

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа з міксології»

Виконав студент групи ІПЗ-22-2

ШЕВЦОВ Микита Сергійович

Керівник роботи:

БАКАЛЮК Тетяна Анатоліївна

Рецензент:

ЧИЖМОТРЯ Олексій Володимирович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програм-
ного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ШЕВЦОВ Микита Сергійович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Вебплатформа з міксології»,**

затверджена Наказом закладу вищої освіти №92/с від 27 лютого 2026р.

Вихідні дані для роботи: **процес автоматизованого підбору коктейльних ре-
цептів на основі наявних інгредієнтів користувача, який реалізується у ви-
гляді вебдодатку з розділеною архітектурою (API + SPA) із підтримкою ав-
тентифікації та персоналізації бару.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т. А.	01.03.2026р.	01.03.2026р.
2	Вакалюк Т. А.	01.03.2026р.	01.03.2026р.
3	Вакалюк Т. А.	01.03.2026р.	01.03.2026р.

РЕФЕРАТ

Завданням на кваліфікаційну роботу було проєктування та розроблення вебплатформи для підтримки користувачів у сфері міксології, зокрема реалізація механізмів автентифікації, управління персональним баром та підбору коктейльних рецептів відповідно до наявних у користувача інгредієнтів.

Пояснювальна записка до кваліфікаційної роботи на тему «Вебплатформа з міксології» складається з вступу, трьох розділів, висновків, списку використаної літератури та додатків.

Текстова частина викладена на 100 сторінках друкованого тексту.

Список використаних джерел містить 39 найменувань і займає 4 сторінки. В роботі наведено 63 рисунки, 3 таблиці, 24 лістинги та 8 додатків. Загальний обсяг роботи – 124 сторінки.

Ключові слова: ВЕБПЛАТФОРМА, МІКСОЛОГІЯ, ASP.NET CORE, NUXT.JS, REST API, АВТЕНТИФІКАЦІЯ, УПРАВЛІННЯ ІНГРЕДІЄНТАМИ, ПОШУК РЕЦЕПТІВ, ПЕРСОНАЛЬНИЙ БАР, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

					ІПЗ.КР.Б – 121 – 26 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Шевцов М. С.			Вебплатформи з міксології				Літ.	Арк.	Аркушів
Керівник		Вакалюк Т. А.								3	124
									Житомирська політехніка, група ІПЗ-22-2		
Рецензент		Чижемотря О. В.									
Зав. каф.		Вакалюк Т. А.									

ABSTRACT

The objective of the qualification work was the design and development of a web platform to support users in the field of mixology, including the implementation of authentication mechanisms, personal bar management, and selection of cocktail recipes based on the ingredients available to the user.

The explanatory note for the qualification work titled “Web Platform for Mixology” consists of an introduction, three chapters, conclusions, a list of references and appendices.

The textual part is presented on 100 pages of printed text.

The list of references contains 39 entries and occupies 4 pages. The work includes 63 figures, 3 tables, 24 code listings and 8 appendices. The total volume of the work is 124 pages.

Ключові слова: WEB PLATFORM, MIXOLOGY, ASP.NET CORE, NUXT.JS, REST API, AUTHENTICATION, INGREDIENT MANAGEMENT, RECIPE SEARCH, PERSONAL BAR, CLIENT-SERVER ARCHITECTURE.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						4

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ РОЗРОБЛЕННЯ ВЕБПЛАТФОРМИ З МІКСОЛОГІЇ.....	11
1.1. Постановка задачі	11
1.2. Аналіз існуючих платформ та аналогів програмного продукту	13
1.3. Вибір архітектури вебплатформи.....	18
1.4. Обґрунтування вибору інструментальних засобів	22
Висновки до першого розділу	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ З МІКСОЛОГІЇ	27
2.1. Вимоги системи.....	27
2.2. Проєктування архітектури та моделей даних	32
2.3. Розробка серверної частини (REST API).....	39
2.4. Розробка клієнтського додатку (SPA)	49
Висновки до другого розділу.....	60
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ З МІКСОЛОГІЇ	63
3.1. Порядок встановлення та налаштування параметрів системи	63
3.2. Структура інтерфейсу вебплатформи з міксології.....	68
3.3. Тестування роботи вебплатформи	81
Висновки до третього розділу	93
ВИСНОВКИ.....	95
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	97
ДОДАТКИ.....	101

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

- КР – кваліфікаційна робота;
- ПЗ – програмне забезпечення;
- БД – база даних;
- СУБД – система управління базою даних;
- 2FA – двофакторна автентифікація (Two-Factor Authentication);
- ABV – вміст спирту за об’ємом (Alcohol by Volume);
- API – прикладний програмний інтерфейс (Application Programming Interface);
- BFS – пошук у ширину (Breadth-First Search);
- CDN – мережа доставки контенту (Content Delivery Network);
- CORS – спільне використання ресурсів між джерелами (Cross-Origin Resource Sharing);
- CQRS – розділення відповідальності команд і запитів (Command Query Responsibility Segregation);
- CRUD – створення, читання, оновлення та видалення (Create, Read, Update, Delete);
- DTO – об’єкт передавання даних (Data Transfer Object);
- ERD – діаграма «сутність-зв’язок» (Entity-Relationship Diagram);
- HMAC – код автентифікації повідомлень на основі хешу (Hash-based Message Authentication Code);
- HTTP – протокол передавання гіпертексту (HyperText Transfer Protocol);
- HTTPS – захищений протокол передавання гіпертексту (HyperText Transfer Protocol Secure);
- JWT – JSON-вебтокен (JSON Web Token);
- MJML – мова розмітки листів (Mailjet Markup Language);
- OAuth – відкрити протокол авторизації (Open Authorization);
- ORM – об’єктно реляційне відображення (Object-Relational Mapping);
- REST – передавання репрезентативного стану (Representational State Transfer);

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						6

SEO – пошукова оптимізація (Search Engine Optimization);
 SPA – односторінковий застосунок (Single Page Application);
 SQL – мова структурованих запитів (Structured Query Language);
 SSR – серверний рендеринг (Server-Side Rendering);
 TLS – захист транспортного рівня (Transport Layer Security);
 TOTP – одноразовий пароль на основі часу (Time-based One-Time Password);
 TTL – час життя даних (Time to Live);
 URL – уніфікований локатор ресурсу (Uniform Resource Locator).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						7

Вступ

Актуальність теми полягає у використанні сучасних вебтехнологій для створення персоналізованих інформаційних систем у сфері міксології. Культура споживання коктейлів останніми роками активно розвивається, а зацікавленість у домашньому приготуванні напоїв охоплює дедалі ширшу аудиторію – від випадкових ентузіастів до досвідчених домашніх барменів та блогерів. Це формує стійкий попит на зручні цифрові інструменти для роботи з рецептами, які виходять за межі простих текстових довідників.

Попри широку доступність рецептів коктейлів в інтернеті, більшість існуючих платформ функціонують як статичні енциклопедії або авторські блоги і не враховують реальний склад інгредієнтів, наявних у користувача. Як наслідок, людина змушена самотійно зіставляти вимоги рецепту з вмістом власного бару, що суттєво ускладнює практичне застосування таких ресурсів. Додатковим недоліком є абстрактне подання інгредієнтів без зазначення бренду, типу чи регіону походження, хоча у міксології ці деталі мають істотне значення для фінального результату. Більшість платформ також позбавлені соціальної складової: користувачі не можуть вільно публікувати власні рецепти, оцінювати чи обговорювати наявні, а отже, не формується повноцінна спільнота.

Сучасні підходи до веброзробки, зокрема клієнт-серверна архітектура на базі REST API, дозволяють долати ці обмеження, будуючи застосунки з персоналізованим контентом, що адаптується до індивідуальних даних кожного користувача. Розділення системи на серверну та клієнтські частини забезпечує технологічну незалежність, масштабованість та можливість обслуговувати одним API одночасно вебзастосунок, мобільний додаток і потенційні сторонні інтеграції. Використання фреймворків ASP.NET Core та Nuxt.js, своєю чергою, гарантує високу продуктивність, надійність коду завдяки строгій типізації, а також зручність та швидкість розроблення подібних систем із підтримкою серверного рендерингу для якісної індексації пошуковими системами.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						8

Таким чином, розроблення вебплатформи, яка поєднує управління персональним баром інгредієнтів із фільтрацією рецептів коктейлів на їх основі та детальною деталізацією інгредієнтів, є актуальним завдання у контексті сучасних тенденцій веброзробки, персоналізації цифрових сервісів та зростання інтересу до культури міксології.

Метою бакалаврської кваліфікаційної роботи є розроблення прототипу вебплатформи з міксології, що реалізує функціонал автентифікації користувачів, управління персональним баром інгредієнтів та пошук коктейльних рецептів на основі наявних інгредієнтів із використанням ASP.NET Core та Nuxt.js.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз предметної області міксології та огляд існуючих програмних рішень для роботи з рецептами коктейлів.
2. Визначити архітектуру та структуру застосунку.
3. Обґрунтувати та здійснити вибір засобів реалізації системи.
4. Спроекувати базу даних та серверний REST API.
5. Реалізувати модуль автентифікації, авторизації та керування сесіями.
6. Розробити функціонал персонального бару та пошук рецептів за інгредієнтами.
7. Реалізувати клієнтський вебзастосунок на базі Nuxt.js.

Об'єктом дослідження є процес підбору коктейльних рецептів на основі персонального набору інгредієнтів користувача в умовах сучасних вимог до зручності та персоналізації цифрових сервісів.

Предметом дослідження є функціональність вебплатформи з міксології, зокрема методи реалізації системи автентифікації користувачів, механізми управління персональним баром інгредієнтів та алгоритми фільтрації рецептів за їх складом із використання клієнт-серверної архітектури на основі ASP.NET Core та Nuxt.js.

В роботі було використано методи системного аналізу предметної області, методи об'єктно-орієнтованого проєктування, REST-архітектурного підходу до

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						9

побудови серверного API [24], а також методи вебпрограмування та проєктування інтерфейсів користувачів.

Практичне значення одержаних результатів полягає у тому, що розроблений прототип вебплатформи може бути використаний як самостійний цифровий сервіс для домашніх барменів та ентузіастів міксології, а також як основа для подальшого комерційного розвитку продукту з розширенням функціоналу.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				10

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ РОЗРОБЛЕННЯ ВЕБПЛАТФОРМИ З МІКСОЛОГІЇ

1.1. Постановка задачі

Міксологія – це наука про створення коктейлів, як алкогольних, так і безалкогольних. Вона охоплює знання про інгредієнти, їх поєднання, техніки приготування та подачі напоїв [7]. На відміну від простого змішування напоїв, міксологія розглядає коктейль як цілісну композицію, у якій кожен компонент має визначену роль, а кінцевий смак залежить не лише від переліку інгредієнтів, а й їх пропорцій, якості, способу обробки та навіть походженням. Саме тому ця сфера традиційно вважалася полем діяльності професійних барменів, проте впродовж останніх років вона дедалі активніше виходить за межі барів та ресторанів, охоплюючи широку аудиторію аматорів.

З розвитком культури споживання коктейлів зростає і зацікавленість широкої аудиторії, від домашніх ентузіастів до професійних барменів, у доступних та зручних інструментах для роботи з рецептами [35]. Домашнє приготування напоїв перетворилося на повноцінне хобі: люди облаштовують власні домашні бари, експериментують зі складниками, діляться результатами у соціальних мережах та шукають нові рецепти в інтернеті. Така зміна поведінки формує стійкий попит на цифрові сервіси, які не просто зберігають рецепти, а допомагають користувачеві орієнтуватися у власних можливостях та підбирати напої відповідно до того, чим він реально володіє. Водночас наявні цифрові рішення значною мірою лишилися на рівні попередньої епохи, коли користувач сприймався як пасивний читач, а не як активний учасник процесу.

Незважаючи на широку доступність рецептів в інтернеті, більшість існуючих платформ функціонують як цифрові довідники або авторські блоги, а не як повноцінні спільноти. Користувачі позбавлені можливості публікувати власні рецепти, взаємодіяти з контентом інших учасників або отримувати персоналізовані рекомендації на основі наявних інгредієнтів. Як наслідок, цінність такого ресурсу для

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				11

повсякденного використання суттєво знижується, адже основне навантаження з адаптації рецепту до власних умов перекладається на самого користувача.

Розроблювана платформа орієнтована на широку аудиторію – від домашніх барменів-початківців, які лише опановують основи приготування коктейлів, до досвідчених міксологів і блогерів, що бажають ділитися власними авторськими рецептами та формувати власну аудиторію. Для початківців платформа стане зручним інструментом, що допомагає підібрати напій із того, що вже є під рукою, і поступово розширювати власний бар, тоді як досвідченим користувачам вона надасть простір для публікації авторських рецептів, отримання зворотного зв'язку від спільноти та взаємодії з однодумцями. Такий підхід дозволяє охопити користувачів з різним рівнем підготовки та зробити платформу корисною як для повсякденного домашнього використання, так і для творчої самореалізації у сфері міксології. Це закладає основу для формування активної спільноти, що самостійно наповнює платформу контентом та підтримує її розвиток.

З огляду на широту окресленої проблематики, у межах кваліфікаційної роботи реалізується не вся платформа, а її ключовий прототип, що охоплює фундаментальний для подальшого розвитку функціонал: автентифікацію та керування обліковим записом користувача, ведення персонального бару інгредієнтів та підбір доступних рецептів на основі його вмісту. Саме ці складові формують технічне та концептуальне ядро системи, без якого неможлива реалізація решти функцій. Решта можливостей – публікація користувацьких рецептів, система оцінювання та коментування, облік залишків інгредієнтів і соціальна взаємодія – розглядаються як напрями розширення, для яких розроблюваний прототип закладає масштабовану архітектурну основу. Таким чином, постановка задачі полягає у створенні працездатного ядра вебплатформи, що демонструє життєздатність центральної ідеї персоналізованого підбору коктейлів та слугує підґрунтям для подальшого розгортання повноцінного сервісу для спільноти міксологів.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				12

1.2. Аналіз існуючих платформ та аналогів програмного продукту

Для визначення недоліків наявних рішень та обґрунтування доцільності розроблення власної платформи, було проведено аналіз чотирьох найбільш відомих платформ у сфері міксології: «Difford's Guide» [8], «Topsy Bartender» [34], «Cocktail Flow» [6], та мобільного застосунку «Jigger» [38, 39].

«Difford's Guide» (www.diffordsguide.com) [8] – одна з найбільших і найавторитетніших онлайн-енциклопедій коктейлів у світі. Платформа заснована Саймоном Діффордом і містить тисячі ретельно описаних рецептів із зазначенням інгредієнтів, технік приготування, історії походження та рекомендацій щодо подачі. Контент відрізняється високим рівнем деталізації: для більшості рецептів вказані конкретні бренди інгредієнтів та їх точні пропорції.

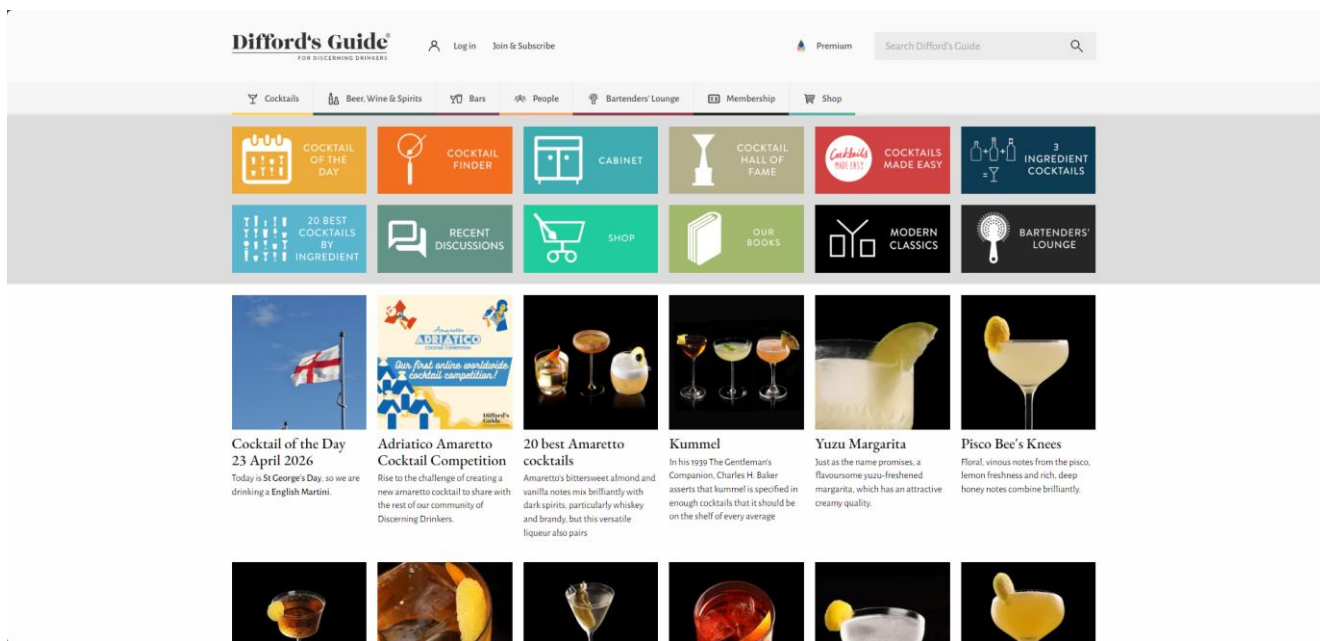


Рис. 1.1. Вебінтерфейс «Difford's Guide» [8]

Попри якість контенту, платформа функціонує переважно як авторська енциклопедія. Технічно користувач може надіслати власний рецепт через спільну форму або електронною поштою безпосередньо засновнику платформи, однак публікація відбувається виключно за рішенням редакції – рецепт має відповідати низці вимог: бути оригінальним, легко відтворюваним та відповідати смакам цільової аудиторії платформи. Після відбору рецепт фотографується командою та додається до каталогу. Таким чином, можливість публікації є номінальною і не формує

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				13

повноцінної спільноти – користувач не може самостійно і вільно поділитися власним рецептом, а процес нагадує подачу матеріалу до редакції журналу, а не публікацію на відкритій платформі. Оцінювання як таке відсутнє, редакція звертає увагу лише на кількість користувачів що додали рецепт до обраного, не надаючи користувачам можливість надати реальний відгук, хоча на сайті присутні коментарі для обговорень, але вони не користуються великим попитом і не поєднані з повноцінним відгуком.

«Topsy Bartender» (www.tipsybartender.com) [34] – медіа-платформа, що бере свій початок з однойменного YouTube-каналу, де публікуються відео-рецепти коктейлів. Вебсайт є супровідним ресурсом до відео-контенту і містить текстові версії рецептів із зображеннями готових напоїв.

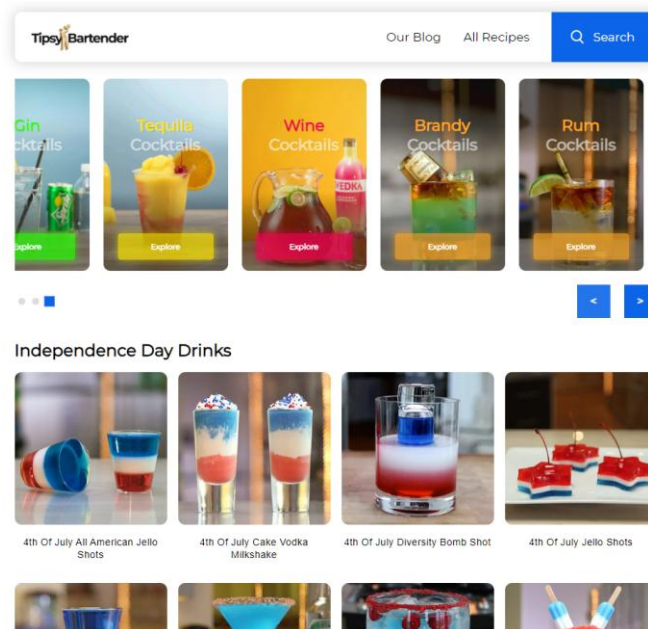


Рис. 1.2. Вебінтерфейс «Topsy Bartender» [34]

Платформа повністю орієнтована на власний авторський контент команди редакції. Спільноти як такої не існує – користувачі не можуть публікувати власні рецепти, оцінювати або коментувати наявні. Будь-яка взаємодія відбувається виключно через сторонні платформи. Пошук рецептів є обмеженим і не передбачає якісної фільтрації за інгредієнтами. Платформа є, по суті, цифровою версією авторського блогу, перенесеною у вебформат.

«Cocktail Flow» (www.cocktailflow.com) [6] – вебплатформа та мобільний застосунок для перегляду рецептів коктейлів із зручним і візуально привабливим інтерфейсом. Платформа пропонує базову фільтрацію рецептів за типом алкоголю та наявними інгредієнтами, що виділяє її серед більшості аналогів.

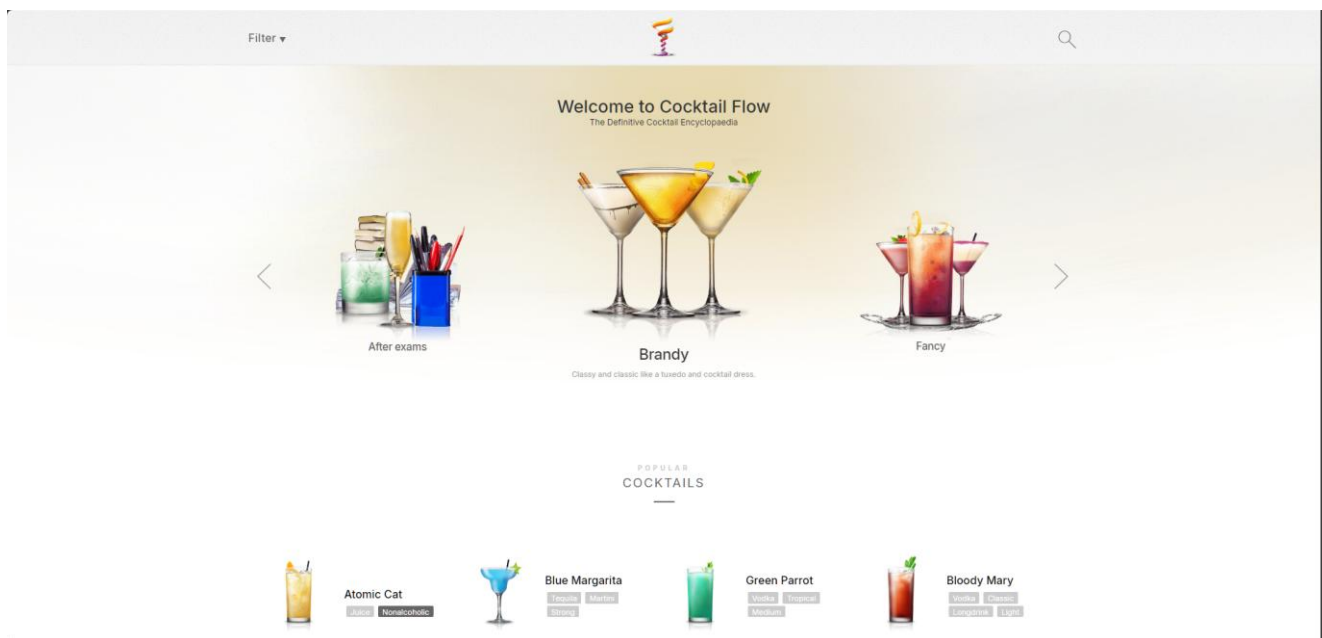


Рис. 1.3. Вебінтерфейс «Cocktail Flow» [6]

Проте реалізація фільтрації за інгредієнтами є поверхневою: інгредієнти зазначаються абстрактно – наприклад, «горілка» або «ром» – без уточнення бренду, типу витримки чи регіону походження. Водночас у міксології ці деталі мають суттєве значення для фінального напою. Альтернативні інгредієнти не пропонуються. Публікація власних рецептів користувачами не передбачена, оцінювання та коментування відсутні. Платформа залишається інструментом перегляду готового каталогу а не майданчиком для спільноти.

«Jigger» [38, 39] – мобільний застосунок, орієнтований на домашніх барменів. На відміну від попередніх платформ, «Jigger» дозволяє користувачу вести власний список наявних інгредієнтів і на основі цього списку підбирати доступні рецепти. Це робить його найближчим за концепцією до функціоналу персонального бару, що реалізується у розроблюваній платформі.

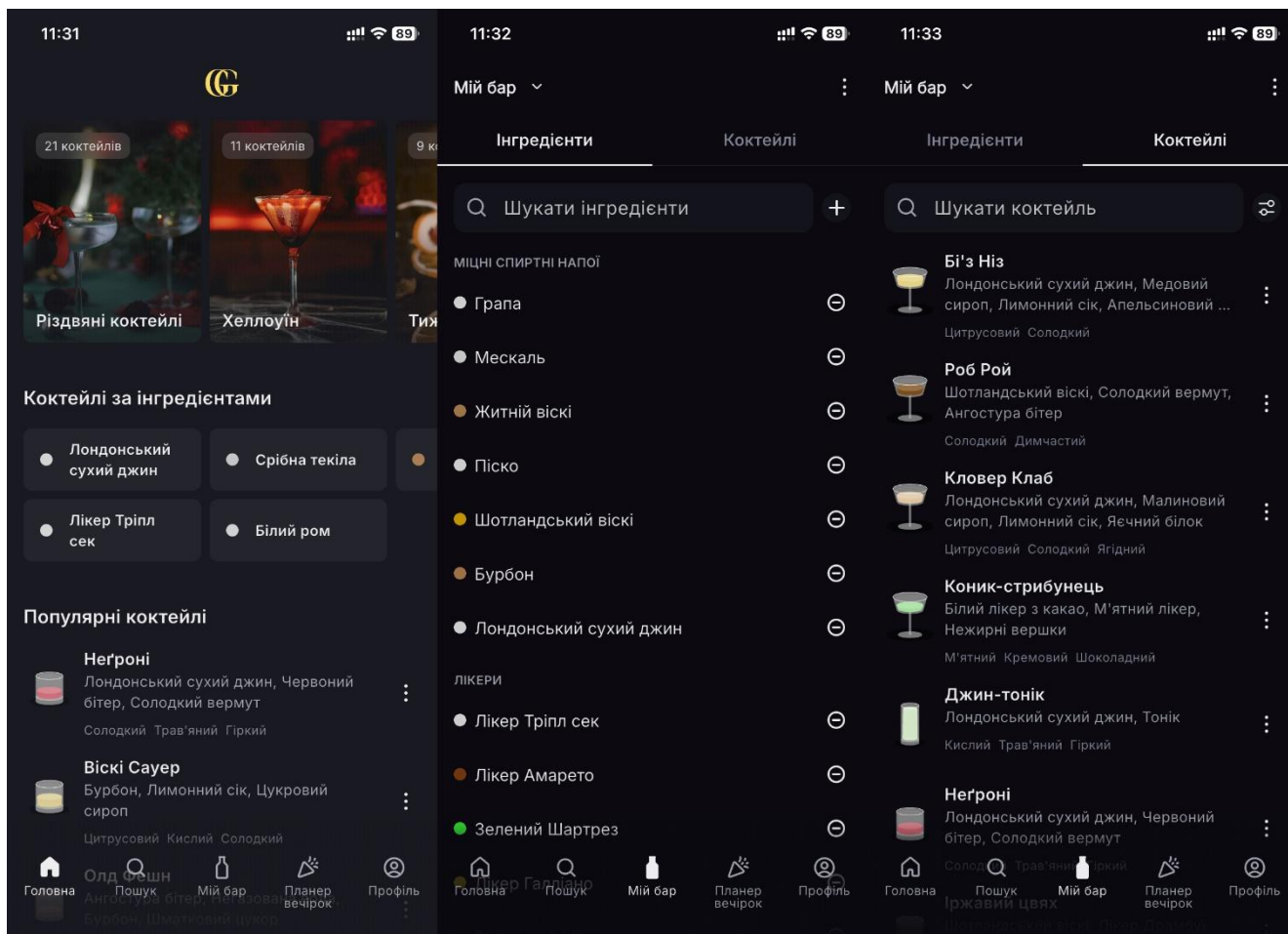


Рис. 1.4. Інтерфейс «Jigger» [38, 39]

Однак «Jigger» є суто приватним персональним інструментом. У застосунку відсутня будь-яка соціальна складова: немає публічних рецептів від користувачів, немає можливості ділитися власними рецептами, немає оцінок, коментарів чи будь-яких форм взаємодії між учасниками. Увесь контент є редакційним. Як і у минулих кандидатів, всі інгредієнти відображаються абстрактно, без будь-якої конкретики специфікації виробника, витримки. Крім того, застосунок доступний виключно як мобільний і не має повноцінної вебверсії.

За результатами проведеного аналізу альтернативного програмного забезпечення було складено зведену таблицю порівняння розглянутих платформ із розроблювальною вебплатформою «Snake It», що відображає функціонал прототипу платформи а також загальну ідею розвитку вебплатформи (Табл. 1.1).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				16

Таблиця 1.1

Порівняльна таблиця платформ

Критерій	Difford's Guide [8]	Tipsy Bartender [34]	Cocktail Flow [6]	Jigger [38,39]	Snake It (прототип)	Snake It (перспектива)
Публікація власних рецептів	Частково	Ні	Ні	Ні	Ні	Так
Пошук за інгредієнтами бару	Ні	Ні	Частково	Так	Так	Так
Деталізація інгредієнтів (бренд, тип)	Так	Частково	Ні	Ні	Так	Так
Оцінки та коментарі до рецептів	Частково	Ні	Ні	Ні	Ні	Так
Управління персональним баром	Ні	Ні	Ні	Так	Так	Так
Облік залишків інгредієнтів	Ні	Ні	Ні	Ні	Ні	Так
Соціальна складова / спільнота	Ні	Ні	Ні	Ні	Ні	Так
Генерація рецептів за допомогою ШІ	Ні	Ні	Ні	Ні	Ні	Так
Рекомендації покупки інгредієнтів	Ні	Ні	Ні	Ні	Ні	Так
Вебверсія	Так	Так	Так	Ні	Так	Так
Мобільний додаток	Ні	Ні	Так	Так	Ні	Так

Проведений аналіз підтверджує, що всі розглянуті платформи функціонують переважно як цифрові довідники або авторські блоги з обмеженими можливостями взаємодії. Жодна з них не поєднують в собі персоналізований підбір рецептів на основі реального вмісту бару користувача з повноцінною відкритою платформою для спільноти. Розроблювальна платформа «Snake It» у рамках КР реалізує ключовий прототип із функціоналом автентифікації, управління персональним баром та пошуку рецептів за інгредієнтами, закладаючи архітектурну основу для подальшого розширення платформи до повноцінного соціального простору для спільноти міксологів.

1.3. Вибір архітектури вебплатформи

Архітектура є фундаментом будь-якої програмної системи, оскільки саме вона визначає, наскільки легко продукт піддаватиметься розвитку, тестуванню та масштабуванню у довгостроковій перспективі. Невдало обрана архітектура згодом призводить до накопичення технічного боргу, ускладнення підтримки та зростання вартості внесення навіть незначних змін. З огляду на те, що розроблювана платформа від самого початку проєктувалася як основа для подальшого розширення до повноцінного соціального простору для спільноти міксологів, до вибору архітектурних рішень було здійснено особливо ретельний підхід. Нижче розглянуто архітектурні рішення, покладені в основу платформи, та наведено обґрунтування кожного з них.

В основу платформи покладено клієнт-серверну архітектуру з чітким розподілом відповідальності між серверною та клієнтськими частинами [18]. Суть такого підходу полягає у тому, що серверна частина повністю відповідає за бізнес-логіку, роботу з даними та формування відповідей, тоді як клієнтська частина зосереджена виключно на представленні інформації та взаємодії з користувачем. Обмін даними між ними відбувається через чітко визначений програмний інтерфейс REST API [24]. Такий поділ обрано з кількох вагомих причин.

Передусім клієнт-серверна архітектура забезпечує технологічну незалежність складових системи. Один і той самий REST API може одночасно обслуговувати вебзастосунок, мобільний додаток та потенційні B2B-інтеграції, не вимагаючи дублювання бізнес-логіки для кожного з них. Це означає, що при майбутньому розширенні платформи мобільним застосунком не доведеться повторно реалізовувати правила підбору рецептів чи управління баром – достатньо буде під'єднати новий клієнт до вже наявного API. Більше того, такий підхід відкриває можливості для зовнішніх інтеграцій: наприклад, інтернет-магазини алкогольних напоїв зможуть використовувати API платформи для відображення рецептів з відповідними інгредієнтами поруч із товарами, що пропонуються до продажу.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				18

Другою важливою перевагою є зниження загального навантаження на сервер завдяки розподілу відповідальності. Статичні ресурси, такі як зображення рецептів та аватари користувачів, обслуговуються через мережу доставки контенту (CDN) незалежно від основного API-сервера. Це дозволяє серверу зосередитися виключно на обробці бізнес-логіки та формуванні відповідей, не витрачаючи обчислювальних ресурсів на роздачу великих медіа-файлів. Така організація позитивно впливає як на продуктивність, так і на масштабованість системи, оскільки навантаження від обслуговування контенту рівномірно розподіляється між вузлами CDN, географічно наближеними до користувача.

Альтернативою клієнт-серверному підходу могла б стати монолітна архітектура із серверним рендерингом сторінок, за якої логіка представлення та бізнес-логіка перебувають у межах одного застосунку. Проте такий підхід суттєво ускладнив би повторне використання серверної частини для інших клієнтів та обмежив би гнучкість розвитку платформи. Саме тому для розроблювальної системи обрано саме клієнт-серверну архітектуру з розділеними API та клієнтом.

Серверна частина платформи реалізована на основі принципів чистої архітектури (Clean Architecture) (Рис. 1.5) [1]. Ключовою ідеєю цього підходу є розподіл коду на концентричні незалежні шари з чітким напрямком залежностей – лише всередину, від зовнішніх деталей реалізації до ядра бізнес-логіки. Завдяки цьому правилу залежностей, внутрішні шари нічого не знають про зовнішні, що дозволяє змінювати технологічні рішення, не торкаючись бізнес-правил системи.

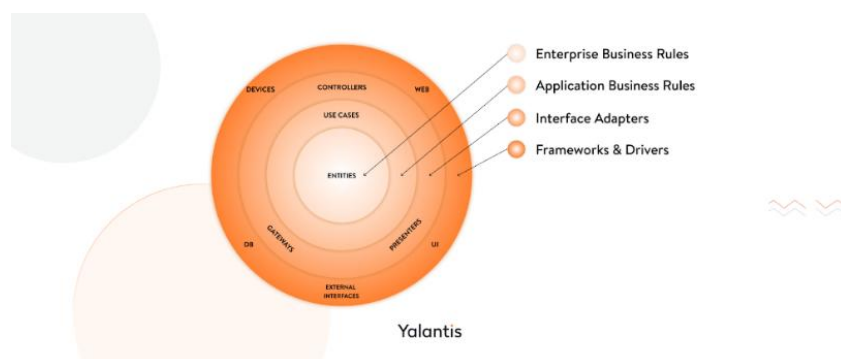


Рис. 1.5. Структура «Чистої архітектури» [1]

Систему поділено на чотири шари. Шар домену є ядром системи і містить сутності предметної області, перерахування та базові класи, він не залежить від

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				19

жодного іншого шару. Шар застосунку містить бізнес-логіку та визначає інтерфейси інфраструктурних сервісів, проте абстрагований від конкретних їх реалізацій. Шар інфраструктури надає конкретні реалізації цих інтерфейсів – доступ до бази даних, роботу з кешем, відправлення пошти, взаємодію зі сторонніми сервісами. Нарешті, зовнішній шар презентації відповідає за приймання HTTP-запитів та формування відповідей.

Подібна організація забезпечує слабку зв'язаність між компонентами системи, що суттєво спрощує її супровід та подальше розширення [1]. Оскільки бізнес-логіка не залежить від конкретних інфраструктурних рішень, будь-який зовнішній адаптер можна замінити без змін у ядрі: наприклад, перейти з однієї системи зберігання файлів на іншу або змінити постачальника кешу.

Принципи чистої архітектури доповнено патерном CQRS (Command Query Responsibility Segregation) [11], реалізованим за допомогою бібліотеки «MediatR» [33]. Суть цього патерну полягає у розділенні операцій читання даних (Query) та операцій, що змінюють стан системи (Command). Кожна операція оформлюється у вигляді окремого класу запиту або команди та відповідного обробника, що забезпечує дотримання принципу єдиної відповідальності та підвищує тестопридатність кожного сценарію.

Таке розділення дозволяє незалежно оптимізувати потоки читання та запису, зберігаючи чіткість коду навіть при суттєвому зростанні обсягу функціоналу [11]. Операції читання можуть бути оптимізовані під швидкий вибір даних та кешування, тоді як операції запису зосереджуються на дотриманні бізнес-правил та цілісності даних. Додатковою перевагою застосування «MediatR» є можливість впровадження крос-функціональних поведінок (Pipeline Behaviors), які автоматично застосовуються до кожної операції – зокрема логування, перевірки дозволів, валідації вхідних даних та кешування. Це дозволяє уникнути дублювання однотипного коду в одному обробнику та зосередити подібну логіку в одному місці.

Для реалізації клієнтської частини обрано фреймворк «Nuxt.js», побудований на базі «Vue.js». При виборі між «Nuxt.js» та зв'язкою «Vite» + «Vue» вирішальну

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				20

роль зіграло насамперед те, що «Nuxt.js» з коробки підтримує серверний рендеринг (SSR). Серверний рендеринг забезпечує швидке завантаження першої сторінки та якісну індексацію контенту пошуковими системами (SEO) [37], що є критично важливим для платформи з публічним контентом у вигляді рецептів, які мають добре знаходитися у пошуковій видачі.

Окрім цього, «Nuxt.js» дозволяє виносити частину логіки на сервер без передачі її у клієнтський JavaScript-код, що водночас підвищує безпеку застосунку та зменшує обсяг коду, який завантажується у браузер користувача. Не менш важливою перевагою є висока швидкість розробки, яку забезпечують зріла екосистема «Vue.js», зручна файлова маршрутизація та готові модулі «Nuxt». У сукупності ці фактори зробили «Nuxt.js» оптимальним вибором для клієнтської частини платформи.

Автентифікація реалізується на основі JWT-токенів у форматі «Access Token + Refresh Token» [23]. Важливо зазначити, що JWT у даній системі використовується виключно для автентифікації, тобто ідентифікатор користувача – токен містить лише ідентифікатор сесії та користувача. Авторизація ж, тобто перевірка прав доступу, відбувається безпосередньо на сервері при кожному запиті на основі актуальних даних з бази даних або кешу. Такий підхід має суттєву перевагу над класичним, за якого права доступу закладаються у сам токен: він дозволяє миттєво реагувати на зміни у правах користувача, тоді як у класичному підході, закладені у токен права залишаються чинними до завершення терміну його дії, навіть якщо фактично вони вже були відкликані.

Для забезпечення зручності та безпеки реалізовано механізм керування сесіями, що дозволяє користувачу бути одночасно авторизованим на кількох пристроях – у вебзастосунку, мобільному додатку тощо – та самостійно переглядати й відкликати активні сесії з міркувань безпеки. Це особливо корисно у випадку втрати пристрою або підозри на несанкціонований доступ.

Окрім стандартного входу за допомогою електронної пошти та паролю, реалізовано вхід через обліковий запис «Google» за допомогою authorization code flow

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				21

[13]. Цей підхід є значно безпечнішим за пряму передачу токена «Google» клієнту, оскільки обмін авторизаційного коду на токен відбувається безпосередньо між серверами, без участі браузера, що повністю виключає ризик перехоплення токена під час передачі [12]. Додатково для підвищення рівня захисту облікових записів реалізовано двофакторну автентифікацію на основі протоколу TOTP (Time-based One-Time Password) – одноразових паролів із часовою прив'язкою, що генеруються у спеціалізованих застосунках-автентифікаторах [22].

1.4. Обґрунтування вибору інструментальних засобів

Вибір технологічного стеку є одним із визначальних рішень на етапі проектування, оскільки саме він зумовлює продуктивність, надійність та зручність подальшого розвитку системи. З огляду на те, що розроблювана платформа від самого початку планувалася як основа для масштабного сервісу для спільноти міксологів, до підбору інструментів було застосовано особливо зважений підхід, за якого кожна технологія обиралася не ізольовано, а як частина цілісної та узгодженої екосистеми. Нижче наведено обґрунтування обраних засобів для серверної та клієнтської частини, а також для допоміжної інфраструктури платформи.

Для реалізації серверної частини обрано платформу «ASP.NET Core 10», що зумовлено кількома вагомими перевагами [16]. Передусім це висока продуктивність, «ASP.NET Core» стабільно входить до числа найшвидших вебфреймворків за результатами незалежних тестувань «TechEmpower» [28]. Така продуктивність досягається завдяки високооптимізованому вебсерверу Kestrel, асинхронній моделі обробки запитів на основі конструкцій `async/await`, а також ефективному управлінні пам'яттю, що дозволяє обслуговувати велику кількість одночасних з'єднань за мінімальних витрат ресурсів. Це особливо важливо для платформи, яка у перспективі має обслуговувати одним API одразу кілька типів клієнтів. Не менш важливою є строга статична типізація мови C#, яка у поєднанні з принципами чистої архітектури та патерном CQRS забезпечує надійність коду, спрощує його підтримку та унеможливорює появу цілого класу помилок ще на етапі компіляції. Сучасні

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				22

можливості мови, зокрема підтримка nullable-типів, записів (records) та зіставлення зі зразком (pattern matching), додатково підвищують виразність коду та зменшують імовірність типових помилок, пов'язаних із некоректною роботою з порожніми значеннями. Додатковим аргументом стала розвинена екосистема пакетів та офіційна довгострокова підтримка від компанії Microsoft, що гарантує стабільність технології у довгостроковій перспективі [16]. Платформа є повністю кросплатформною, що дозволяє розгортати її як у середовищі Windows, так і на Linux-серверах, які традиційно домінують у хмарній інфраструктурі та мають нижчу вартість обслуговування.

В основу серверної частини покладено власний шаблон чистої архітектури, розроблений у рамках переддипломної практики [2, 1]. Шаблон реалізує розподіл коду на незалежні шари – домен, застосунок, інфраструктура та презентація, та використовує патерн CQRS через бібліотеку «MediatR» [33] для чіткого розмежування операцій читання та запису. Використання заздалегідь підготовленого шаблону дозволило зосередитися безпосередньо на реалізації предметної логіки платформи, не витрачаючи час на повторне налаштування інфраструктурних механізмів, які є типовими для більшості сучасних вебзастосунків. Окрім того, такий підхід забезпечує єдиний та передбачуваний стиль організації коду в межах усього проєкту, що спрощує орієнтування у кодовій базі та її подальше розширення.

Функціональність API забезпечується низкою ключових пакетів. Бібліотека «MediatR» [33] реалізує патерн Mediator [26] для розподілу команд та запитів у рамках CQRS, усуваючи пряму залежність між компонентами системи та централізуючи проходження кожної операції через єдиний конвеєр обробки. Бібліотека «FluentValidation» відповідає за декларативну валідацію вхідних даних на рівні команд та запитів, що дозволяє винести правила перевірки в окремі класи, відокремивши їх від основної бізнес-логіки. Надсилання електронної пошти із підтримкою MJML-шаблонів [17] для формування HTML-листів реалізовано через «MailKit» - зрілу та широко вживану бібліотеку, що підтримує сучасні протоколи й механізми захищеного з'єднання. Пакет «Asp.Versioning» забезпечує версіонування API-

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				23

ендпоінтів задля збереження зворотної сумісності при розширенні платформи, що є критично важливим у ситуації, коли одночасно можуть існувати кілька клієнтів, оновлення яких відбувається незалежно один від одного. Автоматична інтерактивна документація ендпоінтів на основі специфікації OpenAPI генерується за допомогою «Scalar» [25], завдяки чому опис API завжди залишається синхронізованим із фактичною реалізацією та не потребує ручного підтримання в актуальному стані.

Як СУБД обрано «PostgreSQL» [31] - надійну реляційну базу даних з відкритим вихідним кодом, що добре масштабується, має широку підтримку спільноти та розвинену екосистему розширень. «PostgreSQL» добре зарекомендувала себе у виробничих середовищах із високим навантаженням і є одним із найпоширеніших рішень для вебзастосунків. Вибір саме реляційної моделі зумовлений характером предметної області, у якій між сутностями існують чіткі та сталі зв'язки – між користувачами, рецептами, інгредієнтами та категоріями, а цілісність цих зв'язків має гарантуватися на рівні бази даних. «PostgreSQL» забезпечує повну підтримку транзакційності за принципами ACID, має ефективні механізми індексації та керування одночасним доступом на основі багатоверсійності, що дозволяє опрацьовувати конкурентні запити без надмірного блокування. Взаємодія з базою даних здійснюється через ORM «Entity Framework Core», що дозволяє працювати з даними у вигляді об'єктів C#, автоматично генерувати міграції схеми бази даних та абстрагуватися від деталей конкретної СУБД, зберігаючи можливість за потреби змінити її без суттєвих змін у коді застосунку [9]. Застосування ORM також підвищує безпеку, оскільки параметризовані запити, які формуються автоматично, унеможливають цілий клас атак на кшталт SQL-ін'єкцій, а строга типізація запитів за допомогою LINQ дозволяє виявляти помилки роботи з даними ще на етапі компіляції.

Для клієнтської частини обрано фреймворк «Nuxt.js» на базі «Vue.js». При виборі між «Nuxt.js» та «Vite» + «Vue» вирішальну роль відіграла насамперед підтримка серверного рендерингу (SSR), що забезпечує швидке завантаження першої сторінки та якісну індексацію пошуковими системами, а це є важливим для платформи з публічним контентом у вигляді рецептів. Окрім того, «Nuxt.js» дозволяє

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				24

виносити частину логіки на сервер без передачі її у клієнтський JavaScript-код, що підвищує безпеку та зменшує розмір клієнтського коду. Нарешті, зріла екосистема «Vue.js», файлова маршрутизація та зручні модулі «Nuxt» суттєво пришвидшують процес розробки.

Клієнтська частина використовує низку допоміжних бібліотек та модулів. Управління глобальним станом застосунку реалізовано через «Pinia» [29] – рекомендоване рішення для «Vue.js 3», а стилізація інтерфейсу виконується за допомогою утилітарного CSS-фреймворку «Tailwind CSS 4» [20], що забезпечує гнучкість та консистентність дизайну. Валідацію форм на стороні клієнта виконує легка бібліотека «Valibot» [30], тоді як модуль «nuxtjs/i18n» відповідає за інтернаціоналізацію застосунку та підтримку кількох мов інтерфейсу. Точне позиціонування спливаючих елементів інтерфейсу, таких як підказки та випадаючі меню, забезпечує «floating-ui/vue», а інтеграція входу через обліковий запис «Google» на стороні клієнта реалізована за допомогою «nuxt-vue3-google-signin». Іконографію інтерфейсу у вигляді SVG-іконок надає «Iconify» [4] із набором «Lucide» та «Tabler». Для підтримки єдиного стилю коду у проєкті використовується «ESLint» та «Prettier», які відповідають за статичний аналіз та автоматичне форматування.

Для зберігання та роздачі медіа-файлів (зображення рецептів, аватари користувачів) підключено CDN «bunny.net» [5], що знімає навантаження з основного API-сервера: сервер відповідає виключно за обробку запитів до API, тоді як статичні ресурси роздаються через мережу розподіленого кешування незалежно від основного сервісу. Завдяки географічно розподіленим вузлам кешування медіа-файли віддаються користувачам з найближчого до нього сервера, що зменшує затримки та підвищує швидкість завантаження контенту незалежно від місцеперебування користувача. Надсилання електронної пошти (підтвердження реєстрації, сповіщення, відновлення доступу) реалізовано через сервіс «Spacemail» [27] з використанням MJML-шаблонів – спеціалізованої мови розмітки для створення адаптивних HTML-листів, що коректно відображаються у різних поштових клієнтах [17].

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						25

Висновки до першого розділу

У першому розділі проведено аналіз предметної області міксології та визначено коло проблем, які вирішує розроблювана платформа. Встановлено, що більшість існуючих платформ функціонують як цифрові довідники або авторські блоги з обмеженими можливостями взаємодії між користувачами, що не відповідає потребам сучасної спільноти міксологів.

Проведений аналіз чотирьох платформ-аналогів – «Difford's Guide», «Tipsy Bartender», «Cocktail Flow» та «Jigger» показав, що жодна з них не поєднує персоналізований підбір рецептів на основі реального вмісту бару користувача з повноцінною відкритою платформою спільноти. Виявлено спільні недоліки: відсутність вільної публікації власних рецептів, абстрактна деталізація інгредієнтів без вказівки бренду чи типу, відсутність повноцінної системи оцінювання та соціальної складової.

Обґрунтовано вибір клієнт-серверної архітектури з REST API як основи для забезпечення технологічної незалежності та масштабованості платформи. Серверна частина реалізується на «ASP.NET Core 10» з використанням принципів чистої архітектури та патерну CQRS на основі власного шаблону, розробленого у рамках переддипломної практики. Система автентифікації побудована на основі JWT у форматі «Access Token + Refresh Token», де токен використовується виключно для ідентифікації користувача, тоді як авторизація відбувається на сервері при кожному запиті на основі актуальних даних, що забезпечує миттєву реакцію на зміни у правах доступу. Додатково реалізовано вхід через «Google» за допомогою code-flow та двох-факторну автентифікацію на основі TOTP.

Клієнтська частина реалізується на «Nuxt.js» із підтримкою серверного рендерингу для забезпечення SEO та швидкого завантаження. Як СУБД обрано «PostgreSQL» з ORM «Entity Framework Core». Для роздачі медіа-файлів підключено CDN «bunny.net», надсилання пошти реалізовано через «Spsacemail» з використанням MJML-шаблонів. Визначено та обґрунтовано повний набір допоміжних бібліотек для серверної та клієнтської частин платформи.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						26

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ З МІКСОЛОГІЇ

2.1. Вимоги системи

Перед початком проєктування вебплатформи визначено перелік функціональних та нефункціональних вимог, що відображають потреби цільової аудиторії та обмеження технічного середовища. Чітке формування вимог на цьому етапі є принципово важливим, оскільки саме воно визначає межі прототипу, дозволяє відокремити функціонал, обов'язковий для демонстрації центральної ідеї платформи, від можливостей, віднесених до перспективного розвитку, та слугує підставою для подальших архітектурних рішень. Вимоги сформовано на основі аналізу предметної області та недоліків наявних рішень, розглянутих у першому розділі, що дозволило врахувати реальні потреби користувачів і уникнути типових обмежень платформ-аналогів.

Функціональні вимоги описують поведінку системи з точки зору користувача та визначають перелік операцій, які платформа має забезпечувати в рамках прототипу (Табл. 2.1).

Таблиця 2.1

Функціональні вимоги до системи

N	Модуль	Опис вимоги
1.	2.	3.
1.	Автентифікація та керування обліковим записом	Реєстрація користувача з підтвердженням електронної пошти
2.	Автентифікація та керування обліковим записом	Вхід за допомогою електронної пошти й паролю або через обліковий запис «Google»
3.	Автентифікація та керування обліковим записом	Відновлення доступу до облікового запису через електронну пошту
4.	Автентифікація та керування обліковим записом	Редагування налаштувань профілю користувача

1.	2.	3.
5.	Персональний бар	Додавання інгредієнтів до персонального бару автентифікованим користувачем
6.	Персональний бар	Видалення інгредієнтів з персонального бару
7.	Персональний бар	Відображення актуального переліку інгредієнтів у барі користувача
8.	Рецепти та пошук	Каталог рецептів коктейлів із деталізованим описом інгредієнтів (бренд, тип)
9.	Рецепти та пошук	Перегляд детальної сторінки рецепту з покроковим описом приготування
10.	Рецепти та пошук	Пошук рецептів за інгредієнтами, наявними у персональному барі користувача
11.	Рецепти та пошук	Фільтрація рецептів за категоріями та іншими характеристиками

Функціональні вимоги згруповано за трьома логічними модулями, кожен з яких відповідає окремому аспекту взаємодії користувача з платформою. Модуль автентифікації та керування обліковим записом є фундаментом усієї системи, адже саме він визначає, які можливості доступні конкретному користувачеві та забезпечує захист його персональних даних. Реєстрація передбачає обов'язкове підтвердження електронної пошти, що дозволяє пересвідчитися у достовірності вказаної адреси та запобігає створенню фіктивних облікових записів. Вхід реалізовано у двох варіантах – за допомогою електронної пошти й пароллю та через обліковий запис «Google», що дає користувачеві змогу обрати зручніший для нього спосіб і знижує бар'єр для початку роботи з платформою. Можливість відновлення доступу через електронну пошту гарантує, що користувач не втратить свій обліковий запис у разі забуття пароллю, а функція редагування налаштувань профілю дозволяє самостійно керувати персональними даними та параметрами застосунку, зокрема мовою інтерфейсу.

Модуль персонального бару реалізує центральну концепцію платформи, що відрізняє її від більшості аналогів. Автентифікований користувач має можливість

додавати інгредієнти до власного бару, формуючи цифрове відображення того, чим він реально володіє вдома. Можливість видалення інгредієнтів дозволяє підтримувати цей перелік в актуальному стані відповідно до фактичних запасів, а відображення повного актуального списку забезпечує користувачеві наочний контроль над вмістом бару. Саме на основі цих даних будується подальша логіка підбору рецептів, тому конкретне ведення персонального бару є необхідною передумовою роботи ключового функціоналу системи.

Модуль рецептів та пошуку відповідає за роботу з основним контентом платформи. Система має надавати каталог рецептів коктейлів із деталізованим описом інгредієнтів, що включає зазначення бренду та типу, оскільки саме ця деталізація, як було показано в першому розділі, суттєво впливає на точність підбору та якість кінцевого результату. Детально сторінка рецепту з покроковим описом приготування забезпечує користувача всією інформацією, необхідною для відтворення напою. Найважливішою вимогою цього модуля є пошук рецептів за інгредієнтами, наявними у персональному барі користувача, що реалізує головну ідею платформи – інверсію традиційного сценарію, за якої не користувач підлаштовується під рецепт, а система визначає, що можна приготувати з наявного набору. Доповнює цей механізм фільтрація рецептів за категоріями та іншими характеристиками, яка дозволяє звужувати результати пошуку відповідно до уподобань користувача.

Нефункціональні вимоги визначають якісні характеристики системи, що не стосуються безпосередньо її функціональності, але суттєво впливають на надійність, безпеку та зручність використання платформи (Табл. 2.2).

Таблиця 2.2

Нефункціональні вимоги до системи

N	Категорія	Опис вимоги
1.	2.	3.
1.	Безпека	Паролі зберігаються у хешованому вигляді із використанням криптографічно стійкого алгоритму

1.	2.	3.
2.	Безпека	Авторизація відбувається на сервері при кожному запиті на основі актуальних даних, що унеможлиблює використання застарілих прав доступу
3.	Безпека	Вхід через «Google» реалізовано за допомогою code-flow [12] для виключення ризику перехоплення токена
4.	Безпека	Статичні ресурси з CDN недоступні за прямими посиланнями; доступ надається сервером на короткий час за токеном автентифікації
5.	Безпека	Захищена передача даних за протоколом HTTPS
6.	Продуктивність	Обробка запитів із мінімальною затримкою завдяки «ASP.NET Core» та оптимізованим запитам до БД
7.	Продуктивність	Роздача статичних ресурсів (зображення, медіа-файли) через CDN незалежно від основного API-сервера
8.	Масштабованість	Можливість розширення функціоналу без суттєвих змін у наявному коді завдяки принципам чистої архітектури
9.	Масштабованість	Проектування REST API з розрахунком на обслуговування як веб-застосунку, так і мобільного додатку
10.	Доступність та SEO	Підтримка серверного рендерингу (SSR) для коректної індексації сторінок пошуковими системами
11.	Доступність та SEO	Адаптивний інтерфейс із коректним відображенням на пристроях із різними розмірами екрану
12.	Надійність	Валідація вхідних даних як на стороні клієнта, так і на стороні сервера
13.	Надійність	Супровід критичних операцій інформативними повідомленнями про помилки

Нефункціональні вимоги охоплюють п'ять категорій, що визначають якість системи незалежно від конкретного функціоналу. Найбільша увага приділена безпеці, оскільки платформа працює з персональними даними користувачів та обліковими записами, компрометація яких є неприпустимою. Зберігання паролів виключно у хешованому вигляді із застосування криптографічно стійкого алгоритму гарантує, що навіть у разі несанкціонованого доступу до бази даних вихідні паролі

залишаються нерозкритими. Принципово важливим рішенням є винесення авторизації на сервер, де права доступу перевіряються при кожному запиті на основі актуальних даних, а не закладаються у виданий раніше токен. Такий підхід унеможливорює використання застарілих прав і дозволяє миттєво реагувати на їх зміну чи відкликання. Вхід через «Google» реалізовано за допомогою механізму code-flow [12], за якого обмін авторизаційного коду на токен відбувається безпосередньо між серверами, що виключає ризик перехоплення токена у трафіку браузера. Додатковий рівень захисту забезпечує організацію доступу до статичних ресурсів: файли у мережі доставки контенту недоступні за прямими посиланнями, а доступ до них надається сервером лише на короткий час за допомогою підписаного токена. Усі взаємодії між клієнтом і сервером здійснюються виключно за захищеним протоколом HTTPS, що унеможливорює перехоплення та підміну даних під час передачі.

Вимоги до продуктивності спрямовані на забезпечення комфортної швидкодії системи навіть за умов зростання навантаження. Серверна частина має опрацьовувати запити з мінімальною затримкою, чому сприяють як висока продуктивність обраної платформи, так і оптимізація запитів до бази даних. Окремою вимогою є роздача статичних ресурсів, зокрема зображень та медіа-файлів, через мережу доставки контенту незалежно від основного API-сервера, що дозволяє розвантажити сервер від ресурсоємних операцій передачі великих файлів і зосередити його обчислювальні потужності на обробці бізнес-логіки.

Вимоги до масштабованості впливають із того, що розроблювальний прототип від самого початку проєктується як основа для значно ширшого сервісу. Архітектура системи має забезпечувати можливість розширення функціоналу без суттєвих змін у наявному коду, що досягається завдяки застосуванню принципів чистої архітектури та чіткому розподілу відповідальності між шарами. REST API проєктується з розрахунком на те, що в перспективі він обслуговуватиме не лише веб-застосунок, а й мобільний додаток та потенційні сторонні інтеграції, тому інтерфейс взаємодії має бути універсальним і не прив'язаним до конкретного типу клієнта.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				31

Категорія доступності та пошукової оптимізації відображає специфіку платформи з публічним контентом. Підтримка серверного рендерингу є обов’язковою умовою коректної індексації сторінок пошуковими системами, адже рецепти повинні легко знаходитися у пошуковій видачі, що є важливим каналом залучення нових користувачів. Інтерфейс має бути адаптивним і коректно відображатися на пристроях із різними розмірами екрану, оскільки значна частина аудиторії взаємодіятиме з платформою саме з мобільних пристроїв. Завершує перелік категорія надійності, яка передбачає валідацію вхідних даних одночасно на стороні клієнта та на стороні сервера: перевірка на клієнті забезпечує швидкий зворотній зв’язок і зручність використання, тоді як дублююча перевірка на сервері гарантує цілісність даних незалежно від поведінки клієнта. Усі критичні операції мають супроводжуватися інформативними повідомленнями про помилки, що допомагають користувачеві зрозуміти причину збою та подальші дії, замість загальних чи технічних формулювань.

Визначений перелік функціональних та нефункціональних вимог формує цілісне технічне завдання на розроблення прототипу та слугує орієнтиром для всіх подальших етапів проєктування. Саме на основі цих вимог у наступних підрозділах обґрунтовуються архітектурні рішення, проєктується модель даних та реалізуються конкретні модулі системи, а відповідність кінцевого продукту визначеним вимогам перевіряється на етапі тестування у третьому розділі.

2.2. Проєктування архітектури та моделей даних

Розпочнемо з планування архітектури платформи в цілому. За основу серверної архітектури для REST API було взято шаблон проєкту на базі ASP.NET, який розроблявся під час зимової переддипломної практики та вже реалізує певний базовий функціонал [2]. Зокрема, шаблон закладає підхід чистої архітектури (Clean Architecture) [1] у поєднанні з патерном CQRS [11], а також містить базову автентифікацію та авторизацію користувачів за допомогою JWT-токенів [23] і механізм валідації форм. Окрім цього, у ньому вже реалізовано кешування запитів та

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						32

обробку задач у фоні, обмеження кількості запитів (Rate Limiting) [32] для захисту від надмірного навантаження, а також логування та обробку помилок за стандартом RFC 7807.

Серверна частина платформи побудована на основі принципів чистої архітектури (Clean Architecture), яка передбачає чіткий поділ системи на концентричні шари з напрямком залежностей лише всередину – від зовнішніх деталей реалізації до ядра бізнес-логіки [1]. Така організація дає змогу розвивати та тестувати кожен шар незалежно, змінювати технологічні рішення без торкання бізнес-правил і забезпечує виконання вимог масштабованості системи, визначені раніше.

Проект поділено на чотири шари, що відповідають рівням відповідальності (Рис. 2.1, Додаток А).

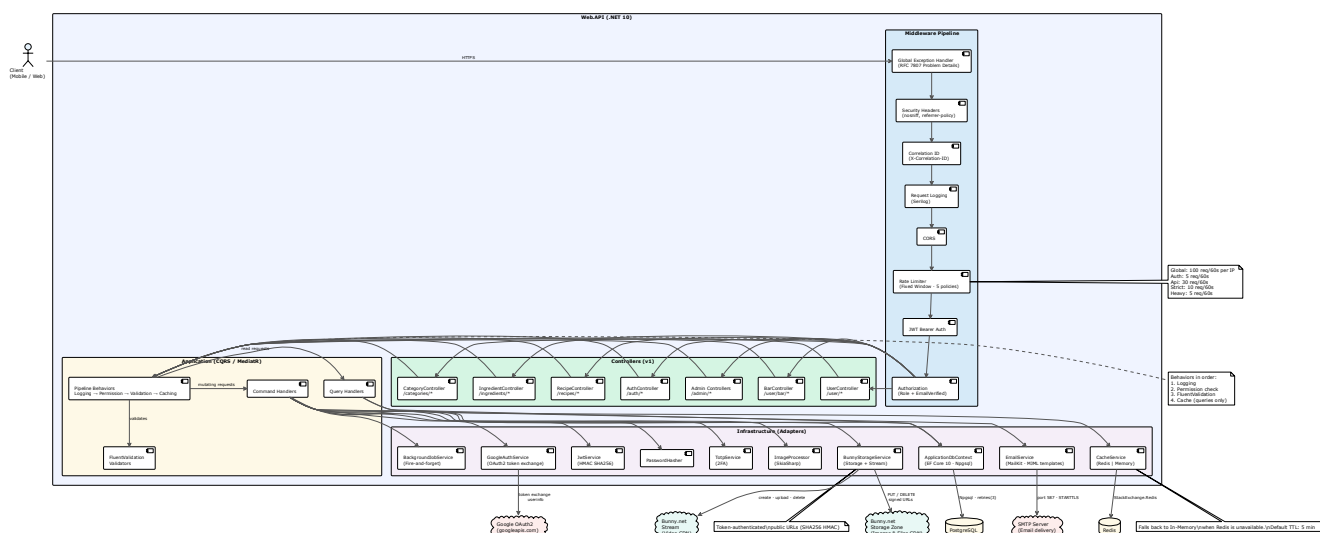


Рис. 2.1. Загальна система взаємодії між шарами та зовнішніми сервісами

Шар домену (Domain) є ядром системи та не залежить від жодного іншого шару. Він містить сутності предметної області, перерахування (Enum), доменні події та базові класи [10]. Усі основні сутності наслідують клас «AuditableEntity», який автоматично зберігає час створення та модифікації запису, а також ідентифікатор відповідного користувача. Сутності, що виступають кореневими агрегатами (User, Recipe, Category), реалізують інтерфейс «IAggregateRoot» та можуть генерувати доменні події, наприклад, для асинхронного видалення файлів з CDN після видалення аватару.

Шар застосунку (Application) містить бізнес-логіку і реалізує патерн CQRS (Command Query Responsibility Segregation) [11] за допомогою бібліотеки MediatR [33]. Усі операції поділяються на команди (що змінюють стан системи) та запити (що лише зчитують дані). Кожна операція реалізується у вигляді окремого класу-запиту та відповідного обробника, що забезпечує єдину відповідальність і тесто-придатність. Між запитом та обробником у ланцюжку MediatR виконуються Pipeline Behaviors – крос-функціональні поведінки, що застосовуються автоматично до кожної операції в такому порядку [33]. Першим виконується логування запиту, далі відбувається перевірка дозволів користувача, після чого вхідні дані проходять валідацію засобами Fluent Validation, відповідної команди або запиту, а завершальною ланкою для запитів є кешування даних.

Шар також визначає інтерфейси інфраструктурних сервісів: «IApplicationDbContext», «ICacheService», «IStorageService», «ICurrentUserService», «ILocaleService», реалізації яких знаходяться в шарі інфраструктури.

Шар інфраструктури (Infrastructure) містить конкретні реалізації інтерфейсів: доступ до бази даних через EF Core 10 [9], генерацію та валідацію JWT-токенів, хешування паролів, генерацію TOTP-кодів для двофакторної автентифікації, обробку зображень, взаємодію із CDN Bunny.net [5], надсилання пошти через MailKit, кешування в Redis та інтеграцію з Google OAuth2 [12]. Такий підхід дає можливість, за необхідності, замінити будь-який адаптер (наприклад, перейти з Redis на інший кеш або з Bunny.net [5] на Amazon S3) без змін у бізнес-логіці.

Шар Web.API є зовнішнім шаром і відповідає за прийняття HTTP-запитів. Він містить контролери, що отримують параметри запиту, формують відповідний об'єкт команди або запиту і передають його до MediatR [33]. Перш ніж запит потрапить до контролера, він проходить через конвеєр проміжного програмного забезпечення (Middleware Pipeline), ланки якого виконуються у чітко визначеному порядку. Спочатку працює глобальна обробка виключень за стандартом RFC 7807 Problem Details, що гарантує єдиний формат відповідей про помилки. Далі до

відповіді додаються HTTP-заголовки безпеки, зокрема nosniff та referrer-policy, після чого кожному запиту присвоюється ідентифікатор кореляції (X-Correlation-ID) для наскрізного відстеження. Наступним кроком виконується логування засобами Serilog та перевірка політики CORS. Після цього застосовується обмеження кількості запитів, що захищає сервер від надмірного навантаження, і лише потім відбувається автентифікація через JWT Bearer. Завершальною ланкою конвеєра є авторизація, яка ґрунтується на ролях та підтверджених дозволах користувача.

Обмеження кількості запитів реалізовано через п'ять окремих політик із різними параметрами:

- Глобальна: 100 запитів на 60 секунд;
- Автентифікація: 5 запитів на 60 секунд;
- API-запити: 30 запитів на 60 секунд;
- Суворі: 10 запитів на 60 секунд;
- Ресурсоємні операції: 5 запитів на 60 секунд.

Це дозволяє ефективно захистити чутливі ендпоінти, зокрема входу та відновлення пароллю, від атак перебору [32].

При проєктуванні системи безпеки було взято до уваги вимоги, визначені раніше. Автентифікація реалізована на основі пари JWT access-токенів та refresh-токенів [23]. Access-токен підписується алгоритмом HMAC SHA256 і містить ідентифікатор користувача. Refresh-токени зберігаються у базі даних і прив'язані до конкретного пристрою та user-agent, що дозволяє користувачу переглядати та відкликати активні сесії з інших пристроїв.

Вхід через обліковий запис Google реалізовано з використанням authorization code flow: клієнт отримує від сервера посилання авторизації, виконує перенаправлення до Google, а отриманий короткоживучий код передає назад на сервер, який обмінює його на токен безпосередньо між серверами [12]. Такий підхід виключає можливість перехоплення access-токенів Google у трафіку браузера, що відповідає вимогам безпеки.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						35

Система підтримує двофакторну автентифікацію (2FA) на основі протоколу TOTP (Time-based One-Time Password). Резервні коди для відновлення доступу зберігаються у хешованому вигляді. Для скидання пароля та підтвердження електронної пошти використовуються одноразові токени з часом дії, які після використання або після закінчення строку дії стають недійсними.

Авторизація в системі ґрунтується не лише на перевірці ролей, а й на гранульованих дозволах (permission-based). Модель ролей спроєктована розширюваною за принципом «роль – набір дозволів»: у межах прототипу реалізовано ролі User та Admin, а проміжні рівні (контент-менеджера, модератора) закладено архітектурно як напрям подальшого розвитку. Перевірка відбувається в «PermissionBehavior» на рівні pipeline MediatR – кожна команда або запит може бути анотована атрибутом «RequirePermission», що усуває необхідність дублювати перевірки в кожному обробнику [33].

Доступ до медіа-файлів (зображень, відео) реалізовано через тимчасові підписані URL-адреси. Статичні ресурси зберігаються в CDN Bunny.net і не є загальнодоступними [5]. При відповіді на запит клієнта сервер генерує короткочасне посилання, підписане алгоритмом HMAC SHA256 із зазначеним часом дії. Таким чином, навіть якщо URL-адреса потрапить третій стороні, вона перестає бути дійсною після закінчення строку дії [3].

Модель даних розроблено з урахуванням функціональних вимог усіх трьох модулів системи та відображено у діаграмі сутність-зв'язок (ERD). Сутності організовані у п'ять логічних пакетів (Рис. 2.2, Додаток Б).

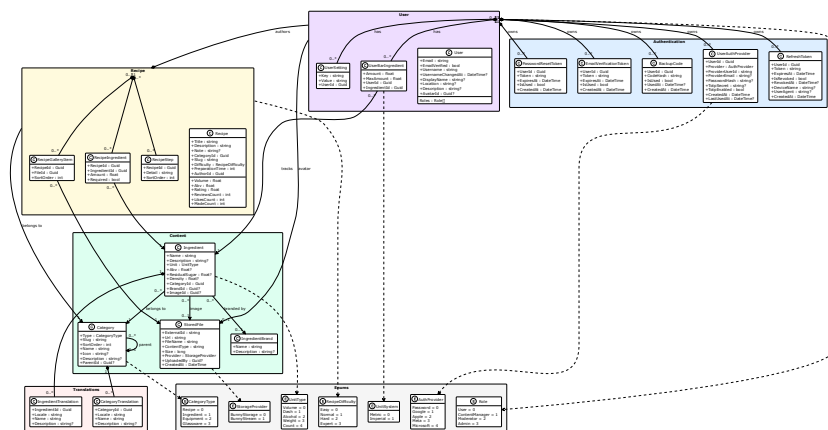


Рис. 2.2. Діаграма зв'язків сутностей (ERD)

Пакет автентифікації (Authentication) забезпечує гнучку систему ідентифікації. Сутність «UserAuthProvider» пов'язує обліковий запис користувача з конкретним методом входу (пароль, Google тощо) і зберігає хеш пароля та секрет TOTP. Такий підхід дозволяє одному обліковому запису мати кілька методів входу. Сутності «RefreshToken», «PasswordResetToken», «EmailVerificationToken» та «BackupCode» реалізують відповідні аспекти управління сесіями та відновлення доступу.

Пакет користувача (User) містить центральну сутність «User», яка є кореневим агрегатом і зберігає публічні дані профілю: ім'я, відображуване ім'я, локацію, опис та аватар. Зміна імені користувача обмежена тримісячним таймером, що відстежується полем «UsernameChangedAt». Налаштування профілю (мова, система одиниць тощо) зберігаються у пов'язаних записах «UserSetting» за принципом ключ-значення. Персональний бар користувача реалізовано через сутність «UserBarIngredient», що пов'язує користувача з конкретним інгредієнтом.

Пакет рецептів (Recipe) моделює рецепти коктейлів. Агрегат «Recipe» містить метадані (назва, опис, категорія, складність, час приготування, автор), а також розраховані властивості – загальний об'єм у мілілітрах та вміст алкоголю (ABV), які обчислюються на основі інгредієнтів і зберігаються денормалізовано для ефективного пошуку та сортування без необхідності виконувати складні агрегатні запити при кожному зверненні. Покроковий опис приготування зберігається у вигляді впорядкованих записів «RecipeStep». Галерея медіа-файлів реалізована через «RecipeGalleryItem», кожен з яких посилається на запис «StoredFile» з пакета контенту.

Пакет контенту (Content) охоплює сутності «Ingredient», «Category», «IngredientBrand» та «StoredFile». Категорії організовано у рекурсивну деревоподібну структуру (self-referencing tree): кожна категорія може мати батьківську через поле «ParentId». Тип категорії «CategoryType» визначає, до чого вона відноситься: рецептів, інгредієнтів чи чогось іншого. Це дозволяє використовувати єдину гнучку структуру для класифікації різних сутностей. Для категорій реалізовано

рекурсивний обхід у ширину (BFS), що збирає ідентифікатори обраної категорії та всіх вкладених підкатегорій будь-якого рівня (Додаток Є). Цей обхід винесено в окремий кешований сервіс і застосовується при фільтрації за категорією як рецептів так і вмісту персонального бару, завдяки чому вибір батьківської категорії повертає всі відповідні елементи незалежно від глибини їх вкладеності в дереві. «StoredFile» абстрагує конкретного провайдера файлового сховища (Bunny Storage або Bunny Stream) [5] та зберігає метадані файлу, необхідні для генерації підписаних URL-адрес [3].

Пакет перекладів (Translations) містить окремі сутності для локалізованих назв та описів інгредієнтів і категорій. Такий підхід дозволяє зберігати переклади, не дублюючи повний запис сутності. При запиті локалізації застосовується стратегія резервного вибору: якщо переклад для запитаної мови відсутній – система повертає англійський варіант.

Для зниження навантаження на базу даних у системі реалізовано дворівневе кешування. Основним кешем служить Redis, до якого звертається «CacheService». При недоступності Redis система автоматично перемикається на кеш в оперативній пам'яті процесу. Кешуванням управляє окремий pipeline behavior, що активується лише для запитів, які реалізують інтерфейс «ICachableQuery» – таким чином команди ніколи не кешуються. Типовий час життя кешу становить 5 хвилин, дерево категорій кешується окремо зі значно більшим TTL (6 годин), оскільки змінюється не часто.

Зберігання медіа-файлів делегується зовнішньому CDN-сервісу Bunny.net [5], що відповідає нефункціональній вимозі про розвантаження основного сервера від роздачі статичних ресурсів. Сервіс підтримує два режими зберігання: «Storage Zone» для зображень та «Stream» для відео-контенту. Надсилання електронної пошти реалізовано через MailKit із STARTTLS на порту 587, а шаблони листів (підтвердження пошти, скидання пароля, сповіщення безпеки) формуються за допомогою MJML [17]. З'єднання з базою даних PostgreSQL [31] виконується через Npgsql з автоматичним повтором при тимчасових збоях (до 3 спроб).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				38

2.3. Розробка серверної частини (REST API)

У цьому розділі описано реалізацію трьох основних функціональних модулів серверної частини: автентифікації, персонального бару та каталогу рецептів. Для кожного модуля наведено ключові фрагменти реалізації з поясненням прийнятих архітектурних рішень.

Усі ендпоінти автентифікації зосереджені в «AuthController». Контролер не містить бізнес-логіки, він лише зчитує параметри HTTP-запиту, формує об'єкт команди та передає його до MediatR [33]. Це дозволяє логіці залишатись повністю незалежною від транспортного шару і легко тестуватися.

Перш ніж команда потрапить до обробника, pipeline behavior «ValidationBehavior» автоматично запускає усі зареєстровані валідатори для цієї команди. Візьмемо команду реєстрації «RegisterCommand» як приклад, для цієї команди валідатором виступає «RegisterCommandValidator» (Лістинг 2.1).

```
public sealed class RegisterCommandValidator : AbstractValidator<RegisterCommand>
{
    public RegisterCommandValidator()
    {
        RuleFor(x => x.Username)
            .NotEmpty().WithMessage("Username is required.")
            .MinimumLength(2).WithMessage("Username must be at least 2 characters.")
            .MaximumLength(128).WithMessage("Username must not exceed 128 characters.")
            .Matches("^[a-zA-Z][a-zA-Z0-9]*([._-][a-zA-Z0-9]+)*$").WithMessage(
                "Username must start with a letter, contain only letters, numbers, or - _ . and must not start, end, or have consecutive special characters.");

        RuleFor(x => x.Email)
            .NotEmpty().WithMessage("Email is required.")
            .EmailAddress().WithMessage("Invalid email format.")
            .MaximumLength(256).WithMessage("Email must not exceed 256 characters.");

        RuleFor(x => x.Password)
            .NotEmpty().WithMessage("Password is required.")
            .MinimumLength(8).WithMessage("Password must be at least 8 characters.")
            .MaximumLength(128).WithMessage("Password must not exceed 128 characters.")
            .Matches("[A-Z]").WithMessage("Password must contain at least one uppercase letter.")
            .Matches("[a-z]").WithMessage("Password must contain at least one lowercase letter.")
            .Matches("[0-9]").WithMessage("Password must contain at least one digit.");
    }
}
```

Лістинг 2.1 Валідатор команди реєстрації

Валідатор перевіряє ім'я користувача за регулярним виразом, що забороняє починати його зі спеціального символу або мати два спеціальних символи підряд – це одразу виключає проблемні значення, які могли б спричинити помилки при формуванні URL або відображенні. Пароль перевіряється на наявність великих і

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				39

малих літер та цифр, що відповідає мінімальним вимогам складності. Якщо хоча б одна перевірка не пройшла – обробник взагалі не викликається, а клієнт отримує структуровану відповідь про помилку.

Безпосередньо реєстрація користувача відбувається всередині обробника команди «RegisterCommandHandler» (Лістинг 2.2 – 2.3).

```
string normalizedEmail = request.Email.ToLowerInvariant();
bool emailExists = await context.Users.AnyAsync(u => u.Email == normalizedEmail, cancellationToken);

if (emailExists)
    return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.EmailTaken, "email"));

var user = User.Create(request.Email);

string passwordHash = passwordHasher.Hash(request.Password);
var authProvider = UserAuthProvider.CreatePassword(user.Id, passwordHash);
context.UserAuthProviders.Add(authProvider);

if (request.Username is null)
{
    user.SetInitialUsername(await GenerateUniqueUsernameAsync(user.Email, context, cancellationToken));
}
else
{
    string normalizedUsername = request.Username.Trim().ToLowerInvariant();

    if (normalizedUsername.Length == 0)
    {
        user.SetInitialUsername(await GenerateUniqueUsernameAsync(user.Email, context, cancellationToken));
    }
    else
    {
        bool isAvailable = await usernameService.IsAvailableAsync(normalizedUsername, cancellationToken);
        if (!isAvailable)
            return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.UsernameTaken, "username"));

        user.UpdateUsername(normalizedUsername);
    }
}

context.Users.Add(user);
```

Лістинг 2.2. Обробка реєстрації користувача

Обробник спочатку перевіряє, чи не зайнятий вказаний email. Якщо так – одразу повертається помилка «EmailTaken», не витрачаючи часу на подальші операції. Потім через статичний фабричний метод «User.Create()» створюється агрегат – такий підхід гарантує, що об’єкт не може існувати в неповному стані, оскільки всі обов’язкові поля задаються в одному місці.

Пароль ніколи не зберігається у відкритому вигляді – він відразу хешується через «IPasswordHasher» і записується до окремої сутності «UserAuthProvider», а не

безпосередньо в «User». Це зроблено навмисно, завдяки такому розділенню один обліковий запис може мати одночасно кілька методів входу (пароль і Google), і додавання нового методу не потребуватиме змін в основній сутності.

Якщо ім'я користувача не передано (наприклад, при вході через Google), воно генерується автоматично на основі email-префіксу, зайві символи відфільтровуються а у разі збігу з вже існуючим іменем додається суфікс з поточним часом.

Refresh-токен при створенні прив'язується до назви пристрою та user-agent браузера. Це дозволяє в майбутньому показати користувачу список активних сесій у зрозумілому вигляді (iPhone 14 – Safari) та дати можливість відкликати кожен з них окремо.

```
DateTime refreshTokenExpiresAt = dateTimeProvider.UtcNow.Add(jwtService.RefreshTokenExpiration);

string refreshTokenValue = jwtService.GenerateRefreshToken();
var refreshToken = Domain.Entities.RefreshToken.Create(
    user.Id,
    refreshTokenValue,
    refreshTokenExpiresAt,
    request.DeviceName,
    request.UserAgent
);

context.RefreshTokens.Add(refreshToken);

await context.SaveChangesAsync(cancellationToken);

await sender.Send(new SendVerificationEmailCommand(user.Email), cancellationToken);

string accessToken = jwtService.GenerateAccessToken(user);

return new AuthTokensResponse(
    accessToken,
    refreshTokenValue,
    AccessTokenExpiresAt: dateTimeProvider.UtcNow.Add(jwtService.AccessTokenExpiration),
    RefreshTokenExpiresAt: refreshTokenExpiresAt
);
```

Лістинг 2.3. Створення токена та відправка листа підтвердження пошти

Листа з підтвердженням email навмисно відправляють після збереження всіх змін до бази даних. Якщо відправити його раніше і транзакція не пройде – користувач отримає лист, але обліковий запис не буде створено, що призведе до плутанини. Після збереження, відправка виконується через внутрішню команду «SendVerificationEmailCommand».

Access-токен передається у тілі відповіді, а refresh-токен виключно через HttpOnly cookie. Це принципово важливо з точки зору безпеки: JavaScript-код на сторінці не може прочитати HttpOnly cookie, тому навіть якщо на сайті є XSS-

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				41

вразливість, зловмисник не зможе вкрати refresh-токен і тихо оновлювати сесію (Лістинг 2.4).

```
return result.ToActionResult(tokens =>
{
    SetRefreshTokenCookie(tokens.RefreshToken, tokens.RefreshTokenExpiresAt);
    return StatusCode(
        StatusCodes.Status201Created,
        new AccessTokenResponse(tokens.AccessToken, tokens.AccessTokenExpiresAt)
    );
});
```

Лістинг 2.4. Встановлення HttpOnly куки з токеном та відповідь

Метод розширення «ToActionResult» інкапсулює перетворення результату операції на HTTP-відповідь: у разі успіху він викликає допоміжний метод «SetRefreshTokenCookie», який встановлює refresh-токен у cookie з атрибутами HttpOnly, Secure та SameSite, а час життя cookie синхронізується з терміном дії самого токена. Сам контроллер при цьому повертає статус «201 Created» і передає у тілі відповіді лише access-токен разом із часом його завершення, тоді як refresh-токен у тіло не потрапляє взагалі. Такий розподіл забезпечує чітке розмежування відповідальності: чутливий refresh-токен лишається недосяжним для клієнтського коду, а контролер не дублює логіку формування відповіді, делегуючи її єдиному уніфікованому механізму, що використовується усіма ендпоінтами автентифікації.

Логіка входу реалізована в команді «LoginCommand», обробником якої відповідно є «LoginCommandHandler» (Лістинг 2.5 – 2.6).

```
string normalizedEmail = request.Email.ToLowerInvariant();
User? user = await context.Users.FirstOrDefaultAsync(u => u.Email == normalizedEmail,
cancellationTokens);

if (user is null)
    return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.InvalidCredentials, "password"));

UserAuthProvider? authProvider = await context.UserAuthProviders.FirstOrDefaultAsync(
p => p.UserId == user.Id && p.Provider == AuthProvider.Password,
cancellationTokens);

if (authProvider?.PasswordHash is null)
    return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.InvalidCredentials, "password"));

if (!passwordHasher.Verify(request.Password, authProvider.PasswordHash))
    return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.InvalidCredentials, "password"));
```

Лістинг 2.5. Пошук користувача та провайдера

При способі входу з неіснуючим email або невірним паролем, система повертає однакову помилку «InvalidCredentials». Якби помилки відрізнялись, зловмисник міг би визначити, чи зареєстровано в системі певну адресу, і побудувати список

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				42

реальних акаунтів. Єдина помилка для обох випадків унеможлиблює таке виведення.

```
if (authProvider.TotpEnabled)
{
    if (string.IsNullOrEmpty(request.TotpCode))
        return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.TotpRequired, "totpCode"));

    if (!totpService.VerifyCode(authProvider.TotpSecret!, request.TotpCode))
        return Result.Failure<AuthTokensResponse>(Error.From(ErrorCode.InvalidTotp, "totpCode"));
}
```

Лістинг 2.6. Перевірка наявності 2FA автентифікації

Перевірка другого фактору свідомо виконується лише після успішної валідації пароля, тому система ніколи не розкриває факт увімкнення двофакторної автентифікації для облікового запису, доки не підтверджено правильність основних облікових даних. Якщо двофакторну автентифікацію активовано, але код не передано, повертається окрема помилка «TotpRequired», яка слугує сигналом для клієнта запросити у користувача одноразовий код, а не індикатором невдалого входу. Якщо ж код передано, але він не збігається з очікуваним, повертається помилка «InvalidTotp», що дозволяє інтерфейсу відрізнити ситуацію «потрібен код» від ситуації «код невірний» і показати відповідне повідомлення. Сама перевірку коду відбувається у сервісі, який звіряє переданий код зі значенням, обчисленим на основі секрету користувача та поточного часового вікна, з урахуванням припустимого відхилення годинників між сервером і пристроєм користувача.

Автентифікація через «Google» реалізована командою «GoogleAuthCommand», головний процес обміну коду на інформацію про користувача відбувається через інтерфейс, що реалізовано в «GoogleAuthService» (Лістинг 2.7).

```
var tokenRequest = new FormUrlEncodedContent(
    new Dictionary<string, string>
    {
        ["code"] = code,
        ["client_id"] = _settings.ClientId,
        ["client_secret"] = _settings.ClientSecret,
        ["redirect_uri"] = _settings.RedirectUri,
        ["grant_type"] = "authorization_code",
    }
);

HttpResponseMessage tokenResponse = await httpClient.PostAsync(TokenEndpoint, tokenRequest,
    cancellationToken);
```

Лістинг 2.7. Обмін коду на токен з Google

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				43

Під час входу через Google, клієнт передає серверу лише одноразовий авторизаційний код. Сервер самостійно обмінює цей код на дані користувача, звертаючись до Google безпосередньо між серверами – без участі браузера. Завдяки цьому жоден токен Google ніколи не проходить через мережу браузера, де він міг би бути перехоплений.

Якщо користувач з таким email вже існує – система перевіряє, чи підключено Google як метод входу, і чи збігається Google ID. Перевірка Google ID захищає від теоретичного сценарію, коли хтось мав однаковий email у своєму Google-акаунті, без цієї перевірки він міг би потрапити в чужий акаунт.

Аватар користувача завантажується з URL, отриманого від Google, вже після того, як запис користувача збережено до бази. Порядок важливий через кругову залежність між сутностями: «User.AvatarId» вказує на «StoredFile», а «StoredFile.UploadedBy» вказує на «User» – якщо намагатися зберегти обидві сутності одночасно – виникне порушення зовнішнього ключа. Помилка завантаження аватару вважається некритичною, вона лише логується але не скасовує реєстрацію, адже аватар є косметичним елементом.

«GetMyBarIngredientsQueryHandler» формує єдиний SQL-запит, послідовно накладаючи лише ті фільтри, що передані у запиті (Додаток В).

Фільтрація за категорією враховує всю гілку дерева: якщо обрати категорію «Міцні напої», у результат потраплять інгредієнти не лише її прямих підкатегорій, а й усіх вкладених рівнів – «Віскі», «Джин», а також їхніх власних підкатегорій. Для цього обробник звертається до «ICategoryService.GetDescendantIdsAsync» (Додаток Є), який повертає ідентифікатори обраної категорії інгредієнта у цю множину. Статус критичного залишку та залишку «зверніть увагу» обчислюється безпосередньо в SQL-умові через ділення «Amount / MaxAmount», тому до пам'яті потрапляють лише відфільтровані записи, але цей функціонал розраховано на майбутнє розширення і не буде демонстровано у КР. Текстовий пошук шукає збіг у назві інгредієнта, його перекладах, назві бренду та категорії – тобто користувач може

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						44

знайти «Scotch Whiskey» пошуком українською мовою «шотландський», якщо є відповідний переклад у БД.

Додавання інгредієнту до бару користувача відбувається в обробнику «AddMyBarIngredientCommandHandler» (Лістинг 2.8), що завантажує інгредієнт разом із перекладами, брендом, зображенням і категорією одним запитом до бази, одразу фільтруючи переклади лише для активної локалізації та англійської мови як резервної. Якщо б переклади не фільтрувалися на рівні SQL, для інгредієнта з 10 перекладами завантажувались би всі 10 записів, а потім один відкидався б у пам'ять, що спровокувало б даремне навантаження.

```
Domain.Entities.Ingredient? ingredient = await context
    .Ingredients.Include(i => i.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
    .Include(i => i.Brand)
    .Include(i => i.Image)
    .Include(i => i.Category)
    .ThenInclude(c => c.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
    .Include(i => i.Category)
    .ThenInclude(c => c.Parent)
    .ThenInclude(p => p!.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
    .FirstOrDefaultAsync(i => i.Id == request.IngredientId, cancellationToken);
```

Лістинг 2.8. Підвантаження інгредієнтів

Перевірка наявності інгредієнта виконується окремим запитом «AnyAsync» безпосередньо до бази даних, а не звіркою із завантаженням раніше у пам'ять списком бару, оскільки список міг застаріти між моментом його отримання та моментом додавання. Це принципово в умовах конкурентного доступу: якщо користувач випадково ініціює два однакові запити майже одночасно (наприклад, подвійним натисканням), обидва звернуться до актуального стану бази, і лише перший встигне додати запис, тоді як другий отримає помилку «BarIngredientExists» (Лістинг 2.9). Остаточною гарантією цілісності виступає унікальне обмеження на рівні самої бази даних на пару «користувач – інгредієнт», яке унеможлиблює появу дублікатів навіть у малоймовірному сценарії, коли обидва запити пройдуть перевірку одночасно. Такий дворівневий підхід поєднує зрозумілу для користувача помилку з технічною гарантією узгодженості даних.

```
bool alreadyExists = await context.UserBarIngredients.AnyAsync(
    b => b.UserId == userId && b.IngredientId == request.IngredientId,
    cancellationToken
);
if (alreadyExists)
    return Result.Failure<BarIngredientResponse>(Error.From(ErrorCode.BarIngredientExists));
```

Лістинг 2.9. Перевірка наявності інгредієнта в барі

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				45

Формування об'єкта відповіді зосереджено в одному місці і поєднує дані з кількох джерел у плоску структуру, зручну для клієнта. Назви інгредієнта та категорії одразу повертаються локалізованими через метод «GetName», що враховує активну мову користувача з резервним відкатом на англійську, тому клієнту не потрібно виконувати додаткову обробку перекладів. Посилання на зображення формується методом «IStorageService.GenerateTemporaryUrl()» лише за наявності файлу і є підписаним посиланням на CDN з обмеженим терміном дії, що унеможливає прямий доступ до сховища за статичною адресою. Прапорці «IsCritical» та «IsAttention» обчислюються на основі частки залишку (менше 25% та менше 50% відповідно) і залишаються порожніми, якщо користувач не задав максимальну кількість, оскільки за відсутності орієнтира, поняття «критичного залишку» втрачає сенс. Цей функціонал відстеження залишків закладено на перспективу і не демонструється в межах прототипу (Лістинг 2.10).

```
return Result.Success(
    new BarIngredientResponse
    {
        Id = ingredient.Id,
        Ingredient = new IngredientShortResponse
        {
            Id = ingredient.Id,
            Name = ingredient.GetName(locale),
            ImageUrl = ingredient.Image != null ?
storageService.GenerateTemporaryUrl(ingredient.Image) : null,
            BrandName = ingredient.Brand?.Name,
            MainCategoryName = mainCategory.GetName(locale),
            Category = new CategoryShortResponse
            {
                Id = category.Id,
                Slug = category.Slug,
                Name = category.GetName(locale),
            },
        },
        Amount = barIngredient.Amount,
        MaxAmount = barIngredient.MaxAmount,
        RemainingPercent = remainingPercent,
        IsCritical = barIngredient.MaxAmount > 0 ? remainingPercent <= 0.25f : null,
        IsAttention = barIngredient.MaxAmount > 0 ? remainingPercent <= 0.50f : null,
    }
);
```

Лістинг 2.10. Формування відповіді

«GetRecipesQueryHandler» (Додаток Г) накладає всі умови фільтрації на «IQueryable» – EF Core трансліює весь ланцюжок в один SQL-запит з усіма умовами одразу, без проміжного завантаження даних у пам'ять. Це особливо важливо, оскільки каталог може містити тисячі рецептів.

Фільтрація за міцністю напою (ABV) підтримує обидві системи одиниці, якщо в налаштуваннях автентифікованого користувача встановлена імперська система вимірювання (Proof), передані значення конвертуються у метричні (ABV %) перед формуванням SQL-умови, оскільки база зберігає дані лише в метричній системі. Це дозволяє клієнту завжди надсилати значення в одиницях користувача, не знаючи про внутрішнє представлення.

Ключовий фільтр «CanMake» реалізує центральну ідею платформи – пошук «що можна приготувати зараз». Спочатку завантажується множина «HashSet» ідентифікаторів інгредієнтів з персонального бару користувача, після чого до запиту додається умова – залишити лише рецепти, де жоден обов’язковий інгредієнт не відсутній у цій множині. Необов’язкові інгредієнти у фільтрі не враховуються – рецепт все одно вважається «доступним», навіть якщо декоративної прикраси немає в барі.

При формуванні відповіді з кожного рецепту завантажується лише перше зображення галереї для обкладинки картки, замість всієї галереї. Для автентифікованих користувачів до кожної картки додається статистика: скільки обов’язкових інгредієнтів є в барі та скільки загалом, це обчислюється в пам’яті після завантаження, використовуючи вже завантажену множину ID з бару.

«GetRecipeBySlugQueryHandler» реалізує найбільш складний запит у системі, для презентації детальної сторінки рецепту (Додаток Г).

Рецепт ідентифікується парою «{username}/{slug}», а не лише «slug», що дозволяє різним авторам мати рецепти з однаковою назвою без конфлікту, за аналогією до репозиторіїв на GitHub. Локалізація та система одиниць визначаються на початку обробки і використовуються протягом всього запиту.

Переклади підвантажуються безпосередньо в Include-ланцюжку з умовою фільтрації – лише для активної локалізації та англійської як резервної. Якби фільтр виконувався пізніше, спочатку з бази підтягувалися би всі переклади для всіх інгредієнтів, що в рецепті з 10 інгредієнтами і 5 мовами означало б 50 зайвих записів.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				47

Навігаційний ланцюжок категорій (breadcrumbs) будується через «ICategoryService.BuildBreadcrumbsAsync()», який кешує дерево категорій на 6 годин, оскільки структура категорій змінюється вкрай рідко, повторювати BFS-обхід при кожному перегляді рецепту немає сенсу.

Для автентифікованих користувачів кожен інгредієнт рецепту позначається прапором «available» залежно від вмісту бару. Для відсутніх інгредієнтів система виконує додатковий пошук інгредієнта-замінника у межах тієї самої категорії бару користувача. Точність такого підбору забезпечується продуманою деревоподібною таксономією інгредієнтів, яку формує команда контент-менеджменту: коректна категоризація гарантує, що інгредієнти однієї категорії є взаємозамінними у рецепті. Якщо заміник знайдено, він повертається у полі «substitute», наприклад: «Рецепт вимагає Cointreau, але у вашому барі є Grand Marnier з тієї ж категорії апельсинових лікерів».

Кількість кожного інгредієнта конвертується із внутрішньої метричної системи в систему одиниць користувача через «UnitType.Convert()», кожен тип одиниці (об'єм, вага, кількість крапель) має власну логіку конвертації, яка застосовується окремо для кожного інгредієнта в циклі (Лістинг 2.11).

```
public static float Convert(this UnitType unit, UnitSystem from, UnitSystem to, float amount, int
digits = -1)
{
    if (from == to || unit == UnitType.Count || unit == UnitType.Dash)
        return digits >= 0 ? MathF.Round(amount, digits, MidpointRounding.AwayFromZero) : amount;

    float coefficient = unit switch
    {
        //                                     M -> I           I -> M
        UnitType.Alcohol => from == UnitSystem.Metric ? 2f : 0.5f,
        UnitType.Volume => from == UnitSystem.Metric ? 0.0338140227f : 29.5735296f,
        UnitType.Weight => from == UnitSystem.Metric ? 0.0352739619f : 28.3495231f,
        _ => 1f,
    };

    float result = amount * coefficient;

    return digits >= 0 ? MathF.Round(result, digits, MidpointRounding.AwayFromZero) : result;
}
```

Лістинг 2.11. Статичний метод конвертації одиниць вимірювання

Метод реалізовано як статичний метод-розширення над переліком «UnitType», що дозволяє викликати конвертацію природним чином безпосередньо для типу одиниці. На початку виконується швидкий вихід для випадків, коли

конвертація не потрібна або неможлива: якщо вихідна та цільова система збігаються, або якщо одиниця є лічильною («Count») чи краплею («Dash»), значення повертається без перетворення, з можливістю округлення. Для решти типів коефіцієнт перетворення обирається конструкцією «switch» залежно від одиниці та напрямку конвертації, причому міцність напою має окремий коефіцієнт, оскільки імперська система виражає її в одиницях proof, що вдвічі перевищують відсотковий показник ABV, тоді як об'єм і вага конвертуються між метричними та імперськими одиницями за відповідними множниками. Округлення виконується режимом «AwayFromZero», який забезпечує передбачуваний для користувача результат на противагу банківському округленню, а параметр «digits» дозволяє керувати точністю або повністю вимкнути округлення, коли потрібне точне проміжне значення для подальших обчислень.

2.4. Розробка клієнтського додатку (SPA)

Клієнтська частина платформи реалізована у вигляді SPA на базі фреймворку Nuxt 4 та Vue 3. Nuxt надає серверний рендеринг (SSR), що відповідає нефункціональній вимозі щодо SEO [37], та автоматично імпортує компоненти, composable-функції та утиліти, що позбавляє від необхідності писати import-оголошення вручну. Для управління станом використовується Pinia [15], для стилізації Tailwind CSS v4 [20] та Reka UI [21] як headless-компонентна бібліотека.

Центральна точка конфігурації застосунку – файл «nuxt.config.ts». Окрім підключення модулів, він містить одну нетривіальну частину – хук «pages:extend», який програмно додає метадані до маршрутів замість того, щоб прописувати «definePageMeta» у кожній сторінці вручну (Лістинг 2.12).

Хук рекурсивно обходить всі маршрути застосунку під час збирання і автоматично позначає «/user/*», як захищені, а «/admin/*», як доступні лише для ролі «admin». Це централізоване рішення, при додаванні нової сторінки в папку «pages/user/», вона автоматично стає захищеною без жодних змін у самому файлі.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						49

```

'pages:extend'(pages) {
  function applyMeta(pages: (typeof import('nuxt/schema')).NuxtPage[]) {
    for (const page of pages) {
      if (page.path?.startsWith('/user')) {
        page.meta ??= {};
        page.meta.auth = true;
      }
      if (page.path?.startsWith('/admin')) {
        page.meta ??= {};
        page.meta.auth = true;
        page.meta.authRoles = ['admin'];
      }
      if (page.children) applyMeta(page.children);
    }
  }
  applyMeta(pages);
}

```

Лістинг 2.12. Централізований хук

Безпосередню перевірку виконує глобальний middleware «auth.global.ts», який спрацює при кожній навігації (Лістинг 2.13).

```

export default defineNuxtRouteMiddleware((to) => {
  if (import.meta.server) return;

  const redirect = useState<string | null>('postLoginRedirect');
  const userStore = useUserStore();

  if (!to.meta.auth) return;

  if (!userStore.isAuthenticated) {
    redirect.value = to.fullPath;
    return navigateTo('/login');
  }

  const requiredRoles = to.meta.authRoles;
  if (!requiredRoles || requiredRoles.length === 0) return;

  const hasRequiredRoles = to.meta.authRolesAll
    ? requiredRoles.every(r) => (userStore.roles as string[]).includes(r)
    : requiredRoles.some(r) => (userStore.roles as string[]).includes(r);

  if (!hasRequiredRoles) {
    throw createError({ statusCode: 403, statusMessage: 'Forbidden', fatal: true });
  }
});

```

Лістинг 2.13. Глобальна перевірка доступу до сторінок

Middleware виконується виключно на клієнті, тому на початку передбачено ранній вихід для серверного середовища, а реальна перевірка прав на доступ до даних усе одно відбувається на сервері API при кожному запиті, тут вона слугує лише для покращення досвіду користувача, запобігаючи відображенню захищеного маршруту навіть на частку секунди. Якщо маршрут не позначено як захищений, перевірка пропускається одразу. Для захищених маршрутів неавторизований користувач перенаправляється на сторінку входу, але перед цим повний шлях зберігається у «useState», щоб після успішного входу повернути користувача саме

туди, куди він прямував. Якщо ж маршрут додатково вимагає певних ролей, застосовується одна з двох стратегій перевірки залежно від прапора «authRolesAll» - користувач має або володіти всіма зазначеними ролями, або хоча б однією з них, і за відсутності потрібних прав генерується фатальна помилка зі статусом 403, яка перенаправляє користувача на сторінку з описом проблеми.

Вся мережева взаємодія з API проходить через один об'єкт «\$api», який доступний у будь-якому місці застосунку (Лістинг 2.14).

```
const baseApi = $fetch.create({
  baseUrl: apiBase, headers: { Accept: 'application/json', },
  onRequest({ options }) {
    options.headers.set('x-api-version', apiVersion || '1');
    const locale = (nuxtApp.$i18n as Composer).locale.value;
    if (locale) options.headers.set('x-locale', locale);
    if (userStore.accessToken) {
      options.headers.set('Authorization', `Bearer ${userStore.accessToken}`);
    }
  },
});

const api = (async (request: string, options?: Parameters<typeof baseApi>[1]) => {
  try {
    return await baseApi(request, options);
  } catch (error) {
    const fetchError = error as ApiFetchError;
    if (
      userStore.isAuthenticated &&
      (fetchError.data?.errorCode === 'invalidToken' || fetchError.response?.status === 401)
    ) {
      const success = await refreshOnce();
      if (success) return await baseApi(request, options);
      navigateTo({ name: 'login' });
    }
    throw error;
  }
}) as typeof baseApi;
```

Лістинг 2.14. API плагін з автоматичним оновленням access-токена

Під час кожного запиту, плагін автоматично додає поточну мову інтерфейсу через заголовок «x-locale», щоб сервер повертав локалізований контент, це означає, що не потрібно явно передавати мову кожному виклику API. Якщо сервер повертає помилку 401 (token expired), запускається «refreshOnce()». Ця функція реалізує «promise duplication», якщо кілька запитів одночасно отримали 401, всі вони чекають на один спільний refresh-запит замість того, щоб надіслати п'ять паралельних запитів на оновлення токена. Це важливо, оскільки друга спроба використати вже недійсний refresh-токен призведе до помилки і виходу із сесії.

Стан авторизації на клієнті зберігається у Pinia-сторі «useUserStore». Access-токен зберігається виключно у пам'яті (реактивна змінна ref) без «localStorage» чи

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				51

«sessionStorage». Це навмисне рішення, пам'ять не доступна між вкладками і очищується при закритті вкладки. Саме тому при першому завантаженні сторінки необхідно відновити сесію – цю задачу вирішує плагін «user.ts» (Лістинг 2.15), який виконується один раз при ініціалізації клієнтської частини застосунку.

```
export default defineNuxtPlugin(async () => {
  if (import.meta.server) return;

  const userStore = useUserStore();
  if (userStore.isAuthenticated) {
    userStore.loading = false;
    return;
  }

  if (!userStore.accessToken) {
    const ok = await userStore.requestTokenRefresh();
    if (!ok) {
      userStore.loading = false;
      return;
    }
  }
  await userStore.fetchAuthenticatedUser();
});
```

Лістинг 2.15. Плагін відновлення сесії при перезавантаженні сторінки

При першому завантаженні access-токен відсутній у пам'яті, він не пережив перезавантаження. Плагін звертається до ендпоінта «/Auth/Refresh», передаючи HttpOnly cookie з refresh-токеном. Якщо він ще дійсний – сервер повертає новий access-токен і сесія відновлюється без залучення користувача. Якщо ні – стан залишається як «не автентифікований». Цей підхід означає, що користувач залишається залогіненим після перезавантаження, не зберігаючи жодних чутливих даних у доступному для JavaScript сховищі.

Сторінка входу реалізована як один компонент «login.vue», який управляє станом кількох логічних кроків: «lookup» (введення email), «login» (пароль), «register» (реєстрація), «otp» (двофакторна автентифікація) тощо. Стан передається в дочірні компоненти через механізм Vue Provide/Inject [19] (Лістинг 2.16).

```
const loading = ref<boolean>(false);
const step = ref<Step>('lookup');
const user = ref<ApiAuthLookupResponse | null>(null);
const form = ref<LoginFormData>({});
const originalLocale = ref<string>(locale.value);
const skipLocaleTracking = ref(false);
provide(originalLocaleKey, originalLocale);
provide(skipLocaleTrackingKey, skipLocaleTracking);
provide(loadingKey, loading);
provide(stepKey, step);
provide(userKey, user);
provide(formKey, form);
```

Лістинг 2.16. Сторінка входу, provide спільного стану для кроків

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				52

Сторінка виступає «контролером» всього потоку входу і надає спільний стан через `provide`, а кожен крок (як `LoginStep`, `LookupStep`) отримує його через `inject`. Такий підхід дозволяє кожному кроку бути незалежним компонентом у своєму файлі, але всі вони розділяють один стан форми та поточний крок. Альтернативою міг би бути `Pinia`-стор [15], але оскільки цей стан актуальний лише в межах сторінки входу – він свідомо винесений у глобальний стан.

Цікавою особливістю є обробка локалізації, коли крок «`LookupStep`» знаходить акаунт користувача, він автоматично перемикає мову інтерфейсу на ту, яку вибрав користувач у власному профілі, але при натисканні кнопки «назад» мова має повернутись до початкової. Для цього «`originalLocale`» зберігає мову на момент відкриття сторінки, а прапор «`skipLocaleTracking`» запобігає перезапису «`originalLocale`» при програмному перемиканні мови, щоб відстежувалась лише мова, обрана самим користувачем.

Для валідації реалізований власний `composable` «`useValidation`» (Додаток Д). Він приймає схему `Valibot` [30] і повертає об'єкт з повним набором інструментів для роботи з формою.

Метод «`setApiErrors`» вирішує конкретну практичну проблему – сервер може повернути помилку валідації зі стандартним кодом (наприклад, «`emailTaken`») або довільним повідомленням. Метод спочатку шукає переклад за кодом у файлах `i18n` [36], якщо переклад знайдено – показується локалізоване повідомлення, якщо ні – відображається текст, який повернув сервер. Назви полів порівнюються реєстро-незалежно, сервер може повернути «`Email`» з великої літери, а поле форми зветься «`email`», зіставлення все одно спрацює.

`Composable` надає набір готових типізованих схем для типових полів форми – електронної пошти, пароля, імені користувача та одноразового коду, кожна з яких уже містить локалізовані повідомлення про помилки, що отримують через функцію перекладу «`t`» (Лістинг 2.17, Додаток Д). Схеми побудовано на основі конвеєрів `Valibot`, причому для синхронних перевірок використовується «`pipe`», а для схем з мережевими зверненнями – «`pipeAsync`». Найскладнішою є «`vUsernameAsync`», яка

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						53

виконує асинхронну перевірку унікальності імені прямо в процесі валідації: спешу відпрацьовують синхронні правила довжини та формату, і лише якщо вони пройшли, надсилається запит до API, що уникає зайвих звернень для свідомо некоректних значень. Параметр «skipValue» дозволяє повторно використовувати цю саму схему у формі редагування профілю – якщо користувач залишив власне ім'я незмінним, перевірка унікальності пропускається, оскільки зайнятість імені самим собою не є конфліктом.

```
export const vEmail = (field: string) => {
  const { t } = useI18n();
  return v.pipe(v.string(t('validation.required', { field })), v.email(t('validation.email')));
};
export const vUsernameAsync = (field: string, skipValue?: string) => {
  const { t } = useI18n();
  const { checkUsername } = useUserStore();
  return v.pipeAsync(
    v.string(t('validation.required', { field })),
    v.minLength(2, t('validation.minLength', { min: 2 })),
    v.maxLength(128, t('validation.maxLength', { max: 128 })),
    v.regex(/^[a-zA-Z][a-zA-Z0-9]*([_-][a-zA-Z0-9]+)*$/, t('validation.username')),
    v.rawCheckAsync(async ({ dataset, addIssue }) => {
      if (dataset.issues?.length) return;
      if (skipValue !== undefined && dataset.value === skipValue) return;
      const available = await checkUsername(dataset.value as string);
      if (!available) addIssue({ message: t('validation.api.usernameTaken') });
    })
  );
};
export const vPassword = (field: string) => {
  const { t } = useI18n();
  return v.pipe(v.string(t('validation.required', { field })), v.minLength(8, t('validation.minLength', { min: 8 })));
};
export const vTOTP = (field: string) => {
  const { t } = useI18n();
  return v.pipe(
    v.string(t('validation.required', { field })),
    v.length(6, t('validation.length', { length: 6 })),
    v.digits(t('validation.digits')),
  );
};
```

Лістинг 2.17. Схеми валідації з підтримкою асинхронної перевірки

Сторінка каталогу рецептів є найбільшим компонентом у застосунку, оскільки управляє трьома незалежними списками з пагінацією (рецепти, категорії, інгредієнти для фільтрів), системою чернеток фільтрів та нескінченною прокруткою. Весь цей стан зберігається у «useState» – composable від Nuxt, що відрізняється від звичного «ref» тим, що стан прив'язаний до ключа і переживає деактивацію компонента при «keepalive: true», тобто повернення зі сторінки рецепта не скидає результати пошуку (Лістинг 2.18, Додаток Е).

```

const recipesQuery = useState<ApiRecipesRequest>('rsi:query', () => cloneDeep(defaultQuery));
const recipesQueryDraft = useState<ApiRecipesRequest>('rsi:queryDraft', () =>
cloneDeep(defaultQuery));
const recipes = useState<ApiRecipeCardResponse[]>('rsi:recipes', () => []);
const recipesTotal = useState<number | undefined>('rsi:total', () => undefined);
const recipesHasNextPage = useState<boolean>('rsi:hasNext', () => false);

const categoriesQuery = useState<ApiCategoriesRequest>('rsi:cat:query', () => ({ page: 1, size: 10,
search: '' }));
const categories = useState<ApiCategoryResponse[]>('rsi:categories', () => []);
const cachedCategories = useState<Record<Guid, ApiCategoryResponse>>('rsi:cachedCat', () => ({}));
const categoriesHasNextPage = useState<boolean>('rsi:cat:hasNext', () => false);

const ingredientsQuery = useState<ApiIngredientsQuery>('rsi:ing:query', () => ({ page: 1, size: 6,
search: '' }));
const ingredients = useState<ApiIngredientResponse[]>('rsi:ingredients', () => []);
const cachedIngredients = useState<Record<Guid, ApiIngredientResponse>>('rsi:cachedIng', () =>
({}));
const ingredientsHasNextPage = useState<boolean>('rsi:ing:hasNext', () => false);

```

Лістинг 2.18. Збереження стану каталогу через «useState» для «keepalive»

Система фільтрів побудована на принципі «чернетки», є два незалежних об'єкти запиту – «recipesQuery» (застосований) та «recipesQueryDraft» (той, що редагується у бічній панелі). Дія «застосувати» копіює чернетку в активний запит і надсилає новий запит до API. Це дозволяє користувачу налаштовувати кілька фільтрів одночасно і застосовувати їх за одне натискання, не спостерігаючи проміжних станів завантаження після кожного вибору. Пошуковий текст і сортування є виключеннями, вони застосовуються негайно з дебаунсом в 400мс, оскільки для них очікується миттєва реакція. Об'єкт запиту за замовчуванням «defaultQuery» містить також окремі поля, зарезервовані на рівні DTO під заплановані фільтри (зокрема виключення згенерованих ШІ рецептів, фільтр за типом посуду та позначку приготованих рецептів). На початковому етапі ці параметри на виведено в інтерфейс і вони не обробляються сервером, проте їх завчасне закладення в структуру запиту дозволяє розширювати фільтрацію без зміни клієнт-серверного контракту.

Універсальна нескінченна прокрутка, що використовується в більшості компонентів зі списками, реалізована через Composable «useInfiniteScroll», який використовує Intersection Observer API [14] для виклику події, якщо користувач догортає список представлених компонентів до кінця, ця подія в свій час, може оброблятися для підвантаження нових даних (Лістинг 2.19). Цей composable використовується як при перегляді рецептів, так і в середині фільтрів при завантаженні категорій та інгредієнтів, також його використання можна побачити на сторінці

персонального бару, як при перегляді наявних інгредієнтів, так і в модальному вікні при їх додаванні. На одній сторінці допускається використання декількох composable одночасно.

```
export function useInfiniteScroll(sentinel: Readonly<Ref<HTMLElement | null>>, onLoadMore: () =>
void, root?: Readonly<Ref<HTMLElement | null>>) {
  let observer: IntersectionObserver | null = null;
  const makeObserver = () => {
    if (observer) return;
    const el = sentinel.value;
    if (!el) return;
    el.style.height = '1px';
    let initial = true;
    observer = new IntersectionObserver([entry] => {
      if (initial) { initial = false; return; }
      if (entry.isIntersecting) onLoadMore();
    },
    { root: root?.value ?? null },
  );
  observer.observe(el);
};
const destroyObserver = () => {
  observer?.disconnect();
  observer = null;
};
watchPostEffect(() => {
  destroyObserver();
  makeObserver();
});
onDeactivated(destroyObserver);
onActivated(makeObserver);
onUnmounted(destroyObserver);
}
```

Лістинг 2.19. Composable нескінченної прокрутки

Елемент-«сигнал» (sentinel) – невидимий «div» висотою в один піксель, розміщений після останньої картки рецепту. Коли браузер повідомляє, що цей елемент з’явився у зоні видимості – спрацьовує завантаження наступної сторінки. Прапор «initial» ігнорує перший виклик, оскільки Intersection Observer синхронно сповіщає про поточний стан видимості в момент підключення спостерігача, без цієї перевірки keeralive-відновлення завжди запускало б зайве завантаження даних для вже видимого sentinel. При деактивації сторінки (перехід на перегляд рецепту) спостерігач відключається через «onDeactivated» і перестворюється при поверненні через «onActivated».

Весь стан та функції передаються дочірнім компонентам фільтрів через provide/inject [19] з типізованими ключами. Це надає можливість розбити складну сторінку на десятки незалежних компонентів (FilterSidebar, FilterMobile, RecipeCard, Sort тощо) без необхідності прокидати пропи через кілька рівнів вкладеності.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				56

Сторінка перегляду деталей про рецепт використовує «useAsyncData» для отримання даних від API (Лістинг 2.20).

```
const cacheKey = `recipe-${route.params.username}-${route.params.slug}`;
const nuxtApp = useNuxtApp();

const { data: recipe, refresh } = useAsyncData(
  cacheKey,
  () => recipeApi.fetchRecipeDetails(route.params.username as string, route.params.slug as string),
  {
    server: false,
    getCacheData: (key, app) => app.payload.data[key] as ApiRecipeDetailsResponse | undefined,
  },
);

const totalVolume = computed(() => (recipe.value ? servings.value * recipe.value.volume : 0));

watch(
  () => locale.value,
  () => {
    nuxtApp.payload.data[cacheKey] = undefined;
    refresh();
  },
);
```

Лістинг 2.20. «useAsyncData» з підтримкою кешу та оновленні при зміні мови

Параметр «server: false» означає, що запит виконується тільки на клієнті, сторінка інформації про рецепт з динамічним контентом (підписані URL зображень, персоналізовані дані бару) не підходить для SSR-кешування. Натомість вона використовує «getCacheData» для читання з «nuxtApp.payload.data» – внутрішнього сховища даних, куди записує плагін попереднього завантаження. При зміні локалізації, кеш очищається вручну і дані перезаписуються, оскільки рецепти є UGC-контентом з перекладами, які повертаються сервером динамічно.

Плагін «recipePrefetch.client.ts» (Лістинг 2.21) перехоплює стандартний механізм prefetch-посилань Nuxt і підвантажує деталі рецепту наперед, поки користувач лише наводить курсор на посилання, що дозволяє відобразити рецепт моментально (за можливості) коли користувач перейде на нього, у випадку коли користувач виявився занадто швидким, обираючи рецепт, або ж відбувся прямий перехід за посиланням на рецепт (коли підвантаження не могло відбутися) він побачить, що рецепт все ще завантажується. Загалом, користувач поводить доволі повільно, щоб побачити завантаження рецепту під час прямої взаємодії з вебдодатком. Таке оптимістичне завчасне підвантаження рецептів покращує досвід користувача.

```
export default defineNuxtPlugin(() => {
  const nuxtApp = useNuxtApp();

  nuxtApp.hook('link:prefetch', async (url: string) => {
    const match = url.match(/^\/recipes\/([^\/]*)\/([^\/]*)$/);
    if (!match) return;
    const username = match[1]!;
    const slug = match[2]!;
    const key = `recipe-${username}-${slug}`;
    if (nuxtApp.payload.data[key]) return;
    try {
      nuxtApp.payload.data[key] = await
nuxtApp.$api<ApiRecipeDetailsResponse>(endpoints.recipe.show(username, slug), {
      method: 'GET',
    });
    } catch {
    }
  });
});
```

Лістинг 2.21. Плагін попереднього завантаження деталей рецепту

При наведенні на посилання рецепту, Nuxt стандартно запускає prefetch HTML сторінки, плагін підписується на цю подію і додатково завантажує JSON з деталями рецепту через API, зберігаючи результат за ключем у «payload.data». Коли користувач дійсно переходить на сторінку – «getCachedData» знаходить готові дані і рецепт відображається миттєво.

Компонент «PageRecipeShowSectionIngredients» реалізує поведінку відображення доступності інгредієнтів з урахуванням персонального бару користувача (Лістинг 2.22).

```
const sortedIngredients = computed<ApiRecipeDetailsIngredientResponse[]>(() =>
[...props.ingredients]
  .sort((a, b) => (a.required === b.required ? 0 : a.required ? -1 : 1))
  .map((i) => ({ ...i, amount: i.amount * props.servings })),
);
```

Лістинг 2.22. Сортування та масштабування кількості інгредієнтів

Обчислювана властивість працює з копією масиву інгредієнтів, створеною через оператор розгортання, щоб не змінювати вхідні дані компонента безпосередньо, що відповідає принципу односторонньої передачі даних у Vue. Сортування переставляє необов'язкові інгредієнти в кінець списку, лишаючи основні складові на початку, завдяки чому декоративні прикраси не відволікають увагу від ключових компонентів напою. Одночасно кожна кількість множиться на поточне число порцій, яке регулює користувач, тому під час зміни цього значення властивість автоматично перераховується і відображені кількості оновлюються миттєво. Обидві

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						58

операції виконуються у пам'яті без додаткових звернень до API, оскільки всі потрібні дані вже отримано разом з рецептом.

Шаблон компонента динамічно змінює вигляд рядка залежно від трьох станів доступності: є у барі (зелений), відсутній але є заміник тієї ж категорії (жовтий) та відсутній (червоний). Для неавтентифікованих користувачів відображається нейтральний вигляд без кольорових індикаторів. При наявності замінника виводиться підказка з назвою альтернативного інгредієнта, яка формується на стороні сервера і передається у полі «substitute».

Компонент «PageUserBarIngredientSection» демонструє типовий патерн роботи зі списком у застосунку: паралельне завантаження статистики та інгредієнтів, пошук через дебаунс, нескінченна прокрутка і обробка подій від дочірнього компонента (Лістинг 2.23).

```
onMounted(async () => {
  if (ingredients.value.length > 0 && !needsRefetch.value) return;
  needsRefetch.value = false;
  ingredientsRequest.value.page = 1;
  const [statsResult, ingredientsResult] = await Promise.all([
    bar.fetchStats(),
    bar.fetchIngredients(ingredientsRequest.value),
  ]);
  stats.value = statsResult;
  ingredients.value = ingredientsResult.items;
  hasNextPage.value = !!ingredientsResult.next;
  loading.value = false;
  loadingIngredients.value = false;
});

onUnmounted(debouncedSearch.cancel);
```

Лістинг 2.23. Ініціалізація компонента бару з перевіркою актуальності кешу

При монтуванні компонент перевіряє, чи вже є завантажені дані, якщо так і прапор «needsRefresh» не встановлений – завантаження пропускається. Це важливо при поверненні зі сторінки перегляду рецепту, де користувач у майбутньому зможе помічати його як «зроблено», а сервер буде перераховувати залишки інгредієнтів в барі. «Keepalive» зберігає стан компонента, але список потрібно оновити, оскільки були внесені зміни в залишки. Прапор «needsRefetch» встановлюється через «useBarRefetch» з модального вікна додавання інгредієнта та у майбутньому зі сторінки перегляду рецепту, таким чином, модальне вікно та інша сторінка спілкуються з компонентом без прямого зв'язку між ними.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				59

Для відкриття модального вікна використовується універсальний Composable «useModal» разом з «modalStore» та глобальним «ModalHost», оголошеному в головному файлі, ця система дозволяє ефективно керувати модальними вікнами з різних компонентів додатку, що виключає постійне використання компонента «Teleport» і повторюваного коду (Лістинг 2.24).

```
export function useModal<T extends Component>(component: T, staticProps: Record<string, unknown> = {}) {
  const store = useModalStore();
  const id = store.register(component, staticProps);

  onScopeDispose(() => store.destroy(id));

  return {
    open: (props?: Record<string, unknown>) => store.open(id, props),
    close: () => store.close(id),
    patch: (props: Record<string, unknown>) => store.patch(id, props),
  };
}
```

Лістинг 2.24. Composable управління модальними вікнами

Система модальних вікон є одним із нетривіальних архітектурних рішень у застосунку. Замість розміщення тегу модального вікна безпосередньо в шаблоні компонента, «useModal» реєструє компонент у глобальному Pinia-сторі та повертає методи управління. «ModalHost» в «app.vue» рендерить усі зареєстровані вікна динамічно через «<component :is="..." />». Це вирішує проблему Vue, якщо модальне вікно оголошено всередині компонента із «transform» або «overflow: hidden» – воно буде візуально обрізане, оскільки fixed-позиціонування прив’язується до найближчого предка з «transform». Через глобальний рендеринг, модальні вікна завжди відображаються поверх усього вмісту, незалежно від CSS-контексту батьківського компонента. Метод «onScopeDispose» гарантує, що при знищенні компонента його модальне вікно автоматично видаляється зі стору без витоку пам’яті.

Висновки до другого розділу

У другому розділі здійснено проєктування та реалізацію ключових компонентів вебплатформи з міксології, що охоплює серверну частину (REST API) та клієнтський вебзастосунок. Розділ став центральним у роботі, оскільки саме в ньому абстрактні вимоги до системи були послідовно трансформовані у конкретні

архітектурні та технічні рішення, а згодом – у працездатний програмний код, що реалізує задеклароване ядро функціоналу платформи.

На початковому етапі було визначено повний перелік функціональних та нефункціональних вимог системи, що слугував орієнтиром для всіх подальших проєктних рішень. Функціональні вимоги структуровано за трьома модулями – автентифікації та керування обліковим записом, персонального бару та рецептів і пошуку, кожен з яких відповідає окремому аспекту взаємодії користувача з платформою. Нефункціональні вимоги охопили аспекти безпеки, продуктивності, масштабованості, доступності у контексті пошукової оптимізації та надійності, що сформувало основу для подальших архітектурних рішень і дозволило від самого початку врахувати не лише те, що система має робити, а й те, якими якісними характеристиками вона повинна володіти.

Було спроектовано архітектуру серверної частини на основі принципів чистої архітектури з поділом на чотири незалежні шари: домену, застосунку, інфраструктури та презентації. Для розмежування операцій читання та запису застосовано патерн CQRS із ланцюжком крос-функціональних поведінок, що забезпечують логування, перевірку дозволів, валідацію та кешування. Розроблено модель даних, представлену у вигляді ERD-діаграми з п'ятьма логічними пакетами сутностей, яка передбачає гнучку систему локалізації, рекурсивну деревоподібну структуру категорій та денормалізоване зберігання розрахункових властивостей рецептів для ефективного пошуку.

Реалізовано комплексну систему безпеки на основі пари JWT access/refresh-токенів, вхід через Google за допомогою authorization code flow, двофакторну автентифікацію на основі TOTP, гранульовану авторизацію дозволами, а також захищений доступ до медіа-файлів через тимчасові підписані URL-адреси. Окрему увагу приділено захисту від типових загроз – атак перебору, енумерації облікових записів, Race Condition, та викрадення токенів. Для зниження навантаження на БД впроваджено дворівневе кешування на основі Redis із резервним кешем в оперативній пам'яті.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				61

У серверній частині детально розглянуто реалізацію трьох функціональних модулів із поясненням прийнятих архітектурних рішень та наведенням ключових фрагментів коду. Продемонстровано повний шлях проходження запиту – від контролера, що не містить бізнес-логіки, через автоматичну валідацію вхідних даних до обробника команди чи запиту, а також розкрито нетривіальні аспекти реалізації, зокрема порядок збереження сутностей за наявності кругових залежностей, формування єдиного оптимізованого SQL-запиту з фільтрацією перекладів на рівні бази та конвертацію одиниць вимірювання відповідно до налаштувань користувача. Особливу увагу приділено реалізації центральної ідеї платформи – фільтра підбору рецептів, доступних для приготування з наявного вмісту персонального бару, та механізму пошуку інгредієнтів-замінників тієї самої категорії.

Розроблено клієнтський застосунок (SPA) на базі Nuxt 4 та Vue 3 із підтримкою серверного рендерингу. Реалізовано централізований захист маршрутів, API-плагін для автентифікації та дедуплікації запитів, відновлення сесії при перезавантаженні без зберігання чутливих даних у доступності JS сховищі. Спроектовано низку нетривіальних архітектурних рішень: багатокроковий потік входу, валідація форм, нескінченна прокрутка, збереження стану та попереднє завантаження.

Таким чином, у другому розділі створено працездатний прототип вебплатформи, що повністю реалізує визначені функціональні вимоги та задовольняє поставлені нефункціональні характеристики. Прийняті архітектурні рішення на серверній та клієнтській частинах не лише забезпечують коректну роботу наявного функціоналу, а й закладають масштабну архітектурну основу для подальшого розширення платформи до повноцінного соціального простору для спільноти міксологів, що й було однією з ключових цілей проектування.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				62

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ З МІКСОЛОГІЇ

3.1. Порядок встановлення та налаштування параметрів системи

Встановлення та налаштування вебплатформи вимагає розгортання всіх компонентів системи та налаштування кожного відповідного сервісу окремо.

Для опису системної архітектури платформи було створено діаграму розгортання вебплатформи (Рис. 3.1).

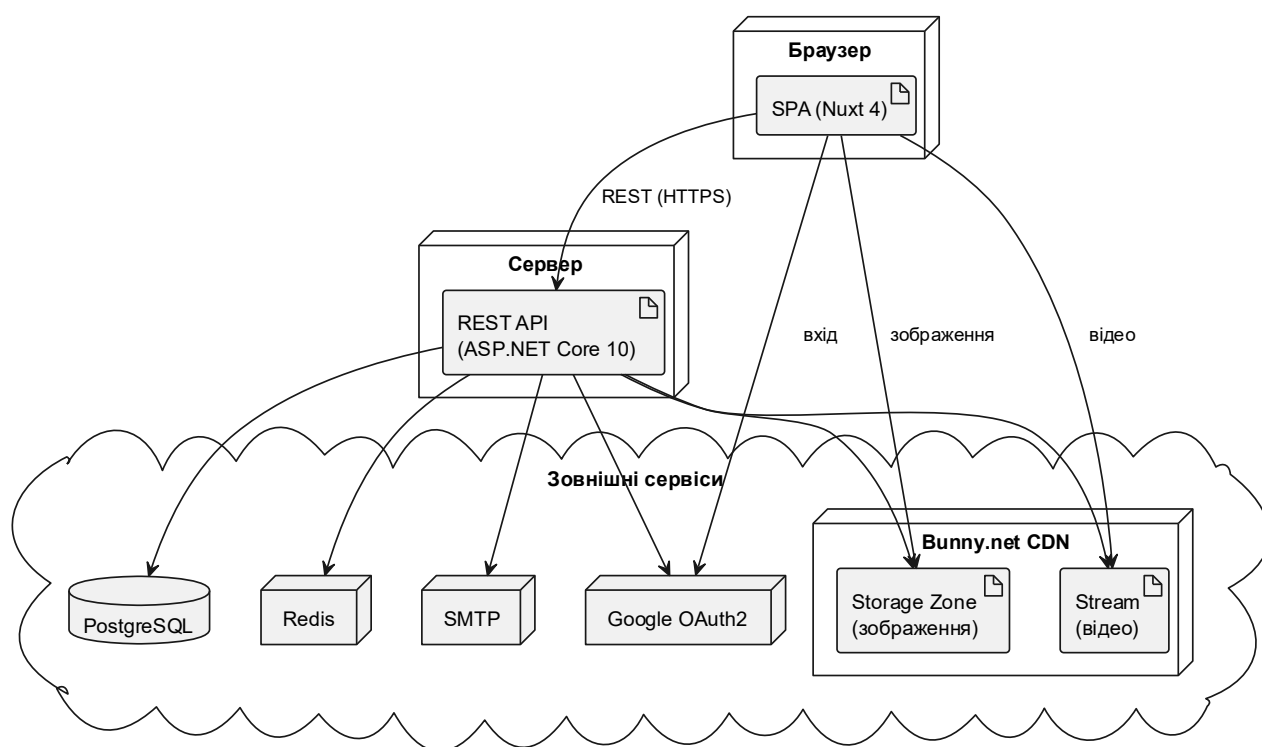


Рис. 3.1. Діаграма розгортання вебплатформи

Перед початком розгортання системи, необхідно мати доступне доменне ім'я захищене TLS сертифікатом, оскільки Google Sign-In та CDN вимагають конкретний домен для надання доступу. Також без домену та коректно налаштованої політики CORS не буде працювати автентифікація з refresh-токеном, тому що більшість сучасних браузерів буде ігнорувати куки з токеном від незахищених серверів.

Розгортання краще розпочати з зовнішніх сервісів, оскільки як API, так і SPA залежні від них. Розгортання кеш-сервера Redis не є обов'язковим для

працездатності: як згадувалося у другому розділі, API сервер також має можливість використовувати власну оперативну для зберігання кешу, що не є оптимальним рішенням, але згодиться для початку роботи вебплатформи. Розгортання бази даних може відбуватися за різними підходами: її можна розмістити поруч з API на одній машині, винести на окрему машину або скористатися готовою базою даних від провайдера, наприклад Neon DB.

Налаштування поштового сервісу не є обов’язковою вимогою для працездатності системи, але він необхідний для надсилання електронних листів для підтвердження пошти (при реєстрації з паролем) або для відновлення паролю.

CDN вимагає розгортання обох сервісів, а саме Storage Zone з відповідним Pull Zone та Stream (для відеоматеріалів). У розділі безпеки Pull Zone необхідно увімкнути параметр блокування прямого доступу до кореневого шляху, щоб звернення до файлів в обхід застосунку поверталися з помилкою 403, а за потреби обмежити коло дозволених реферерів та IP-адрес (Рис 3.2).

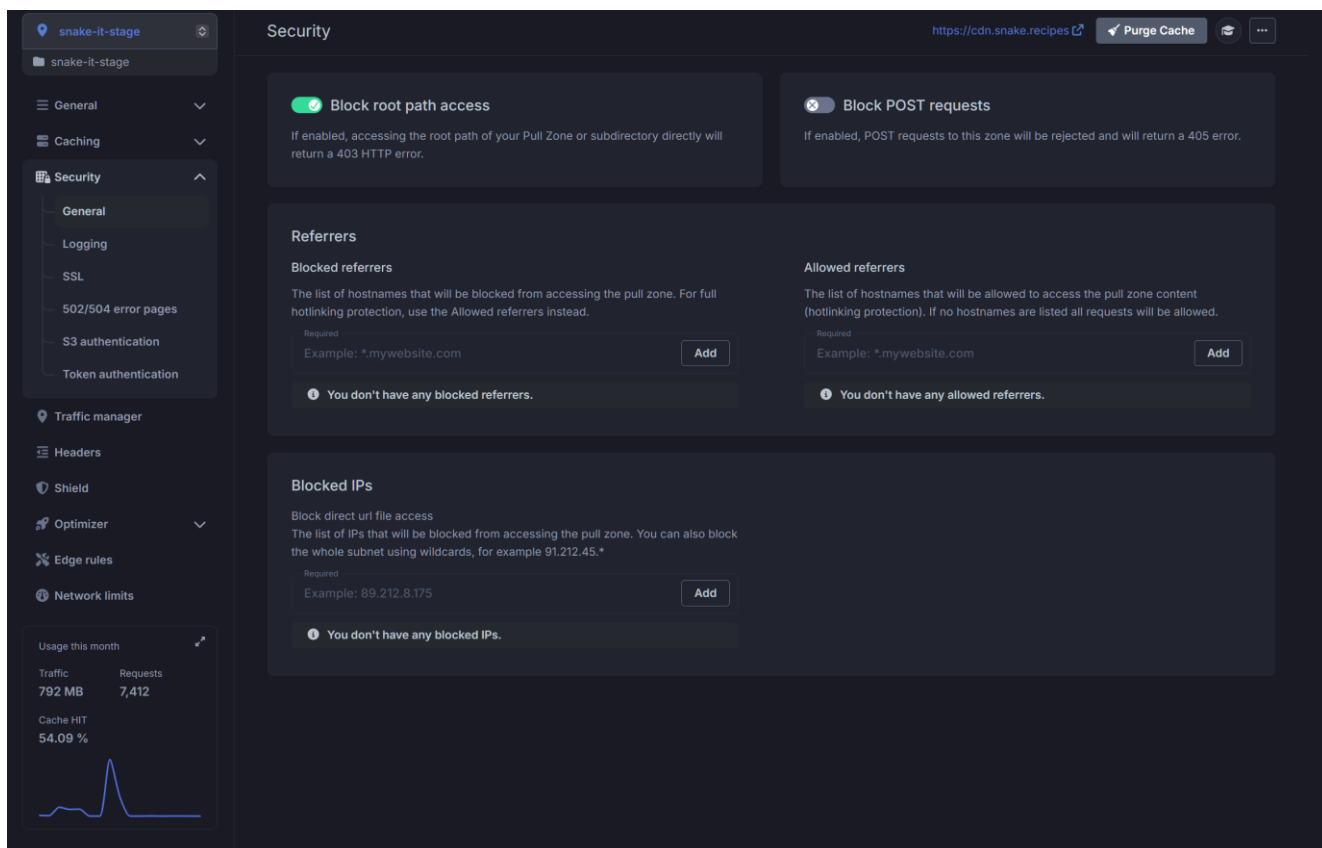


Рис. 3.2. Основні налаштування Pull Zone

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				64

Окремо вмикається автентифікація за допомогою токена, при якій кожен запит до файлу повинен містити дійсний токен зі строком дії, що унеможливорює несанкціоноване завантаження вмісту за прямим посиланням (Рис.3.3). Згенерований приватний ключ токена при цьому зберігається на стороні сервера та використовується для формування підписаних URL посилань.

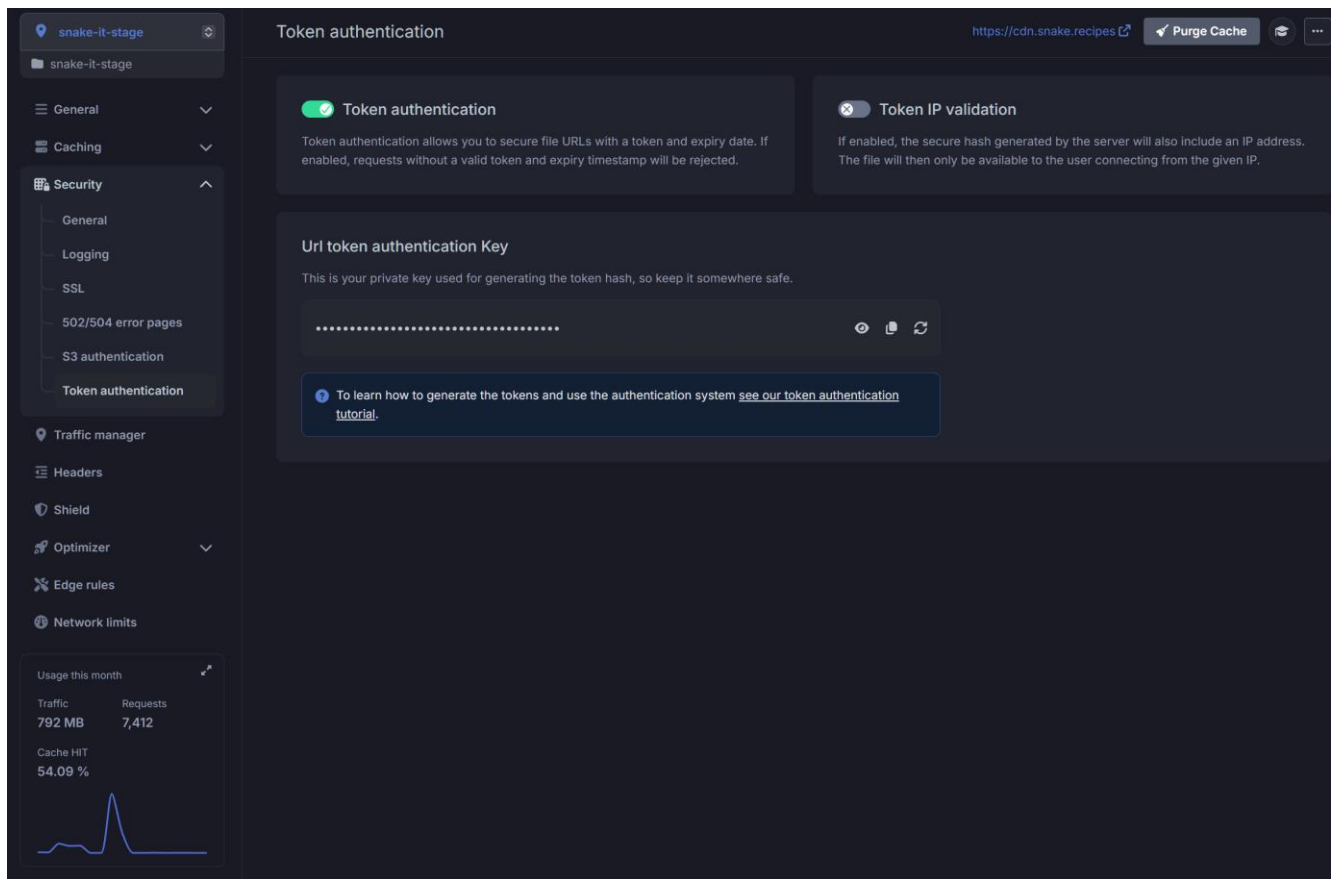


Рис. 3.3. Налаштування автентифікації Pull Zone

Налаштований у такий спосіб Pull Zone має звертатися саме до створеного раніше сховища Storage Zone як до джерела даних. Для цього у параметрах джерела обирається тип «Storage Zone» та вказується відповідне сховище, після чого всі запити користувачів обслуговуються з вузлів CDN, а самі файли підтягуються з обраного сховища (Рис. 3.4). Такий підхід дає змогу кешувати вміст географічно ближче до користувача й одночасно тримати оригінальні файли в одному захищеному місці, не допускаючи несанкціонованого доступу до них.

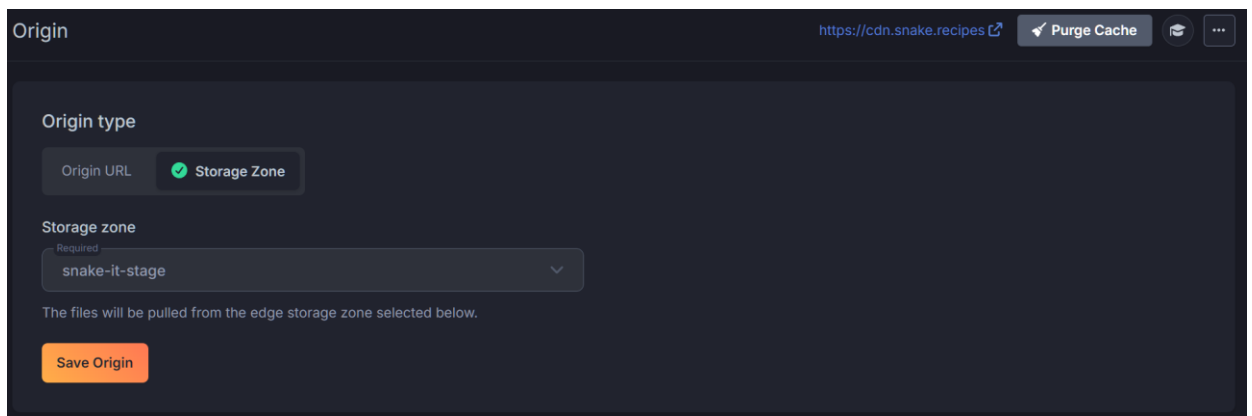


Рис. 3.4. Використання відповідного Storage Zone для Pull Zone

Налаштування безпеки сервісу Stream так само мають блокувати прямий доступ до відеофайлів. Для цього вмикається параметр блокування прямого доступу за URL та, за потреби, обмежується перелік доменів, з яких дозволено відтворення відео, а також активується автентифікація вбудованого плеєра й запитів CDN за допомогою токена (Рис. 3.5). Як і у випадку з Pull Zone, приватний ключ для формування токенів зберігається на стороні сервера, що захищає відеоматеріали від несанкціонованого вбудовування на сторонніх ресурсах.

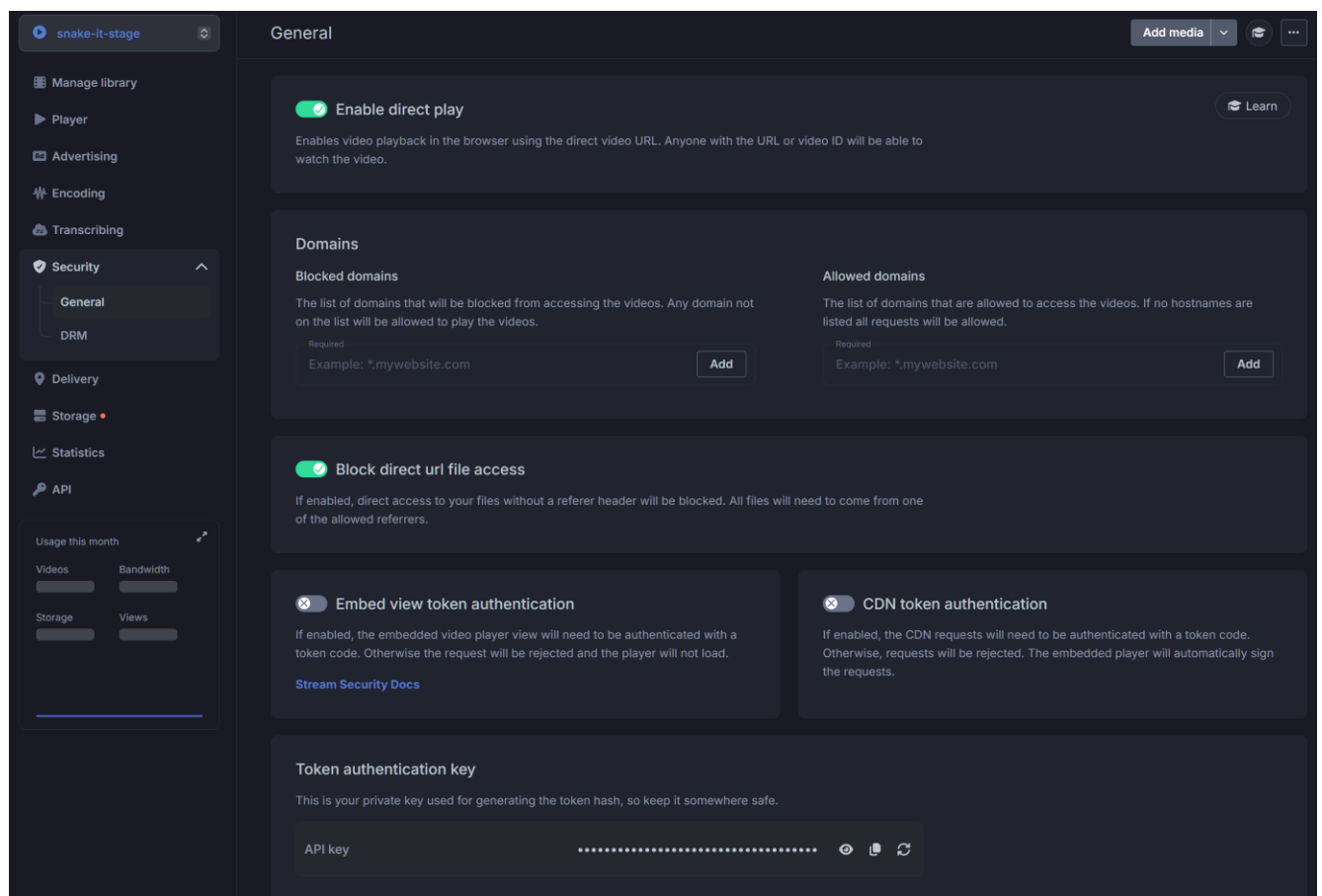


Рис. 3.5. Налаштування безпеки Stream

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				66

Сервіс Stream, своєю чергою, має використовувати ту саму інфраструктуру доставки, що й сховище файлів, тобто звертатися до спільного Pull Zone (Рис. 3.6). На сторінці керування API наведено ідентифікатор бібліотеки, ім'я хоста CDN та ключі доступу, необхідні серверу для програмного завантаження й керування відео. Саме ці значення згодом вказуються в конфігурації застосунку, завдяки чому завантаження та віддавання відеоматеріалів відбувається через єдину мережу доставки контенту.

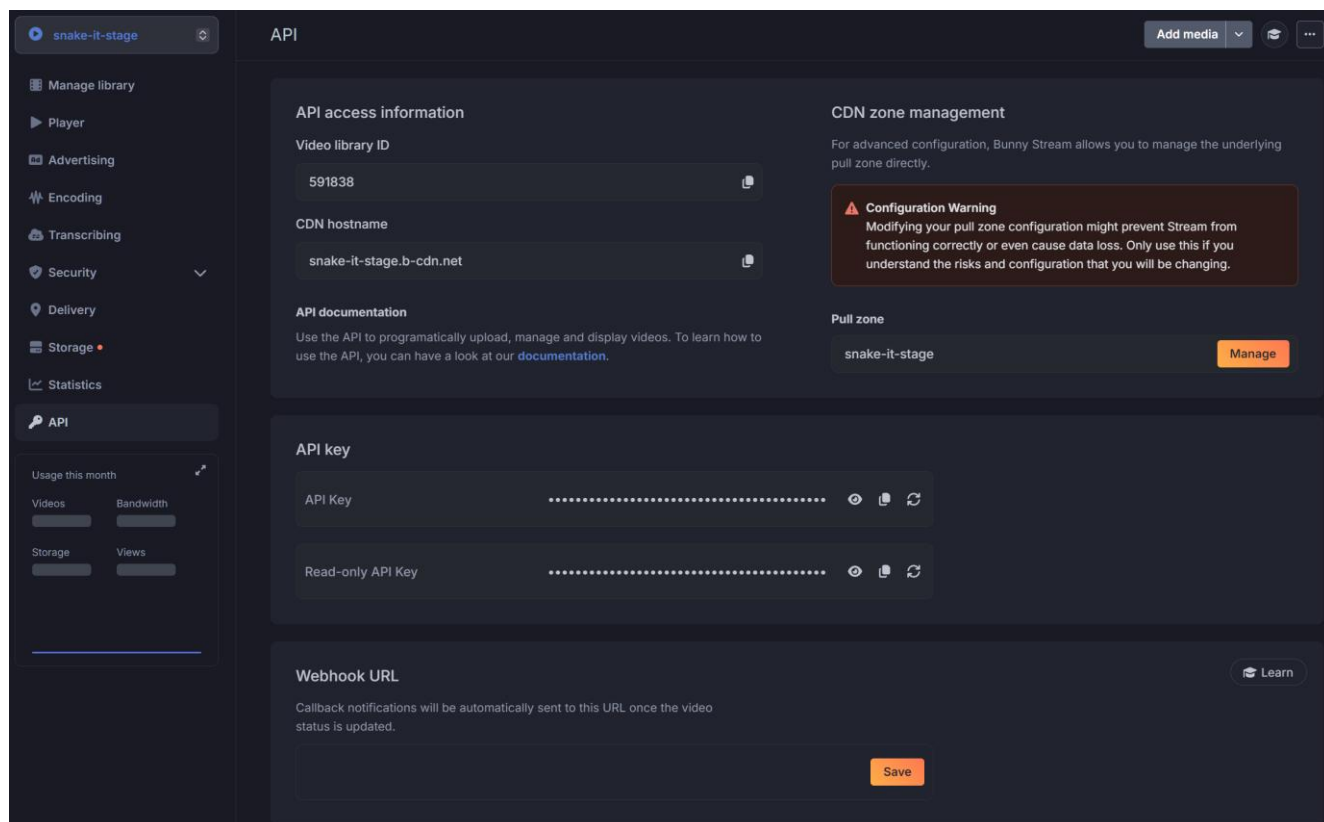


Рис. 3.6. Налаштування Pull Zone в Stream

Для налаштування Google Sign-In у консолі Google Cloud необхідно створити обліковий запис Client ID типу «Web Application» та вказати дозволені джерела для JavaScript коду, тобто домени, з яких застосунку дозволено ініціювати запити автентифікації (Рис. 3.7). Без додавання робочого домену до переліку дозволених джерел браузер блокуватиме спробу входу, тому цей крок є обов'язковим перед розгортанням SPA, оскільки Google Sign-In є однією з функціональних вимог автентифікації на вебплатформі.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				67

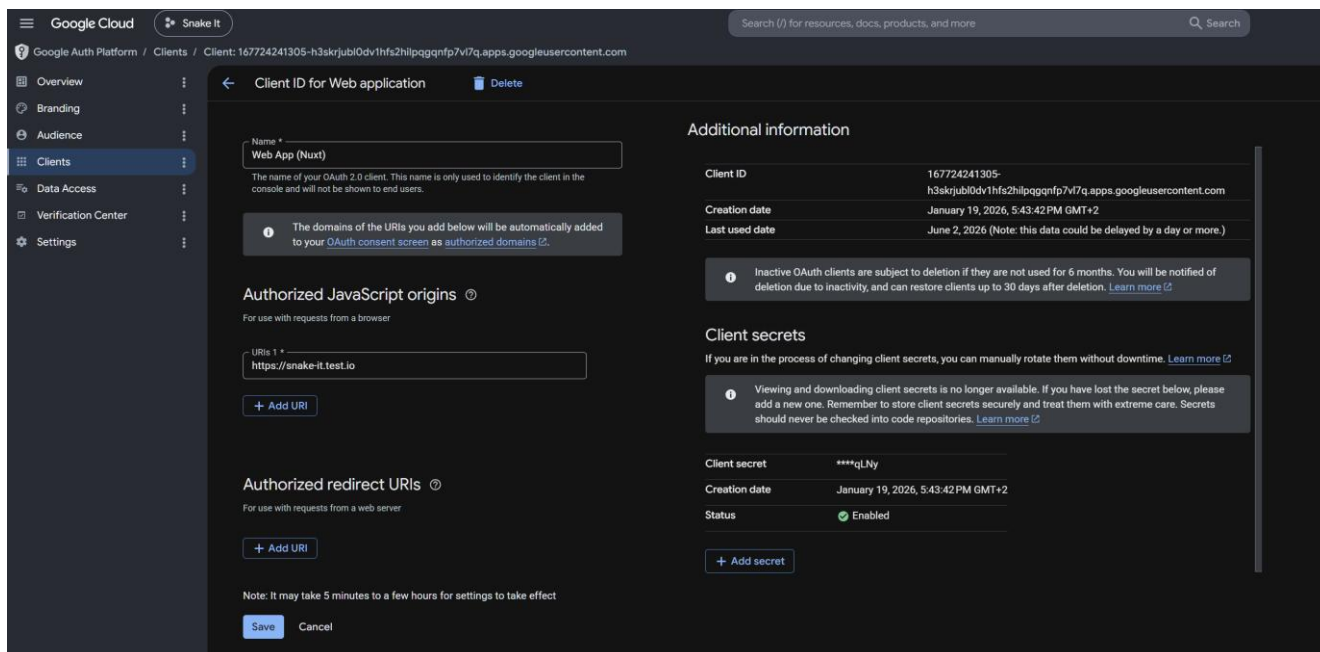


Рис. 3.7. Налаштування Google Client ID

На цій же сторінці генеруються значення Client ID та Client Secret, які згодом вказуються в конфігурації сервера й використовуються ним для перевірки автентичності токенів, отриманих від Google. Маючи всі необхідні дані можна переходити до розгортання REST API, який краще розгортати окремо від SPA застосунку. Після розгортання API серверу та налаштування домену, можна розгортати SPA, в якому треба вказати адресу API.

3.2. Структура інтерфейсу вебплатформи з міксології

Відкривши головну сторінку вебплатформи, користувач потрапляє на привітальний екран із головним закликом до дії та двома основними кнопками, які пропонують або перейти до перегляду списку рецептів, або одразу створити власний персональний бар (Рис. 3.8). Тут же розміщено короткі показники активності спільноти: кількість учасників і доданих рецептів, що формують перше уявлення про масштаб платформи. Привітальний екран оформлено в темній кольоровій гамі з контрастним градієнтовим заголовком, що задає єдиний візуальний стиль усього застосунку та підкреслює його тематику. Така структура першого екрану одразу спрямовує користувача до двох ключових сценаріїв роботи з вебплатформою.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				
						68

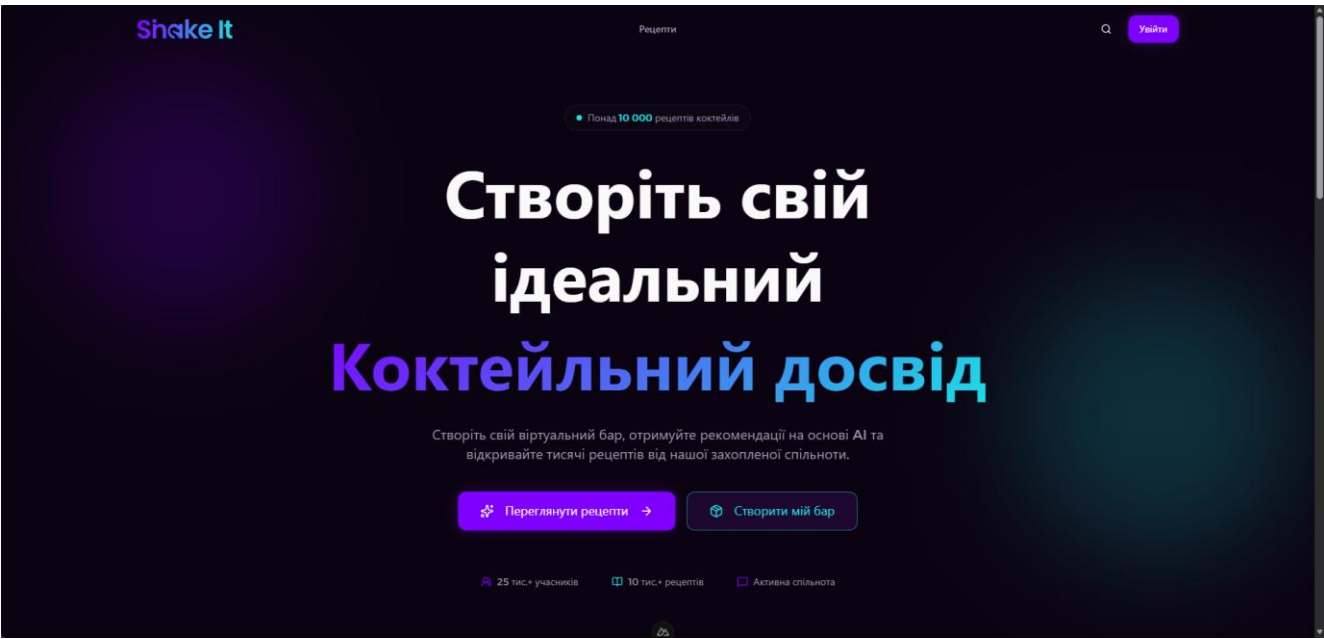


Рис. 3.8. Головна сторінка вебплатформи

Прогорнувши сторінку нижче, користувач бачить добірку рецептів, які наразі є найпопулярнішими серед спільноти (Рис. 3.9). Кожен рецепт подано у вигляді картки із зображенням, назвою, автором та основними показниками – рейтингом, кількістю вподобань і приготувань, орієнтовним часом приготування та міцністю напою. Це дає змогу швидко оцінити коктейль ще до відкриття його повної сторінки та обрати найцікавіший варіант.

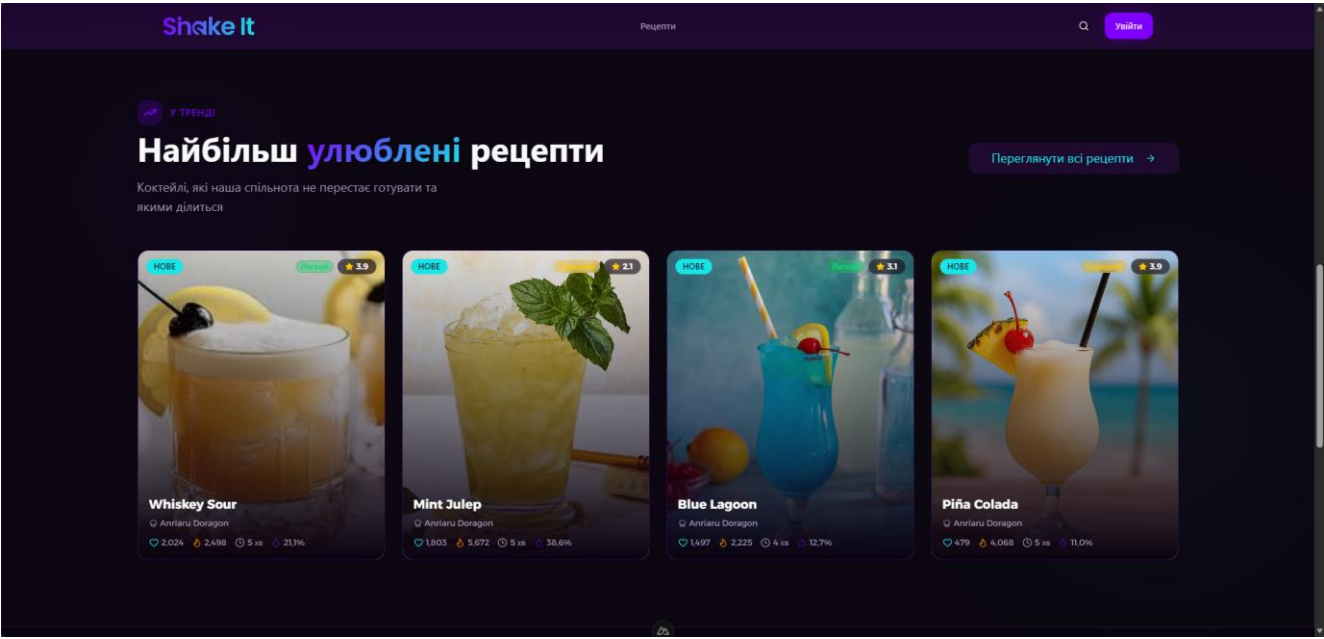


Рис. 3.9. Популярні рецепти на головній сторінці

Користувач може відкрити будь-який рецепт навіть без проходження автентифікації (Рис. 3.10). На сторінці рецепту відображається зображення коктейлю, його опис, рейтинг та міцність, перелік необхідних інгредієнтів із зазначенням кількості, а також покрокова інструкція приготування й елемент керування кількістю порцій.

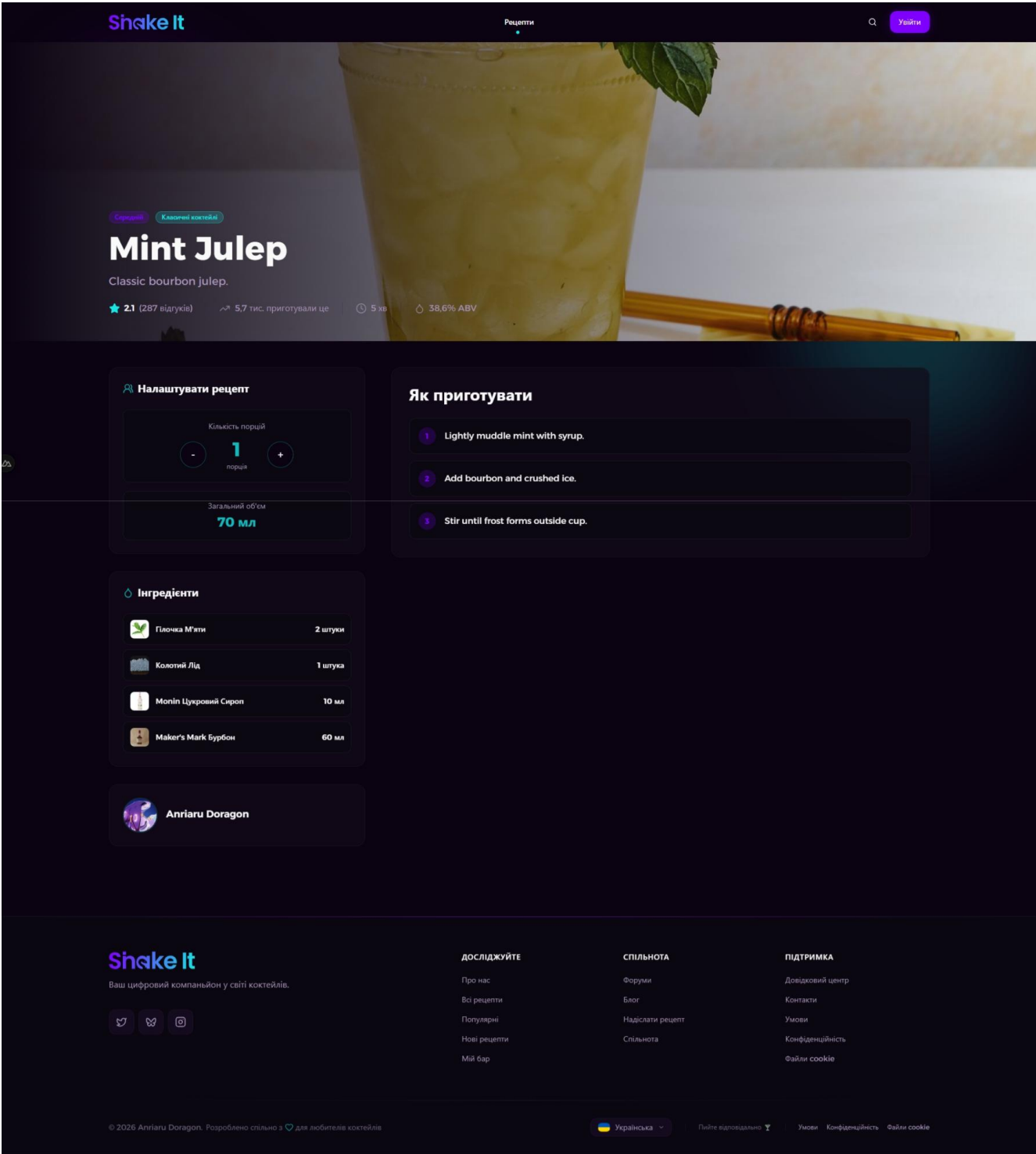


Рис. 3.10. Сторінка перегляду інформації про рецепт

Окрім перегляду окремого рецепту, користувач має змогу відкрити повний каталог усіх доступних рецептів платформи (Рис. 3.11). На поточний момент, всі дані в рецептах згенеровані користувачами (назва, опис, спосіб приготування) не перекладаються, це буде реалізовано автоматично у майбутньому, щоб автори не локалізували власні рецепти для всіх регіонів самі.

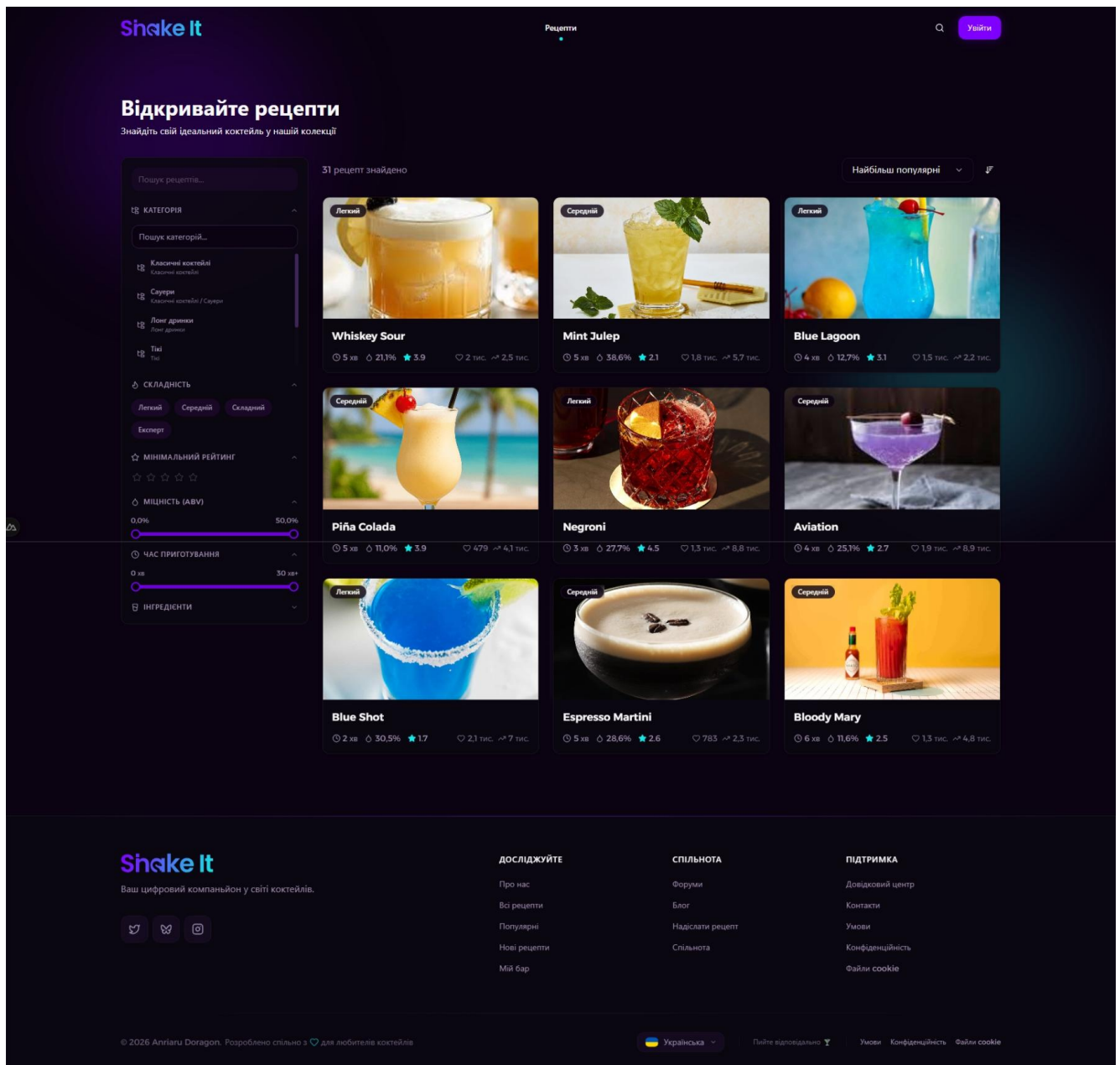


Рис. 3.11. Перегляд каталогу рецептів з фільтрами

На сторінці пошуку рецептів реалізовано нескінченне гортання, тобто коли користувач гортає до самого низу, додаток автоматично підвантажує наступну порцію рецептів з пошуку.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				71

Тут користувач може налаштовувати фільтри на свій смак та знайти підходящий для нього рецепт за обраними фільтрами та відсортувавши у бажаному порядку. Активні фільтри відображаються рядком, якщо результат не дав пошуку, це дає користувачу можливість одразу прибрати суворі фільтри (Рис. 3.12).

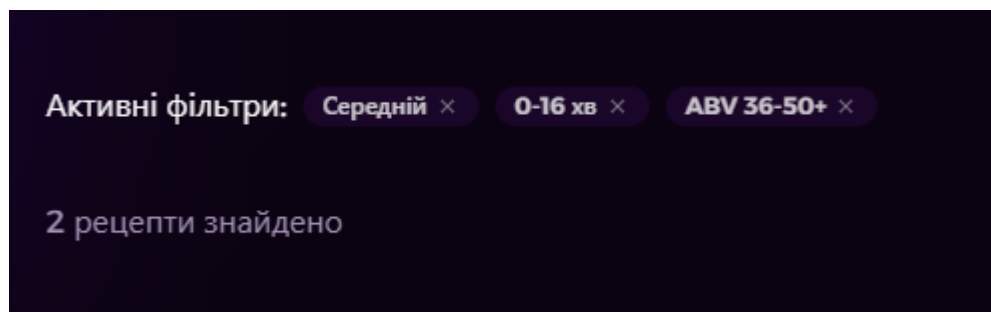


Рис. 3.12. Активні фільтри на сторінці каталогу

Користувач має можливість упорядковувати результати пошуку за різними показниками та в обох напрямках сортування, натиснувши на відповідну кнопку (Рис. 3.13). У спадному списку доступні такі критерії, як популярність, кількість уподобань та приготувань, рейтинг, тривалість приготування та міцність напою.

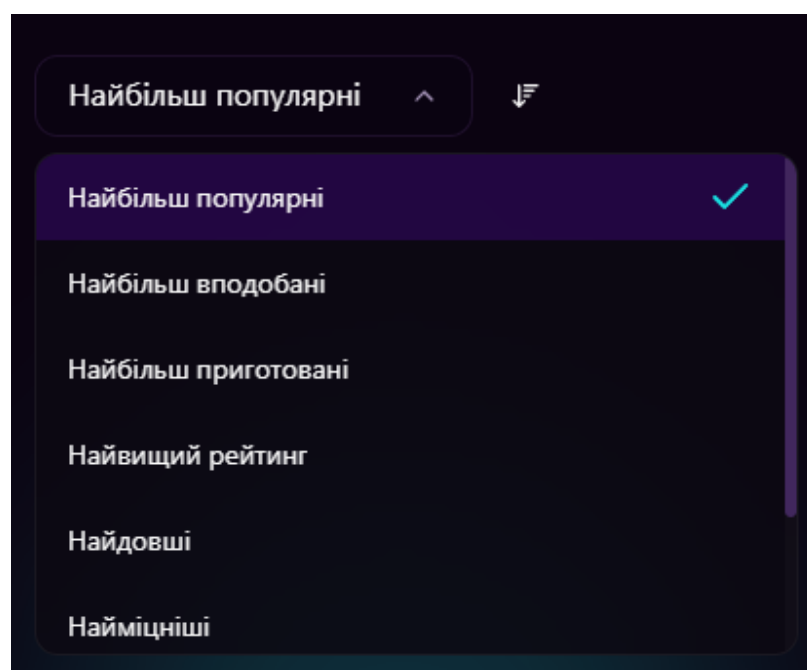


Рис. 3.13. Спадаючий список сортування каталогу

При зміні напрямку сортування, назви критеріїв відповідно змінюються зі спадних на зростаючі, наприклад «Найбільш популярні» перетворюється на «Найменш популярні», що робить поведінку інтерфейсу інтуїтивно зрозумілою та комфортною для сприйняття користувачем (Рис. 3.14).

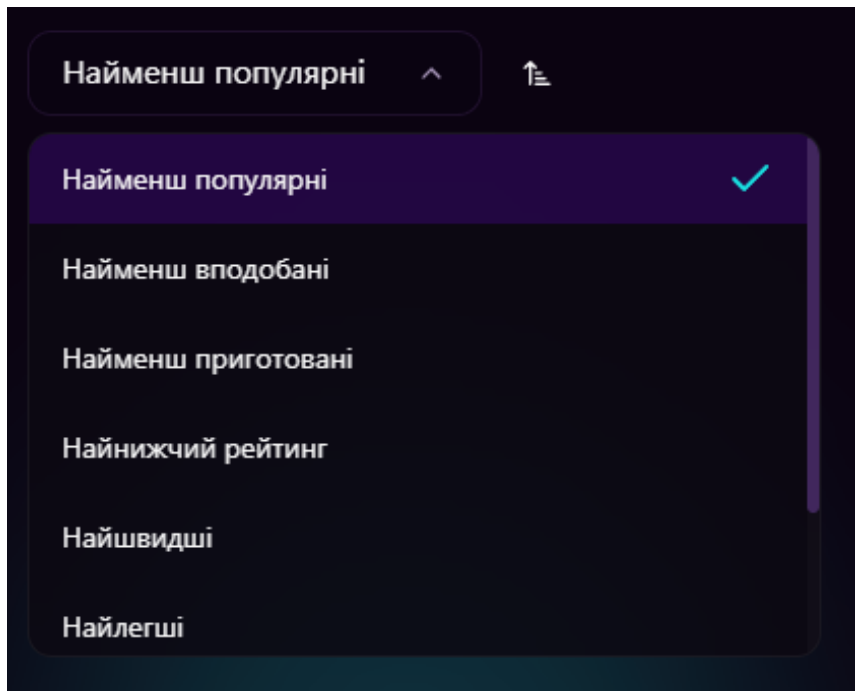


Рис. 3.14. Зростаючий список сортування каталогу

Перейдемо до сторінки входу у вебплатформу (Рис. 3.15). На ній користувачеві пропонується кілька способів автентифікації, проте на поточному етапі повноцінно реалізовано лише вхід за допомогою електронної пошти й паролю та через Google Sign-In, тоді як інші провайдери відображаються як заплановані до підтримки в майбутньому. Процес входу побудовано покроково: спочатку користувач вводить електронну пошту, після чого система визначає доступні для цього облікового запису способи автентифікації.

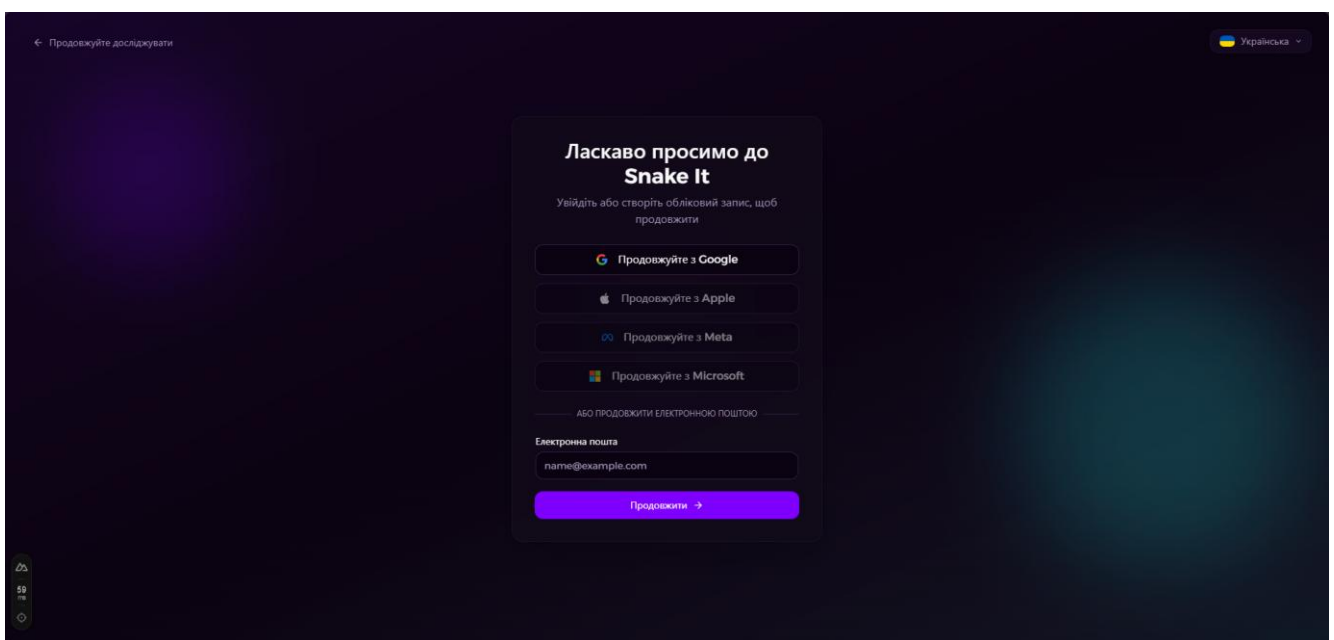


Рис. 3.15. Сторінка автентифікації у вебплатформу

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				73

Автентифікований користувач отримує доступ до сторінки налаштувань, де може керувати параметрами свого облікового запису (Рис. 3.16). У розділі загальних налаштувань, зокрема, доступний вибір мови та регіональних параметрів інтерфейсу, що визначає мову відображення платформи. Налаштування згруповано за вкладками, що спрощує навігацію між загальними параметрами та даними профілю.

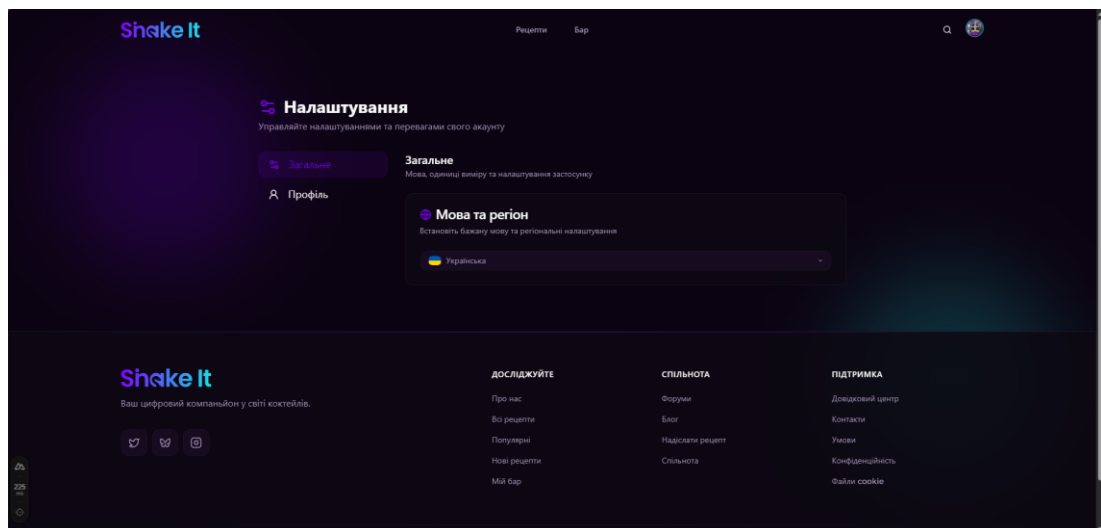


Рис. 3.16. Сторінка налаштування платформи під користувача

Окремий розділ налаштувань дає змогу редагувати публічну інформацію профілю користувача (Рис. 3.17). Тут можна змінити аватар, відображуване ім'я та унікальне ім'я користувача, додати короткий опис у полі біографії та вказати місцезнаходження. Ці дані формують публічну сторінку профілю, яку бачать інші учасники спільноти.

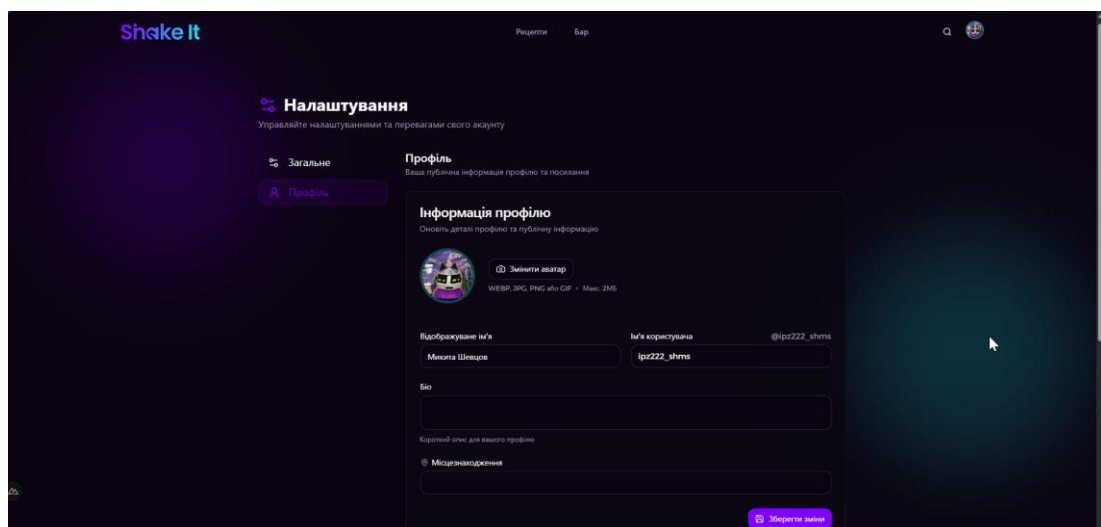


Рис. 3.17. Сторінка налаштування публічного профілю користувача

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				74

Користувач має можливість видалити поточний аватар або завантажити та обробити нове зображення (Рис. 3.18 – Рис. 3.19). Після натискання кнопки зміни аватару відкривається модальне вікно з вибором завантаження нового файлу чи видалення наявного, де також зазначено підтримувані формати та обмеження на розмір файлу (Рис. 3.18). Завантаживши зображення, користувач переходить до вікна обрізання, у якому може перемістити та масштабувати картинку, аби вибрати потрібну ділянку перед збереженням (Рис. 3.19).

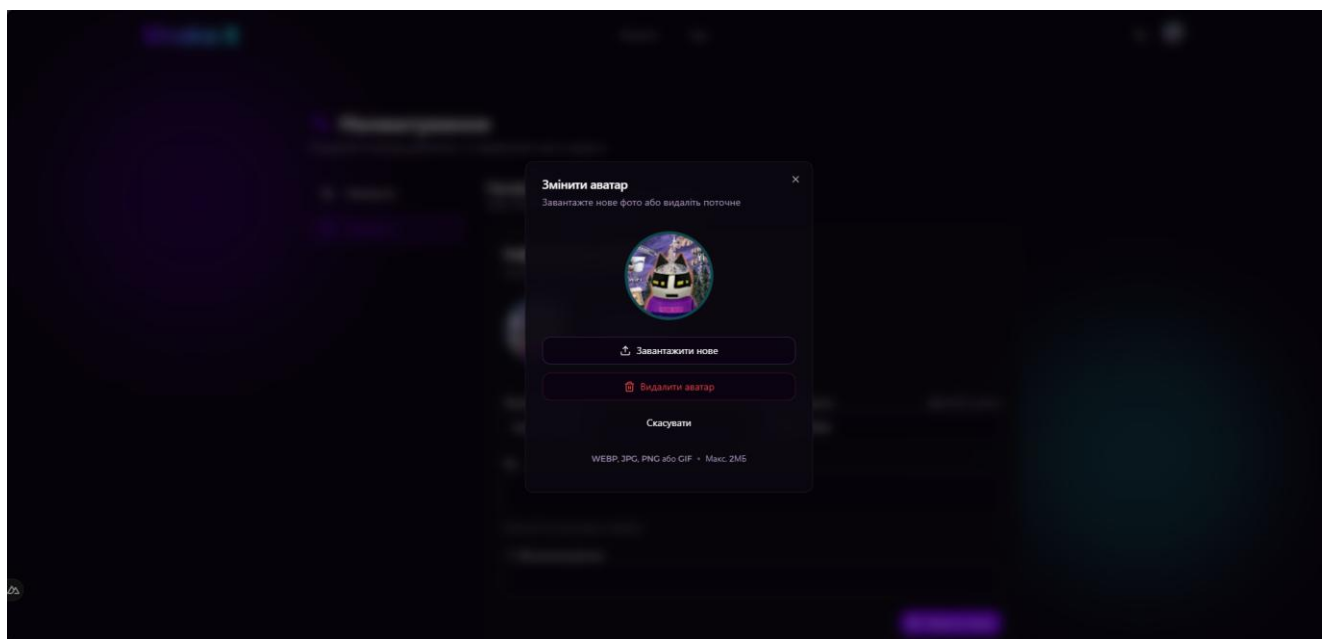


Рис. 3.18. Процес заміни аватару користувача в профілі

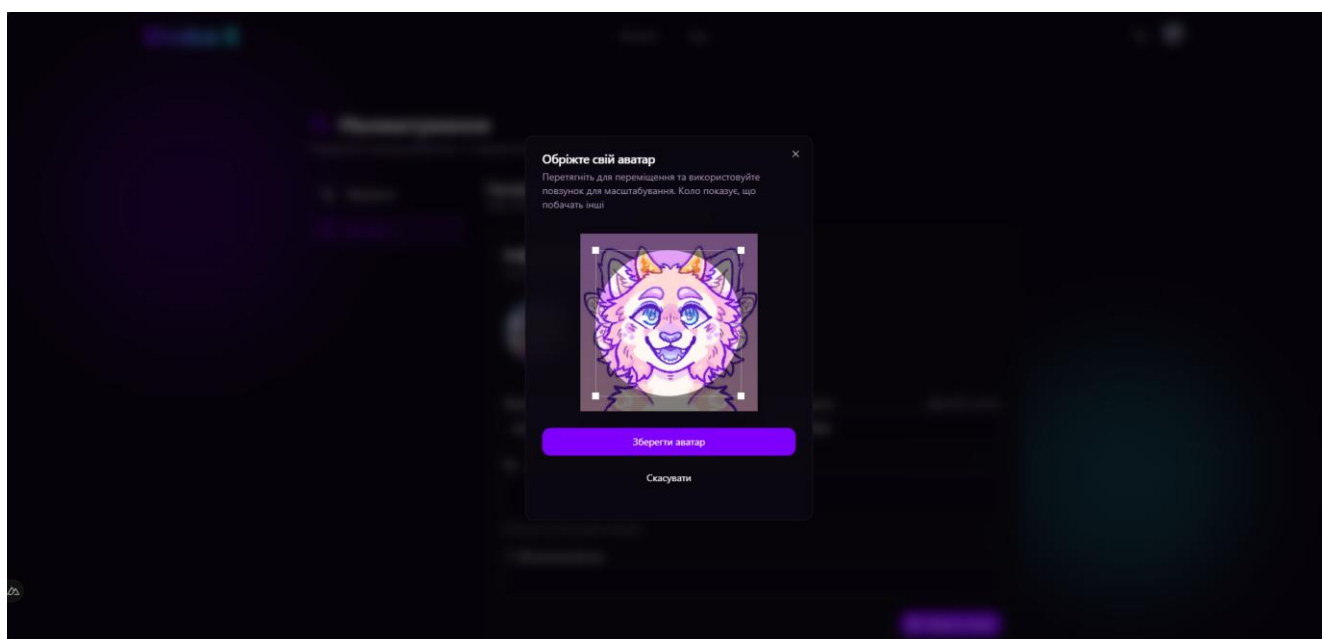


Рис. 3.19. Обрізання та позиціонування аватару перед збереженням

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				75

Результат виконання операцій система повідомляє користувачеві за допомогою спливаючих сповіщень (Рис. 3.20). Успішні дії супроводжуються зеленим повідомленням із відповідним текстом, наприклад про оновлення аватару профілю, тоді як помилки відображаються червоним кольором із поясненням причини. Такі сповіщення з’являються в кутку екрана й автоматично зникають через деякий час, не перешкоджаючи подальшій роботі з вебплатформою.

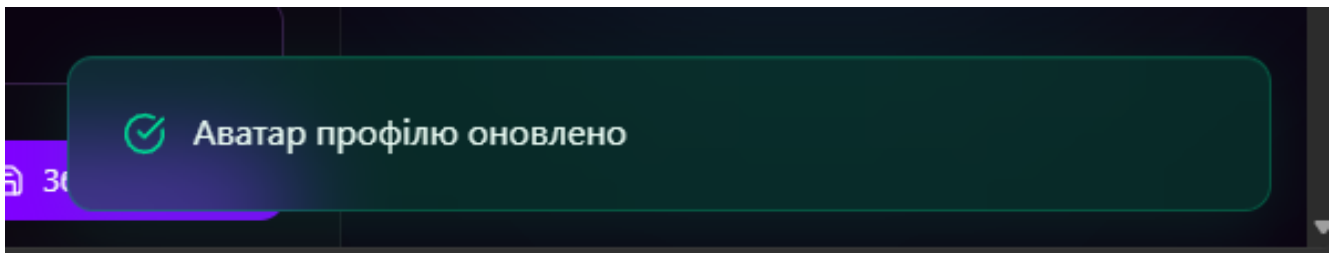


Рис. 3.20. Сповіщення про успішну заміну аватару

У персональному барі користувач керує переліком інгредієнтів, які має у своєму розпорядженні, додаючи нові та видаляючи наявні (Рис. 3.21). Для щойно створеного облікового запису бар порожній, тому замість списку інгредієнтів відображається відповідне повідомлення та запрошення додати перший інгредієнт. Над списком розміщено поле пошуку та елементи сортування, що стають у пригоді в міру наповнення персонального бару.

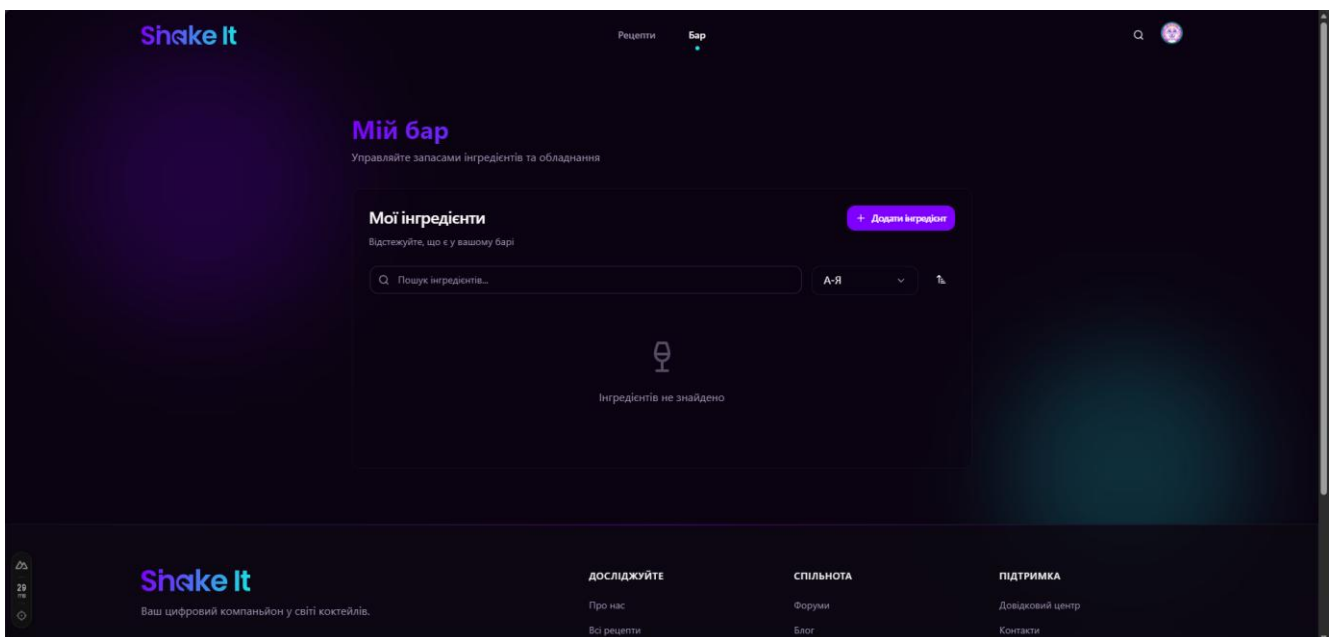


Рис. 3.21. Сторінка персонального бару користувача (порожня)

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				76

Після натискання на кнопку «Додати інгредієнт» відкривається модальне вікно з каталогом доступних інгредієнтів (Рис. 3.22). Кожен інгредієнт подано у вигляді картки із зображенням, назвою, брендом і категорією, а зверху розміщено поле пошуку за назвою, брендом або категорією. Щоб додати інгредієнт до бару, користувачеві достатньо натиснути на відповідну картку.

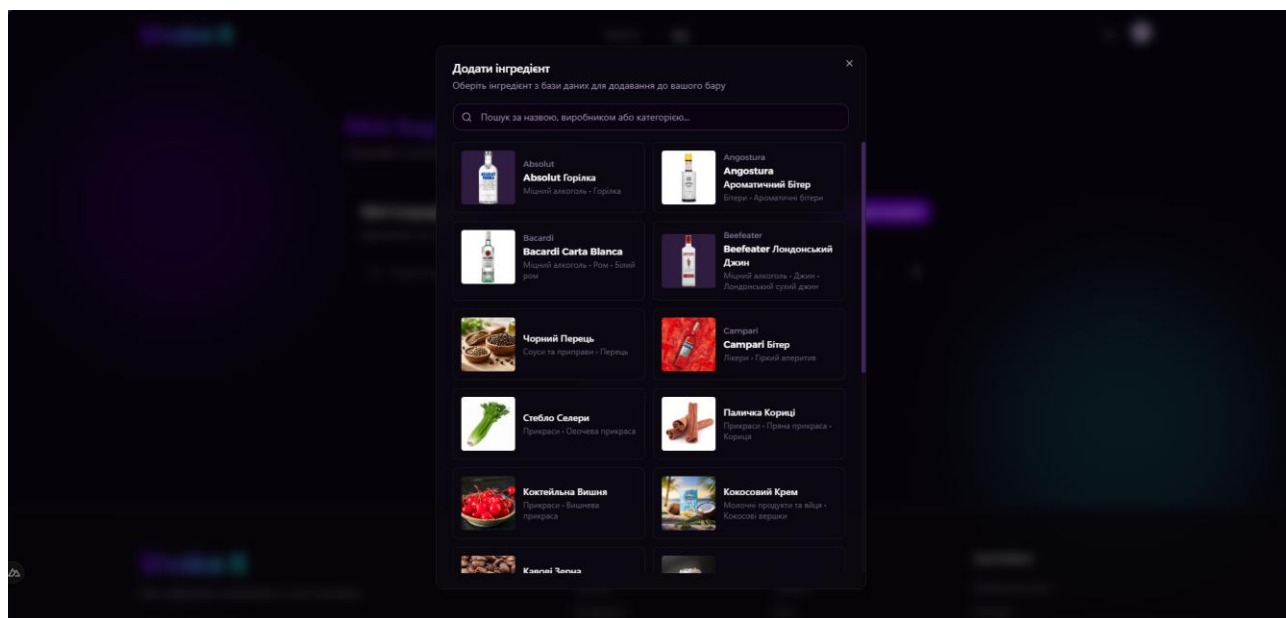


Рис. 3.22. Модальне вікно додавання інгредієнтів в персональний бар

Після натискання на картку з інгредієнтом, вікно не зникає одразу, даючи користувачу можливість додати більше інгредієнтів. Після закриття вікна, користувач бачить власний бар, та може далі керувати наявними інгредієнтами (Рис. 3.23).

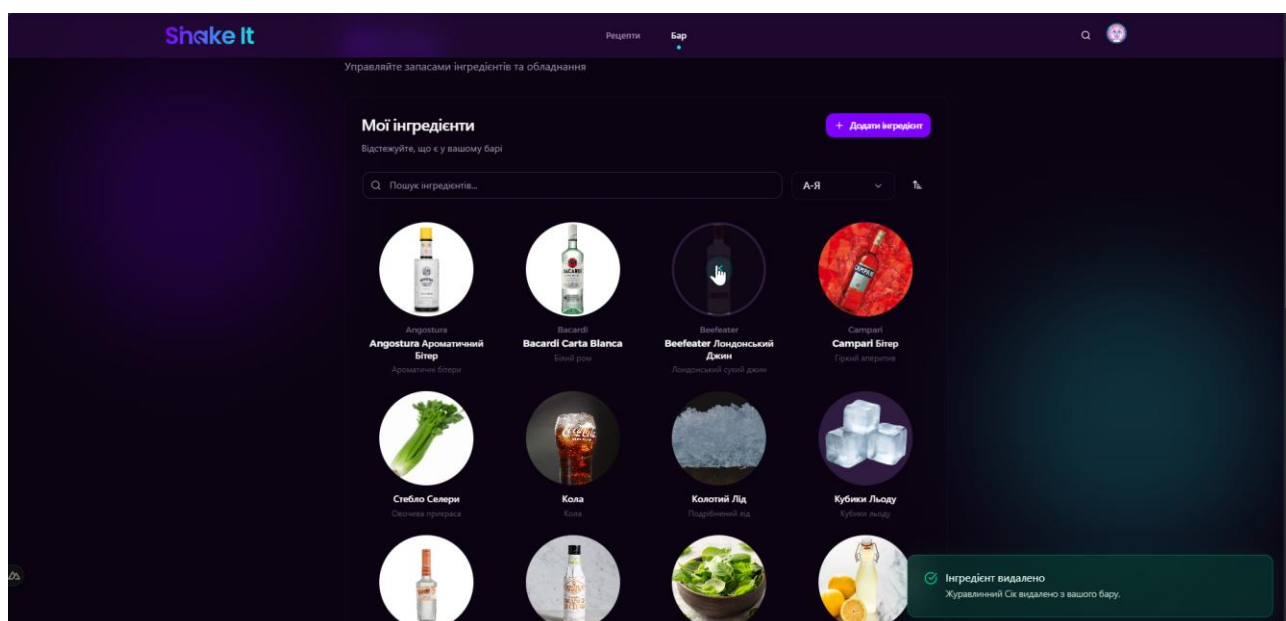


Рис. 3.23. Сторінка персонального бару з можливістю видалення інгредієнта

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				77

Маючи заповнений інгредієнтами персональний бар, автентифікований користувач бачить на картках рецептів співвідношення наявних та необхідних інгредієнтів, наприклад скільки складників із потрібних уже є в його розпорядженні (Рис. 3.24). Це дає змогу з першого погляду оцінити, наскільки той чи інший коктейль доступний для приготування. Окрім того, користувач може скористуватися фільтром «Що я можу приготувати», який залишає в результатах пошуку лише ті рецепти, для яких наявні всі обов’язкові інгредієнти (Рис. 3.25).

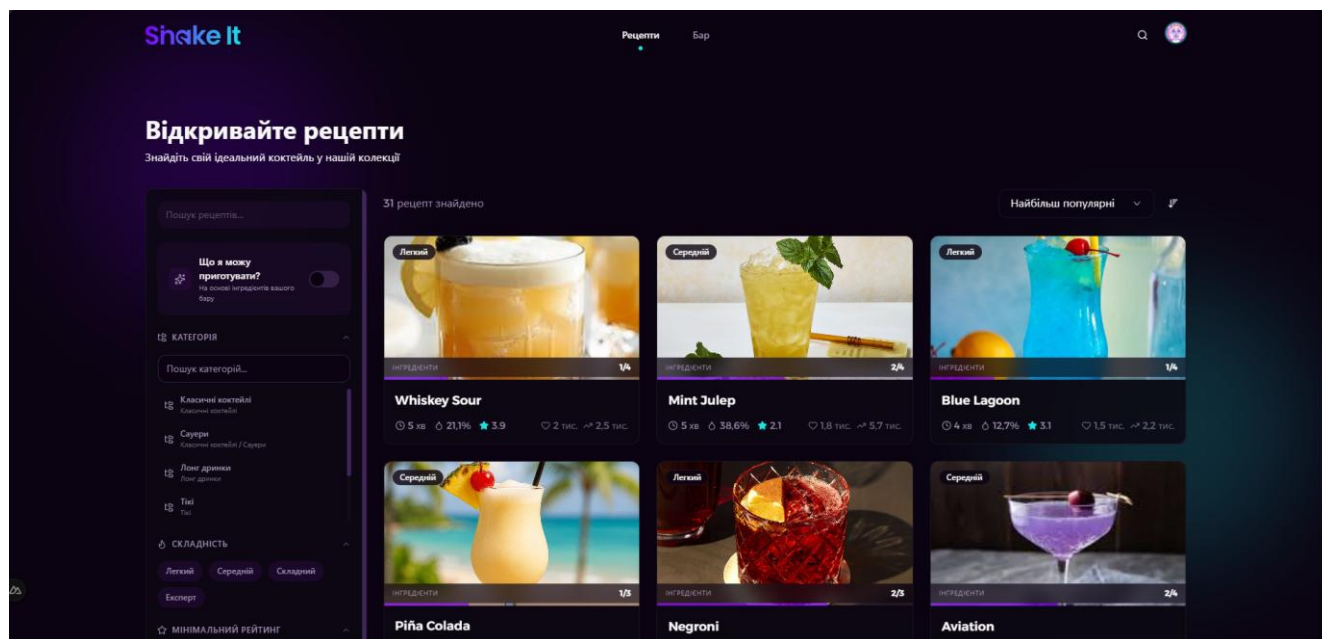


Рис. 3.24. Відображення співвідношення інгредієнтів в каталозі

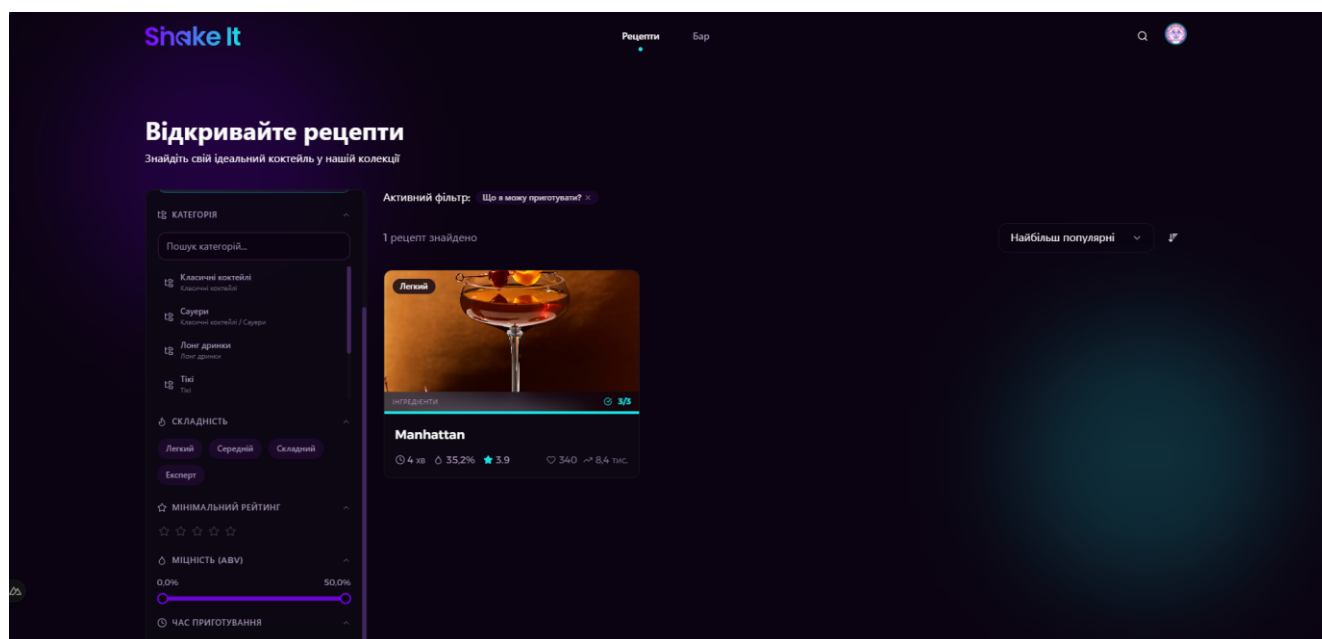


Рис. 3.25. Результат пошуку рецептів, готових до приготування

Відкривши рецепт, який ми можемо приготувати, можна побачити, що у нас наявні 3/4 інгредієнтів, хоча картка відображає 3/3, це пов'язано з тим, що коктейльна вишня помічена як необов'язковий інгредієнт (Рис. 3.26).

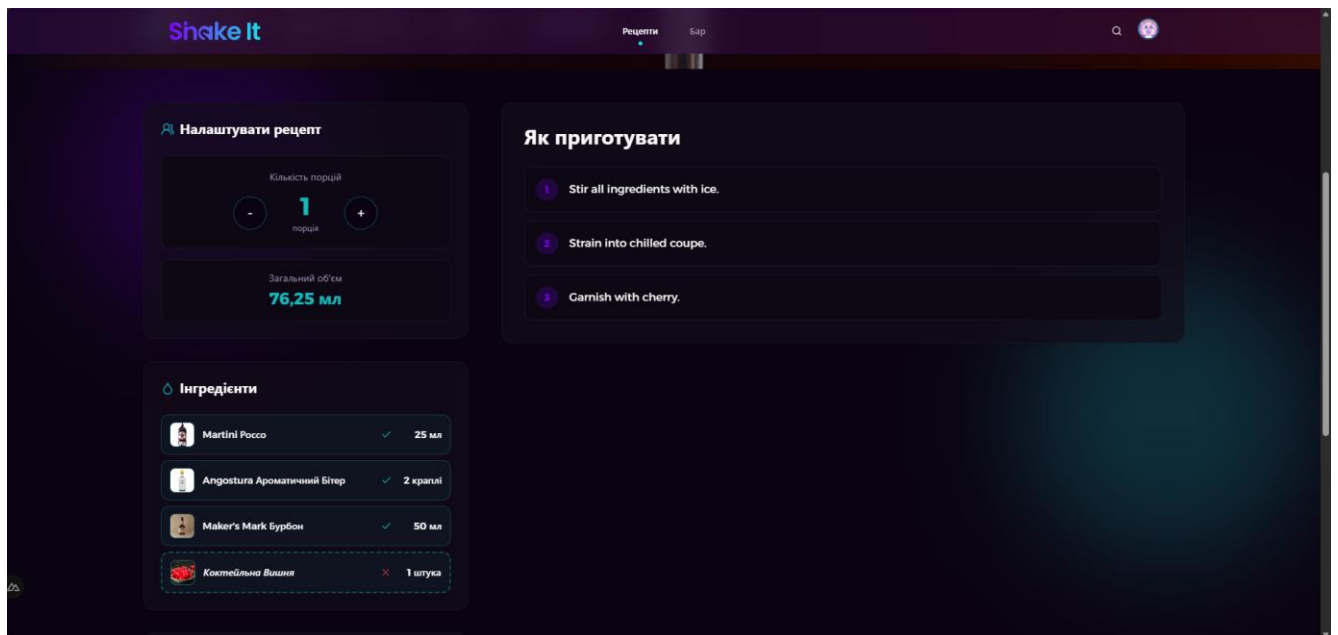


Рис. 3.26. Перегляд деталей рецепту автентифікованим користувачем

Якщо відкрити інший рецепт, можна побачити, що у нас відсутній відповідний джин, але система помічає, що у нас є замітник, інший лондонський джин (Рис. 3.27). Система автоматично помічає, що інгредієнт відсутній у користувача в барі і тому починає пошук схожого інгредієнту за категорією.

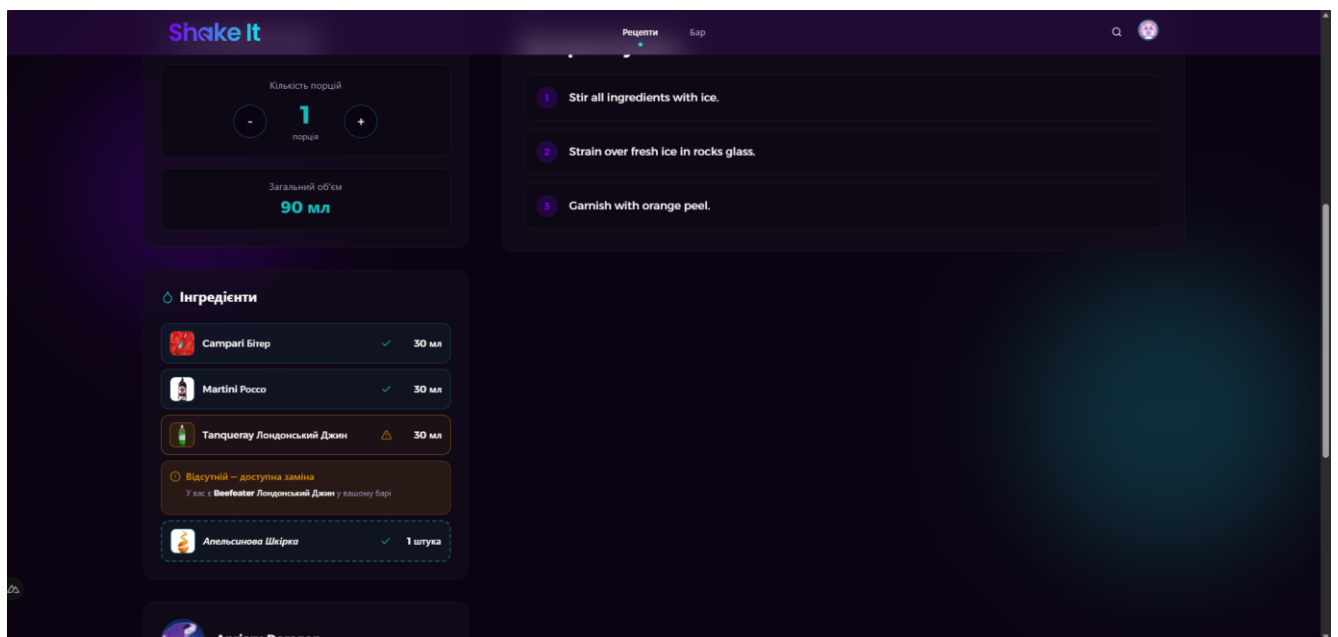


Рис. 3.27. Пропонування заміни відсутнього інгредієнта на альтернативу

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				79

Також, на сторінці рецепту, користувач може змінювати кількість порцій та одразу бачити зміни в кількості необхідних інгредієнтів. Додатково, для зручності приготування великих та складних рецептів, користувач може відмічати кроки для приготування, щоб нічого не пропустити (Рис. 3.28).

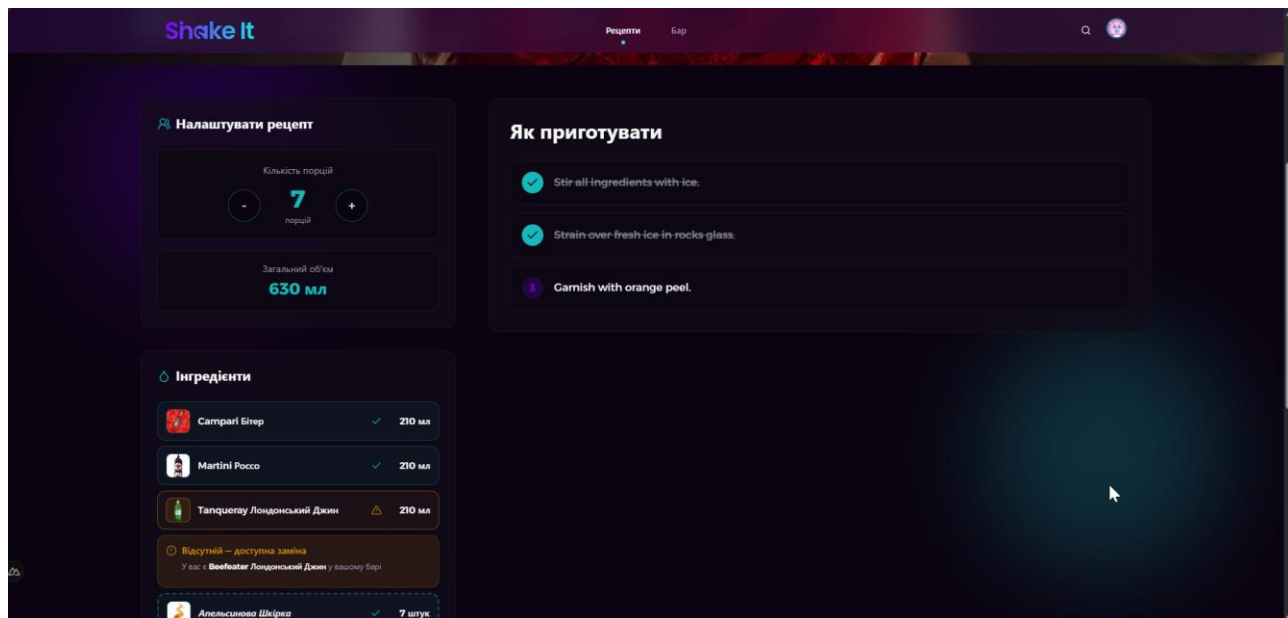


Рис. 3.28. Результат взаємодії з кількістю порцій приготування

Інформація про наявні інгредієнти відображається не лише в каталозі, а й на головній сторінці (Рис. 3.29). Завдяки цьому персоналізований підбір супроводжує користувача з першого ж екрана платформи. Таким чином ключова ідея застосунку послідовно реалізована в усіх місцях, де відображаються рецепти

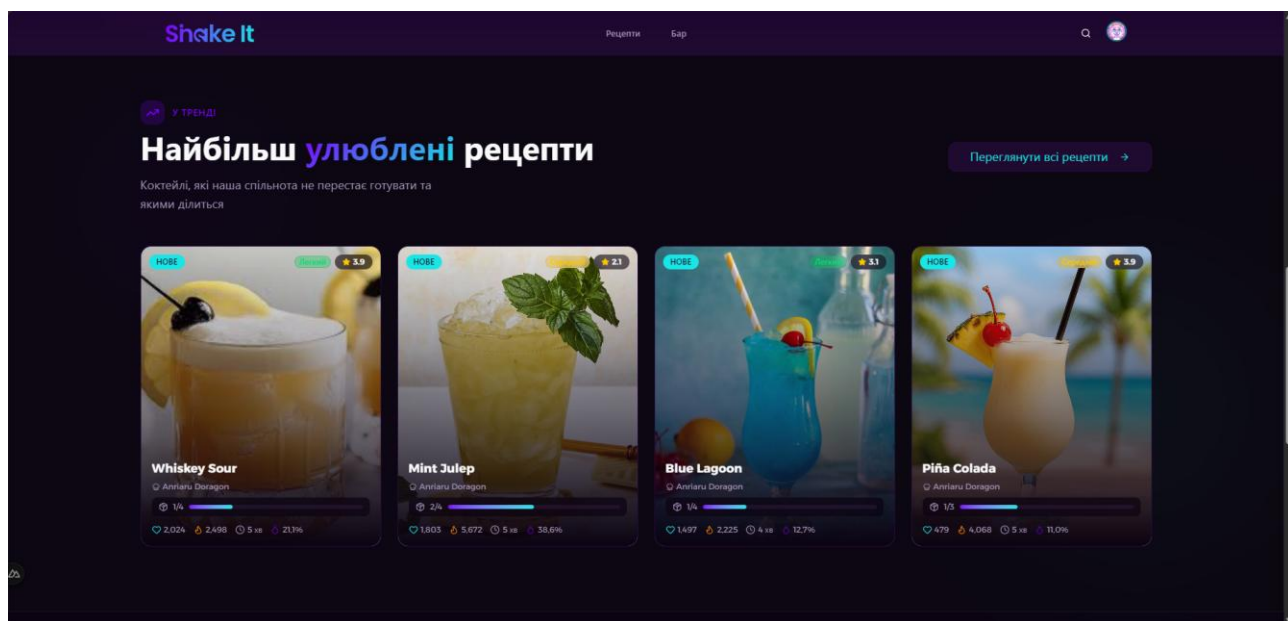


Рис. 3.29. Відображення співвідношення інгредієнтів на головній сторінці

3.3. Тестування роботи вебплатформи

У межах прототипу проведено функціональне тестування – ручну перевірку ключових сценаріїв роботи системи за принципом «чорної скриньки», що підтверджує відповідальність реалізованого функціоналу визначеним вимогам. Розроблення автоматизованих наборів тестів (модульних та інтеграційних) винесено в перспективу розвитку, чому сприяє закладена тестопридатна архітектура.

На сторінці входу проведемо тестування валідації форми реєстрації, послідовно вводячи некоректні значення та перевіряючи, чи система коректно блокує їх та повідомляє користувача про причину помилки.

Першою перевіriamo реакцію на порожнє поле електронної пошти. При спробі продовжити без введення адреси система не пропускає форму далі та виводить повідомлення про те, що електронна пошта є обов'язковою (Рис. 3.30).

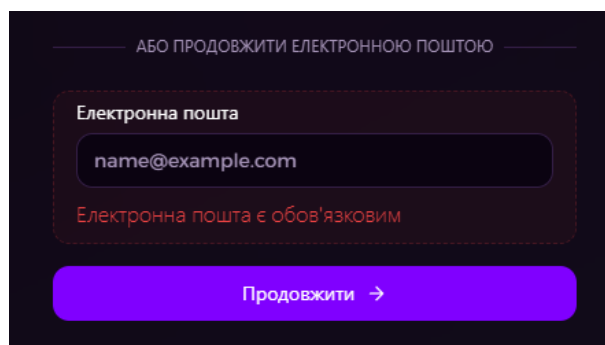
The image shows a web form titled 'АБО ПРОДОВЖИТИ ЕЛЕКТРОННОЮ ПОШТОЮ' (OR CONTINUE WITH ELECTRONIC MAIL). It contains a label 'Електронна пошта' (Email) above a text input field. The input field is empty, and below it, a red error message reads 'Електронна пошта є обов'язковим' (Email is mandatory). At the bottom of the form is a blue button labeled 'Продовжити' (Continue) with a right-pointing arrow.

Рис. 3.30. Перевірка порожнього поля електронної пошти

Далі перевіriamo валідацію формату адреси, ввівши свідомо некоректне значення «test», яке не є електронною поштою. Система розпізнає невідповідність формату та повідомляє, що адреса не є дійсною, не дозволяючи перейти до наступного кроку (Рис. 3.31).

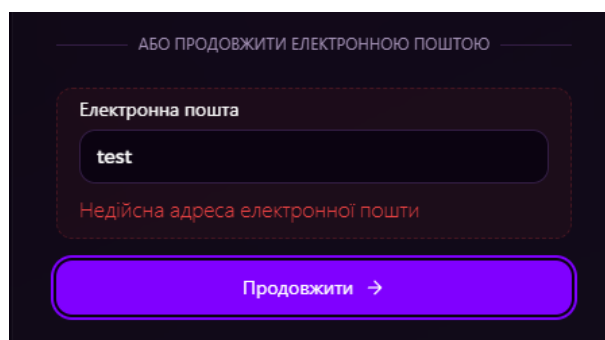
The image shows the same web form as in Figure 3.30. The text input field now contains the value 'test'. Below the field, a red error message reads 'Недійсна адреса електронної пошти' (Invalid email address). The blue 'Продовжити' (Continue) button remains at the bottom.

Рис. 3.31. Перевірка некоректної електронної пошти

Після переходу до кроку реєстрації перевіримо поведінку форми за порожніх обов'язкових полів. Система одночасно підсвічує всі незаповнені поля та виводить відповідні повідомлення про обов'язковість пароля та його підтвердження, не виконуючи відправлення форми (Рис. 3.32).

Рис. 3.32. Перевірка порожніх значень форми реєстрації

Окремо перевіримо правило, що забороняє починати ім'я користувача зі спеціального символу. При введенні значення «_snake» система повідомляє, що ім'я має починатися з літери та може містити лише літери, цифри й окремі спеціальні символи, що відповідає регулярному виразу, описаному в Лістингу 2.1 (Рис. 3.33).

Рис. 3.33. Перевірка початку ім'я зі спец-символу

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				82

Наступним перевіримо заборону двох спеціальних символів підряд. При введенні імені з послідовними роздільниками система так само відхиляє значення та виводить те саме правило формування імені, не дозволяючи зареєструватися (Рис. 3.34).

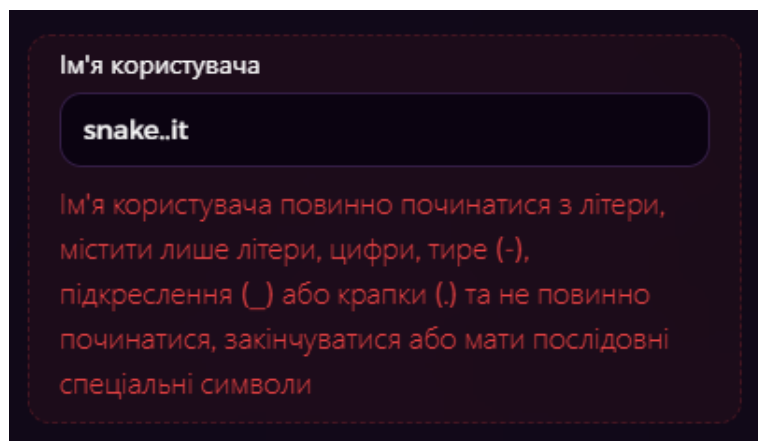


Рис. 3.34. Перевірка дублювання спец-символів в імені

Перевіримо також контроль унікальності імені користувача. При введенні вже наявного у системі імені «anriarudoragon» виконується асинхронна перевірка через API, за результатами якої система повідомляє, що таке ім'я вже зайняте (Рис. 3.35).

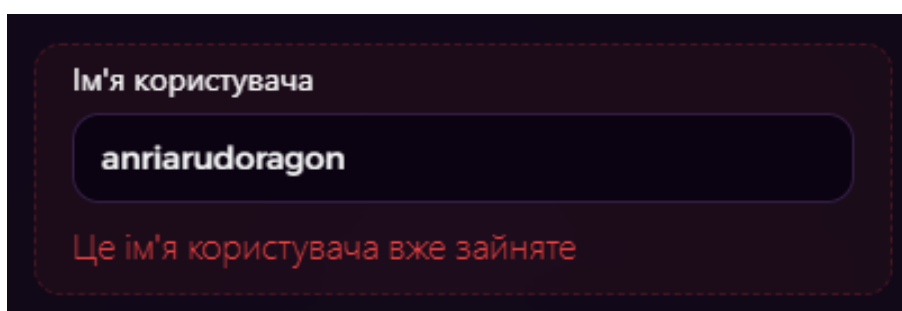


Рис. 3.35. Перевірка вже зайнятого імені

Далі перевіримо вимогу до мінімальної довжини пароля. При введенні закороткого значення система повідомляє що мінімальна кількість символів становить вісім, блокуючи подальше проходження форми (Рис. 3.36).

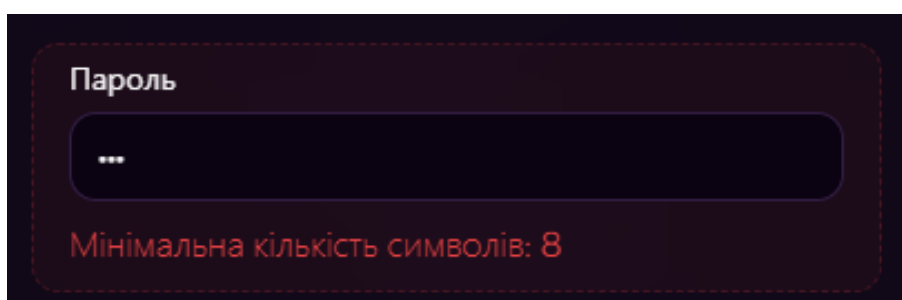


Рис. 3.36. Перевірка короткого паролю

Окремо перевіримо узгодженість пароля та його підтвердження. Якщо у полях «Пароль» і «Підтвердьте пароль» введено різні значення, система виявляє розбіжність та повідомляє що паролі повинні збігатися (Рис. 3.37).

The screenshot shows a dark-themed form with two input fields. The first field is labeled 'Пароль' (Password) and contains several dots. The second field is labeled 'Підтвердьте пароль' (Confirm password) and also contains dots. Below the second field, a red error message reads 'Паролі повинні збігатися' (Passwords must match). The entire form is enclosed in a dashed red border.

Рис. 3.37. Перевірка збігу паролів

Нарешті, заповнимо форму повністю коректними даними. Усі перевірки проходять успішно, повідомлення про помилки зникають, а кнопка створення акаунта стає активною, що підтверджує коректну роботу валідації за нормального сценарію (Рис. 3.38).

The screenshot shows a dark-themed form titled 'Створіть свій акаунт' (Create your account). Below the title is the email address 'lnikshev@gmail.com'. There are three input fields: 'Ім'я користувача' (Username) with the value 'lnikshev', 'Пароль' (Password) with dots, and 'Підтвердьте пароль' (Confirm password) with dots. At the bottom is a large blue button labeled 'Створити акаунт' (Create account).

Рис. 3.38. Коректно заповнена форма реєстрації

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				84

Одразу після успішної реєстрації система виводить сповіщення з проханням перевірити електронну пошту для підтвердження облікового запису та отримання доступу до всіх функцій (Рис. 3.39). Водночас користувача автоматично авторизовано в системі, що підтверджується появою його профілю у верхньому меню навігації з пунктами переходу до налаштувань і виходу (Рис. 3.40). Отже, для початку роботи з платформою повторний вхід після реєстрації не потрібен.

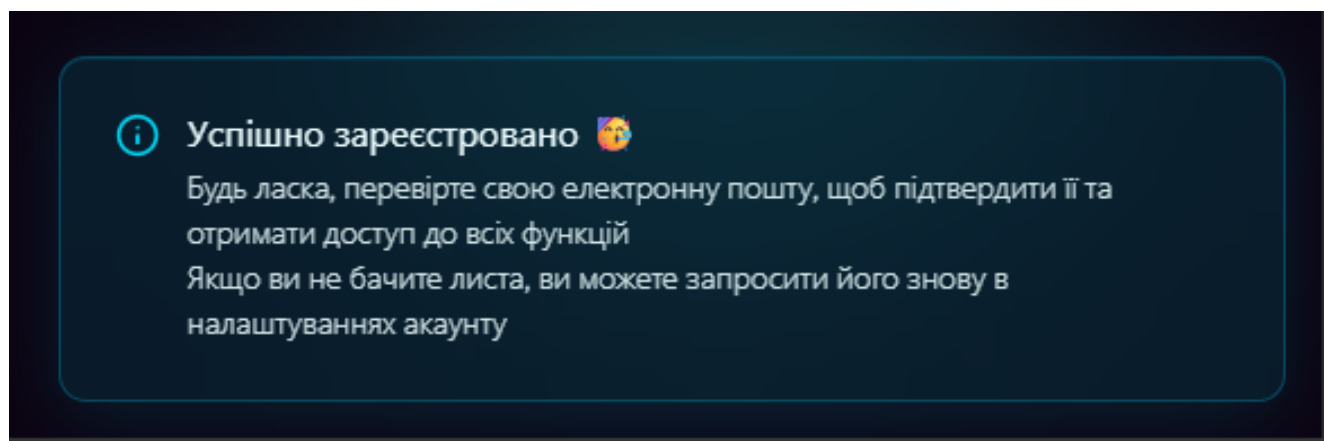


Рис. 3.39. Повідомлення про успішну реєстрацію

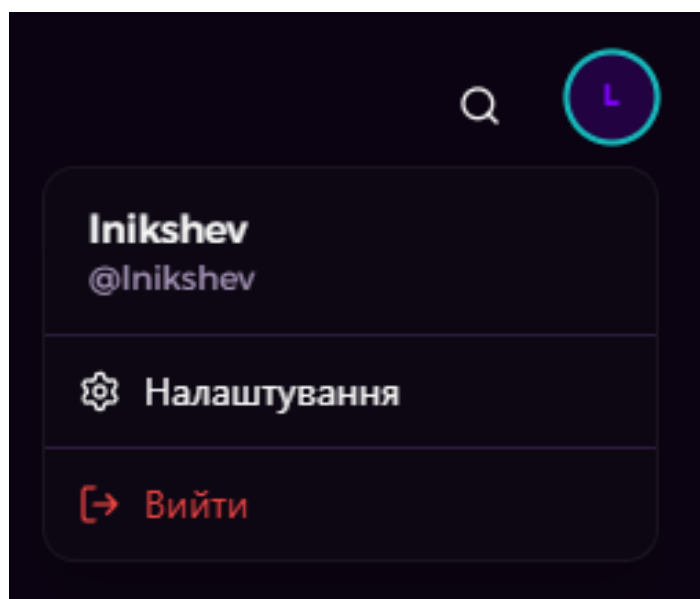


Рис. 3.40. Пункт навігації з користувацьким меню

Невдовзі після реєстрації, на вказану електронну адресу надходить лист для підтвердження пошти (Рис. 3.41). Лист містить персональне звертання до користувача та кнопку підтвердження, а також альтернативне текстове посилання на випадок, якщо кнопка не спрацює. Окремо зазначено, що термін дії посилання обмежений сімома днями, після чого його потрібно буде запросити повторно.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				85

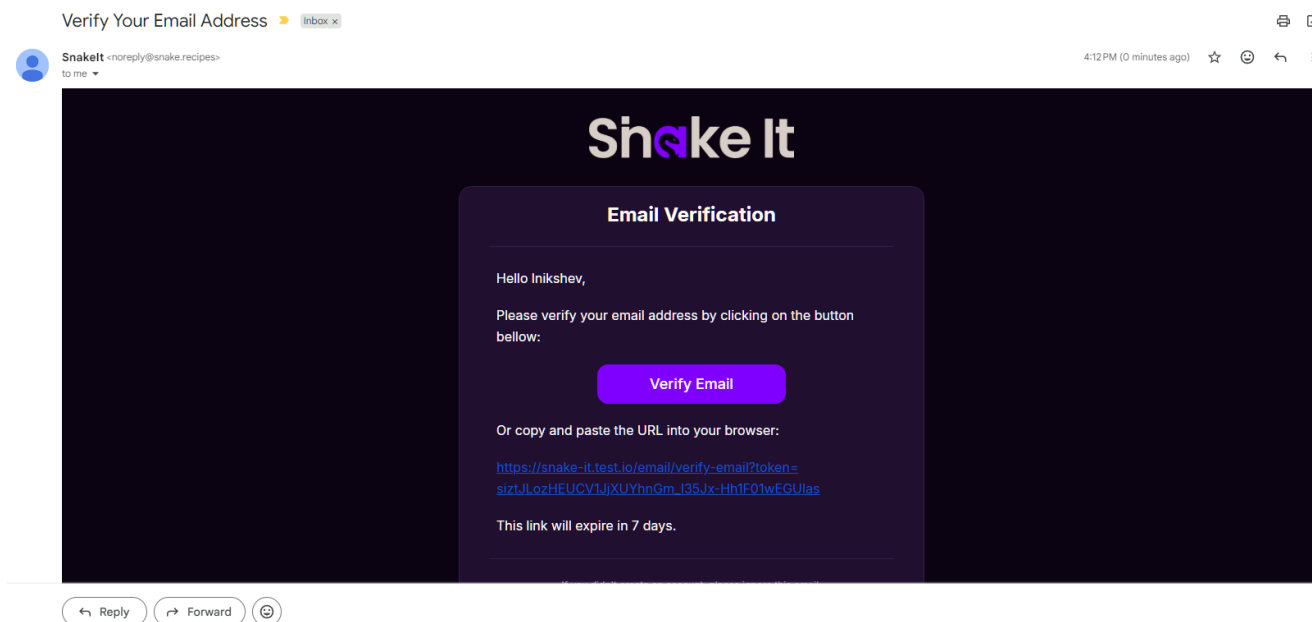


Рис. 3.41. Лист підтвердження електронної пошти

Після переходу за посиланням із листа, застосунок перенаправляє користувача на сторінку, яка повідомляє про успішне підтвердження електронної пошти (Рис. 3.42). На сторінці відображається відповідне повідомлення та кнопка переходу до основного функціоналу платформи. Із цього моменту обліковий запис вважається повністю активованим.

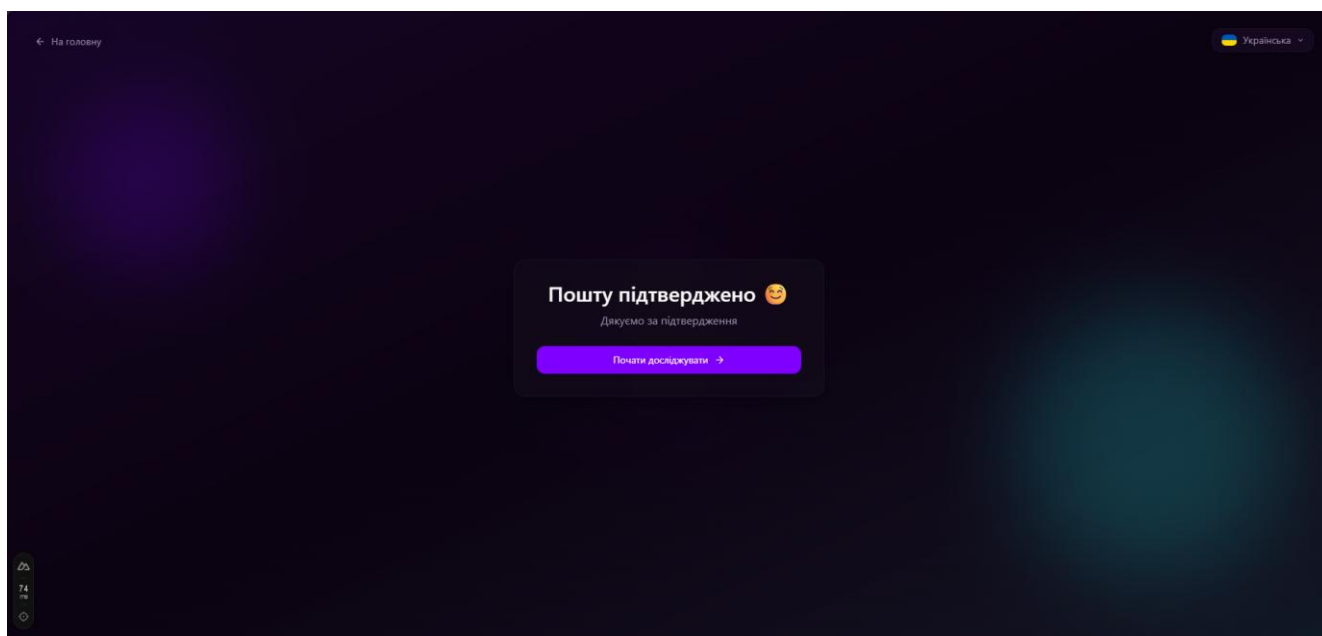


Рис. 3.42. Результат сторінки підтвердження електронної пошти

Якщо спробувати оновити сторінку або повторно перейти за тим самим посиланням, отримаємо помилку, про те що воно вже не дійсне, ідентично

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				86

відбудеться, якщо пройде відповідний термін в тиждень і токен стане недійсним (Рис. 3.43).

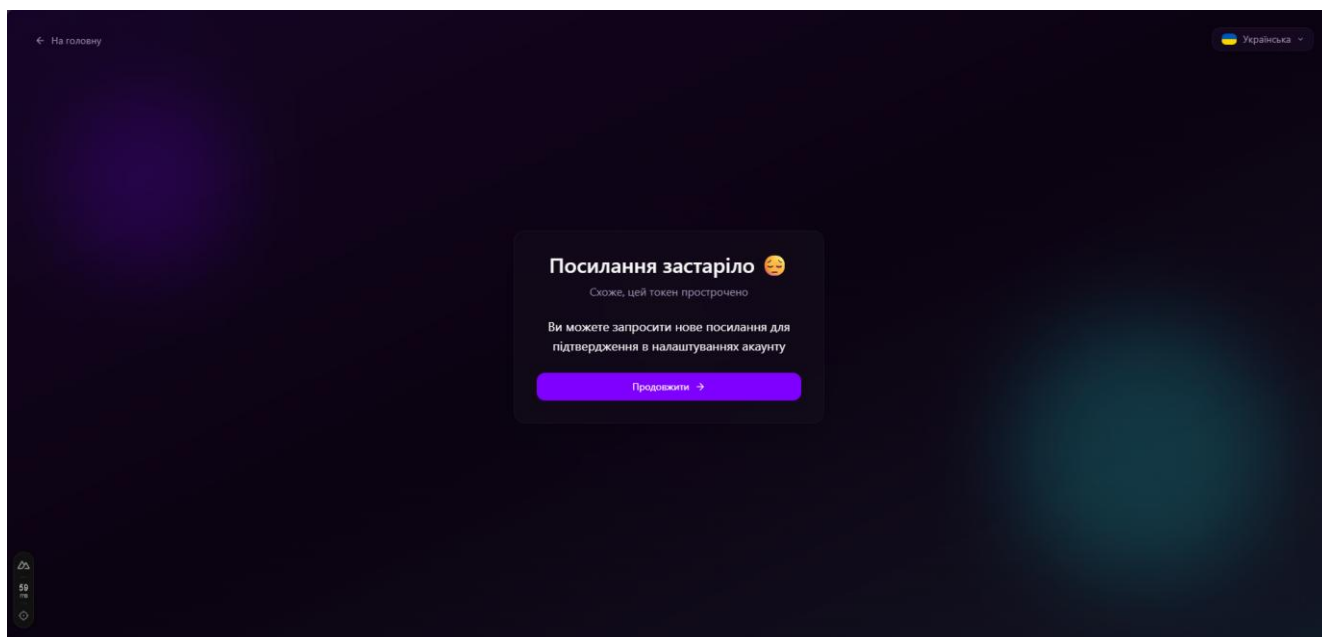


Рис. 3.43. Використання недійсного токена підтвердження

Коректність підтвердження наразі можна перевірити безпосередньо на рівні бази даних (Рис. 3.44). Запит до таблиці користувачів за відповідною електронною адресою показує, що поле статусу підтвердження пошти набуло значення «TRUE», а сам запис містить очікуване ім'я користувача.

×

where

Email

▼

equals

▼

lnikshev@gmail.com

count(*)

+ Add filter

Clear filters

☐	Id uuid ↕	Email varchar(256) ↕	EmailVerified boolean ↕	Username varchar(32) ↕
☐	5ca76e10-14ba-4d89-898...	lnikshev@gmail.com	TRUE	lnikshev

Рис. 3.44. Підтверджена пошта користувача в записі бази даних

Оскільки ми зареєструвалися за допомогою паролю, тепер при спробі увійти за допомогою Google з тією ж самою поштою, ми повинні отримати помилку (Рис. 3.45), оскільки задля безпеки, автоматичне з'єднання акаунтів було заблоковане, користувач повинен власноруч у налаштуваннях додати інший спосіб входу.

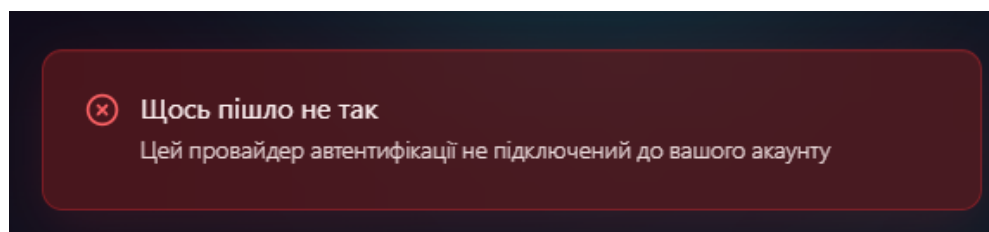


Рис. 3.45. Сповіщення про помилку входу через «Google» в акаунт з паролем

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				87

Після введення пошти, на наступному кроці запитують про пароль, оскільки цей користувач має лише вхід з паролем (Рис. 3.46). Якщо спробуємо іншого користувача, нам запропонує використати Google, оскільки пароль там не встановлено (Рис. 3.47).

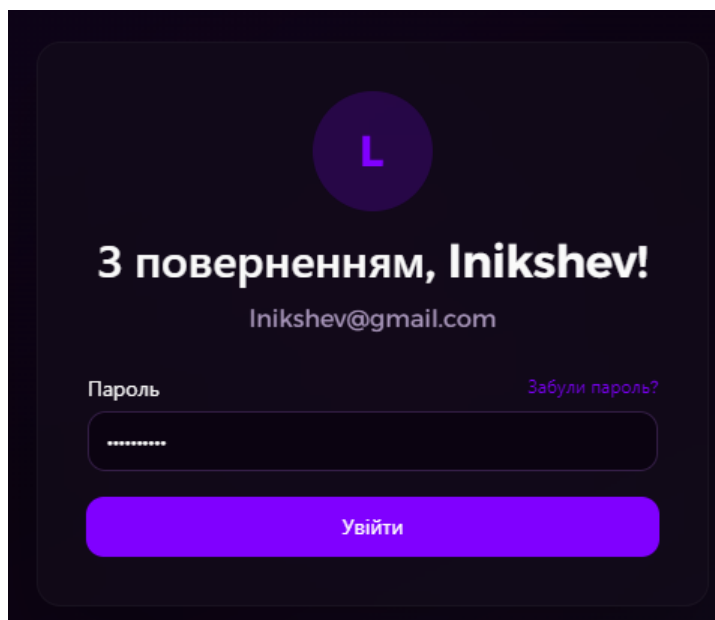


Рис. 3.46. Привітання для користувача з паролем

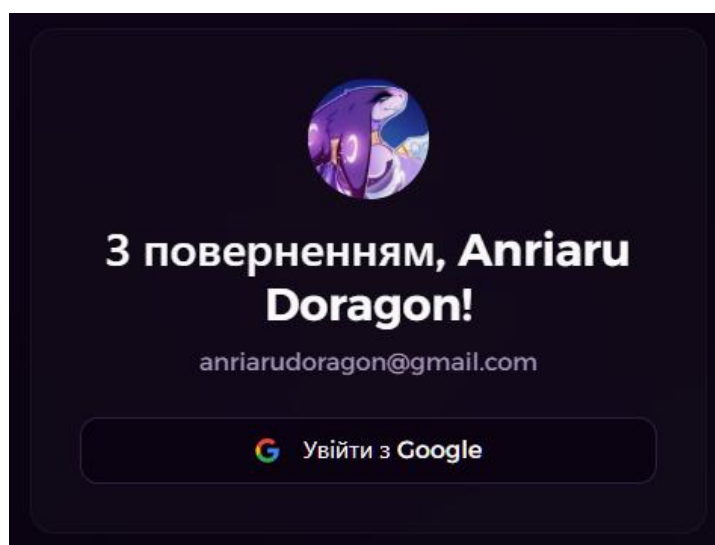


Рис. 3.47. Привітання для користувача з іншим провайдером

Перевіримо також сценарій відновлення пароля для облікового запису, який має вхід за паролем (Рис. 3.48). На сторінці скидання пароля користувачеві потрібно вказати електронну адресу облікового запису, після чого система ініціює процедуру відновлення. Поле адреси при цьому може бути попередньо заповнене, якщо користувач перейшов сюди зі сторінки входу.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				88

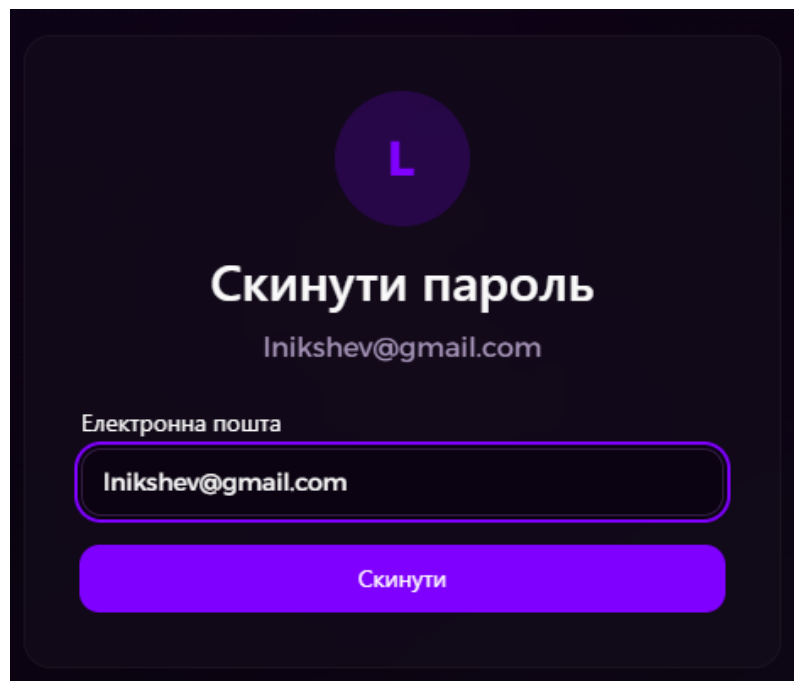


Рис. 3.48. Форма відновлення паролю

Після скидання паролю, система повідомляє нас що було відправлено електронний лист для відновлення на вказану адресу (Рис. 3.49). На цьому ж етапі, користувач може відправити повторний лист, але не одразу, а лише через деякий час, оскільки першому листу ще треба час на відправку, а новий лист знищує токен попереднього. Такий підхід також зменшує кількість запитів до серверу.

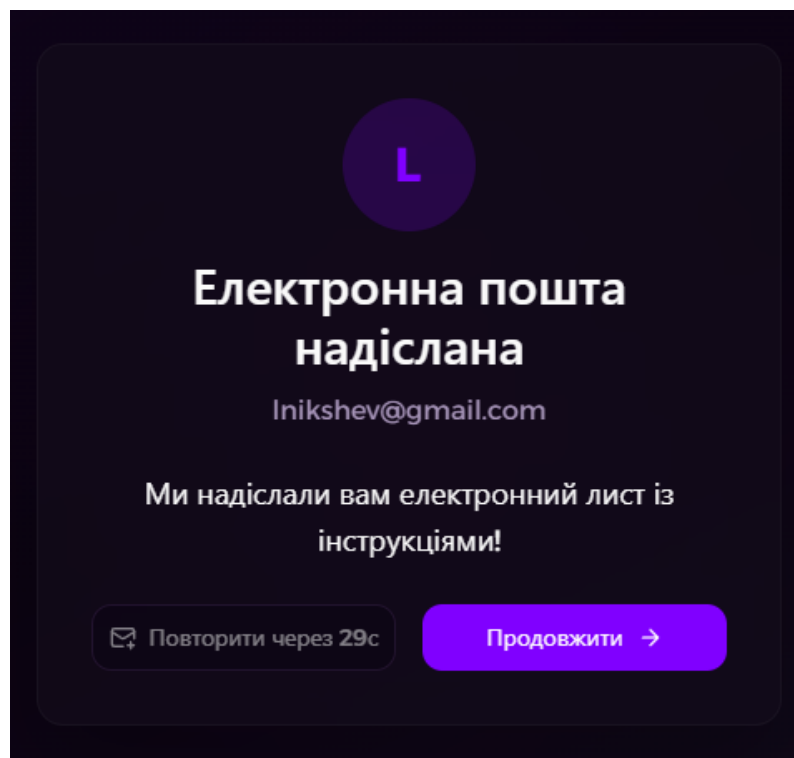


Рис. 3.49. Результат скидання паролю

На вказану електронну адресу надходить лист із посиланням для відновлення пароля (Рис. 3.50). За структурою він подібний до листа підтвердження пошти й містить кнопку скидання пароля та дублююче текстове посилання. Водночас термін дії цього посилання коротший і становить шість годин, що обумовлено вищими вимогами до безпеки операції зміни пароля. Після переходу за посиланням, перевіряється токен, та при успішності перевірки, користувач має можливість змінити пароль на новий (Рис. 3.51).

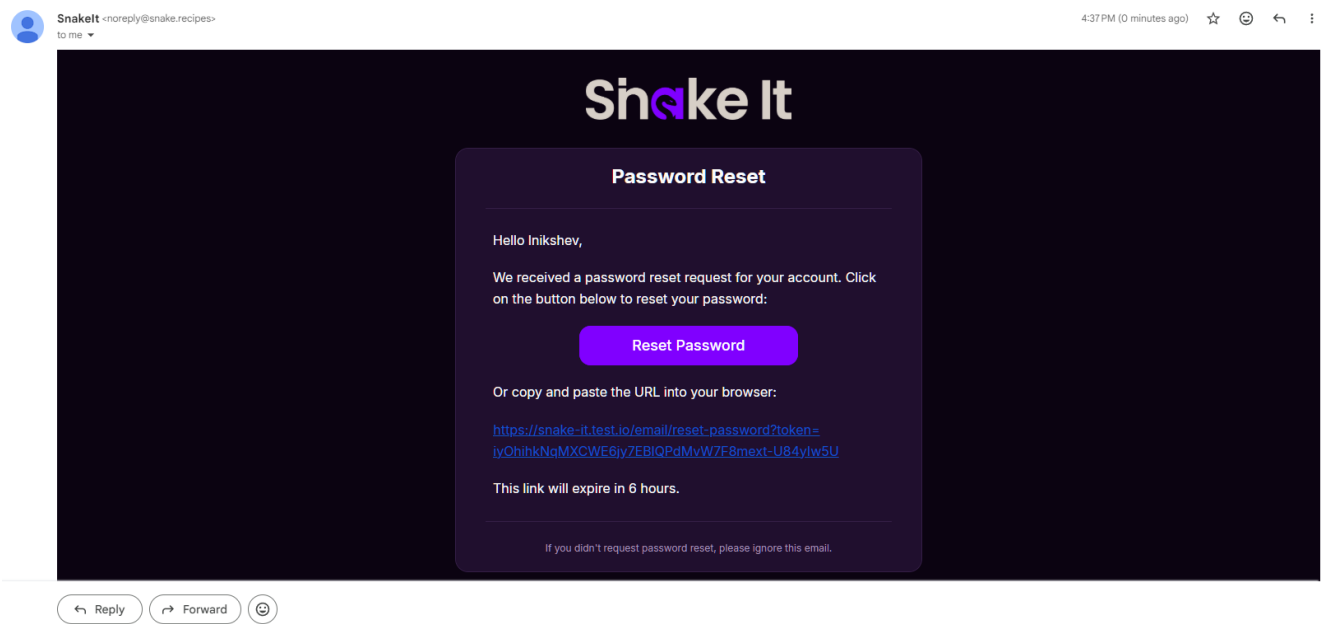


Рис. 3.50. Лист відновлення паролю

The image shows a form for resetting a password. It has a dark background. At the top, it says 'Скинути пароль' in large white letters. Below that is the email address 'l*****v@gmail.com'. There are two input fields for the new password, both labeled 'Новий пароль' in white. The first input field has a blue border and a blue focus ring. The second input field has a blue border and a blue focus ring. Below the input fields is a large blue button labeled 'Оновити пароль' in white. Below the button, it says 'Спливає через 6 годин' in small white text.

Рис. 3.51. Форма встановлення нового паролю

Після збереження нового пароля система виводить сповіщення про успішне завершення операції (Рис. 3.52). У повідомленні також зазначено, що відтепер користувач може входити в систему з новим паролем. Таким чином повний цикл відновлення доступу, від запиту до встановлення нового паролю, відпрацьовує коректно.

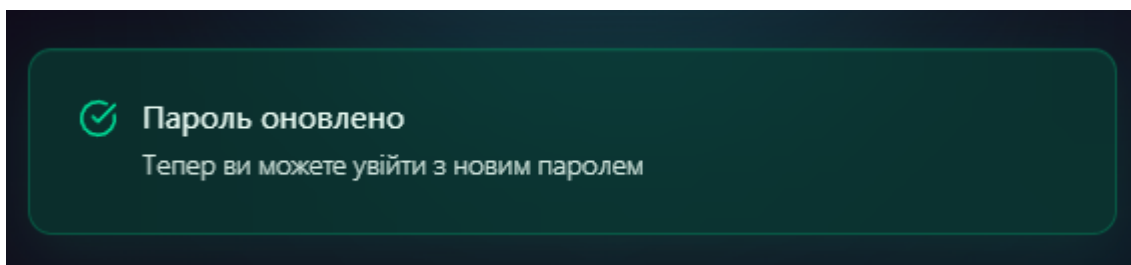


Рис. 3.52. Повідомлення про успішну зміну паролю

При спробі повторно відкрити посилання для відновлення паролю, побачимо помилку про те, що токен більше не дійсний, хоча його час ще не вийшов, його було використано та відкликано (Рис. 3.53).

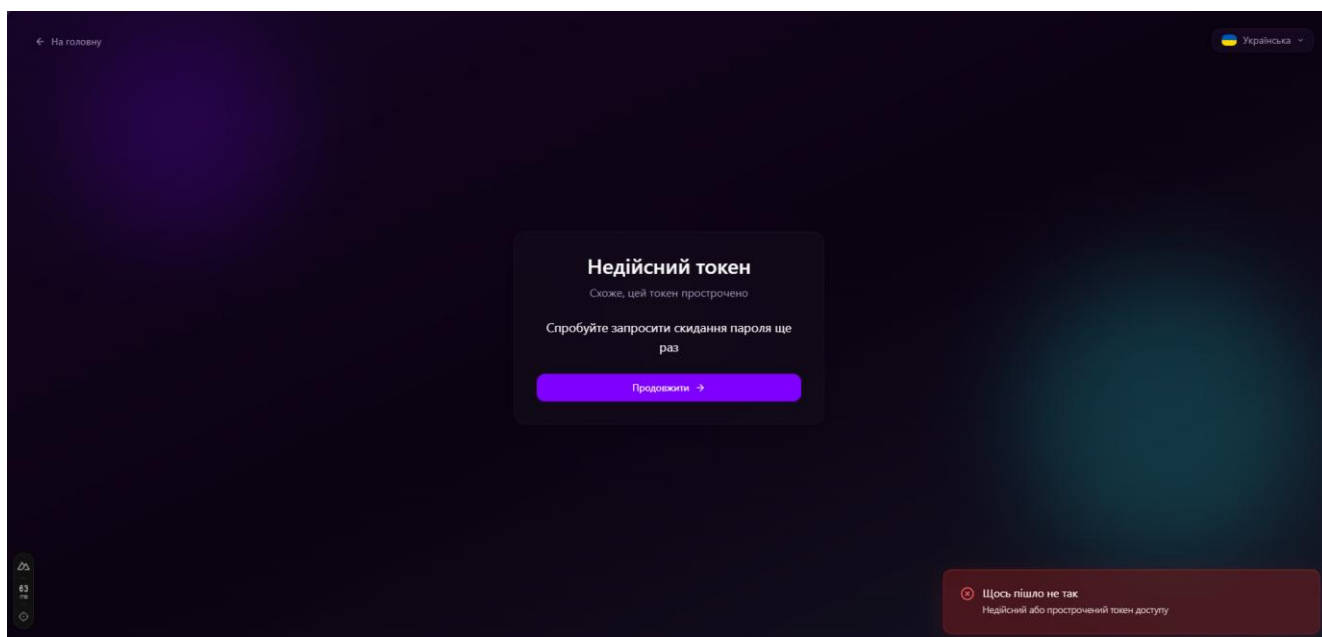


Рис. 3.53. Помилка токenu для відновлення паролю

Перевіримо, що інгредієнти для рецептів коректно співвідносяться до бару користувача, для прикладу візьмемо ось такий рецепт, в якому у нас не вистачає двох інгредієнтів (Рис. 3.54) та додамо ці інгредієнти в бар користувача. Після додавання, при пошуку з фільтром «Що я можу приготувати?», ми повинні побачити цей рецепт в списку, а також, при відкритті, всі інгредієнти мають бути зеленими.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				91

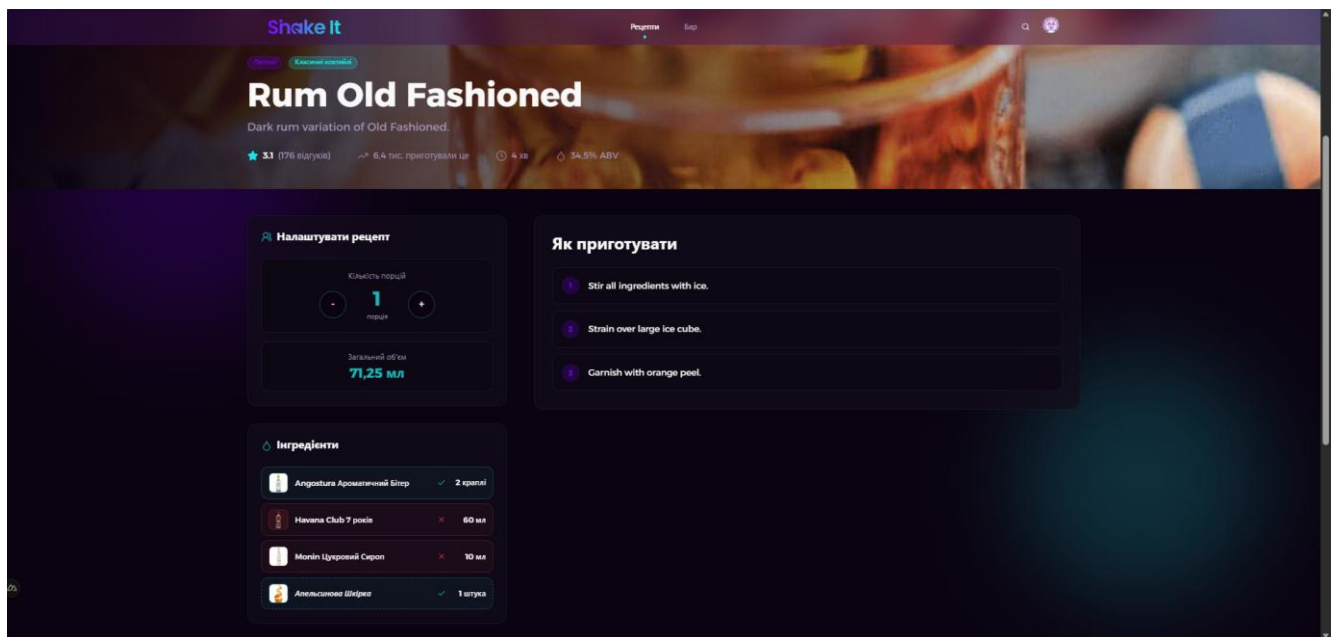


Рис. 3.54. Сторінка перевірного рецепту

Після додавання відсутніх інгредієнтів до бару та повторного пошуку з увімкненим фільтром «Що я можу приготувати?» перевірений рецепт очікувано з'являється у списку результатів з відповідним співвідношенням інгредієнтів рецепту та персонального бару користувача 3/3 (Рис. 3.55). Поряд із ним у видачу потрапляють й інші рецепти, для яких наявних інгредієнтів достатньо. Це підтверджує, що зміни у складі персонального бару одразу враховуються механізмом підбору доступних рецептів.

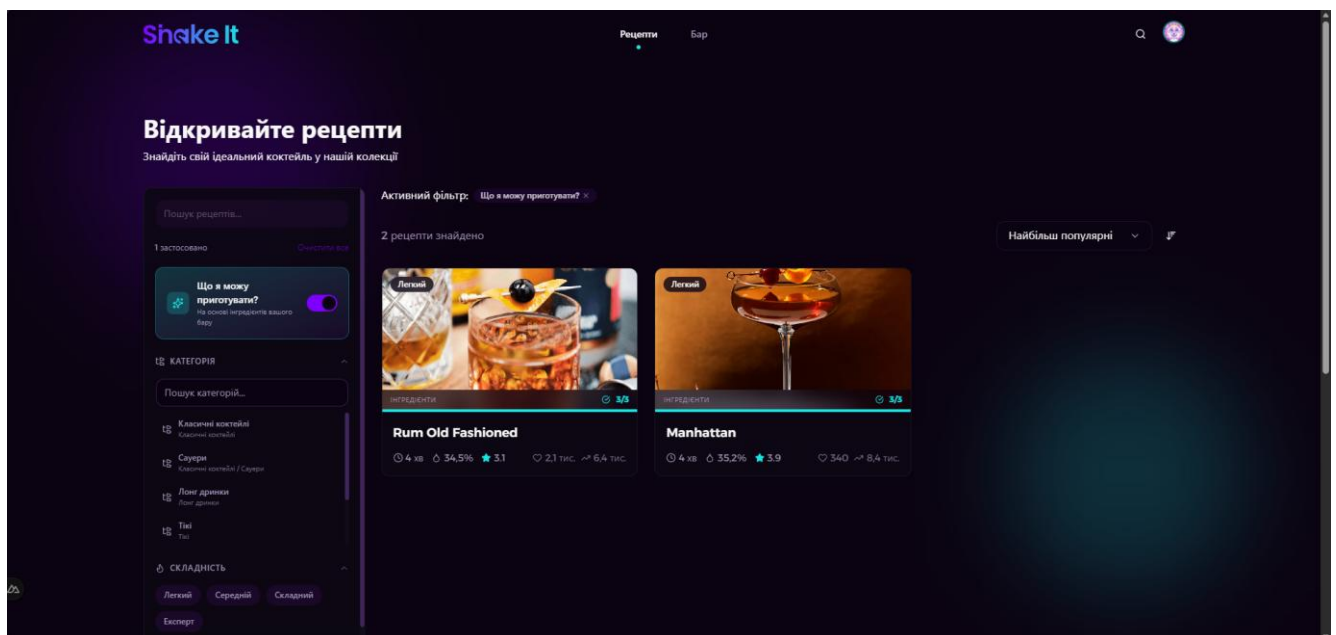


Рис. 3.55. Результат пошуку в каталозі після зміни бару

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				92

Відкривши сам рецепт, можна переконаватися, що тепер усі його інгредієнти позначені зеленим кольором із відмітками наявності (Рис. 3.56). Кожен складник супроводжується зазначенням потрібної кількості, а статус наявності розраховується відносно поточного вмісту бару користувача. Це остаточно підтверджує коректність співвіднесення інгредієнтів рецептів із персональним баром.

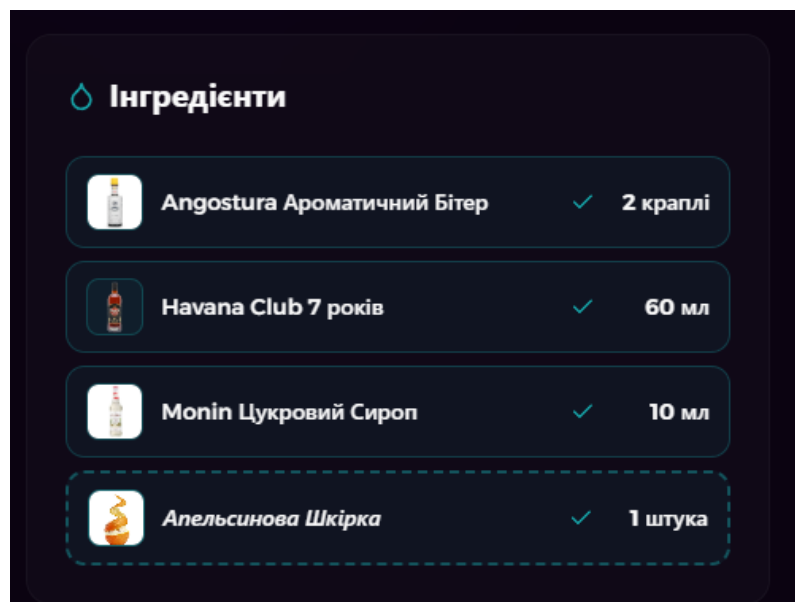


Рис. 3.56. Оновлений список інгредієнтів перевірного рецепту

Висновки до третього розділу

У третьому розділі описано практичні аспекти розгортання, експлуатації та тестування розробленої вебплатформи з міксології. Якщо попередній розділ був присвячений проєктуванню та реалізації системи, то цей розділ підтверджує, що створений прототип є не лише теоретично завершеним, а й працездатним на практиці, він може бути розгорнутий у реальному середовищі та коректно виконує задеклароване ядро функціоналу.

Визначено порядок встановлення та налаштування системи. Обґрунтовано, що розгортання доцільно розпочинати із зовнішніх сервісів, від яких залежить як REST API, так і SPA. Наголошено на необхідності мати доменне ім'я з TLS-сертифікатом, оскільки без захищеного домену та коректно налаштованої політики CORS неможлива робота Google Sign-In, CDN та автентифікації з refresh-токеном у HttpOnly cookie. Описано можливі підходи до розгортання бази даних, а також

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				93

зазначено необов'язковість Redis та поштового сервісу для базової працездатності системи. Детально розглянуто конфігурацію CDN Bunny.net – налаштування Pull Zone, Storage Zone та Stream із заборonoю прямого доступу до файлів і ввімкнення автентифікації за токеном, і наведено повний перелік параметрів, необхідних серверу для роботи. Окремо описано створення Google Client ID зі значеннями дозволеного домену.

Продемонстровано структуру користувацького інтерфейсу платформи. Показано основні сторінки та сценарії взаємодії: головну сторінку, перегляд рецептів, каталог рецептів, сторінку входу, керування налаштуваннями й профілем користувача, а також роботу з персональним баром. Підтверджено працездатність ключової ідеї платформи: для автентифікованого користувача картки відображають кількість наявних інгредієнтів, фільтр «Що я можу приготувати?» коректно підбирає доступні рецепти, враховується статус необов'язкових інгредієнтів, а для відсутніх інгредієнтів система пропонує замітники тієї ж категорії.

Проведено функціональне тестування основних функціональних можливостей системи. Перевірено валідацію форми реєстрації на широкому наборі некоректних значень. Протестовано повний цикл роботи з електронною поштою. Підтверджено коректність політики безпеки щодо з'єднання облікових записів. Окремо протестовано співвіднесення інгредієнтів бару з вимогами рецептів: після додавання відсутніх інгредієнтів, рецепт коректно з'явився у результатах пошуку, а всі його інгредієнти отримали статус наявних.

Таким чином, у третьому розділі підтверджено працездатність розробленого прототипу вебплатформи, відповідність реалізованого функціоналу визначеним раніше вимогам та коректність роботи механізмів автентифікації, управління персональним баром і пошуку рецептів за наявними інгредієнтами. Результати тестування свідчать, що система поводить себе передбачувано як за коректних, так і за помилкових сценаріїв, належно інформує користувача про помилки та надійно реалізує ключову ідею персоналізованого підбору коктейлів, що дає підстави вважати поставлені перед прототипом завдання виконаними.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				94

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано прототип вебплатформи з міксології «Snake It», що забезпечує автентифікацію користувачів, управління персональним баром інгредієнтів та підбір коктейльних рецептів на основі їх наявного складу. Поставлену мету досягнуто, а всі визначені у вступі завдання виконано.

У першому розділі проведено аналіз предметної області міксології та огляд чотирьох існуючих програмних рішень – «Difford's Guide», «Tipsy Bartender», «Cocktail Flow» та «Jigger». За результатами порівняльного аналізу встановлено, що жодне з розглянутих рішень не поєднує персоналізований підбір рецептів на основі реального вмісту бару користувача з повноцінною відкритою платформою для спільноти. Виявлено спільні недоліки аналогів: відсутність вільної публікації власних рецептів, абстрактну деталізацію інгредієнтів без зазначення бренду чи типу та відсутність повноцінної системи оцінювання. На основі цього аналізу сформовано вимоги до платформи та обґрунтовано вибір технологічного стеку: клієнт-серверної архітектури з REST API, серверної частини на ASP.NET Core 10 з принципами чистої архітектури та паттерном CQRS, клієнтської частини на Nuxt.js 4 із серверним рендерингом, СУБД PostgreSQL з ORM Entity Framework Core, а також допоміжної інфраструктури – CDN Bunny.net для медіа-файлів, а також поштового сервісу з MJML-шаблонами.

У другому розділі здійснено проєктування та реалізацію ключових компонентів системи. Визначено повний перелік функціональних та нефункціональних вимог, структурованих за трьома модулями: автентифікації, персонального бару та рецептів і їх пошуку. Спроектовано архітектуру серверної частини з поділом на чотири незалежні шари та розроблено модель даних, представлену у вигляді ERD-діаграми з п'ятьма логічними пакетами сутностей, із гнучкою системою локалізації та рекурсивною структурою категорій. Реалізовано комплексну систему безпеки: автентифікацію на основі пари JWT access та refresh-токенів, вхід через Google за допомогою authorization code flow, двофакторну автентифікацію на основі TOTP,

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				95

гранульовану авторизацію за дозволами та захищений доступ до медіа-файлів через тимчасово підписані URL-адреси. Окрему увагу приділено захисту від типових загроз: атак перебору, енумерацій облікових записів, Race Condition та викрадення токенів. Детально описано реалізацію центральної ідеї платформи – фільтра «Що можна приготувати» на основі вмісту персонального бару та механізму пошуку інгредієнтів-замінників. На стороні клієнта розроблено SPA з централізованим захистом маршрутів, автоматичним оновленням токена, власним механізмом валідації форм, нескінченною прокруткою каталогу та глобальною системою модальних вікон.

У третьому розділі описано практичні аспекти розгортання, експлуатації та тестування платформи. Наведено діаграму розгортання та детально розглянуто порядок встановлення системи, зокрема налаштування зовнішніх сервісів, конфігурацію CDN із заборonoю прямого доступу до файлів та створення Google Client ID. Продемонстровано структуру користувацького інтерфейсу на основні сценарії взаємодії з платформою. Проведено тестування основних функціональних можливостей: валідації форми реєстрації на наборі некоректних значень, повного циклу роботи з електронною поштою (підтвердження пошти, відновлення паролю з відкликанням використаних токенів), політики безпеки щодо з'єднання облікових записів, а також коректність співвіднесення інгредієнтів бару з вимогами рецептів. Результати тестування підтвердили відповідність реалізованого функціоналу визначеним вимогам.

Практичне значення одержаних результатів полягає у тому, що розроблений прототип може використовуватися як самостійний цифровий сервіс для домашніх барменів та ентузіастів міксології. Закладена архітектурна основа забезпечує можливість подальшого розвитку платформи до повноцінного соціального простору для спільноти міксологів із розширенням функціоналу: публікацією користувацьких рецептів, системою оцінювання та коментування, обліком залишків інгредієнтів, III-генерацію рецептів, рекомендаціями придбання відсутніх інгредієнтів та випуском мобільного додатку.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				96

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мартін Р. Чиста архітектура. Київ : Фабула, 2019. 368 с.
2. Шевцов М. ASP.NET core API template implementing clean architecture and CQRS pattern. *GitHub*. URL: <https://github.com/AnriaruDragon/ASP-Clean-Architecture> (дата звернення: 27.03.2026).
3. Advanced token authentication - bunny.net documentation. *Developer Hub - bunny.net Documentation*. URL: <https://docs.bunny.net/cdn/security/token-authentication/advanced#url-structure> (дата звернення: 05.05.2026).
4. All popular icon sets, one framework. *Iconify Design*. URL: <https://iconify.design/> (дата звернення: 10.04.2026).
5. BunnyWay d.o.o. Hop on the fastest content delivery network!. *Bunny CDN*. URL: <https://bunny.net/cdn/> (дата звернення: 15.04.2026).
6. Cocktail flow. *Cocktail Flow*. URL: <https://cocktailflow.com/> (дата звернення: 12.03.2026).
7. Cocktail Mag. What is mixology, and a mixologist? Definition. *Cocktail Mag*. URL: <https://www.cocktailmag.fr/en/magazine/mixology> (дата звернення: 06.03.2026).
8. Difford's Guide - the home of discerning drinkers. *Difford's Guide*. URL: <https://www.diffordsguide.com/> (дата звернення: 12.03.2026).
9. Entity Framework documentation hub. *Microsoft Learn*. URL: <https://learn.microsoft.com/en-us/ef/> (дата звернення: 27.03.2026).
10. Evans E. Domain-Driven design: tackling complexity in the heart of software. Boston : Addison-Wesley, 2003. 560 p.
11. Fowler M. Patterns of enterprise application architecture. Boston : Addison-Wesley, 2002. 533 p.
12. Google. Use code model. *Google for Developers*. URL: <https://developers.google.com/identity/oauth2/web/guides/use-code-model> (дата звернення: 24.03.2026).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				97

- 13.Hardt D., Microsoft. RFC 6749: the oauth 2.0 authorization framework. *Internet Engineering Task Force (IETF)*. URL: <https://www.rfc-editor.org/info/rfc6749/> (дата звернення: 19.03.2026).
- 14.Intersection Observer API - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API (дата звернення: 07.05.2026).
- 15.Introduction | pinia. *Pinia | The intuitive store for Vue.js*. URL: <https://pinia.vuejs.org/introduction.html> (дата звернення: 12.05.2026).
- 16.Lock A. ASP.NET Core in Action. Shelter Island : Manning Publications, 2023. Vol. 3 : Third edition. 968 p.
- 17.Mailjet. The responsive email framework. *MJML*. URL: <https://mjml.io/> (дата звернення: 24.04.2026).
- 18.Newman S. Building microservices: designing fine-grained systems 2nd edition. Sebastopol : O'Reilly Media, 2021. Vol. 2. 612 p.
- 19.Provide / inject. *Vue.js - The Progressive JavaScript Framework | Vue.js*. URL: <https://vuejs.org/guide/components/provide-inject.html> (дата звернення: 12.05.2026).
- 20.Rapidly build modern websites without ever leaving your HTML. *Tailwind CSS*. URL: <https://tailwindcss.com/> (дата звернення: 07.04.2026).
- 21.Reka. *Reka UI - Unstyled, fully accessible UI library | Reka UI*. URL: <https://reka-ui.com/docs/overview/introduction> (дата звернення: 13.05.2026).
- 22.RFC 6238: tOTP: Time-Based One-Time Password Algorithm / M. D та ін. *Internet Engineering Task Force (IETF)*. URL: <https://www.rfc-editor.org/info/rfc6238/> (дата звернення: 23.03.2026).
- 23.RFC 7519: JSON web token (JWT) / M. Jones та ін. *Internet Engineering Task Force (IETF)*. URL: <https://www.rfc-editor.org/info/rfc7519/> (дата звернення: 13.03.2026).
- 24.Richardson L., Ruby S. RESTful web services. Sebastopol : O'Reilly Media, 2007. 446 p.

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				98

25. Scalar - The OpenAPI Company. *Scalar*. URL: <https://scalar.com/> (дата звернення: 27.03.2026).
26. Shvets A. Dive into design patterns. Refactoring.Guru, 2022. 411 p.
27. Spaceship, Inc. Business email solution spacemail by spaceship. *Spacemail*. URL: <https://www.spacemail.com/> (дата звернення: 15.04.2026).
28. TechEmpower framework benchmarks. *TechEmpower*. URL: <https://www.techempower.com/benchmarks/> (дата звернення: 24.03.2026).
29. The intuitive store for vue.js. *Pinia*. URL: <https://pinia.vuejs.org/> (дата звернення: 07.04.2026).
30. The modular and type safe schema library. *Valibot*. URL: <https://valibot.dev/> (дата звернення: 10.04.2026).
31. The PostgreSQL Global Development Group. The world's most advanced open source relational database. *PostgreSQL*. URL: <https://www.postgresql.org/> (дата звернення: 27.03.2026).
32. The Postman Team. What is API rate limiting? Understanding request throttling and best practices. *Postman Blog*. URL: <https://blog.postman.com/what-is-api-rate-limiting/> (дата звернення: 04.05.2026).
33. The simple yet powerful mediator. *MediatR*. URL: <https://mediatr.io/> (дата звернення: 27.03.2026).
34. Topsy bartender. *Topsy Bartender*. URL: <https://tipsybartender.com/> (дата звернення: 12.03.2026).
35. Tornroth M. Cocktail culture: A history of mixology & its evolution - A bar above. *A Bar Above*. URL: <https://abarabove.com/cocktail-culture-a-history-of-mixology-its-evolution/> (дата звернення: 06.03.2026).
36. Usage. *Nuxt II8n*. URL: <https://i18n.nuxtjs.org/docs/getting-started/usage> (дата звернення: 08.05.2026).
37. Wilton H. Rendering for SEO in 2026: hybrid, islands, and edge · nuxt SEO. *Nuxt SEO*. URL: <https://nuxtseo.com/learn-seo/nuxt/routes-and-rendering/rendering> (дата звернення: 24.03.2026).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				99

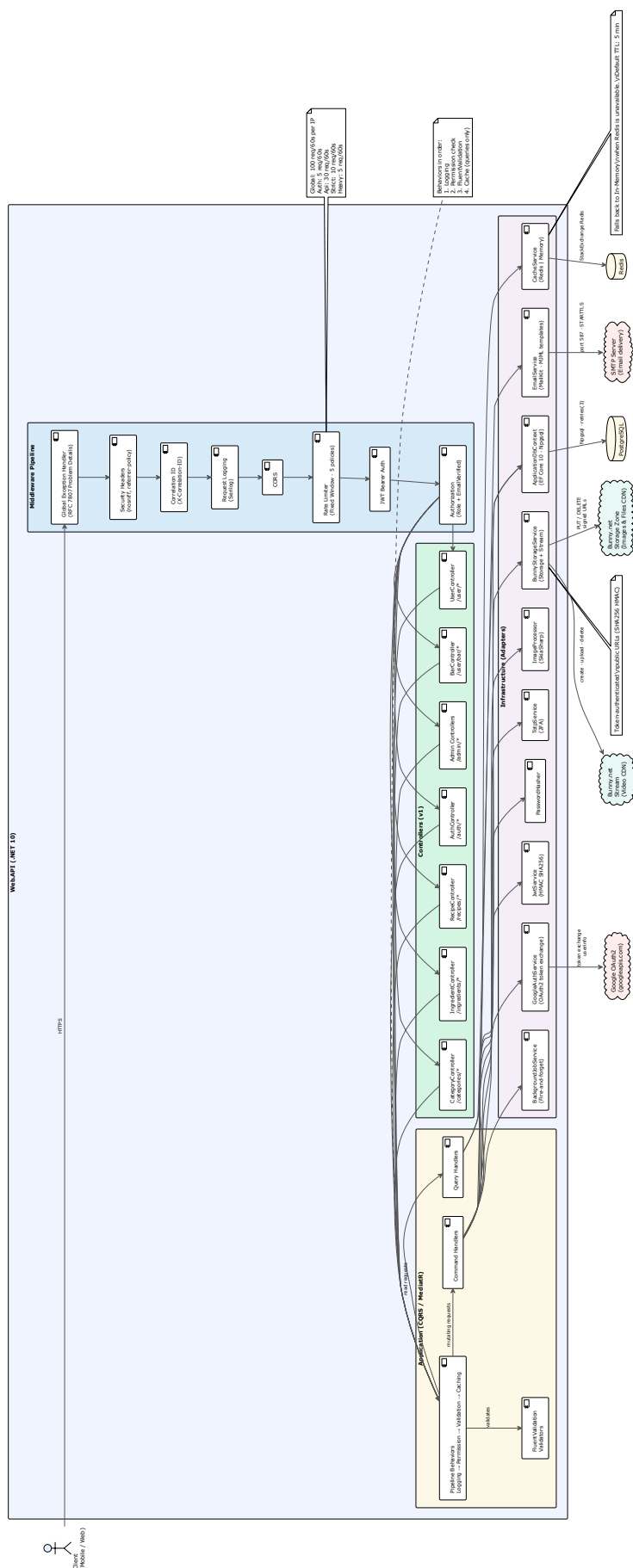
38.Zasadnyy O. Jigger: cocktail drink recipes app - app store. *App Store*. URL: <https://apps.apple.com/ua/app/jigger-cocktail-drink-recipes/id1497308666> (дата звернення: 12.03.2026).

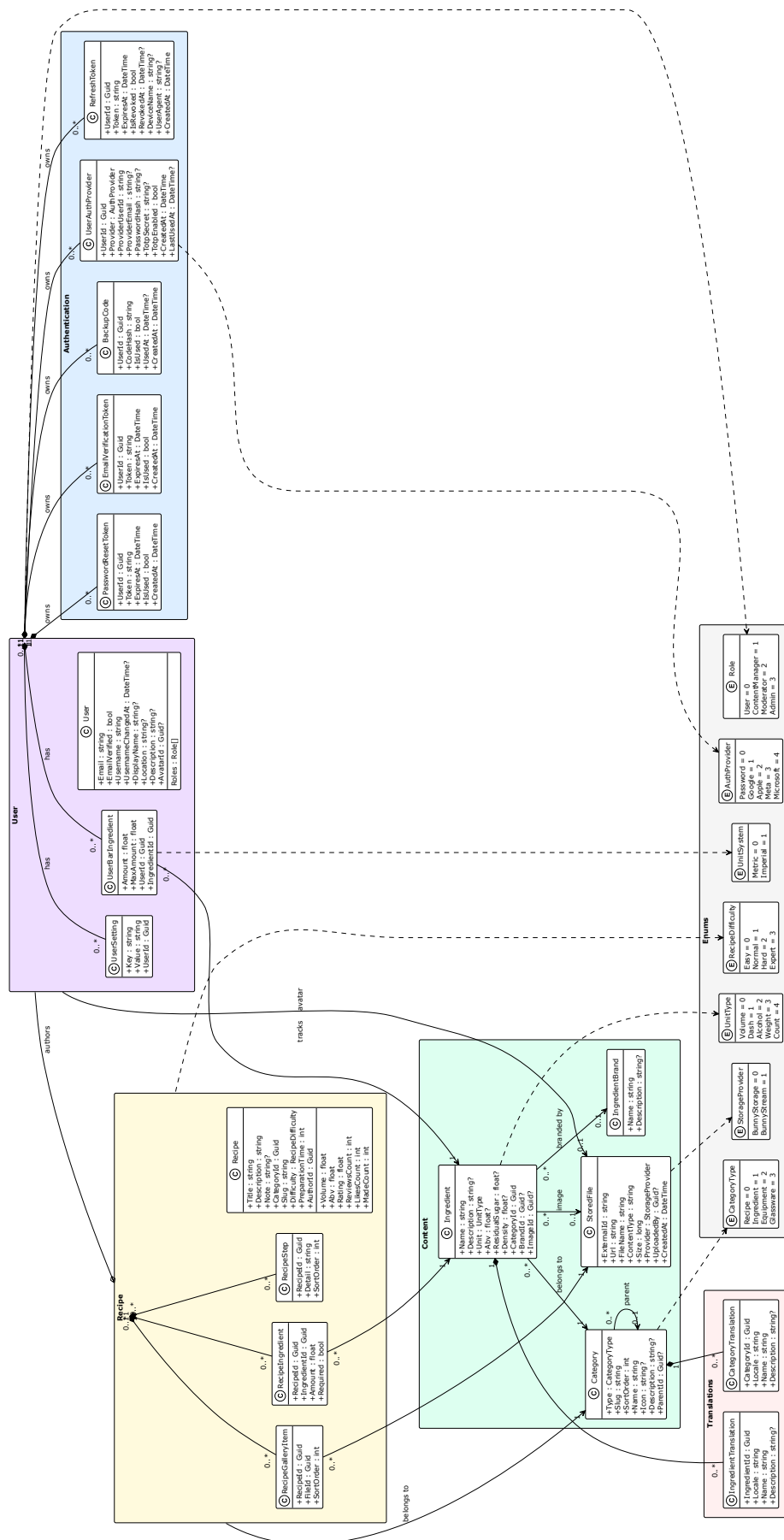
39.Zasadnyy O. Jigger: cocktail drink recipes - apps on google play. *Android Apps on Google Play*. URL: <https://play.google.com/store/apps/details?id=com.zasadnyy.jigger> (дата звернення: 12.03.2026).

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				100

ДОДАТКИ

		Шевцов М. С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				101





```

using Application.Common.Interfaces;
using Application.Common.Messaging;
using Application.Common.Models;
using Application.Features.Category;
using Application.Features.Ingredient;
using Domain.Entities;
using Microsoft.EntityFrameworkCore;
namespace Application.Features.Bar.Queries.GetMyBarIngredients;

public sealed class GetMyBarIngredientsQueryHandler(
    IApplicationDbContext context,
    ICurrentUserService currentUserService,
    ILocaleService localeService,
    IStorageService storageService,
    ICategoryService categoryService
) : IQueryHandler<GetMyBarIngredientsQuery, PagedList<BarIngredientResponse>>
{
    public async Task<Result<PagedList<BarIngredientResponse>>> Handle(
        GetMyBarIngredientsQuery request,
        CancellationToken cancellationToken
    )
    {
        Guid userId = currentUserService.UserId;
        string locale = await localeService.ResolveAsync(cancellationToken);

        IQueryable<UserBarIngredient> query = context.UserBarIngredients.Where(b => b.UserId ==
userId);

        if (request.CategoryId.HasValue)
        {
            HashSet<Guid> descendantIds = await categoryService.GetDescendantIdsAsync(
                request.CategoryId.Value,
                cancellationToken
            );
            query = query.Where(b => descendantIds.Contains(b.Ingredient.CategoryId));
        }

        if (request.Critical == true || request.Attention == true)
        {
            query = query.Where(b =>
                (request.Critical == true && b.MaxAmount > 0 && b.Amount / b.MaxAmount <= 0.25f)
                || (request.Attention == true && b.MaxAmount > 0 && b.Amount / b.MaxAmount <= 0.50f)
            );
        }

        if (!string.IsNullOrEmpty(request.Search))
        {
            string pattern = $"{request.Search.Trim().ToLower()}%";
            query = query.Where(b =>
                EF.Functions.Like(b.Ingredient.Name.ToLower(), pattern)
                || b.Ingredient.Translations.Any(t => EF.Functions.Like(t.Name.ToLower(), pattern))
                || (b.Ingredient.Brand != null &&
EF.Functions.Like(b.Ingredient.Brand.Name.ToLower(), pattern))
                || EF.Functions.Like(b.Ingredient.Category.Name.ToLower(), pattern)
                || b.Ingredient.Category.Translations.Any(t => EF.Functions.Like(t.Name.ToLower(),
pattern))
            );
        }

        int total = await query.CountAsync(cancellationToken);

        if (total == 0)
            return Result.Success(PagedList<BarIngredientResponse>.Empty(request.Page,
request.Size));

        IQueryable<UserBarIngredient> sorted = request.Sort switch
        {
            BarIngredientSort.NameDesc => query.OrderByDescending(b => b.Ingredient.Name),
            BarIngredientSort.AmountAsc => query.OrderBy(b => b.Amount),
        }
    }
}

```

```

        BarIngredientSort.AmountDesc => query.OrderByDescending(b => b.Amount),
        BarIngredientSort.RemainingPercentAsc => query.OrderBy(b => b.MaxAmount > 0 ? b.Amount /
b.MaxAmount : 0f),
        BarIngredientSort.RemainingPercentDesc => query.OrderByDescending(b =>
        b.MaxAmount > 0 ? b.Amount / b.MaxAmount : 0f
    ),
    _ => query.OrderBy(b => b.Ingredient.Name),
};

List<UserBarIngredient> items = await sorted
    .Skip((request.Page - 1) * request.Size)
    .Take(request.Size)
    .Include(b => b.Ingredient)
    .ThenInclude(i => i.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
    .Include(b => b.Ingredient)
    .ThenInclude(i => i.Category)
    .ThenInclude(c => c.Translations.Where(t => t.Locale == locale || t.Locale ==
"en"))
    .Include(b => b.Ingredient)
    .ThenInclude(i => i.Category)
    .ThenInclude(c => c.Parent)
    .ThenInclude(p => p!.Translations.Where(t => t.Locale == locale || t.Locale
== "en"))
    .Include(b => b.Ingredient)
    .ThenInclude(i => i.Brand)
    .Include(b => b.Ingredient)
    .ThenInclude(i => i.Image)
    .ToListAsync(cancellationToken);

var responses = items
    .Select(b =>
    {
        Domain.Entities.Ingredient ingredient = b.Ingredient;
        Domain.Entities.Category category = ingredient.Category;
        Domain.Entities.Category mainCategory = category.Parent ?? category;

        float remainingPercent = b.MaxAmount > 0 ? b.Amount / b.MaxAmount : 0f;
        bool? isCritical = b.MaxAmount > 0 ? remainingPercent <= 0.25f : null;
        bool? isAttention = b.MaxAmount > 0 ? remainingPercent <= 0.50f : null;

        return new BarIngredientResponse
        {
            Id = ingredient.Id,
            Ingredient = new IngredientShortResponse
            {
                Id = ingredient.Id,
                Name = ingredient.GetName(locale),
                ImageUrl =
                    ingredient.Image != null ?
storageService.GenerateTemporaryUrl(ingredient.Image) : null,
                BrandName = ingredient.Brand?.Name,
                MainCategoryName = mainCategory.GetName(locale),
                Category = new CategoryShortResponse
                {
                    Id = category.Id,
                    Slug = category.Slug,
                    Name = category.GetName(locale),
                },
            },
            Amount = b.Amount,
            MaxAmount = b.MaxAmount,
            RemainingPercent = remainingPercent,
            IsCritical = isCritical,
            IsAttention = isAttention,
        };
    })
    .ToListAsync();

return Result.Success(new PagedList<BarIngredientResponse>(responses, total, request.Page,
request.Size));
}

```

```

using Application.Common.Interfaces;
using Application.Common.Messaging;
using Application.Common.Models;
using Domain.Enums;
using Domain.Settings;
using Microsoft.EntityFrameworkCore;

namespace Application.Features.Recipe.Queries.GetRecipes;

public sealed class GetRecipesQueryHandler(
    IApplicationDbContext context,
    ICurrentUserService currentUserService,
    IStorageService storageService,
    ICategoryService categoryService
) : IQueryHandler<GetRecipesQuery, PagedList<RecipeCardResponse>>
{
    private static readonly TimeSpan s_newBadgeThreshold = TimeSpan.FromDays(30);

    public async Task<Result<PagedList<RecipeCardResponse>>> Handle(
        GetRecipesQuery request,
        CancellationToken cancellationToken
    )
    {
        IQueryable<Domain.Entities.Recipe> query = context.Recipes;

        if (!string.IsNullOrEmpty(request.Search))
        {
            string pattern = $"{request.Search.Trim().ToLower()}%";
            query = query.Where(r =>
                EF.Functions.Like(r.Title.ToLower(), pattern) ||
                EF.Functions.Like(r.Description.ToLower(), pattern)
            );
        }

        if (request.Categories is { Length: > 0 })
        {
            HashSet<Guid>[] descendantSets = await Task.WhenAll(
                request.Categories.Select(id => categoryService.GetDescendantIdsAsync(id,
                    cancellationToken))
            );
            var categoryIds = descendantSets.SelectMany(s => s).ToHashSet();
            query = query.Where(r => categoryIds.Contains(r.CategoryId));
        }

        if (request.Difficulties is { Length: > 0 })
        {
            ICollection<RecipeDifficulty> difficulties = request.Difficulties;
            query = query.Where(r => difficulties.Contains(r.Difficulty));
        }

        if (request.MinimumRating.HasValue)
            query = query.Where(r => r.Rating >= request.MinimumRating.Value - 0.5f);

        if (request.StrengthRange is { Length: >= 2 })
        {
            float minAbv = request.StrengthRange[0];
            float maxAbv = request.StrengthRange[1];

            if (currentUserService.IsAuthenticated)
            {
                string? unitValue = await context
                    .UserSettings.Where(s => s.UserId == currentUserService.UserId && s.Key ==
                        UserSettingKey.Unit)
                    .Select(s => s.Value)
                    .FirstOrDefaultAsync(cancellationToken);

                if (
                    unitValue is not null
                    && Enum.TryParse(unitValue, ignoreCase: true, out UnitSystem unitSystem)
                )
            }
        }
    }
}

```

```

        && unitSystem == UnitSystem.Imperial
    )
    {
        minAbv = UnitType.Alcohol.Convert(UnitSystem.Imperial, UnitSystem.Metric,
minAbv);
        maxAbv = UnitType.Alcohol.Convert(UnitSystem.Imperial, UnitSystem.Metric,
maxAbv);
    }

    query = query.Where(r => r.Abv >= minAbv && r.Abv <= maxAbv);
}

if (request.PreparationTimeRange is { Length: >= 2 })
{
    query = query.Where(r =>
        r.PreparationTime >= request.PreparationTimeRange[0]
        && r.PreparationTime <= request.PreparationTimeRange[1]
    );
}

if (request.Ingredients is { Length: > 0 })
{
    query = request.Ingredients.Aggregate(
        query,
        (current, ingredientId) => current.Where(r => r.Ingredients.Any(ri =>
ri.IngredientId == ingredientId))
    );
}

HashSet<Guid>? barIngredientIds = null;
if (currentUserService.IsAuthenticated && request.CanMake)
{
    Guid userId = currentUserService.UserId;
    barIngredientIds = await context
        .UserBarIngredients.Where(b => b.UserId == userId)
        .Select(b => b.IngredientId)
        .ToHashSetAsync(cancellationToken);

    HashSet<Guid> capturedBarIds = barIngredientIds;
    query = query.Where(r =>
        !r.Ingredients.Any(ri => ri.Required && !capturedBarIds.Contains(ri.IngredientId))
    );
}

int total = await query.CountAsync(cancellationToken);

if (total == 0)
    return Result.Success(PagedList<RecipeCardResponse>.Empty(request.Page, request.Size));

IQueryable<Domain.Entities.Recipe> sorted = request.Sort switch
{
    RecipesSort.TitleDesc => query.OrderByDescending(r => r.Title),
    RecipesSort.PopularityAsc => query.OrderBy(r => r.ReviewsCount),
    RecipesSort.PopularityDesc => query.OrderByDescending(r => r.ReviewsCount),
    RecipesSort.LikesAsc => query.OrderBy(r => r.LikesCount),
    RecipesSort.LikesDesc => query.OrderByDescending(r => r.LikesCount),
    RecipesSort.MadeAsc => query.OrderBy(r => r.MadeCount),
    RecipesSort.MadeDesc => query.OrderByDescending(r => r.MadeCount),
    RecipesSort.RatingAsc => query.OrderBy(r => r.Rating),
    RecipesSort.RatingDesc => query.OrderByDescending(r => r.Rating),
    RecipesSort.PreparationTimeAsc => query.OrderBy(r => r.PreparationTime),
    RecipesSort.PreparationTimeDesc => query.OrderByDescending(r => r.PreparationTime),
    RecipesSort.StrengthAsc => query.OrderBy(r => r.Abv),
    RecipesSort.StrengthDesc => query.OrderByDescending(r => r.Abv),
    RecipesSort.VolumeAsc => query.OrderBy(r => r.Volume),
    RecipesSort.VolumeDesc => query.OrderByDescending(r => r.Volume),
    _ => query.OrderBy(r => r.Title),
};

List<Domain.Entities.Recipe> items = await sorted
    .Skip((request.Page - 1) * request.Size)

```

```

        .Take(request.Size)
        .Include(r => r.Author)
        .Include(r => r.Gallery.OrderBy(g => g.SortOrder).Take(1))
            .ThenInclude(g => g.File)
        .Include(r => r.Ingredients)
        .ToListAsync(cancellationToken);

    if (currentUserService.IsAuthenticated)
    {
        Guid userId = currentUserService.UserId;

        barIngredientIds ??= await context
            .UserBarIngredients.Where(b => b.UserId == userId)
            .Select(b => b.IngredientId)
            .ToHashSetAsync(cancellationToken);

        DateTime newBadgeCutoff = DateTime.UtcNow - s_newBadgeThreshold;

        var responses = items
            .Select(r =>
            {
                var badges = new List<RecipeBadge>();
                if (r.CreatedAt >= newBadgeCutoff)
                    badges.Add(RecipeBadge.New);

                Domain.Entities.RecipeGalleryItem? thumbnail = r.Gallery.MinBy(g => g.SortOrder);

                RecipeCardIngredientsResponse? ingredientsResponse = null;
                if (barIngredientIds is not null)
                {
                    int required = r.Ingredients.Count(ri => ri.Required);
                    int available = r.Ingredients.Count(ri =>
                        ri.Required && barIngredientIds.Contains(ri.IngredientId)
                    );
                    ingredientsResponse = new RecipeCardIngredientsResponse(required, available);
                }

                return new RecipeCardResponse(
                    r.Id,
                    r.Slug,
                    r.Title,
                    thumbnail is not null ? storageService.GenerateTemporaryUrl(thumbnail.File) :
null,
                    [.. badges],
                    r.Difficulty,
                    r.PreparationTime,
                    r.Rating,
                    r.LikesCount,
                    r.MadeCount,
                    r.Abv,
                    false,
                    new RecipeCardAuthorResponse(r.Author.Username, r.Author.DisplayName),
                    ingredientsResponse
                );
            })
            .ToListAsync();

        return Result.Success(new PagedList<RecipeCardResponse>(responses, total, request.Page,
request.Size));
    }
}

```

```

public sealed class GetRecipeBySlugQueryHandler(
    IApplicationDbContext context,
    ICurrentUserService currentUserService,
    ILocaleService localeService,
    IStorageService storageService,
    ICategoryService categoryService
) : IQueryHandler<GetRecipeBySlugQuery, RecipeDetailsResponse>
{
    public async Task<Result<RecipeDetailsResponse>> Handle(
        GetRecipeBySlugQuery request,
        CancellationToken cancellationToken
    )
    {
        string locale = await localeService.ResolveAsync(cancellationToken);

        UnitSystem unitSystem = UnitSystem.Metric;
        if (currentUserService.IsAuthenticated)
        {
            string? unitValue = await context
                .UserSettings.Where(s => s.UserId == currentUserService.UserId && s.Key ==
                UserSettingKey.Unit)
                .Select(s => s.Value)
                .FirstOrDefaultAsync(cancellationToken);

            if (unitValue is not null && Enum.TryParse(unitValue, ignoreCase: true, out UnitSystem
                parsed))
            {
                unitSystem = parsed;
            }

            Domain.Entities.Recipe? recipe = await context
                .Recipes.Where(r => r.Author.Username == request.Username && r.Slug == request.Slug)
                .Include(r => r.Author)
                .ThenInclude(a => a.Avatar)
                .Include(r => r.Category)
                .ThenInclude(c => c.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
                .Include(r => r.Gallery.OrderBy(g => g.SortOrder))
                .ThenInclude(g => g.File)
                .Include(r => r.Steps.OrderBy(s => s.SortOrder))
                .Include(r => r.Ingredients)
                .ThenInclude(ri => ri.Ingredient)
                .ThenInclude(i => i.Translations.Where(t => t.Locale == locale || t.Locale ==
                "en"))
                .Include(r => r.Ingredients)
                .ThenInclude(ri => ri.Ingredient)
                .ThenInclude(i => i.Image)
                .FirstOrDefaultAsync(cancellationToken);

            if (recipe is null)
            {
                return Result.Failure<RecipeDetailsResponse>(Error.NotFound("Recipe not found."));
            }

            Dictionary<Guid, CategoryShortResponse[]> breadcrumbs = await
            categoryService.BuildBreadcrumbsAsync(
                [recipe.CategoryId],
                locale,
                cancellationToken
            );

            CategoryResponse categoryResponse = new(
                recipe.Category.Id,
                recipe.Category.Slug,
                recipe.Category.GetName(locale),
                recipe.Category.SortOrder,
                breadcrumbs.GetValueOrDefault(recipe.CategoryId, [])
            );

            HashSet<Guid>? barIngredientIds = null;
            Dictionary<Guid, IngredientShortResponse>? substituteByCategory = null;

            if (currentUserService.IsAuthenticated)

```

```

{
    Guid userId = currentUserService.UserId;

    barIngredientIds = await context
        .UserBarIngredients.Where(b => b.UserId == userId)
        .Select(b => b.IngredientId)
        .ToHashSetAsync(cancellationToken);

    var unavailableCategoryIds = recipe
        .Ingredients.Where(ri => !barIngredientIds.Contains(ri.IngredientId))
        .Select(ri => ri.Ingredient.CategoryId)
        .ToHashSet();

    if (unavailableCategoryIds.Count > 0)
    {
        List<UserBarIngredient> barSubstitutes = await context
            .UserBarIngredients.Where(b =>
                b.UserId == userId &&
                unavailableCategoryIds.Contains(b.Ingredient.CategoryId)
            )
            .Include(b => b.Ingredient)
            .ThenInclude(i => i.Translations.Where(t => t.Locale == locale || t.Locale
                == "en"))
            .Include(b => b.Ingredient)
            .ThenInclude(i => i.Category)
            .ThenInclude(c => c.Translations.Where(t => t.Locale == locale ||
                t.Locale == "en"))
            .Include(b => b.Ingredient)
            .ThenInclude(i => i.Category)
            .ThenInclude(c => c.Parent)
            .ThenInclude(p => p!.Translations.Where(t => t.Locale == locale ||
                t.Locale == "en"))
            .Include(b => b.Ingredient)
            .ThenInclude(i => i.Brand)
            .Include(b => b.Ingredient)
            .ThenInclude(i => i.Image)
            .ToListAsync(cancellationToken);

        substituteByCategory = barSubstitutes
            .GroupBy(b => b.Ingredient.CategoryId)
            .ToDictionary(g => g.Key, g => MapIngredientShort(g.First().Ingredient, locale,
                storageService));
    }

    RecipeDetailsIngredientResponse[] ingredients = recipe
        .Ingredients.Select(ri =>
        {
            Domain.Entities.Ingredient ingredient = ri.Ingredient;
            float convertedAmount = ingredient.Unit.Convert(UnitSystem.Metric, unitSystem,
                ri.Amount);

            bool? available = barIngredientIds?.Contains(ingredient.Id);
            IngredientShortResponse? substitute = null;

            if (available == false)
                substituteByCategory?.TryGetValue(ingredient.CategoryId, out substitute);

            return new RecipeDetailsIngredientResponse(
                ingredient.Id,
                ingredient.GetName(locale),
                ingredient.Image != null ? storageService.GenerateTemporaryUrl(ingredient.Image)
                : null,

                convertedAmount,
                ingredient.Unit,
                ri.Required,
                available,
                substitute
            );
        })
        .ToArray();
}

```

```

RecipeGalleryItemResponse[] gallery = recipe
    .Gallery.Select(g => new RecipeGalleryItemResponse(
        storageService.GenerateTemporaryUrl(g.File),
        g.SortOrder
    ))
    .ToArray();

RecipeDetailsAuthorResponse author = new(
    recipe.Author.Username,
    recipe.Author.DisplayName,
    recipe.Author.Avatar != null ? storageService.GenerateTemporaryUrl(recipe.Author.Avatar)
: null
);

float volume = UnitType.Volume.Convert(UnitSystem.Metric, unitSystem, recipe.Volume);
float abv = UnitType.Alcohol.Convert(UnitSystem.Metric, unitSystem, recipe.Abv);

return Result.Success(
    new RecipeDetailsResponse(
        recipe.Id,
        recipe.Slug,
        recipe.Title,
        recipe.Description,
        recipe.Note,
        categoryResponse,
        recipe.Difficulty,
        author,
        recipe.PreparationTime,
        volume,
        abv,
        recipe.Rating,
        recipe.ReviewsCount,
        recipe.LikesCount,
        recipe.MadeCount,
        recipe.Steps.Select(s => s.Detail).ToArray(),
        gallery,
        ingredients
    )
);
}

private static IngredientShortResponse MapIngredientShort(
    Domain.Entities.Ingredient ingredient,
    string locale,
    IStorageService storageService
)
{
    Domain.Entities.Category category = ingredient.Category;
    Domain.Entities.Category mainCategory = category.Parent ?? category;

    return new IngredientShortResponse
    {
        Id = ingredient.Id,
        Name = ingredient.GetName(locale),
        ImageUrl = ingredient.Image != null ?
storageService.GenerateTemporaryUrl(ingredient.Image) : null,
        BrandName = ingredient.Brand?.Name,
        MainCategoryName = mainCategory.GetName(locale),
        Category = new CategoryShortResponse
        {
            Id = category.Id,
            Slug = category.Slug,
            Name = category.GetName(locale),
        },
    };
}
}

```

		Шевцов М. С.			ИПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т. А.				111

```

/* eslint-disable @typescript-eslint/no-dynamic-delete */
import * as v from 'valibot';

export interface UseValidationOptions<T> {
  initialValues?: Partial<T>;
  validateOn?: 'blur' | 'change' | 'submit';
}

export const useValidation = <
  TSchema extends
    | v.BaseSchema<unknown, unknown, v.BaseIssue<unknown>>
    | v.BaseSchemaAsync<unknown, unknown, v.BaseIssue<unknown>>,
>(
  schema: TSchema,
  options: UseValidationOptions<v.InferOutput<TSchema>> = {},
) => {
  type FormData = v.InferOutput<TSchema>;
  type FieldName = keyof FormData;

  const { t } = useI18n();
  const { initialValues = {} as Partial<FormData>, validateOn = 'blur' } = options;

  const populatedInitialValues = { ...initialValues };
  if ('entries' in schema && !!schema.entries && typeof schema.entries === 'object') {
    Object.keys(schema.entries).forEach((key) => {
      if (!(key in populatedInitialValues)) populatedInitialValues[key as FieldName] = undefined;
    });
  }

  // Form state

  const values = ref<Partial<FormData>>>(populatedInitialValues);
  const errors = ref<Partial<Record<FieldName, string[]>>>>({});
  const touched = ref<Partial<Record<FieldName, boolean>>>>({});
  const dirty = ref<Partial<Record<FieldName, boolean>>>>({});

  // Form meta state

  const isSubmitting = ref(false);
  const isValidating = ref(false);
  const submitCount = ref(0);

  // Computed states

  const isDirty = computed(() => Object.keys(dirty.value).some((k) => dirty.value[k as FieldName]));
  const isValid = computed(() => Object.keys(errors.value).length === 0);
  const isTouched = computed(() => Object.keys(touched.value).some((k) => touched.value[k as FieldName]));

  /**
   * Validate entire form
   * @return Validation result
   */
  const validate = async (): Promise<boolean> => {
    isValidating.value = true;
    clearErrors();

    try {
      const result = await v.safeParseAsync(schema, values.value);

      if (!result.success) {
        const flattened = v.flatten(result.issues);

        if (flattened.nested) {
          Object.entries(flattened.nested).forEach(([key, issues]) => {
            if (issues && issues.length > 0) {
              errors.value[key as FieldName] = issues;
            }
          });
        }
      }
    }
  };

```

```

    }

    if (flattened.root && flattened.root.length > 0) {
        console.warn('Form-level validation errors:', flattened.root);
    }

    return false;
}

return true;
} finally {
    isValidating.value = false;
}
};

/**
 * Validate single field
 * @param field Field to validate
 * @return Validation result
 */
const validateField = async (field: FieldName): Promise<boolean> => {
    clearFieldError(field);

    try {
        const result = await v.safeParseAsync(schema, values.value);

        if (!result.success) {
            const flattened = v.flatten(result.issues);

            if (flattened.nested?.[field as string]) {
                const fieldIssues = flattened.nested[field as string];
                if (fieldIssues && fieldIssues.length > 0) {
                    errors.value[field] = fieldIssues;
                    return false;
                }
            }
        }

        return true;
    } catch (error) {
        console.error(`Validation error for field ${String(field)}:`, error);
        return false;
    }
};

/**
 * Set field value
 * @param field Field to set
 * @param value New value
 */
const setFieldValue = async (field: FieldName, value: unknown): Promise<void> => {
    const oldValue = values.value[field];
    values.value[field] = value;

    if (oldValue !== value) setFieldDirty(field);

    if (validateOn === 'change' && touched.value[field]) await validateField(field);
};

/**
 * Set multiple values at once
 * @param newValues New form values
 */
const setValues = (newValues: Partial<FormData>): void => {
    Object.entries(newValues).forEach(([key, value]) => {
        const field = key as FieldName;
        const oldValue = values.value[field];

        values.value[field] = value;

        if (oldValue !== value) setFieldDirty(field);
    });
};

```

```

};

/**
 * Get current form values
 * @return Current values
 */
const getValues = (): Partial<FormData> => ({ ...values.value });

/**
 * Set field error manually
 * @param field Field to update
 * @param error Error message(s)
 */
const setFieldError = (field: FieldName, error: string | string[] | undefined): void => {
  if (error) {
    if (typeof error === 'string') error = [error];
    errors.value[field] = error;
  } else clearFieldError(field);
};

/**
 * Clear field errors
 * @param field Field to erase
 */
const clearFieldError = (field: FieldName): void => {
  delete errors.value[field];
};

/**
 * Clear all form errors
 */
const clearErrors = (): void => {
  errors.value = {};
};

/**
 * Flag field as touched
 * @param field Field to flag
 * @param isTouched Touched flag
 */
const setFieldTouched = async (field: FieldName, isTouched: boolean = true): Promise<void> => {
  if (isTouched) touched.value[field] = isTouched;
  else delete touched.value[field];

  if (isTouched && validateOn === 'blur') await validateField(field);
};

/**
 * Remove touched flag from all fields
 */
const clearTouched = (): void => {
  touched.value = {};
};

/**
 * Flag field as dirty
 * @param field Field to flag
 * @param isDirty Dirty flag
 */
const setFieldDirty = (field: FieldName, isDirty: boolean = true): void => {
  if (isDirty) dirty.value[field] = isDirty;
  else delete dirty.value[field];
};

/**
 * Remove dirty flag from all fields
 */
const clearDirty = (): void => {
  dirty.value = {};
};

/**

```

```

* Reset entire form
* @param newValues New initial state
*/
const resetForm = (newValues?: Partial<FormData>): void => {
  values.value = { ...(newValues || initialValues) };
  clearErrors();
  clearTouched();
  clearDirty();
  isSubmitting.value = false;
  isValidating.value = false;
  submitCount.value = 0;
};

/**
* Reset field to initial value
* @param field Field to reset
*/
const resetField = (field: FieldName): void => {
  values.value[field] = initialValues[field];
  clearFieldError(field);
  delete touched.value[field];
  setFieldDirty(field, false);
};

/**
* Handle form submission
* @param onValid Callback on successful form validation
* @param onInvalid Callback on invalid form
*/
const handleSubmit = (
  onValid: (values: FormData, dirtyValues: Partial<FormData>) => void | Promise<void>,
  onInvalid?: (errors: Partial<Record<FieldName, string[]>>) => void,
) => {
  return async (event?: Event) => {
    event?.preventDefault();

    Object.keys(values.value).forEach((key) => {
      touched.value[key as FieldName] = true;
    });

    submitCount.value++;
    isSubmitting.value = true;

    try {
      const isValid = await validate();

      if (isValid) {
        const dirtyValues = Object.keys(dirty.value).reduce((acc, key) => {
          if (dirty.value[key as FieldName]) acc[key as FieldName] = values.value[key as FieldName];
          return acc;
        }, {} as Partial<FormData>);

        await onValid(values.value, dirtyValues);
      } else if (onInvalid) onInvalid(errors.value);
    } catch (error) {
      console.error('Form submission error:', error);
      throw error;
    } finally {
      isSubmitting.value = false;
    }
  };
};

/**
* Set field errors from API validation response
* @param apiErrors API validation result record
*/
const setApiErrors = (apiErrors: ApiValidationErrors): void => {
  const fieldNames = Object.keys(values.value);

  for (const [key, issues] of Object.entries(apiErrors)) {

```

```

const normalized = key.toLowerCase();
const match = fieldNames.find((f) => (f as string).toLowerCase() === normalized);
if (match && issues.length > 0) {
  errors.value[match as FieldName] = issues.map((e) => {
    const translatedError = t(`validation.api.${e.code}`);
    return translatedError === `validation.api.${e.code}` ? e.detail : translatedError;
  });
}
};

/**
 * Get field props for v-model binding
 * @param field Field to retrieve
 */
const getFieldProps = (field: FieldName) => ({
  modelValue: values.value[field],
  'onUpdate:modelValue': (value: unknown) => setFieldValue(field, value),
  onBlur: () => setFieldTouched(field),
  error: errors.value[field],
  touched: touched.value[field],
  dirty: dirty.value[field],
});

/**
 * Register field for custom components
 * @param field Field to register
 */
const registerField = (field: FieldName) => ({
  value: computed({
    get: () => values.value[field],
    set: (value: unknown) => setFieldValue(field, value),
  }),
  error: computed(() => errors.value[field]),
  touched: computed(() => touched.value[field]),
  dirty: computed(() => dirty.value[field]),
  setError: (error: string | string[] | undefined) => setFieldError(field, error),
  clearError: () => clearFieldError(field),
  setTouched: (isTouched = true) => setFieldTouched(field, isTouched),
  setDirty: (isDirty = true) => setFieldDirty(field, isDirty),
  reset: () => resetField(field),
});

return {
  // Values

  values: readonly(values) as Readonly<Ref<Partial<FormData>>>,
  errors: readonly(errors),
  touched: readonly(touched),
  dirty: readonly(dirty),

  // Meta state

  isSubmitting: readonly(isSubmitting),
  isValidating: readonly(isValidating),
  submitCount: readonly(submitCount),
  isDirty,
  isValid,
  isTouched,

  // Methods

  validate,
  validateField,
  setFieldValue,
  setValues,
  getValues,
  setFieldError,
  clearFieldError,
  clearErrors,
  setFieldTouched,
  clearTouched,

```

```

    setFieldDirty,
    clearDirty,
    resetForm,
    resetField,
    handleSubmit,
    setApiErrors,
    field: getFieldProps,
    register: registerField,
  });
};

export const vEmail = (field: string) => {
  const { t } = useI18n();
  return v.pipe(v.string(t('validation.required', { field })), v.email(t('validation.email')));
};

export const vUsernameAsync = (field: string, skipValue?: string) => {
  const { t } = useI18n();
  const { checkUsername } = useUserStore();
  return v.pipeAsync(
    v.string(t('validation.required', { field })),
    v.minLength(2, t('validation.minLength', { min: 2 })),
    v.maxLength(128, t('validation.maxLength', { max: 128 })),
    v.regex(/^[a-zA-Z][a-zA-Z0-9]*([_-][a-zA-Z0-9]+)*$/, t('validation.username')),
    v.rawCheckAsync(async ({ dataset, addIssue }) => {
      if (dataset.issues?.length) return;
      if (skipValue !== undefined && dataset.value === skipValue) return;
      const available = await checkUsername(dataset.value as string);
      if (!available) addIssue({ message: t('validation.api.usernameTaken') });
    })
  );
};

export const vPassword = (field: string) => {
  const { t } = useI18n();
  return v.pipe(v.string(t('validation.required', { field })), v.minLength(8, t('validation.minLength', { min: 8 })));
};

export const vTOTP = (field: string) => {
  const { t } = useI18n();
  return v.pipe(
    v.string(t('validation.required', { field })),
    v.length(6, t('validation.length', { length: 6 })),
    v.digits(t('validation.digits')),
  );
};

export { v };

```

```

<script setup lang="ts">
import { cloneDeep, debounce } from 'lodash-es';

import {
  activeFiltersCountKey,
  applyDraftFiltersKey,
  cachedCategoriesKey,
  cachedIngredientsKey,
  categoriesKey,
  categoriesLoadingKey,
  categoriesQueryKey,
  clearAllFiltersKey,
  clearDraftFiltersKey,
  draftFiltersCountKey,
  getRecipesKey,
  hasAppliedKey,
  ingredientsKey,
  ingredientsLoadingKey,
  ingredientsQueryKey,
  isDirtyKey,
  loadMoreCategoriesKey,
  loadMoreIngredientsKey,
  preparationTimeRangeLimitsKey,
  recipesQueryDraftKey,
  recipesQueryKey,
  strengthRangeLimitsKey,
} from '~/features/pages/recipe/search/injectionKeys';

definePageMeta({ keepalive: true });

const { locale } = useI18n();
const recipeApi = useApiRecipes();
const categoryApi = useApiCategory();
const ingredientApi = useApiIngredient();
const userStore = useUserStore();

const strengthRangeLimits = computed<[number, number]>(() => (userStore.unitSystem === 'metric' ?
[0, 50] : [0, 100]));
const preparationTimeRangeLimits: [number, number] = [0, 30];

const defaultQuery: ApiRecipesRequest = {
  page: 1,
  size: 9,
  search: '',
  canMake: false,
  made: false,
  excludeAi: false,
  categories: undefined,
  difficulties: undefined,
  minimumRating: undefined,
  strengthRange: undefined,
  preparationTimeRange: undefined,
  ingredients: [],
  glassware: [],
  sort: 'popularityDesc',
};

const recipesQuery = useState<ApiRecipesRequest>('rsi:query', () => cloneDeep(defaultQuery));
const recipesQueryDraft = useState<ApiRecipesRequest>('rsi:queryDraft', () =>
cloneDeep(defaultQuery));
const recipes = useState<ApiRecipeCardResponse[]>('rsi:recipes', () => []);
const recipesTotal = useState<number | undefined>('rsi:total', () => undefined);
const recipesHasNextPage = useState<boolean>('rsi:hasNext', () => false);

const categoriesQuery = useState<ApiCategoriesRequest>('rsi:cat:query', () => ({ page: 1, size: 10,
search: '' }));
const categories = useState<ApiCategoryResponse[]>('rsi:categories', () => []);
const cachedCategories = useState<Record<Guid, ApiCategoryResponse>>('rsi:cachedCat', () => ({}));
const categoriesHasNextPage = useState<boolean>('rsi:cat:hasNext', () => false);

```

```

const ingredientsQuery = useState<ApiIngredientsQuery>('rsi:ing:query', () => ({ page: 1, size: 6,
search: '' }));
const ingredients = useState<ApiIngredientResponse[]>('rsi:ingredients', () => []);
const cachedIngredients = useState<Record<Guid, ApiIngredientResponse>>('rsi:cachedIng', () =>
({}));
const ingredientsHasNextPage = useState<boolean>('rsi:ing:hasNext', () => false);

const recipesLoading = ref(!recipes.value.length);
const categoriesLoading = ref(!categories.value.length);
const ingredientsLoading = ref(!ingredients.value.length);

const filterKeys = (query: ApiRecipesRequest) =>
  Object.fromEntries(Object.entries(query).filter(([k]) => !['page', 'size', 'sort'].includes(k)));

const countActiveFilters = (query: ApiRecipesRequest) =>
  Object.values(filterKeys(query)).filter(
    (v) =>
      v === true || (typeof v === 'number' && v > 0) || ((typeof v === 'string' || Array.isArray(v))
&& v.length > 0),
  ).length;

const activeFiltersCount = computed(() => countActiveFilters(recipesQuery.value));
const draftFiltersCount = computed(() => countActiveFilters(recipesQueryDraft.value));
const isDirty = computed(
  () => JSON.stringify(filterKeys(recipesQuery.value)) !==
JSON.stringify(filterKeys(recipesQueryDraft.value)),
);
const hasApplied = computed(() => activeFiltersCount.value > 0);

const getCategories = async (upsert: boolean) => {
  if (upsert && (categoriesLoading.value || !categoriesHasNextPage.value)) return;
  categoriesLoading.value = true;
  if (!upsert) categories.value = [];
  categoriesQuery.value.page = upsert ? (categoriesQuery.value.page ?? 1) + 1 : 1;
  const result = await recipeApi.fetchCategories(categoriesQuery.value);
  categories.value = upsert ? [...categories.value, ...result.items] : result.items;
  categoriesHasNextPage.value = !!result.next;
  for (const category of result.items) cachedCategories.value[category.id] = category;
  categoriesLoading.value = false;
};
const debouncedSearchCategories = debounce(async () => await getCategories(false), 400);
const loadMoreCategories = async () => await getCategories(true);

const getIngredients = async (upsert: boolean) => {
  if (upsert && (ingredientsLoading.value || !ingredientsHasNextPage.value)) return;
  ingredientsLoading.value = true;
  if (!upsert) ingredients.value = [];
  ingredientsQuery.value.page = upsert ? (ingredientsQuery.value.page ?? 1) + 1 : 1;
  const result = await ingredientApi.fetchIngredients(ingredientsQuery.value);
  ingredients.value = upsert ? [...ingredients.value, ...result.items] : result.items;
  ingredientsHasNextPage.value = !!result.next;
  for (const ingredient of result.items) cachedIngredients.value[ingredient.id] = ingredient;
  ingredientsLoading.value = false;
};
const debouncedSearchIngredients = debounce(async () => await getIngredients(false), 400);
const loadMoreIngredients = async () => await getIngredients(true);

const resolveSelected = async () => {
  const selectedCategoryIds = recipesQuery.value.categories ?? [];
  const selectedIngredientIds = recipesQuery.value.ingredients ?? [];

  const [resolvedCategories, resolvedIngredients] = await Promise.all([
    selectedCategoryIds.length > 0 ? categoryApi.resolveCategories(selectedCategoryIds) :
Promise.resolve([]),
    selectedIngredientIds.length > 0 ? ingredientApi.resolveIngredients(selectedIngredientIds) :
Promise.resolve([]),
  ]);

  for (const category of resolvedCategories) cachedCategories.value[category.id] = category;
  for (const ingredient of resolvedIngredients) cachedIngredients.value[ingredient.id] = ingredient;
};

```

```

const getRecipes = async (upsert: boolean) => {
  if (upsert && (recipesLoading.value || !recipesHasNextPage.value)) return;

  recipesLoading.value = true;
  if (!upsert) recipes.value = [];

  recipesQuery.value.page = upsert ? (recipesQuery.value.page ?? 1) + 1 : 1;
  const updatedQuery = cloneDeep(toRaw(recipesQuery.value)) as ApiRecipesRequest;
  if (updatedQuery.strengthRange && updatedQuery.strengthRange[1] >= strengthRangeLimits.value[1])
    updatedQuery.strengthRange[1] = 999999;
  if (updatedQuery.preparationTimeRange && updatedQuery.preparationTimeRange[1] >=
    preparationTimeRangeLimits[1])
    updatedQuery.preparationTimeRange[1] = 999999;

  const result = await recipeApi.fetchRecipes(updatedQuery);
  recipes.value = upsert ? [...recipes.value, ...result.items] : result.items;
  recipesTotal.value = result.total;
  recipesHasNextPage.value = !!result.next;
  recipesLoading.value = false;
};

const debouncedSearchRecipes = debounce(async () => await getRecipes(false), 400);
const loadMoreRecipes = async () => await getRecipes(true);

const applyDraftFilters = async () => {
  recipesQuery.value = cloneDeep(toRaw(recipesQueryDraft.value));
  await getRecipes(false);
};

const clearDraftFilters = () => {
  recipesQueryDraft.value = cloneDeep(toRaw(recipesQuery.value));
};

const clearAllFilters = async () => {
  recipesQueryDraft.value = cloneDeep(defaultQuery);
  await applyDraftFilters();
};

const sentinel = useTemplateRef<HTMLElement>('recipeSentinel');
useInfiniteScroll(sentinel, loadMoreRecipes);

provide(strengthRangeLimitsKey, strengthRangeLimits);
provide(preparationTimeRangeLimitsKey, preparationTimeRangeLimits);
provide(recipesQueryKey, recipesQuery);
provide(recipesQueryDraftKey, recipesQueryDraft);
provide(categoriesQueryKey, categoriesQuery);
provide(categoriesKey, categories);
provide(categoriesLoadingKey, categoriesLoading);
provide(cachedCategoriesKey, cachedCategories);
provide(ingredientsQueryKey, ingredientsQuery);
provide(ingredientsKey, ingredients);
provide(ingredientsLoadingKey, ingredientsLoading);
provide(cachedIngredientsKey, cachedIngredients);
provide(activeFiltersCountKey, activeFiltersCount);
provide(draftFiltersCountKey, draftFiltersCount);
provide(isDirtyKey, isDirty);
provide(hasAppliedKey, hasApplied);
provide(loadMoreCategoriesKey, loadMoreCategories);
provide(loadMoreIngredientsKey, loadMoreIngredients);
provide(applyDraftFiltersKey, applyDraftFilters);
provide(clearDraftFiltersKey, clearDraftFilters);
provide(clearAllFiltersKey, clearAllFilters);
provide(getRecipesKey, getRecipes);

watch(
  () => recipesQueryDraft.value.search,
  async () => {
    recipesQuery.value.search = recipesQueryDraft.value.search;
    await debouncedSearchRecipes();
  },
);
watch(

```

```

() => recipesQueryDraft.value.sort,
async () => {
  recipesQuery.value.sort = recipesQueryDraft.value.sort;
  debouncedSearchRecipes.cancel();
  await getRecipes(false);
},
);
watch(() => categoriesQuery.value.search, debouncedSearchCategories);
watch(() => ingredientsQuery.value.search, debouncedSearchIngredients);

watch(
  () => locale.value,
  async () => {
    await Promise.all([getCategories(false), getIngredients(false), resolveSelected()]);
  },
);

onMounted(async () => {
  if (recipes.value.length > 0) return;
  await Promise.all([getRecipes(false), getCategories(false), getIngredients(false)]);
});

onUnmounted(() => {
  debouncedSearchRecipes.cancel();
  debouncedSearchCategories.cancel();
  debouncedSearchIngredients.cancel();
});
</script>

<template>
  <div class="container mx-auto px-4">
    <div class="mb-8">
      <h1 class="text-foreground mb-2 text-4xl font-bold">{{ $t('page.recipes.search.title') }}</h1>
      <p class="text-muted-foreground">{{ $t('page.recipes.search.subtitle') }}</p>
    </div>

    <div class="flex flex-col items-start gap-6 lg:flex-row">
      <PageRecipeSearchFilterMobile />
      <PageRecipeSearchFilterSidebar />

      <div class="flex-1 space-y-6">
        <PageRecipeSearchFilterActiveRow />

        <div class="flex items-center justify-between">
          <p class="text-muted-foreground">
            <template v-if="recipesTotal">
              {{ $t('page.recipes.search.found', recipesTotal) }}
            </template>
          </p>
          <PageRecipeSearchSort v-model="recipesQueryDraft.sort" />
        </div>

        <div v-if="recipesLoading && recipes.length === 0" class="py-8">
          <SharedThrobber />
        </div>

        <SharedUiCard v-else-if="recipes.length === 0" class="glass-dark border-border/10 p-12 text-center">
          <Icon name="lucide:search" class="text-muted-foreground mx-auto mb-4 size-16 opacity-50" />
          <h3 class="mb-2 text-xl font-semibold">{{ $t('page.recipes.search.notFound.title') }}</h3>
          <p class="text-muted-foreground mb-4">{{ $t('page.recipes.search.notFound.subtitle') }}</p>
          <SharedButton variant="outline" :label="$t('page.recipes.search.notFound.action')" @click="clearAllFilters" />
        </SharedUiCard>

        <TransitionGroup
          v-else
          tag="div"
          move-class="transition-transform!"
          class="grid grid-cols-1 gap-6 md:grid-cols-2 lg:grid-cols-3"

```

```

    >
    <PageRecipeSearchRecipeCard v-for="recipe in recipes" :key="recipe.id" :recipe="recipe" />
    <SharedThrobber v-if="recipesLoading" key="recipesThrobber" />
    <div key="recipeSentinel" ref="recipeSentinel" />
  </TransitionGroup>
</div>
</div>
</div>
</template>
<style scoped></style>

```

```

using Application.Common.Interfaces;
using Application.Features.Category;
using Domain.Entities;
using Microsoft.EntityFrameworkCore;

namespace Application.Common.Services;

public sealed class CategoryService(IApplicationDbContext context, ICacheService cacheService) :
    ICategoryService
{
    private const string CacheKey = "category-tree";
    private static readonly TimeSpan s_cacheTtl = TimeSpan.FromHours(6);

    private sealed record CategoryNode(Guid Id, Guid? ParentId, string Slug);

    public async Task<HashSet<Guid>> GetDescendantIdsAsync(Guid rootId, CancellationToken
cancellationToken = default)
    {
        Dictionary<Guid, CategoryNode> tree = await GetTreeAsync(cancellationToken);
        return ComputeDescendants(tree, rootId);
    }

    public async Task<Dictionary<Guid, CategoryShortResponse[]>> BuildBreadcrumbsAsync(
        IEnumerable<Guid> categoryIds,
        string locale,
        CancellationToken cancellationToken = default
    )
    {
        Dictionary<Guid, CategoryNode> tree = await GetTreeAsync(cancellationToken);
        var ids = categoryIds.ToHashSet();
        HashSet<Guid> ancestorIds = CollectAncestorIds(tree, ids);

        Dictionary<Guid, Category> names = await context
            .Categories.Where(c => ancestorIds.Contains(c.Id))
            .Include(c => c.Translations.Where(t => t.Locale == locale || t.Locale == "en"))
            .ToDictionaryAsync(c => c.Id, cancellationToken);

        return ids.ToDictionary(id => id, id => BuildBreadcrumb(tree, names, id, locale));
    }

    private async Task<Dictionary<Guid, CategoryNode>> GetTreeAsync(CancellationToken
cancellationToken)
    {
        CategoryNode[]? cached = await cacheService.GetAsync<CategoryNode[]>(CacheKey,
cancellationToken);
        if (cached is not null)
            return cached.ToDictionary(n => n.Id);

        CategoryNode[] nodes = await context
            .Categories.Select(c => new CategoryNode(c.Id, c.ParentId, c.Slug))
            .ToArrayAsync(cancellationToken);

        await cacheService.SetAsync(CacheKey, nodes, s_cacheTtl, cancellationToken);

        return nodes.ToDictionary(n => n.Id);
    }

    private static HashSet<Guid> ComputeDescendants(Dictionary<Guid, CategoryNode> tree, Guid
rootId)
    {
        var childrenByParent = tree
            .Values.Where(n => n.ParentId.HasValue)
            .GroupBy(n => n.ParentId!.Value)
            .ToDictionary(g => g.Key, g => g.Select(n => n.Id).ToList());

        var result = new HashSet<Guid> { rootId };
        var queue = new Queue<Guid>();
        queue.Enqueue(rootId);
    }

```

```

while (queue.Count > 0)
{
    Guid current = queue.Dequeue();
    if (!childrenByParent.TryGetValue(current, out List<Guid>? childIds))
        continue;
    foreach (Guid childId in childIds.Where(result.Add))
    {
        queue.Enqueue(childId);
    }
}

return result;
}

private static HashSet<Guid> CollectAncestorIds(Dictionary<Guid, CategoryNode> tree,
IEnumerable<Guid> leafIds)
{
    var result = new HashSet<Guid>();
    foreach (Guid leafId in leafIds)
    {
        Guid? current = leafId;
        while (current.HasValue && tree.TryGetValue(current.Value, out CategoryNode? node))
        {
            if (!result.Add(current.Value))
                break;
            current = node.ParentId;
        }
    }
    return result;
}

private static CategoryShortResponse[] BuildBreadcrumb(
Dictionary<Guid, CategoryNode> tree,
Dictionary<Guid, Category> names,
Guid leafId,
string locale
)
{
    var breadcrumb = new List<CategoryShortResponse>();
    Guid? current = leafId;
    while (current.HasValue && tree.TryGetValue(current.Value, out CategoryNode? node))
    {
        if (names.TryGetValue(node.Id, out Category? category))
        {
            breadcrumb.Insert(
                0,
                new CategoryShortResponse
                {
                    Id = category.Id,
                    Slug = node.Slug,
                    Name = category.GetName(locale),
                }
            );
            current = node.ParentId;
        }
    }
    return breadcrumb.ToArray();
}

```