на рисунку 3.2.

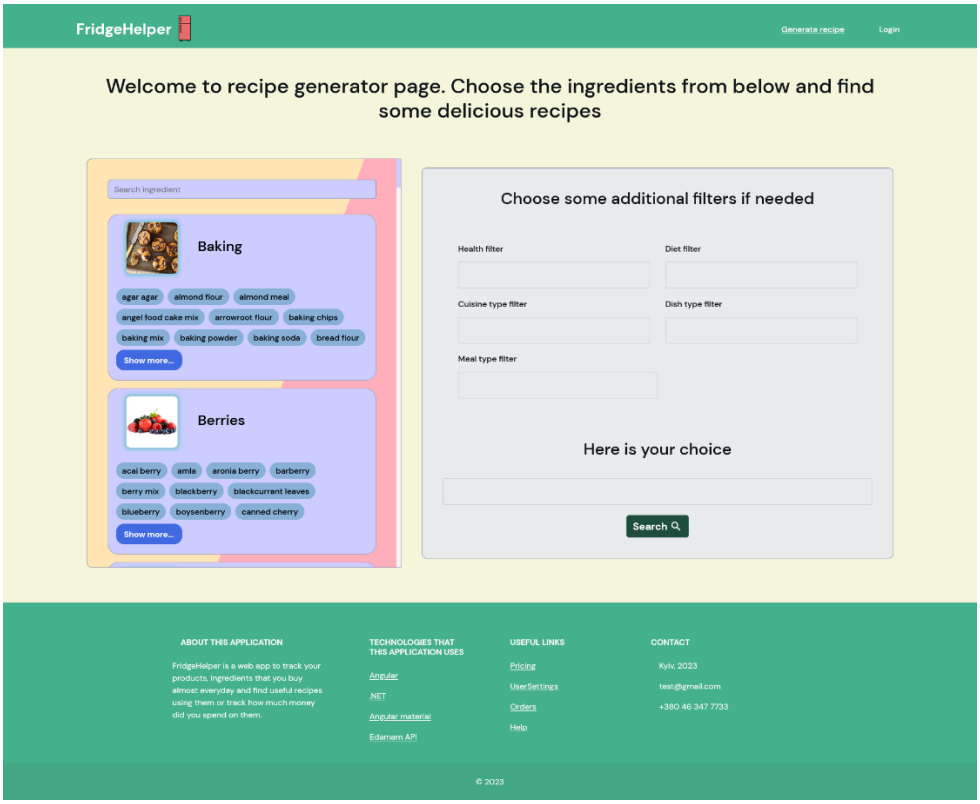


Рисунок 3.2 – Сторінка пошуку рецептів

Користувач обирає інгредієнти та параметри для пошуку, вони з’являються в правому меню. (Рисунок 3.3).

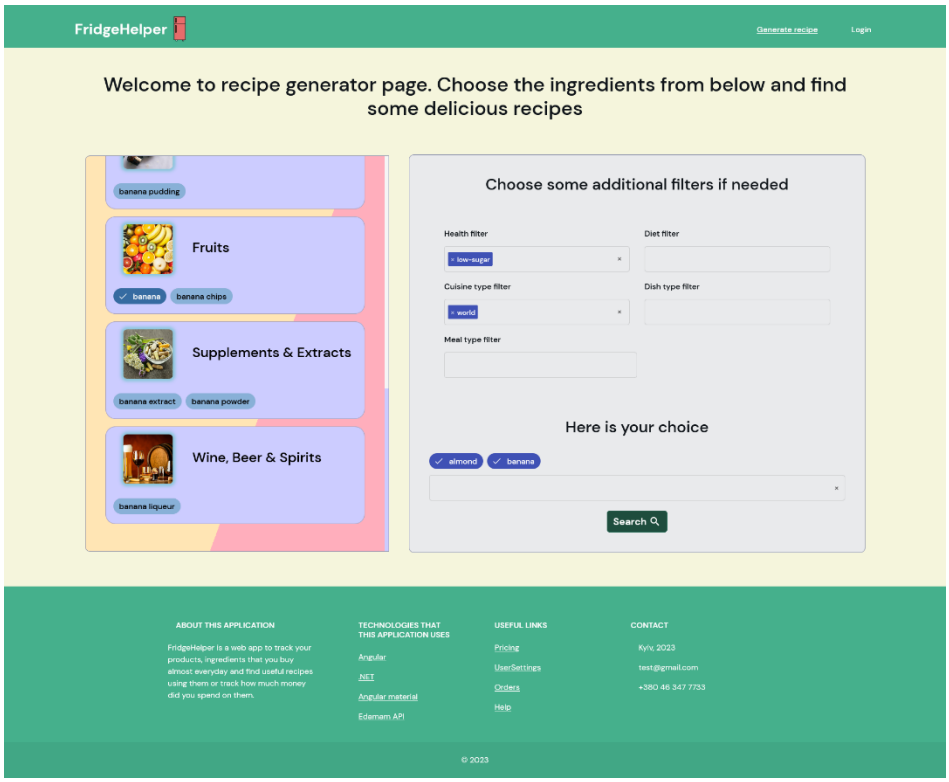


Рисунок 3.3 – Сторінка пошуку рецептів після обирання параметрів

Користувач натискає на кнопку пошуку. Вона не є активною, якщо не обраний хоча б один інгредієнт для пошуку. Відбувається пошук рецептів за заданими параметрами. Результат пошуку зображений на рисунку 3.4.

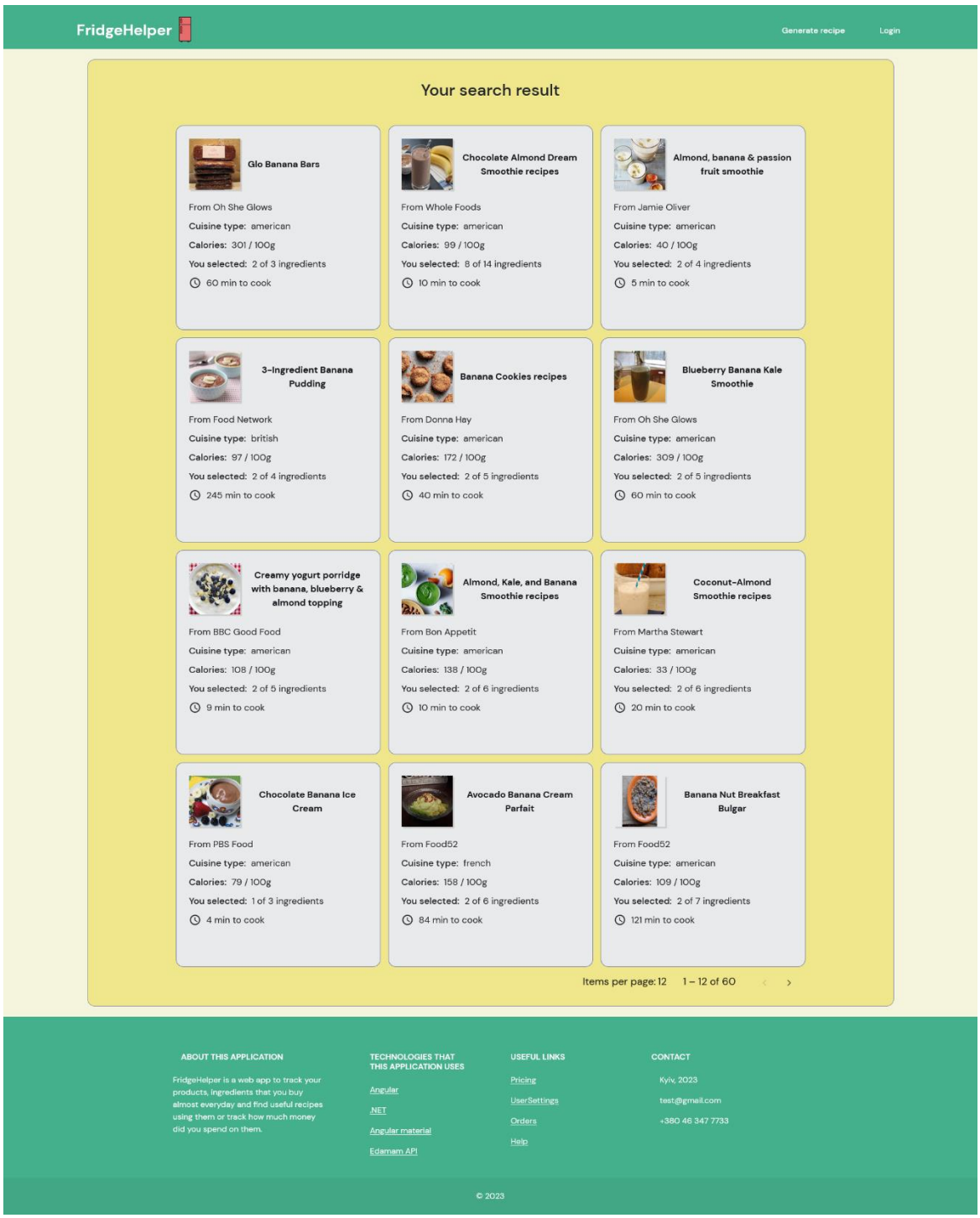


Рисунок 3.5 – Результат пошуку рецептів за обраними параметрами

Користувач натискає на картку рецепту для того, щоб отримати більш детальну інформацію про нього. З’являється діалогове вікно з повною інформацією. (Рисунок 3.6).

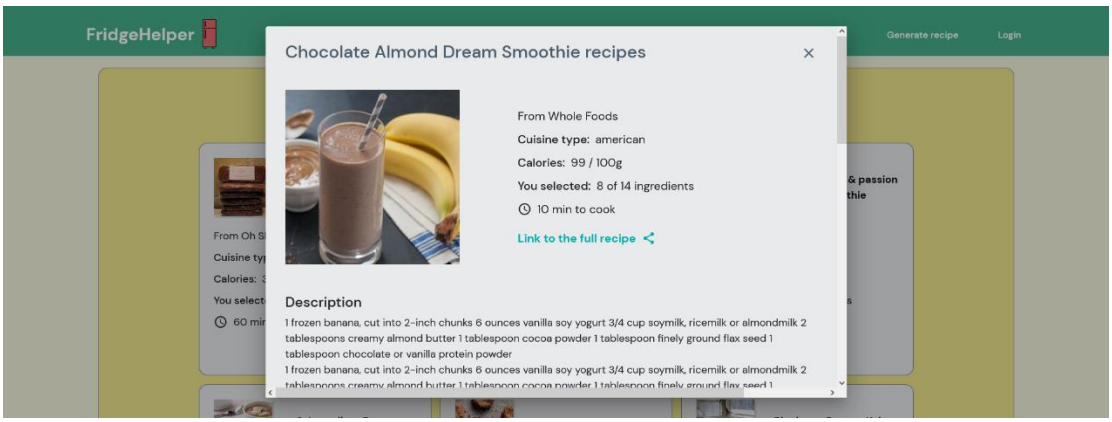


Рисунок 3.6 – Повна інформація про рецепт

Користувач бажає зареєструватися в застосунку для отримання доступу до повного функціоналу веб-застосунку. Для цього він натискає на кнопку “Login”, що знаходиться на навігаційній панелі в правій частині екрану. Відкривається діалогове вікно авторизації, користувач натискає на посилання «зареєструватися», відкривається вікно реєстрації. (Рисунки 3.7 – 3.8).

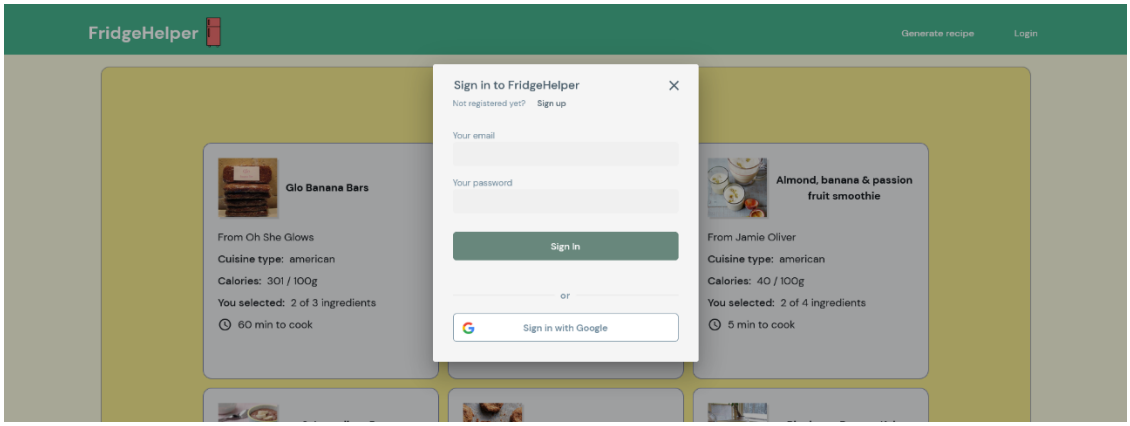


Рисунок 3.7 – Діалогове вікно авторизації в застосунку

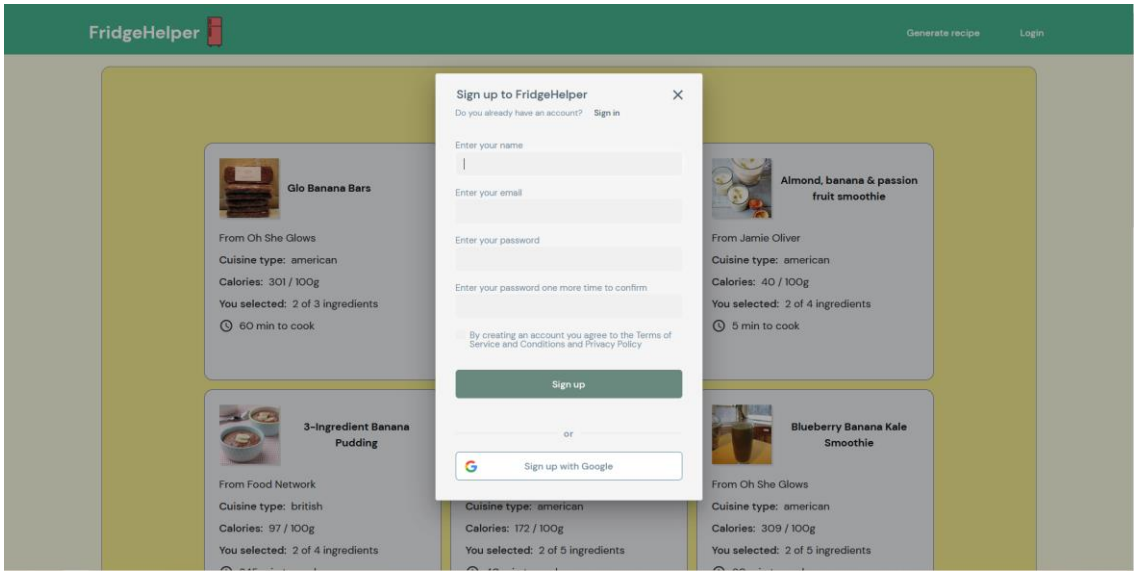


Рисунок 3.8 – Діалогове вікно реєстрації в системі

Користувач вводить у форму, що з’явилася, необхідні дані у коректному форматі, стає активною кнопка «Зареєструватися». Після натискання на неї обліковий запис створюється та користувач перенаправляється на активне вікно авторизації з формою для входу в систему. Користувач вводить адресу електронної пошти та пароль від облікового запису, що був створений та натискає на кнопку «Авторизуватися». Відбувається авторизація користувача в системі, він перенаправляється на голову сторінку. На навігаційній панелі з’являються нові вкладки. (Рисунки 3.9 – 3.11).

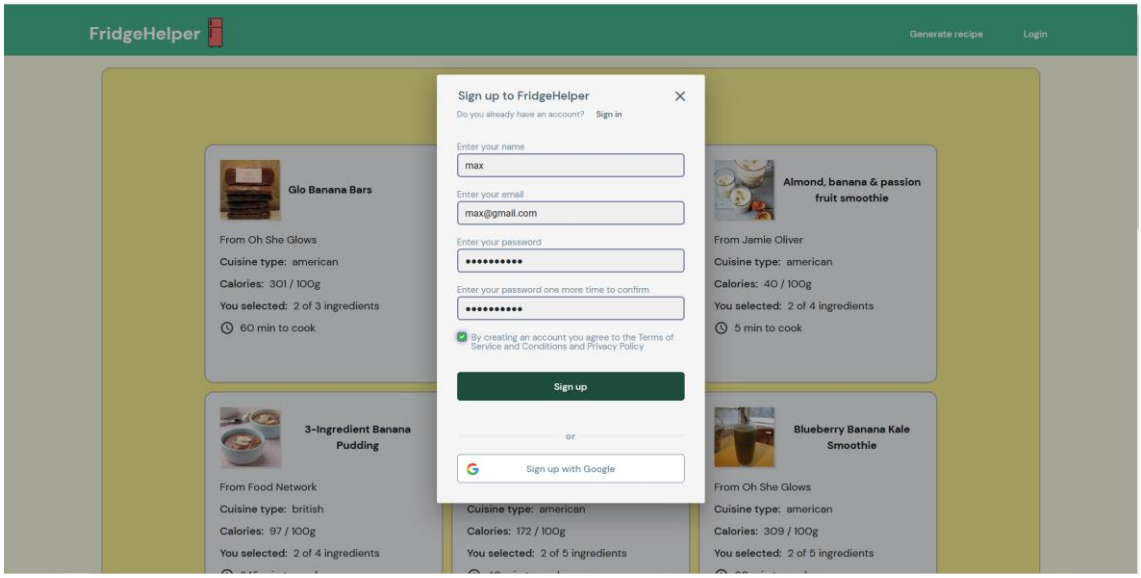


Рисунок 3.9 – Кнопка реєстрації стає активною після введення даних

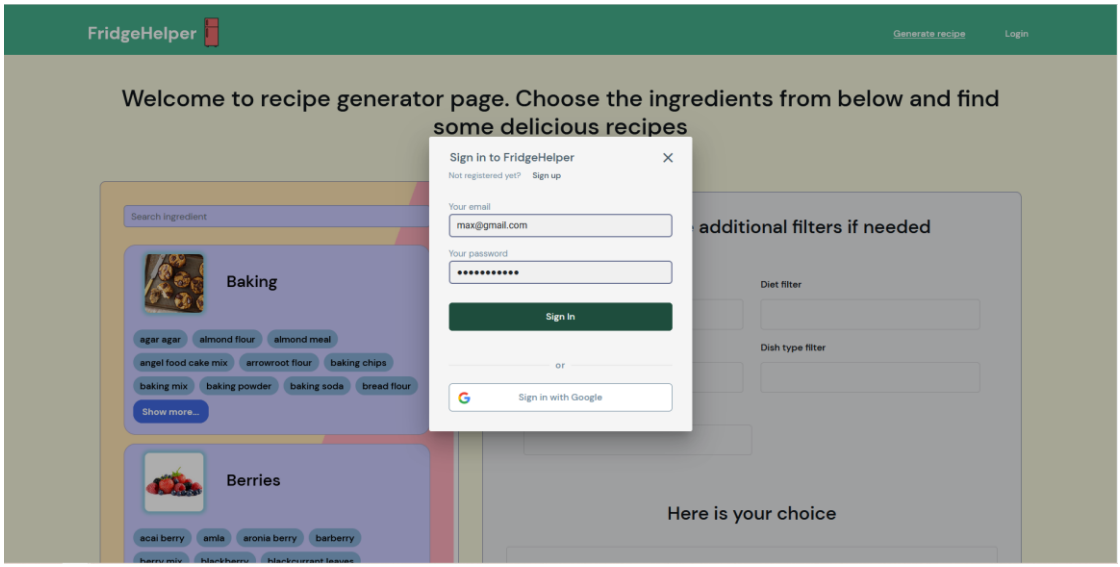


Рисунок 3.10 – Введення даних для авторизації

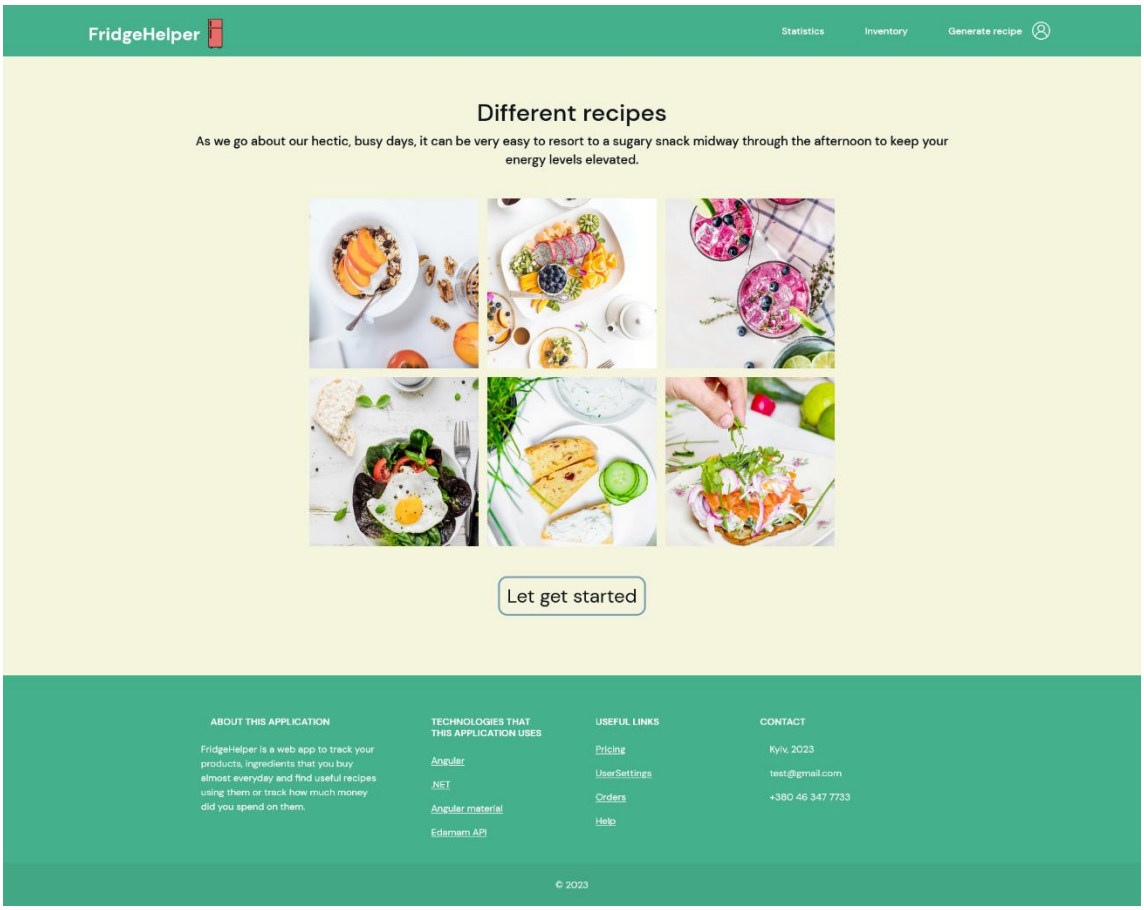


Рисунок 3.11 – Вигляд головної сторінки для авторизованого користувача

Після успішної авторизації користувач заходить на сторінку свого профілю для налаштування. (Рисунок 3.12).

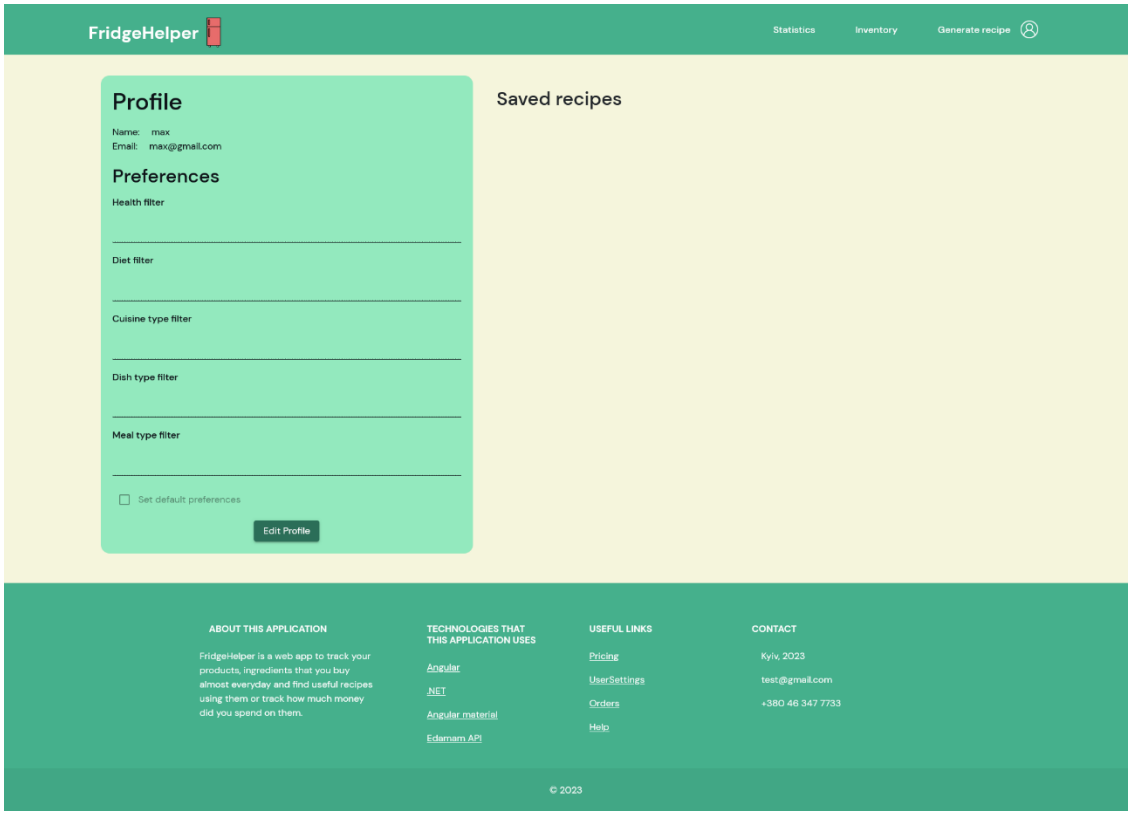


Рисунок 3.12 – Вигляд сторінки особистого профілю для нового користувача

Користувач переходить на сторінку редагування та обирає ті параметри, які йому до вподоби шляхом натискання на кнопку “Edit profile”. (Рисунок 3.13).

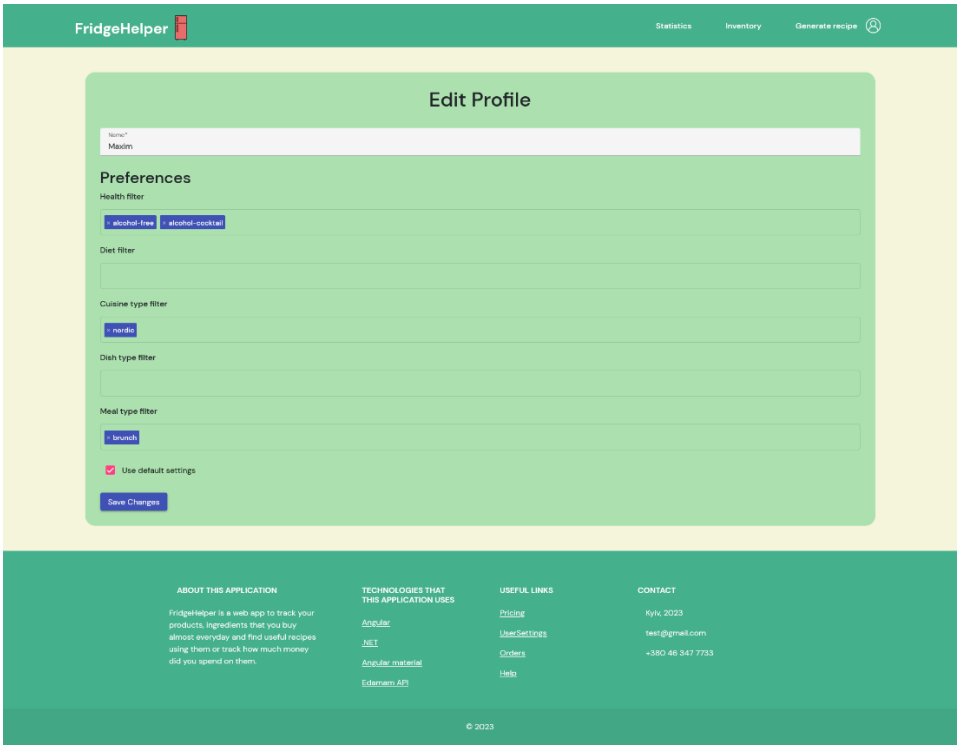


Рисунок 3.13 – Вигляд сторінки зміни налаштувань та редагування інформації

Користувач натискає на кнопку “Save changes”, відбувається перехід на сторінку особистого профілю. (Рисунок 3.14).

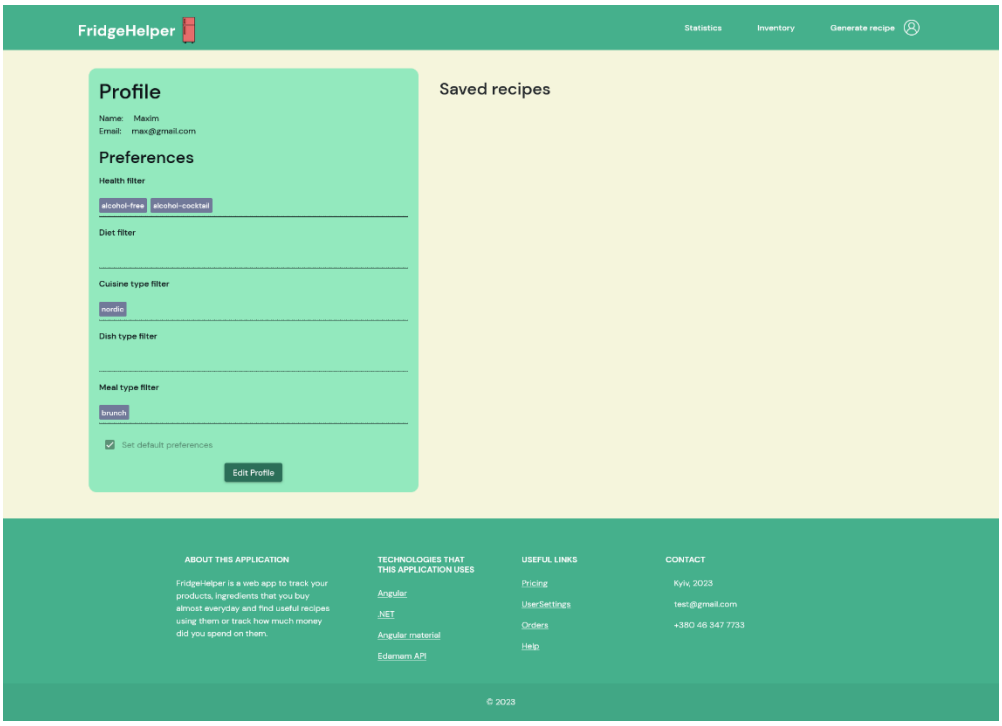


Рисунок 3.14 – Вигляд сторінки особистого профілю після зміни налаштувань

Якщо користувач проставив прапорець біля меню “Use as default”, то його вподобання будуть обрані за замовчуванням на сторінці пошуку рецептів. Користувач переходить на дану сторінку.

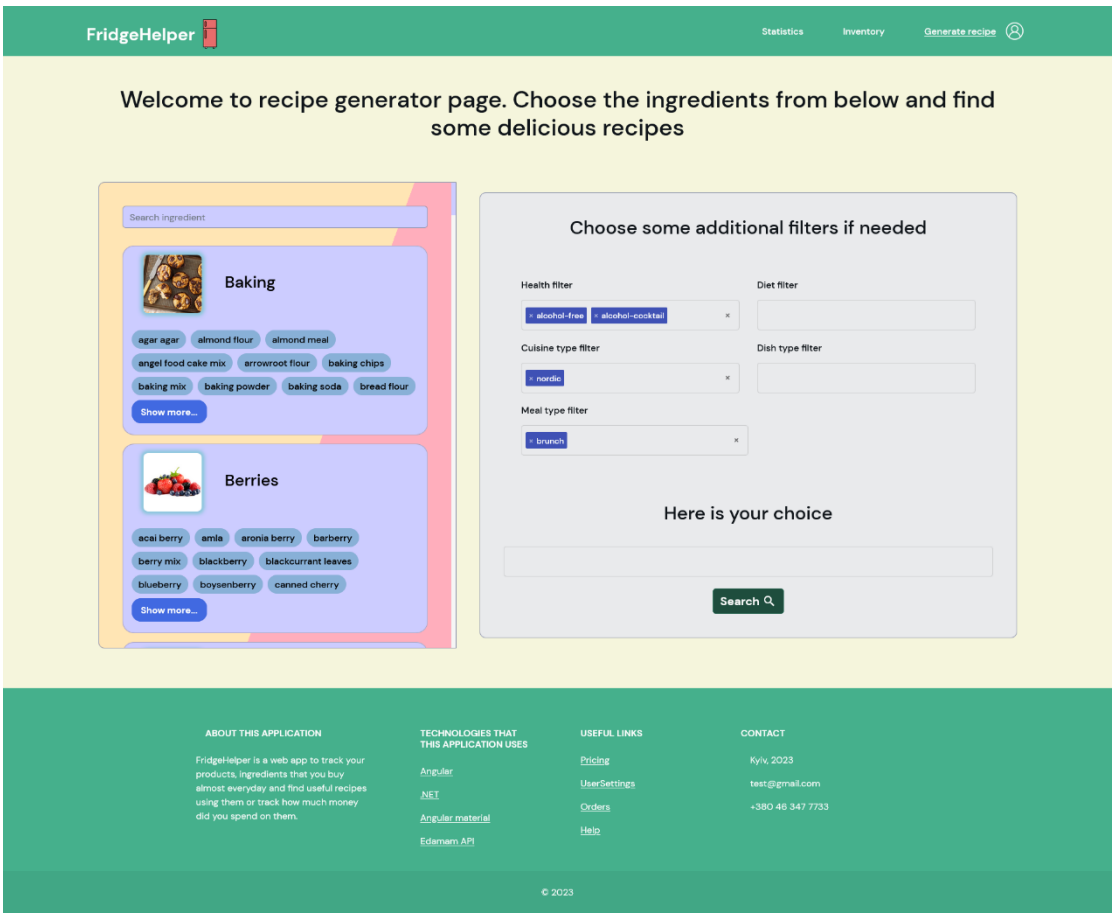


Рисунок 3.15 – Вигляд сторінки вибору параметрів для пошуку рецептів з використанням налаштувань за замовчуванням

Користувач обирає інгредієнти, додаткові параметри при необхідності. Натискає на кнопку пошуку. З’являється перелік знайдених рецепті, користувач відкриває один з них та бачить у верхньому правому куті вікна кнопку, яка відповідає за збереження даного рецепту до списку улюблених. Натискає на неї. Рецепт є збереженим, вигляд кнопки змінюється та він може зайти до сторінки особистого профілю та побачити, що даний рецепт був збережений. При бажані – користувач може знову відкрити повну інформацію про нього. (Рисунки 3.16 – 3.19).

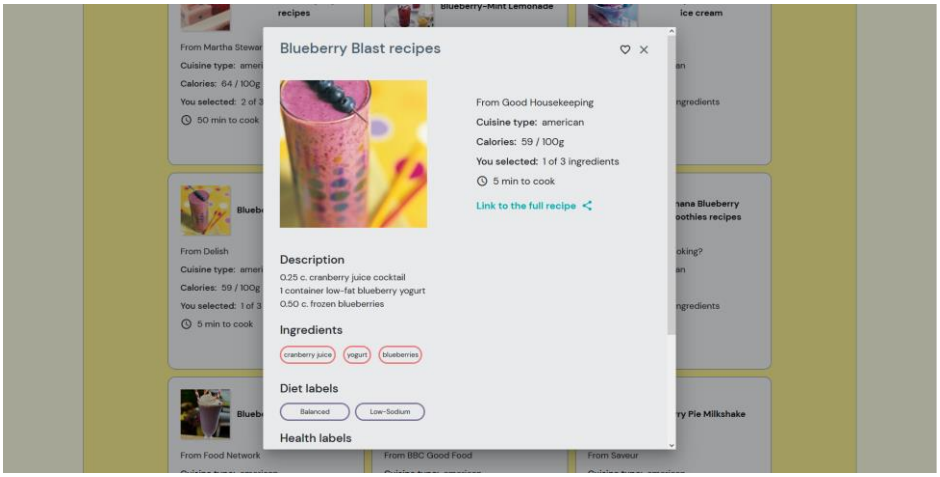


Рисунок 3.16 – Вигляд вікна з повним описом рецепту для авторизованого користувача

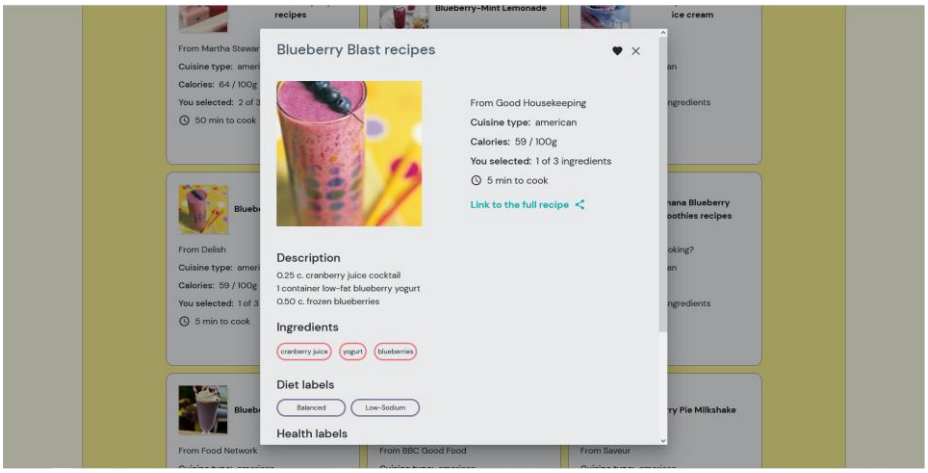


Рисунок 3.17 – Збереження рецепту до списку улюблених

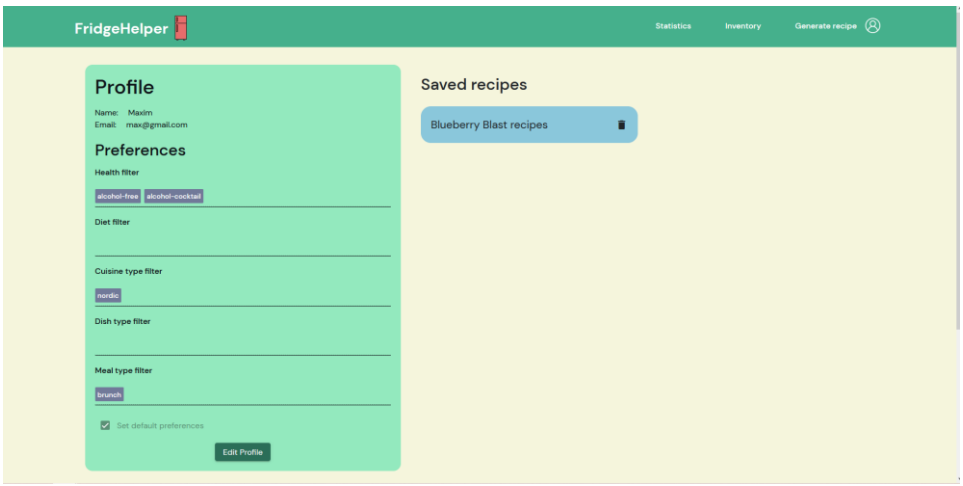


Рисунок 3.18 – Список збережених рецептів на сторінці особистого профілю

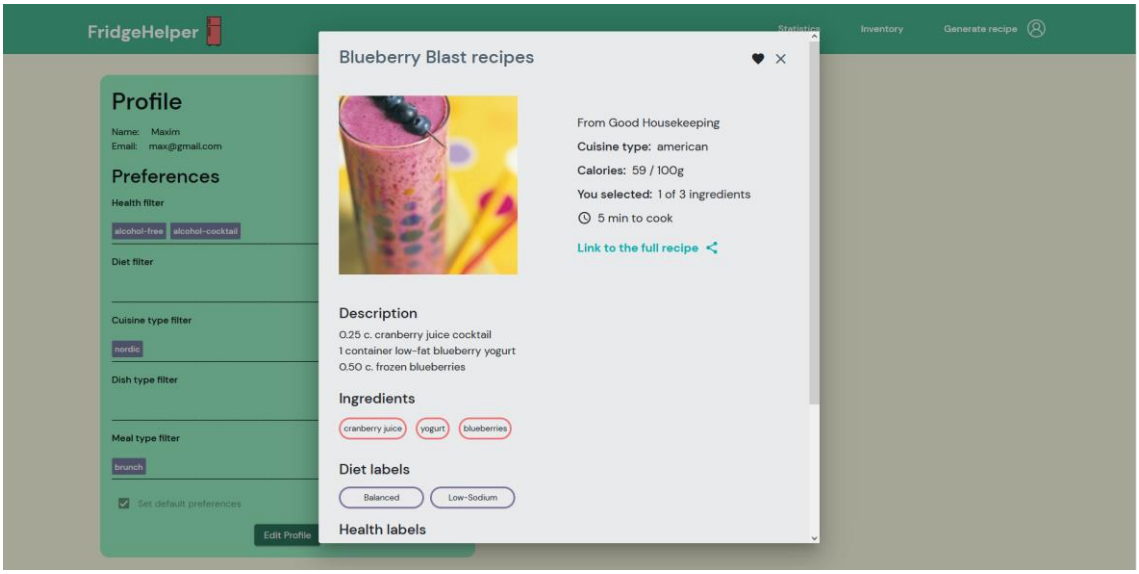


Рисунок 3.19 – Повна інформація про рецепт, відкрита на сторінці особистого профілю

Користувач може видаляти збережені рецепти зі списку. (Рисунки 3.20 – 3.21).

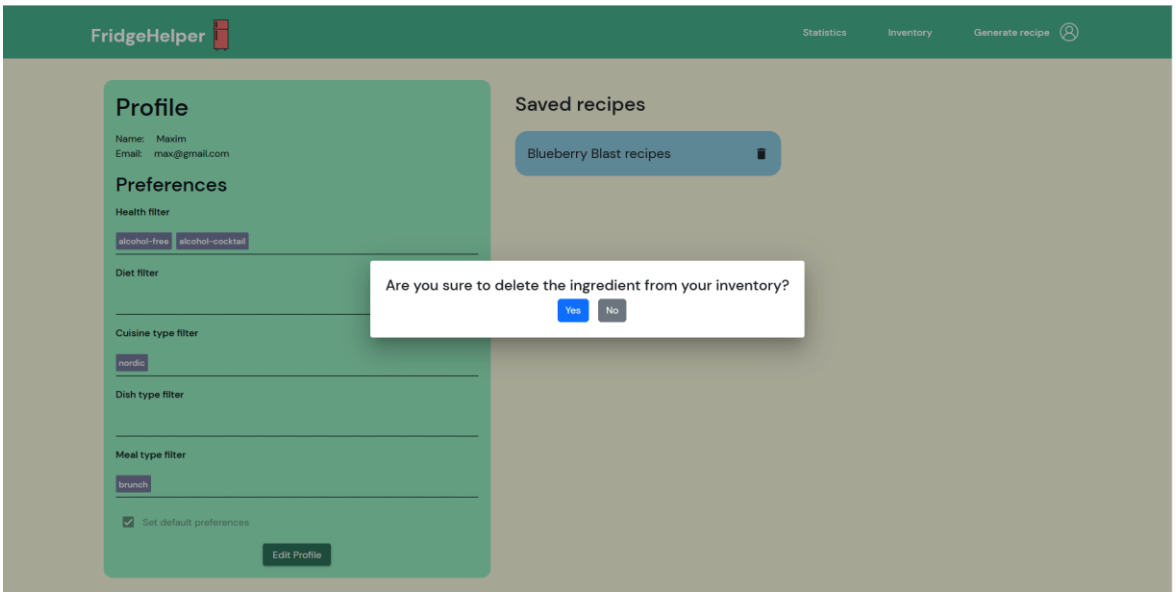


Рисунок 3.20 – Підтвердження видалення запису зі списку збережених рецептів

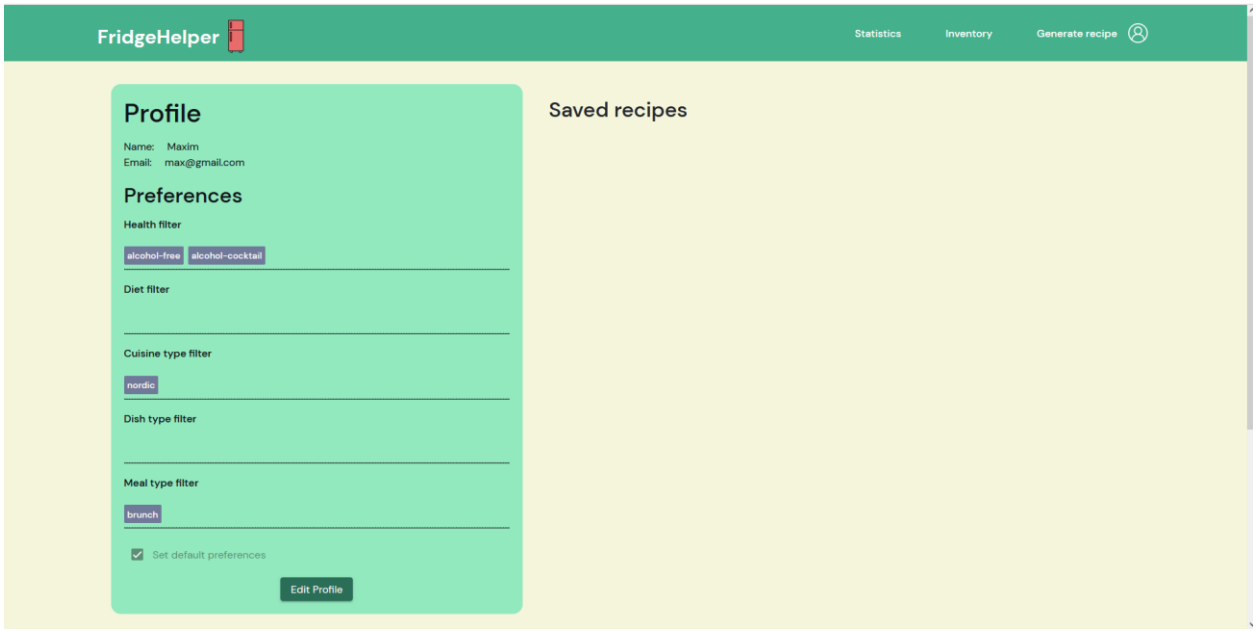


Рисунок 3.21 – Результат видалення рецепту зі списку збережених

Користувач може перейти на сторінку відстеження запасів продуктів. Для цього він натискає на посилання “Inventory”, що знаходиться на навігаційній панелі та переходить на сторінку інвентаря. Тут користувач натискає на кнопку додавання продукту, з’являється форма, користувач заповнює всі необхідні дані та натискає на кнопку “Add to inventory”. (Рисунки 3.22 – 3.24).

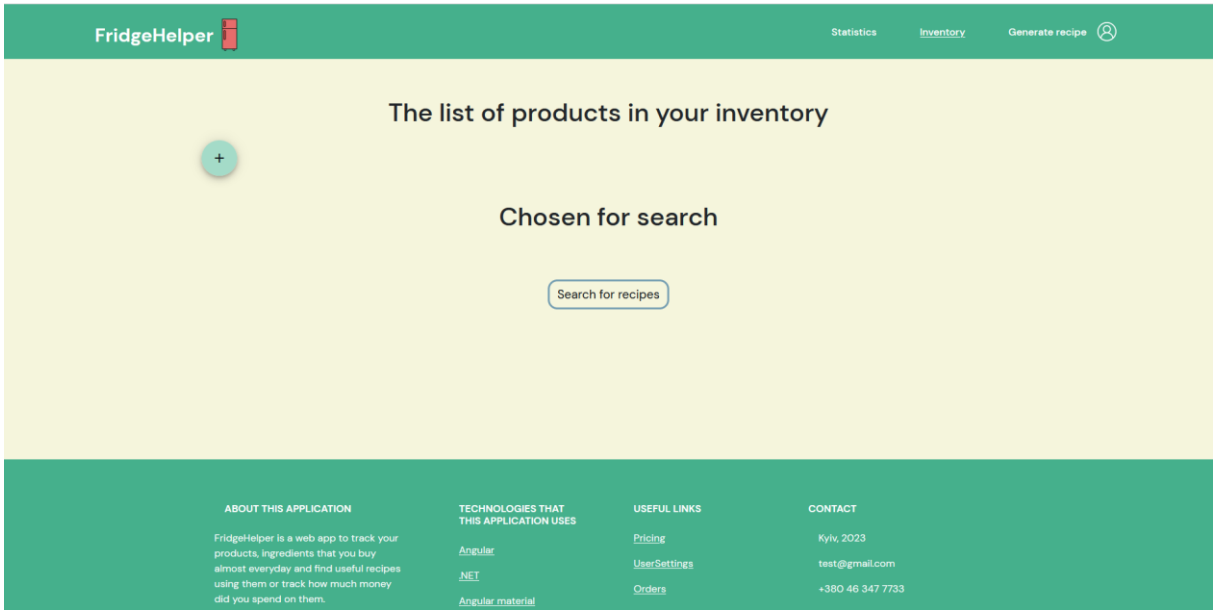


Рисунок 3.22 – Вигляд сторінки інвентаря

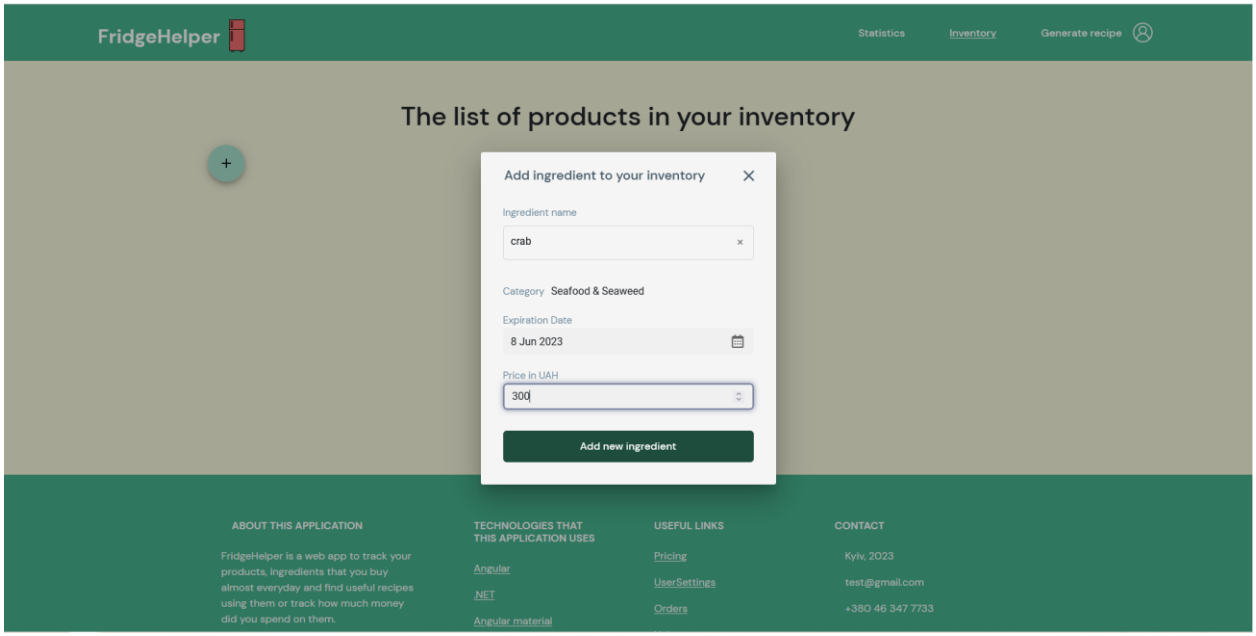


Рисунок 3.23 – Додавання нового продукту до інвентаря

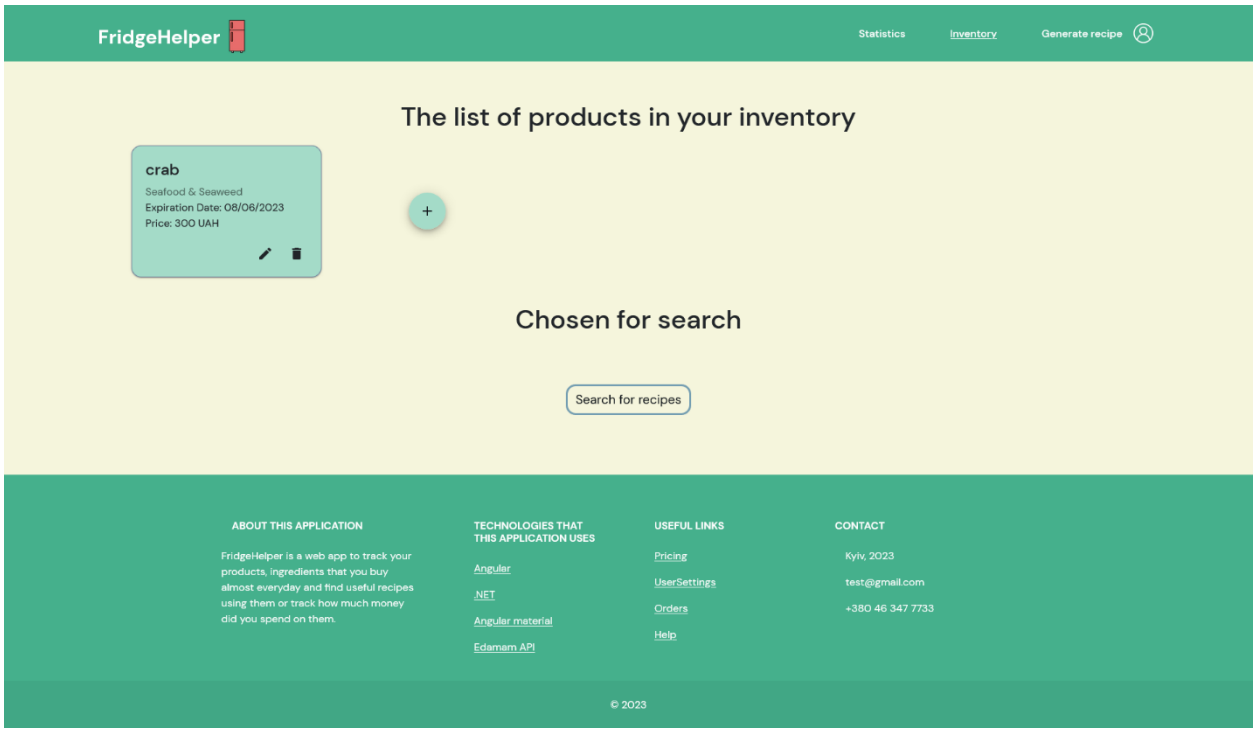


Рисунок 3.24 – Доданий продукт відображається на сторінці

Користувач може видаляти, редагувати продукти в інвентарі шляхом натискання на відповідні кнопки на картці продукту. Та користувач може обрати продукт для пошуку за ним рецептів або перейти на сторінку

статистики, де зображена інфографіка витрат коштів на продукти за певний період. (Рисунок 3.25).

Рисунок 3.25 – Вигляд сторінки статистики з інформацією про грошові витрати користувача на продукти

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів

Графічний матеріал

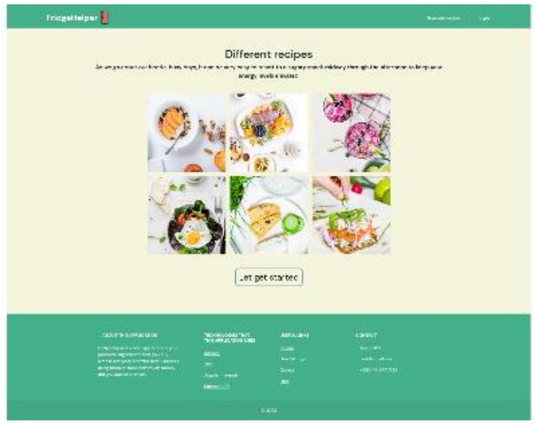
КПІ.ІП-9128.045440.06.99

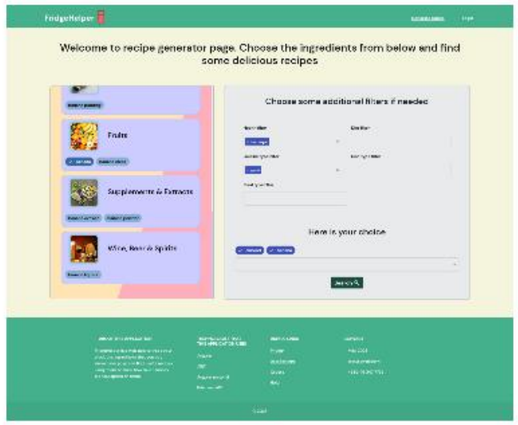
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

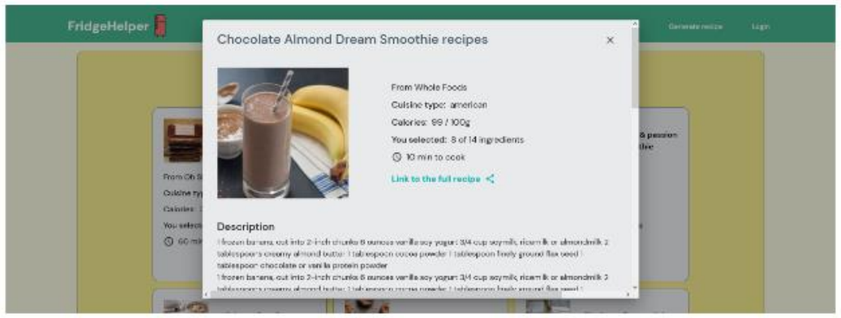
Нормоконтроль:
_____ Катерина ЛІЩУК

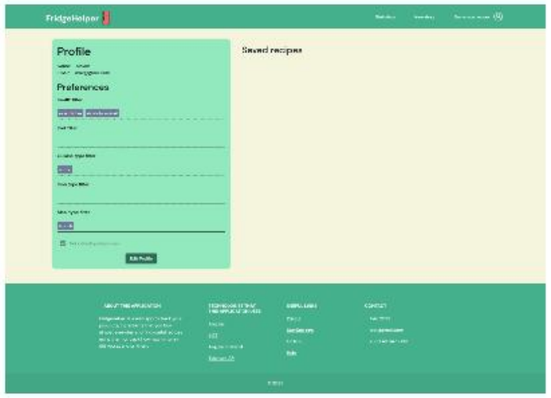
Виконавець:
_____ Максим ШЕЙКО

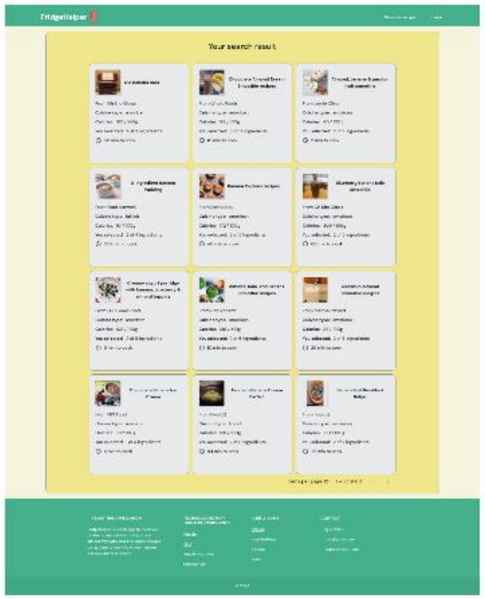
Київ – 2023

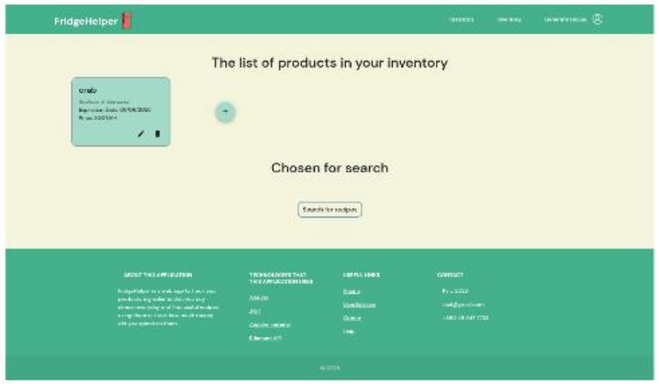


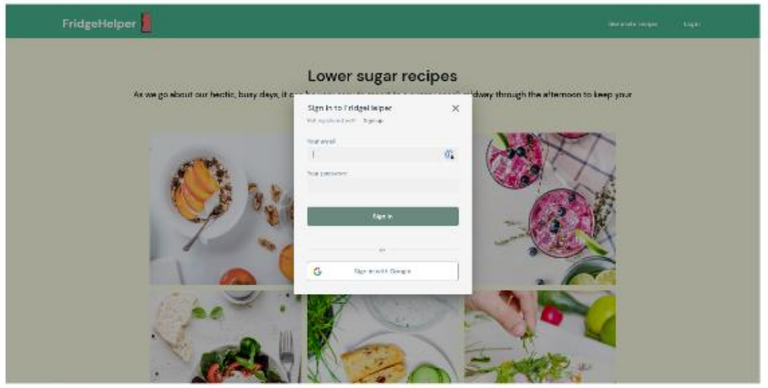


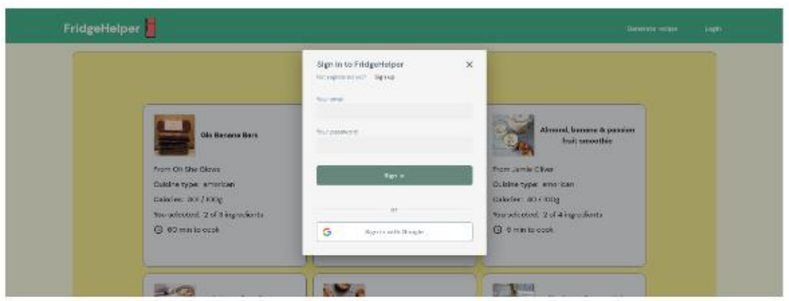






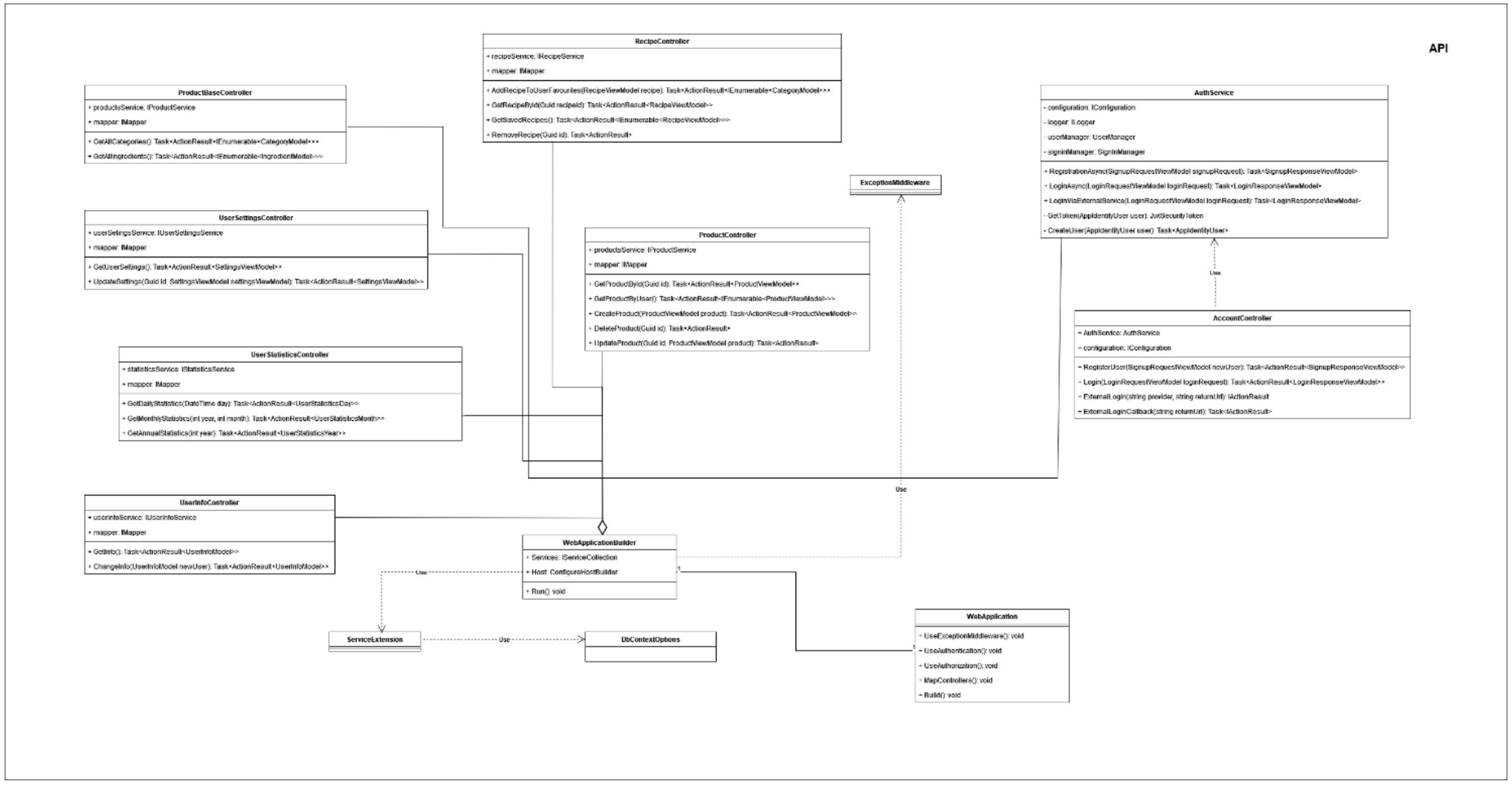






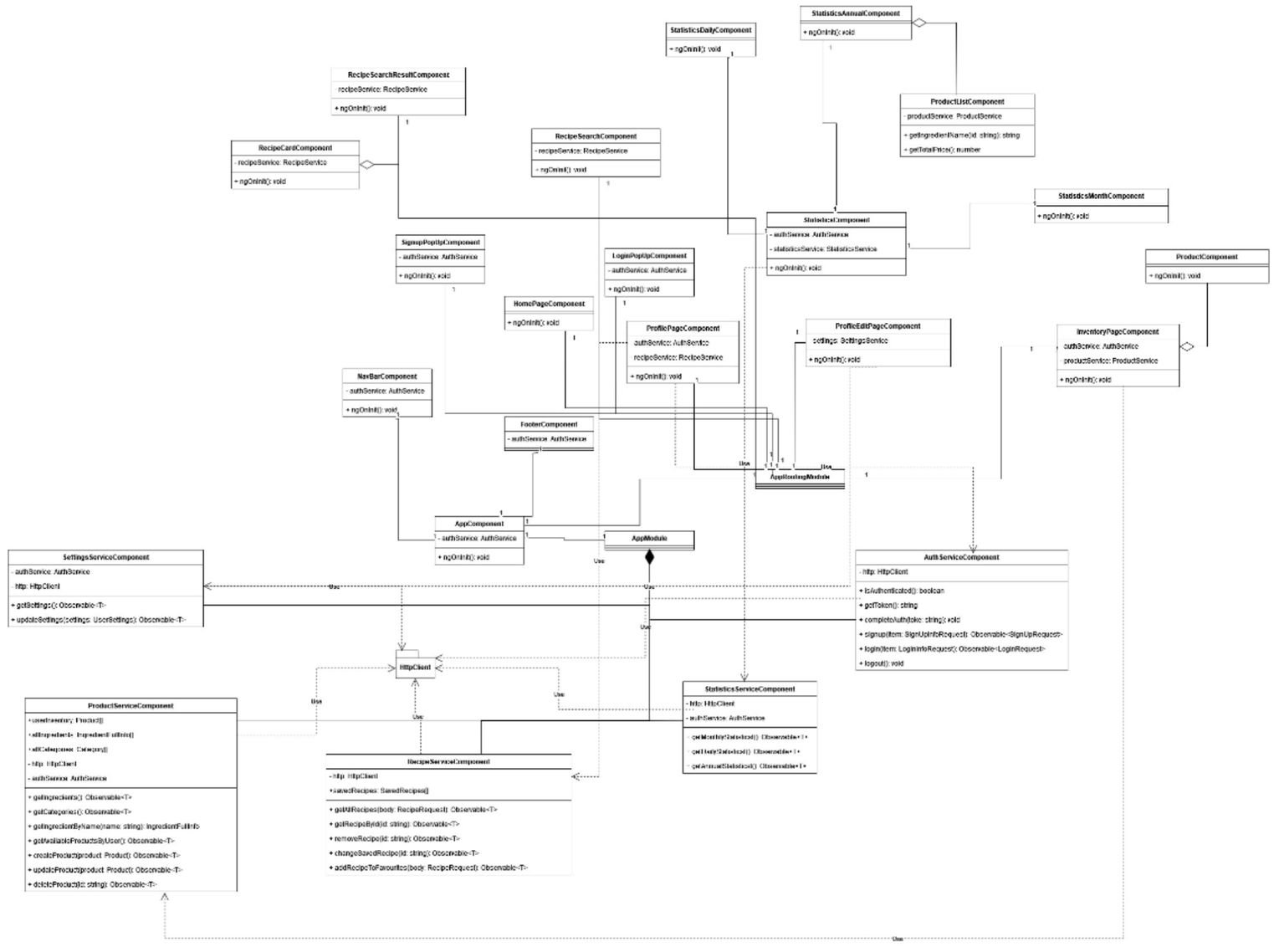
					КПІ.ІП-9128.045440.06.99.КЕ
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм
Розробив		Шейко М.О.			
Перевірив		Халус О.А.			
Т. кон.					
Н. кон.		Ліщук К.І.			Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів
Затвердив		Жаріков Е.В.			

Літера	Маса	Масштаб
Аркуш	Аркушів	
КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-91		



					КПІ.ІП-9128.045440.06.99.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Шейко М.О.				
Перевірив		Халус О.А.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-91
Затвердив		Жаріков Е.В.				

КПІ.ІП-9128.045440.06.99.ССМ



					КПІ.ІП-9128.045440.06.99.ССМ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Шейко М.О.					
Перевірив		Халус О.А.				Аркуш	Аркушів
Т. кон.							
Н. кон.		Ліщук К.І.			Веб-застосунок для відстеження запасів продуктів та підбору рецептів кулінарних блюд із наявних продуктів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-91	
Затвердив		Жаріков Е.В.					