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

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Веб-застосування «Кулінарний помічник»*

Виконав: студент IV курсу, групи *ІП-61 Брідня Вероніка Ігорівна* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

доц. каф. ТК, к.т.н. доц., Лісовиченко О.І.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Брідні Вероніці Ігорівні
(прізвище, ім’я, по батькові)

1. Тема проєкту *Веб-застосування «Кулінарний помічник»*

керівник проєкту *Ліщук Катерина Ігорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: загальні положення, опис функціональних та нефункціональних вимог, аналіз відомих програмних продуктів, визначення цілі та задач розробки

2) Інформаційне та алгоритмічне забезпечення: опис вхідних даних, опис бази даних, визначення та обґрунтування методів розв’язання математичних задач

3) Програмне та технічне забезпечення: моделювання та аналіз програмного забезпечення, опис архітектури веб-застосування, аналіз безпеки даних

4) Технологічний розділ: випробування та опис розгортання веб-застосування

5) Керівництво користувача, опис програми

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*21.02.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*03.03.2020*	
3.	*Постановка та формалізація задачі*	*19.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*30.03.2020*	
5.	*Алгоритмізація задачі*	*05.04.2020*	
6.	*Моделювання програмного забезпечення*	*10.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*14.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*20.04.2020*	
9.	*Розробка програмного забезпечення*	*29.04.2020*	
10.	*Налагодження програми*	*11.05.2020*	
11.	*Виконання графічних документів*	*15.05.2020*	
12.	*Оформлення пояснювальної записки*	*17.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Вероніка БРІДНЯ
(підпис)

Керівник _____ Катерина ЛІЩУК
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6104.045440.01.81	Пояснювальна записка	92	
3	А4	КПІ.ІП-6104.045440.02.91	Технічне завдання	12	
4	А4	КПІ.ІП-6104.045440.03.34	Керівництво користувача	33	
5	А4	КПІ.ІП-6104.045440.04.13	Опис програми	47	
6	А3	КПІ.ІП-6104.045440.05 СС	Схема структурна варіантів використань	1	
7	А3	КПІ.ІП-6104.045440.06 СБД	Схема бази даних	1	
8	А3	КПІ.ІП-6104.045440.07 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6104.045440.00.90		
Розробн.		Брідня В.І.			Відомість дипломного проєкту	Літ.	Лист 4 / Листів 1
Керівн.		Ліщук К.І					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 22 рисунка, 40 таблиць, 3 додатки, 11 джерел.

Мета. Дипломний проєкт присвячений розробці програмного забезпечення для полегшення процесу пошуку кулінарних рецептів серед наявних за багатьма критеріями, а також спрощення представлення рецептів у вигляді, придатному для такого пошуку, з метою передачі рецептів цільовій аудиторії.

Основними задачами є створення зручного інтерфейсу для керування рецептами та продуктами, розробка алгоритмів пошуку рецептів за багатьма критеріями та збереження критеріїв у вигляді фільтрів.

У розділі аналізу вимог та програмного забезпечення були поставлені цілі та задачі, визначені основні вимоги, сценарії роботи та ролі користувачів.

У розділі інформаційного та алгоритмічного забезпечення були визначені вхідні дані та структура бази даних, розроблені алгоритми пошуку за інгредієнтами з різними умовами.

У розділі програмного та технічного забезпечення було виконано моделювання програмного забезпечення та розроблено архітектуру серверної та клієнтської частин.

У технологічному розділі було виконано випробування та описано процес розгортання програмного продукту.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУВАННЯ, КУЛІНАРНІ РЕЦЕПТИ, ХАРЧОВІ ПРОДУКТИ

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of four sections, contains 22 images, 40 tables, 3 applications, 11 sources.

Goal. The diploma project is devoted to the development of software to facilitate the search for recipes among available using many criteria, as well as to simplify the presentation of recipes in a form suitable for such a search, in order to convey recipes to the target audience.

The main tasks are to create a user-friendly interface for managing recipes and products, to develop algorithms for finding recipes by many criteria and to save the criteria as filters.

In the requirements and software analysis section goals and main tasks were determined, key requirements, work scenarios, and user roles were identified.

In the information and algorithms section the input data and the structure of the database were determined, the algorithms for searching by ingredients with different conditions were developed.

In the software and hardware section, software modeling was performed, and server and client part architectures were developed.

In the technological section, tests were performed, and the process of software product deployment was described.

KEY WORDS: WEB APPLICATIONS, CULINARY RECIPES, FOOD

Пояснювальна записка
до дипломного проєкту

на тему: Веб-застосування «Кулінарний помічник»_____
_____
_____

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Тег – ключове (релевантне) слово або термін, що асоціюється або надається фрагменту інформації, описуючи в такий спосіб фрагмент та дозволяючи здійснювати класифікацію на основі ключових слів та пошук інформації [8].

Токен – це електронний ключ доступу до певної інформації.

JWT – стандарт створення токенів для надання доступу до певної набору інформації з використанням формату JSON.

API – програмний інтерфейс застосування, сукупність засобів та правил, що вможливлюють взаємодію між окремими складниками програмного забезпечення або між програмним та апаратним забезпечення [7)].

Вебсайт – сукупність, пов’язаних між собою веб-сторінок, зазвичай об’єднаних одним доменним ім’ям [6].

Тест кейс – сукупність кроків, умов, вхідних параметрів та очікуваних результатів для випробування продукту та перевірки виконання зазначених вимог.

ВСТУП

У зв'язку з появою великої кількості різноманітних харчових продуктів і дослідженнями впливу окремих складників на організм людини, питання правильного підбору продуктів набирає важливості, особливо для молодого покоління. Все більше людей, обираючи страви, обмежують їх певним набором продуктів, який може бути обумовлений релігійними поглядами, моральними принципами, профілактикою захворювань, необхідністю зміцнення фізичного стану, наявністю алергій, медичними показаннями, тощо. З точки зору здорового харчування важливою є й збалансованість харчування, яка вимагає правильного співвідношення жирів, білків, вуглеводів, вітамінів та інших поживних речовин.

Збільшення різноманітності та виділення чи створення нових підкатегорій характерно не тільки для харчових продуктів, а й для багатьох сфер життя сучасної людини. Тож, питання ефективного зберігання сильно розгалужених ієрархічних систем і швидкого пошуку в них за декількома умовами, є не менш актуальним.

Створення веб-застосування «Кулінарний помічник» дозволить користувачам швидко і зручно шукати кулінарні рецепти, орієнтуючись на певні продукти чи цілі категорії та виключаючи інші, з можливістю відстежувати орієнтований вміст поживних речовин в інгредієнтах. Наддасть можливість економічно розраховувати можливі варіанти приготування їжі, враховуючи наявність певних продуктів в домашньому запасі, чи розрахувати необхідну кількість інгредієнтів, що необхідно купити.

Функціональність створення власних та використання стандартних добірок з певними інгредієнтами та іншими критеріями для подальшого пошуку, є особливо актуальною у зв'язку з тенденцією підбору унікальних дієт під конкретну людину.

Наявність визначеного списку продуктів та категорії з описом та складом кожного елементу, а також детального шаблону кулінарного рецепта,

спростить додавання та оновлення рецептів для кухарів та редакторів. Оскільки різноманітність продуктів постійно змінюються, важливою є й можливість підтримки адміністратором актуальності списку продуктів, шляхом створення, редагування опису та категоризації продуктів через веб-застосування без необхідності оновлення рецептів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Опис предметного середовища

1.1.1 Загальні положення

Кулінарний рецепт – вказівка про склад та спосіб приготування будь-якої страви [2)].

Кулінарний рецепт може містити:

– назву;

– зображення страви;

– загальний опис;

– покроковий опис процесу приготування;

– зображення процесу приготування;

– список інгредієнтів та їх вага чи кількість;

– список альтернативних інгредієнтів;

– спосіб приготування;

– тип страви (за країною походження, за часом доби, за наявністю продуктів тваринного походження, тощо);

– відео процесу приготування;

– примітки;

– назву автора;

– час приготування;

– кількість порцій;

– вагу страви;

– наявність поживних речовин;

– орієнтовану калорійність;

– складність приготування;

– інше.

Якщо всі рецепти статичні та формуються розробником клієнтської частини, то кожен з них, задля досягнення привабливого дизайну, може виглядати по-різному в залежності від наявної інформації, зображень (їх кількості, кольорів, розмірів, залежності від тексту), наявності додаткових елементів, тощо. Проте, якщо рецепти можуть додаватися користувачами, то шаблон рецепту має бути заздалегідь визначений.

Аналізуючи наявні веб-застосування для роботи з рецептами, були виведені декілька найбільш розповсюджених загальних шаблонів рецепту.

а) Лише назва та опис рецепту.

Вся додаткова інформація зазначається в описі у вигляді цільного тексту.

Переваги: легкий для розробки, займає мало пам'яті та швидко завантажується, легко адаптується під будь-які розміри екрану.

Недоліки: малоінформативний та незручний для користувача, не приваблює, майже унеможливлює (або вимагає машинного навчання) пошук за додатковими критеріями (інгредієнти, тип страви, час приготування, тощо).

б) Назва рецепту, опис, фотографія страви, окремі елементи для різних критеріїв пошуку (інгредієнти, калорійність, час приготування, тощо).

Переваги: через наявність лише одного зображення все ще займає не багато пам'яті та текст може адаптуватися під його розміри, більш інформативний та привабливий для користувача, спрощено пошук за певними критеріями.

Недоліки: зросла складність створення універсального дизайну сторінки, нема можливості додати зображення процесу приготування, зручно редагувати.

в) Назва рецепту, опис, фотографія страви, окремі елементи для різних критеріїв пошуку, галерея зображень.

Переваги: збільшена інформативність та привабливість для користувача, одразу видно всі зображення приготування.

Недоліки: рецепт може займати багато пам'яті, додаткова складність у розробці галереї, з точку зору редактора – необхідність посилатися в описі на певні зображення та слідкувати за їх послідовністю, з точку зору користувача – необхідність багато разів переходити до галереї та повертатися до опису процесу приготування.

г) Назва рецепту, фотографія страви, окремі елементи для різних критеріїв пошуку, покроковий опис рецепту з можливістю додання фотографії до кожного кроку.

Переваги: можливість додання необмеженої кількості кроків та розміщення зображень поряд з їх описом, не сильно ускладнює дизайн сторінки, можливість переміщення/видалення зображення та опису кроку як одного елементу.

Недоліки: рецепт не завжди може бути коректно розділений на кроки, один крок може вимагати декілька зображень, необхідність контролювати розміри зображень (щоб вони привабливо виглядали незалежно від пропорцій).

д) Повноцінний конструктор рецептів з можливістю додавання різноманітних елементів та розміщення їх в довільних місцях.

Переваги: гнучкість створення рецепту, можливість збільшення привабливості окремих рецептів для користувачів, можливість зробити рецепти більш унікальними.

Недоліки: дуже ускладнюється як процес збереження таких рецептів так і їх показ на сторінці, проблеми створення дизайну переходять до редакторів.

е) Специфічні та комбіновані шаблони.

Для того, щоб зробити веб-застосування якомога зручнішим для користувачів та редакторів, був обраний четвертий варіант: назва рецепту, фотографія страви, окремі елементи для різних критеріїв пошуку, покроковий опис з фотографіями.

Програмне забезпечення для збереження та пошуку рецептів може мати різні вигляди:

– веб-застосування;

– застосування під конкретну платформу (наприклад, мобільне застосування);

– інше (наприклад, гібридні платформи, консольний додаток, модуль іншого забезпечення).

Розробка застосувань під конкретні платформи має ряд переваг:

– можливість використовувати специфічні функції платформи;

– легше гарантувати дієздатність забезпечення на визначеній платформі;

– можливість організувати швидкий доступ до даних, через те що всі або частина з них зберігається на пристрої;

– можливість організувати доступ до всієї або певної інформації без підключення до інтернету.

Недоліками застосувань під конкретні платформи є:

– використання застосувань на інших платформах потребує значних змін в програмному забезпеченні чи взагалі створення нового;

– на кожному новому пристрої потрібно окремо встановлювати застосування;

– можливе отримання неактуальних даних;

– для перевірки роботи на певних платформах обов’язкове має бути її екземпляр чи налаштована віртуальна машина (що не гарантує дієздатності на реальному пристрої);

– може займати значну кількість пам’яті на пристрої;

– можлива втрата даних через несправність пристроїв;

– необхідність підтримка та оновлення забезпечення платформах, шо швидко змінюються.

Перевагами розробки веб-застосувань є:

– можливість використання одного застосування для різних платформ;

– можливість швидкого доступу до ресурсів з різних пристроїв без необхідності встановлення;

– не займає пам'ять на пристроях;

– не потребує особливих знань роботи з окремими платформами;

– не потребує постійних оновлень забезпечення, адже старі функції майже не припиняють підтримуватися в браузерах, а оновлення орієнтовані більше на додаткові покращення.

Недоліками веб-застосувань є:

– може бути складно створити дизайн, що буде виглядати прийнятним на різних платформах та пристроях, та інколи доводиться порушувати універсальність коду;

– певні функції можуть не працювати у деяких браузерах чи платформах;

– необхідність розгортати сервер, який може використовувати багато енергії та пам'яті;

– можлива втрата всіх даних через несправність серверу;

– необхідність обов'язкового підключення до інтернету.

Аналізуючи наведені переваги та недоліки я обрала веб-застосування як найбільш прийнятний для моїх задач та навичок варіант.

1.1.2 Опис функціональної моделі

Спочатку необхідно визначити ролі дійових осіб та функції, які їм доступні.

Веб-застосування для кулінарних рецептів має, перш за все, надавати можливість пошуку та перегляду рецептів. Оскільки всі рецепти є публічними, а пошук та перегляд рецептів лише отримує дані та ніяк їх не змінює, то користувачам, які бажають лише швидко знайти потрібний рецепт, нема потреби проходити процес реєстрації. Оскільки рецепти містять інгредієнти та існує необхідність пошуку за ними, то зрозумілим є, що користувач також

повинен мати змогу переглядати список продуктів без реєстрації. Тож, виникає перша роль – гість (незареєстрований користувач).

Проте, якщо користувач матиме бажання зберегти критерії пошуку (фільтр) для повторного використання у майбутньому, то застосунок повинен якимось чином ідентифікувати його при повторному вході. Для цього користувач повинен зареєструватися за допомогою унікального ім’я (для того, щоб відрізнити його від інших користувачів) та паролю (для того, щоб закрити доступ іншим користувачам до приватних даних), та використати їх при повторному вході. Тож, виникає наступна роль – зареєстрований користувач.

Оскільки розглядається програмне забезпечення призначене не лише для висвітлення рецептів якогось конкретного кухаря, а наповнення бази рецептів даними з інших джерел може порушувати авторські права, то користувач повинен мати змогу додавати власні рецепти. Логічним є те, що лише автор може змінювати та видаляти власні рецепти, для цього застосунок має його ідентифікувати, тож, ця функціональність теж належить до ролі зареєстрованого користувача.

Створення початкового списку продуктів, з детальним описом кожного, та визначення зв’язків між ними потребує занадто багато часу та зусиль, якщо додавати їх напряму в базу даних. Окрім цього, список продуктів може оновлюватися з часом. Тож, необхідно створити інтерфейс для додавання та редагування продуктів на клієнтському рівні. Проте, підтримка списку продуктів, впливає на коректність рецептів, і якщо будь-який зареєстрований користувач матиме змогу змінювати продукти, то він зможе порушити коректність даних та результатів пошуку рецептів. Тож, виникає ще одна роль – адміністратор. Адміністратор також може відповідати за створення публічних фільтрів та рецептів.

У додатку «Схема структурна варіантів використань» детально описані функції, що належать кожній ролі.

Нижче описані основні варіанти використання даного програмного забезпечення у вигляді таблиць.

Таблиця 1.1 - Варіант використання UC-1

Назва	Реєстрація користувача
Опис	Створення облікового запису користувачем та вхід в нього
Учасники	Гість (незареєстрований користувач)
Постумови	Створено обліковий запис користувача, виконано вхід в систему.
Основний сценарій	Користувач обирає пункт меню «Реєстрація». З'являється форма з наступними полями: ідентифікатора, повне ім'я, електронна пошта, пароль, підтвердження паролю. Користувач вводить усі необхідні дані та натискає кнопку реєстрації. З'являється повідомлення про успішну реєстрацію. Повне ім'я чи ідентифікатор користувача з'являється на екрані, стають доступними додаткові пункти меню (створення рецепту, перегляд власних рецептів).
Розширення сценаріїв	Введені некоректні дані (паролі не збігаються / поля ідентифікатору та/або паролів не заповнені / ідентифікатор не є унікальним): – кнопка реєстрації не активна, некоректні поля підсвічуються червоним

Таблиця 1.2 - Варіант використання UC-2

Назва	Реєстрація адміністратора
Опис	Адміністратор створює обліковий запис для нового адміністратора
Учасники	Адміністратор
Передумови	Адміністратор авторизований в системі
Постумови	Створено новий обліковий запис адміністратора
Основний сценарій	Адміністратор обирає пункт меню «Реєстрація адміністратора». З'являється форма з аналогічними для реєстрації користувача полями. Адміністратор заповнює поля вводу та натискає кнопку реєстрації. З'являється повідомлення про успішну реєстрацію адміністратора.
Розширення сценаріїв	Введені некоректні дані: – кнопка реєстрації не активна, некоректні поля підсвічуються Час авторизації адміністратора в системі вичерпано: – з'являється повідомлення про відсутність прав доступу; виникає вихід з облікового запису та перенаправлення на сторінку входу в систему.

Таблиця 1.3 - Варіант використання UC-3

Назва	Вхід та вихід з облікового запису
Опис	Аутентифікація користувача в системі
Учасники	Зареєстрований користувач, адміністратор
Передумови	Користувач має бути зареєстрованим в системі
Постумови	Виконано вхід в обліковий запис користувача
Основний сценарій	Користувач обирає в меню сторінку входу в обліковий запис. З'являється форма з наступними полями: ідентифікатор та пароль. Користувач вводить усі необхідні дані та натискає кнопку входу. З'являється повідомлення про успішний вхід в систему. Ідентифікатор користувача з'являється на екрані, стають доступними додаткові пункти меню. Після натиснення опції виходу в головному меню, ідентифікатор користувача зникає з екрана, усі додаткові функції перестають бути доступними для користувача.
Розширення сценаріїв	Введено некоректний пароль чи ідентифікатор: – виникає повідомлення про помилку.

Таблиця 1.4 - Варіант використання UC-4

Назва	Перегляд продуктів
Опис	Перегляд списку продуктів, розподілених за категоріями, та їх детального опису
Учасники	Гість, зареєстрований користувач, адміністратор
Передумови	Список продуктів заповнений адміністратором
Основний сценарій	Користувач обирає в меню пункт «Список продуктів». Виконується перехід на сторінку з продуктами, які не мають категорій. Список продуктів виглядає як набір карток з назвою та зображенням продукту. Після натискання на будь-який продукт, виконується перехід до сторінки цього продукту. На сторінці з’являються: кнопка повернення до попередньої сторінки та початкової сторінки; секція Деталі, де знаходиться збільшене зображення продукту, його опис, калорійність, вміст поживних речовин, та список категорій, до яких він відноситься; список продуктів, для яких цей продукт є категорією (в такому ж вигляді, як на основній сторінці, проте з вказанням, до яких ще категорій відноситься конкретний продукт). Після натиснення на продукт зі списку чи на назву категорії, виконується аналогічний перехід до сторінки цього продукту.

Таблиця 1.5 - Варіант використання UC-5

Назва	Пошук рецептів
Опис	Пошук рецептів за заданими критеріями
Учасники	Гість, зареєстрований користувач, адміністратор
Постумови	Показ списку рецептів, що відповідають заданим критеріям
Основний сценарій	Користувач обирає в меню пункт «Пошук рецептів». Виконується перехід на сторінку, що складається з двох частин: секції для критеріїв (Загальні обмеження, Фільтри, Інгредієнти, Тегі, Прев'ю) та список рецептів (у вигляді карток, на яких розташовані назва рецепту, зображення, загальний опис та кнопка Деталі). Користувач заповнює критерій в перших чотирьох секціях, перевіряє у секції Прев'ю та натискає кнопку «Застосувати фільтр». Перша сторінка рецептів, що відповідають заданим критеріям з'являється на екрані. Далі можна змінювати кількість рецептів на сторінці та переміщатися між сторінками за допомогою навігаційного меню до та після списку рецептів.
Розширення сценаріїв	– Якщо нема рецептів за заданими критеріями: з'являється повідомлення про відсутність рецептів. – Пошук серед результатів за назвою, шляхом введення частини назви в поле «Пошук серед результатів» – Повернення до повного списку рецептів, шляхом натискання кнопки «Усі рецепти» – Збереження критеріїв пошуку (для власного використання зареєстрованим користувачем чи для публічного доступу адміністратором) та використання доступних в секції Фільтри.

Таблиця 1.6 - Варіант використання UC-6

Назва	Перегляд рецепту
Опис	Перегляд детального опису кулінарного рецепта
Учасники	Гість, зареєстрований користувач, адміністратор
Передумови	Рецепт створено користувачем чи адміністратором
Основний сценарій	Користувач обирає будь-який рецепт на сторінці пошуку рецептів та натискає кнопку «Деталі». Виконується перехід на сторінку обраного рецепта. На сторінці знаходяться наступні елементи: секція основних відомостей (назва страви, основне зображення страви, тривалість приготування, калорійність, кількість порцій, загальний опис), список інгредієнтів (для кожного з них вказано: назва, вага, обов'язковість, примітка від автора та кнопка перегляду опису продукту), примітка до вибору інгредієнтів, список ключових слів, покроковий опис рецепту (кожен крок може містити зображення, назву та опис).
Розширення сценаріїв	– Якщо якийсь елемент не вказаний автором рецепту, то він не показується на сторінці. – Можливий перехід до деталей конкретного продукту, шляхом натиснення кнопки поряд з його назвою.

Таблиця 1.7 - Варіант використання UC-7

Назва	Керування власними рецептами
Опис	Перегляд власних рецептів, редагування їх, видалення непотрібних та створення нових.
Учасники	Зареєстрований користувач, адміністратор
Основний сценарій	Користувач переходить на сторінку створення рецепту, шляхом вибору відповідного пункту меню. З'являється форма створення рецепту, яка складається з наступних елементів: назва, опис, калорійність, кількість порцій, тривалість приготування, поле для додання ключових слів (тегів), кнопка для завантаження головного зображення, кнопка додання інгредієнту, примітка до списку інгредієнтів, кнопка додання кроку. Після заповнення даних, користувач натискає кнопку створення рецепту та отримує повідомлення про успішне створення. Користувач переходить на сторінку списку власних рецептів через меню, де з'являються всі створені їм рецепти (що не були видалені). На кожному рецепті розміщені три кнопки (Деталі, Видалення та Редагування). Після натиснення на кнопку Редагування з'являється аналогічна до створення форма. Після зміни певних деталей, користувач натискає кнопку Збереження рецепту та повертається до списку власних рецептів. Після натиснення на кнопку Видалення, рецепт зникає зі сторінки та більше ніколи не з'являється.
Розширення сценаріїв	– Якщо під час створення чи редагування рецепта були введені некоректні дані або не були заповнені обов'язкові: кнопки збереження та створення не доступні, а відповідні некоректні поля підсвічені.

Таблиця 1.8 - Варіант використання UC-8

Назва	Керування продуктами
Опис	Створення, видалення та редагування продуктів
Учасники	Адміністратор
Основний сценарій	Користувач переходить на сторінку створення рецепту, шляхом вибору відповідного пункту меню. З'являється форма створення рецепту, яка складається з наступних елементів: назва, опис, калорійність, кількість порцій, тривалість приготування, поле для додання ключових слів (тегів), кнопка для завантаження головного зображення, кнопка додання інгредієнту, примітка до списку інгредієнтів, кнопка додання кроку. Після заповнення даних, користувач натискає кнопку створення рецепту та отримує повідомлення про успішне створення. Користувач переходить на сторінку списку власних рецептів через меню, де з'являються всі створені їм рецепти (що не були видалені). На кожному рецепті розміщені три кнопки (Деталі, Видалення та Редагування). Після натиснення на кнопку Редагування з'являється аналогічна до створення форма. Після зміни певних деталей, користувач натискає кнопку Збереження рецепту та повертається до списку власних рецептів. Після натиснення на кнопку Видалення, рецепт зникає зі сторінки та більше ніколи не з'являється.
Розширення сценаріїв	– Якщо під час створення чи редагування рецепта були введені некоректні дані або не були заповнені обов'язкові: кнопки збереження та створення не доступні, а відповідні некоректні поля підсвічені.

Далі описані функціональні вимоги для даного програмного забезпечення у вигляді таблиць.

Таблиця 1.9 - Функціональна вимога FR-1

Назва	Створення облікового запису користувача
Опис	Система має надавати можливість створення облікового запису користувача для подальшого входу в систему.

Таблиця 1.10 - Функціональна вимога FR-2

Назва	Вхід в обліковий запис зареєстрованого користувача
Опис	Система має надавати можливість входу в обліковий запис зареєстрованого користувача, шляхом введення ідентифікатора та паролю, визначених користувачем.

Таблиця 1.11 - Функціональна вимога FR-3

Назва	Вихід з облікового запису
Опис	Система має надавати можливість виходу з облікового запису користувача з забороною доступу до керування рецептами, фільтрами та продуктами.

Таблиця 1.12 - Функціональна вимога FR-4

Назва	Вхід в обліковий запис адміністратора
Опис	Система має надавати можливість входу в обліковий запис адміністратора, за допомогою визначеного в налаштуваннях ідентифікатора та паролю, або за допомогою даних, вказаних при реєстрації адміністратором.

Таблиця 1.13 - Функціональна вимога FR-5

Назва	Створення облікового запису адміністратора
Опис	Система має надавати можливість створення облікового запису нового адміністратора через обліковий запис іншого адміністратора

Таблиця 1.14 - Функціональна вимога FR-6

Назва	Перегляд списку кулінарних рецептів
Опис	Система має надавати можливість перегляду усіх рецептів, створених в системі, будь-яким користувачам.

Таблиця 1.15 - Функціональна вимога FR-7

Назва	Перегляд детального опису обраного рецепта
Опис	Система має надавати можливість перегляду детального опису будь-якого рецепту в системі, з показом всіх вказаних автором елементів (назва страви, загальний опис, тривалість приготування, кількість порцій, калорійність, основне зображення, список інгредієнтів та примітки до них, список тегів, зображень та описів до кожного кроку приготування) будь-якому користувачу.

Таблиця 1.16 - Функціональна вимога FR-8

Назва	Створення рецепту
Опис	Система має надавати можливість створення рецепту з вказанням визначеного списку елементів (назва страви, загальний опис, тривалість приготування, кількість порцій, калорійність, основне зображення, список інгредієнтів та примітки до них, список тегів, зображень та описів до кожного кроку приготування) за бажанням автора, але лише автентифікованим користувачам.

Таблиця 1.17 - Функціональна вимога FR-9

Назва	Редагування рецепту
Опис	Система має надавати можливість редагування усіх елементів створеного рецепта та додання нових лише автентифікованими авторами рецепту, або стандартних рецептів адміністраторами.

Таблиця 1.18 - Функціональна вимога FR-10

Назва	Видалення рецепту
Опис	Система має надавати можливість повного видалення рецептів автентифікованими авторами або адміністраторами, проте забороняти це для інших користувачів.

Таблиця 1.19 - Функціональна вимога FR-11

Назва	Створення рецепту на основі іншого рецепта
Опис	Система має надавати можливість створення рецепту, де значеннями за замовчуванням є значення іншого рецепту автора (лише для автентифікованих користувачів).

Таблиця 1.20 - Функціональна вимога FR-12

Назва	Перегляд списку продуктів
Опис	Система має надавати можливість перегляду списку продуктів, розділених за категоріями, для будь-яких користувачів.

Таблиця 1.21 - Функціональна вимога FR-13

Назва	Перегляд детального опису продуктів
Опис	Система має надавати можливість перегляду наявної в системі інформації про обраний продукт (назва, опис, зображення, список категорій, список продуктів, приблизний вміст основних поживних речовин та калорій на 100 г продукту) для будь-якого користувача.

Таблиця 1.22 - Функціональна вимога FR-14

Назва	Збереження критеріїв пошуку
Опис	Система має надавати можливість збереження поточних критеріїв у вигляді фільтру, з вказанням назви, під час пошуку рецептів аутентифікованими користувачами; а також створення публічних фільтрів для всіх користувачів в системі адміністратором.

Таблиця 1.23 - Функціональна вимога FR-15

Назва	Перегляд доступних фільтрів
Опис	Система має надавати можливість перегляду списку назв доступних фільтрів (для гостя – список публічних фільтрів, для зареєстрованого користувача – список публічних та власних фільтрів) та їх критеріїв на сторінці пошуку рецептів.

Таблиця 1.24 - Функціональна вимога FR-16

Назва	Використання доступних фільтрів
Опис	Система має надавати можливість використання одного з доступних фільтрів для пошуку рецептів, змінювати його значення для поточного пошуку та створення нових фільтрів на його основі.

Таблиця 1.25 - Функціональна вимога FR-17

Назва	Пошук рецептів за назвою
Опис	Система має надавати можливість пошуку рецептів за частиною назви страви для усіх користувачів.

Таблиця 1.26 - Функціональна вимога FR-18

Назва	Пошук рецептів за калорійністю
Опис	Система має надавати можливість пошуку рецептів, у яких калорійність відповідає вказаному проміжку, або відсутня в описі.

Таблиця 1.27 - Функціональна вимога FR-19

Назва	Пошук рецептів за тривалістю приготування
Опис	Система має надавати можливість пошуку рецептів, у яких тривалість приготування відповідає вказаному проміжку, або відсутня в описі.

Таблиця 1.28 - Функціональна вимога FR-20

Назва	Пошук рецептів за тегами
Опис	Система має надавати можливість пошуку рецептів, з вказанням обов’язкових та заборонених тегів). Має надаватися можливість перегляду та обрання тегів, що наразі є в рецептах системи.

Таблиця 1.29 - Функціональна вимога FR-21

Назва	Перегляд доступних продуктів під час пошуку рецептів
Опис	Система має надавати можливість перегляду усіх продуктів, продуктів що належать певним категоріям, та, за потребою, детального опису обраного продукту. Також має надаватися можливість пошуку продуктів за назвою.

Таблиця 1.30 - Функціональна вимога FR-22

Назва	Пошук рецептів за обов’язковими та забороненими інгредієнтами
Опис	Система має надавати можливість пошуку рецептів, з вказанням списку інгредієнтів, що обов’язково мають бути вказані в рецепті (або їх категорії чи підкатегорії), та списку інгредієнтів, які разом з їх підкатегоріями можуть бути лише опціональними в рецепті.

Таблиця 1.31 - Функціональна вимога FR-23

Назва	Пошук рецептів за наявними та обов’язковими інгредієнтами
Опис	Система має надавати можливість пошуку рецептів, з вказанням списку продукту, що обов’язково мають бути вказані в рецепті (або їх категорії чи підкатегорії), та списку продуктів, якими (разом з їх категоріями та вказаними обов’язковими інгредієнтами) обмежено список обов’язкових інгредієнтів рецепту.

Таблиця 1.32 - Функціональна вимога FR-24

Назва	Пошук рецептів за інгредієнтами без категорій
Опис	Система має надавати можливість пошуку рецептів за обов’язковими/забороненими та наявними/обов’язковими інгредієнтами без аналізу їх категорії та підкатегорії.

Таблиця 1.33 - Функціональна вимога FR-25

Назва	Швидкий пошук рецептів за назвою
Опис	Система має надавати можливість пошуку за частиною назви рецепту серед усіх наявних або отриманих у результаті фільтрації кулінарних рецептів, без необхідності отримувати нові дані з серверної частини.

Таблиця 1.34 - Функціональна вимога FR-26

Назва	Редагування продуктів
Опис	Система має надавати можливість редагування будь-яких продуктів в системі та їх категорій (без зміни ідентифікаторів) лише адміністраторам.

Таблиця 1.35 - Функціональна вимога FR-27

Назва	Видалення рецепту
Опис	Система має надавати можливість видалення продукту (без автоматичного видалення продуктів, пов’язаних з ним) з системи лише адміністраторам.

Таблиця 1.36 - Функціональна вимога FR-28

Назва	Створення продукту
Опис	Система має надавати можливість створення нового продукту, вказавши його назву, вмісту поживних речовин, опис, зображення та список продуктів (серед уже створених), що є його категоріями та підкатегоріями, але лише адміністратором.

Таблиця 1.37 - Функціональна вимога FR-29

Назва	Перегляд списку власних рецептів
Опис	Система має надавати можливість перегляду списку створених рецептів їх автором (автентифікованим користувачем), або адміністратором, якщо автора нема. З можливістю подальшого видалення чи редагування рецептів.

Аналізуючи функціональні вимоги та варіанти використання була побудована матриця трасування вимог (Рисунок 1.1 - Рисунок 1.1 -).

	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	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26	FR-27	FR-28	FR-28
UC-1 (реєстрація користувача)	■																												
UC-2 (реєстрація адміністратора)					■																								
UC-3 (вхід та вихід з облікового запису)		■	■	■																									
UC-4 (пошук рецептів)						■								■	■	■	■	■	■	■	■	■	■	■					
UC-5 (перегляд продуктів)												■	■																
UC-6 (перегляд рецепту)							■																						
UC-7 (керування продуктами)																										■	■	■	
UC-8 (керування рецептами)								■	■	■	■																		■

Рисунок 1.1 - Матриця трасування вимог

1.1.2 Опис нефункціональних вимог

Дане програмне забезпечення має відповідати наступним нефункціональним вимогам:

– коректний показ в браузерах Yandex версії 17.0 та вище, Google Chrome версії 48.0 та вище;

– мова клієнтського інтерфейсу: українська;

– адаптованість інтерфейсу під пристрої з діагоналлю екрану від 7 дюймів.

1.2 Аналіз відомих програмних продуктів

Наразі існує велика кількість програмних продуктів, що орієнтовані на роботу з рецептами. Серед них найбільш популярними є веб-застосування. Далі буде проаналізовано декілька з них та виявлено їх переваги й недоліки, що будуть виправлені в даній розробці.

Зауважимо, що для аналізу було обрано сайти, написані іноземними мовами, оскільки нема українських варіантів, що надають достатній об’єм функціональності.

1.2.1 Аналіз веб-застосування Webspoon.ru

Webspoon.ru (Рисунок 1.2 -) надає наступну функціональність:

– керування рецептами;

– перегляд інгредієнтів;

– пошук рецептів за інгредієнтами, типом та категорією страви;

– перегляд статей з порадами та добірками рецептів.

Перелік переваг:

– наявність добірок рецептів та статей з порадами до приготування;

– наявність списку інгредієнтів з фотографіями до кожного;

– наявність фільтрації за визначеними категоріями та типами страв.

Перелік недоліків:

– продукти розподілені лише за основними категоріями, нема підкатегорій;

– нема пошуку за категоріями інгредієнтів;

– можна обрати лише 4 інгредієнти для пошуку;

– нема можливості заборонити певні продукти;

– нема можливості зберегти критерії пошуку.

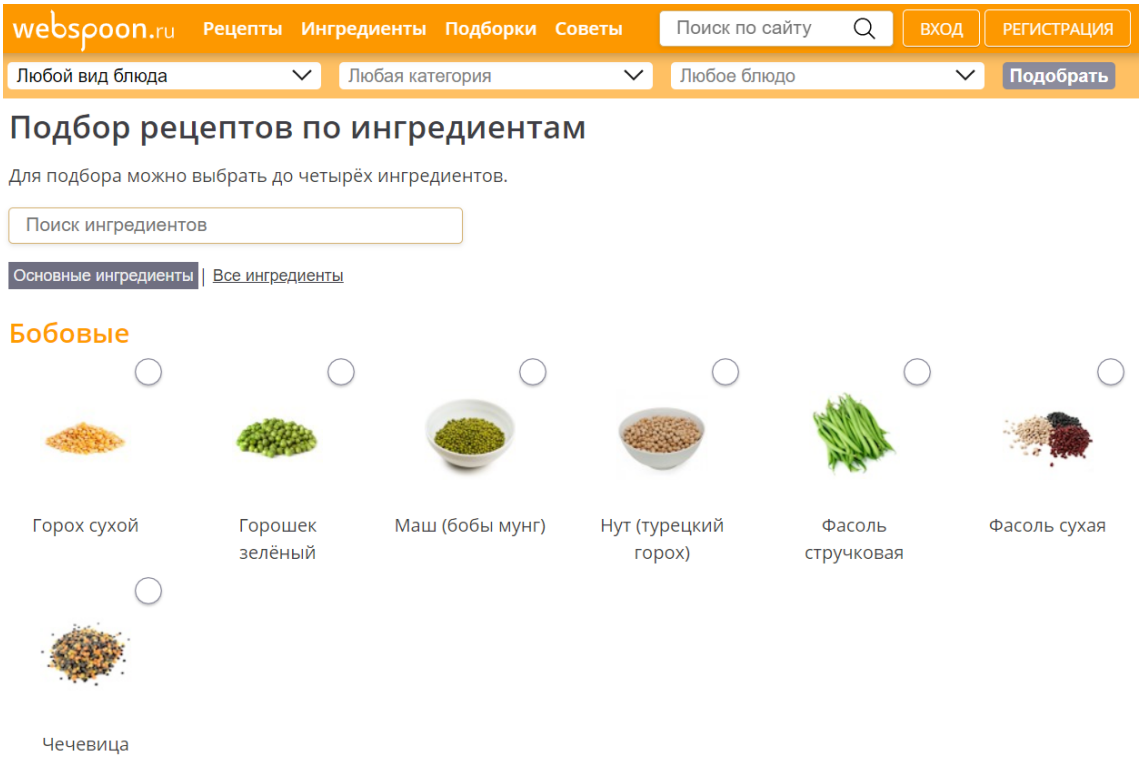


Рисунок 1.2 - Інтерфейс веб-застосування Webspoon.ru

1.2.2 Аналіз веб-застосування Povarenok.ru

Povarenok.ru (Рисунок 1.3 -) надає наступну функціональність:

– керування рецептами;

– пошук рецептів за інгредієнтами, назвою, країною походження та типом страви;

– перегляд рецептів за автором;

– оцінювання рецептів;

– перегляд кулінарного словника;

– перегляд різноманітних статей за темою.

Перелік переваг:

– можливість оцінити рецепти та подивитися їх рейтинг;

– наявність великої кількості рецептів у системі;

– наявність різноманітних статей пов’язаних з приготуванням страв та кулінарного словника;

– наявність фільтрації за визначеними типами страв;

– розширена категоризація рецептів;

– наявність відеорецептів;

– наявність швидкого пошуку за конкретними типами страв.

Перелік недоліків:

– нема визначеного списку продуктів та їх категорій;

– нема пошуку за категоріями інгредієнтів;

– нема можливості зберегти критерії пошуку;

– нема добірок рецептів;

– ускладнений процес додавання рецептів.

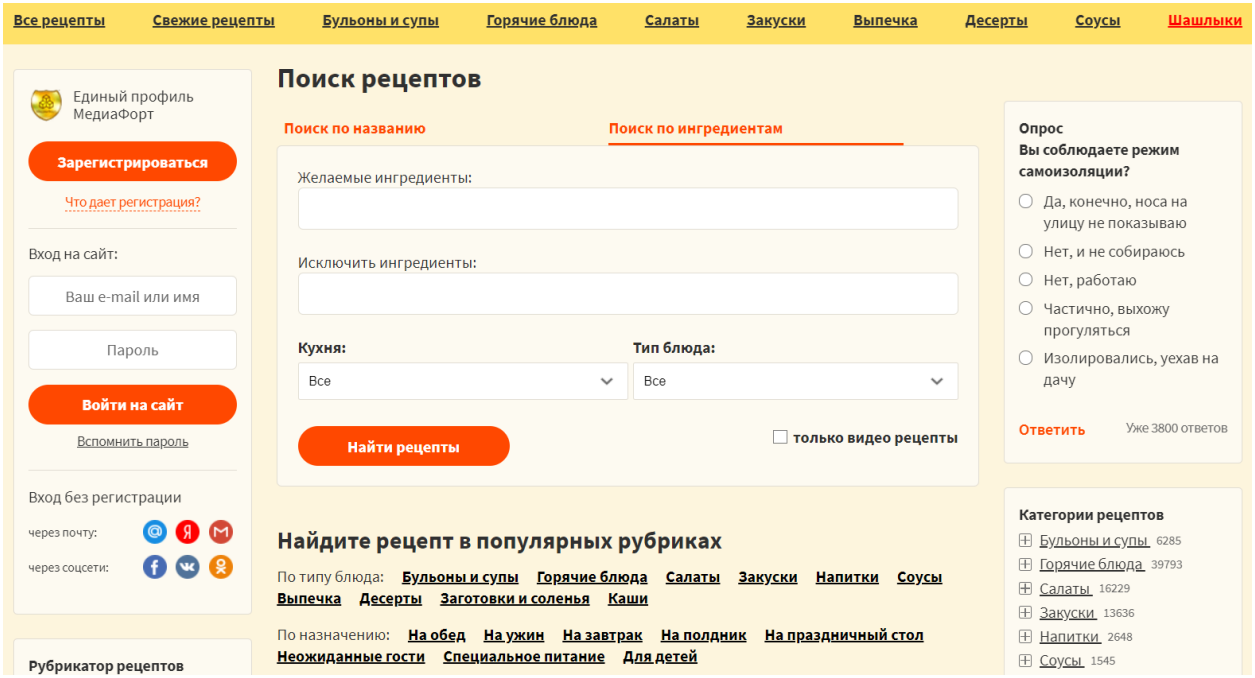


Рисунок 1.3 - Інтерфейс веб-застосування Povarenok.ru

1.2.3 Аналіз веб-застосування Supercook.com

Supercook.com (Рисунок 1.4 -) надає наступну функціональність:

– пошук рецептів за інгредієнтами, назвою, дієтами, країною походження та типом страви;

– завантаження мобільного застосування;

– оцінювання рецептів;

– збереження продуктів у список;

– перетворення рецептів у вигляд, придатний для друкування.

Перелік переваг:

– можливість оцінити рецепти та подивитися їх рейтинг;

– наявність фільтрації за визначеними типами страв;

– можливість одразу подивитися список зайвих інгредієнтів;

– наявність мобільного застосування;

– компактність списку інгредієнтів;

– наявність збереження списку продуктів;

– можливість перетворення рецептів у вигляд, придатний для друкування.

Перелік недоліків:

– нема підкатегорій продуктів;

– нема можливості додати власні рецепти;

– заборонити продукти можливо лише після пошуку, шляхом виключення зі списку зайвих інгредієнтів;

– відсутність пояснень чи зображень до інгредієнтів;

– нема можливості зберегти критерії пошуку.

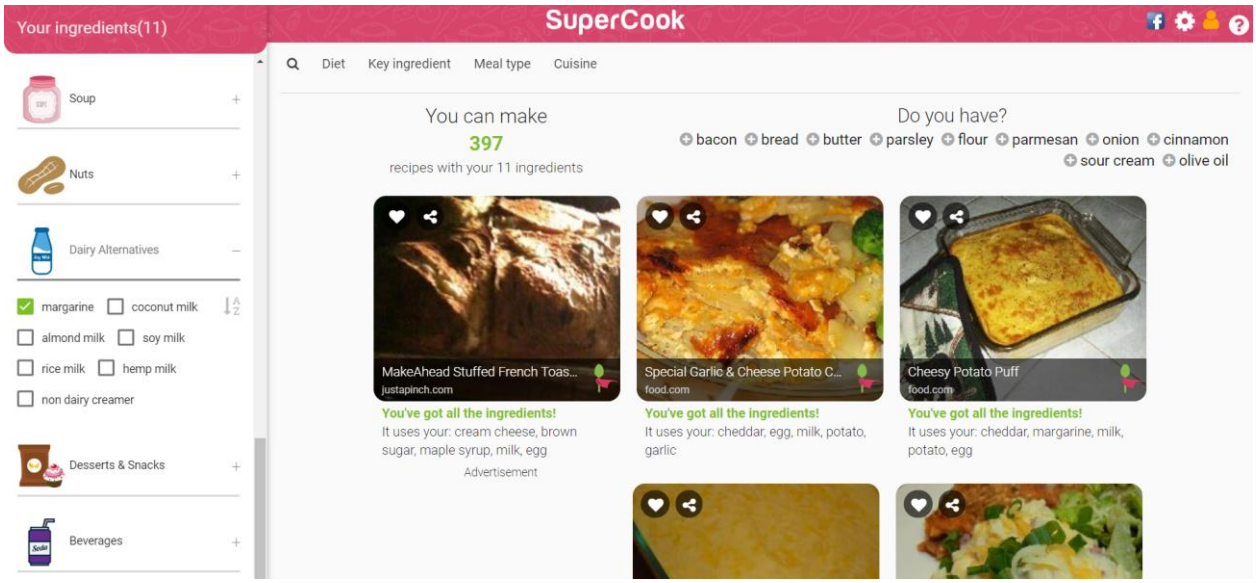


Рисунок 1.4 - Інтерфейс веб-застосування Supercook.com

1.3 Постановка задачі

1.3.1 Призначення розробки

Дане програмне забезпечення призначене для людей, які при виборі кулінарних рецептів зважають на різноманітні специфічні критерії, а також для кухарів, які бажають поділитися рецептами своїх страв з відповідною цільовою аудиторією.

1.3.2 Цілі та задачі розробки

Метою розробки є полегшення процесу пошуку кулінарних рецептів серед наявних за критеріями, обумовленими певними вподобаннями, релігійними поглядами, моральними принципами, медичними показаннями і протипоказаннями та наявністю ресурсів, а також спрощення представлення рецептів у вигляді, придатному для пошуку за вказаними критеріями, для передачі рецептів цільовій аудиторії.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– реєстрація та авторизація користувачів;

– перегляд, створення, редагування та видалення рецептів;

– створення та оновлення списку продуктів та категорій;

– пошук рецептів за назвою, ключовими словами, інгредієнтами, тривалістю приготування, калорійністю та часом;

– зберігання, видалення та використання фільтрів для пошуку рецептів;

– перегляд продуктів за категоріями та детального опису конкретного продукту.

Висновок до розділу

У цьому розділі були описані основна предметна область, призначення та мета розробки. Було висвітлено чому, дане програмне забезпечення має

вигляд саме веб-застосування, на основі порівняння цієї технології з застосуваннями під конкретні платформи.

Після аналізу різноманітних способів представлення рецептів онлайн, був обраний найбільш прийнятний з точки того, хто створює рецепт, хто його використовує та розробника програмного забезпечення.

Були описані основні сценарії роботи з сайтами та веб-застосуваннями, орієнтованими на кулінарні рецепти. Було проаналізовано наявні аналоги та виявлені основні проблеми, що мають бути вирішенні в даній розробці

Виділено три основні ролі користувачів та розподіллено функції між ними. Були описані усі функціональні та не функціональні вимоги до веб-застосування та проаналізовано їх відношення до основних сценаріїв роботи.

2 ІНФОРМАЦІЙНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Для надання можливості пошуку рецептів за інгредієнтами, перш за все, необхідно сформувати базу продуктів та розподілити їх на рівні категорій, таким чином, щоб множина усіх можливих кінцевих продуктів певного продукту, повністю входила в множину значень усіх категорій цього продукту. Один продукт може мати декілька категорій та множину продуктів, які є його видами.

Окрім цього, для основних продуктів необхідно приблизно вказати наступні дані:

– калорійність (в ккал на 100 г);

– вміст жирів (в г на 100 г);

– вміст білків (в г на 100 г);

– вміст вуглеводів (в г на 100 г);

– вміст води (в г на 100 г);

– вміст золи (в г на 100 г);

– вміст цукру (в г на 100 г);

– вміст крохмалю (в г на 100 г);

– вміст холестерину (в г на 100 г);

– вміст клітковини (в г на 100 г);

– вміст транс жирів (в г на 100 г);

– загальний опис продукту.

Оскільки, під час створення списку продуктів та визначення їх категорій та підвидів, можливо заплутатися та створити цикли, рекомендується наступна послідовність створення бази продуктів: спочатку формується якомога повніший список кінцевих продуктів; далі найбільш схожі один на одного продукти об’єднуються у категорії з якомога меншою множиною продуктів; потім формуються категорії над отриманими категоріями та

кінцевими продуктами за тим же принципом; наприкінці, створюються вже популярні категорії (тобто, об'єднання виникає вже не через виявлення у набору продуктів певної ознаки, а через те, що категорія є часто вживаною і вона має бути наявною в системі), які можуть мати велику множину продуктів; якщо продукти всередині категорії можливо поділити на групи за певним принципом, то краще поділити категорію на декілька підкатегорій, перемістити відповідні продукти всередину них, а в категорії залишити лише нові підкатегорії та залишені продукти.

2.2 Опис структури бази даних

В системі існують дві бази даних, одна призначена лише для ідентифікації користувачів, а інша для збереження основних даних веб-застосування.

Структура другої бази (діаграма відношень між її сутностями) зображена в додатку «Структура бази даних». Перша база даних створена для стороннього програмного модулю, що забезпечує реєстрацію та ідентифікацію користувачів, тож її структура не є важливою для роботи основного програмного забезпечення. Єдина інформація, яка є важливою та поєднує обидві бази даних, це ідентифікатор користувача, за яким ми можемо знайти необхідний профіль в основній базі даних (поле UserId в таблиці Profile).

Головна база даних складається з 10 таблиць:

а) чотири основні сутності:

- Filter – призначена для зберігання фільтрів (критеріїв для пошуку рецептів);
- Recipe – призначена для зберігання опису рецептів;
- Product – призначена для зберігання опису продуктів;
- Profile – призначена для зберігання додаткових даних про користувача;

б) сутність для зберігання усіх зображень – Image;

в) п'ять зв'язних сутностей:

– FilterPart – зберігає інформацію про те, які продукти та тегі належать фільтру та з якими параметрами;

– RecipeTag – зберігає інформацію про те, які тегі належать рецепту;

– RecipeStep – зберігає інформацію про те, які кроки приготування належать рецепту;

– Ingredient – зберігає інформацію про те, які інгредієнти належать рецепту та з якими параметрами;

– ProductCategory – зберігає інформацію про те, які продукти належать до якої категорії.

2.3 Змістовна постановка задачі

Однією з головних задач даного веб-застосування є пошук рецептів за різними критеріями. Пошук за назвою, тривалістю приготування, калорійністю чи ключовими словами є досить простим та не потребує особливих алгоритмів. Вбудовані в мову програмування алгоритми пошуку, зазвичай, є достатніми для цих задач. Можливе прискорення їх виконання, внаслідок індексування визначених даних.

Проте проблема збереження ієрархії продуктів та пошуку не лише за конкретними інгредієнтами, але й, враховуючи їх категорії чи кінцеві продукти, є не такою тривіальною задачею, та вона буде проаналізована нижче.

Зауважимо, що кожен рецепт містить набір інгредієнтів, в якому зазначаються необхідні та опціональні продукти з точки зору приготування визначеної страви. Можуть бути вказані, як кінцеві продукти (наприклад, сицилійський апельсин), так і категорії продуктів (наприклад, цитрусові).

Проаналізуємо способи пошуку за інгредієнтами.

Набір продуктів, може обмежуватися трьома способами:

– наявні продукти (набір продуктів, що доступний користувачеві, і за рамки якого він не може вийти під час приготування страви);

– обов’язкові продукти (набір продуктів, які користувач обов’язково хоче використати під час приготування їжі);

– заборонені продукти (набір продуктів, які через певні обставини чи вподобання, не має бути присутнім в страві).

Користувачу доступні лише два обмеження одночасно: наявні/обов’язкові та заборонені/обов’язкові продукти.

За замовчуванням, пошук за інгредієнтами здійснюється з урахуванням категорій та підвидів продуктів (правила, за якими перевіряються залежні продукти будуть проаналізовані далі). Проте, для користувачів, які бажають здійснити пошук лише за визначеними продуктами, без перевірки їх категорій та підвидів, існує ще один режим пошуку, який теж має два варіанти: пошук за обов’язковими/забороненими та за обов’язковими/наявними продуктами.

2.4 Обґрунтування методу розв’язання

Якщо користувач бажає, щоб в рецепті обов’язково був певний продукт, то очевидним є те, що рецепти, які використовують підвиди цього продукту є прийнятними. З іншого боку, якщо автор рецепту не вказав кінцевий продукт для інгредієнта, а зазначив цілу категорію, це означає, що він вважає, що будь-який підвид цієї категорії може бути використаний в рецепті. Тож, під час пошуку за обов’язковими продуктами, рецепти в яких вказані категорії цього продукту теж є прийнятними.

Якщо користувач, забороняє певний продукт, то очевидним є те, що він бажає заборонити й усі його підвиди (тобто, рецепти, що містять підвиди заборонених продуктів теж не попадають в вибірку). Проте, те, що певний продукт є забороненим не означає, що його категорія теж є забороненою, адже можуть бути використані інші її підвиди. Тож рецепти, що містять лише категорії заборонених продуктів, не повинні зникати з вибірки. Під час фільтрування за забороненими продуктами, потрібно перевіряти

лише обов’язкові інгредієнти, адже, якщо навіть серед опціональних інгредієнтів є заборонені продукти, ми завжди можемо їх просто не використовувати.

Як було зазначено раніше, те що автор вказує категорію продуктів, позначає що будь-який підвид є прийнятним. Отже, якщо користувач, орієнтується лише на наявні продукти, то очевидним є те, що він може використати їх в рецептах, де вказані їх категорії. Проте, те, що певний продукт є наявним (наприклад, твердий сир), не означає, що є наявним будь-який з його підвидів (наприклад, сир Рокфор), а якщо автор зазначив саме цей підвид в рецепті, то, скоріш за все, саме його використання є важливим для правильного приготування страви. Тож, якщо в рецепті зазначено продукт, що є категорією одного з наявних продуктів, проте його та його категорії нема в цьому списку, то такий рецепт не є прийнятним. Якщо користувач не має певних продуктів, зазначених в рецепті (вони не вказані в його списку наявних продуктів), але вони є опціональними інгредієнтами, то очевидним є те, що він може просто їх не використовувати й рецепт все ще є прийнятним. Тобто, як і у випадку з забороненими продуктами, під час пошуку за наявними продуктами, потрібно перевіряти лише обов’язкові інгредієнти.

Під час пошуку одночасно за обов’язковими та за наявними продуктами, логічним є те, що список наявних продуктів, має доповнюватися списком обов’язкових продуктів.

Якщо під час пошуку одночасно за обов’язковими та забороненими продуктами, виникає спірна ситуація, коли певний продукт або його підвид є обов’язковим, але одна з його категорій є забороненою, то вважається, що продукт є забороненим, адже заборона, зазвичай, є більш важливим обмеженням (наприклад, якщо у користувача є алергія на певні категорії продуктів). Ця ситуація я виключною, і може бути пов’язана з тим, що користувач не знав про приналежність цього продукту до забороненої категорії під час вибору. В такій ситуації пошук все ще може надати якісь рецепти, тільки якщо вони містять лише незаборонені категорії обов’язкового

продукту. Коли категорія певного продукту є обов'язковою, а сам продукт чи інша його категорія є забороненою, то продукт має теж стати забороненим, адже таким чином користувач може просто обмежувати продукти певної категорії (тож, ця ситуація вже не є виключною).

Оскільки, продукти можуть відноситися до декількох категорій та мати багато підвидів (тобто продукт може бути як кінцевим так і категорією), множину продуктів можна представити у вигляді незваженого орієнтованого ациклічного графа [1)], де вершинами є продукти, а дугами – зв'язки категорія-продукт (дуга веде від категорії до її продукту). Приклад множини продуктів у такому вигляді зображено нижче (Рисунок 2.1 -). Тоді ми можемо представити продукти у базі даних, як, зазвичай, описуються орієнтовані графи в програмі: список вершин з ідентифікаторами та список усіх дуг, кожен елемент якого складається з двох ідентифікаторів, які позначають початкову та кінцеву вершину дуги. В базі даних це виглядає як таблиця Product, в якій є ідентифікатор Id (список вершин), та таблиця ProductCategory (список дуг), в якій вказані початкова (CategoryId) та кінцева (ProductId) вершини. Для зручності переходу між вершинами, на рівні сервісів кожен продукт містить два списки (категорії та підвиди), що посилаються на інші продукти.

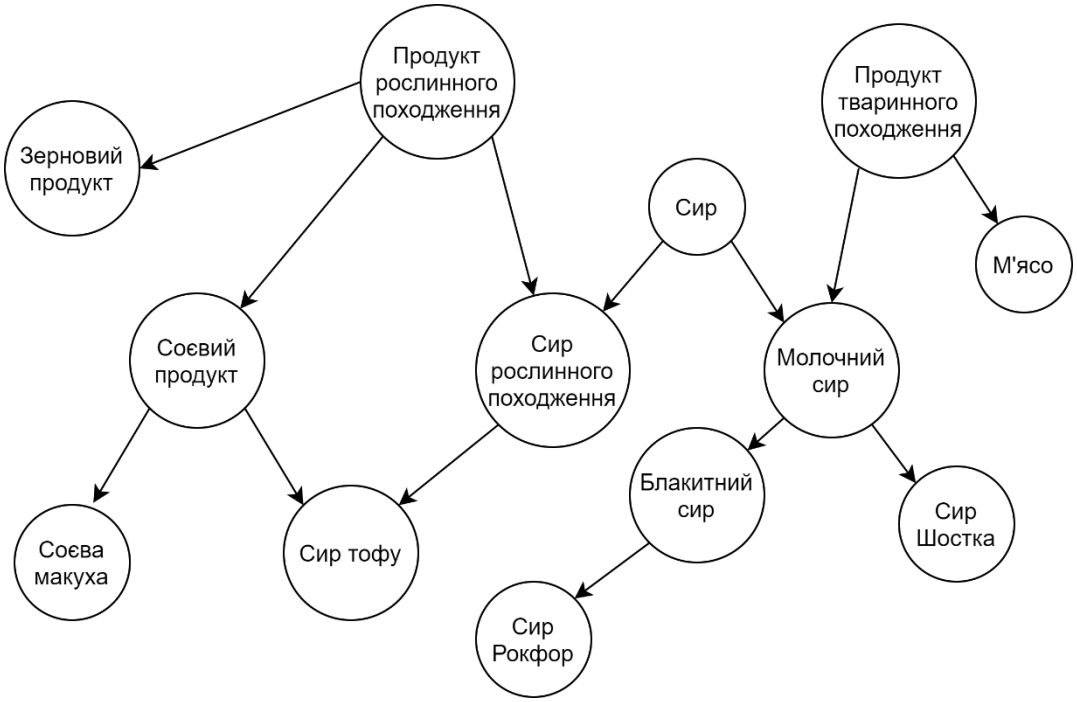


Рисунок 2.1 - Приклад множини продуктів у вигляді графу

2.5 Опис методів розв'язання

2.5.1 Алгоритм фільтрації рецептів за обов'язковими та забороненими продуктами з категоріями

Вхідні значення:

required – список обов'язкових продуктів (їх ідентифікатори);

forbidden – список заборонених продуктів (їх ідентифікатори);

recipes – список рецептів з інгредієнтами.

Крок 1. Розширення списку заборонених продуктів.

1) Витягнення з бази даних списку початкових заборонених продуктів разом з їх нащадками (за допомогою списку forbidden) – forbiddenProducts.

2) Для кожного продукту в forbiddenProducts.

2.1) Для кожної підкатегорії (child) поточного продукту:

2.1.1) ЯКЩО поточна підкатегорія наявна в списку forbidden, ТО виконується перехід до наступної підкатегорії (пункт 2.1.4).

2.1.2) Додання поточної категорії до списку заборонених продуктів forbidden.

2.1.3) Повторення пункту 2.1 для усіх нащадків (підкатегорій) поточної підкатегорії.

2.1.4) Перехід до наступної підкатегорії.

3) Передача розширеного списку заборонених продуктів – forbidden.

Крок 2. Формування розширеного списку обов'язкових продуктів з альтернативами.

1) Витягнення з бази даних списку обов'язкових продуктів разом з їх нащадками та пращурами (за допомогою списку required) – requiredProducts.

2) Створення порожнього списку обов’язкових продуктів (елементами якого будуть списки альтернативних варіантів) – newRequired.

3) Для кожного продукту (product) в requiredProducts.

3.1) Створення порожнього списку альтернативних продуктів (variants) та додавання туди поточного продукту.

3.2) Для кожної підкатегорії поточного продукту.

3.2.1) ЯКЩО підкатегорія наявна в списку forbidden чи variants, ТО виконується перехід до наступної підкатегорії (пункт 3.2.5).

3.2.2) ЯКЩО підкатегорія наявна в списку required, ТО видаляється поточний продукт (product) зі списку required та виконується перехід до наступного продукту (пункт 3.5).

3.2.3) Додавання поточної підкатегорії до списку variants.

3.2.4) Повторення пункту 3.2 для усіх нащадків (підкатегорій) поточної підкатегорії.

3.2.5) Перехід до наступної підкатегорії.

3.3) Для кожної категорії поточного продукту.

3.3.1) ЯКЩО категорія наявна в списку forbidden чи variants, ТО виконується перехід до наступної категорії (пункт 3.3.4).

3.3.2) Додання поточної категорії до списку variants.

3.3.3) Повторення пункту 3.3 для категорій поточної категорії.

3.3.4) Перехід до наступної категорії.

3.4) Додання списку альтернативних варіантів variants (як елементу списку) до списку newRequired.

3.5) Перехід до наступного продукту.

4) Передача списку обов’язкових продуктів з альтернативами requiredProducts.

Крок 3. Фільтрування рецептів за обраними продуктами.

1) Для кожного рецепту в recipes.

1.1) Витягнення списку обов’язкових продуктів для рецепта – requiredIngredients.

1.2) ЯКЩО requiredIngredients містить хоча б один продукт зі списку forbidden, ТО видаляється поточний рецепт з recipes та виконується перехід до наступного (пункт 1.5).

1.3) Витягнення списку усіх інгредієнтів для поточного рецепта – ingredients.

1.4) Для кожного списку альтернатив (variants) в списку newRequired.

1.4.1) ЯКЩО ingredients не містить жодного продукту зі списку variants, ТО видаляється поточний рецепт з recipes та виконується перехід до наступного (пункт 1.5).

1.4.2) Перехід до наступного списку.

1.5) Перехід до наступного рецепта.

2) Повернення відфільтрованого списку рецептів.

2.5.2 Алгоритм фільтрації рецептів за наявними та обов’язковими продуктами з категоріями

Вхідні значення:

required – список обов’язкових продуктів (їх ідентифікатори);

available – список наявних продуктів (їх ідентифікатори);

recipes – список рецептів з інгредієнтами.

Крок 1. Розширення списку наявних продуктів.

1) Витягнення з бази даних списку початкових наявних продуктів разом з їх нащадками (за допомогою об’єднаних в один списків available та required) – availableProducts.

2) Для кожного продукту в availableProducts.

2.1) Для кожної категорії поточного продукту.

2.1.1) ЯКЩО поточна категорія наявна в списку available, ТО виконується перехід до наступної підкатегорії (пункт 2.1.4).

2.1.2) Додання поточної категорії до списку наявних продуктів available.

2.1.3) Повторення пункту 2.1 для усіх категорій поточної категорії.

2.1.4) Перехід до наступної категорії.

3) Передача розширеного списку наявних продуктів – available.

Крок 2. Формування розширеного списку обов’язкових продуктів з альтернативами.

Аналогічно до попереднього алгоритму, якщо вважати, що список заборонених продуктів порожній.

Крок 3. Фільтрування рецептів за обраними продуктами.

1) Для кожного рецепту в recipes.

1.1) Витягнення списку обов’язкових продуктів для рецепта – requiredIngredients.

1.2) ЯКЩО requiredIngredients містить хоча б один продукт, якого нема в списку available, ТО видаляється поточний рецепт з recipes та виконується перехід до наступного (пункт 1.5).

1.3) Витягнення списку усіх інгредієнтів для поточного рецепта – ingredients.

1.4) Для кожного списку альтернатив (variants) в списку newRequired.

1.4.1) ЯКЩО ingredients не містить жодного продукту зі списку variants, ТО видаляється поточний рецепт з recipes та виконується перехід до наступного (пункт 1.5.).

1.4.2) Перехід до наступного списку.

1.5) Перехід до наступного рецепта.

2) Повернення відфільтрованого списку рецептів.

2.5.3 Алгоритми фільтрації рецептів за продуктами без перевірки категорій

Алгоритми пошуку за продуктами без урахування категорій є аналогічними до попередніх, але без перших двох кроків. Третій крок (фільтрування рецептів за обраними продуктами) виконується на основі початкових списків.

Висновок до розділу

В даному розділі було проаналізовано та обмежено множину необхідних вхідних даних та визначено шляхи її створення, а також виявлено необхідне математичне забезпечення.

Було виявлено основні сутності, які мають зберігатися, знайдені залежні дані, та виділення окремих даних таким чином, щоб мати можливість гарантувати цілісність та релевантність даних. Була розроблена структурна схема бази даних, що зберігає основну інформацію необхідну для коректної роботи веб-застосування.

Було виявлено основні проблеми розробки з математичної точки зору. Розроблено та деталізовано алгоритми пошуку за обов’язковими й забороненими та обов’язковими й наявними продуктами, орієнтуючись лише на конкретні продукти, та аналізуючи їх категорії та підвиди. Було обґрунтовано обрані методи розв’язання задач.

3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання та аналіз програмного забезпечення

Для моделювання та аналізу програмного забезпечення, будуть розглянуті основні процеси використання даного веб-застосування користувачем за допомогою BPMN діаграм.

Послідовний опис реєстрації користувача (створення облікового запису) (Рисунок 3.1 -):

– користувач переходить на сторінку реєстрації;

– користувач заповнює форму основними даними (логін – ідентифікатор користувача, пароль, ім’я, електронну пошту);

– програма перевіряє наявність усіх обов’язкових даних та їх формату:

– якщо є помилки, то виводиться повідомлення про помилку, та користувач знову заповнює некоректні поля;

– інакше виконується підтвердження паролю;

– якщо паролі збігаються, то стає доступною кнопка реєстрації, інакше користувач знову підтверджує пароль;

– користувач натискає кнопку реєстрації та чекає на відповідь від сервера;

– сервер перевіряє унікальність логіну (виконує пошук у базі даних);

– якщо логін існує, то повертається відповідне повідомлення, та користувач вводить нові дані;

– інакше сервер повторно перевіряє наявність та формат даних;

– якщо є помилки, то повертається відповідне повідомлення, та користувач вводить нові дані;

– інакше сервер створює обліковий запис та відправляє його до бази даних; повертається повідомлення про успішну реєстрацію.

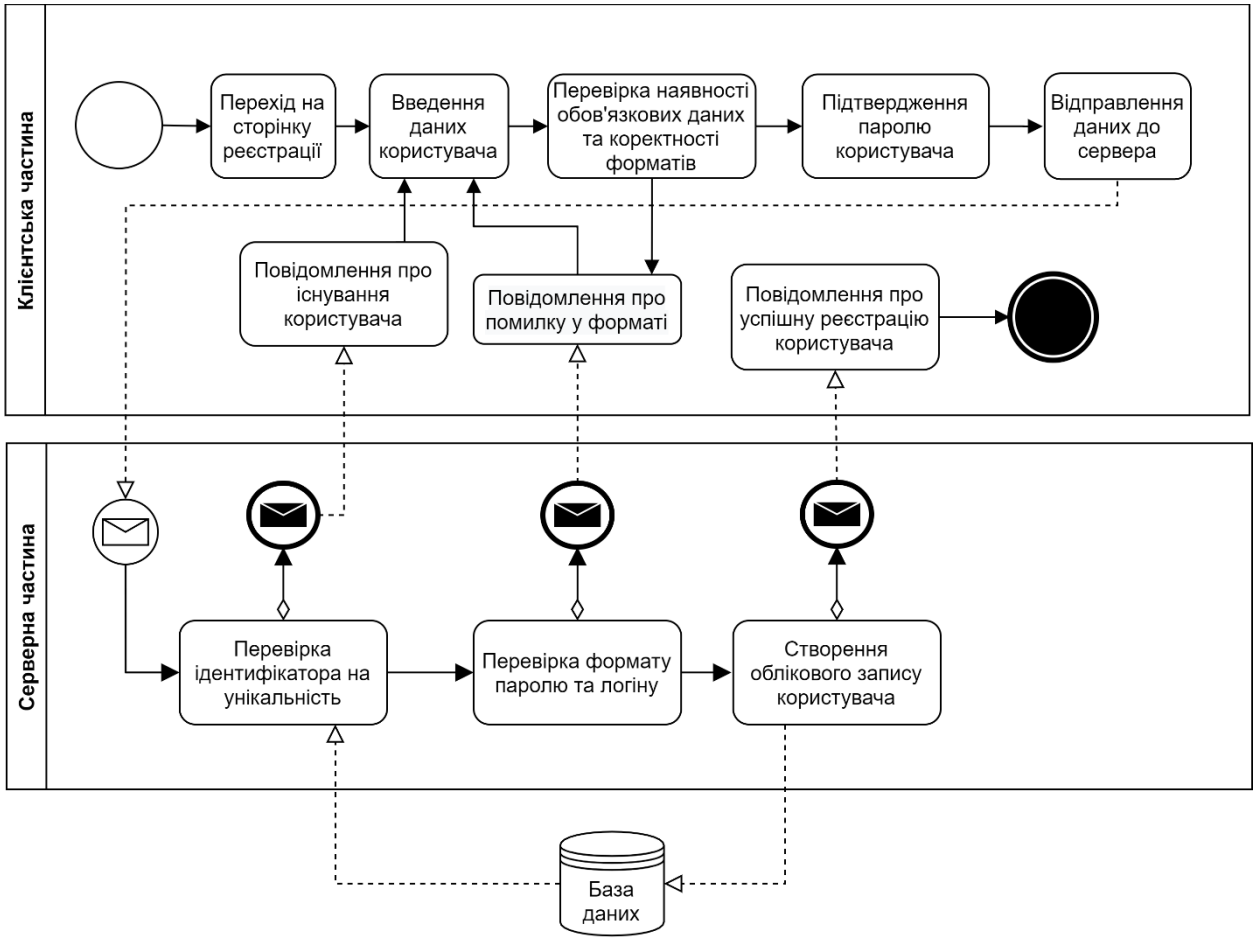


Рисунок 3.1 - Схема бізнес-процесу реєстрації користувача

Послідовний опис входу користувача в обліковий запис (Рисунок 3.2 -):

– користувач переходить на сторінку входу в обліковий запис;

– користувач заповнює форму входу (логін та пароль);

– програма перевіряє наявність та формат даних;

– якщо дані некоректні, то підсвічуються помилки та користувач знову вводить дані для входу;

– інакше, стає доступною кнопка входу;

– користувач натискає кнопку входу та очікує відповіді від сервера;

– сервер отримує логін та пароль, та виконує їх перевірку (перевіряє існування користувача й відповідність даних);

– якщо знайдені помилки, то повертається відповідне повідомлення про помилку;

– інакше сервер дістає дані користувача з бази даних;

– сервер створює тимчасовий токен;

– сервер повертає токен та загальні дані про користувача (ім’я, логін, роль);

– клієнтська частина отримує дані та записує їх у локальне сховище браузера;

– надається доступ до дій, що дозволені ролі даного користувача;

– виводиться повідомлення про успішний вхід в обліковий запис.

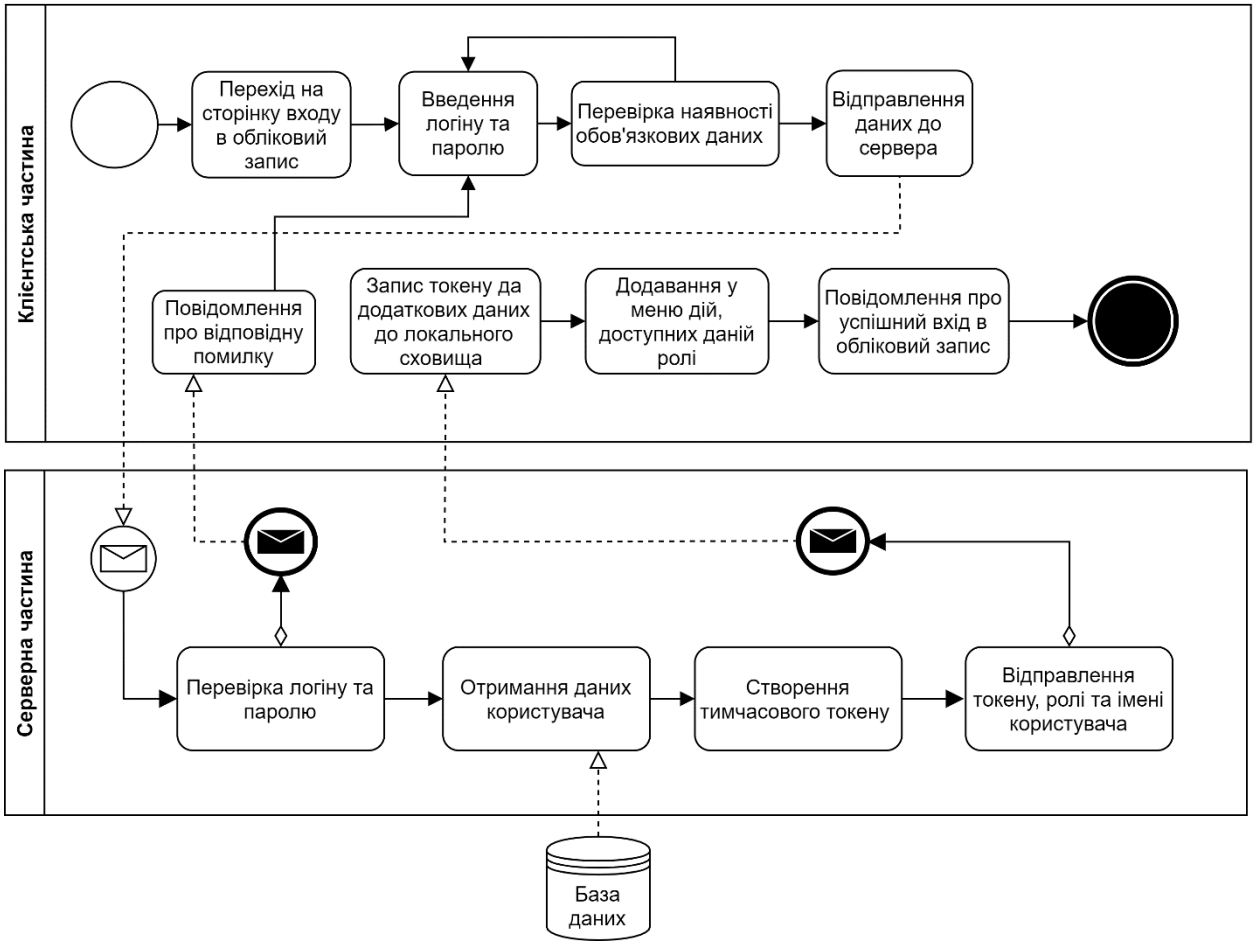


Рисунок 3.2 - Схема бізнес-процесу входу в обліковий запис

Послідовний опис пошуку рецептів (Рисунок 3.3 -):

– користувач переходить на сторінку пошуку рецептів;

– робиться запит на отримання з сервера списку інгредієнтів, наявних тегів та фільтрів, що доступні користувачеві;

– сервер дістає відповідні дані з бази даних та повертає їх;

– користувач заповнює критерії пошуку у секціях: головне, інгредієнти, фільтри, тегі;

– критерії пошуку з’являються у секції прев’ю;

– користувач перевіряє дані (за необхідності змінює їх) та натискає кнопку пошуку за критеріями;

– сервер отримує запит на пошук рецептів у вигляді фільтру;

– сервер дістає з бази даних усі рецепти, що відповідають критеріям назви, калорійності та часу;

– якщо у фільтрі вказані тегі, то виконується фільтрація рецептів за тегами;

– якщо у фільтрі вказані інгредієнти, то сервер дістає з бази даних усі вказані та, пов’язані з ними, продукти, та фільтрує рецепти за алгоритмом;

– сервер повертає список відфільтрованих рецептів;

– клієнтська частина отримує список рецептів та показує їх на екрані, розподіляючи на сторінки.

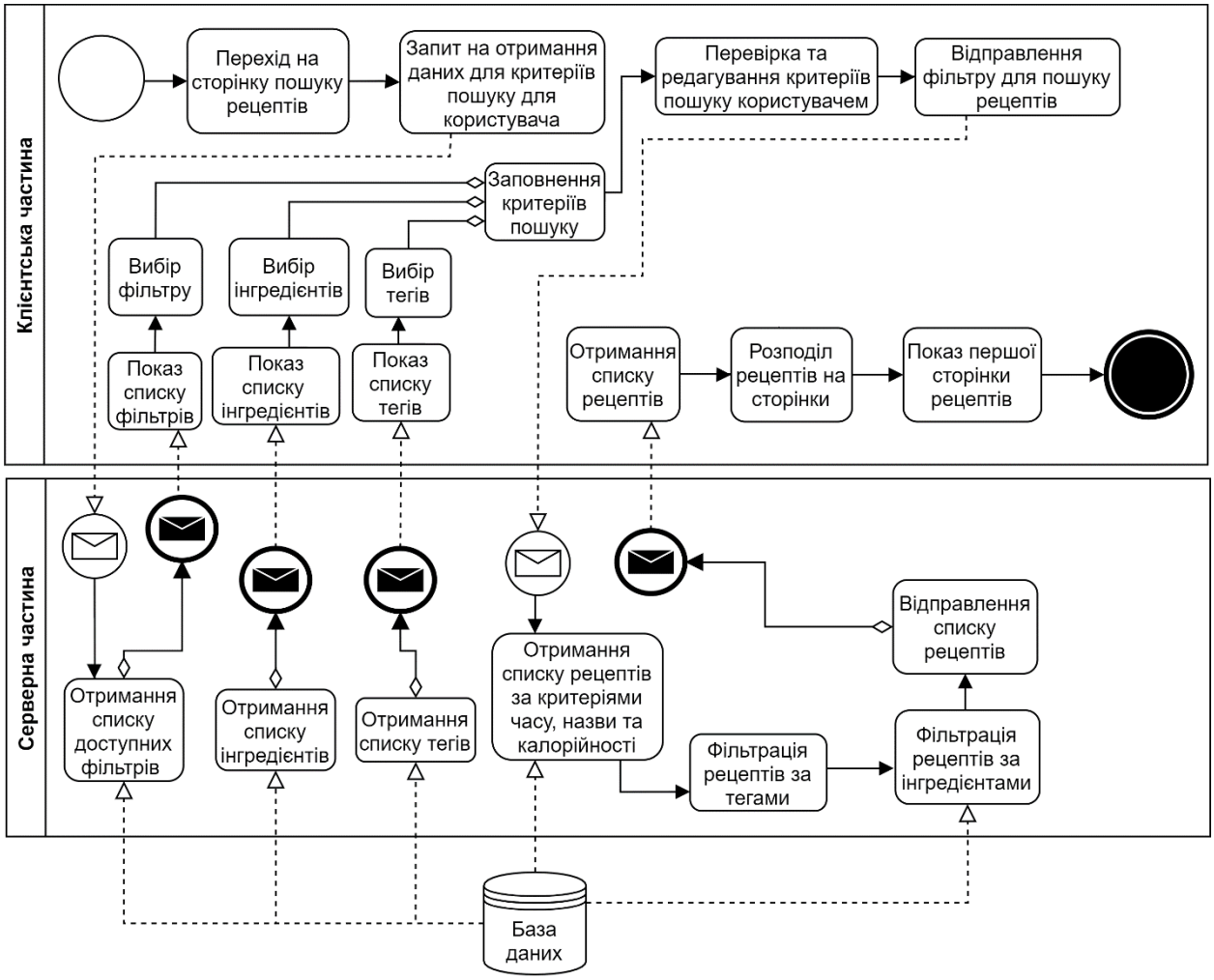


Рисунок 3.3 - Схема бізнес-процесу пошуку рецептів

Послідовний опис створення рецепту (Рисунок 3.4 -):

– користувач входить в обліковий запис;

– користувач переходить до сторінки створення рецептів;

– користувач заповнює основну інформацію про рецепт (назва, опис, калорійність, фотографія, тривалість приготування, кількість порцій);

– користувач додає інгредієнти: програма робить запит до сервера; сервер отримує список продуктів з бази даних та відсилає його до клієнтської частини; користувач обирає інгредієнти зі списку та описує їх);

– користувач додає кроки приготування (їх назви, опис та фотографії);

– програма перевіряє наявність обов’язкової інформації та формати даних;

– якщо знайдені помилки, то підсвічуються відповідні поля, та користувач вводить нові дані;

– інакше стає доступною кнопка створення рецепту;

– користувач натискає кнопку створення рецепту;

– програма відправляє запит на збереження рецепту до сервера;

– сервер отримує рецепт та додає ідентифікатор поточного користувача у якості автора рецепту;

– сервер відправляє рецепт до бази даних та отримує його ідентифікатор;

– сервер повертає ідентифікатор нового рецепта;

– клієнтська частина виводить повідомлення про успішне створення.

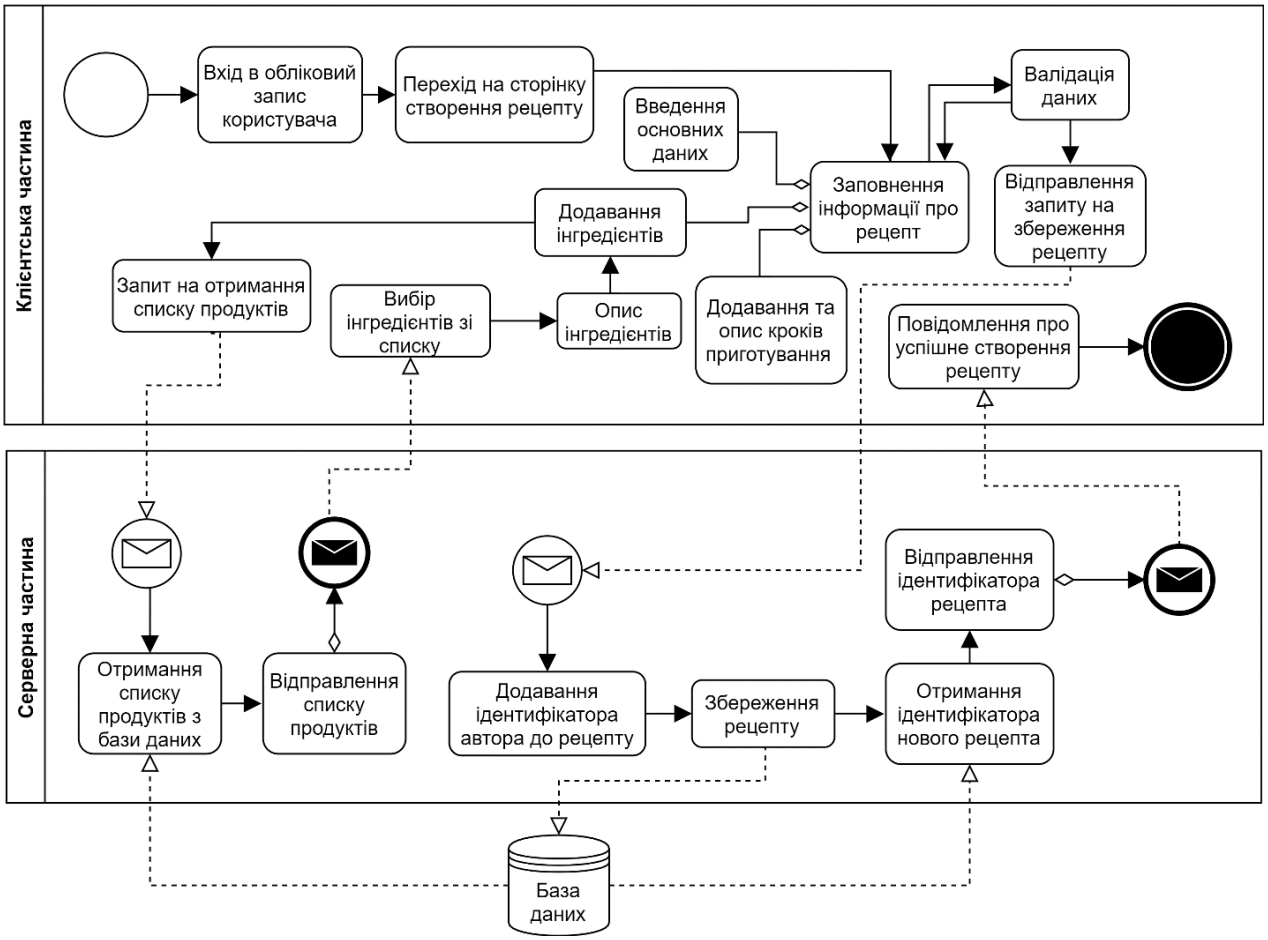


Рисунок 3.4 - Схема бізнес-процесу пошуку рецептів

3.2 Архітектура програмного забезпечення

Дане програмне забезпечення є веб-застосування, тож, воно складається з двох основних частин – серверної та клієнтської. Далі детально описано архітектуру кожної з цих частин.

3.2.1 Архітектура серверної частини

Для побудови серверу була обрана модульна платформа .NET Core версії 3.0 та мова програмування C# версії 8.

Серверна частина має трирівневу архітектуру та поділяється на наступні рівні:

– рівень доступу до бази даних (проєкти CulinaryAssistant.DAL.Abstracts та ...DAL.Infrastructure);

– рівень бізнес-логіки (проєкти CulinaryAssistant.BLL.Abstracts та ...BLL.Infrastructure);

– рівень прикладного програмного інтерфейсу (проєкт CulinaryAssistant.WebAPI).

Рівні доступу до бази даних та бізнес-логіки були розділені ще на два проєкти, щоб відділити інтерфейси та сутності від їх реалізації, задля того, щоб зменшити зв’язність рівнів програмного забезпечення та надати можливість проєктам використовувати інтерфейси іншого проєкту без можливості перегляду його конкретної реалізації (таким чином, якщо потрібно буде змінити реалізацію певного рівня на принципово нову, це можна буде зробити, просто замінивши посилання на новий проєкт в місці зв’язки проєктів, без необхідності змінювати інші). Рівень прикладного інтерфейсу сильно залежить від своєї реалізації та використовує сутності проєкту бізнес-логіки, тож, він не може бути використаний окремо, та розбиття його на два окремих проєкти є недоцільним.

Використання одних класів всередині інших відбувається за шаблоном «Впровадження залежностей», способом впровадження залежностей через

конструктор класу. Усі залежності та час їх життя визначається на верхньому рівні за допомогою вбудованого в програмне забезпечення ASP.NET Core IoC-контейнера.

3.2.1.1 Рівень доступу до бази даних

Для роботи з базою даних був обраний підхід Code-First технології Entity Framework Core. Цей підхід використовує класи (моделі), написані мовою C#, для формування з них таблиць баз даних. Оскільки для даної програми необхідна відносно невелика база даних (десять таблиць), то використання цього підходу є доречним.

Проєкт абстракцій містить сутності бази даних (Recipe, Profile, Filter, FilterPart, RecipeStep, RecipeTag, Product, Image, ProductCategory та Ingredient) та інтерфейси репозиторіїв (Рисунок 3.5 -).

Щоб не зазначати поле ідентифікатора Id в кожній сутності, був створений окремий абстрактний клас Entity, в якому знаходиться ідентифікатор типу long, та від якого наслідуються усі інші сутності. Детальна схема сутностей наведена в додатку «Схема бази даних».

Для доступу до даних в Entity Framework є особливий клас Context, який реалізую шаблон проєктування Unit of Work, проте, оскільки інші проєкти не мають знати про деталі реалізації доступу до бази даних, то цей клас не може використовуватися іншими проєктами. Тож, для впровадження доступу до даних використовуються класи репозиторіїв, що реалізують шаблон проєктування Репозиторій.

Базовими функціями, необхідними для роботи з базою даних є: додання нового об’єкту, додання множини нових об’єктів, редагування об’єкту, витягнення об’єкту за його ідентифікатором, витягнення усіх об’єктів, пошук об’єктів за умовою, перевірка наявності об’єкту за його ідентифікатором. Оскільки, всі ці функції повторюються для кожної сутності, їх опис винесено у базовий узагальнений інтерфейс IRepository<T>, який може використовувати лише об’єкти класів, що наслідуються від Entity.

Поставлені задачі вимагають різноманітної фільтрації великого об’єму даних, а проста фільтрація значно швидше відпрацьовує на стороні бази даних. Для цього можна використовувати метод пошуку за умовою, проте, в певних випадках нам потрібно ще витягнути деякі пов’язані з об’єктами дані, без необхідності робити додаткові запроси, що впливають на продуктивність. Тож, окрім стандартних методів, в інтерфейсах репозиторіїв потрібно вказати ще специфічні функції для кожної окремої сутності.

Деякі сутності є додатковими та створені лише для збереження множин даних, пов’язаних з основною сутністю. Тож, нема потреби надавати інтерфейси для роботи з ними, достатньо лише надати можливість за необхідністю підтягнути ці дані під час витягнення головних сутностей. Не були створені окремі репозиторії для сутностей RecipeStep, FilterPart, Ingredient та FilterPart.

Проєкт реалізації доступу до бази даних містить класи репозиторіїв, налаштувань таблиць, файли для міграції, контекст та клас з визначенням зв’язків між інтерфейсами та їх імплементаціями за замовчуванням (Рисунок 3.6 -).

Конфігурації (налаштування) для кожної сутності були винесені в окремі класи та файли для наочності та спрощення змін в конфігураціях окремих сутностей.

Щоб не було дублювання коду стандартні основні функції репозиторію були реалізовані в узагальненому класі GenericRepository<T>, а всі його публічні методи були створені віртуальними, щоб надати змогу перевизначати їх в репозиторіях-нащадках. Наприклад, для функції видалення, адже стандартна поведінка (коли при видаленні об’єкту, видаляються усі залежні об’єкти) є некоректною у випадку, коли непотрібно видаляти пов’язані дані або потрібно видалити лише частину з них. Більш того, це налаштування, робить неможливим видалення об’єктів з таблиць, що теоретично можуть утворювати цикли (наприклад, таблиці продуктів та категорій).

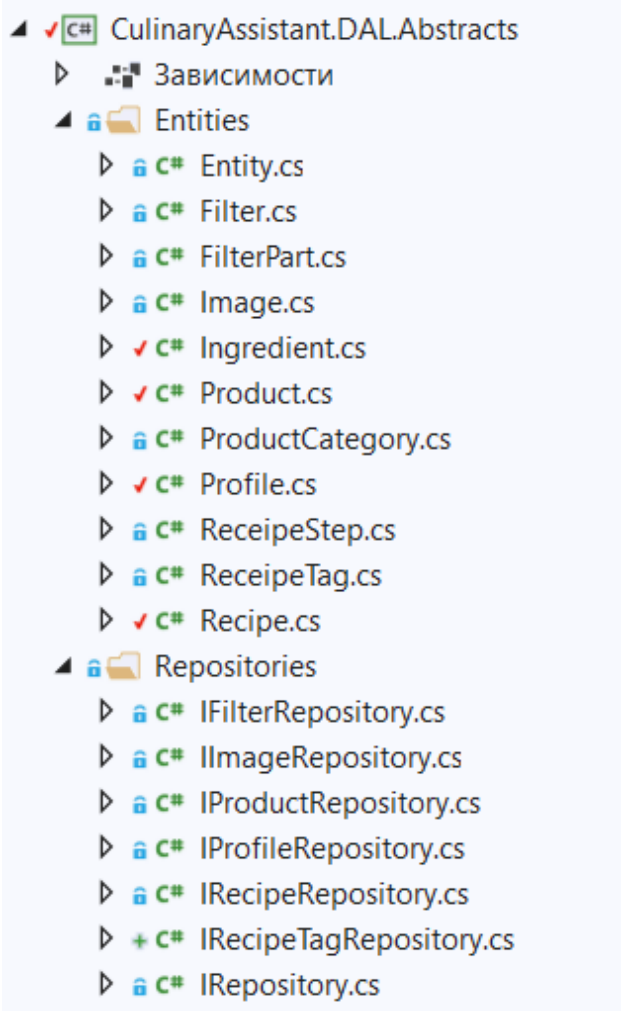


Рисунок 3.5 - Структура проєкту абстракцій доступу до бази даних

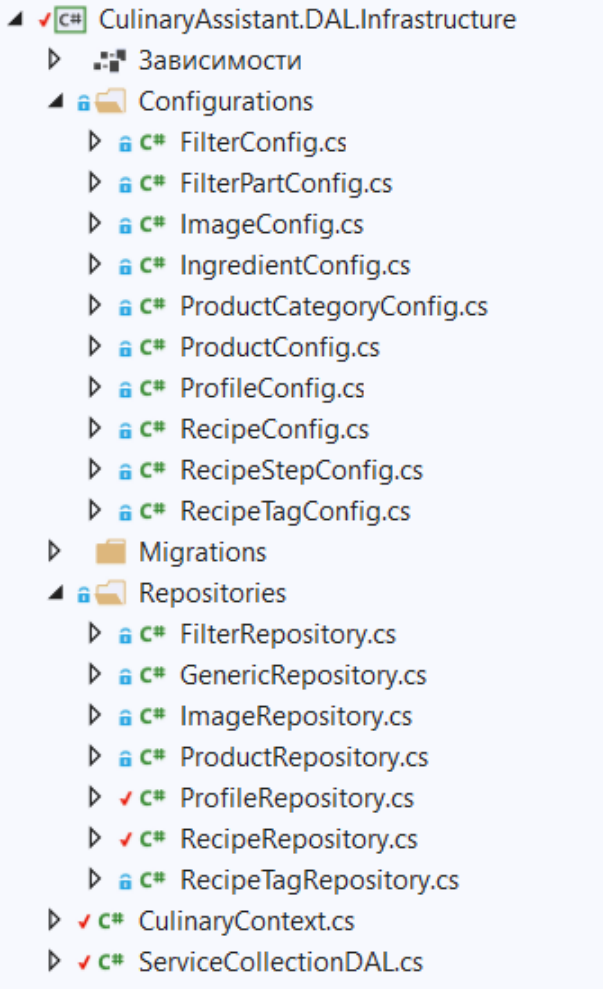


Рисунок 3.6 - Структура проєкту доступу до бази даних

3.2.1.2 Рівень бізнес-логіки

Проєкт абстракцій містить об’єкти для передачі даних на наступний рівень та інтерфейси сервісів.

Всі функції, що реалізовує дане програмне забезпечення можна умовно поділити на чотири основні сценарії (керування рецептами, керування продуктами, керування обліковими записами та фільтрація рецептів), кожне з яких має одну головну сутність (відповідно: рецепт, продукт, профіль та фільтр). Тож, були створені окремі сервіси для взаємодії з кожним з них. Оскільки, зображення займають багато пам’яті, то для можливості завантаження їх з сервера лише за необхідністю, був створений окремий сервіс роботи з зображеннями. Методи всіх сервісів описані на схемі структурній класів сервісів (Рисунок 3.7 -).

Основні сутності були розбиті на декілька класів, в залежності від набору даних, що має передаватися чи приходити під час різних процесів, а також були вилучені зайві навігаційні поля (Рисунок 3.8 -).

Проєкт реалізацій бізнес-логіки, окрім сервісів, містить класи-компаратори та статичні класи розширення функціональності.

В сервісах всі об’єкти витягуються у вигляді сутностей рівня бази даних, проте мають передаватися на рівень представлення у вигляді власних сутностей (через те, що, по перше, рівень представлення не має знати про сутності бази даних, а, по друге, певні дані потрібно перевести у більш зручний вигляд), теж саме стосується й передачі даних від рівня представлення до бази даних.

Майже кожна функція в сервісах потребує перетворення одних об’єктів в інші, тож, потрібно винести процес перетворення об’єктів з сервісів. Для перетворення схожих об’єктів існує стороння бібліотека AutoMapper, проте, вона має ряд значних недоліків: необхідність визначення правил перетворення в проєкті представлення (чи принаймні виклик конфігурацій на рівні представлення); складність опису правил перетворення ієрархічних та комплексних класів, а також перетворень зі складною логікою; більшість помилок, припущених під час опису правил перетворень, будуть виявлені лише після запуску серверу. Тож, було обрано інший варіант – створення класів розширення функціональності (недоступних для інших проєктів) для кожної сутності рівнів бізнес-логіки та доступу до бази даних.

Також, для швидкого видалення дублікатів тегів в рецепті та залежностей між продуктами, були створені два компаратори для методу Distinct (метод видалення дублікатів, що за замовчуванням порівнює об’єкти за посиланнями).

Рисунок 3.7 - Схема структурна класів сервісів

Рисунок 3.8 - Схема структурна класів об’єктів передачі даних

3.2.1.3 Рівень прикладного програмного інтерфейсу

Для організації передачі повідомлень між клієнтом та сервером було обрано спосіб побудови веб-застосування ASP.NET Core Web API, призначений для роботи за принципами REST (Representation State Transfer – «передача станів представлення».

Проєкт рівня прикладного програмного інтерфейсу містить: класи, призначені для ідентифікації користувачів; контролери, додаткові моделі, фільтр перехвату помилок та файли налаштування.

Для спрощення процесів реєстрації, аутентифікації та авторизації користувачів було використано набір бібліотек AspNetCore.Identity. Цей модуль вимагає створення додаткового контексту та бази даних для збереження персональних даних користувачів. Підтримка релевантності двох різних баз даних на різних рівнях, вимагає додаткових перевірок та міграцій, проте, класи для доступу до бази користувачів не були перенесені на рівень доступу до бази даних та контексти не були об’єднані через кілька проблем: якщо буде викрадена дані основної бази даних, то будуть й втрачені дані користувачів; складність поєднання двох баз даних, адже сутності користувачів мають строкові ідентифікатори; необхідність перенесення класів керування даними користувачів на рівень бізнес-логіки; контекст бази даних користувачів має наслідуватися від особливого контексту (IdentityContext), використання якого потребує встановлення відповідних пакетів (така ж проблема на рівні бізнес-логіки), а це своєю чергою, порушує один з головних принципів побудови багаторівневих програм (нижній рівень не має знати та залежати від реалізації верхнього рівня).

Для аутентифікації користувачів використовується JWT (Json Web Token), який отримується після вводу ім’я-ідентифікатора та паролю. Надалі цей токен відправляється у заголовку запита, а система перевіряє його і в залежності від результату надає чи забороняє доступ до визначеного функціонала. Токени мають час придатності та користувач повинен заходити у систему повторно після вичерпання часу.

Для тестування працездатності програмного забезпечення на етапі розробки, була використана бібліотека Swagger, що показує усі публічні адреси методів контролерів, структуру вхідних та вихідних даних, приклади вхідних даних, та надає змогу легко здійснити запити до сервера. Для імітування процесу знаходження в обліковому записі, було додано додатковий компонент, що зберігає токен тестового користувача та додає його у заголовок запиту (Рисунок 3.9 -).

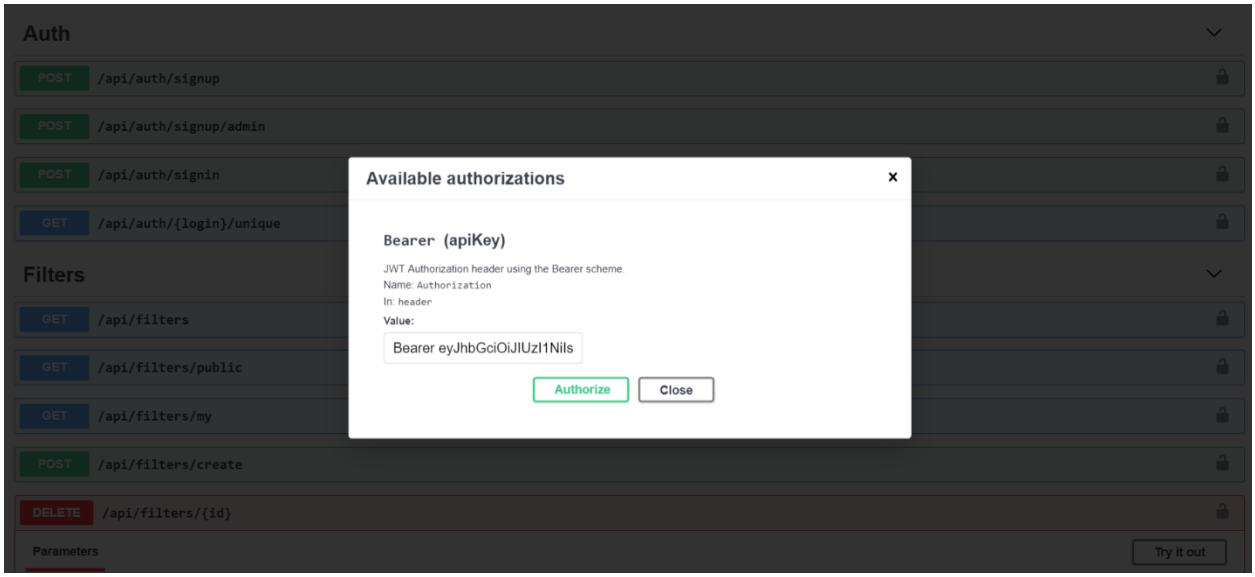


Рисунок 3.9 - Вхід в обліковий запис через Swagger

Далі наведено таблицю, що описує усі запити доступні клієнтській частині, їх короткий опис та приклади вхідних/вихідних значень (Таблиця 3.1 -).

Таблиця 3.1 - Специфікація публічних методів контролерів

Запит	Опис	Приклад вхідних даних	Приклад вихідних даних
AuthController			

/auth/signup
POST
	Реєстрація користувача	{
		"login": "SomeLogin",
		"email": "all@me.com",
		"password": "1234Test",
		"fullName": "A Stark"
		}
			{
			"succeeded": true,
			"errors": []
			}

/auth/signup/
/admin
POST
	Реєстрація адміністратора (метод є доступним лише для адміністраторів)	{
		"login": "Name",
		"email": "it@me.com",
		"password": "abc45678",
		}
			{
			"succeeded": false,
			"errors": [“Invalid password”]
			}

Продовження таблиці 3.1

/auth/signin
POST
	Вхід в обліковий запис	{
		"login": "string",
		"password": "string"
		}
			{
			“token”:
			“euFwiG23cDefg91…”,
			“isAdmin”: false
			}

/auth/passwords
/change
POST
	Зміна паролю користувача	{
		"login": "TestLogin",
		"oldpassword": "Test1234",
		"newpassword": "Iss1234",
		}
			{
			"succeeded": true,
			"errors": []
			}

/auth/{login}/ /unique GET	Перевірка існування користувача з вказаним ідентифікатором	/auth/banana/unique login = banana	true (вже існує користувач з таким ідентифікатором)
FiltersController			
/filters GET	Отримання списку доступних фільтрів (усіх публічних та створених цим користувачем)	_____	[{ "id": 2, "filterTitle": "Main filter", "description": "no desc", "isDefault": false }]
/filters/public GET	Отримання списку публічних фільтрів		[{ "id": 76, "filterTitle": "Carrots filter", "description": "-“, "isDefault": true }]
/filters/my GET	Отримання списку фільтрів, створених поточним автентифікованим користувачем		[{ "id": 32, "filterTitle": "Test me", "description": null, "isDefault": false }]
/filters/{id} DELETE	Видалення фільтру за його ідентифікатором (видалення власного фільтра доступно лише автентифікованому користувачу; видалення публічних фільтрів – тільки адміністратору)	/filters/12 id = 12	No content

Продовження таблиці 3.1

/filters/create
POST
	Створення фільтру (є доступним тільки для автентифікованих користувачів, інакше – помилка доступу; створення публічних фільтрів є доступним тільки для адміністраторів, інакше – помилка доступу)	{
		"recipeTitle": "Re",
		"onlyProducts": true,
		"byAvailableProducts": true,
		"authorId": 0,
		"minDuration": 20,
		"maxDuration": 345,
		"minCalories": 0,
		"maxCalories": 5.2,
		"ingredients": [{
		"id": 0,
		"necessity": true,
		"productId": 5
		}],
		"tags": [{
		"id": 0,
		"necessity": true,
		"tag": "Tag1"
		}],
		"id": 0,
		"filterTitle": "Filter Tad",
		"description": "no",
		"isDefault": false
		}
			5
			
			(ідентифікатор, створеного фільтру)

/filters/{id}/
/details
GET
	Отримання детального опису фільтру (доступно лише, якщо користувач викликає публічний чи власний фільтр; або запит робить адміністратор)	/filters/2/details
		
		id = 2
			{
			"recipeTitle": null,
			"onlyProducts": false,
			"byAvailableProducts": true,
			"authorId": 125,
			"minDuration": 3,
			"maxDuration": null,
			"minCalories": 7.5,
			"maxCalories": 333.3,
			"ingredients": [{
			"id": 3,
			"necessity": true,
			"productId": 100
			}],
			"tags": [{
			"id": 0,
			"necessity": true,
			"tag": "Tag1"
			}],
			"id": 2,
			"filterTitle": "Filter Tad",
			"description": "no",
			"isDefault": false
			}

Продовження таблиці 3.1

ImagesController			
/images GET	Отримання усіх зображень (доступно лише адміністратору)		[{ "id": 1, "title": "unnamed.jpg", "data": "/9j/4AAQSkZgABAQAAB…" }]

/images
POST
	Завантаження на сервер зображення	{
		"id": 0,
		"title": "Melon",
		"data": "/9j/24gQAB…"
		}
			65
			
			(ідентифікатор нового зображення)

/images/{id}
GET
	Отримання зображення за його ідентифікатором	/images/5
		
		id = 5
			{
			"id": 5,
			"title": "meg.jpg",
			"data":
			"/9j/2BAQSkzGABAQaAB…"
			}

/images/{id} DELETE	Видалення зображення за його ідентифікатором (лише для адміністратора)	/images/7 id = 7	No content

/images/get/
/bulk
POST
	Отримання множини зображень за їх ідентифікаторами	[11, 2]	[{
			"id": 11,
			"title": "unnamed2.jpg",
			"data": "…"
			}, {
			"id": 2,
			"title": "l.png",
			"data": "…"
			}]

ProductsController			

/products
GET
			[{
			"id": 1,
			"name": "Печиво",
			"imageId": 123
			}, {
			"id": 2,
			"name": "Сир",
			"imageId": null
			}]

/products/{id}
/subcategories
GET
	Отримання загального опису усіх підвидів заданого продукту (за його ідентифікатором)	/products/4
		/subcategories
			[{
			"id": 6,
			"name": "Молоко",
			"imageId": null
			}, {
			"id": 1014,
			"name": "Лимон",
			"imageId": null
			}]

Продовження таблиці 3.1

/products/{id}
/details
GET
		/products/1/details	{
			"image": {
			"id": 5,
			"title": "meg.jpg",
			"data": "..."
			},
			"subcategories": [],
			"categories": [{
			"id": 23,
			"productId": 12
			}],
			"description": "d",
			"calories": 83,
			"fats": 5.26,
			"carbohydrates": 9.98,
			"squirrels": 1.18,
			"water": 82.31,
			"ash": 1.28,
			"sugar": 0.7,
			"cellulose": 1,
			"starch": null,
			"transFats": 0,
			"cholesterol": 0,
			"id": 1002,
			"name": "Соєвий сир тофу",
			"imageId": 5
			}

/products/{id} DELETE	Видалення продукту за ідентифікатором (доступно лише адміністратору)	/products/5	No content
/products/{id}/ /categories GET	Отримання загального опису усіх категорій заданого продукту	/products/1001 /categories	[{ "id": 12, "name": "Соєві продукти", "imageId": 32 }]
/products/ /withRelations GET	Отримання детального опису (без фотографій) усіх продуктів, де категорії та підвиди продукти, задаються списком ідентифікаторів		[{ "subcategories": [3 ,4, 7], "categories": [100, 200], "description": null, "calories": null, "fats": null, "carbohydrates": null, "squirrels": 100, "water": null, "ash": null, "sugar": null, "cellulose": null, "starch": null, "transFats": null, "cholesterol": null, "id": 789, "name": "Apple", "imageId": 45 }]

Продовження таблиці 3.1

/products/
/create
POST
	Створення продукту (доступно лише адміністратору)	{
		"image": {
		"id": 0,
		"data": "..."
		},
		"subcategories": [{
		"id": 0,
		"productId": 9
		}],
		"categories": [{
		"id": 0,
		"productId": 13
		}],
		"description": "string",
		"calories": 0,
		"fats": 0,
		"carbohydrates": 0,
		"squirrels": 234,
		"water": 0,
		"ash": 0,
		"sugar": 0,
		"cellulose": 0,
		"starch": 0,
		"transFats": 11.5,
		"cholesterol": 0,
		"id": 0,
		"name": "Super",
		"imageId": 0
		}
			888
			
			(ідентифікатор нового продукту)

/products/edit
POST
	Редагування продукту (доступно лише адміністратору)	{
		"image": null,
		"subcategories": [],
		"categories": [{
		"id": 23,
		"productId": 12
		}],
		"description": "DDD",
		"calories": 83,
		"fats": 5.26,
		"carbohydrates": 9.98,
		"squirrels": 1.18,
		"water": 82.31,
		"ash": 1.28,
		"sugar": 0.7,
		"cellulose": 1,
		"starch": null,
		"transFats": 0,
		"cholesterol": 0,
		"id": 1002,
		"name": "Соєвий сир",
		"imageId": 0
		}
			true
			
			(продукт оновлено)

Продовження таблиці 3.1

ProfilesController			

/profiles/
/current
GET
	Отримання детального поточного облікового запису (лише для аутентифікованих користувачів)		{
			"email":
			"melo@epam.com",
			"fullName": "Full Name",
			"birthDay": null,
			"description": null,
			"avatarId": null,
			"avatar": null,
			"id": 1,
			"displayName": "Melo",
			"role": "USER",
			}

/profiles/{id}
GET
	Отримання детального опису облікового запису (за його ідентифікатором)	/profiles/1	{
			"email":
			"melo@epam.com",
			"fullName": "Full Name",
			"birthDay": null,
			"description": null,
			"avatarId": null,
			"avatar": null,
			"id": 1,
			"displayName": "Melo",
			"role": "USER",
			}

/profiles/edit
POST
	Редагування облікового запису (доступно лише його аутентифікованого власника)	{
		"email": "m@k.co",
		"fullName": "C&B",
		"birthDay": null,
		"description": "no",
		"avatarId": null,
		"avatar": null,
		"id": 1,
		"displayName": "Melo",
		"role": "USER",
		}
			true
			
			(профіль оновлено)

RecipesController			
/recipes GET	Отримання усіх рецептів (їх загального опису)		[{ "id": 1, "title": "Хачапури", "description": "----", "time": "2020-05- 12T14:04:49.4742831", "authorId": 5, "imageId": 131 }]
/recipes/{id} DELETE	Видалення рецепту за його ідентифікатором (лише для автентифікованого автора рецепта)	/recipes/5	No content

Продовження таблиці 3.1

/recipes/{id}
 	
GET
	Отримання рецепту за його ідентифікатором (без детального опису, але з деякими подробицями)	/recipes/1
		
		id = 1
			{
			"image": null,
			"calories": 450,
			"portions": null,
			"duration": null,
			"ingredientsDescription": "ввввввв",
			"ingredients": [],
			"steps": [],
			"tags": [],
			"id": 1,
			"title": "Хачапури",
			"description": "К---",
			"time": "2020-05-12T14:04:49.4742831",
			"authorId": 5,
			"imageId": 131
			}

/recipes/create
 	
POST
	Створення рецепту (лише для зареєстрованих користувачів)	{
		"image": {
		"id": 0,
		"data": "..."
		},
		"calories": 210.4,
		"portions": 5,
		"duration": 60,
		"ingredientsDescription": "...",
		"ingredients": [{
		"id": 0,
		"note": "some note",
		"weight": 234.5,
		"necessity": true,
		"productId": 4,
		}],
		"steps": [{
		"id": 0,
		"title": "T",
		"description": "D",
		"imageId": 0,
		"image": {
		"id": 0,
		"title": "T.png",
		"data": "..."
		}}],
		"tags": [{
		"id": 0,
		"tag": "Eat"
		}],
		"id": 0,
		"title": "Tile",
		"description": "none",
		"authorId": 0,
		"imageId": 0
		}
			4
			
			(ідентифікатор створеного рецепту)

Продовження таблиці 3.1

/recipes/create
/default
POST
	Створення стандартного (без автора) рецепту адміністратором	{
		"image": { … }
		"calories": null,
		"portions": 5,
		"duration": 60,
		"steps": [...],
		"tags": [],
		"id": 0,
		"name": "RED"
		}
			5
			
			(ідентифікатор створеного рецепту)

/recipes/edit
PUT
	Редагування рецепту (доступно лише автентифікованому автору рецепта)	{
		"image": { … }
		"calories": null,
		"portions": 5,
		"duration": 60,
		"tags": [],
		"id": 13,
		"steps": [{
		"id": 4,
		"description": "New",
		"imageId": 0,
		"image": {
		"id": 0,
		"title": "Ttt.png",
		"data": "…"
		}}],
		"name": "red 2"
		…
		}
			true
			
			(рецепт оновлено)

/recipes/tags GET	Отримання списку усіх тегів, що зустрічаються у рецептах в системі		["t1", "sleep", "Russian", "cakes"]
/recipes/profiles /{id} GET	Отримання усіх рецептів, що належать певному автору (за ідентифікатором автора)	/recipes/profiles/2 id = 2	[{ "id": 4, "title": "Melon", "description": "-", "time": "2020-05-16T14:04:49.4742831", "authorId": 2, "imageId": 66 }]
/recipes/default GET	Отримання списку усіх стандартних рецептів (доступно лише адміністратору)		[{ "id": 352, "title": "Golden cake", "description": "---", "time": "2020-05-11T17:04:29.8312", "authorId": null, "imageId": 601 }]

Продовження таблиці 3.1

/recipes/{id}
/details
GET
	Отримання детального опису рецепта за його ідентифікатором	/recipes/55/details
		
		id = 55
			{
			"image": {
			"id": 678,
			"data": "…"
			},
			"calories": 200,
			"portions": 5,
			"duration": 0,
			"ingredientsDescription":
			"…",
			"ingredients": [{
			"id": 6,
			"note": "some note",
			"weight": 23.5,
			"title": "Лимон",
			"necessity": false,
			"productId": 8,
			}],
			"steps": [{
			"id": 3,
			"title": "you",
			"description": "D",
			"imageId": null,
			"image": null,
			"tags": [{
			"id": 53,
			"tag": "Eat"
			}],
			"id": 55,
			"title": "Tile",
			"description": "none",
			"authorId": 2,
			"imageId": 678
			}

/recipes/filters
POST
	Пошук рецептів за вказаними критеріями	{
		"recipeTitle": "O",
		"onlyProducts": false,
		"authorId": 0,
		"minDuration": 0,
		"maxCalories": 0,
		"ingredients": [{
		"id": 0,
		"necessity": false,
		"productId": 5
		}],
		"tags": [{
		"id": 0,
		"necessity": true,
		"tag": "tag"
		}],
		"id": 0,
		"filterTitle": "MM",
		"description": "no",
		"isDefault": false
		}
			[{
			"id": 4,
			"title": "Melon",
			"description": "-",
			"time": "2020-05-
			16T14:04:49.4742831",
			"authorId": 2,
			"imageId": 66
			}]

Продовження таблиці 3.1

/recipes/filters/{ id} GET	Пошук рецептів за ідентифікатором фільтру	/recipes/filters/3 id = 3	[{ "id": 987, "title": "Juice", "description": "-", "time": "2020-05-03T11:04:49.4742831", "authorId": 5, "imageId": 551 }]
/recipes/my GET	Отримання усіх рецептів, автором якого є поточний користувач (лише для автентифікованих користувачів)		[{ "id": 20, "title": "Fille", "description": "-", "authorId": 7, "imageId": 660 }]
/recipes/my DELETE	Видалення усіх створених рецептів (лише для автентифікованого власника рецептів)		No content

3.2.2 Архітектура клієнтської частини

Клієнтська частина побудована за допомогою платформи Angular версії 9 на мовах програмування TypeScript (надбудова над мовою Javascript), HTML, SCSS. Для виконання асинхронних операцій (наприклад, для отримання даних з сервера) використовується бібліотека RxJS. Розміщення елементів на сторінці та їх адаптованість під середні екрани спрощується, завдяки використанню модулю макету гнучкого контейнера Flexbox.

Дизайн сайту розроблений за концепцію Material Design, а для базової стилізації окремих елементів використовується бібліотека Angular Material.

Проєкт клієнтської частини окрім основного, поділено ще на 3 модулі за основними направленнями використання:

– модуль облікового запису, що відповідає за реєстрацію, вхід та вихід з системи;

– модуль рецептів, що відповідає за фільтрацію рецептів, їх перегляд та керування ними;

– модуль продуктів, що відповідає за перегляд продуктів та керування ними адміністратором.

Загалом проєкт має наступну структуру (враховуються лише елементи, створені розробником даного веб-застосування):

а) components – містить компоненти, що використовуються в головному модулі або в декількох модулях:

– HeaderComponent – компонент, що відповідає за показ головного навігаційного меню;

– ConfirmDialogComponent – компонент, що є вікном підтвердженням якихось дій;

– NotFoundComponent – компонент, куди перенаправляється користувач, якщо він використав некоректне посилання;

б) models – містить усі інтерфейси та класи, що використовуються як моделі; має теку server, що містить інтерфейси усіх моделей, що можуть використовуватися для обміну даними з серверною частиною;

в) products – модуль продуктів, окрім файлу опису модуля та налаштування маршрутів, містить три компоненти:

– ProductDetailsComponent – компонент, що використовується як діалогове вікно для швидкого показу інформації про продукт;

– ProductListComponent – компонент для демонстрації повної сторінки продукту, що містить секцію деталей (зображення, опис, вміст поживних речовин та список категорій продуктів) та пре в'ю усіх продуктів, що є підвидами поточного продукту; можливий перехід між сторінками продуктів шляхом натискання на їх категорії чи на назву продуктів (за замовчуванням першою сторінкою є список усіх продуктів без категорій);

– ProductEditor – компонент, що відповідає за редагування та створення нового продукту адміністратором;

г) profiles – модуль облікових записів, що містить:

– SignUpComponent – компонент, що відповідає за реєстрацію користувача;

– SignInComponent – компонент, що відповідає за вхід користувача в обліковий запис;

– ChangePasswordComponent – компонент, що відповідає за зміну паролю користувача;

– UserComponent – головний компонент модуля, що розміщує усі інші у вигляді внутрішніх сторінок;

д) recipes – модуль рецептів, що містить наступні компоненти:

– RecipeEditorComponent – компонент, що містить форму для створення та редагування рецептів;

– RecipeDetailsComponent – компонент, що демонструє детальний опис рецепту (використовується в компонентах RecipeDetailsDialog та RecipeDetailsPage);

– RecipeDetailsDialogComponent – компонент, що розміщує опис рецепту у вигляді вікна;

– RecipeDetailsPageComponent – компонент, що розміщує опис рецепту у вигляді окремої сторінки;

– RecipeListComponent – компонент, що демонструє список рецептів у вигляді карток з їхніми назвами, описом та фотографією (використовується в компонентах MyRecipes та RecipeSearch);

– MyRecipesComponent – компонент, що надає можливість керування створеними продуктами (демонстрація їх списку, видалення та перехід на сторінку перегляду та редагування);

– RecipeSearchComponet – компонент пошуку інгредієнтів, окрім демонстрації результатів (списку рецептів), має п’ять секцій для керування критеріями фільтру:

– LimitsSearchSectionComponent – секція загальних обмежень, надає можливість вказати частину назви рецепта та обрати граничні значення калорійності й тривалості приготування;

– IngredientsSearchSectionComponent – секція визначення списку інгредієнтів та типу для пошуку, що містить декілька кнопок керування та список продуктів у вигляді таблиці, в якій вказані назви продуктів та вміст основних поживних речовин в них;

– TagsSearchSectionComponent – секція визначення списків заборонених та обов’язкових тегів;

– FiltersSearchSectionComponent – секція містить список вибору фільтру для застосування пошуку або використання його як основи для іншого фільтру, та поле збереження поточного фільтра для зареєстрованого користувача;

– PreviewSearchSectionComponent – секція, що демонструє усі обрані критерії пошуку, вона не надає можливості додавання критеріїв, але надає можливість видалення певних критеріїв;

е) shared – інші файли (не компоненти), що можуть використовуватися у різних модулях:

– MaterialModule – модуль, що об’єднує усі необхідні модулі бібліотеки Angular Material;

– ServerUrls – список основних шляхів для взаємодії з серверною частиною;

– SharedModule – модуль, що об’єдную елементи, які мають знаходитися в різних модулях;

– Measurements – список стандартних одиниць виміру інгредієнтів;

– UkrainianPaginatorIntl – клас перекладу секції переходи між сторінками українською мовою;

ж) styles – містить файли глобальних стилів;

з) auth – містить interceptor (що додає за наявністю токен до кожного запиту, та обробляє помилки пов'язані з авторизацією та аутентифікацією) та guard, що ставиться на певні маршрути з метою заборонити переходити на ці сторінки незареєстрованим користувачам;

и) services – містить наступні сервіси:

– AuthService – сервіс, що здійснює збереженням, очищення та діставання значень з local storage, пов’язаних з авторизацією; був створений для зменшення зв’язності local storage та різних компонент, що спростить подальше тестування;

– FiltersService – сервіс, призначений для збереження стану поточного фільтра та передачі даних про зміни усім секціям пошуку рецептів;

– ImagesService – сервіс для отримання зображень з сервера та збереження їх у системі (можливість доступу до зображень без необхідності посилання запитів до сервера) до оновлення сторінки;

– ProductsService – сервіс для швидкого доступу до продуктів та збереження їх у системі;

– RecipesService – сервіс для збереження та керування списку рецептів під час пошуку;

– ThemeService – сервіс для керування темами;

– ServerHttpService – сервіс, що містить функції всіх можливих запитів до сервера.

Кожний компонент складається з трьох основних файлів: .ts (містить логіку компонента), .scss (стилізує компонент), .html (створює макет розташування елементів на сторінці).

3.3 Аналіз безпеки даних

Усі публічні методи контролерів, що певним чином можуть змінити дані в системі недоступні для незареєстрованих користувачів. Під час спроби оновлення та видалення рецептів чи фільтрів, завжди перевіряється, що користувач, який здійснює ці запити є автором вказаних елементів і тільки тоді йому надається одноразовий доступ до цієї функціональності. Під час кожної спроби додати, змінити чи видалити продукти, а також створити публічні фільтри здійснюється перевірка ролі користувача та надається доступ лише у тому випадку, якщо він є адміністратором.

Ідентифікатор та пароль адміністратора за замовчуванням записується в базу даних з файлу особистих налаштувань, який не зберігається в системі, а розташований за її межами програмою менеджеру секретів.

Всі паролі користувачів зберігаються в окремій базі даних у вигляді хешів без можливості відтворити пароль, проте з можливістю перевірити його збіг з введеним користувачем паролем. Ключ до хешування теж зберігається в окремому файлі налаштувань, який розміщується менеджером секретів.

Система приймає запити лише від визначеного списку клієнтів та забороняє інші. Передача даних між клієнтом та сервісом виконується за безпечним протоколом HTTPS.

Висновок до розділу

В цьому розділі було створено архітектури веб-застосування в цілому, його серверної та клієнтської частин та їх компонентів. Серверна частина була створена з використанням трирівневої архітектури та виокремленням усіх абстракції та інтерфейсів за межі проєкту з реалізацією.

Були визначені технології, які використовуються для розробки даного програмного забезпечення: Entity Framework Core (підхід Code First), ASP Net Core Web.API, AspNetCore.Identity, Angular, RxJS, Angular Material, Swagger.

Були визначені мови розробки: C#, TypeScript, HTML, SCSS.

Було проаналізовано безпеку даних та не знайдено критичних ризиків для програмного забезпечення.

4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1 Випробування програмного продукту

Тестування є невід'ємною частиною розробки програмного забезпечення та є дуже важливим з точку зору аналізу якості програмного продукту та його підтримки.

Випробування даної розробки складається з двох частин: інтеграційне тестування усієї серверної частини (за допомогою виклику методів контролерів) та тестування наскрізних сценаріїв разом з клієнтською частиною (за допомогою веб-браузера).

4.1.1 Мета випробувань

Метою випробувань є визначення працездатності програмного продукту та відповідності його усім зазначеним вимогам.

4.1.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603-92. Інформаційна технологія. Види випробувань автоматизованих систем;

– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

4.1.3 Випробування серверної частини

Для спрощення процесу тестування до проекту була підключена бібліотека Swagger, що надає можливість побачити усі публічні адреси методів контролерів та список моделей, які використовуються для передачі даних з сервера чи на сервер. Окрім цього, за допомогою цього модулю можна побачити приклади вхідних даних для запиту та детальну структуру моделі, що передається. Проте, однією з головних функцій цієї бібліотеки є надання

можливості швидко та зручно здійснювати запити на сервер. Для імітування процесу знаходження в обліковому записі, було додано додатковий компонент, що зберігає токен тестового користувача та додає його у заголовок запиту. Токен же можна отримати через публічні методи реєстрації та ідентифікації в системі (Рисунок 3.9).

Перевірка усіх методів контролерів окремо було виконана під час створення таблиці (Таблиця 3.1). Було виявлено дві помилки тож, було проведено повторне тестування з кількома зв’язаними сценаріями та використанням критичних значень. Для тестування була створена окрема база даних, в якій початкові дані, заповнювалися ініціалізатором контексту.

Під час тестування серверної частини перевірено 39 методів контролерів та виконано 72 різних тестів. Результати є наступними:

– 61 тест пройшов успішно з першого разу;

– 7 тестів було провалені через помилки при описі моделі та при повторному проходженні стали успішними;

– 3 тести були провалені через відсутність перевірки на пусті в одному з методів бізнес-логіки, проблема була швидко вирішена та подальші тести показали успішний результат;

– 1 тест було провалено через використання некоректного методу, проблема була вирішена та подальше тестування було успішним.

Також під час тестування були знайдені два методи, що надають одну і ту ж саму функціональність, але мають різні назви, тож довелося видалити один з методів.

Тестування усіх функцій, що пов’язані з зазначеними вимогами, пройшло успішно.

Приклади результатів тестів наведені нижче (Рисунки 4.1 – 4.8).

Рисунок 4.1 - Запит на вхід в обліковий запис

Рисунок 4.2 - Відповідь на запит входу в обліковий запис

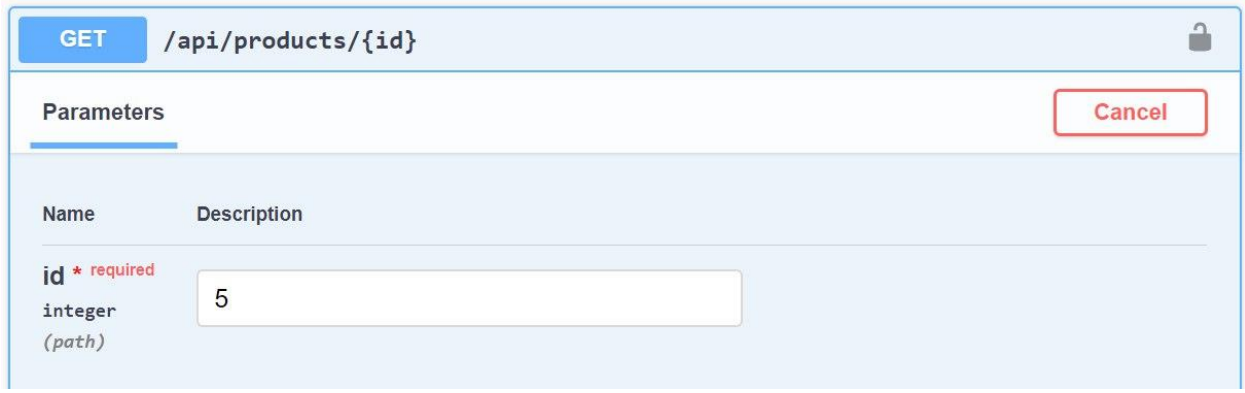


Рисунок 4.3 - Запит на отримання продукту за ідентифікатором

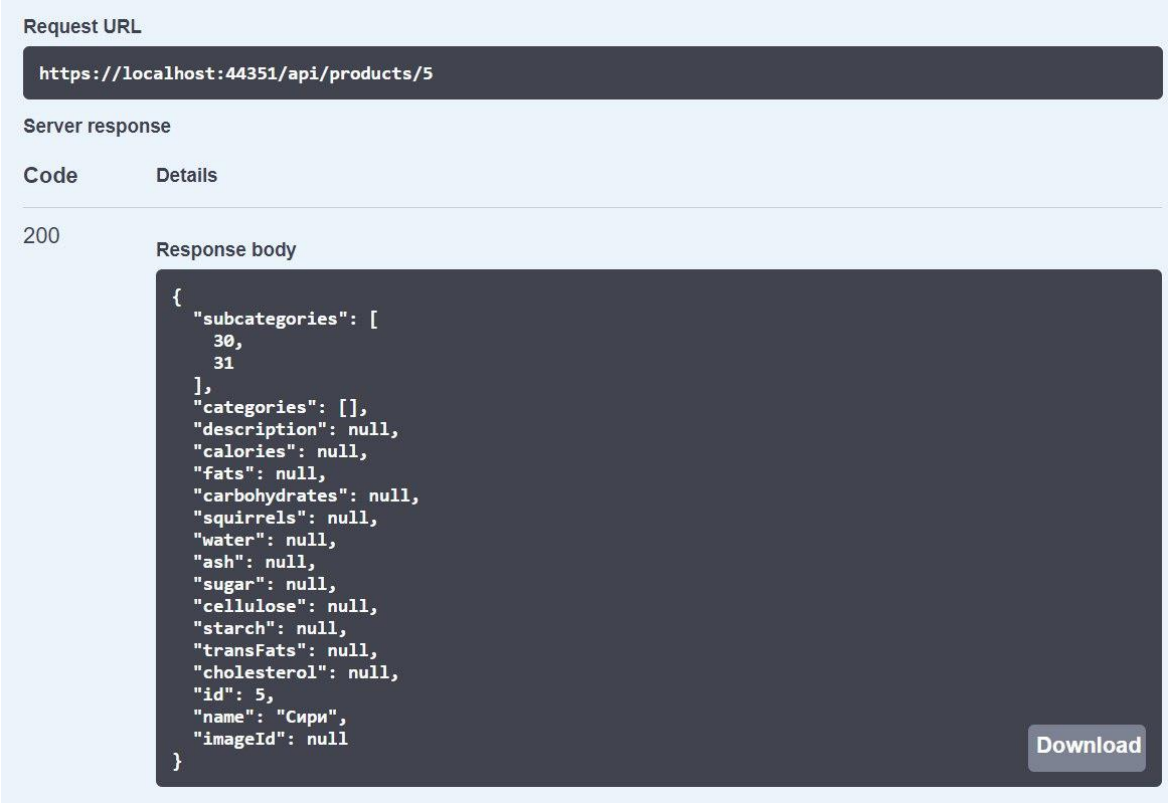


Рисунок 4.4 - Відповідь на запит отримання продукту за ідентифікатором

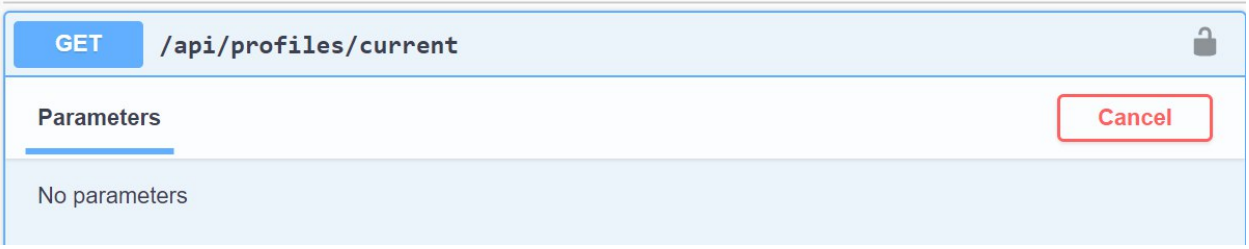


Рисунок 4.5 - Запит на отримання поточного опису поточного облікового запису (без входу у обліковий запис)

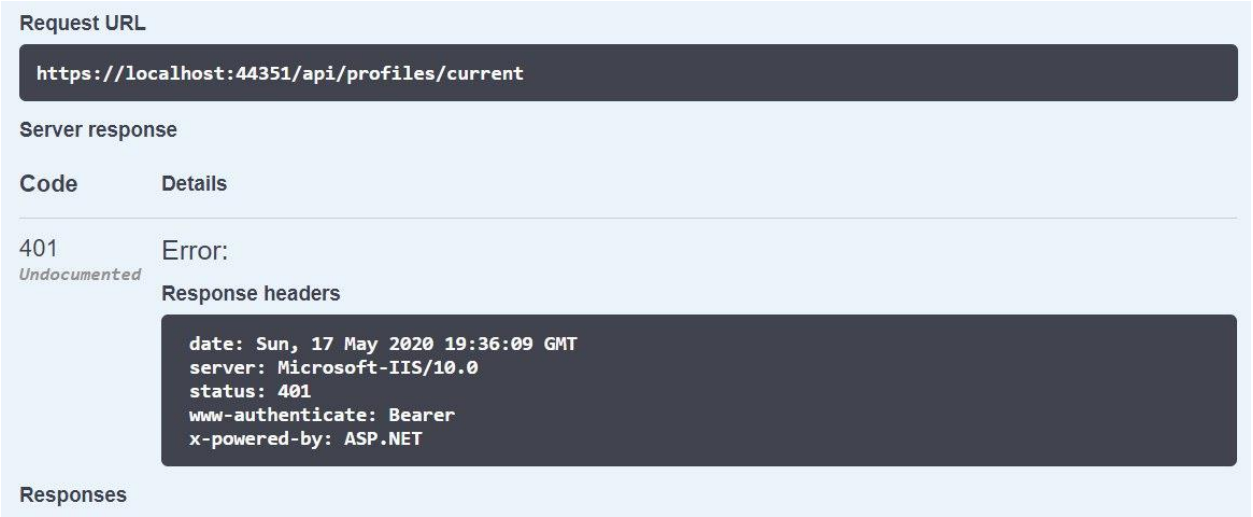


Рисунок 4.6 - Відповідь на запит отримання поточного облікового запису для неавторизованих користувачів

Рисунок 4.7 - Запит на отримання рецептів за назвою

Рисунок 4.8 - Відповідь на запит отримання рецептів за назвою

4.1.4 Випробування клієнтської та серверної частин разом

Для тестування роботи програмного забезпечення в цілому було обрано декілька ключових сценаріїв, які описані у вигляді тест кейсів. Тест кейси поділяються на позитивні та негативні в залежності від очікуваних результатів. Позитивними є тест кейси, що перевіряють роботу програмного

забезпечення з коректними даними. Негативні тест кейси призначені для перевірки поведінки у виняткових ситуаціях.

Далі наведено перелік головних позитивних тест кейсів (Таблиця 4.1) та приклади негативних (Таблиця 4.2).

Таблиця 4.1 - Позитивні тест кейси

Тест кейс	Кроки відтворення	Очікуваний результат
Реєстрація та вхід в обліковий запис	1) перейти на сторінку реєстрації, заповнити поле логіну унікальним ідентифікатором, ввести складний пароль та повторити його у полі підтвердження; 2) натиснути кнопку реєстрації; 3) перейти на сторінку входу в систему, ввести логін та пароль з першого кроку та натиснути кнопку входу.	Після першого кроку стає доступною кнопка реєстрації. Після другого кроку виводиться повідомлення про успішну реєстрацію. Після третього кроку виводиться повідомлення про успішний вхід в систему, ім’я користувача з’являється на верхній панелі, пункти меню доповнюються діями керування рецептами та зміни паролю.
Керування фільтрами	1) увійти в обліковий запис, перейти на сторінку пошуку рецептів та заповнити хоча б один критерій на кожній секції пошуку; 2) перейти на секцію «Фільтри», ввести ім’я фільтру у відповідне поле; 3) натиснути кнопку збереження фільтру; 4) перезавантажити сторінку, перейти на секцію фільтрів та натиснути на фільтр з другого кроку; 5) переглянути секції пошуку; 6) повернутися до секції фільтрів та натиснути хрестик біля створеного фільтру;	Після другого кроку стає доступною кнопка збереження фільтру. Після третього кроку з’являється повідомлення про успішне створення фільтру та він з’являється у списку на секції фільтрів. Після четвертого кроку з’являється повідомлення про завантаження фільтру. На п’ятому кроці видно, що усі критерії пошуку з першого кроку активні. Після шостого кроку з’являється повідомлення про видалення фільтру та він зникає зі списку.

Продовження таблиці 4.1

Пошук рецептів за інгредієнтами	1) перейти на сторінку пошуку рецептів та відкрити секцію інгредієнтів; 2) обрати дві категорії з основних, такі що: існує хоча б один рецепт (А), інгредієнтом якого є перша категорія чи її продукти але нема продуктів другої; існує хоча б один рецепт (Б), інгредієнтами якого є перша та друга категорії одночасно або їх продукти; існує хоча б один рецепт (В), що не містить продуктів обох категорій. 3) першу категорія відмітити як обов'язкову, а другу як заборонену; 4) натиснути кнопку «Застосувати фільтр»; 5) натиснути на будь-який з отриманих рецептів.	Після першого кроку на екрані з'являється список рецептів, які відповідають критерію А, та тільки вони. Після другого кроку відкривається вікно з детальним описом рецепту та списком інгредієнтів, в якому є перша категорія чи її продукт.
Вихід з облікового запису	1) увійти в обліковий запис; 2) натиснути пункт головного меню «Обліковий запис» та обрати підпункт «Вихід».	Ім'я користувача зникає з екрану, кнопки керування рецептами та зміну паролю стають недоступними.

Продовження таблиці 4.1

Керування рецептами	1) увійти в обліковий запис та перейти на сторінку створення рецепту; 2) ввести коректні дані: назву, опис, калорійність, тривалість приготування, кількість порцій та тегі рецепту; завантажити зображення з розширенням .png та розміром до 1 мб; додати декілька інгредієнтів та обрати для них продукти; додати та повністю заповнити кроки приготування; 3) натиснути кнопку створення; 4) перейти на сторінку керування; 5) натиснути на створений рецепт; 6) натиснути на кнопку редагування; 7) видалити зображення та назву одного з кроків та натиснути кнопку оновлення; 8) перейти на сторінку редагування та натиснути на оновлений рецепт; 9) натиснути кнопку видалення на рецепті.	Після другого кроку стає доступною кнопка створення рецепту. Після третього кроку з'являється повідомлення про успішне створення. Після четвертого кроку, з'являється список рецептів користувача, серед яких є новостворений. Після п'ятого кроку з'являється вікно рецепту, що містить усі заповнені в другому кроці дані. Після шостого кроку з'являється сторінка редагування рецепту, заповнена даними з другого кроку. Після сьомого кроку з'являється повідомлення про успішне оновлення рецепту. Після восьмого кроку з'являється вікно з рецептом, де один з кроків має лише опис та номер кроку замість назви. Після дев'ятого кроку з'являється повідомлення про успішне видалення рецепту та він зникає зі списку рецептів.

Таблиця 4.2 - Негативні тест кейси

Тест кейс	Кроки відтворення	Очікуваний результат
Некоректне підтвердження паролю	1) перейти на сторінку реєстрації; 2) заповнити поля логіну та паролю коректними даними; 3) заповнити поле підтвердження паролю будь-яким іншим словом; 4) прибрати фокус з форми.	Поле підтвердження паролю стає червоним та з'являється попередження про те, що паролі не збігаються. Кнопка реєстрації залишається не доступною.

Продовження таблиці 4.2

Завантаження некоректних файлів зображення	1) увійти в обліковий запис та перейти на сторінку створення рецепту; 2) натиснути на кнопку завантаження зображення та обрати текстовий документ; 3) натиснути на кнопку завантаження зображення та обрати файл з розширенням .png та розміром більшим за 1 мб.	Після другого кроку з'являється повідомлення про недопустимий формат зображення. Після третього кроку з'являється повідомлення про занадто великий розмір зображення. У обох випадках прев'ю зображення не з'являється.
Некоректні межі тривалості приготування	1) перейти на сторінку пошуку рецептів та відкрити секцію основних обмежень; 2) заповнити межі тривалості приготування: поле «від» маленьким дробовим числом, а поле «до» великим цілим числом (наприклад, 1.1 та 999); 3) перейти на секцію прев'ю; 4) повернутися до секції основних обмежень та заповнити поля тривалості так, щоб поле «від» мало ціле значення більше за ціле значення полю «до» (наприклад, 100 та 1); 5) перейти на секцію прев'ю.	Після другого кроку поле «від» стане червоного кольору. На третьому кроці буде заповнена тільки верхня межа приготування. Після четвертого кроку обидва поля тривалості стануть червоними. На п'ятому кроці межі тривалості приготування не будуть заповнені.
Спроба відкрити сторінку створення рецепту неавторизованим користувачем	1) увійти в обліковий запис; 2) перейти на сторінку створення рецепту; 3) скопіювати посилання на сторінку; 4) вийти з облікового запису; 5) вставити посилання у адресний рядок та натиснути кнопку переходу.	Користувача буде пере направлено на сторінку 404 (сторінку не знайдено) та з'явиться повідомлення про необхідність входу в обліковий запис.

4.2 Розгортання програмного забезпечення

Дане веб-застосування орієнтоване на операційну систему Windows 10, тож далі описані кроки розгортання саме для цієї системи. Для використаних платформ є аналоги для операційних систем Mac OS та Linux, проте вони можуть вимагати додаткові налаштування.

Далі описано процес локального розгортання програмного забезпечення, розгортання на віддалених серверах (наприклад, на сервері Microsoft Azure) вже залежить від специфіки роботи з конкретними системами.

4.2.1 Розгортання серверної частини

Для розгортання серверної частини веб-застосування необхідно:

– встановити SDK .NET Core 3.0 (і вище);

– встановити Visual Studio 2017 (і вище) з налаштуваннями для .NET та C#;

– встановити Microsoft SQL Server 2008 (і вище);

– запустити проєкт з серверною частиною в Visual Studio;

– натиснути на проєкт CulinaryAssistant.WebAPI та обрати Керування секретами користувачів;

– заповнити відкритий файл налаштування наступними даними:

```
{
  "ConnectionStrings": {
    "AuthDBConnection": строка з’єднання з базою даних користувачів
    "MainDBConnection": строка з’єднання з головною базою даних
  },
  "ApplicationSettings": {
    "JWT_Secret": ключ для створення токенів
    "Admin_Login": ідентифікатор для головного адміністратора
    "Admin_Password": пароль для головного адміністратора
    "Client_URL": адреса клієнтської частини
  }
```

}

– запустити проєкт CulinaryAssistant.WebAPI в IIS (Internet Information Services).

4.2.2 Розгортання клієнтської частини

Для розгортання клієнтської частини веб-застосування необхідно:

– встановити сервер Node.js 10-12, пакетний менеджер npm та Angular CLI 9.0.7 (рекомендовано встановити саме такі версії, адже працездатність застосування з іншими версіями не є гарантованою);

– рекомендовано встановити Visual Code для роботи з кодом програми;

– відкрити теку з проєктом клієнтської частини та встановити необхідні пакети за допомогою консольної команди npm install;

– в файлі environment.ts вказати адресу api серверної частини в змінній serverBaseUrl;

– запустити проєкт за допомогою консольної команди npm start;

– відкрити сторінку за вказаною адресою в браузері.

4.3 Робота з програмним забезпеченням

Детальний опис роботи з даним веб-застосуванням наведено в документі «Керівництво користувача».

Висновок до розділу

У даному розділі було проаналізовано якість розробленого програмного забезпечення. Було виконано тестування серверної частини та знайдено декілька незначних помилок, що підтверджують необхідність тестування надалі. Створено, описано та перевірено тест кейси для тестування основних наскрізних сценаріїв. Було підтверджено виконання усіх вимог.

Було описано процеси локального розгортання серверної та клієнтської частин даного веб-застосування.

ВИСНОВКИ

Під час виконання даного дипломного проєкту було створено веб-застосування для пошуку та керування кулінарними рецептами, яке дозволяє: шукати рецепти за різноманітними критеріями (включаючи, пошук за продуктами та їх категоріями); зберігати та використовувати готові критерії пошуку; представляти рецепти у вигляді зручному для перегляду та специфічного пошуку; переглядати список продуктів, їх категорій та підкатегорій, а також їх складники та зображення; керувати списком продуктів адміністратору.

У розділі «Аналіз вимог до програмного забезпечення» було досліджено предметну область, проаналізовано наявні аналоги, визначено основні сценарії роботи, необхідні функціональні та нефункціональні вимоги, мету та задачі розробки. Проаналізовано наявні аналоги (їх переваги та недоліки), знайдено необхідні акценти та основні переваги даного програмного забезпечення.

У розділі «Інформаційне та алгоритмічне забезпечення» було визначено формат та спосіб створення додаткових вхідних даних, створено структуру бази даних. Досліджено основні математичні задачі у проєкті, знайдені та обґрунтовані їх рішення. Описано покрокові алгоритми пошуку кулінарних рецептів за інгредієнтами та їх категоріями.

У розділі «Програмне та технічне забезпечення» було розроблено модель та детальну архітектуру веб-застосування на різних рівнях. Описано використані технології, мови програмування та сторонні програмні забезпечення. Проаналізовано безпеку даних та не виявлено значних ризиків.

У розділі «Технологічний розділ» було проаналізовано якість програмного продукту, розроблено основні тест кейси та описано процес розгортання веб-застосування на локальній машині.

Розроблене веб-застосування відповідає усім вказаним вимогам, має належну якість, забезпечує безпеку та цілісність даних, має можливості для

швидкого розширення та модифікації, є зручним у використанні як користувачами так й адміністраторами.

ПЕРЕЛІК ПОСИЛАНЬ

1) Бєлов В. В., Воробйов Є. М., Шаталов В. Є. Теорія графів — М .: Вища. школа, 1976. — с. 392.

2) Великий тлумачний словник сучасної української мови: 250000 / уклад. та гол. ред. В. Т. Бусел. – Київ; Ірпінь: Перун, 2005. – 1728 с.

3) Еспозіто Д. Programming ASP.NET Core. — М.: Prentice Hall Ptr, 2018. – 1215 с.

4) Фрімен А. Angular для профессионалов. — СПб.: Пітер, 2018. — с. 800

5) Шілдт Г. Полный справочник по C#. Пер. с англ. — Видавничий Дім "Вільямс", 2014. – 752с.

6) Вебсайт – Вікіпедія [Електронний ресурс]. – 2020 – Режим доступу: https://uk.wikipedia.org/wiki/Вебсайт

7) Прикладний програмний інтерфейс [Електронний ресурс]. – 2019 – Режим доступу: https://developer.mozilla.org/uk/docs/Glossary/API

8) Тлумачний словник української мови [Електронний ресурс]. – 2020. – Режим доступу: https://slovnyk.ua/index.php

9) Documentation TypeScript [Електронний ресурс]. – 2019. – Режим доступу: https://www.typescriptlang.org/

10) RxJS Tutorial [Електронний ресурс]. – 2019. – Режим доступу: http://reactivex.io/rxjs/manual/tutorial.html

11) Категорії продуктів [Електронний ресурс]. – 2020. – Режим доступу: https://fitaudit.ru/categories

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «КУЛІНАРНИЙ ПОМІЧНИК»

Технічне завдання

КПІ.ІП-6104.045440.02.91

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Виконавець:

_____В.І. Брідня

Нормоконтроль:

_____ К.І. Ліщук

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосування «Кулінарний помічник»

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб-застосування «Кулінарний помічник» КПІ.ІП-6104.045440, котра використовується для створення та пошуку рецептів з певними критеріями та призначена для полегшення пошуку необхідного рецепту та передачі власних рецептів цільовій аудиторії.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосування є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для людей, які під час вибору кулінарних рецептів зважають на різноманітні специфічні критерії, а також для кухарів, які бажають поділитися рецептами своїх страв з відповідною цільовою аудиторією.

Метою розробки є полегшення процесу пошуку кулінарних рецептів серед наявних за критеріями, обумовленими певними вподобаннями, релігійними поглядами, моральними принципами, медичними показаннями і протипоказаннями та наявністю ресурсів, а також спрощення формування рецептів у вигляді, придатному для пошуку за вказаними критеріями, для передачі рецептів цільовій аудиторії.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- перегляд списку усіх рецептів;
- перегляд детального опису рецепту;
- перегляд списку продуктів;
- перегляд детального опису продукту;
- пошук рецептів за ключовими словами;
- використання готових фільтрів для пошуку рецептів;
- пошук рецептів за інгредієнтами;
- пошук рецептів за назвою;
- пошук рецептів за тривалістю приготування чи калорійністю;
- реєстрація облікового запису.

4.1.1.2 Для зареєстрованого користувача:

- функції, що належать звичайному користувачеві;
- вхід в обліковий запис;
- створення рецепту;
- редагування рецепту;
- видалення рецепту;
- перегляд списку власних рецептів;
- збереження критеріїв пошуку рецепту у вигляді фільтру;
- використання, збережених фільтрів;
- видалення власних збережених фільтрів.

4.1.1.3 Для адміністратора системи:

– всі функції, що належать зареєстрованому користувачеві;
– додання продукту до списку;
– видалення продукту;
– редагування продукту;
– створення публічних фільтрів;
– видалення публічних фільтрів.

4.1.2 Розробку виконати на платформі Windows.

4.1.3 Додаткові вимоги:

– мова клієнтського інтерфейсу – українська;
– адаптованість інтерфейсу під пристрої з діагоналлю екрану від 7 дюймів.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

Всі запроси, що передбачають зміну даних в системі, мають бути доступні лише для автентифікованих користувачів.

4.2.2 Передбачити захист від некоректних дій користувача.

Під час надходження запитів на оновлення та видалення певних даних, система має виконати перевірку того, що поточний користувач є автором цих даних, а інакше повернути повідомлення про помилку.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються

4.3.2 Обслуговування

Програмне забезпечення не вимагає специфічного обслуговування

4.3.3 Обслуговуючий персонал

Є необхідність в людині (адміністраторі), що створить базу продуктів та періодично буде її оновлювати.

4.4 Вимоги до складу і параметрів технічних засобів

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

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору ..... Intel i3.

4.4.2.2 Об‘єм ОЗП..... 2 ГБ.

4.4.2.2 Підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейств Windows 7, 8 та 10.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: HTTP запити з можливими URL параметрами, та JSON значеннями у тілі запиту.

4.5.3 Результати повинні бути представлені в наступному форматі: Відповіді на HTTP запити у форматі JSON.

4.5.4 Програмне забезпечення повинно обробляти запити, надіслані лише з визначених адресів.

4.5.5 Програмне забезпечення повинно взаємодіяти з сервером за допомогою протоколу HTTPS.

4.5.6 Клієнтська частина повинна бути розроблена за допомогою фреймворку Angular та мов програмування TypeScript, HTML, SASS, серверна частина має бути створена з використанням технології ASP.Net Core та мови програмування C#.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Розгорнути серверну та клієнтську частини на одному або окремих серверах.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

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

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 90 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Опис програми.

5.3.4 Керівництво користувача.

5.4 Графічна частина повинна бути виконана на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки.

5.4.1 Схема структурна варіантів використання.

5.4.2 Схема бази даних.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02.2020	
2.	Розробка технічного завдання	03.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	05.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	20.04.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до підрозділу «Випробування програмного продукту».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” ____ _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «КУЛІНАРНИЙ ПОМІЧНИК»

Керівництво користувача

КПІ.ІП-6104.045440.03.34

“ПОГОДЖЕНО”

Керівник проєкту:

К.І. Ліщук

Виконавець:

В.І. Брідня

Нормоконтроль:

К.І. Ліщук

Київ – 2020 року

КЕРІВНИЦТВО КОРИСТУВАЧА

Загальний опис інтерфейсу

Основні елементи сторінки:

- головне меню (недоступне для мобільних екранів) (рисунок 1 п.1);
- кнопка відкриття бокового меню (рисунок 1 п.2);
- кнопки швидкого керування обліковим записом (логін, реєстрація, вхід/вихід) (рисунок 1 п.3);
- кнопка зміни теми (рисунок 1 п.4);
- заголовок сторінки (рисунок 1 п.5);
- бокове меню (рисунок 2);
- контент сторінки (рисунок 1 п.7).

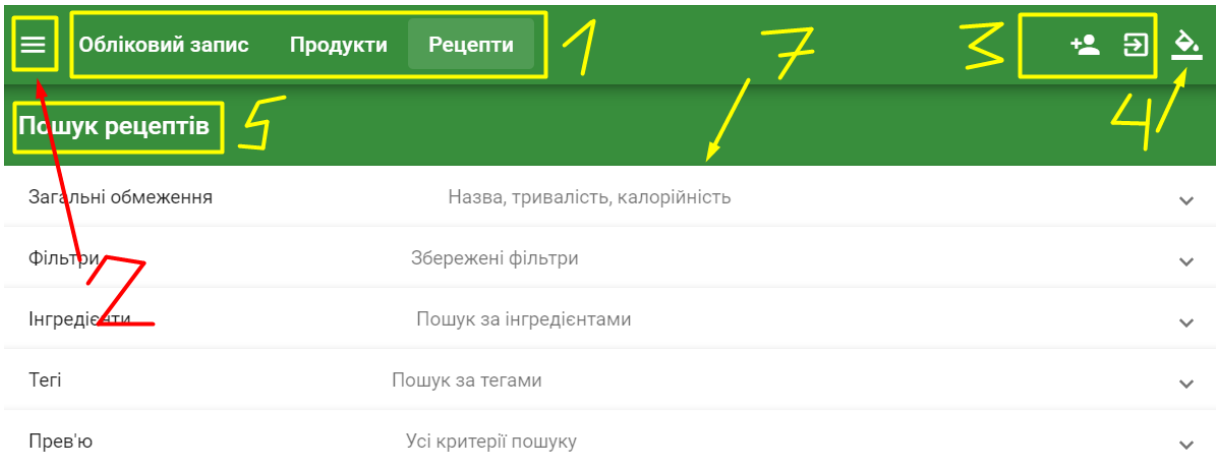


Рисунок 1 - Інтерфейс веб-застосування «Кулінарний помічник»

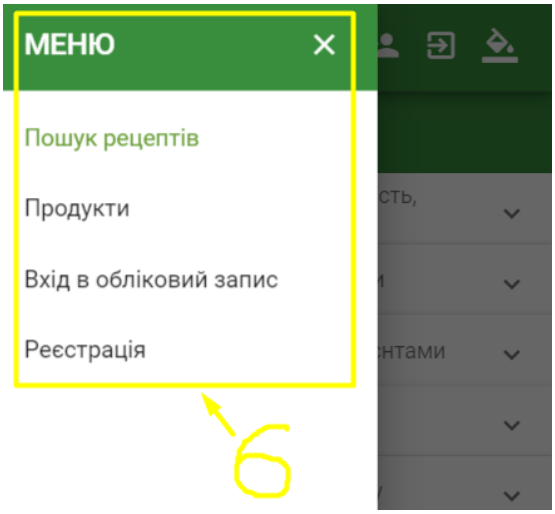


Рисунок 2 - Вигляд бокового меню для гостя

Зміна теми

Для зміни теми необхідно:

- натиснути на іконку зміни теми (рисунок 3 п.1) на верхній панелі;
- обрати тему серед наявних у відкритому списку (рисунок 3 п.2) (цифрами відмічені головні теми (рисунок 3 п.3), а інші є додатковими; іконка першого кольору позначає головний колір теми, а друга – колір акценту).

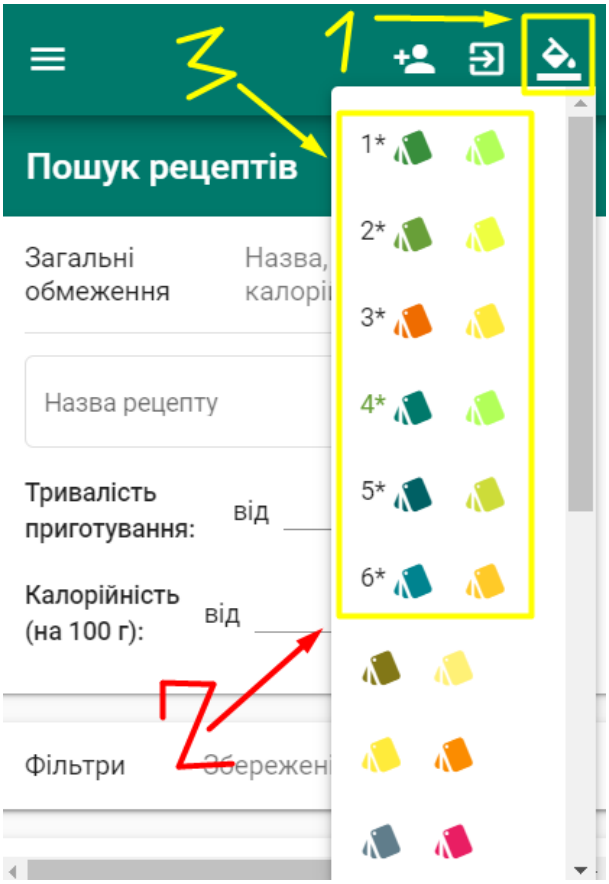


Рисунок 3 - Зміна теми веб-застосування

Пошук рецептів. Загальний опис

Перш за все необхідно перейти на сторінку пошуку рецептів. Зробити це можна трьома шляхами:

- натиснути кнопку «Рецепти» у головному меню;
- натиснути на кнопку бокового меню та обрати пункт «Пошук рецептів»;

- через пряме посилання.

Опис інтерфейсу сторінки пошуку рецептів (рисунок 4):

- секція для виставлення загальних обмежень (назва, тривалість, калорійність);
- секція створення та використання готових фільтрів;
- секція для виставлення обмежень за інгредієнтами;
- секція для виставлення обмежень за тегами;
- секція для перегляду обраних критеріїв пошуку;
- кнопки керування пошуком (кнопка для застосування фільтру, кнопка для перегляду усіх наявних рецептів, поле пошуку рецептів за назвою, кнопка обрання шляху сортування рецептів);
- список рецептів після фільтрації.

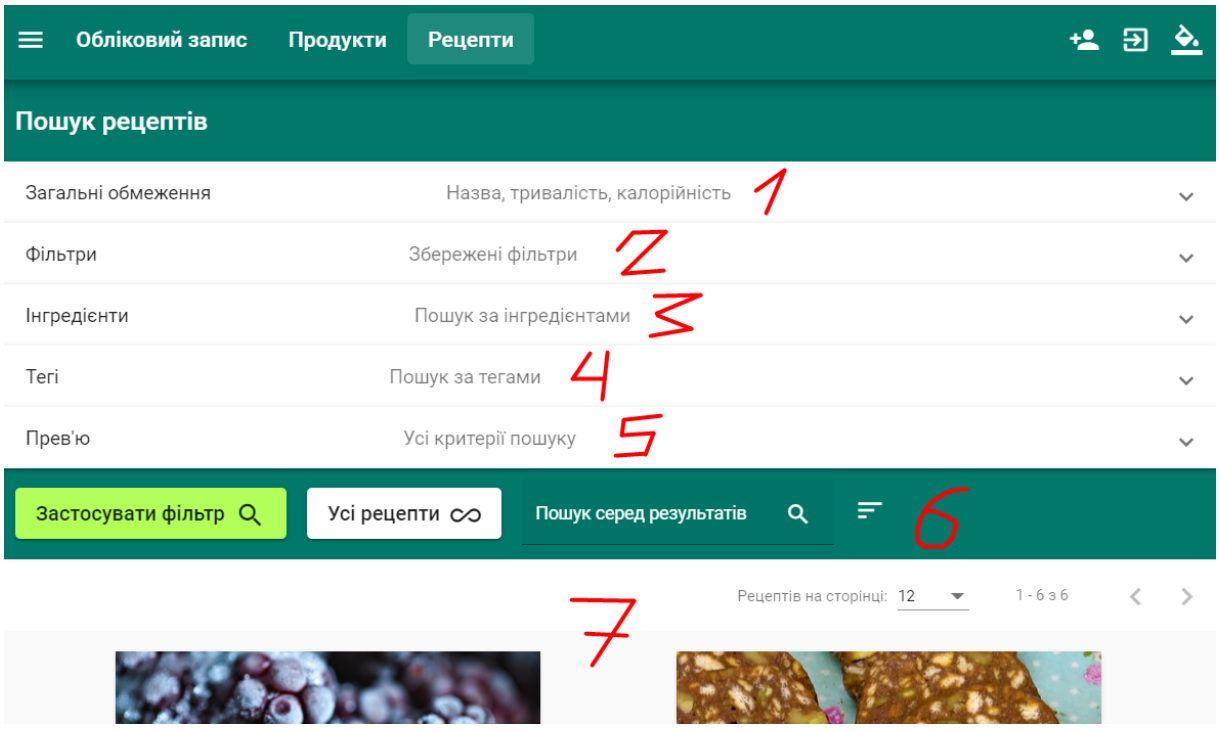


Рисунок 4 - Інтерфейс сторінки «Пошук рецептів»

Пошук рецептів. Секція «Загальні обмеження»

Список елементів секції :

а) «Назва рецепту» (рисунок 5 п.1) – поле для пошуку рецептів за частиною назви;

б) «Тривалість приготування» (рисунок 5 п.2) – поля для вводу обмежень (граничних значень) тривалості приготування страв у хвилинах (лише цілі числа);

в) «Калорійність» (рисунок 5 п.3) – поля для вводу обмежень (граничних значень) калорійності страв у ккал на 100 г готової страви (можна вводити дробові числа через крапку).

Для елементів б і в необхідно вводити граничні значення (включно), проте необов'язково заповнювати всі значення (достатньо не заповнювати взагалі або заповнити лише верхню чи нижню межі). Якщо дані некоректні (від'ємні числа, верхня межа менша за нижню, дробові числа для тривалості), то відповідні поля підсвічуються червоним кольором.

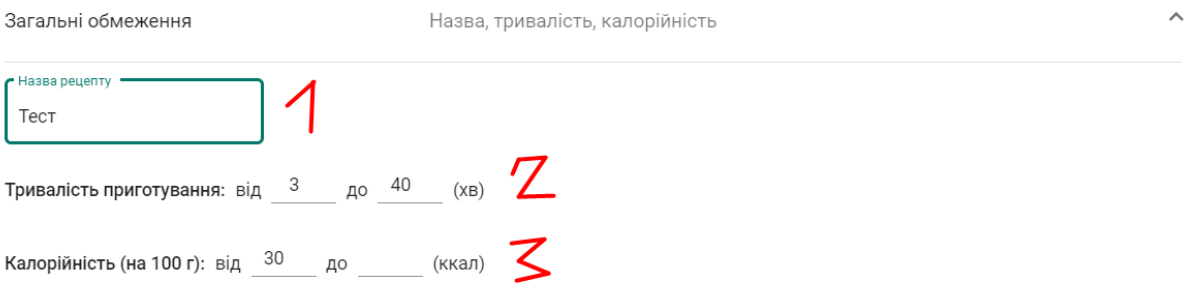


Рисунок 5 - Секція «Загальні обмеження»

Пошук рецептів. Секція «Фільтри»

Список елементів секції:

- кнопка вибору нового фільтру (рисунок 6 п.1);
- список стандартних фільтрів (рисунок 6 п.2);
- список власних фільтрів з кнопками для видалення фільтрів (рисунок 6 п.3);
- поле введення назви фільтру (доступне лише для авторизованих користувачів) (рисунок 6 п.4);
- кнопка збереження поточного фільтру (доступна лише для авторизованих користувачів; неактивна поки поле назви не буде заповнено) (рисунок 6 п.5);
- кнопка скидання фільтру (очищення усіх обраних критеріїв).

Для використання стандартного чи раніше збереженого фільтру за основу потрібно натиснути на його назву. Після цього усі інші секції заповняться критеріями обраного фільтру, їх можна буде змінити за бажанням та виконати пошук.

Для збереження поточних критеріїв пошуку у вигляді фільтру потрібно увійти в обліковий запис, ввести назву фільтру та натиснути кнопку «Зберегти поточний фільтр». Після цього новий фільтр з’явиться у списку вище та його можна буде видалити за допомогою кнопки видалення.

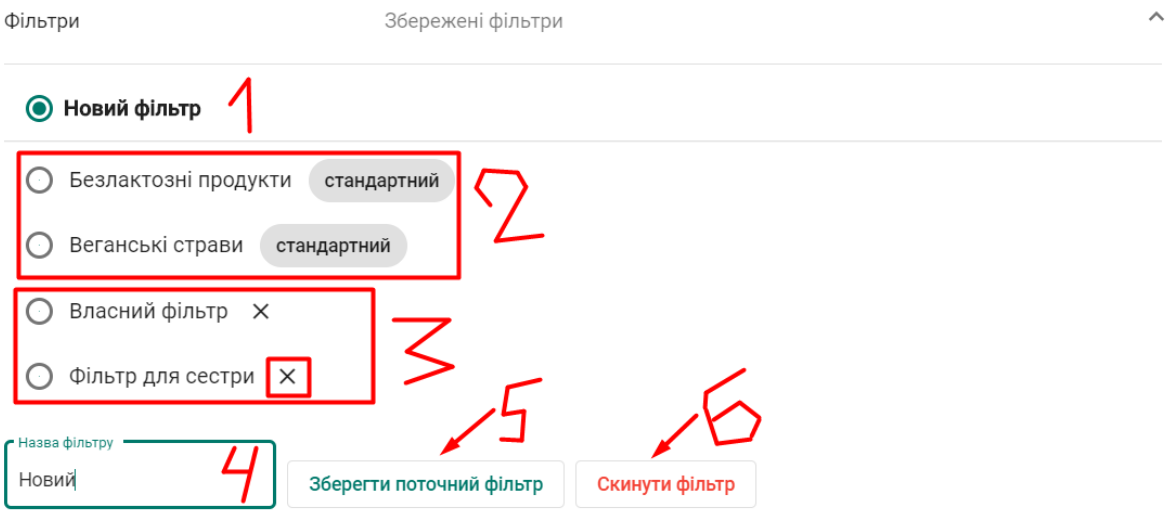


Рисунок 6 - Секція «Фільтри»

Пошук рецептів. Секція «Інгредієнти»

Список елементів секції:

а) «Шукати за наявними продуктами» – вибір типу пошуку за наявними продуктами (за замовчуванням, можна обирати обов’язкові та заборонені інгредієнти, проте після обрання цього типу – можна обирати наявні та обов’язкові інгредієнти) (рисунок 7 п.1);

б) «Шукати за точною назвою продуктів» – вибір типу пошуку за конкретними продуктами (за замовчуванням, пошук здійснюється з урахуванням також категорій та підкатегорій продуктів, проте тут це налаштування можна виключити) (рисунок 7 п.2);

в) таблиця зі списком продуктів для вибору (рисунок 7 п.3);

г) поле фільтрації продуктів у поточній таблиці за назвою (рисунок 7 п.4);

д) поле пошуку продукту за назвою та переходу на таблицю підкатегорій цього продукту (рисунок 7 п.5);

е) кнопка переходу на таблицю основних категорій (є таблицею за замовчуванням) (рисунок 7 п.6);

ж) кнопка переходу на таблицю з усіма продуктами (рисунок 7 п.7).

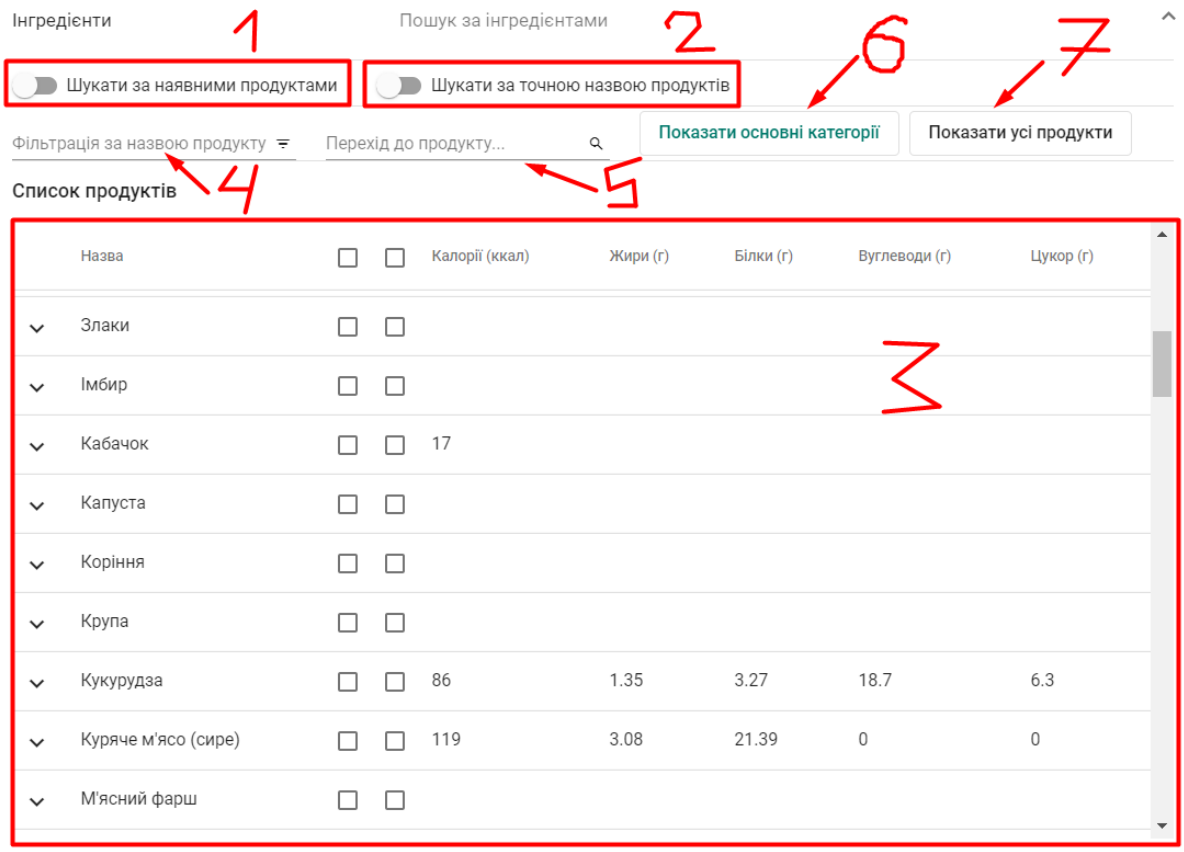


Рисунок 7 - Секція «Інгредієнти»

Список продуктів представлений у вигляді таблиці (рисунок 8), яка показує список основних категорій чи усіх продуктів, або список продуктів обраної категорії.

Для списку основних категорій чи усіх продуктів заголовок таблиці є лише загальною назвою «Список продуктів», проте для списку продуктів конкретної категорії він містить: кнопку повернення до попередньої таблиці (рисунок 8 п.1); назву поточної категорії (рисунок 8 п.2), при натисненні на яку з'являється вікно з детальним описом продукту; статус поточного

продукту/категорії (рисунок 8 п.3), який змінюється шляхом натискання на нього та має три значення (невизначений, обов'язковий, заборонений/наявний), підсвічені відповідними кольорами.

Таблиця складається з дев'яти колонок: колонка з кнопками переходу до таблиці підкатегорій продукту (рисунок 8 п.4) (лише для продуктів, що мають підкатегорії), колонка з назвами продуктів (рисунок 8 п.5) (при натисненні на неї з'являється вікно з детальним описом продукту), колонка вибору обов'язкових продуктів (рисунок 8 п.7), колонка вибору заборонених чи наявних продуктів (рисунок 8 п.8) (її колір та значення змінюються в залежності від типу пошуку – за наявними чи обов'язковими продуктами), колонки що містять інформацію про приблизний склад продукту на 100 грам (калорії, жири, білки, вуглеводи та цукор) (рисунок 8 п.6).

Для колонок вибору продуктів можна виділити або відмінити виділення усіх продуктів шляхом натискання кнопок в заголовку колонки. Оскільки одночасно продукт не може мати два статуси, то виділення однієї колонки автоматично відміняє видалення іншої.

Усі колонки окрім колонок вибору продуктів можна сортувати, для цього потрібно натиснути на стрілку біля назви колонки (рисунок 8 п.9).

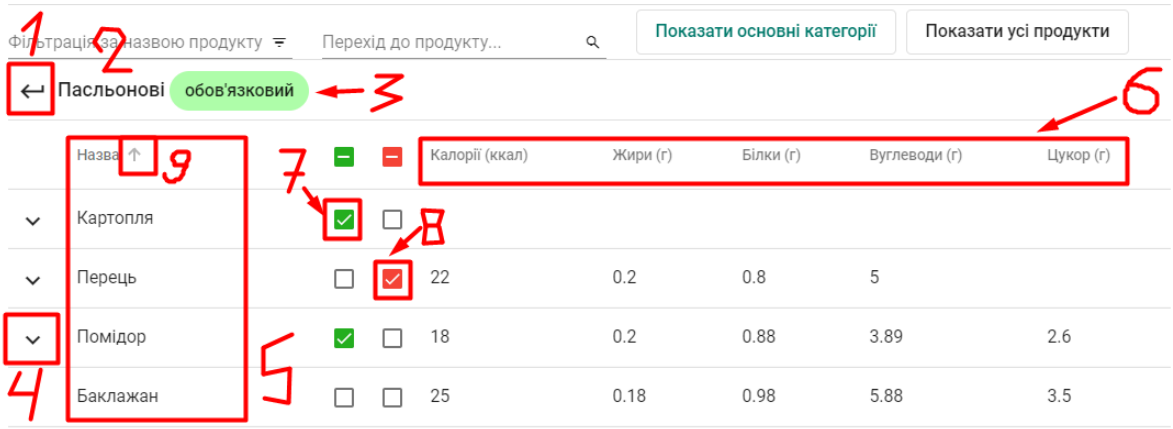


Рисунок 8 - Таблиця продуктів для пошуку за інгредієнтами

Пошук рецептів. Секція «Тегі»

Секція призначена для пошуку за тегами (ключовими словами) та має два поля вводу – для обов'язкових тегів та заборонених. Кожне з полів має список вже обраних словосполучень (рисунок 9 п.1), які виділені

відповідними кольорами. Додати тег до списку можна за допомогою вибору елементу у списку існуючих тегів (рисунок 9 п.2), або шляхом вводу власного та натискання клавіші Enter чи кнопки додавання (рисунок 9 п.3).

Для видалення тегу зі списку потрібно натиснути на хрестик біля назви (рисунок 9 п.4). Оскільки тег не може бути одночасно обов'язковим та забороненим, то при додаванні до одного списку він автоматично видаляється в іншому.

Рисунок 9 - Секція пошуку за тегами

Пошук рецептів. Секція «Прев'ю»

Секція призначена для перегляду усіх критеріїв пошуку, що були обрані в попередніх секціях. Вона не надає можливості додати критерії, проте дозволяє видалити деякі з них (інгредієнти, тегі, налаштування пошуку без категорій). Усі дані автоматично оновлюються під час здійснення змін на інших секціях.

Список елементів секції:

- частина назви рецепту (рисунок 10 п.1);
- межі тривалості приготування (рисунок 10 п.2);
- межі калорійності страви (рисунок 10 п.3);
- назва фільтру, обраного у якості основи для пошуку (рисунок 10 п.4);
- показник налаштування пошуку без категорій (рисунок 10 п.5);

- списки обов’язкових та наявних/заборонених продуктів в залежності від типу пошуку (список наявних продуктів автоматично містить список обов’язкових продуктів, проте без можливості видалити їх) (рисунок 10 п.6);
- списки обов’язкових та заборонених тегів (рисунок 10 п.7);
- кнопка здійснення пошуку за фільтром (рисунок 10 п.8);
- кнопка очищення всіх критеріїв (рисунок 10 п.9).

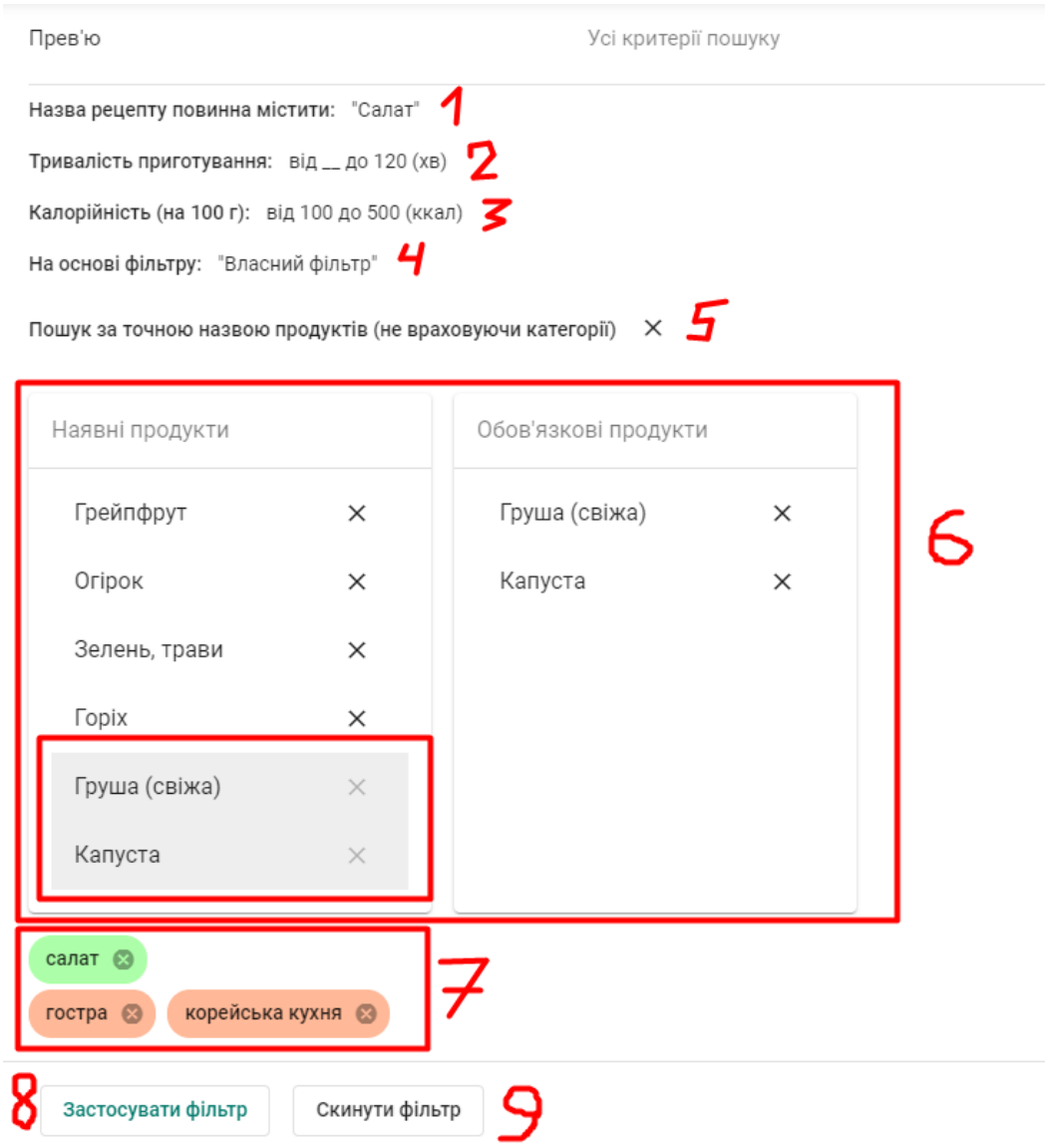


Рисунок 10 - Секція «Прев’ю»

Пошук рецептів. Кнопки керування пошуком

Для того щоб застосувати фільтр (здійснити пошук) без відкриття секції прев’ю, достатньо натиснути виділену кнопку «Застосувати фільтр» (рисунок 11 п.1).

Для того, щоб переглянути усі наявні рецепти, необхідно натиснути кнопку «Усі рецепти» (рисунок 11 п.2).

Поле «Пошук серед результатів» (рисунок 11 п.3) дозволяє здійснити швидкий пошук за назвою серед вже відфільтрованих рецептів (рецепти автоматично сортуються за назвою).

Також є можливість змінити стандартний тип сортування на інший за допомогою кнопки вибору типу сортування (рисунок 11 п.4). Після натиснення цієї кнопки з'являється список з чотирма варіантами сортування (рисунок 11 п.5): за назвою від А до Я, за назвою від Я до А, за датою створення від нових до старих, за датою створення від старих до нових.

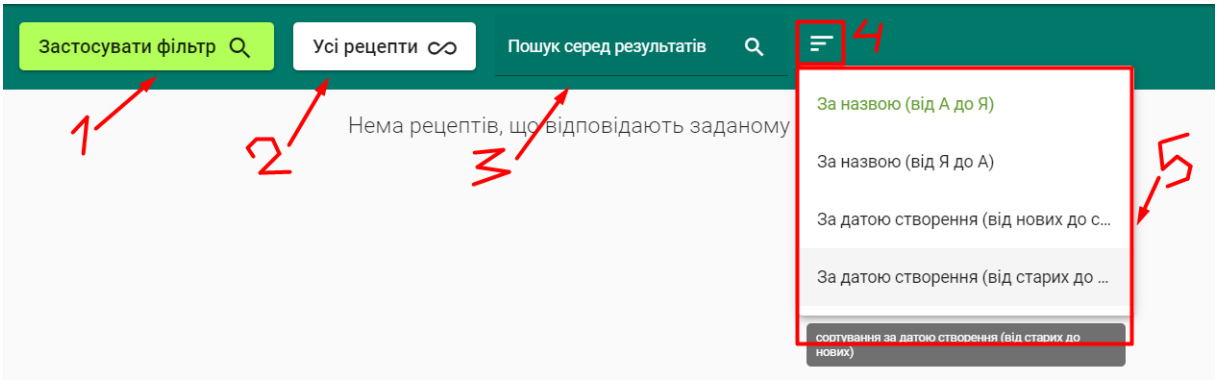


Рисунок 11 - Кнопки керуванням пошуку рецептів

Пошук рецептів. Список рецептів

Усі рецепти, що відповідають заданому пошуку, розподіляються на сторінки. Секції керування сторінками знаходяться перед та після списку рецептів (рисунок 12 п.1). Вони містять: кнопку вибору кількості рецептів на сторінці на основі визначеного списку (рисунок 12 п.2), номери рецептів поточної сторінки та кнопки переходу на попередню/наступну сторінку.

Кожен рецепт представлено у вигляді картки (рисунок 12 п.4), яка може складатися з наступних елементів:

- фотографія страви (рисунок 12 п.5);
- назва рецепту (рисунок 12 п.6);
- короткий опис рецепту (рисунок 12 п.7);
- дата та час створення рецепту (рисунок 12 п.8);

- кнопка переходу на сторінку детального опису рецепту (рисунок 12 п.9).

Є можливість швидкого перегляду детального опису рецепту без переходу на нову сторінку шляхом натискання на фотографію чи назву рецепту (відкриється вікно для відповідного рецепту) (рисунок 13).

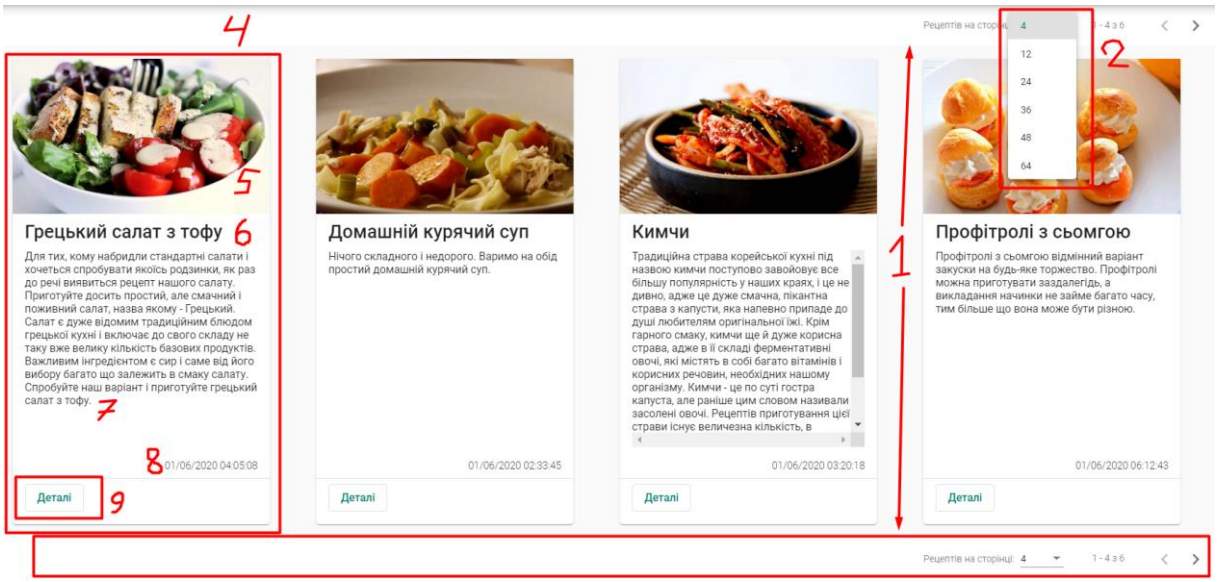


Рисунок 12 - Список рецептів

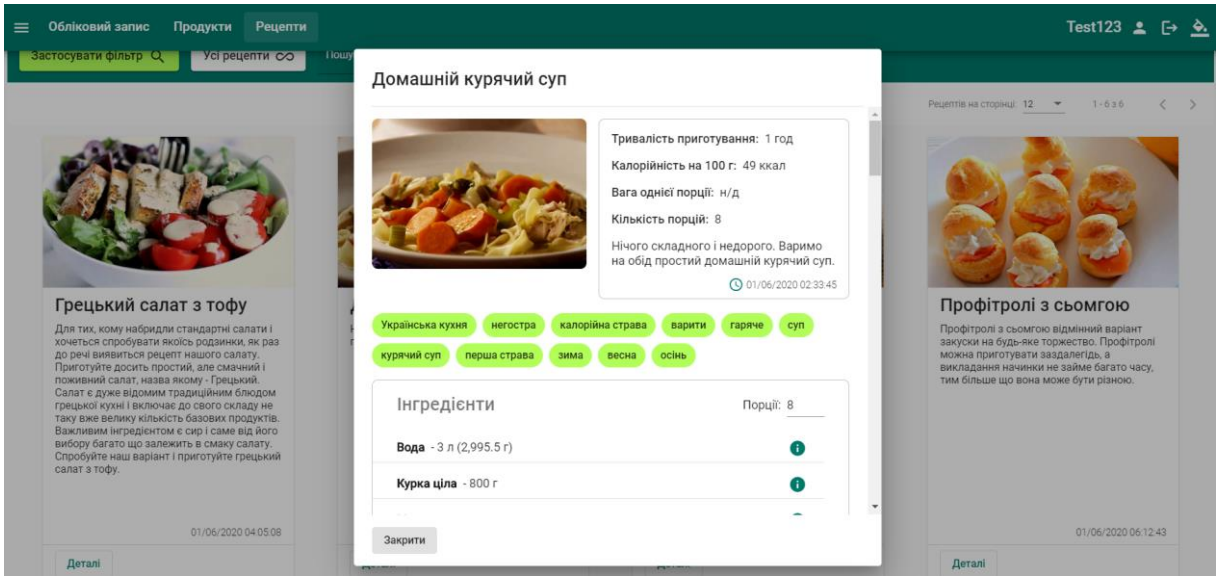


Рисунок 13 - Вікно швидкого перегляду рецепту

Перегляд детального опису рецепта

На сторінку детального опису рецепту можна перейти зі сторінки пошуку рецептів, сторінки керування рецептами чи за допомогою прямого посилання.

У заголовку сторінки перегляду рецепту знаходиться назва сторінки та кнопка повернення на попередню сторінку (рисунок 14 п.1). Під заголовком сторінки розміщена назва рецепту (рисунок 14 п.2).

Опис рецепту умовно можна розділити на три частини: опис основної інформації, список інгредієнтів та кроки приготування.

Опис основної інформації має наступні елементи (усі елементи можуть бути відсутні):

- фотографія страви (головне зображення рецепту) (рисунок 14 п.3);
- тривалість приготування (де зазначається кількість діб, годин та хвилин необхідних для приготування рецепту) (рисунок 14 п.4);
- кількість калорій у 100 г готової страви (рисунок 14 п.5);
- вага однієї порції у грамах (рисунок 14 п.6);
- кількість порцій за замовчуванням (рисунок 14 п.7);
- короткий опис рецепту (рисунок 14 п.8);
- ідентифікатор автору рецепта (рисунок 14 п.9);
- час створення рецепту (рисунок 14 п.10);
- список тегів (ключових слів), що мають відношення до рецепту (рисунок 14 п.11).

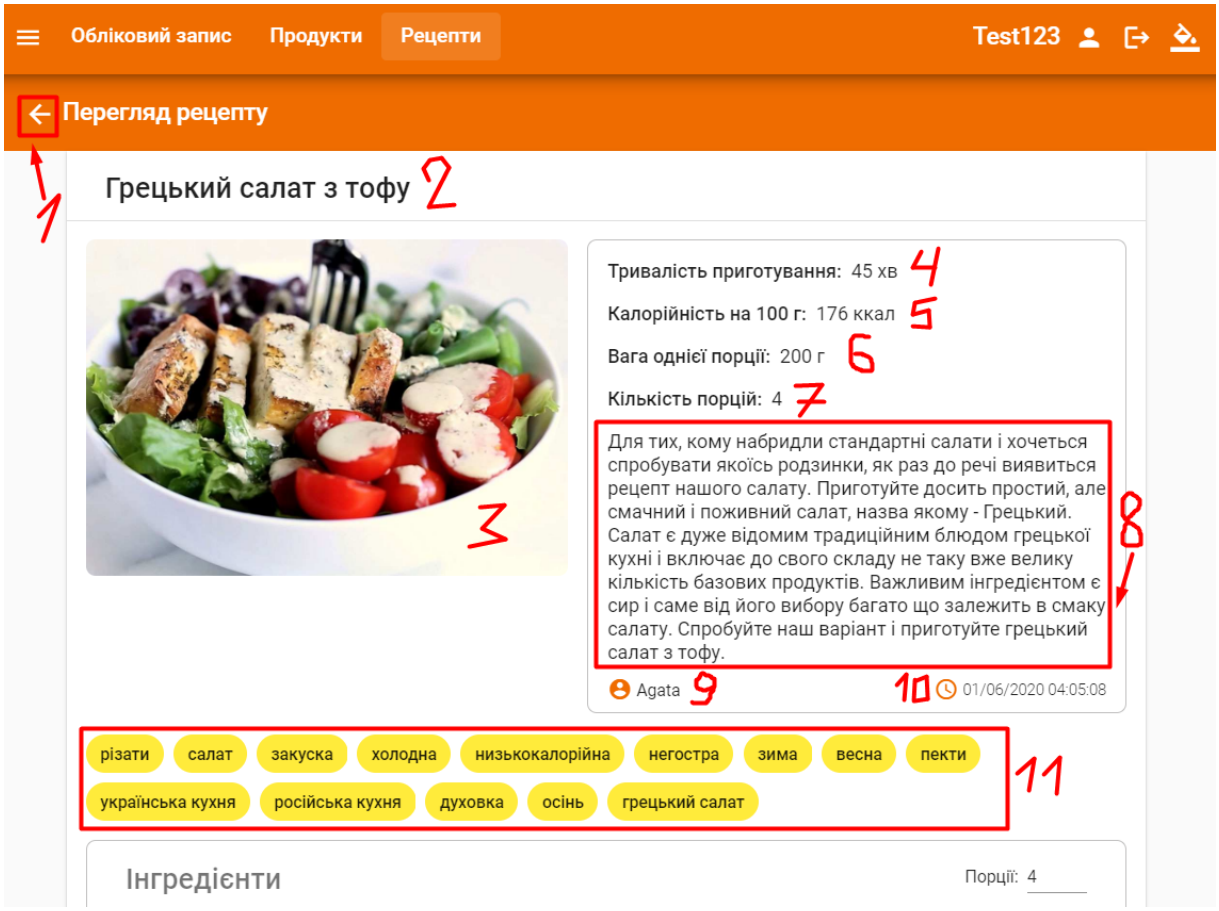


Рисунок 14 - Сторінка перегляду рецепту – основна інформація

Секція інгредієнтів має вигляд списку, який містить наступні елементи:

- назва продукту (рисунок 15 п.1);
- необхідна кількість продукту у вказаній одиниці виміру (рисунок 15 п.2);
- вага продукту у грамах (якщо вказано) (рисунок 15 п.3);
- кнопка відкриття вікна детального опису продукту (рисунок 15 п.4);
- позначення опціональних (необов'язкових) інгредієнтів (рисунок 15 п.1);
- примітка автора для інгредієнту (необов'язково) (рисунок 15 п.6).

Окрім цього, вкінці розміщена примітка автора до усього списку інгредієнтів (рисунок 15 п.7). Якщо у рецепті вказана кількість порцій, то з'являється поле її зміни (рисунок 15 п.8).

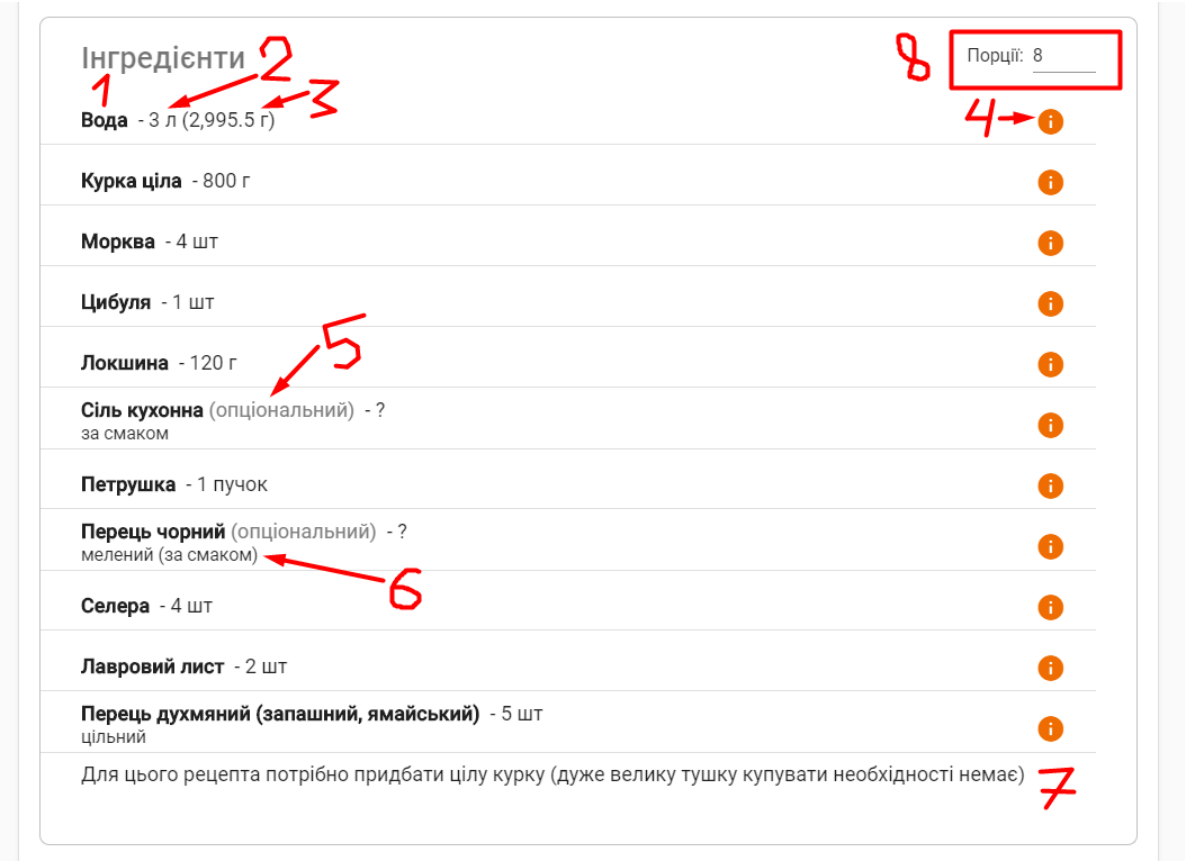


Рисунок 15 - Сторінка перегляду рецепту – список інгредієнтів

За замовчуванням, вага усіх інгредієнтів вказана для кількості порцій у описі рецепту. Проте можна побачити вагу продуктів для інших порцій, шляхом заповнення полю у списку інгредієнтів (рисунок 16 п.1). Після зміни порцій усі значення продуктів пропорційно змінюються теж та виділяються похилим шрифтом (рисунок 16 п.2) (шрифт стає нормальним коли кількість порцій повертається до початкового).

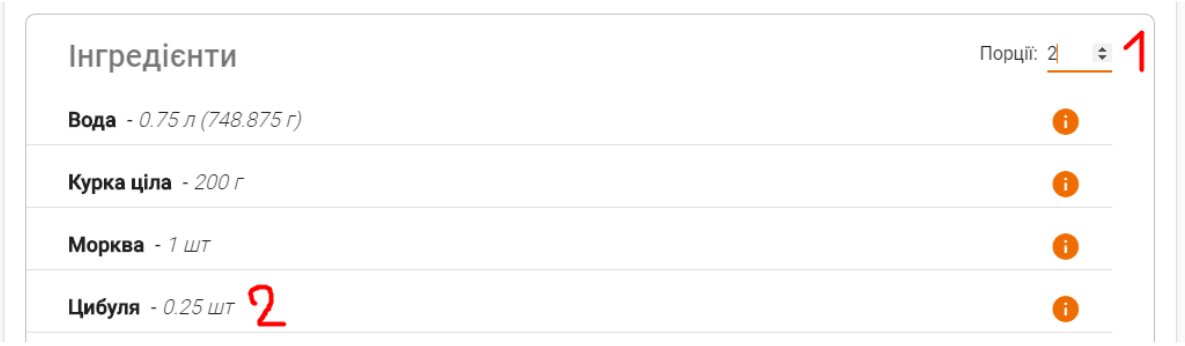


Рисунок 16 - Сторінка перегляду рецепту – зміна кількості порцій

Далі йде опис кроків приготування, які складаються з наступних елементів:

- фотографія процесу приготування (за наявності) (рисунок 17 п.1);
- назва кроку чи, за відсутності, його номер (рисунок 17 п.2);
- опис кроку приготування (за наявності) (рисунок 17 п.3).

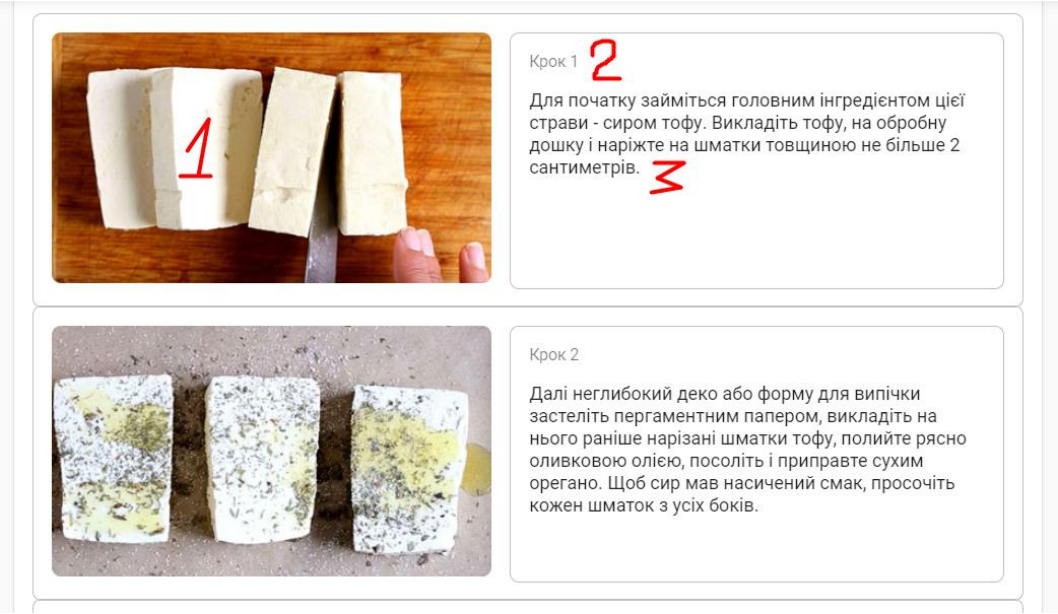


Рисунок 17 - Сторінка перегляду рецепту – кроки приготування

Перегляд списку продуктів

Перейти на сторінку списку продуктів можна трьома шляхами: натиснути кнопку «Продукти» у головному меню; натиснути на кнопку бокового меню та обрати пункт «Список продуктів»; через пряме посилання.

Першою сторінкою списку продуктів є сторінка, що містить список головних категорій та продуктів без категорій; вона складається з наступних елементів:

- заголовок сторінки (рисунок 18 п.1);
- картки продуктів (містять назву та зображення продукту) (рисунок 18 п.2);
- поле пошуку певного продукту серед усіх (рисунок 18 п.3).

Для переходу на сторінку конкретного продукту чи категорії необхідно натиснути на відповідну картку чи обрати назву продукту зі списку пошуку продуктів (рисунок 18 п.3).

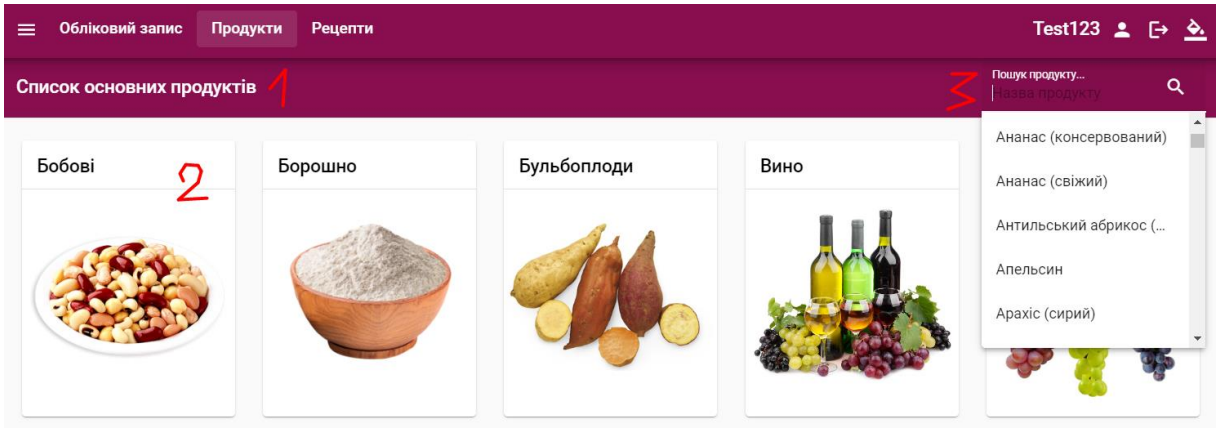


Рисунок 18 - Сторінка списку основних продуктів

Сторінка продукту містить наступні елементи:

- кнопка переходу на сторінку основних категорій (рисунок 19 п.1);
- кнопка повернення на попередню сторінку (рисунок 19 п.2);
- назва продукту (рисунок 19 п.3);
- «Деталі» (рисунок 19 п.4) – секція опису продукту;
- список продуктів (рисунок 19 п.5), що є підвидами поточного продукту (кожен продукт містить назву, зображення та список власних категорій (рисунок 19 п.6)).

Щоб перейти до сторінки категорії одного з продукту, достатньо натиснути на її назву на картці продукту.

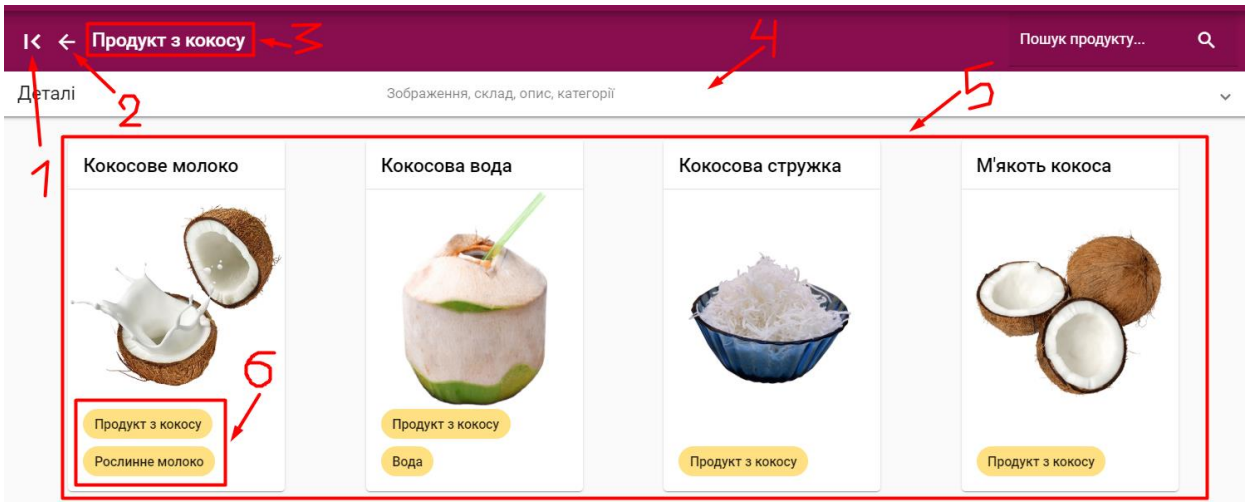


Рисунок 19 - Сторінка списку категорії продукту

Секція деталі містить наступні елементи:

- зображення продукту (рисунок 20 п.1);
- приблизний хімічний склад та харчова цінність на 100 грам продукту (калорійність, жири, білки, вуглеводи, вода, зола, цукор, крохмаль, холестерин, клітковина, транс жири) (рисунок 20 п.2);
- короткий опис продукту за наявності (рисунок 20 п.3);
- список категорій продукту (після натиснення на які відкривається сторінка цієї категорії) (рисунок 20 п.4).

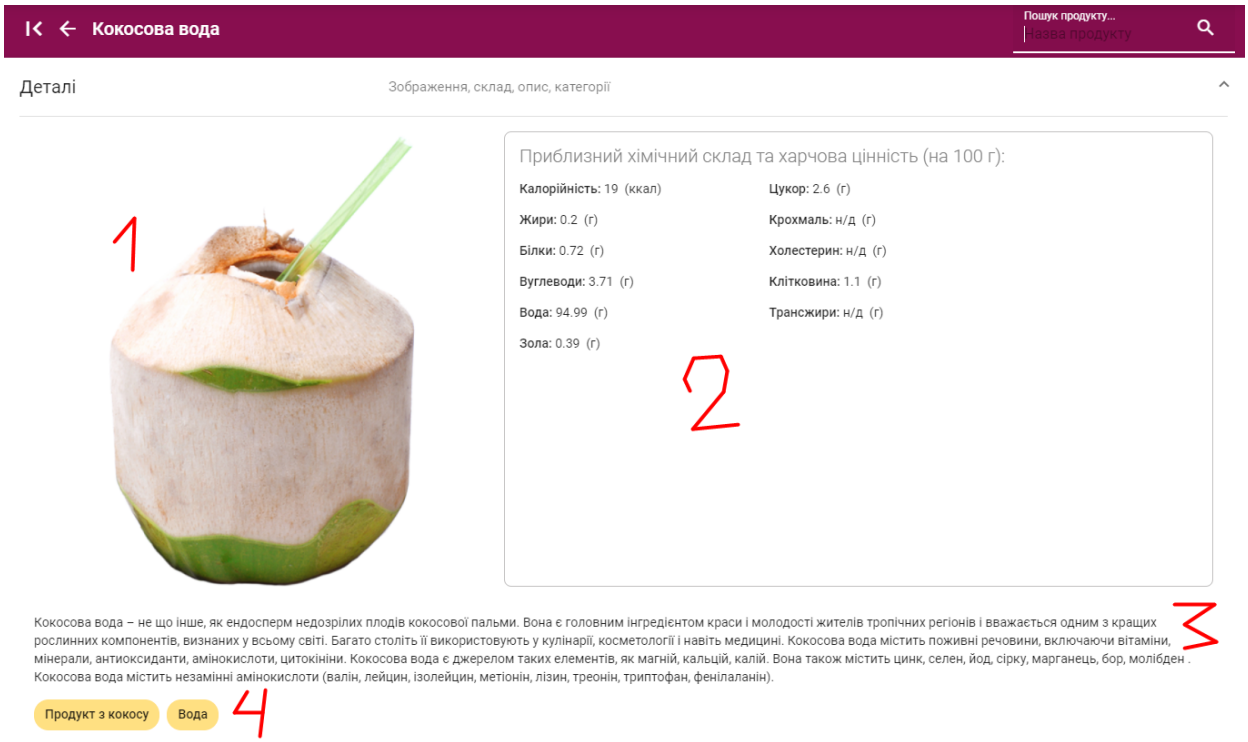


Рисунок 20 - Секція «Деталі» на сторінці продукту

Перегляд вікна опису продукту

Вікно швидкого перегляду продукту можна відкрити з секції інгредієнти сторінки пошуку рецептів та сторінки перегляду рецепту.

Вікно має наступні елементи:

- назва продукту (рисунок 21 п.1);
- зображення продукту (рисунок 21 п.2);
- приблизний хімічний склад та харчова цінність (рисунок 21 п.3);
- список категорій продукту (рисунок 21 п.4);
- список підкатегорій продукту (рисунок 21 п.5);

- кнопка закриття вікна (рисунок 21 п.6).

Після натискання на назву категорії чи підкатегорії відкривається вікно для цього продукту.

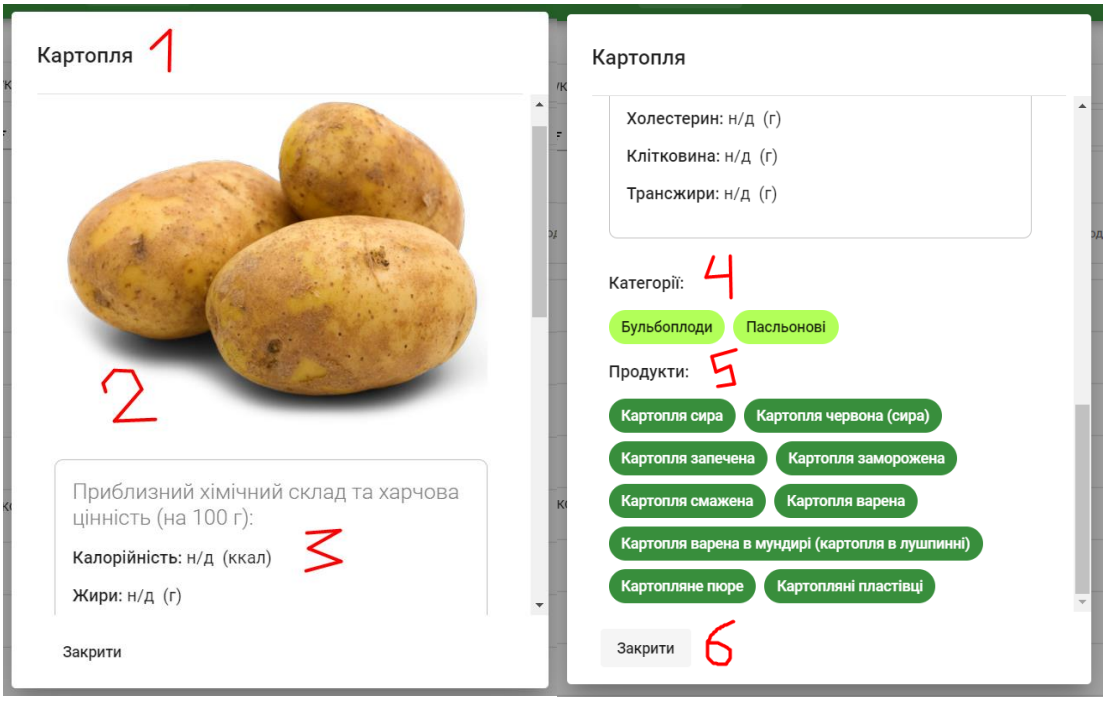


Рисунок 21 - Вікно швидкого перегляду продукту

Керування рецептами. Загальний опис

Для керування рецептами перш за все потрібно зареєструватися та увійти в обліковий запис. Перейти на сторінку керування рецептами можна трьома шляхами: через пряме посилання (перенаправить на сторінку 404 для неавторизованих користувачів); натиснути пункт «Керування рецептами» бокового меню; натиснути на кнопку «Рецепти» головного меню та обрати пункт «Керування рецептами» у відкритому підменю.

Сторінка керування містить список рецептів, створених поточним користувачем. Опис списку рецептів було надано під час опису сторінки пошуку рецептів, проте сторінка керування містить ще кнопку створення рецепту (рисунок 22 п.1), кнопки редагування (рисунок 22 п.2) та видалення (рисунок 22 п.3) рецепту на його картці.

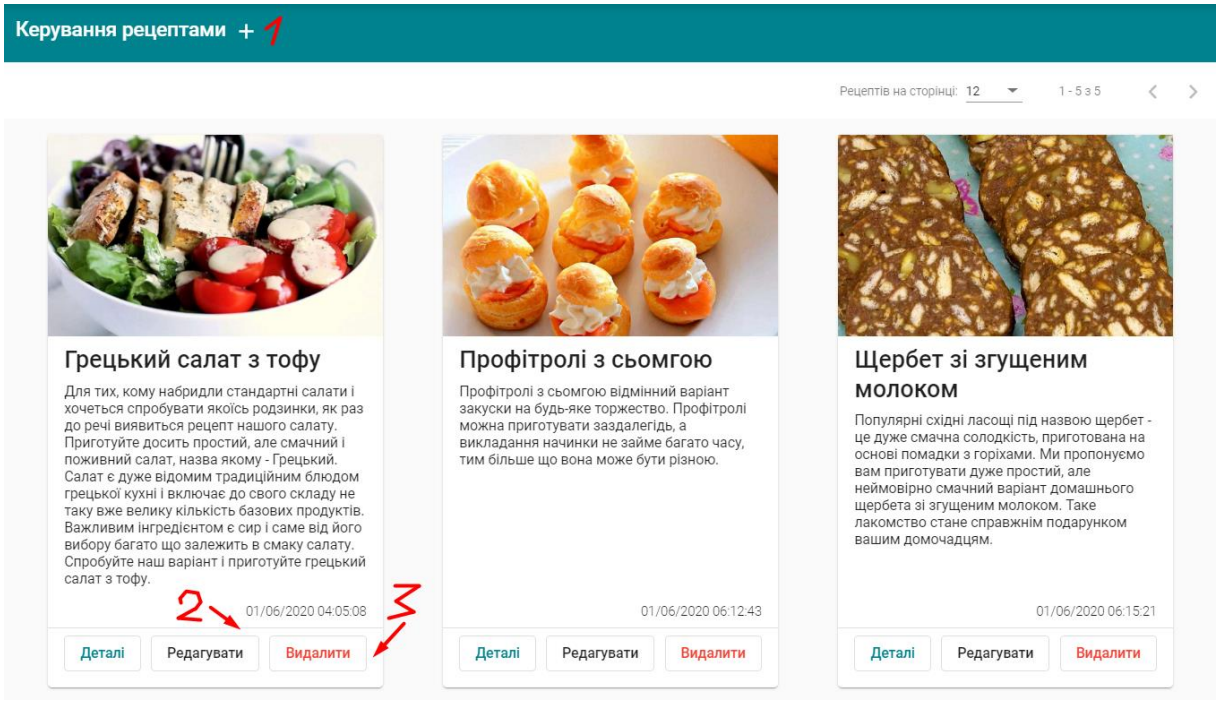


Рисунок 22 - Сторінка керування рецептами

Створення рецепту. Загальний опис

Перейти на сторінку створення рецепту можна через підпункт головного меню, пункт бокового меню та пряме посилання.

Інтерфейс сторінки створення рецепту містить наступні елементи:

- поле назви рецепту (рисунок 23 п.1);
- поле короткого опису рецепту (рисунок 23 п.2);
- поле кількості порцій (ціле число) (рисунок 23 п.3);
- поле ваги однієї порції у грамах (дробове число) (рисунок 23 п.4);
- поле часу приготування у хвилинах (ціле число) (рисунок 23 п.5);
- поле калорійності у ккал (дробове число) (рисунок 23 п.6);
- кнопка завантаження головного зображення (рисунок 23 п.7);
- поле додання тегів (з можливістю додання власних чи вибору тегів зі списку існуючих) (рисунок 23 п.8);
- кнопка додавання інгредієнтів (рисунок 23 п.9);
- поле примітки до списку інгредієнтів (рисунок 23 п.10);
- кнопка додавання кроку рецепту (рисунок 23 п.11);
- кнопка створення рецепту (стає доступною за умов заповнення усіх обов'язкових полів та відсутності помилок) (рисунок 23 п.12);

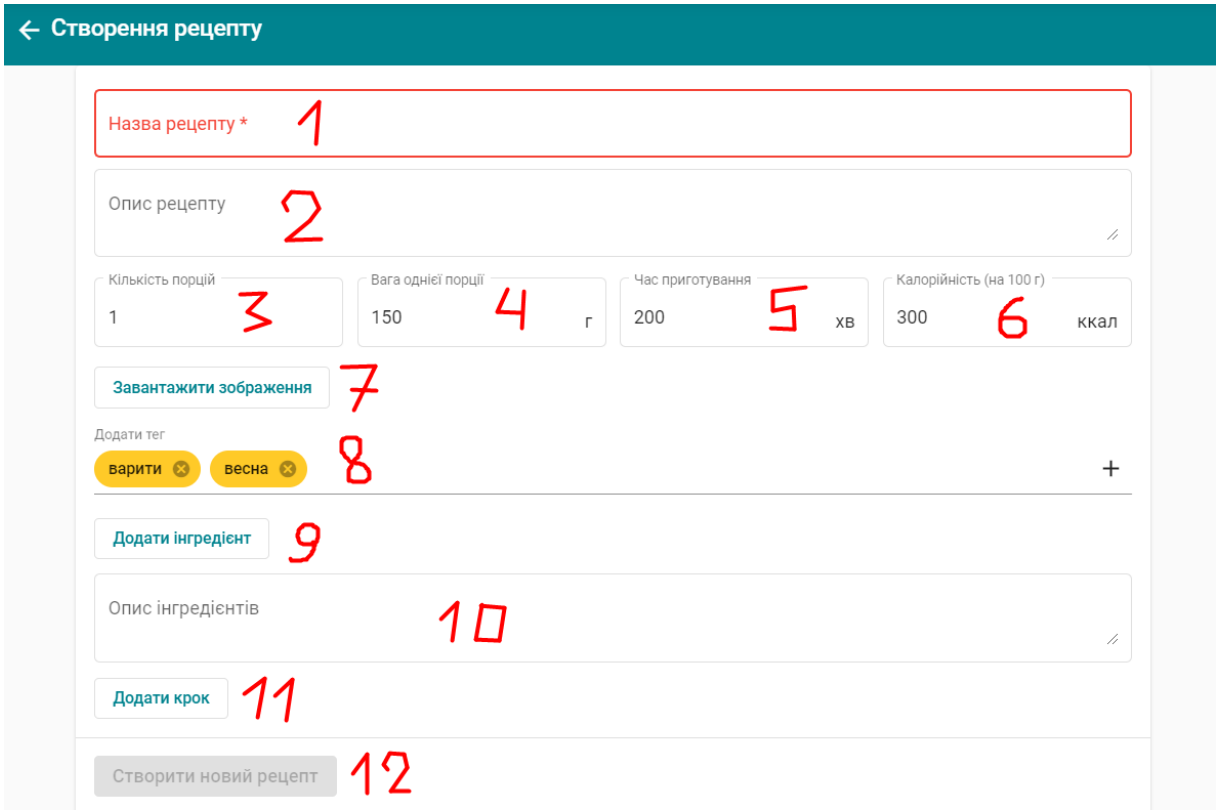


Рисунок 23 - Інтерфейс сторінки створення рецепту

Створення рецепту. Керування інгредієнтами

Для додавання інгредієнту до рецепту необхідно натиснути кнопку «Додати інгредієнт» (рисунок 24 п.1), після цього з’явиться картка створення інгредієнту. Для видалення інгредієнту необхідно натиснути кнопку «Видалити» (рисунок 24 п.2) на картці інгредієнту. Вибір продукту здійснюється через поле «Продукт», шляхом вводу частини назви та обрання елементу з відкритого списку (рисунок 24 п.3).

Картка додавання інгредієнту складається з наступних елементів:

- поле продукту (рисунок 25 п.1);
- поле кількості продукту (рисунок 25 п.2);
- поле одиниці виміру (можна самостійно ввести одиницю виміру або вибрати зі списку) (рисунок 25 п.3);
- поле ваги в грамах (якщо обрано іншу одиницю виміру) (рисунок 25 п.4);
- поле примітки до інгредієнта (рисунок 25 п.5);
- позначення обов’язковості інгредієнту (рисунок 25 п.6);

- кнопка швидкого перегляду продукту (доступна лише після його обрання) (рисунок 25 п.7).

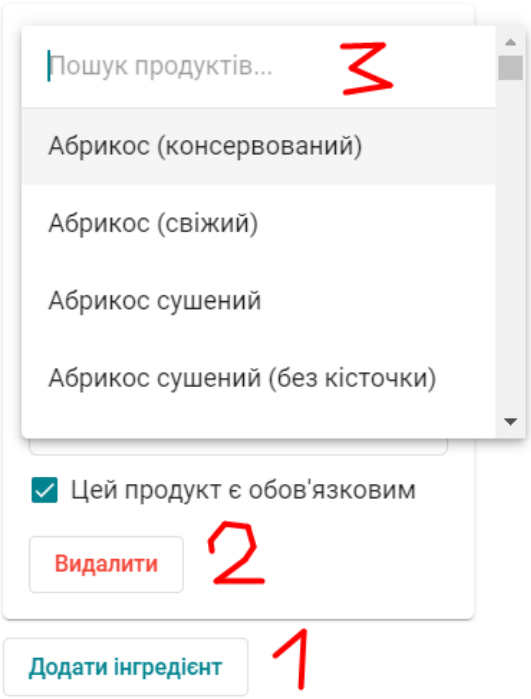


Рисунок 24 - Картка додавання інгредієнту (до заповнення)

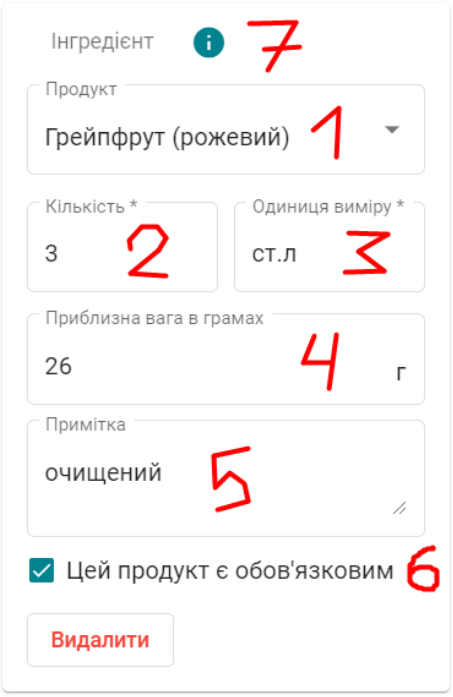


Рисунок 25 - Картка додавання інгредієнту (після заповнення)

Створення рецепту. Керування кроками приготування

Для додавання кроку до рецепту необхідно натиснути кнопку «Додати крок», після цього з’явиться картка створення кроку приготування. Для

видалення кроку необхідно натиснути кнопку «Видалити» (рисунок 26 п.6) на картці кроку.

Картка кроку рецепту містить наступні поля:

- поле назви кроку (рисунок 26 п.1);
- поле опису кроку (рисунок 26 п.2);
- кнопка завантаження зображення (рисунок 26 п.3);
- прев'ю завантаженого зображення (рисунок 26 п.4);
- кнопка видалення зображення (рисунок 26 п.5);
- кнопка видалення кроку (рисунок 26 п.6).

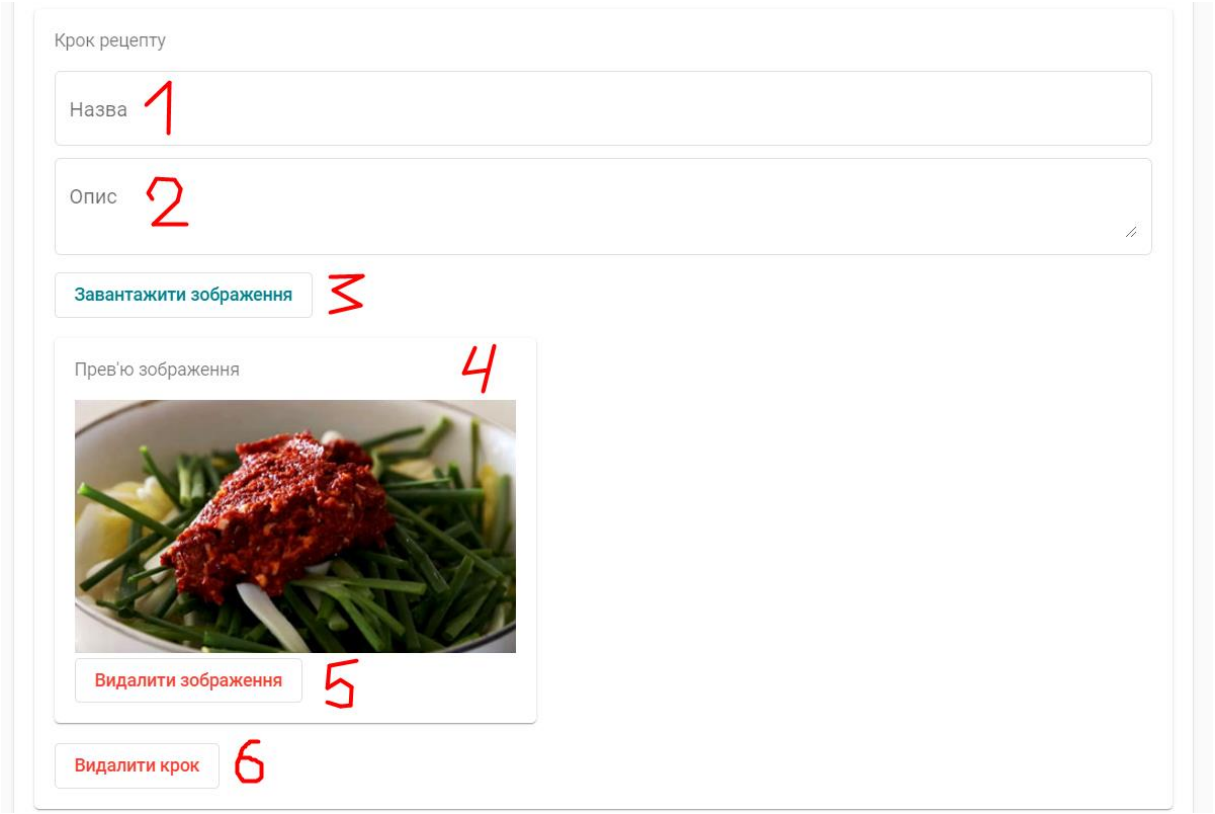


Рисунок 26 - Картка додавання кроку приготування

Редагування рецепту

Після створення рецепту сторінка автоматично перетворюється на сторінку редагування створеного рецепту. Також можна перейти нас сторінку редагування, натиснувши кнопку «Редагувати» на сторінці керування рецептами.

Сторінка редагування рецепту має такі ж елементи як і сторінка створення, проте має дві кнопки керування: кнопка збереження (оновлення)

(рисунок 27 п.1) та кнопка створення нового рецепту (використовуючи дані існуючого) (рисунок 27 п.2).

На відміну від сторінки створення кнопка біля заголовку (рисунок 27 п.3) не створює новий рецепт, а зберігає зміни поточного.

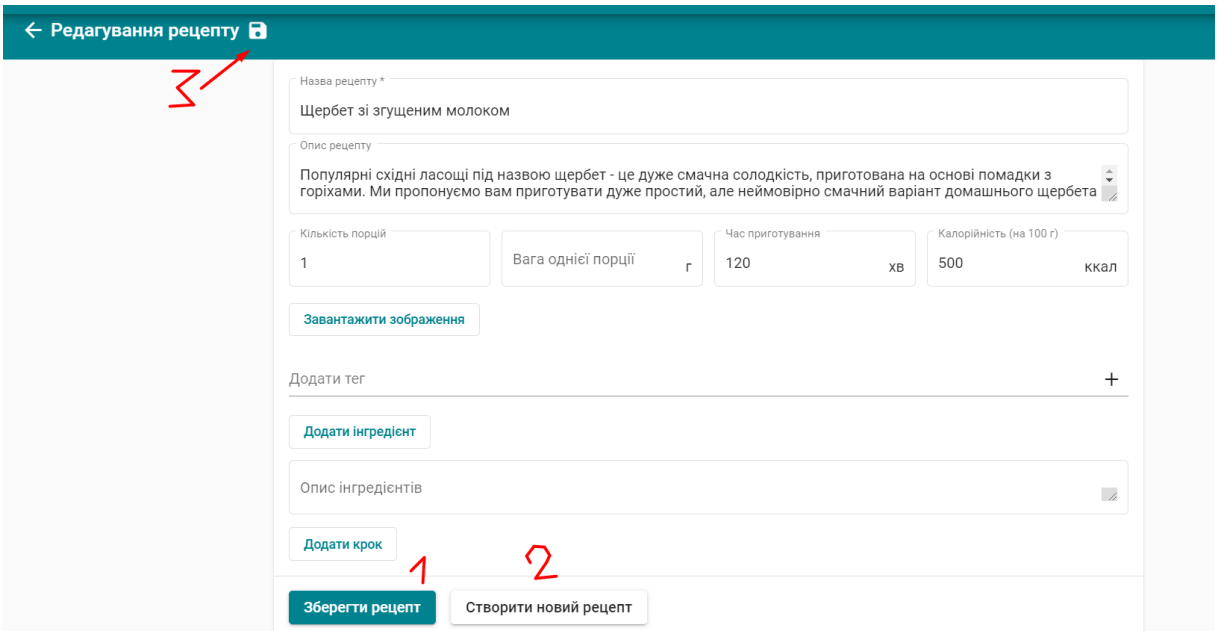


Рисунок 27 - Сторінка редагування рецепту

Видалення рецепту

Кроки видалення рецепту:

а) перейти на сторінку керування;

б) обрати рецепт та натиснути кнопку видалення;

в) підтвердити видалення у вікні підтвердження (рисунок 28);

г) отримати повідомлення про успішне видалення.

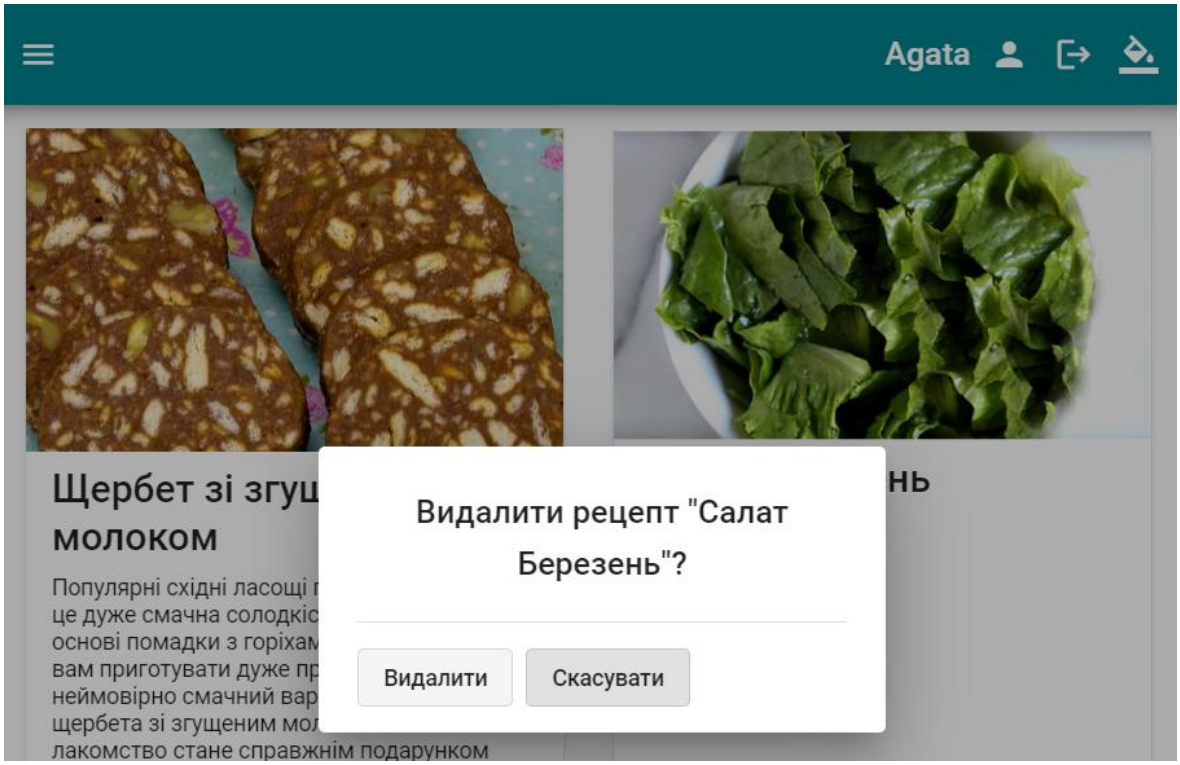


Рисунок 28 - Вікно підтвердження видалення рецепту

Реєстрація користувача

Перейти на сторінку реєстрації можна наступними шляхами: натиснути пункт бокового меню «Реєстрація»; через підменю пункту «Обліковий запис» головного меню (рисунок 29 п.1); за допомогою іконки реєстрації на верхній панелі (доступна до входу в обліковий запис) (рисунок 29 п.2).

Для реєстрації, перш за все, потрібно ввести логін (ідентифікатор користувача) у відповідне поле (рисунок 29 п.3). Після заповнення логіну автоматично посилається запит на сервер для перевірки його унікальності. Якщо користувач з таким логіном вже існує, то виводиться відповідне повідомлення (рисунок 29 п.4).

Поля електронної пошти (рисунок 29 п.5) та повного ім’я (рисунок 29 п.6) є не обов’язковими для заповнення.

Далі йдуть поля паролю (рисунок 29 п.7) та його підтвердження (рисунок 29 п.8). Поле підтвердження паролю стає активним тільки після заповнення паролю коректними даними. Якщо після вводу паролі не співпадають, то виводиться відповідне повідомлення (рисунок 29 п.9). За

замовчуванням, пароль має вигляд крапок, проте можна подивитися вихідні символи за допомогою кнопки бачення (рисунок 29 п.10).

Кнопка реєстрації (рисунок 29 п.11) стає доступною тільки, якщо всі обов'язкові поля заповнені коректними даними. Після натиснення кнопки реєстрація буде виведено повідомлення про успішну реєстрацію.

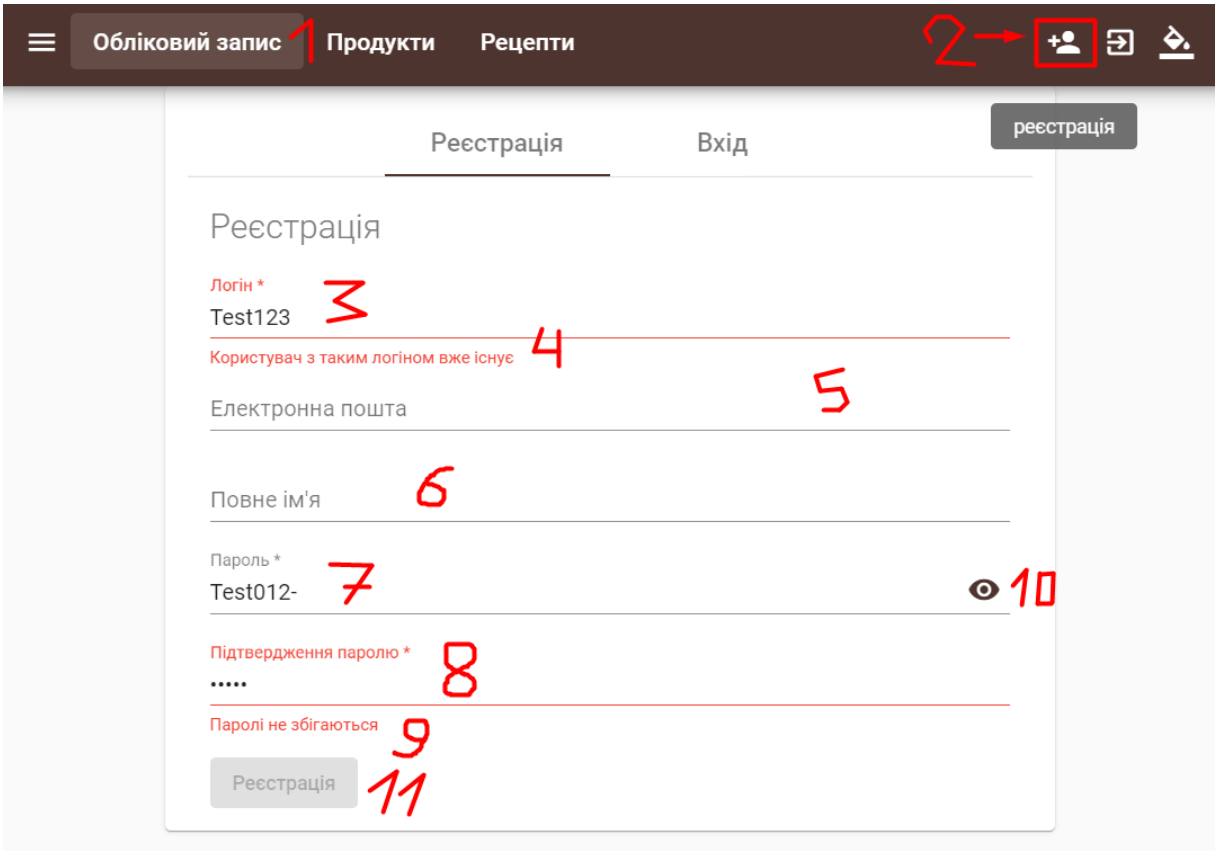


Рисунок 29 - Вікно реєстрації користувача (некоректні дані)

Для того, щоб зареєструвати адміністратора необхідно щоб інший адміністратор увійшов у систему, перейшов на сторінку реєстрації, заповнив поля та натиснув кнопку «Реєстрація адміністратора» (рисунок 30).

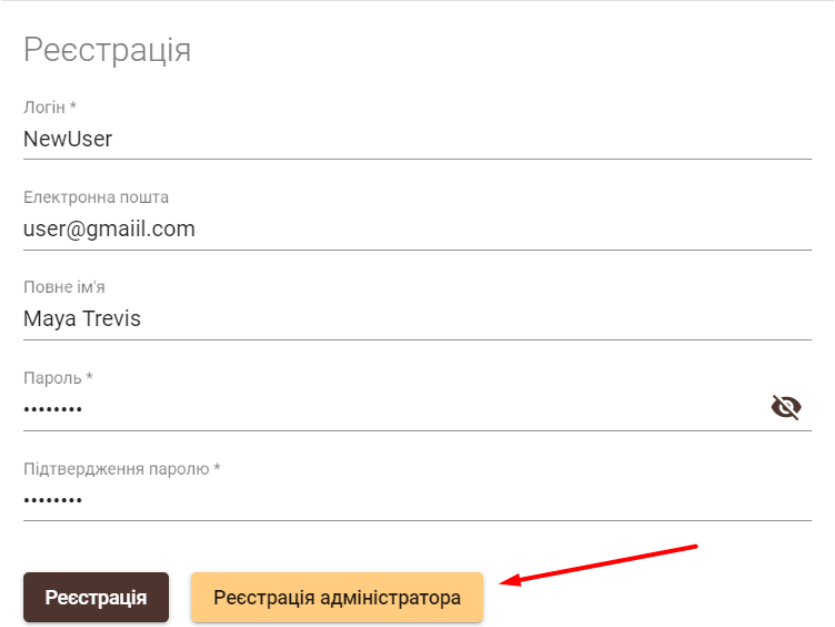


Рисунок 30 - Вікно реєстрації адміністратора

Вхід в обліковий запис

Перейти на сторінку входу в обліковий запис можна через головне меню, бокове меню, шляхом переключення зі сторінок реєстрації та зміни паролю (рисунок 31 п.1), через кнопку входу на верхній панелі (рисунок 31 п.2).

Для входу необхідно правильно заповнити поля логіну та паролю (рисунок 31 п.3-4), та натиснути кнопку «Вхід» (рисунок 31 п.5).

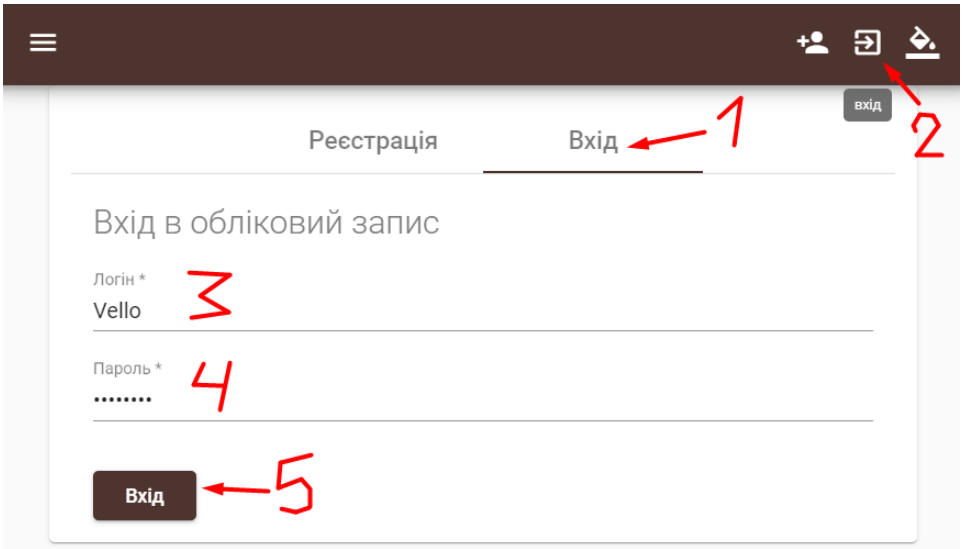


Рисунок 31 - ' входу в обліковий запис

Якщо дані були введені вірно, то з'являється повідомлення про успішний вхід в систему. Кнопку реєстрації та входу на верхній панелі

замінюються ім’ям користувача (рисунок 32 п.1) та кнопкою виходу з облікового запису (рисунок 32 п.2). В меню з’являється чотири-п’ять додаткових опцій: створити продукт (тільки для адміністратора), керувати рецептами, створити рецепт, змінити пароль та вихід (рисунок 32 п.3).

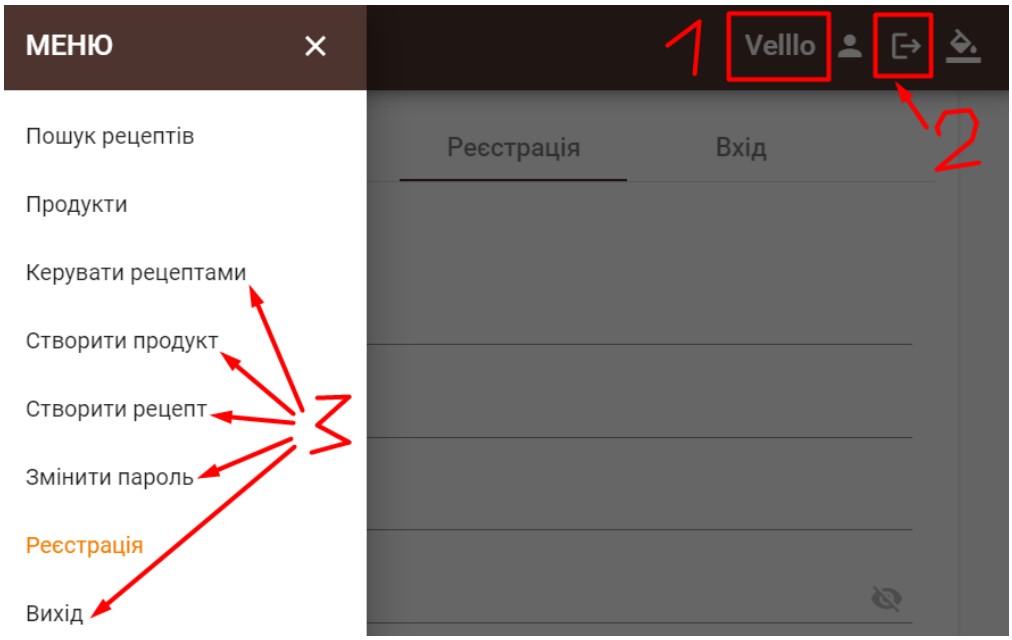


Рисунок 32 - Вхід в обліковий запис

Вихід з облікового запису

Коли буде вичерпано час існування токену та буде виконано запит, потребуючий авторизації, тоді система виведе повідомлення про помилку та автоматично здійснить вихід з облікового запису. Для самостійного виходу потрібно натиснути кнопку виходу на верхній панелі або обрати відповідну опцію головного чи бокового меню.

Зміна паролю

Для зміни паролю потрібно перейти на відповідну сторінку через меню або через сторінки реєстрації та входу. Змінити можна лише пароль поточного користувача. Тож, неможливо змінити пароль без входу в обліковий запис.

Вікно зміни паролю має наступні поля:

- логін користувача (рисунок 33 п.1);
- старий пароль (має кнопку бачення) (рисунок 33 п.2);
- новий пароль (має кнопку бачення) (рисунок 33 п.3);

- підтвердження нового паролю (рисунок 33 п.4);
- кнопка зміни паролю (рисунок 33 п.5).

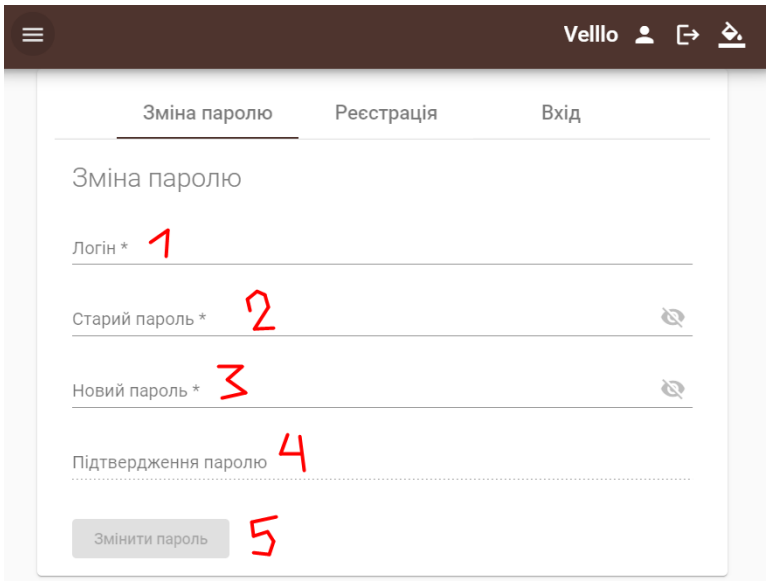


Рисунок 33 - Вікно зміни паролю

Керування продуктами

Керування продуктами доступно лише адміністраторам. Для адміністратора на сторінках продуктів з’являються додаткові елементи:

- кнопка редагування категорії (рисунок 34 п.1);
- кнопка видалення категорії (рисунок 33 п.2);
- кнопка створення нового продукту в категорії (рисунок 33 п.3);
- кнопка редагування обраного продукту (рисунок 33 п.4);
- кнопка видалення обраного продукту (рисунок 33 п.5).

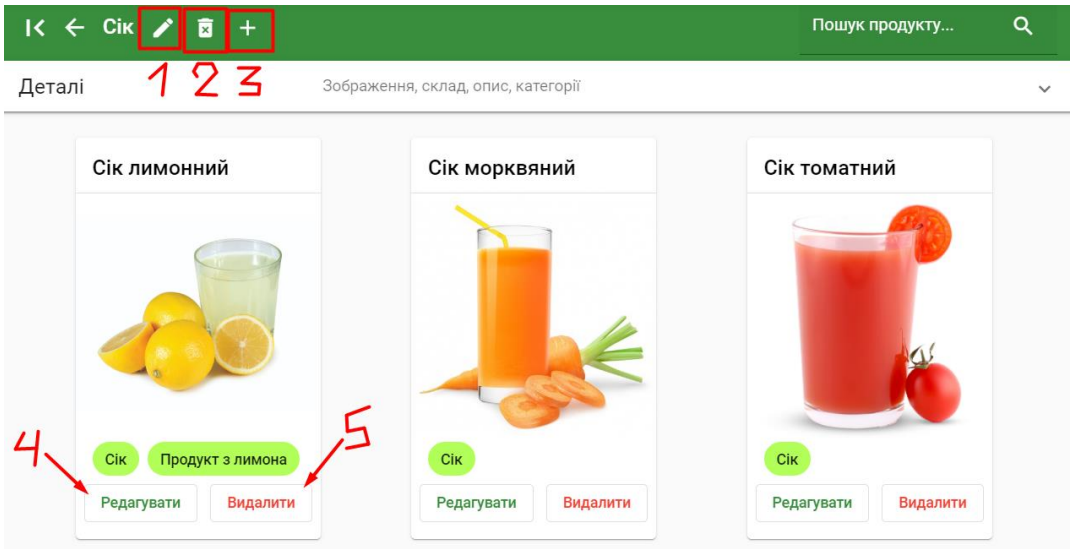


Рисунок 34 - Сторінка продуктів для адміністратора

Створення продукту

Перейти на сторінку створення продуктів можна через відповідні пункти меню, пряме посилання, а також шляхом натискання кнопки додавання на сторінці продуктів (у такому випадку в категоріях продукту автоматично з’явиться продукт сторінки переходу).

Сторінка створення продукту складається з наступних елементів:

- кнопка повернення до попередньої сторінки продуктів (рисунок 35 п.1);
- кнопка перегляду прев’ю поточного продукту (рисунок 35 п.2);
- кнопка швидкого збереження продукту (рисунок 35 п.3);
- поле назви продукту (обов’язкове) (рисунок 35 п.4);
- поле короткого опису продукту (рисунок 35 п.5);
- кнопка завантаження зображення продукту (рисунок 35 п.6);
- прев’ю завантаженого зображення (рисунок 35 п.7);
- кнопка видалення зображення (рисунок 35 п.8);
- список категорій продукту (рисунок 36 п.9);
- список підкатегорій продукту (рисунок 36 п.10);
- поля складників продукту у грамах (дробові числа від 0 до 100, калорійність – без обмеження) (рисунок 36 п.11);
- кнопка створення рецепту (рисунок 36 п.12).

Для зручного додавання категорій/підкатегорій використовується список усіх продуктів з багатьма виборами (рисунок 37). Продукт не може бути одночасно й категорією й підкатегорією одного продукту, тому виділення продукту в одному списку автоматично знімає виділення в іншому.

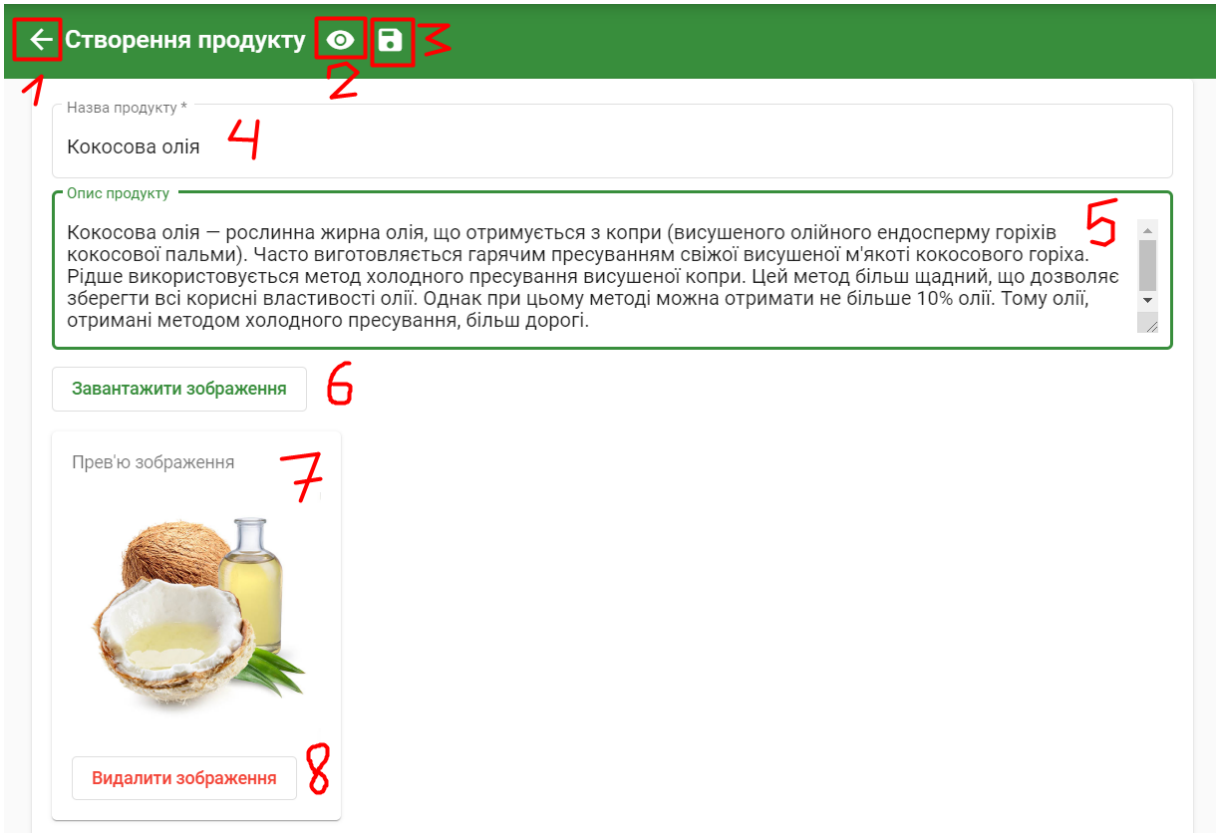


Рисунок 35 - Сторінка створення продукту (верхня частина)

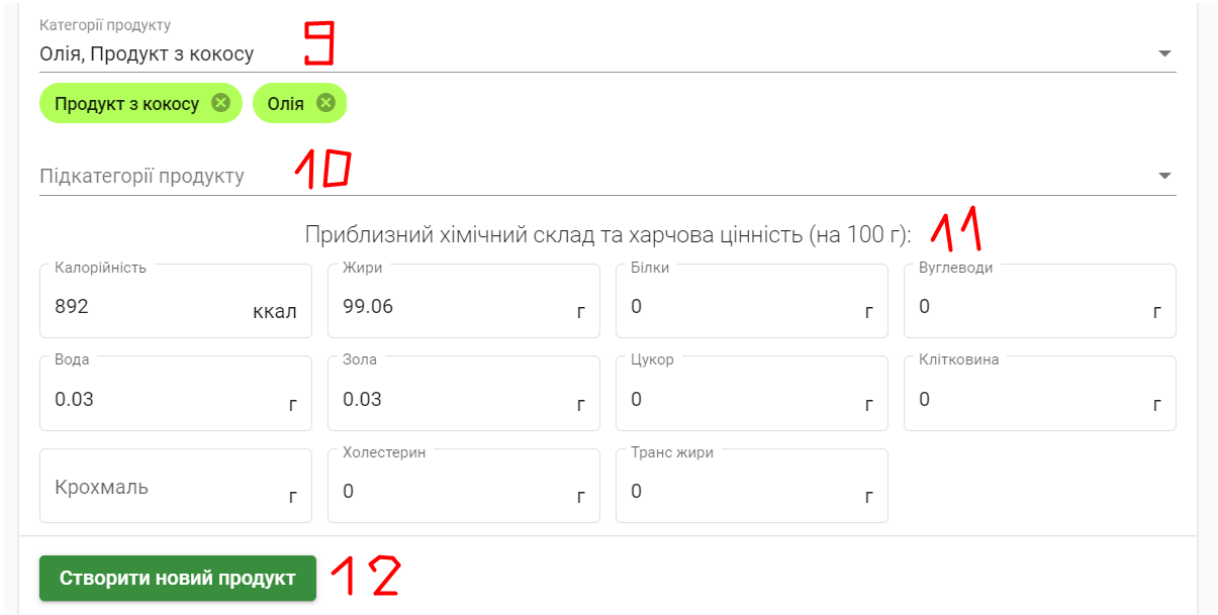


Рисунок 36 - Сторінка створення продукту (нижня частина)

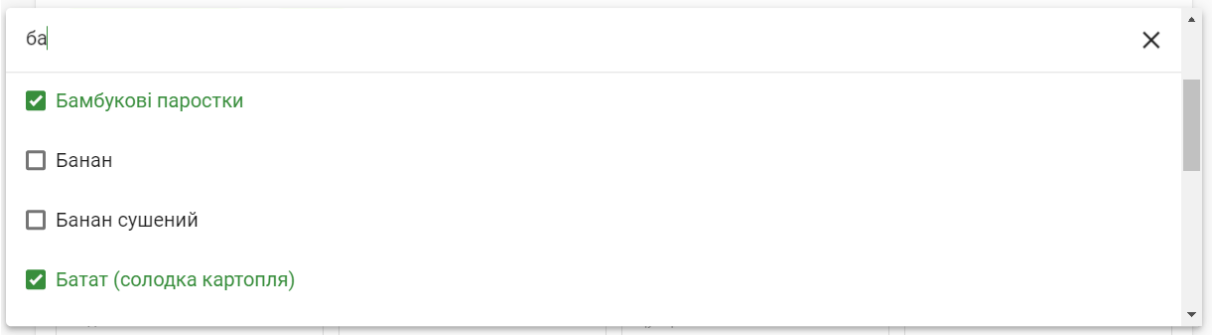


Рисунок 37 - Список вибору продуктів/категорій

Редагування продукту

Сторінка редагування продукту відкривається автоматично після створення продукту. Для того щоб самостійно перейти на сторінку редагування треба використати пряме посилання, кнопку «Редагувати» на картці продукту або кнопку редагування у заголовку сторінки продукту.

Сторінка редагування має всі ті ж самі елементи, що й сторінка створення, проте керуючих кнопок дві: кнопка оновлення продукту (основна) та кнопка створення нового продукту на основі цього.

Видалення продукту

Видалити продукт можна двома шляхами: натиснути кнопку «Видалити» на картці продукту, або натиснути кнопку видалення у заголовку сторінки продукту. Також необхідно підтвердити намір видалити продукт у вікні підтвердження (рисунок 38). Якщо продукт було успішно видалено, то він зникне з екрану, та з’явиться повідомлення про успішне видалення (рисунок 39).

Рисунок 38 - Підтвердження видалення продукту

Рисунок 39 - Повідомлення про видалення продукту

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «КУЛІНАРНИЙ ПОМІЧНИК»

Опис програми

КПІ.ІП-6104.045440.04.13

“ПОГОДЖЕНО”
Керівник проєкту:
К.І. Ліщук

Виконавець:
В.І. Брідня

Нормоконтроль:
К.І. Ліщук

Київ – 2020 року

Тексти програмного коду

Веб-застосування «Кулінарний помічник»

(Найменування програми (документа))

DVD-R

(Вид носія даних)

45 арк, 210 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

СЕРВЕРНА ЧАСТИНА

```
    public class GenericRepository<TEntity> : IRepository<TEntity>
        where TEntity : Entity
    {
        protected readonly DbSet<TEntity> _entities;
        protected readonly CulinaryContext _context;

        public GenericRepository(CulinaryContext context)
        {
            _context = context;
            _entities = context?.Set<TEntity>()
                ?? throw new ArgumentException($"Context must be not null!");
        }
        public virtual bool Contains(long id)
            => _entities.Any(x => x.Id == id);

        public virtual void Delete(TEntity entity)
        {
            _entities.Remove(entity
                  ?? throw new ArgumentNullException("Entity to remove must be not
null!"));
            _context.SaveChanges();
        }
        public virtual void DeleteById(long id)
            => Delete(GetById(id));

        public virtual TEntity GetById(long id)
            => _entities.Find(id);

        public virtual long Add(TEntity entity)
        {
            var result = _entities.Add(entity
                ?? throw new ArgumentNullException("Entity to add must be not
null!"));
            _context.SaveChanges();
            return result.Entity.Id;
        }
        public virtual bool Update(TEntity entity)
        {
            if (entity == null || !_entities.Any(x => x.Id == entity.Id))
            {
                return false;
            }
            _entities.Update(entity);
            _context.SaveChanges();
            return true;
        }
        public virtual ICollection<TEntity> GetAll()
            => _entities.ToHashSet();

        public virtual void AddRange(ICollection<TEntity> entities)
        {
            _entities.AddRange(entities
                ?? throw new ArgumentNullException("Entities must be not null!"));
            _context.SaveChanges();
        }
        public virtual ICollection<TEntity> GetByCondition(Expression<Func<TEntity,
bool>> condition)
            => _entities.Where(condition).ToHashSet();
        public virtual TEntity GetByIdWithDetails(long id)
            => GetById(id);
    }
```

```
        public class ProductRepository : GenericRepository<Product>, IProductRepository
        {
            public ProductRepository(CulinaryContext context)
                : base(context)
            { }
            public override void DeleteById(long id)
            {
                var entity = _entities
                    .Include(x => x.FilterParts)
                    .Include(x => x.Ingredients)
                    .Include(x => x.Categories)
                    .Include(x => x.Subcategories)
                    .FirstOrDefault(x => x.Id == id);
                Delete(entity);
            }
            public override void Delete(Product entity)
            {
                var filterParts = entity.FilterParts;
                var categories = entity.Categories;
                var subcategories = entity.Subcategories;
                var image = entity.Image;
                if (image != null)
                    _context.Remove(image);
                if (filterParts?.Count() > 0)
                    _context.RemoveRange(filterParts);
                if (categories?.Count() > 0)
                    _context.RemoveRange(categories);
                if (subcategories?.Count() > 0)
                    _context.RemoveRange(subcategories);
                base.Delete(entity);
            }
            public override bool Update(Product product)
            {
                if (product == null || !_entities.Any(x => x.Id == product.Id))
                {
                    return false;
                }
                _entities.Update(product);

                long id = product.Id;
                var categories = product.Categories.Select(x => x.Id);
                var subcategories = product.Subcategories.Select(x => x.Id);
                var newRelations = (categories?.Count() > 0
                    ? subcategories?.Count() > 0
                        ? categories.Concat(subcategories)
                        : categories
                    : subcategories).ToHashSet();
                var relationsToDelete = _context.ProductCategories.Where(x => (x.ProductId
    == id || x.CategoryId == id)
                    && !newRelations.Contains(x.Id));
                _context.RemoveRange(relationsToDelete);
                _context.SaveChanges();
                return true;
            }
            public override Product GetByIdWithDetails(long id)
            {
                return _entities
                    .Include(x => x.Image)
                    .Include(x => x.Subcategories)
                        .ThenInclude(child => child.Product)
                    .Include(x => x.Categories)
                        .ThenInclude(child => child.Category)
                    .FirstOrDefault(x => x.Id == id);
            }
```

```
        public ICollection<Product> GetProductsWithRelations(HashSet<long> productIds)
        {
            _entities.Load();
            _context.ProductCategories.Load();
            return _entities
                .Where(x => productIds.Contains(x.Id))
                .ToList();
        }
        public Product GetProductWithRelations(long id)
        {
            return _entities
                .Include(x => x.Categories)
                .Include(x => x.Subcategories)
                .FirstOrDefault(x => x.Id == id);
        }
        public ICollection<Product> GetProductsWithRelations()
        {
            return _entities
                .Include(x => x.Categories)
                .Include(x => x.Subcategories)
                .ToList();
        }
    }
    public class CulinaryContext : DbContext
    {
        public DbSet<Filter> Filters { get; set; }
        public DbSet<Image> Images { get; set; }
        public DbSet<FilterPart> FilterParts { get; set; }
        public DbSet<Ingredient> Ingredients { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<ProductCategory> ProductCategories { get; set; }
        public DbSet<Profile> Profiles { get; set; }
        public DbSet<RecipeTag> RecipeTags { get; set; }
        public DbSet<Recipe> Recipes { get; set; }
        public DbSet<RecipeStep> RecipeSteps  { get; set; }

        public CulinaryContext(DbContextOptions<CulinaryContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder
                .ApplyConfiguration(new FilterConfig())
                .ApplyConfiguration(new ImageConfig())
                .ApplyConfiguration(new IngredientConfig())
                .ApplyConfiguration(new ProductConfig())
                .ApplyConfiguration(new ProfileConfig())
                .ApplyConfiguration(new RecipeTagConfig())
                .ApplyConfiguration(new RecipeStepConfig())
                .ApplyConfiguration(new ProductCategoryConfig())
                .ApplyConfiguration(new FilterPartConfig())
                .ApplyConfiguration(new RecipeConfig());

            foreach (var relationship in
modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
            {
                relationship.DeleteBehavior = DeleteBehavior.ClientSetNull;
            }
        }
    }
```

```
        public class FilterService: IFilterService
        {
            private readonly IFilterRepository _filterRepository;
            public FilterService(IFilterRepository filterRepository)
            {
                _filterRepository = filterRepository;
            }
            public ICollection<FilterDTO> GetAllFilters()
            {
                return _filterRepository
                    .GetAll()
                    .Select(filter => filter?.ToDTO())
                    .ToList();
            }
            public ICollection<FilterDTO> GetDefaultFilters()
            {
                return _filterRepository
                    .GetByCondition(filter => filter.IsDefault)
                    .Select(filter => filter?.ToDTO())
                    .ToList();
            }
            public FilterDetailsDTO GetFilterDetails(long filterId)
            {
                return _filterRepository.GetByIdWithDetails(filterId).ToDetailsDTO();
            }
            public ICollection<FilterDTO> GetFiltersByAuthor(long? profileId)
            {
                return _filterRepository
                    .GetByCondition(filter => filter.AuthorId == profileId)
                    .Select(filter => filter?.ToDTO())
                    .ToList();
            }
            public ICollection<FilterDTO> GetAvailableFiltersByProfile(long? profileId)
            {
                return _filterRepository
                    .GetByCondition(filter => filter.AuthorId == profileId ||
    filter.IsDefault)
                    .Select(filter => filter?.ToDTO())
                    .ToList();
            }
            public long CreateFilter(FilterDetailsDTO filterDTO)
            {
                return _filterRepository.Add(filterDTO?.ToEntity());
            }
            public bool UpdateFilter(FilterDetailsDTO filterDTO)
            {
                throw new NotImplementedException();
            }
            public void DeleteFilter(long filterId)
            {
                _filterRepository.DeleteById(filterId);
            }

            public bool SafeDelete(long filterId, long profileId)
            {
                var filter = _filterRepository.GetByIdWithDetails(filterId);
                if (filter?.AuthorId != profileId)
                {
                    return false;
                }
                _filterRepository.Delete(filter);
                return true;
            }
        }
```

```
public class ProductService: IProductService
{
    private readonly IProductRepository _productRepository;
    public ProductService(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }
    public long CreateProduct(ProductFullDetailsDTO productDTO)
    {
        return _productRepository.Add(productDTO?.ToEntity());
    }
    public void DeleteProduct(long id)
    {
        _productRepository.DeleteById(id);
    }
    public ICollection<ProductDTO> GetParentCategories(long id)
    {
        var product = _productRepository.GetByIdWithDetails(id);
        return product.Categories
            .Select(x => x?.Category?.ToDTO()).ToList();
    }
    public ProductDTO GetProductById(long id)
    {
        return _productRepository.GetById(id)?.ToDTO();
    }
    public ProductFullDetailsDTO GetProductFullDetails(long id)
    {
        var product = _productRepository.GetByIdWithDetails(id);
        return product?.ToFullDetailsDTO();
    }
    public ICollection<ProductDTO> GetProducts()
    {
        return _productRepository
            .GetAll()
            .Select(product => product?.ToDTO())
            .ToList();
    }
    public ICollection<ProductDetailsWithRelationsDTO> GetProductsWithRelations()
    {
        return _productRepository
            .GetProductsWithRelations()
            .Select(product => product?.ToCategoryDTO())
            .ToList();
    }
    public ProductDetailsWithRelationsDTO GetProductDetails(long id)
    {
        return _productRepository.GetProductWithRelations(id)?.ToCategoryDTO();
    }
    public ICollection<ProductDTO> GetSubcategories(long id)
    {
        var product = _productRepository.GetByIdWithDetails(id);
        return product.Subcategories
            .Select(product => product?.Product?.ToDTO()).ToList();
    }
    public bool UpdateProduct(ProductFullDetailsDTO product)
    {
        return _productRepository.Update(product?.ToEntity());
    }
}
```

```
public class RecipeService: IRecipeService
{
        private readonly IRecipeRepository _recipeRepository;
        private readonly IProductRepository _productRepository;
        private readonly IRecipeTagRepository _tagRepository;
        private readonly IFilterRepository _filterRepository;
        public RecipeService
        (
            IRecipeRepository recipeRepository,
            IProductRepository productRepository,
            IRecipeTagRepository tagRepository,
            IFilterRepository filterRepository
        )
        {
            _recipeRepository = recipeRepository;
            _filterRepository = filterRepository;
            _productRepository = productRepository;
            _tagRepository = tagRepository;
        }
        public HashSet<string> GetAllTags()
        {
            return _tagRepository.GetTags().ToHashSet();
        }
        public RecipeDTO GetRecipeById(long id)
        {
            var recipe = _recipeRepository.GetById(id);
            return recipe?.ToDTO();
        }
        public long CreateRecipe(RecipeDetailsDTO recipeDTO)
        {
            var recipe = recipeDTO?.ToEntity();
            return _recipeRepository.Add(recipe);
        }

        public RecipeDetailsDTO GetRecipeDetails(long id)
        {
            var recipe = _recipeRepository.GetByIdWithDetails(id);
            return recipe?.ToDetaisDTO();
        }
        public RecipeDetailsDTO GetRecipeDetailsWithoutImages(long id)
        {
            return _recipeRepository.GetDetailsWithoutImages(id)?.ToDetaisDTO();
        }
        public ICollection<RecipeDTO> GetRecipes()
        {
            return _recipeRepository.GetAll().Select(recipe =>
recipe?.ToDTO()).ToList();
        }
        public bool UpdateRecipe(RecipeDetailsDTO recipe)
        {
            return _recipeRepository.Update(recipe.ToEntity());
        }
        public ICollection<RecipeDTO> GetRecipesByAuthor(long profileId)
        {
            return _recipeRepository
                .GetByCondition(filter => filter.AuthorId == profileId)
                .Select(filter => filter?.ToDTO())
                .ToList();
        }
        public ICollection<RecipeDTO> GetDefaultRecipes()
        {
            return _recipeRepository
                .GetByCondition(filter => filter.AuthorId == 0 || filter.AuthorId ==
null)
```

```
                .Select(filter => filter?.ToDTO())
                .ToList();
        }
        public void DeleteRecipe(long id)
        {
            _recipeRepository.DeleteById(id);
        }
        public bool SafeDeleteRecipe(long recipeId, long profileId)
        {
            var recipe = _recipeRepository.GetByIdWithDetails(recipeId);
            if (recipe?.AuthorId != profileId)
            {
                return false;
            }
            _recipeRepository.Delete(recipe);
            return true;
        }
       public void DeleteProfileRecipes(long profileId)
        {
            _recipeRepository.DeleteRecipesByProfile(profileId);
        }

        public ICollection<RecipeDTO> GetRecipesByFilter(long filterId)
        {
            var filter = _filterRepository.GetByIdWithDetails(filterId);
            return GetRecipesByFilter(filter?.ToDetailsDTO());
        }
        public ICollection<RecipeDTO> GetRecipesByFilter(FilterDetailsDTO filter)
        {
            if (filter == null)
            {
                throw new ArgumentNullException("Filter must be not null!");
            }

            bool filterByTags = filter.Tags?.Count > 0;
            bool filterByIngredients = filter.Ingredients?.Count > 0;
            string lowerTitle = filter.RecipeTitle?.ToLower() ?? "";
            int minDuration = filter.MinDuration ?? 0;
            int maxDuration = filter.MaxDuration ?? int.MaxValue;
            double minCals = filter.MinCalories ?? 0;
            double maxCals = filter.MaxCalories ?? int.MaxValue;

            var recipes = _recipeRepository.GetRecipesByCondition(recipe =>
(recipe.Title == null || recipe.Title.Contains(lowerTitle))
                && (recipe.Duration == null || (minDuration <= recipe.Duration &&
maxDuration >= recipe.Duration))
                && (recipe.Calories == null || (minCals <= recipe.Calories && maxCals
>= recipe.Calories)),
                filterByIngredients, filterByTags);

            if (filterByTags && recipes.Count > 0)
            {
                recipes = FilterRecipesByTags(filter, recipes);
            }
            if (filterByIngredients && recipes.Count > 0)
            {
                recipes = filter.ByAvailableProducts
                    ? FilterRecipesByAvailableIngredients(filter, recipes)
                    : FilterRecipesByIngredients(filter, recipes);
            }
            return recipes.Select(recipe => recipe?.ToDTO()).ToList();
        }
        public long? GetRecipeAuthorId(long recipeId)
        {
```

```
            return _recipeRepository.GetRecipeAuthorId(recipeId);
        }
        private ICollection<Recipe> FilterRecipesByTags(FilterDetailsDTO filter,
ICollection<Recipe> recipes)
        {
            var requiredTags = filter.Tags?
                .Where(part => part.Necessity && part.Tag?.Length > 0)
                .Select(part => part.Tag.Trim().ToLower())
                .ToHashSet();
            var forbiddenTags = filter.Tags?
                .Where(part => !part.Necessity && part.Tag?.Length > 0)
                .Select(part => part.Tag.Trim().ToLower())
                .ToHashSet();
            var filteredRecipes = recipes.Where(recipe =>
            {
                var tags = recipe.Tags?.Select(x =>
x.Tag?.Trim().ToLower()).ToHashSet();
                if (forbiddenTags.Intersect(tags).Any() ||
!tags.IsSupersetOf(requiredTags))
                { return false; }
                return true;
            }).ToList();

            return filteredRecipes;
        }
        private ICollection<Recipe> FilterRecipesByIngredients(FilterDetailsDTO
filter, ICollection<Recipe> recipes)
        {
            var rootRequired = filter.Ingredients?
                .Where(part => part.Necessity)
                .Select(part => part.ProductId)
                .ToHashSet();
            var rootForbidden = filter.Ingredients?
                .Where(part => !part.Necessity)
                .Select(part => part.ProductId)
                .ToHashSet();
            if (filter.OnlyProducts)
            {
                return recipes.Where(recipe =>
                {
                    var requiredIngs = recipe.Ingredients
                        .Where(ingredient => ingredient.ProductId.HasValue &&
ingredient.Necessity == true)
                        .Select(x => x.ProductId.Value);
                    if (rootForbidden.Intersect(requiredIngs).Any())
                    { return false; }
                    if (rootRequired.Count > 0)
                    {
                        var ingredients = recipe.Ingredients
                            .Where(ingredient => ingredient.ProductId.HasValue)
                            .Select(x => x.ProductId.Value).ToHashSet();
                        if (!rootRequired.Intersect(ingredients).Any())
                        { return false; }
                    }
                    return true;
                }).ToList();
            }

            var requiredProducts =
_productRepository.GetProductsWithRelations(rootRequired);
            var forbiddenProducts =
_productRepository.GetProductsWithRelations(rootForbidden);
            var forbiddenList = FindForbiddenProducts(forbiddenProducts);
```

```
            var requiredList = FindRequiredProducts(forbiddenList, rootRequired,
requiredProducts);
            var filteredRecipes = recipes.Where(recipe =>
            {
                var requiredIngs = recipe.Ingredients
                    .Where(ingredient => ingredient.ProductId.HasValue &&
ingredient.Necessity == true)
                    .Select(x => x.ProductId.Value);
                if (forbiddenList.Intersect(requiredIngs).Any())
                {
                    return false;
                }
                var ingredients = recipe.Ingredients
                    .Where(ingredient => ingredient.ProductId.HasValue)
                    .Select(x => x.ProductId.Value).ToHashSet();
                foreach (var optionalProducts in requiredList)
                {
                    if (!optionalProducts.Intersect(ingredients).Any())
                    { return false; }
                }
                return true;
            }).ToList();
            return filteredRecipes;
        }
        private ICollection<Recipe>
FilterRecipesByAvailableIngredients(FilterDetailsDTO filter, ICollection<Recipe>
recipes)
        {
            var rootRequired = filter.Ingredients
                .Where(part => part.Necessity)
                .Select(part => part.ProductId)
                .ToHashSet();
            var rootAvailable = filter.Ingredients
                .Select(part => part.ProductId)
                .ToHashSet();
            if (filter.OnlyProducts)
            {
                return recipes.Where(recipe =>
                {
                    var requiredIngs = recipe.Ingredients
                        .Where(ingredient => ingredient.ProductId.HasValue &&
ingredient.Necessity == true)
                        .Select(x => x.ProductId.Value);
                    if (!rootAvailable.IsSupersetOf(requiredIngs))
                    {
                        return false;
                    }
                    if (rootRequired.Count > 0)
                    {
                        var ingredients = recipe.Ingredients
                            .Where(ingredient => ingredient.ProductId.HasValue)
                            .Select(x => x.ProductId.Value).ToHashSet();
                        if (!rootRequired.Intersect(ingredients).Any())
                        {
                            return false;
                        }
                    }
                    return true;
                }).ToList();
            }
            var requiredProducts =
_productRepository.GetProductsWithRelations(rootRequired);
            var availableProducts =
_productRepository.GetProductsWithRelations(rootAvailable);
```

```
            var availableList = FindAvailableProducts(availableProducts);
            var requiredList = FindRequiredProducts(new HashSet<long>(), rootRequired,
requiredProducts);

            var filteredRecipes = recipes.Where(recipe =>
            {
                var requiredIngs = recipe.Ingredients
                    .Where(ingredient => ingredient.ProductId.HasValue &&
ingredient.Necessity == true)
                    .Select(x => x.ProductId.Value).ToHashSet();
                if (!availableList.IsSupersetOf(requiredIngs))
                { return false; }
                var ingredients = recipe.Ingredients
                    .Where(ingredient => ingredient.ProductId.HasValue)
                    .Select(x => x.ProductId.Value).ToHashSet();
                foreach (var optionalProducts in requiredList)
                {
                    if (!optionalProducts.Intersect(ingredients).Any())
                    {
                        return false;
                    }
                }
                return true;
            }).ToList();

            return filteredRecipes;
        }
    private HashSet<long> FindAvailableProducts(ICollection<Product> available)
        {
            var availableList = new HashSet<long>();
            FindAvailableParents(availableList, available);
            return availableList;
        }
        private void FindAvailableParents(ICollection<long> available,
IEnumerable<Product> products)
        {
            if (products == null)
            { return; }
            foreach (var product in products)
            {
                long id = product.Id;
                if (available.Contains(id))
                { continue; }
                available.Add(id);
                FindAvailableParents(available, product.Categories?.Select(x =>
x.Category));
            }
        }
        private List<HashSet<long>> FindRequiredProducts(HashSet<long> forbidden,
            HashSet<long> rootRequired, ICollection<Product> requiredProducts)
        {
            List<HashSet<long>> required = new List<HashSet<long>>();
            foreach (var product in requiredProducts)
            {
                var optional = new HashSet<long>() { product.Id };
                if (FindRequiredChildren(rootRequired, forbidden, optional,
product.Subcategories?.Select(x => x.Product)))
                {
                    FindRequiredParents(forbidden, optional,
product.Categories?.Select(x => x.Category));
                    required.Add(optional);
                }
            }
            return required;
```

```
        }
        private void FindRequiredParents(HashSet<long> forbidden, HashSet<long>
optional, IEnumerable<Product> products)
        {
            if (products == null)
            { return; }
            foreach (var product in products)
            {
                long id = product.Id;
                if (forbidden.Union(optional).Contains(id))
                { continue; }
                optional.Add(id);
                FindRequiredParents(forbidden, optional, product.Categories?.Select(x
=> x.Category));
            }

            return;
        }
        private bool FindRequiredChildren(HashSet<long> required, HashSet<long>
forbidden,
            HashSet<long> optional, IEnumerable<Product> products)
        {
            if (products == null)
            { return true; }
            foreach (var product in products)
            {
                long id = product.Id;
                if (forbidden.Union(optional).Contains(id))
                { continue; }
                if (required.Contains(id))
                {
                    required.Remove(id);
                    return false;
                }
                optional.Add(id);
                if (!FindRequiredChildren(required, forbidden, optional,
product.Subcategories?.Select(x => x.Product)))
                { return false; }
            }
            return true;
        }
        private void FindForbiddenChildren(ICollection<long> forbidden,
IEnumerable<Product> products)
        {
            if (products == null)
            { return; }
            foreach (var product in products)
            {
                long id = product.Id;
                if (forbidden.Contains(id))
                { continue; }
                forbidden.Add(id);
                FindForbiddenChildren(forbidden, product.Subcategories?.Select(x =>
x.Product));
            }
        }
        private HashSet<long> FindForbiddenProducts(ICollection<Product> forbidden)
        {
            var forbiddenList = new HashSet<long>();
            FindForbiddenChildren(forbiddenList, forbidden);
            return forbiddenList;
        }
    }
```

```
public class ProfileService: IProfileService
{
    private readonly IProfileRepository _profileRepository;
    public ProfileService(IProfileRepository profileRepository)
    {
        _profileRepository = profileRepository;
    }
    public long CreateProfile(ProfileDetailsDTO profileDTO, string userId)
    {
        if (profileDTO == null)
        {
            return 0;
        }
        var profile = profileDTO.ToEntity();
        profile.UserId = userId;
        return _profileRepository.Add(profile);
    }
    public void DeleteProfile(long profileId)
    {
        _profileRepository.DeleteById(profileId);
    }
    public ProfileDTO GetProfile(long profileId)
    {
        return _profileRepository.GetById(profileId)?.ToDTO();
    }
    public ProfileDTO GetProfileByUser(string userId)
    {
        return _profileRepository.GetProfileByUser(userId)?.ToDTO();
    }
    public long? GetProfileIdByUser(string userId)
    {
        return _profileRepository.GetProfileByUser(userId)?.Id;
    }
    public ProfileDetailsDTO GetProfileDetails(long profileId)
    {
        return _profileRepository.GetById(profileId)?.ToDetailsDTO();
    }
    public ProfileDetailsDTO GetProfileDetailsByUser(string userId)
    {
        return _profileRepository.GetProfileByUser(userId)?.ToDetailsDTO();
    }
    public ICollection<ProfileDTO> GetProfiles()
    {
        return _profileRepository.GetAll()
            .Select(profile => profile?.ToDTO())
            .ToList();
    }
    public bool UpdateProfile(ProfileDetailsDTO profile)
    {
        return _profileRepository.Update(profile?.ToEntity());
    }
}

[Route("api/recipes")]
[ApiController]
public class RecipesController : ControllerBase
{
    private readonly IRecipeService _recipeService;
    private readonly IProfileService _profileService;

    private long? CurrentProfileId
    {
        get
        {
```

```
                string userId = User?.Claims?.FirstOrDefault(c => c.Type ==
Claims.USER_ID)?.Value;
                return _profileService.GetProfileIdByUser(userId);
            }
        }
        public RecipesController(IRecipeService recipeService, IProfileService
profileService)
        {
            _recipeService = recipeService;
            _profileService = profileService;
        }
        [HttpGet]
        public IActionResult GetAllRecipes()
        {
            var result = _recipeService.GetRecipes();
            if (result == null)
                return BadRequest(result);
            return Ok(result);
        }
        [HttpGet("{id}")]
        public IActionResult GetRecipeById(long id)
        {
            var result = _recipeService.GetRecipeById(id);
            if (result == null)
                return BadRequest(result);
            return Ok(result);
        }
        [HttpPost("create")]
        [Authorize]
        public IActionResult CreateRecipe([FromBody] RecipeDetailsDTO recipe)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            recipe.AuthorId = CurrentProfileId;
            recipe.Time = DateTime.UtcNow;
            var result = _recipeService.CreateRecipe(recipe);
            if (result <= 0)
                return BadRequest(result);
            return Ok(result);
        }
        [HttpPost("create/default")]
        [Authorize(Roles = Roles.ADMIN)]
        public IActionResult CreateDefaultRecipe([FromBody] RecipeDetailsDTO recipe)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            recipe.AuthorId = 0;
            recipe.Time = DateTime.UtcNow;
            var result = _recipeService.CreateRecipe(recipe);
            if (result <= 0)
                return BadRequest(result);
            return Ok(result);
        }
        [HttpPut("edit")]
        [Authorize]
        public IActionResult EditRecipe([FromBody] RecipeDetailsDTO recipe)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            if (recipe.AuthorId != CurrentProfileId)
                return Forbid();
            recipe.Time ??= DateTime.UtcNow;
            var result = _recipeService.UpdateRecipe(recipe);
            if (!result)
```

```
            return BadRequest(result);
        return Ok();
    }
    [HttpGet("{id}/details")]
    public IActionResult GetRecipeWithDetails(long id)
    {
        var result = _recipeService.GetRecipeDetails(id);
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("{id}/details/noimages")]
    public IActionResult GetRecipeDetailsWithoutImages(long id)
    {
        var result = _recipeService.GetRecipeDetailsWithoutImages(id);
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpPost("filters")]
    public IActionResult GetRecipesByFilter([FromBody] FilterDetailsDTO filter)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);
        var result = _recipeService.GetRecipesByFilter(filter);
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("filters/{id}")]
    public IActionResult GetRecipesByFilter(long id)
    {
        var result = _recipeService.GetRecipesByFilter(id);
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("profiles/{id}")]
    public IActionResult GetRecipesByProfile(long id)
    {
        var result = _recipeService.GetRecipesByAuthor(id);
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("tags")]
    public IActionResult GetTags()
    {
        var result = _recipeService.GetAllTags();
        if (result == null)
            return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("my")]
    [Authorize]
    public IActionResult GetCurrentProfileRecipes()
    {
        var profileId = CurrentProfileId;
```

```
            if (!profileId.HasValue)
                return NoContent();
            var result = _recipeService.GetRecipesByAuthor(profileId.Value);
            if (result == null)
                return BadRequest(result);
            return Ok(result);
        }

        [HttpGet("default")]
        [Authorize(Roles = Roles.ADMIN)]
        public IActionResult GetDefaultRecipes()
        {
            var result = _recipeService.GetDefaultRecipes();
            if (result == null)
                return BadRequest(result);
            return Ok(result);
        }

        [HttpDelete("{id}")]
        [Authorize]
        public IActionResult DeleteRecipe(long id)
        {
            var roleClaim = (new IdentityOptions()).ClaimsIdentity.RoleClaimType;
            var userId = User?.Claims?.FirstOrDefault(c => c.Type ==
Claims.USER_ID)?.Value;
            var role = User?.Claims?.FirstOrDefault(cl => cl.Type ==
roleClaim)?.Value;

            if (role == Roles.ADMIN)
            {
                _recipeService.DeleteRecipe(id);
            }
            else
            {
                var authorId = CurrentProfileId;
                if (authorId == null || !_recipeService.SafeDeleteRecipe(id,
authorId.Value))
                {
                    return Forbid();
                }
            }
            return NoContent();
        }

        [HttpDelete("my")]
        [Authorize]
        public IActionResult DeleteCurrentProfileRecipes()
        {
            var profileId = CurrentProfileId;
            if (profileId == null)
            {
                return Forbid();
            }
            _recipeService.DeleteProfileRecipes(profileId.Value);
            return NoContent();
        }
    }

    [Route("api/auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly ApplicationSettings _appSettings;
```

```
        private readonly IProfileService _profileService;
        public AuthController
        (
            UserManager<ApplicationUser> userManager,
            IOptions<ApplicationSettings> appSettings,
            IProfileService profileService
        )
        {
            _userManager = userManager;
            _appSettings = appSettings.Value;
            _profileService = profileService;
        }
        [HttpPost("signup")]
        public async Task<IActionResult> SignUpUser([FromBody]ApplicationUserModel
userModel)
        {
            return await SignUp(userModel, Roles.USER);
        }
        [HttpPost("signup/admin")]
        [Authorize(Roles=Roles.ADMIN)]
        public async Task<IActionResult> SignUpAdmin([FromBody]ApplicationUserModel
userModel)
        {
            return await SignUp(userModel, Roles.ADMIN);
        }
        private async Task<IActionResult> SignUp(ApplicationUserModel userModel,
string role)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            var appUser = new ApplicationUser()
            {
                UserName = userModel.Login,
                Email = userModel.Email,
            };
            var result = await _userManager.CreateAsync(appUser, userModel.Password);
            if (!result.Succeeded)
            {
                return BadRequest();
            }
            var user = await _userManager.FindByNameAsync(userModel.Login);
            await _userManager.AddToRoleAsync(user, role);
            var profile = new ProfileDetailsDTO
            {
                FullName = userModel.FullName,
                DisplayName = userModel.Login,
                Email = userModel.Email
            };
            _profileService.CreateProfile(profile, user.Id);
            return Ok(result);
        }
        [HttpPost("passwords/change")]
        [Authorize()]
        public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordModel
model)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            var user = await _userManager.FindByNameAsync(model.Login);
            if (user?.Id != User?.Claims?.FirstOrDefault(c => c.Type ==
Claims.USER_ID)?.Value)
            {
                return Forbid();
```

```
            }
            var result = await _userManager.ChangePasswordAsync(user,
model.OldPassword, model.NewPassword);
            if (result != null && result.Succeeded)
                return Ok(result);
            return BadRequest(result);
        }
        [HttpPost("signin")]
        public async Task<IActionResult> SignIn([FromBody] SignInModel userModel)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            var user = await _userManager.FindByNameAsync(userModel.Login);
            if (user != null && await _userManager.CheckPasswordAsync(user,
userModel.Password))
            {
                var role = (await _userManager.GetRolesAsync(user))?.FirstOrDefault();
                var profile = _profileService.GetProfileByUser(user?.Id);
                if (profile == null && role != Roles.ADMIN)
                {
                    return new UnauthorizedResult();
                }
                var claims = new List<Claim>() { new Claim(Claims.USER_ID,
user.Id.ToString()) };
                if (role?.Length > 0)
                {
                    IdentityOptions _options = new IdentityOptions();
                    claims.Add(new Claim(_options.ClaimsIdentity.RoleClaimType,
role));
                }
                var key = Encoding.UTF8.GetBytes(_appSettings.JWT_Secret);
                var tokenDescriptor = new SecurityTokenDescriptor
                {
                    Subject = new ClaimsIdentity(claims),
                    Expires = DateTime.UtcNow.AddDays(1),
                    SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256)
                };
                var tokenHandler = new JwtSecurityTokenHandler();
                var securityToken = tokenHandler.CreateToken(tokenDescriptor);
                var token = tokenHandler.WriteToken(securityToken);
                var response = new SignInResponse
                {
                    Token = token,
                    IsAdmin = role == Roles.ADMIN,
                    DisplayName = profile?.DisplayName
                };
                return Ok(response);
            }
            return new BadRequestResult();
        }
        [HttpGet("{login}/unique")]
        public async Task<bool> LoginCheckForUniqueness(string login)
        {
            if (await _userManager.FindByNameAsync(login) == null)
            {
                return true;
            }
            return false;
        }
    }
```

```
        [Route("api/filters")]
        [ApiController]
        public class FiltersController : ControllerBase
        {
            private readonly IProfileService _profileService;
            private readonly IFilterService _filterService;
            private long? CurrentProfileId
            {
                get
                {
                    string userId = User?.Claims?.FirstOrDefault(c => c.Type ==
    Claims.USER_ID)?.Value;
                    return _profileService.GetProfileIdByUser(userId);
                }
            }
            public FiltersController(IFilterService filterService, IProfileService
    profileService)
            {
                _filterService = filterService;
                _profileService = profileService;
            }
            [HttpGet]
            public IActionResult GetAvailableFilters()
            {
                var result = CurrentProfileId.HasValue
                    ? _filterService.GetAvailableFiltersByProfile(CurrentProfileId)
                    : _filterService.GetDefaultFilters();
                if (result == null)
                    return BadRequest(result);
                return Ok(result);
            }
            [HttpGet("public")]
            public IActionResult GetPublicFilters()
            {
                var result = _filterService.GetDefaultFilters();
                if (result == null)
                    return BadRequest(result);
                return Ok(result);
            }
            [HttpGet("my")]
            [Authorize]
            public IActionResult GetOwnFilters()
            {
                long? profileId = CurrentProfileId;
                if (!profileId.HasValue)
                    return Forbid();
                var result = _filterService.GetFiltersByAuthor(profileId.Value);
                if (result == null)
                    return BadRequest();
                return Ok(result);
            }
            [HttpPost("create")]
            [Authorize]
            public IActionResult CreateFilter([FromBody] FilterDetailsDTO filter)
            {
                if (!ModelState.IsValid)
                    return BadRequest(ModelState);
                var roleClaim = (new IdentityOptions()).ClaimsIdentity.RoleClaimType;
                var userId = User?.Claims?.FirstOrDefault(c => c.Type ==
    Claims.USER_ID)?.Value;
                var role = User?.Claims?.FirstOrDefault(cl => cl.Type ==
    roleClaim)?.Value;

                if (filter.IsDefault && role != Roles.ADMIN)
```

```
            {
                return Forbid();
            }
            filter.AuthorId = CurrentProfileId;
            var result = _filterService.CreateFilter(filter);
            if (result <= 0)
                return BadRequest(result);
            return Ok(result);
        }
        [HttpDelete("{id}")]
        [Authorize]
        public IActionResult DeleteFilter(long id)
        {
            var roleClaim = (new IdentityOptions()).ClaimsIdentity.RoleClaimType;
            var userId = User?.Claims?.FirstOrDefault(c => c.Type ==
Claims.USER_ID)?.Value;
            var role = User?.Claims?.FirstOrDefault(cl => cl.Type ==
roleClaim)?.Value;
            if (role == Roles.ADMIN)
            {
                _filterService.DeleteFilter(id);
                return NoContent();
            }
            var authorId = CurrentProfileId;
            if (authorId == null || !_filterService.SafeDelete(id, authorId.Value))
                return Forbid();
            return NoContent();
        }
        [HttpGet("{id}/details")]
        public IActionResult GetByIdWithDetails(long id)
        {
            var result = _filterService.GetFilterDetails(id);
            if (result == null)
                return BadRequest();
            if (!result.IsDefault && result.AuthorId != CurrentProfileId)
                return Forbid();
            return Ok(result);
        }
    }
```

КЛІЄНТСЬКА ЧАСТИНА

```
export class FiltersService {
  private filters: Map<number, IFilterModel> =
new Map();
  private currRootProductId: number;
  private breadCrumbs: number[] = [];
  private currProducts: number[] = [];
  readonly products: Map<number,
IFilterProduct> = new Map();
  readonly filtersChanged$:
BehaviorSubject<IFilterGeneralModel[]> = new
BehaviorSubject<IFilterGeneralModel[]>(null);
  readonly currProductsChanged$:
BehaviorSubject<number[]> = new
BehaviorSubject<number[]>(null);
  readonly onProductsUpdated$:
BehaviorSubject<any> = new
BehaviorSubject<any>(null);
  readonly currRootProductChanged$:
BehaviorSubject<number> = new
BehaviorSubject<number>(null);
  readonly requiredTags: Set<string> = new Set();
  readonly forbiddenTags: Set<string> = new
Set();
  readonly onCurrFilterChanged$:
BehaviorSubject<IFilterModel> = new
BehaviorSubject(null);
  currFilter: IFilterModel;
  private areProductsUpdated = false;
  constructor(
    private auth: AuthService,
    private server: ServerHttpService,
    private notifications: NotificationsService,
    private productStore: ProductsService,
    private imageStore: ImagesService
  ) {
    this.initCurrFilter();
  }
  get hasBreadCrumbs(): boolean {
    return !!this.breadCrumbs?.length;
  }
  get needUpdate() {
    return !(this.areProductsUpdated &&
this.productStore.isUpdated);
  }
  updateProducts() {
    this.areProductsUpdated = false;
    if (this.productStore.isUpdated) {
this.updateByProducts(this.productStore.store.val
ues());
    }
    else {
      this.productStore.updateProducts()
        .subscribe(products =>
this.updateByProducts(products));
    }
  }
  private updateByProducts(newProducts:
IterableIterator<IProductModel> |
IProductModel[]) {
    this.products.clear();
    const necessity = ProductNecessity.Undefined;
    for (const product of newProducts) {
      this.products.set(product.id, {...product,
necessity});
    }
    this.setRootProduct(0, false);
    this.onProductsUpdated$.next(null);
    this.areProductsUpdated = true;
  }
  clearProducts() {
    this.products.clear();
    this.areProductsUpdated = false;
    this.onProductsUpdated$.next(null);
  }
  private setCurrProductsByRoot(rootProduct?:
number) {
    let products = [...this.products.values()];
    if (rootProduct) {
      products = products.filter(x =>
x?.categories?.includes(rootProduct));
    }
    else if (rootProduct === 0) {
      products = products.filter(x =>
!x.categories?.length);
    }
    this.currProducts =
this.sortProductsByNameAndType(products);

this.currProductsChanged$.next(this.currProducts
);
  }
  getProductViewDetails(productId: number):
IProductView {
    const product = this.products.get(productId);
    if (!product) {
```

```
      return null;
    }
    return {
      ...product,
      categories: null,
      subcategories: null,
      imageSrc$:
this.imageStore.getImage(product.imageId),
      categoryNames: product.categories?.map(x =>
this.products.get(x)),
      subcategoryNames:
product.subcategories?.map(x =>
this.products.get(x)),
    };
  }
  setCurrProductsByName(namePart: string) {
    this.breadCrumbs = [];
    this.currRootProductId = null;
    this.currRootProductChanged$.next(null);
    let products = [...this.products.values()];
    if (namePart) {
      const check = new RegExp(namePart, 'i');
      products = products.filter(x =>
check.test(x.name));
    }
    this.currProducts =
this.sortProductsByNameAndType(products);

this.currProductsChanged$.next(this.currProducts
);
  }
  selectTag(tag: string, isRequired: boolean) {
    if (isRequired) {
      this.forbiddenTags.delete(tag);
      this.requiredTags.add(tag);
    }
    else {
      this.forbiddenTags.add(tag);
      this.requiredTags.delete(tag);
    }
  }
  isProductSelected(productId: number, necessity:
ProductNecessity): boolean {
    return this.products.get(productId)?.necessity
=== necessity;
  }
  areAllCurrentProductsSelected(necessity:
ProductNecessity): boolean {
    for (const id of this.currProducts) {
      if (this.products.get(id).necessity !==
necessity) {
        return false;
      }
    }
    return true;
  }
  areNotAllCurrentProductsSelected(necessity:
ProductNecessity): boolean {
    let anySelected = false;
    let anyNotSelected = false;
    for (const id of this.currProducts) {
      if (this.products.get(id).necessity ===
necessity) {
        if (!anySelected) {
          if (anyNotSelected) {
            return true;
          }
          anySelected = true;
        }
      } else if (!anyNotSelected) {
        if (anySelected) {
          return true;
        }
        anyNotSelected = true;
      }
    }
    return false;
  }
  selectAllCurrentProducts(necessity:
ProductNecessity) {
    this.currProducts.forEach(id =>
this.products.get(id).necessity = necessity);
    this.onProductsUpdated$.next(true);
  }

unselectAllCurrentProductsByNecessity(necessity
: ProductNecessity){
    this.currProducts.forEach(id => {
      const product = this.products.get(id);
      if (product.necessity === necessity){
        product.necessity =
ProductNecessity.Undefined;
      }
    });
    this.onProductsUpdated$.next(true);
  }
  removeTag(tag: string) {
    this.forbiddenTags.delete(tag);
```

```
    this.requiredTags.delete(tag);
  }
  private sortProductsByNameAndType(products:
IFilterProduct[]): number[] {
    return products
      .sort((x, y) => {
        const xChildren = x.subcategories?.length ? 1
: 0;
        const yChildren = y.subcategories?.length ? 1
: 0;
        if (xChildren === yChildren) {
          return x.name?.localeCompare(y.name);
        }
        return yChildren - xChildren;
      })
      .map(product => product.id);
  }
  setRootProduct(id: number, saveBreadCrumb =
true) {
    if (!id) {
      this.breadCrumbs = [];
    }
    if (saveBreadCrumb) {

this.breadCrumbs.push(this.currRootProductId);
    }
    this.currRootProductId = id;
    this.currRootProductChanged$.next(id);
    this.setCurrProductsByRoot(id);
  }
  updateFilters() {
    this.server.getFilters().pipe(
      take(1),
    ).subscribe(
      filters => {
        this.filters = new Map();
        filters.forEach(f => this.filters.set(f.id, f));
        this.onChangeFilters();
      },
      _ => this.createNotification('Помилка під час
завантаження фільтрів')
    );
  }
  setProductNecessity(productId: number,
necessity: ProductNecessity, needUpdate: boolean
= true) {
    const product = this.products?.get(productId);
    if (product) {
      product.necessity = necessity;
      // TO DO: check for forbidden
      if (needUpdate) {
        this.onProductsUpdated$.next(productId);
      }
    }
  }
  selectProduct(productId: number) {
    const product = this.products?.get(productId);
    if (product) {
      const currNecessity = product.necessity;
      switch (product.necessity) {
        case ProductNecessity.Required:
product.necessity =
ProductNecessity.NotRequired; break;
        case ProductNecessity.NotRequired:
product.necessity = ProductNecessity.Undefined;
break;
        default: product.necessity =
ProductNecessity.Required;
      }
      this.onProductsUpdated$.next(productId);
    }
  }
  returnBack() {
    if (this.hasBreadCrumbs) {
      this.setRootProduct(this.breadCrumbs.pop(),
false);
    }
  }
  applyFilter(filterId: number) {
    if (!filterId) {
      this.resetCurrentFilter();
      return;
    }
    this.server.getFilter(filterId).pipe(take(1),
filter(f => !!f)).subscribe(
      filterModel => {
        const newFilter = (filterModel as IFilter);
        this.filters.set(filterId, newFilter);
        this.currFilter = newFilter; // TO DO: clear
ingredients and tags?

this.onCurrFilterChanged$.next(this.currFilter);

this.updateProductsNecessity(newFilter.ingredient
s, newFilter.byAvailableProducts);
        this.updateTags(this.currFilter.tags);
        this.createNotification('Фільтр
встановлено', `Фільтр:
```

```
"${newFilter.filterTitle}"`,
NotificationType.Success);
      },
      _ => this.createNotification('Фільтр не
встановлено', 'Помилка під час завантаження
фільтру')
    );
  }
  removeFilter(filterId: number) {
    if (this.currFilter.id === filterId) {
      this.currFilter.filterTitle = '';
      this.currFilter.id = 0;
    }
    this.filters.delete(filterId);
    this.onChangeFilters();
    this.server.deleteFilter(filterId).subscribe(
      _ => this.createNotification('Фільтр успішно
видалено', '', NotificationType.Success),
      error => this.createNotification('Фільтр не
видалено', 'Помилка під час видалення
фільтру')
    );
  }
  resetCurrentFilter() {
    this.initCurrFilter();
    this.updateProductsNecessity([], false);
    this.updateTags([]);
  }
  private updateTags(tags: IFilterTagModel[]) {
    this.requiredTags.clear();
    this.forbiddenTags.clear();
    tags?.forEach(tag =>
this.selectTag(tag.tag.toLocaleLowerCase(),
tag.necessity));
  }
  private updateProductsNecessity(ingredients:
IFilterIngredientModel[], byAvailableProducts:
boolean) {
    this.clearProductsNecessity(false);
    ingredients?.filter(x =>
x?.productId).forEach(ingredient => {
      const necessity = ingredient.necessity
        ? ProductNecessity.Required
        : ProductNecessity.NotRequired;
      this.setProductNecessity(ingredient.productId,
necessity, false);
    });
    this.onProductsUpdated$.next(true);
  }
  clearProductsNecessity(needUpdate = true) {
    this.products?.forEach(product =>
product.necessity = ProductNecessity.Undefined);
    if (needUpdate) {
      this.onProductsUpdated$.next(true);
    }
  }
  changeByAvailableProducts(byAvailable:
boolean) {
    this.currFilter.byAvailableProducts =
byAvailable;
    this.products?.forEach(product => {
      if (product.necessity ===
ProductNecessity.NotRequired) {
        product.necessity =
ProductNecessity.Undefined;
      }
    });
    this.onProductsUpdated$.next(true);
  }
  getProductsByNecessity(necessity:
ProductNecessity): IFilterProduct[] {
    return [...this.products.values()].filter(x =>
x.necessity === necessity);
  }
  private onChangeFilters() {

this.filtersChanged$.next([...this.filters.values()]);
  }
  getProduct(id: number): IFilterProduct {
    return this.products.get(id);
  }
  getRecipesByCurrentFilter():
Observable<IRecipeGeneralModel[]> {
    return
this.server.getRecipesByFilter(this.getCurrentFilte
rModel());
  }
  getRecipesByFilter(filterId: number):
Observable<IRecipeGeneralModel[]> {
    return this.server.getRecipesByFilterId(filterId);
  }
  saveCurrFilter(filterName: string) {
    const filterModel =
this.getCurrentFilterModel(filterName);
    if (this.auth.isAdmin) {
      filterModel.isDefault = true;
    }
```

```
    return
this.server.createFilter(filterModel).pipe(take(1))
      .subscribe(
        id => {
          filterModel.id = id;
          this.filters.set(id, filterModel);
          this.currFilter.id = 0;
          this.onChangeFilters();
          this.createNotification(`Фільтр
"${filterName}" збережено`, '',
NotificationType.Success, 300);
        },
        _ => this.createNotification('Фільтр не
збережено', 'Помилка під час збереження
фільтру')
      );
  }
  private getCurrentFilterModel(filterTitle: string
= ''): IFilterModel {
    const products = [...this.products.values()];
    const tags: IFilterTagModel[] = [];
    const ingredients: IFilterIngredientModel[] = [];
    products.filter(p => p.necessity !==
ProductNecessity.Undefined && p.necessity)
      .forEach(product => {
        ingredients.push({
          id: 0,
          productId: product.id,
          necessity: product.necessity ===
ProductNecessity.Required
        });
      });
    this.requiredTags.forEach(tag => tags.push({
id: 0, tag, necessity: true }));
    this.forbiddenTags.forEach(tag => tags.push({
id: 0, tag, necessity: false }));
    return {
      id: 0,
      recipeTitle: this.currFilter.recipeTitle,
      onlyProducts: this.currFilter.onlyProducts,
      byAvailableProducts:
this.currFilter.byAvailableProducts,
      authorId: this.currFilter.authorId,
      minDuration: this.currFilter.minDuration,
      maxDuration: this.currFilter.maxDuration,
      minCalories: this.currFilter.minCalories,
      maxCalories: this.currFilter.maxCalories,
      ingredients,
      tags,
      filterTitle,
      description: this.currFilter.description,
      isDefault: false,
    };
  }
  private initCurrFilter() {
    this.currFilter = {
      id: 0,
      filterTitle: '',
      isDefault: false,
      onlyProducts: false,
      byAvailableProducts: false,
    };
    this.onCurrFilterChanged$.next(this.currFilter);
  }
  createNotification(title: string, content: string =
'', type = NotificationType.Error, time: number =
3000) {
    this.notifications.create(title, content, type, {
      timeOut: time,
      showProgressBar: true,
      pauseOnHover: true,
      clickToClose: true
    });
  }
}
export class ImagesService {
  private readonly images: Map<number,
BehaviorSubject<string>> = new Map();
  imageLimitKb = 1024;
  allowedExtensions: string[] = ['png', 'jpeg', 'jpg'];
  defaultInvalidExtensionContent =
this.allowedExtensionsString;
  defaultInvalidExtensionTitle = 'Некоректний
формат зображення';
  defaultInvalidSizeTitle = 'Зображення надто
велике';
  defaultInvalidSizeContent = 'Зображення має
не перевищувати 1 МБ';

  constructor(
    private server: ServerHttpService,
    private auth: AuthService,
    private notificationService:
NotificationsService
  ) { }

  deleteImage(id: number) {
    if (this.auth.isAuthorized) {
```

```
this.server.deleteImage(id).pipe(take(1)).subscribe
(
      res => this.images.get(id)?.next(null)
    );
   }
  }

  deleteImages(ids: number[]) {
   if (this.auth.isAuthorized) {
    ids.forEach(id =>
this.server.deleteImage(id).pipe(take(1))
      .subscribe(res =>
this.images.get(id)?.next(null)));
   }
  }

  getImage(id: number): Observable<string> {
   if (!id) {
    return of(null);
   }
   let imageSubj = this.images.get(id);
   if (!imageSubj) {
    imageSubj = new
BehaviorSubject<string>(null);
    this.images.set(id, imageSubj);
    this.server.getImage(id).pipe(
     take(1),
     filter(image => !!image?.data),
     map(image => '' +
image.data),
      tap(newImage =>
imageSubj.next(newImage))
    ).subscribe();
   }
   return imageSubj;
  }
  updateImage(id: number) {
   this.updateImageFromServer(id,
this.images.get(id));
  }

  transformFileToImage(imageFile: File):
Observable<IImageModel> {
   const sbj = new Subject<IImageModel>();
   if (!imageFile) {
    return sbj;
   }
   const reader = new FileReader();
   reader.onload = (e: any) => {
    const src = e.target.result;
    const image = {
     id: 0,
     data: src.slice(src.indexOf(',') + 1),
     title: imageFile.name
    };
    sbj.next(image);
   };
   reader.readAsDataURL(imageFile);
   return sbj;
  }
  validateImageSize(file: File, sizeLimit: number
= this.imageLimitKb): boolean {
   return !sizeLimit || file.size <= sizeLimit *
1024;
  }servi
  validateFileExtension(file: File) {
   if (this.allowedExtensions?.length > 0) {
    return this.allowedExtensions.some(ext =>
file.name.toLocaleLowerCase().endsWith(
     (ext.startsWith('.') ? '' : '.') +
ext.toLowerCase())
    );
   }
   return true;
  }
  validateImageWithNotifications(file: File):
boolean {
   if (!file) {
    return false;
   }
   if (!this.validateFileExtension(file)) {

this.createNotification(this.defaultInvalidExtensio
nTitle, this.defaultInvalidExtensionContent);
    return false;
   }
   if (!this.validateImageSize(file)) {

this.createNotification(this.defaultInvalidSizeTitle
, this.defaultInvalidSizeContent,
NotificationType.Error);
    return false;
   }
   return true;
  }
  get allowedExtensionsString(): string {
```

```
    let allowedExtensionsListText;
    if (this.allowedExtensions?.length > 0){
      allowedExtensionsListText =
`.${this.allowedExtensions[0]}`;
      for (let i = 1; i <
this.allowedExtensions.length; i++) {
        allowedExtensionsListText += `,
.${this.allowedExtensions[i]}`;
      }
    }
    return 'Дозволені формати зображень: ' +
allowedExtensionsListText;
  }
  private updateImageFromServer(id: number,
imageSubj: BehaviorSubject<string>) {
    if (!imageSubj) {
      imageSubj = new
BehaviorSubject<string>(null);
      this.images.set(id, imageSubj);
    }
    this.server.getImage(id).pipe(
      take(1),
      filter(image => !!image?.data),
      map(image => '' +
image.data)
    ).subscribe(
      newImage => imageSubj.next(newImage),
    );
  }
  clearImages() {
    this.images.clear();
  }
}
export class ProductsService {
  store: Map<number, IProductModel> = new
Map();
  sortByNameProducts: IProductModel[] = [];
  isUpdated = false;
  updatedProducts$ = new
Subject<IProductModel[]>();
  private sendForUpdate = false;

  constructor(
    private server: ServerHttpService,
    private imageStore: ImagesService,
    private dialog: MatDialog,
    private auth: AuthService
  ) {
  }
  deleteProduct(id: number, name?: string):
Observable<any> {
    if (!this.auth.isAdmin) {
      return;
    }
    const data: IConfirmData = {
      question: 'Видалити продукт' + (name ? `
"${name}"?` : '?'),
      confirmation: 'Видалити',
      cancellation: 'Скасувати'
    };
    const dialogRef =
this.dialog.open(ConfirmDialogComponent,
      { width: '350px', data });
    return dialogRef.afterClosed().pipe(
      take(1),
      filter(res => !!res),
      switchMap(() =>
this.server.deleteProduct(id).pipe(
        take(1),
        tap(_ => {
          this.isUpdated = false;
          this.store.delete(id);
          this.updateSortByNameProducts();
        }),
      ))
    );
  }
  getProduct(id: number) {
    return this.store.get(id);
  }
  private updateSortByNameProducts() {
    this.sortByNameProducts =
this.products.sort((x, y) =>
x.name?.localeCompare(y.name));
  }
  get products(): IProductModel[] {
    return [...this.store.values()];
  }
  get hasProducts(): boolean {
    return !!this.productsNumber;
  }
  get productsNumber(): number {
    return this.store.size;
  }
  clearProducts() {
    this.isUpdated = false;
    this.store.clear();
    this.updateSortByNameProducts();
```

```
  }
  setProduct(id: number, product: IProductModel)
{
    this.store.set(id, product);
    this.updateSortByNameProducts();
  }
  hasProduct(id: number): boolean {
    return this.store.has(id);
  }
  updateProduct(id: number) {
    this.isUpdated = false;

this.server.getProduct(id).pipe(take(1)).subscribe(
product => {
      this.store.set(id, product);
      this.updateSortByNameProducts();
    });
  }
  updateProducts(): Observable<IProductModel[]>
{
    this.isUpdated = false;
    if (!this.sendForUpdate) {
      this.sendForUpdate = true;

this.server.getProductsWithRelations().pipe(take(
1)).subscribe(
        products => {
          this.sendForUpdate = false;
          this.store.clear();
          this.isUpdated = true;
          products?.filter(product =>
!!product).forEach(product =>
            this.store.set(product?.id, product));
          this.updateSortByNameProducts();
          this.updatedProducts$.next(products);
        },
        _ => this.sendForUpdate = false
      );
    }
    return this.updatedProducts$;
  }
  sortProductsByNameAndType(products:
IProductView[]): IProductModel[] {
    return products
      .sort((x, y) => {
        const xChildren = x.subcategories?.length ? 1
: 0;
        const yChildren = y.subcategories?.length ? 1
: 0;
        if (xChildren === yChildren) {
          return x.name?.localeCompare(y.name);
        }
        return yChildren - xChildren;
      });
  }
  filterProductsByName(name: string):
IProductModel[] {
    const check = new RegExp(name, 'i');
    return this.sortByNameProducts
      .filter(product => check.test(product.name));
  }
  getCategoriesNames(product: IProductModel):
IProductName[] {
    return product?.categories?.map(id => ({id,
name: this.store.get(id)?.name}));
  }
  getSubcategoriesNames(product:
IProductModel): IProductName[] {
    return product?.subcategories?.map(id => ({id,
name: this.store.get(id)?.name}));
  }
  getProductView(id: number): IProductView {
    const product = this.store.get(id);
    if (!this.isUpdated || !product) {
      return null;
    }
    return {
      ...product,
      imageSrc$:
this.imageStore.getImage(product?.imageId),
      categoryNames:
this.getCategoriesNames(product),
      subcategoryNames:
this.getSubcategoriesNames(product)
    };
  }
}

export class RecipesService {
  private resultRecipes: IRecipeGeneralModel[];
  currRecipes: IRecipeGeneralModel[];
  isUpdated = false;
  private currSortType = RecipeSort.TitleAsc;
  constructor(
    private serverService: ServerHttpService,
  ) { }
  setRecipes(value: IRecipeGeneralModel[], sort:
RecipeSort = this.currSortType) {
```

```
    this.isUpdated = value ? true : false;
    this.resultRecipes = value;
    this.currRecipes = (sort && value) ?
this.sortRecipes(value, sort) : value;
  }
  set sortType(value: RecipeSort) {
    this.sortCurrentRecipes(value);
  }
  get sortType(): RecipeSort {
    return this.currSortType;
  }
  get recipes(): IRecipeGeneralModel[] {
    return this.resultRecipes;
  }
  sortRecipes(recipes: IRecipeGeneralModel[],
sort: RecipeSort = this.currSortType):
IRecipeGeneralModel[] {
    switch (sort) {
      case RecipeSort.TitleAsc: return
this.sortRecipesByTitle(recipes, true);
      case RecipeSort.TitleDesc: return
this.sortRecipesByTitle(recipes, false);
      case RecipeSort.DateAsc: return
this.sortRecipesByDate(recipes, true);
      case RecipeSort.DateDesc: return
this.sortRecipesByDate(recipes, false);
      default: return recipes;
    }
  }
  sortCurrentRecipes(sort: RecipeSort): void {
    this.currSortType = sort;
    this.currRecipes = [
...(this.sortRecipes(this.currRecipes, sort) || []) ];
  }
  sortRecipesByTitle(newRecipes:
IRecipeGeneralModel[], asc = true):
IRecipeGeneralModel[] {
    const recipes = newRecipes?.sort((x, y) => {
      if (!x.title) {
        return 1;
      }
      return x.title?.localeCompare(y.title);
    });
    return asc ? recipes : recipes?.reverse();
  }
  sortRecipesByDate(newRecipes:
IRecipeGeneralModel[], asc = true):
IRecipeGeneralModel[] {
    const recipes = newRecipes?.sort((x, y) => {
      if (!x.time) {
        return 1;
      }
      return x.time > y.time ? 1 : -1;
    });
    return asc ? recipes : recipes?.reverse();
  }
  clearRecipes() {
    this.isUpdated = false;
    this.currRecipes = null;
    this.resultRecipes = null;
  }
  filterRecipesByTitle(name: string) {
    const check = new RegExp(name, 'i');
    this.currRecipes = this.resultRecipes?.filter(r =>
check.test(r.title));
    this.sortType = RecipeSort.TitleAsc;
  }
}

export class ServerHttpService {

  constructor(private http: HttpClient) { }
  getTags(): Observable<string[]> {
    return this.http.get<string[]>(serverUrls.tags);
  }

  // RECIPES

  getRecipe(id: number):
Observable<IRecipeGeneralModel> {
    return
this.http.get<IRecipeGeneralModel>(serverUrls.re
cipes + id);
  }
  getRecipeWithDetails(id: number):
Observable<IRecipeModel> {
    const url =
`${serverUrls.recipes}/${id}/details`;
    return this.http.get<IRecipeModel>(url);
  }

  getRecipeDetailsWithoutImages(id: number):
Observable<IRecipeModel> {
    const url =
`${serverUrls.recipes}/${id}/details/noimages`;
    return this.http.get<IRecipeModel>(url);
  }
```

```
  getRecipesByFilter(filter: IFilterModel):
Observable<IRecipeGeneralModel[]> {
    const url = `${serverUrls.recipes}/filters`;
    return
this.http.post<IRecipeGeneralModel[]>(url,
filter);
  }
  getRecipesByProfileId(id: number):
Observable<IRecipeGeneralModel[]> {
    const url =
`${serverUrls.recipes}/profiles/${id}`;
    return
this.http.get<IRecipeGeneralModel[]>(url);
  }
  getMyRecipes():
Observable<IRecipeGeneralModel[]> {
    const url = `${serverUrls.recipes}/my`;
    return
this.http.get<IRecipeGeneralModel[]>(url);
  }
  getDefaultRecipes():
Observable<IRecipeGeneralModel[]> {
    const url = `${serverUrls.recipes}/default`;
    return
this.http.get<IRecipeGeneralModel[]>(url);
  }
  getRecipesByFilterId(id: number):
Observable<IRecipeGeneralModel[]> {
    const url =
`${serverUrls.recipes}/filters/${id}`;
    return
this.http.get<IRecipeGeneralModel[]>(url);
  }
  getRecipes():
Observable<IRecipeGeneralModel[]> {
    return
this.http.get<IRecipeGeneralModel[]>(serverUrls.
recipes);
  }
  createRecipe(recipe: IRecipeModel):
Observable<number> {
    const url = `${serverUrls.recipes}/create`;
    return this.http.post<number>(url, recipe);
  }
  editRecipe(recipe: IRecipeModel):
Observable<any> {
    const url = `${serverUrls.recipes}/edit`;
    return this.http.put(url, recipe);
  }
  deleteRecipe(id: number): Observable<any> {
    const url = `${serverUrls.recipes}/${id}`;
    return this.http.delete(url);
  }
  deleteMyRecipes(id: number): Observable<any>
{
    const url = `${serverUrls.recipes}/my`;
    return this.http.delete(url);
  }

  // AUTH
  signUp(model: ISignUpModel):
Observable<any> {
    return
this.http.post(`${serverUrls.auth}/signup`,
model);
  }
  isLoginUnique(login: string):
Observable<boolean> {
    const url =
`${serverUrls.auth}/${login}/unique`;
    return this.http.get<boolean>(url);
  }
  signIn(model: ISignInModel):
Observable<ISignInResponse> {
    return
this.http.post<ISignInResponse>(`${serverUrls.au
th}/signin`, model);
  }
  signUpAdmin(model: ISignUpModel):
Observable<any> {
    const url = `${serverUrls.auth}/signup/admin`;
    return this.http.post<any>(url, model);
  }
  changePassword(model:
IChangePasswordModel): Observable<any> {
    const url =
`${serverUrls.auth}/passwords/change`;
    return this.http.post<any>(url, model);
  }

  // PROFILES

  getCurrentProfileWithDetails():
Observable<IProfileModel> {
    const url =
`${serverUrls.profiles}/current/details`;
    return this.http.get<IProfileModel>(url);
  }
```

```
  getCurrentProfile():
Observable<IProfileGeneralModel> {
    const url = `${serverUrls.profiles}/current`;
    return
this.http.get<IProfileGeneralModel>(url);
  }
  getProfileById(id: number):
Observable<IProfileGeneralModel> {
    const url = `${serverUrls.profiles}/${id}`;
    return
this.http.get<IProfileGeneralModel>(url);
  }
  getProfileByIdDetails(id: number):
Observable<IProfileModel> {
    const url =
`${serverUrls.profiles}/${id}/details`;
    return this.http.get<IProfileModel>(url);
  }
  editProfile(profile: IProfileModel):
Observable<any> {
    const url = `${serverUrls.profiles}/edit`;
    return this.http.put(url, profile);
  }

  deleteProfile(id: number): Observable<any> {
    const url = `${serverUrls.profiles}/${id}`;
    return this.http.delete(url);
  }

  // FILTERS
  getFilters(): Observable<IFilterGeneralModel[]>
{
    return
this.http.get<IFilterGeneralModel[]>(serverUrls.fi
lters);
  }
  getFilter(id: number): Observable<IFilterModel>
{
    const url = `${serverUrls.filters}/${id}/details`;
    return this.http.get<IFilterModel>(url);
  }
  getPublicFilters():
Observable<IFilterGeneralModel[]> {
    const url = serverUrls.filters + '/public';
    return
this.http.get<IFilterGeneralModel[]>(url);
  }
  getMyFilters():
Observable<IFilterGeneralModel[]> {
    const url = serverUrls.filters + '/my';
    return
this.http.get<IFilterGeneralModel[]>(url);
  }
  createFilter(filter: IFilterModel):
Observable<number> {
    const url = serverUrls.filters + '/create';
    return this.http.post<number>(url, filter);
  }
  editFilter(filter: IFilterModel):
Observable<number> {
    const url = serverUrls.filters + '/edit';
    return this.http.put<number>(url, filter);
  }
  deleteFilter(id: number): Observable<any> {
    const url = `${serverUrls.filters}/${id}`;
    return this.http.delete(url);
  }

  // IMAGES
  postImage(image: IImageModel):
Observable<number> {
    return
this.http.post<number>(serverUrls.images,
image);
  }
  getImage(id: number):
Observable<IImageModel> {
    const url = `${serverUrls.images}/${id}`;
    return
this.http.get<IImageModel>(url).pipe(tap(x =>
console.log('get image')));
  }
  getImages(images: number[]):
Observable<IImageModel[]> {
    const url = `${serverUrls.images}/get/bulk`;
    return this.http.post<IImageModel[]>(url,
images);
  }
  deleteImage(id: number): Observable<any> {
    const url = `${serverUrls.images}/${id}`;
    return this.http.delete(url);
  }

  // PRODUCTS
  getProducts():
Observable<IProductGeneralModel[]> {
```

```
    return
this.http.get<IProductGeneralModel[]>(serverUrls
.products);
  }
  getProductWithFullDetails(id: number):
Observable<IProductManageModel> {
    const url =
`${serverUrls.products}/${id}/details`;
    return
this.http.get<IProductManageModel>(url);
  }
  getProduct(id: number):
Observable<IProductModel> {
    const url = `${serverUrls.products}/${id}`;
    return this.http.get<IProductModel>(url);
  }
  getProductsWithRelations():
Observable<IProductModel[]> {
    const url =
`${serverUrls.products}/withrelations`;
    return this.http.get<IProductModel[]>(url);
  }
  getProductCategories(productId: number):
Observable<IProductGeneralModel[]> {
    const url =
`${serverUrls.products}/${productId}/categories`;
    return
this.http.get<IProductGeneralModel[]>(url);
  }
  getProductSubcategories(productId: number):
Observable<IProductGeneralModel[]> {
    const url =
`${serverUrls.products}/${productId}/subcategori
es`;
    return
this.http.get<IProductGeneralModel[]>(url);
  }
  createProduct(product: IProductManageModel):
Observable<number> {
    const url = serverUrls.products + '/create';
    return this.http.post<number>(url, product);
  }
  editProduct(product: IProductManageModel):
Observable<any> {
    const url = serverUrls.products + '/edit';
    return this.http.put(url, product);
  }
  deleteProduct(id: number): Observable<any> {
    const url = `${serverUrls.products}/${id}`;
    return this.http.delete(url);
  }
}
```

app.component.html

```
<mat-sidenav-container class="example-
container">
  <mat-sidenav #sidenav mode="over"
[(opened)]="opened">
    <mat-toolbar class="h-64" color="primary">
      МЕНЮ
      <button mat-icon-button
(click)="sidenav.close()"><mat-icon>close</mat-
icon></button>
    </mat-toolbar>
    <mat-nav-list>
      <ng-container *ngFor="let item of
menuItems">
        <a mat-list-item
          *ngIf="!item.visible || item.visible()"
          [routerLink]="[item.routerLink]"
          routerLinkActive

#routerLinkActiveInstance="routerLinkActive"
          [class.list-item-
active]="routerLinkActiveInstance.isActive"
          (click)="sidenav.close()"
        >{{ item.label }}</a>
      </ng-container>
      <a mat-list-item
*ngIf="authService.isAuthorized"
(click)="authService.logout()">
        Вихід</a>
    </mat-nav-list>
  </mat-sidenav>

  <mat-sidenav-content>
    <div class="content pt-64">
      <app-header class="h-64"
(toggleDrawer)="sidenav.toggle()"></app-
header>
      <simple-notifications></simple-notifications>
      <router-outlet></router-outlet>
    </div>
  </mat-sidenav-content>
</mat-sidenav-container>
```

tags-search-section.component.html

```
<div class="TagsSection">
```

```
  <mat-form-field class="required">
    <mat-chip-list #requiredList>
      <mat-chip *ngFor="let tag of requiredTags"
selectable="true" removable="true"
(removed)="removeTag(tag)">
        {{ tag }}<mat-icon
matChipRemove>cancel</mat-icon>
      </mat-chip>
      <input
        placeholder="Обов'язкові теги"
        #requiredTagInput
        [formControl]="requiredTagsCtrl"
        [matAutocomplete]="autoRequired"
        [matChipInputFor]="requiredList"
[matChipInputSeparatorKeyCodes]="separatorKe
ysCodes"

(matChipInputTokenEnd)="addRequiredTag($eve
nt); requiredTagInput.value = null">
    </mat-chip-list>
    <button mat-icon-button matSuffix
(click)="setRequiredTag(requiredTagInput.value);
requiredTagInput.value = null">
      <mat-icon>add</mat-icon>
    </button>
    <mat-autocomplete
#autoRequired="matAutocomplete"
(optionSelected)="selectRequired($event)">
      <mat-option *ngFor="let tag of
filteredRequiredTags$ | async" [value]="tag">
        {{ tag?.toLocaleLowerCase() }}
      </mat-option>
    </mat-autocomplete>
  </mat-form-field>
  <mat-form-field class="forbidden">
    <mat-chip-list #forbiddenList>
      <mat-chip *ngFor="let tag of forbiddenTags"
selectable="true" removable="true"
(removed)="removeTag(tag)">
        {{ tag }}<mat-icon
matChipRemove>cancel</mat-icon>
      </mat-chip>
      <input
        placeholder="Заборонені теги"
        #forbiddenTagInput
        [formControl]="forbiddenTagsCtrl"
        [matAutocomplete]="autoForbidden"
        [matChipInputFor]="forbiddenList"
[matChipInputSeparatorKeyCodes]="separatorKeys
Codes"

(matChipInputTokenEnd)="addForbiddenTag($ev
ent); forbiddenTagInput.value = null">
    </mat-chip-list>
    <button mat-icon-button matSuffix
(click)="setForbiddenTag(forbiddenTagInput.valu
e); forbiddenTagInput.value = null">
      <mat-icon>add</mat-icon>
    </button>
    <mat-autocomplete
#autoForbidden="matAutocomplete"
(optionSelected)="selectForbidden($event)">
      <mat-option *ngFor="let tag of
filteredForbiddenTags$ | async" [value]="tag">
      {{ tag?.toLocaleLowerCase() }}
      </mat-option>
    </mat-autocomplete>
  </mat-form-field>
</div>
```

recipe-editor.component.html

```
<mat-toolbar color="primary">
  <button mat-icon-button
    routerLink="/recipes/my"
    matTooltip="повернення до списку
рецептів">
    <mat-icon class="prev-page-
icon">arrow_back</mat-icon>
  </button>
  <h1>{{ isNew ? 'Створення рецепту' :
'Редагування рецепту' }}</h1>
  <!-- <button mat-icon-button
    *ngIf="currRecipe"
    matTooltip="перегляд прев'ю продукту"
    (click)="openCurrProductPreview()">
    <mat-icon>visibility</mat-icon>
  </button> -->
  <button mat-icon-button
    matTooltip="збереження рецепту"
    *ngIf="currRecipe &&
!recipeFormRef?.invalid"
    (click)="isNew ? onCreate() : onSave()">
    <mat-icon>save</mat-icon>
  </button>
</mat-toolbar>
<div class="RecipeEditor">
```

```
  <mat-card class="RecipeContainer"
*ngIf="this.currRecipe; else loading">
    <form #recipeForm="ngForm">
      <mat-form-field appearance="outline"
class="">
        <mat-label>Назва рецепту</mat-label>
        <input matInput name="recipeTitle"
maxlength="256" required
          pattern=".*\S+.*"
[(ngModel)]="currRecipe.title">
      </mat-form-field>
      <mat-form-field appearance="outline"
class="">
        <mat-label>Опис рецепту</mat-label>
        <textarea matInput
name="recipeDescription" maxlength="1024"

[(ngModel)]="currRecipe.description"></textarea
>
      </mat-form-field>
      <div class="numbers">
        <mat-form-field appearance="outline"
class="">
          <mat-label>Кількість порцій</mat-label>
          <input matInput type="text" #portions
name="portions" pattern="[0-9]*"
           [ngModel]="currRecipe.portions"
(change)="currRecipe.portions =
convertToNumber(portions)">
        </mat-form-field>
        <mat-form-field appearance="outline">
          <mat-label>Вага однієї порції</mat-label>
          <input matInput #servingWeight
[ngModel]="currRecipe.servingWeight"
name="servingWeight"
           type="text" pattern="[0-9.]*"
(change)="currRecipe.servingWeight =
convertToNumber(servingWeight)">
          <span matSuffix>г</span>
        </mat-form-field>
        <mat-form-field appearance="outline"
class="">
          <mat-label>Час приготування</mat-label>
          <input matInput type="text" #duration
name="duration" pattern="[0-9]*"
           [ngModel]="currRecipe.duration"
(change)="currRecipe.duration =
convertToNumber(duration)">
          <span matSuffix>хв</span>
        </mat-form-field>
        <mat-form-field matTooltipPosition="above"
appearance="outline" matTooltip="калорійність
у 100 г готової страви">
          <mat-label>Калорійність (на 100 г)</mat-
label>
          <input matInput name="calories" #cals
type="text" pattern="[0-9.]*"
           [ngModel]="currRecipe.calories"
(change)="currRecipe.calories =
convertToNumber(cals)">
          <span matSuffix>ккал</span>
        </mat-form-field>
      </div>
      <section class="main-image">
        <input type="file" class="hide"
(change)="uploadRecipeImage($event)"
#recipeImage>
        <button (click)="recipeImage.click()"
class="indent" mat-stroked-button
color="primary">
          Завантажити зображення</button>
        <mat-card class="imagePreview"
*ngIf="currRecipe.image?.data as data">
          <mat-card-subtitle>Прев'ю
зображення</mat-card-subtitle>
          <img [src]="'' +
data" />
          <button mat-stroked-button color="warn"
(click)="removeRecipeImage()">Видалити
зображення</button>
        </mat-card>
      </section>

      <mat-form-field class="tags">
        <mat-chip-list #tagsList>
          <mat-chip class="mat-chip-selected"
color="accent" (removed)="removeTag(index)"
           *ngFor="let tag of currRecipe.tags || []; let
index = index">
            {{ tag?.tag }}<mat-icon
matChipRemove>cancel</mat-icon>
          </mat-chip>
          <input
            placeholder="Додати тег"
            #tagInput matInput
            maxLength="128"
            [formControl]="tagsCtrl"
            [matAutocomplete]="autoRequired"
```

```
        [matChipInputFor]="tagsList"

[matChipInputSeparatorKeyCodes]="separatorKeysCodes"

(matChipInputTokenEnd)="addTag($event)">
      </mat-chip-list>
      <button mat-icon-button matSuffix (click)="setTag(tagInput.value); tagInput.value = null">
        <mat-icon>add</mat-icon>
      </button>
      <mat-autocomplete #autoRequired="matAutocomplete" (optionSelected)="selectTag($event)">
        <mat-option *ngFor="let tag of filteredTags$ | async" [value]="tag">
          {{ tag?.toLocaleLowerCase() }}
        </mat-option>
      </mat-autocomplete>
    </mat-form-field>

    <section class="Ingredients">
      <div class="Ingredients-list">
        <mat-card *ngFor="let ingredient of currRecipe.ingredients; let index = index">
          <mat-card-header>
            <button mat-icon-button color="primary"
              *ngIf="ingredient?.productId as id"
              (click)="openProduct(id)">
              <mat-icon matListIcon>info</mat-icon>
            </button>
            <mat-card-subtitle>Інгредієнт</mat-card-subtitle>
          </mat-card-header>
          <mat-form-field appearance="outline">
            <mat-label>Продукт</mat-label>
            <mat-select name="productSelect-{{index}}"
              [(ngModel)]="ingredient.productId"
              placeholder="Оберіть продукт">
              <ngx-mat-select-search ngModel name="productFilter"
(ngModelChange)="filterMyProducts($event)"
                [placeholderLabel]="'Пошук продуктів...'"
[noEntriesFoundLabel]="'Продукти не знайдено'">
              </ngx-mat-select-search>
              <mat-option *ngFor="let product of filteredProducts" [matTooltip]="product.name"
[value]="product.id">
                {{ product.name }}</mat-option>
            </mat-select>
          </mat-form-field>
          <section class="weight">
            <mat-form-field appearance="outline" matTooltip="Кількість"
matTooltipPosition="above">
              <mat-label matTooltip="Кількість">Кількість</mat-label>
              <input matInput #weight name="weight-{{index}}" pattern="[0-9.]*"
maxlength="10" required
                [ngModel]="ingredient.weight"
(change)="ingredient.weight = convertToNumber(weight)">
            </mat-form-field>
            <mat-form-field appearance="outline" matTooltip="Одиниця виміру"
matTooltipPosition="above">
              <mat-label>Одиниця виміру</mat-label>
              <input matInput required #measurement
                name="measurement-{{index}}"
                maxlength="32"

[ngModel]="ingredient.weightMeasurement"
                (input)="setMeasurement(ingredient, measurement.value)"

[matTooltip]="getMeasurementTitle(ingredient.weightMeasurement)"

[matAutocomplete]="autoMeasurements">
              <mat-autocomplete #autoMeasurements="matAutocomplete"

(optionSelected)="setMeasurement(ingredient, $event?.option.value)">
                <mat-option *ngFor="let measurement of measurements"
                  [matTooltip]="measurement.title || null"
                  [value]="measurement.value">{{ measurement.value }}</mat-option>
              </mat-autocomplete>
```

```
          </mat-form-field>
        </section>
        <mat-form-field
*ngIf="ingredient.weightInGrams ||
ingredient.weightMeasurement !== 'г'"
          appearance="outline">
          <mat-label>Приблизна вага в
грамах</mat-label>
          <input matInput #weightInGrams
pattern="[0-9.]*"
[ngModel]="ingredient.weightInGrams"
            name="weightInGrams-{{index}}"
(change)="ingredient.weightInGrams =
convertToNumber(weightInGrams)">
          <span matSuffix>г</span>
        </mat-form-field>
        <mat-form-field appearance="outline"
class="">
          <mat-label>Примітка</mat-label>
          <textarea matInput
name='ingredientName-{{index}}'

[(ngModel)]="ingredient.note"></textarea>
        </mat-form-field>
        <mat-checkbox name="necessity-
{{index}}" color="primary"
[(ngModel)]="ingredient.necessity">
          Цей продукт є обов'язковим</mat-
checkbox>
        <button
(click)="removeIngredient(index)" class="indent"
mat-stroked-button
color="warn">Видалити</button>
        </mat-card>
      </div>
      <button (click)="addIngredient()"
class="ingredient-button" mat-stroked-button
color="primary">Додати інгредієнт</button>
      <mat-form-field appearance="outline">
        <mat-label>Опис інгредієнтів</mat-label>
        <textarea matInput
name="ingredientDescription" maxlength="512"
[(ngModel)]="currRecipe.ingredientsDescription"
></textarea>
      </mat-form-field>
    </section>
    <mat-card *ngFor="let step of
currRecipe.steps; let index = index">
      <mat-card-subtitle>Крок рецепту</mat-
card-subtitle>
      <mat-form-field appearance="outline">
        <mat-label>Назва</mat-label>
        <input matInput name="stepTitle-
{{index}}" maxlength="128"
          [(ngModel)]="step.title">
      </mat-form-field>
      <mat-form-field appearance="outline">
        <mat-label>Опис</mat-label>
        <textarea matInput name="stepDescription-
{{index}}" maxlength="2048"

[(ngModel)]="step.description"></textarea>
      </mat-form-field>
      <input type="file" class="hide"
(change)="uploadStepImage(index, $event)"
#stepImage>
      <button (click)="stepImage.click()"
class="indent" mat-stroked-button
color="primary">Завантажити
зображення</button>
      <mat-card  class="imagePreview"
*ngIf="step.image?.data as stepImage">
        <mat-card-subtitle>Прев'ю
зображення</mat-card-subtitle>
        <img [src]="'' +
stepImage" />
        <button mat-stroked-button color="warn"
(click)="removeStepImage(index)">Видалити
зображення</button>
      </mat-card>
      <button (click)="removeStep(index)" mat-
stroked-button color="warn">Видалити
крок</button>
    </mat-card>
    <button (click)="addStep()" class="indent"
mat-stroked-button color="primary">Додати
крок</button>
    <mat-divider></mat-divider>
    <div class="actions">
      <button
        [disabled]="recipeForm.invalid"
        mat-raised-button
        color="primary"
        class="edit-button"
        *ngIf="!isNew"
        (click)="onSave()">
        Зберегти рецепт</button>
```

```
        <button
          [disabled]="recipeForm.invalid"
          mat-raised-button
          [color]="isNew ? 'primary' : 'basic'"
          class="save-button"
          (click)="onCreate()">
          Створити новий рецепт</button>
      </div>
    </form>
  </mat-card>
</div>
<ng-template #loading>
  <div class="spinner-container">
    <mat-spinner></mat-spinner>
    <span>Завантаження рецепту...</span>
  </div>
</ng-template>
```

theme.scss

```
@import '~@angular/material/theming';
@include mat-core();
@mixin createTheme($primaryColor,
$primaryNum, $accentColor, $accentNum) {
  $primary: mat-palette($primaryColor,
$primaryNum);
  $accent: mat-palette($accentColor,
$accentNum);
  @include angular-material-theme(mat-light-
theme($primary, $accent));
  .list-item-active { color: mat-color($accent,
darker) !important; }
  &.mat-menu-item.theme-button {
    @include angular-material-theme(mat-light-
theme($primary, $accent));
  }
}
.theme {
  &-cyan {
    &-900-yellow {
      @include createTheme($mat-cyan, 900, $mat-
yellow, 200);
    }
    &-900-lime {
      @include createTheme($mat-cyan, 900, $mat-
lime, 500);
    }
    &-800-amber {
      @include createTheme($mat-cyan, 800, $mat-
amber, 400);
    }
  }
  &-teal {
    &-900-green {
      @include createTheme($mat-teal, 900, $mat-
green, A700);
    }
    &-700-light-green {
      @include createTheme($mat-teal, 700, $mat-
light-green, A200);
    }
  }
  &-pink {
    &-900-amber {
      @include createTheme($mat-pink, 900, $mat-
amber, 200);
    }
    &-700-yellow {
      @include createTheme($mat-pink, 700, $mat-
yellow, 300);
    }
  &-deep-orange-A700-amber {
    @include createTheme($mat-deep-orange,
A700, $mat-amber, A200);
  }
  &-orange-800-yellow {
    @include createTheme($mat-orange, 800,
$mat-yellow, 500);
  }
  &-yellow {
    &-500-orange {
      @include createTheme($mat-yellow, 500,
$mat-orange, 600);
    }
  }
  &-brown {
    &-800-orange {
      @include createTheme($mat-brown, 800,
$mat-orange, 200);
    }
    &-green {
    &-700-light-green {
      @include createTheme($mat-green, 700,
$mat-light-green, A200);
    }
    &-800-amber {
      @include createTheme($mat-green, 800,
$mat-amber, A200);
    }
```

```
  }
  &-lime-900-yellow {
    @include createTheme($mat-lime, 900, $mat-
yellow, 300);
  }
  &-light-green-700-lime {
    @include createTheme($mat-light-green, 700,
$mat-lime, A200);
  }
  &-purple-300-lime {
    @include createTheme($mat-purple, 300,
$mat-lime, 300);
  }
  @include createTheme($mat-green, 700, $mat-
light-green, A200);
}

export class
IngredientsSearchSectionComponent
implements OnInit, OnDestroy {
  displayedColumns: string[] = [
    'subcategories', 'name', 'requiredSelect',
'notRequiredSelect', 'calories',
    'fats', 'squirrels', 'carbohydrates', 'sugar',
  ];
  productsSource:
MatTableDataSource<IFilterProduct> = null;

  @ViewChild(MatSort) sort: MatSort;
 paginator: MatPaginator;

  rootProductId: number;
  subscriptions = new Subscription();
  columnHint = 'вміст в 100 грамах продукту';
  isLoaded = false;

  constructor(
    private filterService: FiltersService,
    private productsService: ProductsService,
    private dialog: MatDialog,
  ) { }
  ngOnInit(): void {
    if (this.filterService.needUpdate) {
      this.filterService.updateProducts();
    }
this.subscriptions.add(this.filterService.currRootP
roductChanged$
      .subscribe(productId => this.rootProductId =
productId));
    const currProducts$ =
this.filterService.currProductsChanged$.pipe(
      tap(_ => this.productsSource = null),
      delay(this.isLoaded ? 100 : 300),
      tap(products => {
        this.productsSource = new
MatTableDataSource(products?.map(id =>
this.getProduct(id)) || []);
        this.productsSource.filterPredicate = (data:
IFilterProduct, name: string) => new
RegExp(name, 'i').test(data.name);
        this.isLoaded = true;
      }),
      delay(100),
      tap(_ => this.productsSource.sort = this.sort)
    );

this.subscriptions.add(currProducts$.subscribe());
  }
  getFilteredProducts(name: string):
IProductModel[] {
    return
this.productsService.filterProductsByName(name)
;
  }
  ngOnDestroy() {
    this.subscriptions.unsubscribe();
  }
  get areAllProducts(): boolean {
    return !(this.rootProductId >= 0);
  }
  changeRootProduct(id: number) {
    this.filterService.setRootProduct(id);
  }
  goBack() {
    this.filterService.returnBack();
  }
  getProduct(id: number): ExpandedProduct {
    return this.filterService.getProduct(id);
  }
  searchByName(name: string) {
    this.productsSource.filter =
name?.toLocaleLowerCase();
  }
  searchAllByName(name: string) {

this.filterService.setCurrProductsByName(name);
  }
  toggleByAvailableProducts(value: boolean) {
```

```
    if (value !== this.byAvailable) {

this.filterService.changeByAvailableProducts(!!va
lue);
    }
  }
  toggleByExactProducts(value: boolean) {
    if (value !== this.byExactProducts) {
      this.filterService.currFilter.onlyProducts =
value;
    }
  }
  get byAvailable(): boolean {
    return
this.filterService.currFilter.byAvailableProducts;
  }
  get byExactProducts(): boolean {
    return this.filterService.currFilter.onlyProducts;
  }
  isNotRequired(product: IFilterProduct): boolean
{
    return
this.filterService.isProductSelected(product.id,
ProductNecessity.NotRequired);
  }
  isRequired(product: IFilterProduct): boolean {
    return
this.filterService.isProductSelected(product.id,
ProductNecessity.Required);
  }
  get isAllRequired(): boolean {
    return
this.filterService.areAllCurrentProductsSelected(P
roductNecessity.Required);
  }
  get isAllNotRequired(): boolean {
    return
this.filterService.areAllCurrentProductsSelected(P
roductNecessity.NotRequired);
  }
  get areAnyRequired(): boolean {
    return
this.filterService.areNotAllCurrentProductsSelect
ed(ProductNecessity.Required);
  }
  get areAnyNotRequired(): boolean {
    return
this.filterService.areNotAllCurrentProductsSelect
ed(ProductNecessity.NotRequired);
  }
  onGroupRequiredChecked(event:
MatCheckboxChange) {
    this.toggleAllProducts(event?.checked,
ProductNecessity.Required);
  }
  onGroupNotRequiredChecked(event:
MatCheckboxChange) {
    this.toggleAllProducts(event?.checked,
ProductNecessity.NotRequired);
  }
  onRequiredChecked(event:
MatCheckboxChange, product: IFilterProduct) {

this.filterService.setProductNecessity(product.id,
event?.checked
      ? ProductNecessity.Required :
ProductNecessity.Undefined);
  }
  onNotRequiredChecked(event:
MatCheckboxChange, product: IFilterProduct) {

this.filterService.setProductNecessity(product.id,
event?.checked
      ? ProductNecessity.NotRequired :
ProductNecessity.Undefined);
  }
  private toggleAllProducts(checked: boolean,
necessity: ProductNecessity) {
    if (checked) {

this.filterService.selectAllCurrentProducts(necessi
ty);
    }
    else {
this.filterService.unselectAllCurrentProductsByN
ecessity(necessity);
    }
  }
  get notRequiredHint(): string {
    return this.byAvailable ? 'опціональні
продукти' : 'заборонені продукти';
  }
  get notRequireProductHint(): string {
    return this.byAvailable ? 'опціональний
продукт' : 'заборонений продукт';
  }
  changeProductStatus(productId: number) {
    this.filterService.selectProduct(productId);
```

```
  }
  statusLabel(necessity: ProductNecessity): string
{
    switch (necessity) {
      case ProductNecessity.Required: return
'обов\'язковий';
      case ProductNecessity.NotRequired:
        return this.byAvailable ? 'опціональний' :
'заборонений';
      default: return 'невизначений';
    }
  }
  statusColor(necessity: ProductNecessity): string
{
    switch (necessity) {
      case ProductNecessity.Required: return
'#aefda9';
      case ProductNecessity.NotRequired:
        return this.byAvailable ? '#fdfaa9' : '#ffafaf';
      default: return  '#e2e2e2';
    }
  }
  openProductDialog(productId: number): void {

this.dialog.open(ProductDetailsDialogComponent
, {
      width: '500px',
      data:
this.filterService.getProductViewDetails(productI
d)
    }).afterClosed().subscribe();
  }
}

export class ProductListComponent
implements OnInit, OnDestroy {
  currProducts: IProductView[];
  currProduct: IProductView;
  breadCrumbs: number[] = [];
  subscriptions = new Subscription();
  productsForSearch: IProductModel[] =
this.productStore.sortByNameProducts;
  openDetails = false;
  get products(): Map<number, IProductView> {
    return this.productStore.store;
  }
  constructor(
    private route: ActivatedRoute,
    private router: Router,
    public auth: AuthService,
    private server: ServerHttpService,
    private notifications: NotificationsService,
    public dialog: MatDialog,
    private imageStore: ImagesService,
    private productStore: ProductsService,
  ) {
  }
  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }
  ngOnInit(): void {
    this.breadCrumbs = [];
    this.subscriptions.add(this.route.params?.pipe(
      tap(_ => {
        this.openDetails = false;
        this.currProducts = null;
      }),
      delay(50)
    ).subscribe(x => {
      if (this.productStore.isUpdated) {
        this.setRootProduct(+x.id);
        this.productsForSearch =
this.productStore.sortByNameProducts;
      } else {
        this.updateProducts(+x.id);
      }
    }));
  }
  private updateProducts(productId: number) {
this.subscriptions.add(this.productStore.updatePro
ducts().subscribe(products => {
      if (products) {
        this.setRootProduct(productId, true);
      }
      this.productsForSearch =
this.productStore.sortByNameProducts;
    }));
  }
  deleteProduct(productId: number, name: string)
{
    this.productStore.deleteProduct(productId,
name)?.subscribe(_ => {
      this.currProducts = null;
      this.updateProducts(this.currProduct?.id);
      this.createNotification('Продукт видалено');
    },
    error => {
```

```
      this.createNotification('Продукт не видалено', NotificationType.Error, 'Помилка під час видалення продукту');
    });
  }
  createNotification(title: string, type = NotificationType.Success, content: string = '') {
    this.notifications.create(title, content, type, {
      timeOut: 3000,
      showProgressBar: true,
      pauseOnHover: true,
      clickToClose: true
    });
  }
  navigateToProduct(productId: number, saveBreadCrumb: boolean) {
    if (productId !== this.currProduct?.id) {
      if (saveBreadCrumb) {
        this.breadCrumbs.push(this.currProduct?.id);
      }
      this.router.navigate(['products/list', productId || 0]);
    }
  }
  onSelectProduct($event: MatAutocompleteSelectedEvent) {
    this.navigateToProduct($event?.option?.value, true);
  }

  filterProductsByName(name: string) {
    this.productsForSearch = this.productStore.filterProductsByName(name);
  }
  private setRootProduct(rootProduct?: number, force: boolean = false) {
    if (rootProduct === this.currProduct?.id && !force) {
      return;
    }
    if (!rootProduct) {
      this.breadCrumbs = [];
    }
    let products = this.productStore.products;
    this.currProduct = this.products.get(rootProduct);
    if (this.currProduct) {
      this.currProduct.imageSrc$ = this.imageStore.getImage(this.currProduct.imageId);
      this.currProduct.categoryNames = this.productStore.getCategoriesNames(this.currProduct);
      products = this.currProduct?.subcategories.map(id => this.products.get(id));
      this.openDetails = !products?.length;
    }
    else {
      products = products.filter(x => !x.categories?.length);
    }
    this.currProducts = this.productStore.sortProductsByNameAndType(products)
      .map(product => ({
        ...product,
        imageSrc$: this.imageStore.getImage(product?.imageId),
        categoryNames: this.productStore.getCategoriesNames(product)
      }));
  }
  returnBack() {

    this.navigateToProduct(this.breadCrumbs.pop(), false);
  }
}
```

product-list.component.scss

```
:host {
  width: 100%;
}
strong {
  font-weight: 500;
}
h1 + a {
  margin-left: 8px;
}
.header {
  min-height: 64px;
  height: fit-content;
  flex-wrap: wrap;
  justify-content: space-between;
  &-actions {
```

```
      display: flex;
      align-items: center;
      min-height: 64px;
      .first-page-icon {
        font-size: 32px;
      }
      .prev-page-icon {
        font-size: 26px;
      }
      h1 {
        word-break: break-word;
        white-space: normal;
      }
      button + h1 {
        margin-left: 8px;
      }
    }
  }
  .products-search {
    margin-bottom: -16px;
    line-height: 18px;
    font-size: 16px;
    button {
      font-size: 16px;
      height: 40px;
      mat-icon {
        margin-left: 8px;
      }
    }
  }
  .details {
    display: flex;
    background-color: white;
    flex-direction: column;
    .description {
      padding: 0px 16px 8px 16px;
    }
    mat-expansion-panel {
      border-radius: 0;
    }
    .main {
      padding: 16px;
      padding-top: 0px;
      padding-right: 0px;
      display: flex;
      flex-direction: row;
      flex-wrap: wrap;
      justify-content: center;
      & > div {
        margin-right: 16px;
      }
      .image-container {
        margin-top: 16px;
        overflow: hidden;
        border-radius: 8px;
        margin-right: 16px;
        max-height: 501px;
        img {
          max-height: 501px;
          border-radius: 8px;
          max-width: 100%;
        }
      }
    }
    .composition {
      &-container {
        padding: 16px;
        border-radius: 8px;
        border: 1px solid #c2c1c1;
        margin-top: 16px;
        flex-basis: 232px;
        flex-grow: 1;
        .p > span {
          font-size: 14px;
        }
        .label {
          font-size: 20px;
          word-break: break-word;
          color: grey;
          font-weight: 300;
        }
      }
      &-data {
        display: flex;
        flex-wrap: wrap;
        .col {
          width: 200px;
          &:first-of-type {
            margin-right: 10%;
          }
        }
      }
    }
  }
  .ProductsList {
    &.container {
      width: 100%;
      display: flex;
```

```
    justify-content: center;
    margin-bottom: 16px;
  }
  .content {
    width: 1600px;
    .grid {
      margin-top: 16px;
      display: flex;
      flex-direction: row;
      flex-wrap: wrap;
      justify-content: space-evenly;
      img {
        width: 100%;
        margin: 0px;
      }
      mat-card {
        display: flex;
        flex-direction: column;
        justify-content: space-between;
        mat-card-title {
          word-break: break-word;
        }
        mat-card-header {
          margin-left: -16px;
          margin-right: -16px;
        }
        .card-content {
          display: flex;
          flex-direction: column;
          flex-grow: 2;
          justify-content: space-between;

          mat-card-content {
            margin-bottom: 0px;
            display: flex;
            flex-direction: column;
            justify-content: space-between;
            flex-grow: 2;
            .image-outer {
              display: flex;
              flex-grow: 2;
              flex-direction: column;
              justify-content: center;
            }
          }
        }
        mat-card-actions {
          display: flex;
          justify-content: space-between;
          margin-bottom: 0;
          padding-bottom: 0;
          button {
            margin: 0px;
          }
        }
      }
    }
    @media screen and (max-width: 532px) {
      .grid .product-card {
        width: 100%;
        mat-card-content {
          display: flex;
          flex-direction: column;
          align-items: center;
        }
        .imageContainer {
          max-width: 232px;
        }
      }
    }
    .product-card {
      margin: 8px;
      width: 200px;
      background-color: white;
      mat-card-header {
        color: black;
      }
      mat-divider {
        color: rgba(0, 0, 0, 0.87);
        border-top-color: rgba(0, 0, 0, 0.12);
      }
      mat-card-actions {
        a, button {
          border-color: rgba(0, 0, 0, 0.12);
        }
      }
    }
    .imageContainer {
      flex-shrink: 0;
      max-height: 232px;
      overflow: hidden;
      margin: 0 -16px;
    }
    .categories {
      margin-top: 4px;
      mat-chip {
        height: auto;
        word-break: break-word;
```

```
        text-align: center;
      }
    }
  }
}
mat-panel-title {
  font-size: 20px;
  align-items: center;
}
mat-panel-description {
  line-height: 24px;
}
```

export class AuthService {

```
  readonly tokenChanged$:
BehaviorSubject<string> = new
BehaviorSubject(null);
  get isAuthorized(): boolean {
    return !!this.getToken();
  }
  get isAdmin(): boolean {
    return !!localStorage.getItem('admin') &&
this.isAuthorized;
  }
  get name(): string {
    return localStorage.getItem('name');
  }
  getToken(): string {
    return localStorage.getItem('token');
  }
  setToken(token: string): void {
    if (token) {
      localStorage.setItem('token', token);
    }
    else {
      localStorage.removeItem('token');
    }
    this.tokenChanged$.next(token);
  }

  setName(name: string): void {
    if (name) {
      localStorage.setItem('name', name);
    }
    else {
      localStorage.removeItem('name');
    }
    this.tokenChanged$.next(name);
  }
  setIsAdmin(isAdmin: boolean): void {
    if (isAdmin) {
      localStorage.setItem('admin', 'true');
    }
    else {
      localStorage.removeItem('admin');
    }
  }
  clearToken(): void {
    localStorage.removeItem('token');
    this.tokenChanged$.next(null);
  }
  signIn(model: ISignInResponse) {
    this.setIsAdmin(model?.isAdmin);
    this.setToken(model?.token);
    this.setName(model?.displayName);
  }
  logout(): void {
    this.clearToken();
    localStorage.removeItem('admin');
    localStorage.removeItem('name');
  }
}
```

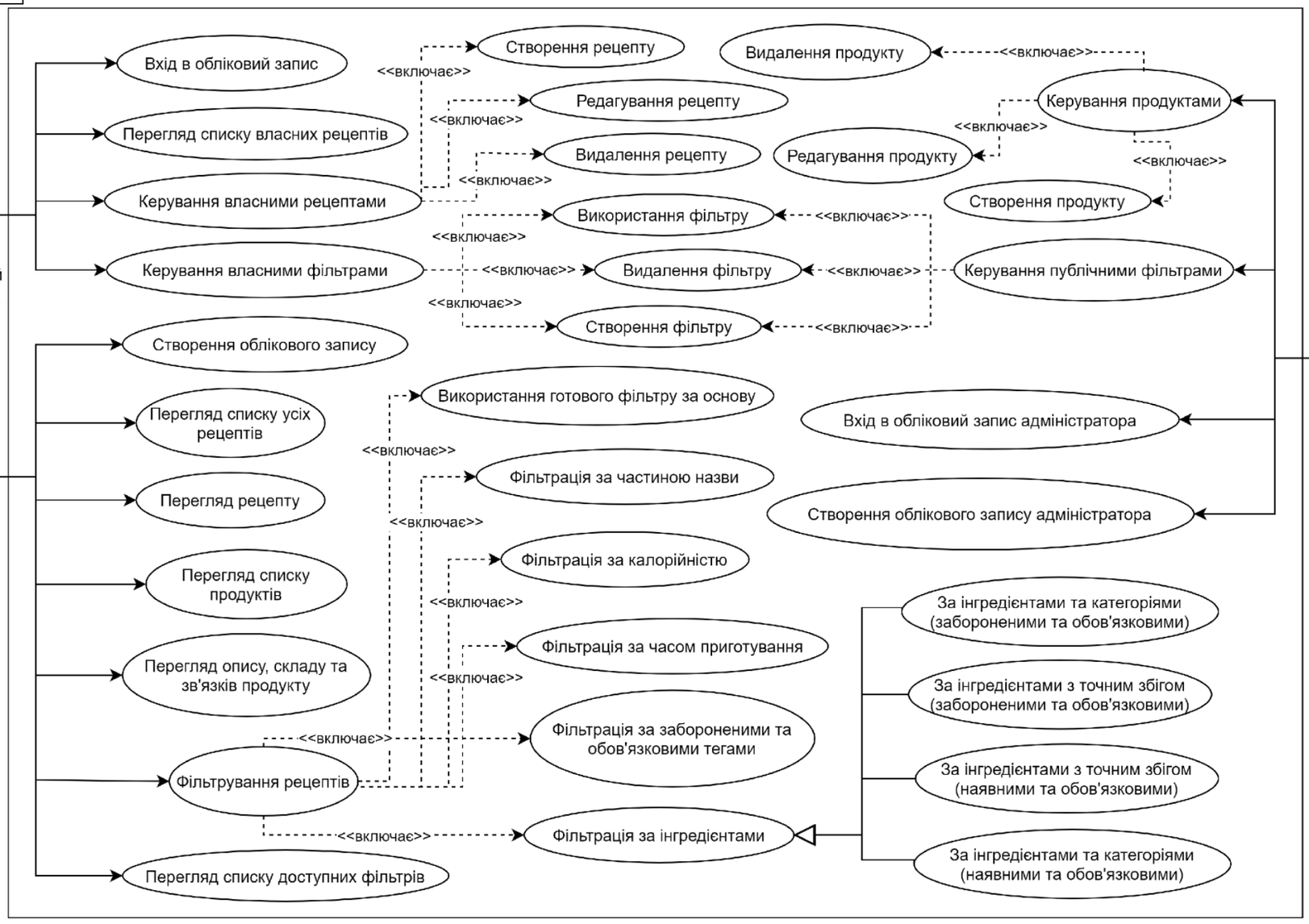
sign-up.component.html

```
<div class="SignUp">
  <h2>Реєстрація</h2>
  <form [formGroup]="formModel"
autocomplete="off">
    <mat-form-field>
      <mat-label>Логін</mat-label>
      <input matInput formControlName="login"
required (blur)="checkLogin()">
      <mat-error
*ngIf="loginCtrl.hasError('unique')">Користувач
з таким логіном вже існує</mat-error>
    </mat-form-field>
    <mat-form-field>
      <mat-label>Електронна пошта</mat-label>
      <input matInput formControlName="email">
    </mat-form-field>
    <mat-form-field>
      <mat-label>Повне ім'я</mat-label>
      <input matInput
formControlName="fullName"
autocomplete="off">
    </mat-form-field>
    <div formGroupName="passwords"
class="passwords">
```

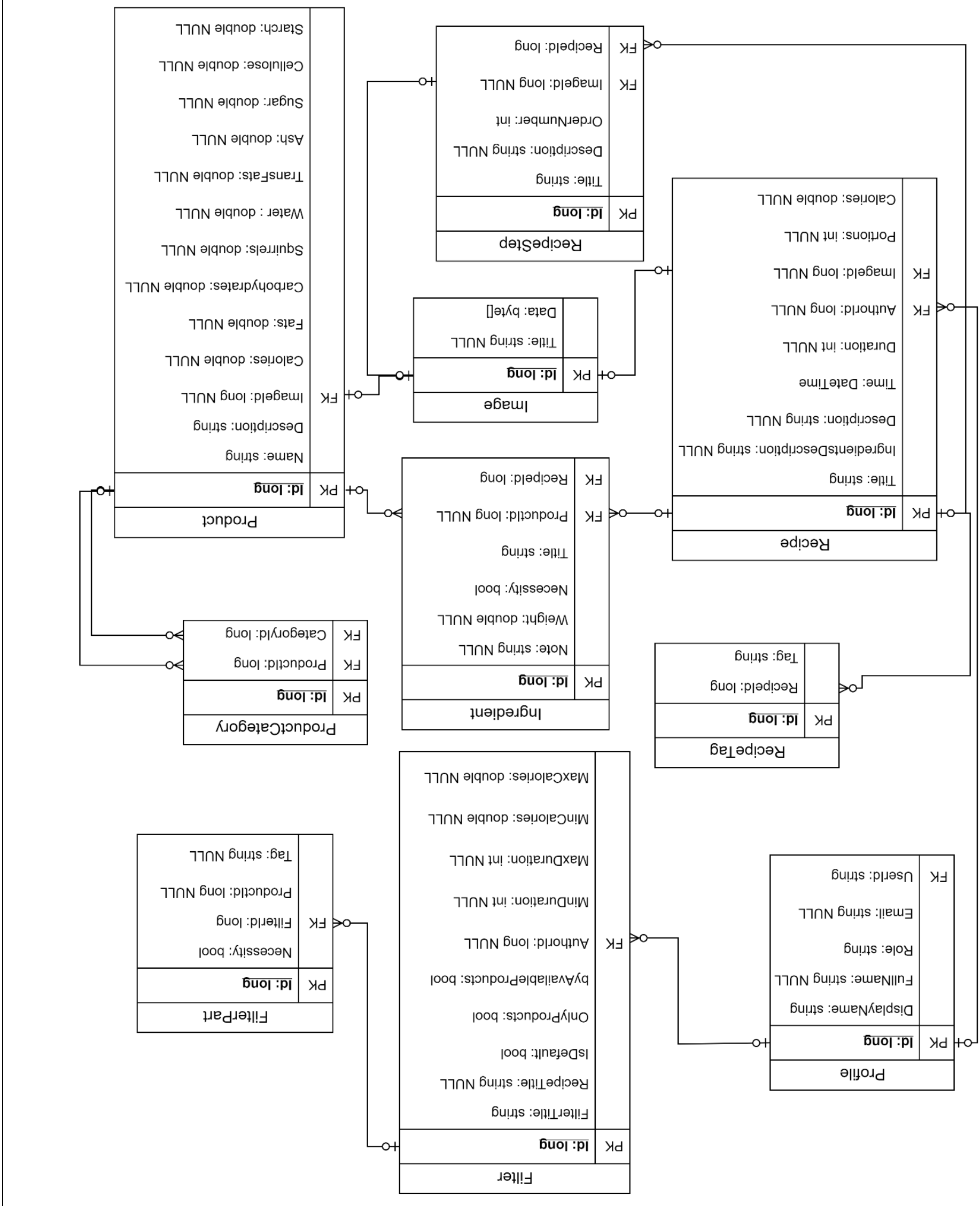
```
      <mat-form-field>
       <mat-label>Пароль</mat-label>
       <input matInput [type]="hidePswrd ?
'password' : 'text'" formControlName="password"
required autocomplete="new-password">
       <button [disabled]="passwordCtrl.value ?
null : true" class="visibility"
        color="primary" mat-icon-button matSuffix
(click)="hidePswrd = !hidePswrd">
        <mat-icon>{{ hidePswrd ? 'visibility_off' :
'visibility' }}</mat-icon>
       </button>
      </mat-form-field>
      <mat-form-field>
       <mat-label>Підтвердження паролю</mat-
label>
       <input type="password" matInput required
        formControlName="confirmPassword"
autocomplete="new-password">
       <mat-error
*ngIf="confirmPswrdCtrl.hasError('passwordMis
match')">Паролі не збігаються</mat-error>
      </mat-form-field>
    </div>
    <div class="actions">
     <button mat-raised-button
      color="primary"
      [disabled]="!formModel.valid"
      (click)="onSubmit()">
      Реєстрація
     </button>
     <button mat-raised-button
      class="submit-admin"
      color="accent"
      *ngIf="auth.isAdmin"
      [disabled]="!formModel.valid"
      (click)="onSubmit(true)">
      Реєстрація адміністратора
     </button>
    </div>
   </form>
</div>

export class SignUpComponent implements
OnInit, OnDestroy {
  checkSubs = new Subscription();
  subs = new Subscription();
  loginCtrl = this.formBuilder.control('',
Validators.required);
  passwordCtrl = this.formBuilder.control('',
[Validators.required, Validators.minLength(4)]);
  confirmPswrdCtrl = this.formBuilder.control({
value: '', disabled: true },
    [Validators.required,
Validators.minLength(4)]);
  hidePswrd = true;
  formModel: FormGroup =
this.formBuilder.group({
    login: this.loginCtrl,
    email: ['', Validators.email],
    fullName: [''],
    passwords: this.formBuilder.group({
     password: this.passwordCtrl,
     confirmPassword: this.confirmPswrdCtrl,
    }, { validator: this.comparePasswords }),
  });
  constructor(
    private formBuilder: FormBuilder,
    private notifications: NotificationsService,
    private serverService: ServerHttpService,
    public auth: AuthService
  ) { }
  ngOnDestroy(): void {
    this.checkSubs.unsubscribe();
    this.subs.unsubscribe();
  }
  ngOnInit(): void {
    this.formModel.reset();
this.subs.add(this.passwordCtrl.valueChanges.sub
scribe(password =>
      (!this.passwordCtrl?.errors && password)
       ? this.confirmPswrdCtrl.enable()
       : this.confirmPswrdCtrl.disable()
    ));
  }
  onSubmit(isAdmin = false) {
    this.getsignUpRequest(isAdmin).subscribe(
      (res: any) => {
       if (res.succeeded) {
        this.createNotification('Реєстрація
пройшла успішно', '', NotificationType.Success);
       }
       else {
        this.createNotification('Помилка під час
реєстрації');
        res.errors.array.forEach(err => {
         console.log(err);
        });
```

```
        }
      },
      err => {
        if (err.status === 400) {
          this.createNotification('Реєстрація
провалена', 'Некоректні дані');
        }
        else {
          this.createNotification('Помилка під час
реєстрації');
        }
      }
    );
  }
  private getsignUpRequest(isAdmin: boolean):
Observable<any> {
    const model: ISignUpModel = {
      login: this.formModel.value.login,
      email: this.formModel.value.email,
      fullName: this.formModel.value.fullName,
      password:
this.formModel.value.passwords.password,
    };
    if (isAdmin && this.auth.isAdmin) {
      return
this.serverService.signUpAdmin(model);
    }
    return this.serverService.signUp(model);
  }
  createNotification(title: string, content: string =
'', type = NotificationType.Error) {
    this.notifications.create(title, content, type, {
      timeOut: 3000,
      showProgressBar: true,
      pauseOnHover: true,
      clickToClose: true
    });
  }
  comparePasswords(fg: FormGroup) {
    const confirmPswrdCtrl =
fg.get('confirmPassword');
    if (confirmPswrdCtrl.errors === null ||
'passwordMismatch' in confirmPswrdCtrl.errors) {
      if (fg.get('password').value !==
confirmPswrdCtrl.value) {
        confirmPswrdCtrl.setErrors({
passwordMismatch: true });
      }
      else {
        confirmPswrdCtrl.setErrors(null);
      }
    }
  }
  checkLogin() {
    const login = this.loginCtrl.value;
    if (login) {
      this.checkSubs =
this.serverService.isLoginUnique(login)
        .subscribe(unique =>
this.loginCtrl.setErrors(unique ? null : { unique:
true }));
    }
  }
}
```



					КПІ.ІП-6104.045440.05.СС	
Зм.	Арк.	№ докум.	Підпис	Дата	Схема структурна варіантів використань	Літ. / Маса / Масш.
Розробив		Брідня В.І.				
Перевірив		Ліщук К.І.				Аркуш 1 / Аркушів 1
Т. Контр.						
Н. Контр.		Ліщук К.І.			Веб-застосування «Кулінарний помічник»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61
Затверд.						



КПІ.ІП6104.045440.06.СБД

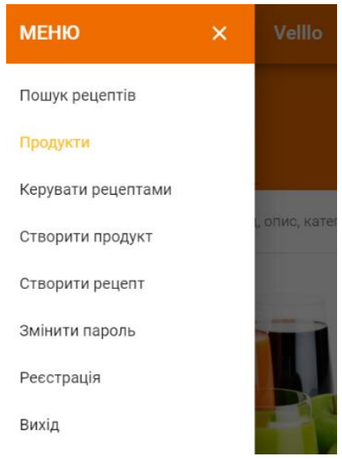
Зм.	Арк.	№ докум.	Підпис	Дата	КПІ.ІП-6104.045440.06.СБД		
Розробив		Брідна В.І.			Схема бази даних	Літ.	Маса	Масш.
Перевірив		Лищук К.І.					
Т. Контр.						Аркуш 1	Аркушів 1
Н. Контр.		Лищук К.І.			Веб-застосування «Кулінарний помічник»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61	
Затверд.							

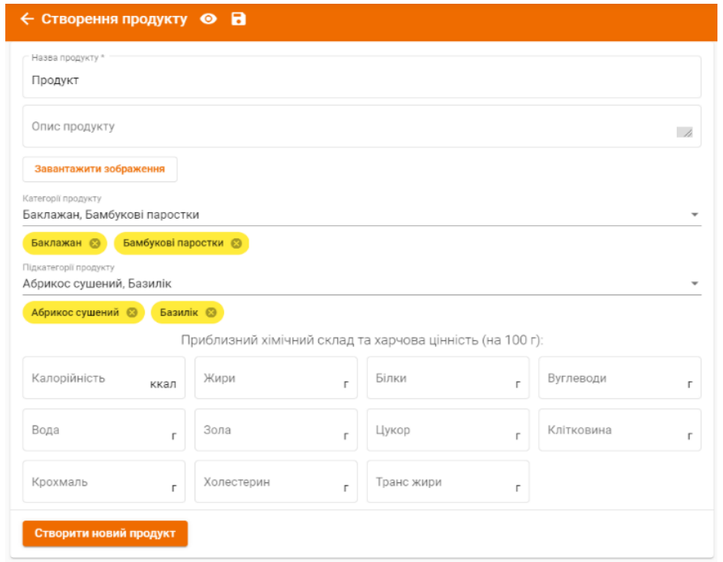
Щербет зі згущеним молоком

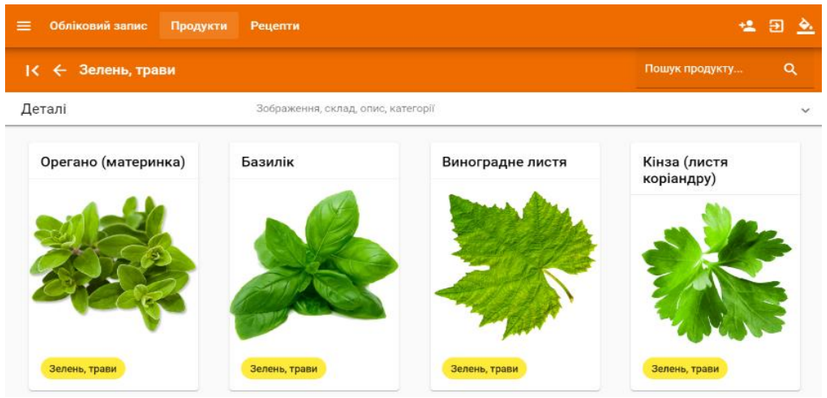
Популярні східні ласощі під назвою щербет - це дуже смачна солодкість, приготована на основі помадки з горіхами. Ми пропонуємо вам приготувати дуже простий, але неймовірно смачний варіант домашнього щербета зі згущеним молоком. Таке лакомство стане справжнім подарунком вашим домочадцям.

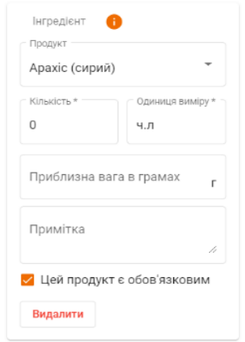
01/06/2020 06:15:21

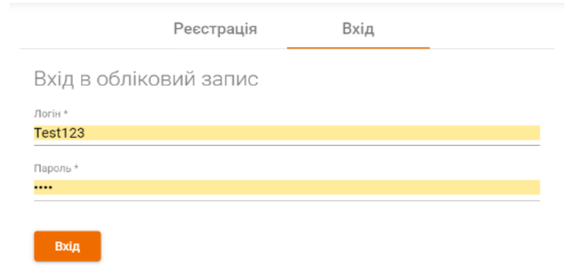
Деталі

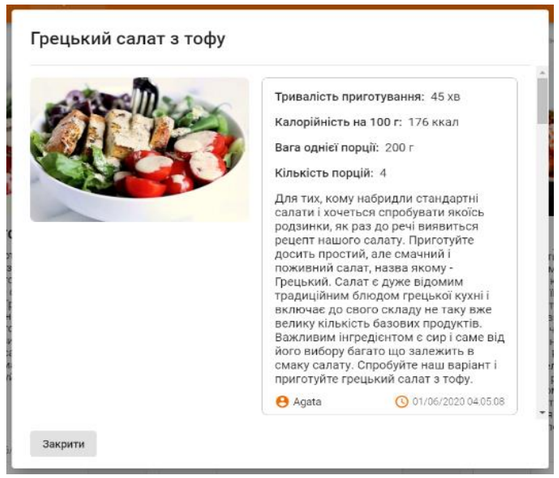


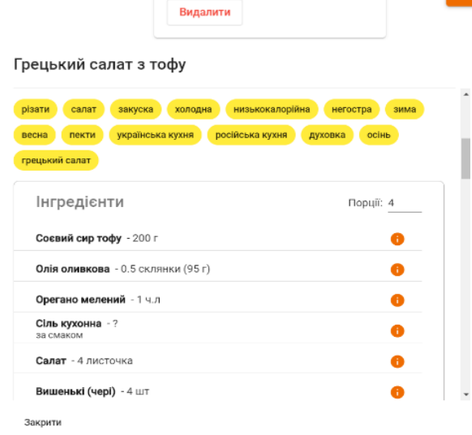


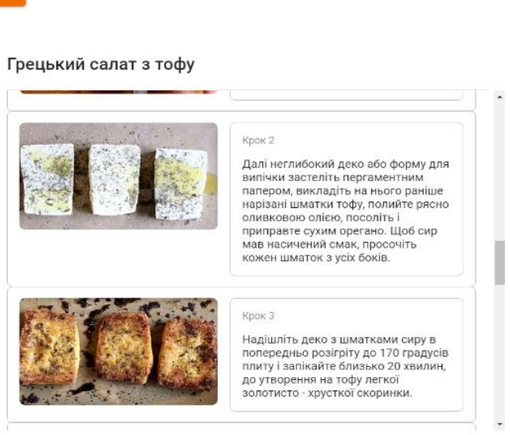


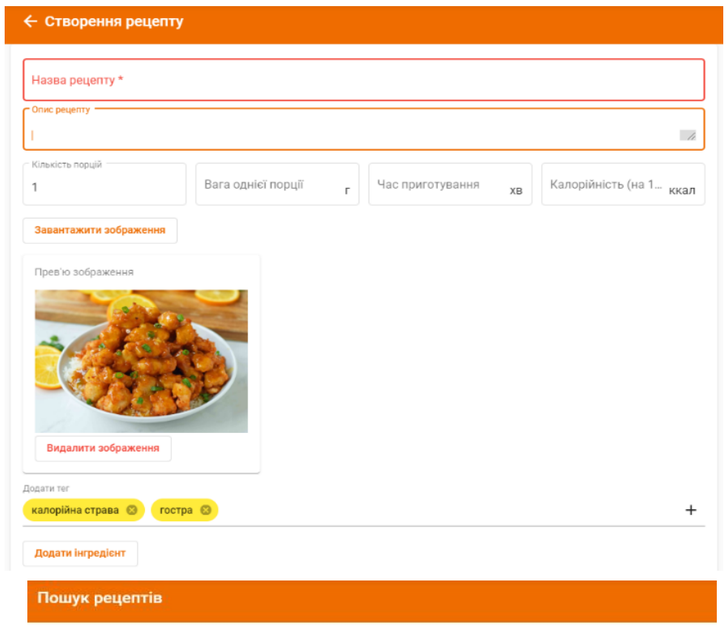


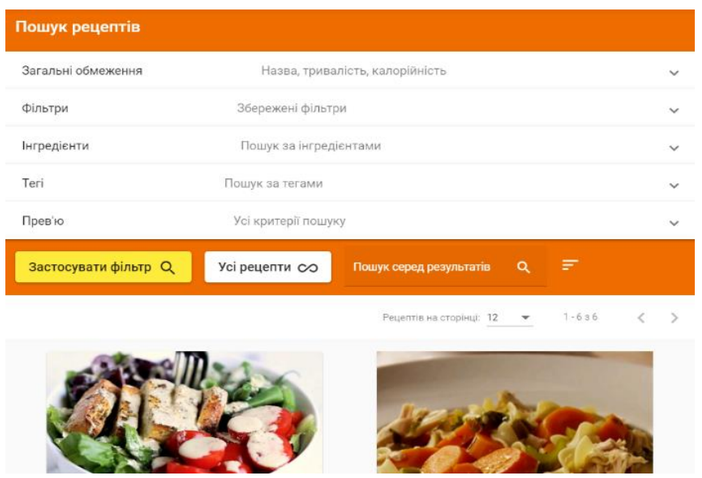












					КПІ.ІП-6104.045440.07.КЕ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Брідня В.І.			Креслення вигляду екранних форм	Літ.	Маса / Масш.
Перевірив		Ліщук К.І.					
Т. Контр.						Аркуш 1	Аркушів 1
Н. Контр.		Ліщук К.І.			Веб-застосування «Кулінарний помічник»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ група ІП-61	
Затверд.		Ліщук К.І.					