МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА
Факультет економіки і підприємництва
Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2025 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ДОДАТКУ ДЛЯ ОПТИМІЗАЦІЇ ЗБУТУ ХАРЧОВИХ ПРОДУКТІВ

Виконав – студент 4 курсу, групи 41-кн
освітньої програми "Комп’ютерні науки"
спеціальності 122 Комп’ютерні науки
Семенда Дмитро Віталійович

Науковий керівник – к.т.н., доцент
Ліщук Роман Ігорович

Умань – 2025

Зміст

ВСТУП 3

РОЗДІЛ 1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ РЕАЛІЗАЦІЇ НАДЛИШКІВ ХАРЧОВИХ ПРОДУКТІВ 5

1.1. Аналіз стану питання розробки системи оптимізації процесу реалізації харчових продуктів 5

1.2. Огляд наявних аналогів 7

1.3. Аналіз методів розв'язання та постановка задачі 12

РОЗДІЛ 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ДОДАТКУ 15

2.1. Аналіз мов програмування та середовищ розробки для реалізації додатку 15

2.2. Розробка моделі системи та алгоритмів роботи додатку 17

2.3. Розробка структури бази даних 21

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ ОПТИМІЗАЦІЇ ЗБУТУ ХАРЧОВИХ ПРОДУКТІВ 30

3.1. Реалізація модуля реєстрації та авторизації користувача в системі 30

3.2. Реалізація модуля пошуку харчових пропозицій 33

3.3. Реалізація інтерфейсу системи харчових пропозицій 35

ВИСНОВКИ 40

СПИСОК ЛІТЕРАТУРИ 42

ДОДАТКИ 46

ВСТУП

У контексті глобалізації та зростання обсягів споживання проблема раціонального використання харчових ресурсів набуває дедалі більшого значення. Згідно з даними Продовольчої та сільськогосподарської організації ООН (FAO), близько третини усіх вироблених у світі харчових продуктів щорічно втрачаються або йдуть у відходи, що призводить до значних економічних, екологічних і соціальних наслідків. Надлишки харчових продуктів, які залишаються невикористаними в закладах роздрібної торгівлі, громадського харчування чи у домогосподарствах, створюють не лише логістичні, а й моральні дилеми на тлі світової проблеми недоїдання.

Одним із перспективних напрямів вирішення цієї проблеми є створення інформаційно-технологічних рішень, що дозволяють ефективно координувати передачу та реалізацію харчових залишків між постачальниками та споживачами. Розробка спеціалізованого програмного забезпечення для цих цілей здатна забезпечити прозорість, оперативність та доступність таких процесів.

Проблема харчових залишків охоплює всі етапи харчового ланцюга – від виробництва до кінцевого споживання. Незважаючи на наявність численних ініціатив, спрямованих на зменшення втрат їжі, ефективність їх реалізації залишається низькою через відсутність зручних цифрових інструментів для обліку, управління та передачі надлишків. У цьому контексті розробка програмного засобу, що дозволяє об'єднати зацікавлені сторони – підприємства, благодійні організації, споживачів – в єдину інформаційну екосистему, є вкрай важливою.

Застосування сучасних інформаційних технологій, таких як веб-платформи та мобільні додатки, дозволяє не лише зменшити кількість харчових відходів, а й сприяє формуванню екологічно та соціально відповідального підходу до споживання ресурсів.

Метою дослідження є розробка програмного засобу, що забезпечує ефективну реалізацію харчових надлишків шляхом автоматизації процесів пошуку, обміну та обліку продуктів харчування між зацікавленими сторонами.

Об’єктом дослідження є процеси управління харчовими надлишками в умовах цифрової трансформації продовольчих систем.

Предметом дослідження є програмні методи, алгоритми та інструменти, які забезпечують оптимізацію обміну харчовими надлишками шляхом створення веб-застосунку.

У ході дослідження застосовувались такі методи: аналіз предметної області – для вивчення джерел утворення харчових надлишків та моделей їх реалізації; методи системного аналізу – для формалізації вимог до програмного засобу; алгоритмічні методи – для розробки механізмів пошуку та сортування пропозицій за географічним та категорійним критеріями; методи проєктування програмного забезпечення – для створення архітектури веб-застосунку; моделювання та тестування – для оцінювання ефективності запропонованих рішень; засоби розробки інтерфейсів користувача (зокрема фреймворк *Vaadin*) – для реалізації інтерактивного та інтуїтивного інтерфейсу.

РОЗДІЛ 1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ РЕАЛІЗАЦІЇ НАДЛИШКІВ ХАРЧОВИХ ПРОДУКТІВ

1.1. Аналіз стану питання розробки системи оптимізації процесу реалізації харчових продуктів

Проблема раціонального використання харчових ресурсів набула особливої актуальності у зв’язку зі зростанням масштабів харчових втрат по всьому світу. За даними [2], щороку близько третини усіх вироблених харчових продуктів не доходить до кінцевого споживача, що призводить до значних соціальних, економічних та екологічних наслідків. З метою зменшення втрат харчових продуктів дедалі частіше використовуються програмні рішення, які дозволяють оптимізувати процеси обліку, обміну та реалізації харчових надлишків.

На сьогоднішній день існує низка ініціатив, серед яких: онлайн-платформи для передачі їжі (наприклад, Too Good To Go, OLIO), благодійні сервіси, сервіси локального донорства, логістичні рішення на рівні продуктових мереж. Водночас ці сервіси здебільшого зосереджені на конкретних сегментах ринку або мають обмежену функціональність, яка ускладнює масштабування або гнучке налаштування під потреби конкретного регіону чи організації.

Важливим аспектом створення ефективного програмного забезпечення для реалізації харчових надлишків є правильний вибір архітектурного підходу. Одним з найпоширеніших та зручних у підтримці підходів є архітектурний шаблон MVC (рис 1.1) (Model-View-Controller) [5].

Цей шаблон забезпечує логічне розділення програми на три компоненти:

- Model (Модель) – містить логіку обробки даних, зберігає інформацію про об’єкти, наприклад, продукти, користувачів, точки видачі або доставки. Вона також відповідає за роботу з базою даних та бізнес-логіку, таку як

розрахунок термінів зберігання, перевірку наявності або пошук за параметрами.

- View (Представлення) – відповідає за відображення даних користувачу. У контексті веб-застосунку це веб-інтерфейс, через який користувач може переглядати пропозиції продуктів, знаходити найближчі точки видачі, додавати нові позиції або переглядати історію взаємодій. У представлення не входить логіка обробки даних, що сприяє кращому розмежуванню обов'язків.
- Controller (Контролер) – слугує посередником між View і Model, обробляє запити користувача, викликає відповідні методи моделі та повертає оновлене представлення. Наприклад, при пошуку харчових продуктів за геолокацією, контролер отримує параметри пошуку, звертається до моделі для обробки запиту, а потім повертає користувачу результати у вигляді зручного інтерфейсу.

Використання архітектурної моделі MVC дозволяє забезпечити:

- масштабованість системи – окремі компоненти можна змінювати або розширювати незалежно один від одного;
- зручність підтримки – зміни у логіці обробки даних не потребують втручання у відображення або навпаки;
- гнучкість у розробці – різні команди можуть працювати над окремими частинами застосунку (база даних, бізнес-логіка, інтерфейс);
- високу швидкість розробки та тестування – за рахунок модульної побудови системи.

У контексті розробки програмного забезпечення для реалізації харчових надлишків архітектура MVC є обґрунтованим вибором, оскільки дозволяє створити масштабовану, зручну у використанні та гнучку систему, яка може адаптуватися до змін у бізнес-процесах чи нормативно-правовій базі.

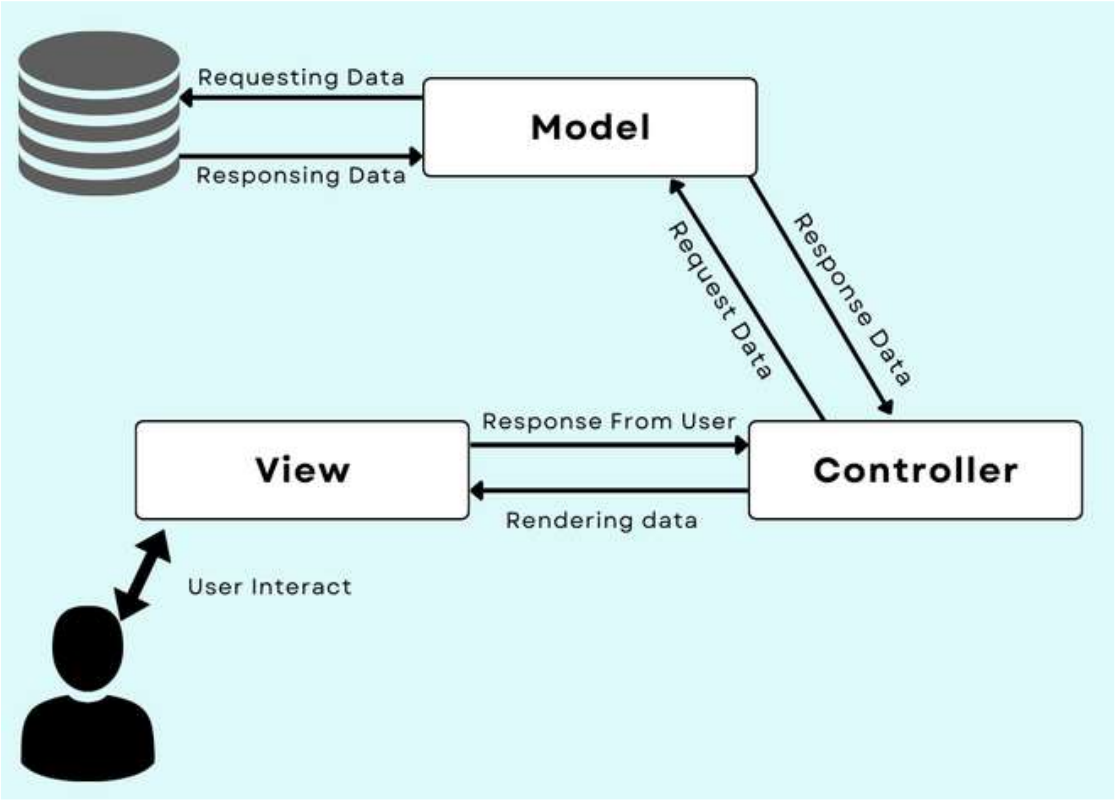


Рисунок 1.1 – Архітектура системи MVC

Таким чином, аналіз сучасного стану питання показує, що хоча вже існують рішення для зменшення харчових відходів, більшість із них не мають відкритої архітектури, гнучкості та повного спектра функціональності для локального впровадження. Застосування підходу MVC у нових рішеннях дозволяє побудувати ефективну, адаптивну та стійку програмну систему для оптимізації процесу реалізації харчових продуктів.

1.2. Огляд наявних аналогів

На сучасному етапі існує кілька популярних цифрових рішень, спрямованих на зменшення харчових відходів та ефективну реалізацію харчових надлишків. Нижче представлено аналіз чотирьох додатків, які мають різну географію застосування, функціональність та цільову аудиторію.

Too Good To Go (рис. 1.1) – це мобільний додаток, створений у Данії, який дозволяє користувачам купувати непродані продукти у ресторанів, супермаркетів та пекарень за зниженою ціною. Основна ідея полягає у тому,

щоб зменшити кількість харчових відходів, продаючи «сюрприз-бокси» з їжею, термін реалізації якої добігає кінця.

Переваги:

- Вдало реалізована бізнес-модель, яка об’єднує вигоду користувача (дешева їжа) та закладу (реалізація залишків).
- Простий та інтуїтивно зрозумілий інтерфейс.
- Активна підтримка з боку локальних закладів, що розширює асортимент пропозицій.
- Гейміфікація процесу – користувач не знає, що саме отримає, що створює елемент зацікавленості.

Недоліки:

- Відсутня можливість фільтрації за алергенами, дієтами або точним складом.
- Слабкий соціальний компонент – додаток більше спрямований на прибуткову реалізацію, ніж на благодійність.
- Немає карти з точним відображенням усіх локацій на загальному огляді.



Рисунок 1.1 - Too Good To Go

OLIO (рис. 1.2) – це британський мобільний застосунок, основна мета якого полягає в обміні їжею між сусідами, а також передачі харчів від закладів

харчування. Платформа дозволяє користувачам безкоштовно ділитися продуктами, які вони не встигають спожити.

Переваги:

- Чіткий соціальний фокус: платформа заохочує до безоплатного обміну та волонтерства.
- Широкі можливості фільтрації: категорії, продукти, стан, дата.
- Активна спільнота, що сприяє сталому розвитку локальних ініціатив.
- Можливість ділитися не лише їжею, а й побутовими товарами.

Недоліки:

- Менш приваблива для комерційних учасників, оскільки модель не передбачає прибутку.
- Залежність від активності користувачів у конкретному регіоні: у малонаселених районах функціональність знижується.
- Обмежена аналітика та персоналізація рекомендацій.



Рисунок 1.2 - OLIO

Food Rescue US (рис. 1.3) функціонує здебільшого у США й орієнтований на перерозподіл надлишкової їжі від ресторанів, магазинів та виробників до

благодійних організацій. У системі важливу роль відіграють волонтери, які здійснюють доставку харчових продуктів.

Переваги:

- Орієнтація на благодійність і допомогу малозабезпеченим верствам населення.
- Добре організована логістика з залученням волонтерів.
- Інтеграція з календарем для запланованих доставок та відстеження прогресу.
- Система рейтингу та підтвердження успішної передачі товарів.

Недоліки:

- Відсутність комерційного механізму для підприємств, що обмежує зацікавленість великого бізнесу.
- Складніша процедура реєстрації та залучення волонтерів у порівнянні з простішими платформами.
- Географічно обмежена присутність (переважно США).



Рисунок 1.3 - Food Rescue US

Хоча ShareTheMeal (рис. 1.4) технічно не займається реалізацією харчових залишків, а є платформою пожертв на їжу для людей, що голодують (під егідою ООН), вона заслуговує на увагу як аналогічний приклад використання IT-технологій у сфері продовольчої допомоги.

Переваги:

- Глобальна платформа з прозорими механізмами фінансування та великим охопленням.
- Емоційний компонент, що стимулює користувача робити пожертви.
- Високий рівень довіри завдяки співпраці з ООН (World Food Programme).
- Мінімалістичний інтерфейс, що дозволяє зробити пожертву у кілька кліків.

Недоліки:

- Відсутня функціональність для локального впливу чи взаємодії з продуктами у місті користувача.
- Не вирішує проблему локальних харчових залишків або перерозподілу їжі між підприємствами.
- Більше працює як фінансова платформа, ніж як інструмент логістики або оптимізації продуктового ланцюга.



Рисунок 1.4 - ShareTheMeal

Аналіз наведених аналогів демонструє, що кожен з розглянутих додатків вирішує проблему харчових залишків по-своєму, однак має свої обмеження. Too Good To Go орієнтований на комерційний прибуток і не вирішує соціальної складової. OLIO, навпаки, сильний у соціальному вимірі, але не мотивує бізнес. Food Rescue US має потужну логістичну складову, але слабку комерційну привабливість. ShareTheMeal діє на глобальному рівні, проте не зачіпає локального аспекту поводження з надлишками.

Відтак, існує потреба у створенні гібридного рішення, яке б поєднало:

- комерційну привабливість (як у Too Good To Go),
- локальну соціальну взаємодію (як у OLIO),
- волонтерську логістику (як у Food Rescue US),
- зручність архітектури й масштабованість (через MVC-шаблон), що дозволить ефективно вирішувати проблему реалізації харчових надлишків у різних регіонах.

1.3. Аналіз методів розв'язання та постановка задачі

Розробка програмного забезпечення, призначеного для оптимізації процесів реалізації харчових продуктів, потребує детального аналізу існуючих підходів до побудови подібних систем. Вибір оптимальної архітектурної моделі та платформи реалізації суттєво впливає на продуктивність, масштабованість і зручність користування майбутнім програмним продуктом. У даному підрозділі розглянуто найбільш поширені підходи до реалізації подібних систем, а також проаналізовано їх переваги та обмеження.

Одним із базових варіантів є створення консольного застосунку, що є технічно простим у реалізації. Його основними перевагами є мінімальні витрати часу на розробку, висока стабільність роботи, спрощене тестування функціональності, а також знижена ймовірність виникнення помилок під час експлуатації – завдяки обмеженому інтерфейсу користувача. Водночас подібне рішення є малопридатним для широкого кола користувачів, оскільки не

забезпечує належного рівня зручності, візуалізації даних та інтерактивності. Таким чином, консольні застосунки частіше використовуються у внутрішніх сервісах або технічних модулях систем.

Іншим варіантом є розробка мобільного застосунку, що орієнтований на користувачів смартфонів. Основною перевагою такого підходу є адаптивний інтерфейс, який забезпечує зручну взаємодію з системою у мобільному середовищі. Проте подібна архітектура має обмеження у вигляді відсутності доступу до функціоналу з персонального комп’ютера, що унеможливлює кросплатформне використання додатку без додаткових засобів.

Широко розповсюдженим сучасним підходом є розробка веб-застосунку з поділом на клієнтську (frontend) та серверну (backend) частини. Такий підхід дозволяє створити гнучку й масштабовану архітектуру, яка може обслуговувати значну кількість користувачів та адаптуватися до змін бізнес-логіки без необхідності переробки всього застосунку. Завдяки розділенню функціональних частин, можлива паралельна робота кількох команд розробників. Проте недоліками цього підходу є зростання складності розгортання та налаштування, а також збільшення термінів розробки через необхідність синхронізації обох частин системи.

Альтернативою вищезгаданому варіанту є веб-застосунок з інтегрованими клієнтською та серверною частинами, реалізований на основі єдиного фреймворку або платформи (наприклад, Vaadin, Meteor, тощо). Такий підхід дозволяє об’єднати переваги класичних веб-застосунків, зберігаючи при цьому простоту в розробці, тестуванні та підтримці. Поєднання фронтенду й бекенду в межах одного середовища суттєво знижує поріг входу, а також спрощує виявлення помилок і впровадження нової функціональності. Обмеженням такого методу може бути менша гнучкість порівняно з розділеними архітектурами в масштабних системах.

Ураховуючи переваги та недоліки розглянутих архітектурних підходів, у даній роботі було прийнято рішення реалізувати веб-застосунок з об’єднаними

клієнтською та серверною частинами. Такий підхід забезпечує баланс між функціональністю, зручністю використання, придатністю до тестування та ефективністю розробки.

На підставі аналізу предметної області, оцінки існуючих систем та порівняння альтернативних архітектур, було визначено основний функціонал майбутньої інформаційної системи:

- проєктування структури бази даних для зберігання інформації про продукти, користувачів та операції;
- реалізація серверної частини для обробки запитів, керування даними та бізнес-логіки;
- розробка графічного інтерфейсу користувача з урахуванням принципів адаптивності та UX-дизайну;
- інтеграція всіх компонентів в єдиний веб-застосунок з використанням сучасних програмних засобів і фреймворків.

Таким чином, обґрунтовано вибір архітектурного рішення та визначено ключові напрями розробки системи оптимізації реалізації харчових продуктів.

РОЗДІЛ 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ДОДАТКУ

2.1. Аналіз мов програмування та середовищ розробки для реалізації додатку

У процесі проєктування програмних модулів для інформаційної системи, орієнтованої на оптимізацію процесів реалізації харчових продуктів, надзвичайно важливим є обґрунтований вибір технологічного стеку. Обрана сукупність мов програмування, фреймворків, середовищ розробки та допоміжних бібліотек суттєво впливає на ефективність розробки, продуктивність системи, а також її масштабованість, підтримуваність і переносимість між платформами.

Традиційно, у мовах програмування компіляторного типу, таких як C або C++, вихідний код попередньо транслюється у виконуваний двійковий код (машинний код), який є низкою інструкцій у вигляді бітових послідовностей, що безпосередньо інтерпретуються апаратною архітектурою конкретного комп’ютера. Навіть якщо вихідний код C/C++ пишеться з урахуванням міжплатформеності – без використання специфічних розширень або залежностей від ОС, – згенерований виконуваний файл залишається залежним від архітектури, на якій був скомпільований. Таким чином, виконуваний файл, створений для Linux, не функціонуватиме в середовищі Windows чи macOS без перекомпіляції або застосування емуляторів.

На противагу цьому, мова Java передбачає компіляцію вихідного коду не в машинний код, а у байт-код – проміжну форму представлення програми, яка є незалежною від конкретної апаратної платформи та операційної системи. Запуск байт-коду здійснюється у віртуальному середовищі – Java Virtual Machine (JVM), адаптованому до конкретної платформи. Це дозволяє забезпечити високий рівень переносимості: один і той самий байт-код може виконуватись на будь-якій платформі, для якої існує сумісна JVM.

Для розробки цільової системи було обрано мову програмування Java, фреймворк Spring, інтегроване середовище розробки IntelliJ IDEA, а також фреймворк Vaadin для реалізації користувацького інтерфейсу.

Java – це об'єктно-орієнтована, високорівнева мова програмування, яка використовується у широкому спектрі застосунків – від десктопних до вбудованих систем, серверних додатків і веб-платформ. Синтаксис Java багато в чому наслідує C/C++, однак поєднує в собі більш високий рівень абстракції з розвиненою інфраструктурою виконання та підтримки багатоплатформеності.

Spring – це потужний фреймворк для створення корпоративних застосунків на Java, який забезпечує гнучке управління залежностями (через механізм інверсії контролю), підтримку транзакцій, обробку REST-запитів, безпеку доступу до ресурсів, а також інші критично важливі аспекти програмної архітектури. Однією з ключових переваг Spring є наявність великої кількості готових до використання модулів, які можна адаптувати до потреб конкретного проєкту, що значно скорочує загальний час розробки і зменшує кількість потенційних помилок у бізнес-логіці.

IntelliJ IDEA – це сучасне інтегроване середовище розробки, яке надає розробникам широкий спектр інструментів для ефективного написання, аналізу та налагодження коду. Воно підтримує розширення через плагіни, має потужні засоби автодоповнення коду, інспекції помилок у реальному часі, рефакторингу та інтеграції з системами керування версіями. Ці можливості суттєво прискорюють процес створення програмного забезпечення, особливо у великих проєктах.

Vaadin – це веб-фреймворк з відкритим вихідним кодом, що дозволяє створювати сучасні веб-інтерфейси безпосередньо на Java без необхідності використання JavaScript або HTML. Vaadin забезпечує глибоку інтеграцію з Java-стеком, включаючи Spring, і дозволяє реалізовувати реактивні та адаптивні інтерфейси з мінімальними витратами часу та ресурсів.

У межах реалізації системи передбачено створення низки функціональних модулів, зокрема:

- модуль автентифікації та реєстрації користувачів, який забезпечуватиме захищене зберігання облікових даних, перевірку прав доступу та обробку помилок при некоректному введенні;
- модуль геолокаційної обробки даних, який базуватиметься на використанні відкритого API Geocode для конвертації введених адрес у географічні координати та навпаки. Він забезпечить подальшу інтеграцію координат з іншими модулями системи;
- модуль управління публікаціями, що забезпечуватиме створення, редагування та пошук об’яв (постів), які відображають інформацію про пропозиції щодо обміну або продажу харчових продуктів. Пошук реалізовуватиметься за ключовими словами (назвою, адресою тощо).

Таким чином, використання Java у поєднанні з фреймворками Spring і Vaadin, а також середовищем IntelliJ IDEA, дає змогу створити масштабовану, надійну та зручну у використанні інформаційну систему. Успішна реалізація таких модулів дозволить автоматизувати й значно оптимізувати процес реалізації харчових продуктів, забезпечуючи нову якість взаємодії користувачів із платформою.

2.2. Розробка моделі системи та алгоритмів роботи додатку

З метою наочного представлення логіки функціонування основних програмних компонентів системи, що реалізує процес оптимізації реалізації харчових продуктів, було створено модель поведінки користувача у вигляді UML-діаграми діяльності.

На рисунку 2.1 відображено загальну послідовність дій, які виконує користувач під час взаємодії з системою. Згідно з цією моделлю, початковим етапом є процедура автентифікації, яка забезпечує захищений доступ до функціоналу платформи. Після успішного входу до системи користувач має

можливість ініціювати створення нової пропозиції щодо харчових продуктів або здійснити пошук серед наявних публікацій, відповідно до власних потреб.

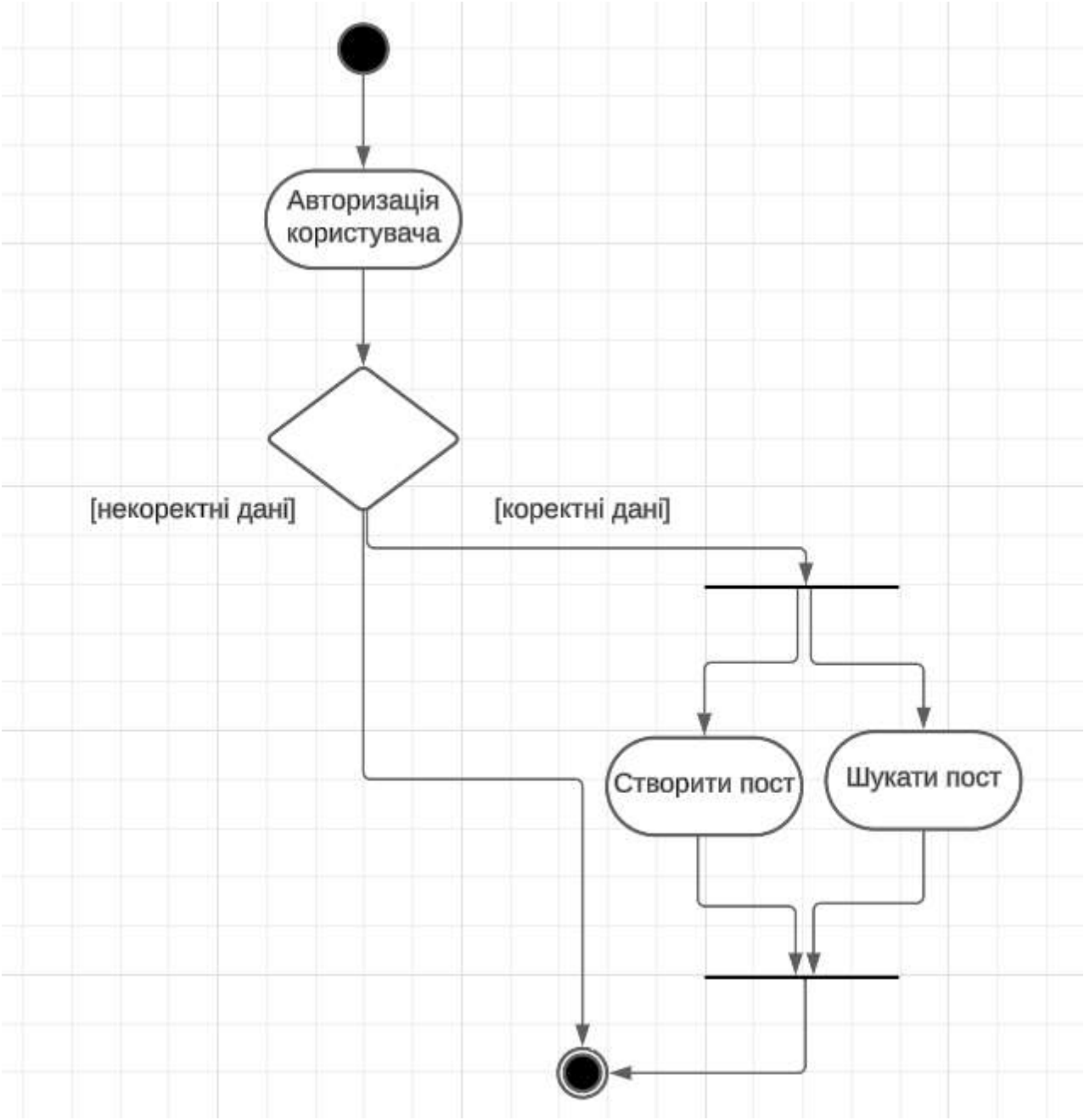


Рисунок 2.1 – UML-діаграма роботи системи

Для забезпечення ефективної реалізації платформи, призначеної для оптимізації процесу реалізації харчових продуктів, необхідно сформувати чіткий алгоритм функціонування веб-застосунку.

Структурна схема цього алгоритму представлена на рисунку 2.2.

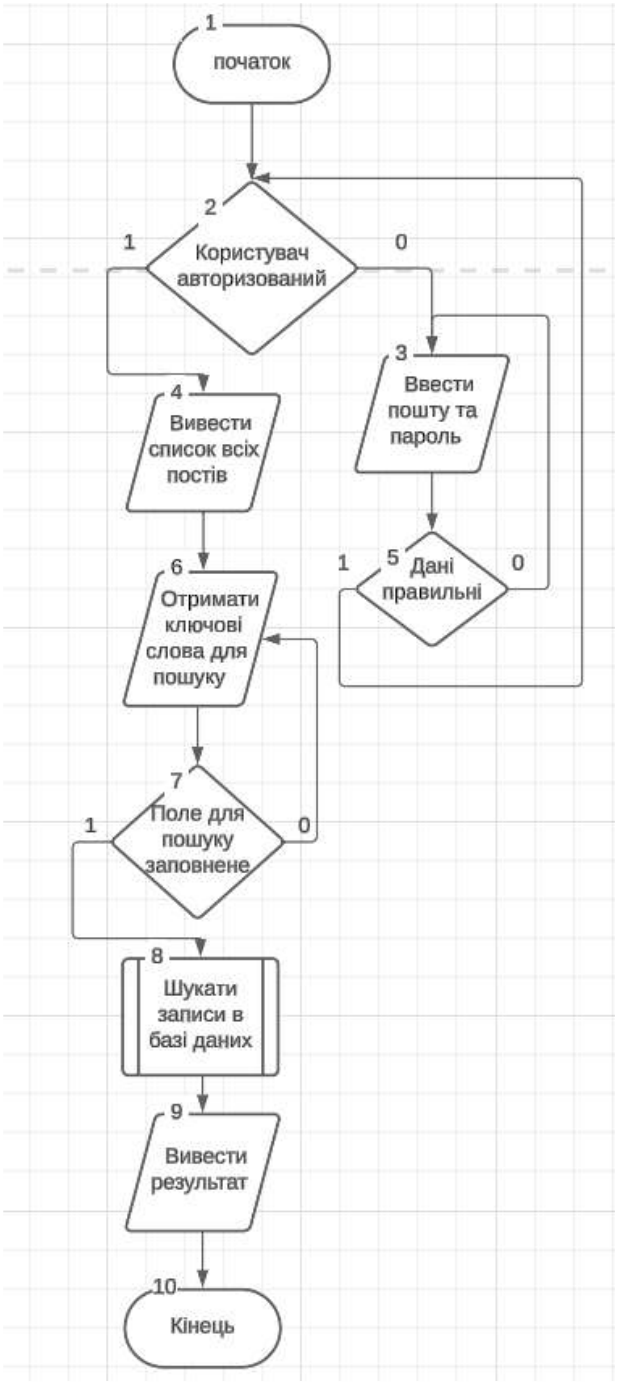


Рисунок 2.2 – Алгоритм функціонування веб-застосунку

Відповідно до поданого алгоритму, першим етапом є перевірка наявності авторизованого доступу користувача. У випадку, якщо користувач ще не пройшов процедуру авторизації, система ініціює запит на введення облікових даних. Якщо введені дані виявляються некоректними, веб-застосунок повторно запитує їх до моменту успішного проходження перевірки.

Після підтвердження достовірності введеної інформації та успішного входу в систему, користувач отримує доступ до загального переліку публікацій (постів) із

можливістю здійснити пошук за ключовими словами. Як тільки користувач вводить критерії пошуку, система ідентифікує запит та відображає лише ті пости, які відповідають заданим умовам. Ключовим елементом безпечної взаємодії з платформою є реалізація надійного алгоритму авторизації та реєстрації, що забезпечує захист даних користувача та стабільність роботи системи.

Схематичне представлення алгоритму наведено на рисунку 2.3.

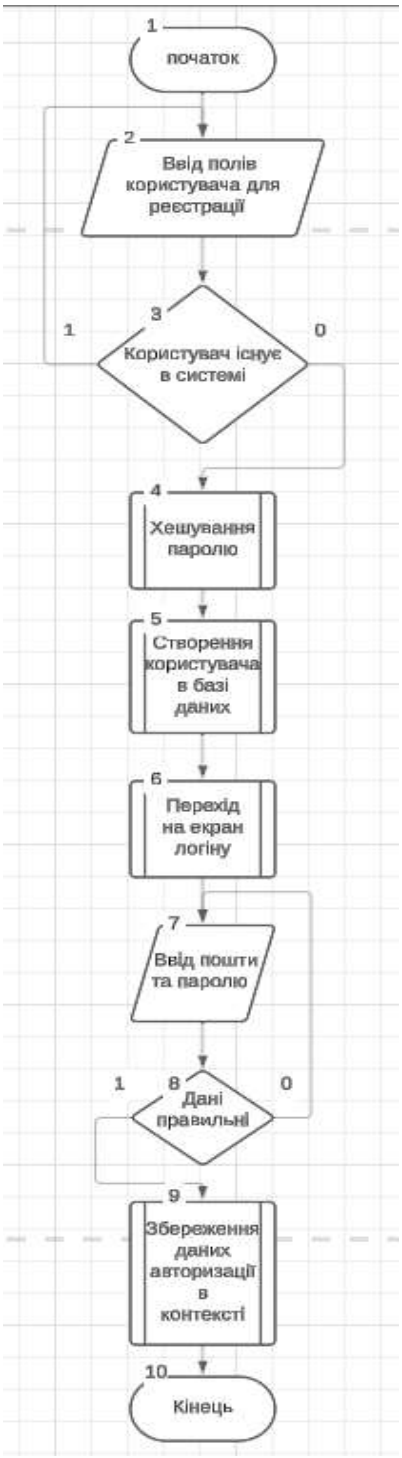


Рисунок 2.3 – Алгоритм автентифікації користувачів

Згідно з розробленим алгоритмом, первинним етапом взаємодії користувача із системою є введення реєстраційних даних у відповідні поля форми. Після отримання введених даних виконується перевірка їх на унікальність, зокрема здійснюється пошук наявності облікового запису з ідентичними параметрами у базі даних. У разі відсутності дублікатів система ініціює процедуру реєстрації, яка включає хешування паролю з метою забезпечення конфіденційності та подальше збереження даних користувача у системі зберігання. Завершення цього етапу супроводжується перенаправленням користувача на інтерфейс автентифікації, де очікується введення електронної пошти та паролю. У разі успішної верифікації облікових даних відповідна інформація про авторизованого користувача заноситься до контексту безпеки системи, що забезпечує ідентифікацію суб’єкта доступу під час подальших сеансів взаємодії.

Проведений аналіз демонструє, що реалізований веб-застосунок забезпечує як зручний пошук контенту, так і безпечну процедуру реєстрації та автентифікації користувачів. Запропоновані алгоритмічні рішення є ефективними й адаптованими до вимог сучасних інформаційних систем, що обумовлює їх доцільність і практичну застосовність у рамках даної програмної реалізації.

2.3. Розробка структури бази даних

Для забезпечення збереження, обробки та керування як користувацькими, так і системними даними, необхідним етапом проєктування інформаційної системи є формування реляційної структури бази даних. Саме ця структура виконує функцію централізованого сховища, в якому акумулюється інформація щодо географічного розташування об’єктів харчової інфраструктури, а також облікові дані зареєстрованих користувачів.

У процесі реалізації прикладного програмного забезпечення було інтегровано інструмент **Liquibase**, що є засобом управління версіями бази

даних з відкритим програмним кодом. Liquibase надає можливість декларативного опису змін у схемі бази даних, що значно спрощує процес супроводу, оновлення та синхронізації структурних змін у різних середовищах. На відміну від традиційного підходу, який передбачає написання SQL-скриптів вручну, використання Liquibase дає змогу формалізувати зміни у форматах XML, YAML або JSON, що підвищує читаємість конфігурацій, спрощує відстеження історії змін та забезпечує їх контрольованість за допомогою систем керування версіями [17].

Структурна схема функціонування механізму Liquibase представлена на рисунку 2.4.

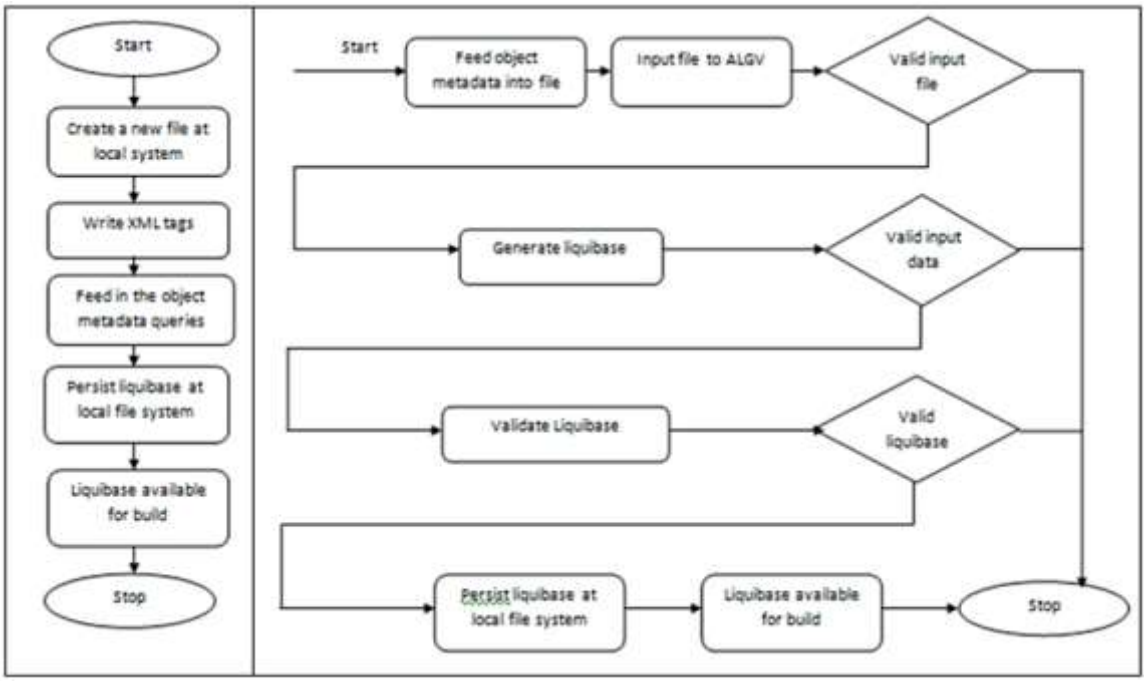


Рисунок 2.4 – Схема роботи liquibase.

Використання Liquibase дозволяє розробникам уникати типових проблем, пов’язаних із несумісністю версій бази даних та її структури під час виконання міграцій. Варто зазначити, що Liquibase не виконує автоматичної генерації таблиць, запитів чи коду – його основною функцією є керування змінами в схемі бази даних та забезпечення контролю за їхнім застосуванням.

Система відслідковує застосовані зміни за допомогою спеціалізованої службової таблиці, в яку вноситься інформація про кожен виконаний набір змін (change set). Це дозволяє уникнути повторного застосування одних і тих самих змін, а також виявити несанкціоновані модифікації вже відпрацьованих скриптів. Крім того, кожен change set може містити інструкції для скасування змін (rollback), що суттєво спрощує відновлення попереднього стану бази даних у разі помилок.

Типовий сценарій використання Liquibase передбачає створення SQL-файлу, в якому розробник прописує необхідні інструкції для створення об’єктів бази даних, наприклад, таблиці користувачів з відповідними полями. Приклад такої таблиці наведено на рисунку 2.5.

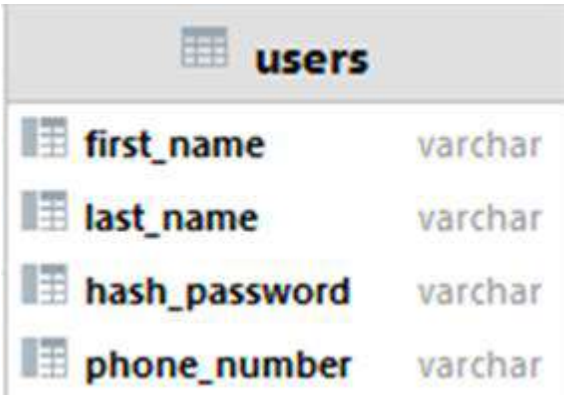


Рисунок 2.5 – Приклад таблиці користувачів.

Після первинного запуску застосунку в консольному виведенні з’являється повідомлення про успішне виконання скрипту та створення відповідної таблиці у базі даних. У результаті цього в системній таблиці, яку створює Liquibase для ведення обліку застосованих змін, з’являється новий запис. Ця службова таблиця містить детальну інформацію про кожен виконаний скрипт, зокрема назву відповідного файлу, дату його застосування, ім’я користувача, що ініціював виконання, а також інші метадані.

Одним із ключових атрибутів запису є контрольна сума (checksum), яка дозволяє визначити, чи зазнав скрипт змін після його початкового виконання.

Таким чином, контрольна сума забезпечує цілісність історії змін у структурі бази даних і гарантує відстежуваність модифікацій. Приклад вмісту системної таблиці Liquibase після виконання декількох скриптів наведено на рисунку 2.6.

	installed_rank	version	description	type	script
1	1	1	Create User Entity	SQL	V1__Create_User_Entity.sql
2	2	2	Create Credit Card And Address Entities	SQL	V2__Create_Credit_Card_And_Addres
3	3	3	Create Overview Entity	SQL	V3__Create_Overview_Entity.sql
4	4	4	Create Customer Organisation Entities	SQL	V4__Create_Customer_Organisation_
5	5	5	Create Post Entity	SQL	V5__Create_Post_Entity.sql
6	6	6	Create Order Entities	SQL	V6__Create_Order_Entities.sql
7	7	7	Create Email Entity	SQL	V7__Create_Email_Entity.sql
8	8	8	Create Review Report Notification Entities	SQL	V8__Create_Review_Report_Notifica
9	9	9	Create Organisation Many To Many Relations	SQL	V9__Create_Organisation_Many_To_M
10	10	10	Create Customer Many To Many Relations	SQL	V10__Create_Customer_Many_To_Many

Рисунок 2.6 – Приклад системної таблиці Liquibase.

Розглянемо типову ситуацію, коли виникає необхідність модифікації схеми бази даних, зокрема – додавання нового поля до існуючої таблиці користувачів. У такому випадку розробник вносить відповідні зміни до SQL-скрипту, після чого здійснюється повторний запуск програмного застосунку.

Під час ініціалізації Liquibase автоматично виконує перевірку всіх наявних файлів змін, зіставляючи їх контрольні суми з тими, що зберігаються у службовій таблиці відстеження змін. У разі виявлення нового файлу (тобто такого, який раніше не застосовувався), система безперешкодно виконує його та реєструє відповідну інформацію в системній таблиці. Однак, якщо виявляється, що раніше застосований файл було модифіковано, але його контрольна сума не збігається з початковою, Liquibase розцінює це як потенційне порушення цілісності історії змін.

У такій ситуації виконання подальших скриптів призупиняється, а в консолі з’являється повідомлення про виявлену помилку. Візуалізація типової помилки, яка виникає при порушенні контрольної суми скрипту, наведена на рисунку 2.7.

```
Migration checksum mismatch for migration version 1
-> Applied to database : -1887983629
-> Resolved locally    : -1315272917
Either revert the changes to the migration, or run repair to update the schema history.
Need more flexibility with validation rules? Learn more: https://rd.gt/3AbJUZE
```

Рисунок 2.7 – Помилка при порушенні контрольної суми скрипту.

Liquibase є ефективним інструментом підтримки розробника, зокрема в ситуаціях, коли під час написання SQL-запитів для створення або модифікації об’єктів бази даних виникають помилки. У разі виключної ситуації система автоматично зупиняє виконання скриптів, тим самим запобігаючи пошкодженню структури або вмісту бази даних. У процесі розробки програмного забезпечення часто виникає потреба в неочікуваних структурних змінах, які не були враховані на етапі первинного проєктування. Така гнучкість є нормальною практикою, і застосування Liquibase значно полегшує інтеграцію змін до схеми бази даних, знижуючи ризики критичних помилок. Для реалізації функціональності системи управління продуктами харчування доцільним є використання реляційної бази даних, що забезпечує структуроване зберігання інформації. Зокрема, таблиця users містить дані про зареєстрованих користувачів: контактну інформацію, роль у системі тощо. Основні поля таблиці включають:

- id – унікальний ідентифікатор;
- first_name, last_name – ім’я та прізвище;
- hash_password – хешований пароль;
- phone_number – номер телефону;
- user_role_id – ідентифікатор ролі;
- creation_date – дата реєстрації;
- email – електронна адреса.

Структура таблиці разом з типами даних подана на рисунку 2.8.

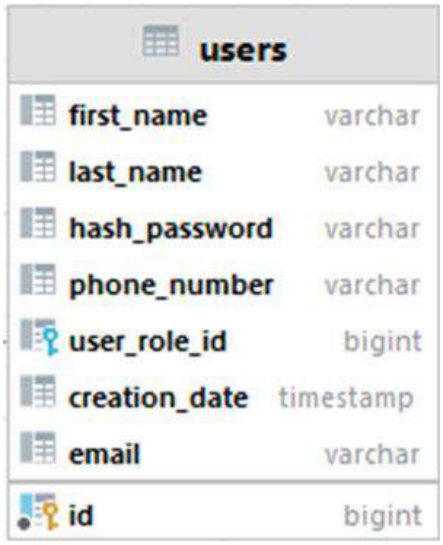


Рисунок 2.8 – Поля таблиці Users.

Таблиця user_roles слугує для зберігання інформації про типи ролей користувачів та їхні відповідні повноваження в межах системи. Вона містить такі основні поля:

- id – унікальний ідентифікатор ролі;
- role – назва ролі користувача.

Структура таблиці разом із відповідними типами даних представлена на рисунку 2.9.



Рисунок 2.9 – Поля таблиці User_roles.

Таблиця posts містить інформацію про пропозиції харчової продукції, яка використовується для їх подальшого пошуку та реалізації. Її структура включає такі ключові поля:

- id – унікальний ідентифікатор запису;
- user_id – ідентифікатор користувача, який створив пост;
- name – коротка назва пропозиції;

- description – текстовий опис продукції;
- photo_url – посилання на зображення, що ілюструє пост;
- creation_date – дата додавання запису;
- price – вартість одиниці продукції;
- amount – наявна кількість товару;
- location_id – ідентифікатор географічного розміщення пропозиції.

Структура таблиці з визначеними типами даних подана на рисунку 2.10.

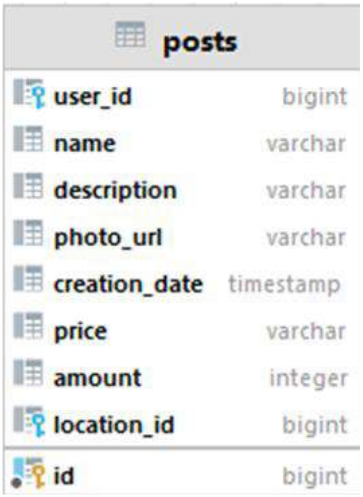


Рисунок 2.10 – Поля таблиці posts.

Таблиця locations призначена для зберігання інформації про географічне розташування харчових пропозицій, що забезпечує можливість їх просторового пошуку. Її структура включає такі поля:

- id – унікальний ідентифікатор місцезнаходження;
- country – назва країни;
- region – адміністративний регіон;
- city – населений пункт;
- street – назва вулиці;
- building – номер будинку;
- apartments – номер квартири;

- geom – просторові координати, що визначають геолокацію об’єкта.

Візуалізація структури таблиці з відповідними типами даних подана на рисунку 2.11.



Рисунок 2.11 – Поля таблиці locations.

Після поетапного проєктування кожної з таблиць та визначення логічних зв’язків між ними формується цілісна модель структури даних, яка відображає логіку збереження, доступу та обробки інформації в межах програмного застосунку, призначеного для оптимізації реалізації харчових продуктів. Така модель дозволяє ефективно організувати взаємодію між користувачами, їхніми пропозиціями, географічними координатами постачання продукції та відповідними ролями в системі.

Побудовані реляційні зв’язки забезпечують узгодженість даних, підвищують цілісність інформації та полегшують виконання складних запитів, необхідних для реалізації функціоналу застосунку. Комплексна схема бази даних із урахуванням усіх таблиць та встановлених між ними зв’язків наведена на рисунку 2.12.

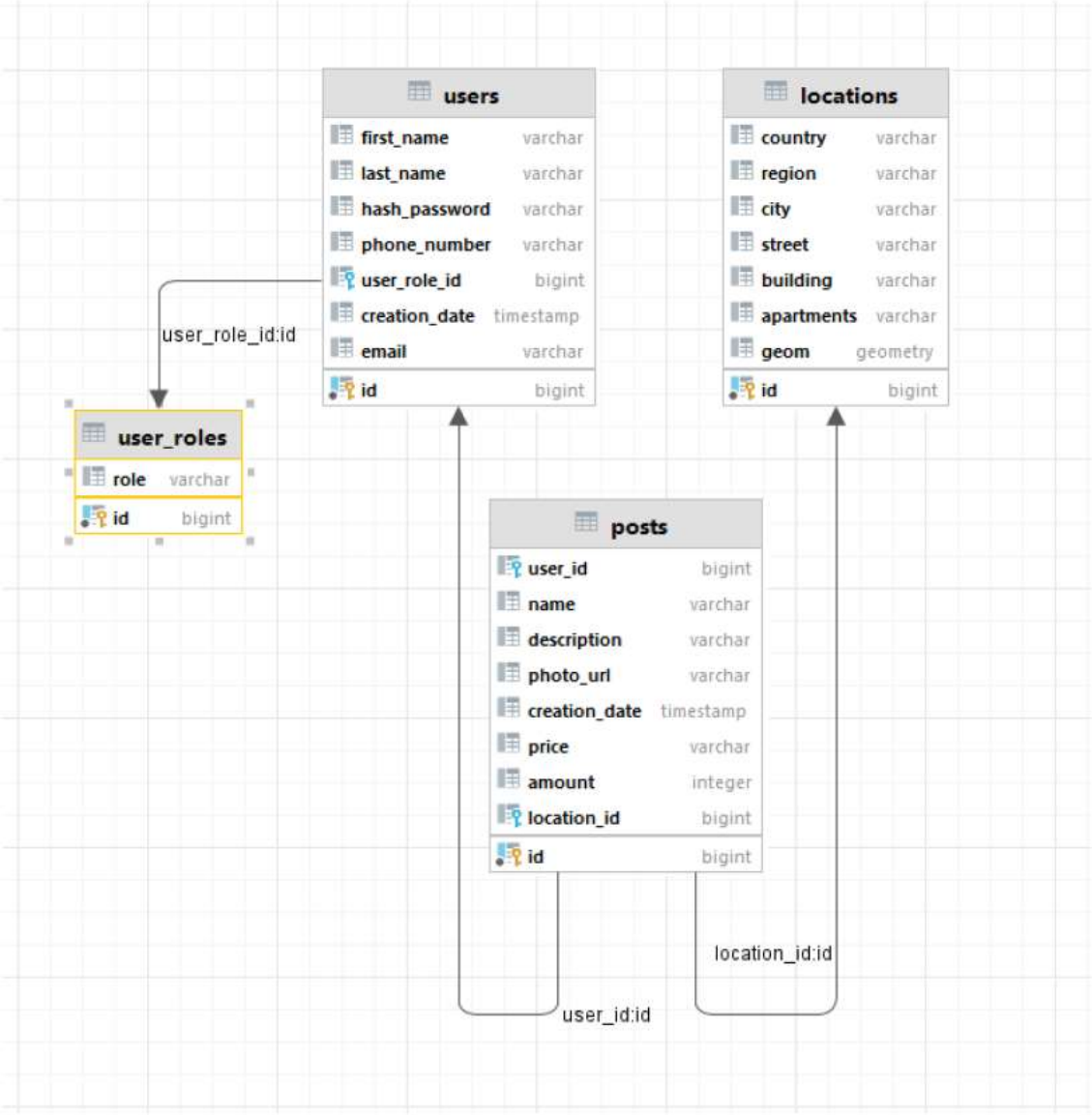


Рисунок 2.12 – Структура бази даних із урахуванням усіх таблиць

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ ОПТИМІЗАЦІЇ ЗБУТУ ХАРЧОВИХ ПРОДУКТІВ

3.1. Реалізація модуля реєстрації та авторизації користувача в системі

У сучасних умовах проблема забезпечення конфіденційності та безпеки персональних даних користувачів в мережі Інтернет набула особливої актуальності. У зв’язку з цим, розробка функціональних модулів, відповідальних за обробку та захист даних користувача, повинна здійснюватися з максимальною обачністю та дотриманням високих стандартів безпеки.

Потрапляння навіть частини облікових даних до рук зловмисників може спричинити не лише компрометацію окремого облікового запису, але й становити загрозу безпеці всієї інформаційної системи. Недостатній захист модуля автентифікації відкриває можливості для реалізації широкого спектра атак, серед яких – перехоплення сеансів, впровадження шкідливого коду, підміна ідентифікаційних даних тощо.

Коректно реалізований модуль автентифікації відіграє ключову роль у запобіганні несанкціонованому доступу до системи, перевантаженню серверів та іншим ускладненням, пов’язаним із помилками в процесі авторизації користувачів.

При розробці інтерфейсних контролерів з використанням фреймворку Vaadin рекомендується наслідувати базові класи компонування, зокрема VerticalLayout або HorizontalLayout [19]. У конструкторі відповідного класу здійснюється ініціалізація та конфігурація необхідних елементів графічного інтерфейсу, таких як кнопки, текстові поля, написи, зображення тощо. Для інтеграції логіки взаємодії з користувачем, необхідно реалізувати обробку подій, зокрема слухачів кнопок для реєстрації та входу.

У відповідь на подію натискання кнопки, обробник взаємодіє з сервісом автентифікації, передаючи йому дані, введені користувачем. Сервіс, у свою чергу, здійснює валідацію отриманої інформації, шифрує пароль із

застосуванням спеціалізованого модуля, обробляє дані та передає їх далі за логікою системи.

Шифрування паролів реалізується за допомогою вбудованої в Spring хеш-функції bcrypt. Алгоритм bcrypt розроблений із урахуванням стійкості до основних типів атак на паролі, зокрема атак грубою силою, з використанням райдужних таблиць і словників. Комбінація механізмів засолювання, розтягування ключа та підтримка паролів змінної довжини суттєво підвищує рівень захисту збережених облікових даних. Крім того, можливість налаштування складності обчислень забезпечує гнучкість у відповідності до конкретних вимог безпеки [20].

Далі сервіс автентифікації передає оброблену інформацію до користувацького сервісу, реалізуючи принцип інкапсуляції, що є фундаментальною складовою об’єктно-орієнтованого програмування. Також у структурі реалізації застосовано принцип поліморфізму: зокрема, метод list представлений із однаковою сигнатурою, але реалізує різну поведінку залежно від переданих аргументів. Ілюстрація цього механізму наведена на рисунку 3.1.

```
Page<User> list(Pageable pageable) {  1 usage  new *
    return userRepository.findAll(pageable);
}

Page<User> list(Pageable pageable, Specification<User> filter) {  1 usage
    return userRepository.findAll(filter, pageable);
}
```

Рисунок 3.1 – Сервіс автентифікації користувачів.

На наступному етапі обробки інформації сервіс користувача передає отримані дані до відповідного репозиторію з метою їх збереження у базі даних. Репозиторій виконує роль окремого рівня абстракції, що інкапсулює логіку взаємодії прикладного рівня із системою зберігання даних. У контексті архітектури Spring, репозиторії забезпечують уніфікований синтаксис доступу

до даних, незалежно від конкретного типу бази даних, що значно спрощує процес розробки та підтримки програмного забезпечення.

Ключовою перевагою використання репозиторіїв у Spring Framework є можливість автоматичної генерації реалізацій методів доступу до даних на основі імен методів, а також високий рівень кастомізації. Завдяки використанню анотацій та можливості розширення стандартних інтерфейсів, розробник може визначати специфічну поведінку методів відповідно до вимог бізнес-логіки системи.

Після успішного збереження даних у базі, відповідь формується у зворотному напрямку – результати обробки повертаються через сервісний рівень до контролера, який, у свою чергу, здійснює відображення відповідної інформації користувачеві.

Послідовність взаємодії компонентів під час процесу реєстрації та авторизації користувача ілюструється на діаграмі послідовності, поданій на рисунку 3.2.

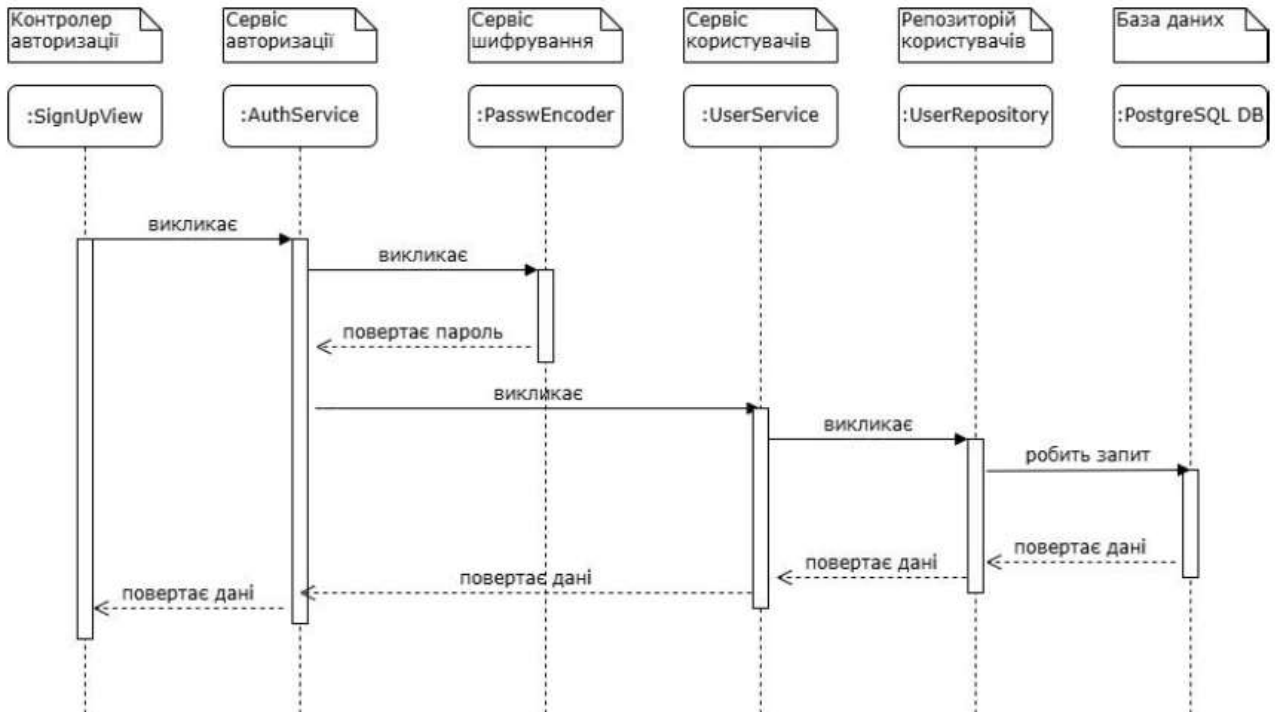


Рисунок 3.2 – UML-діаграма взаємодії компонентів під час процесу реєстрації та авторизації користувача.

3.2. Реалізація модуля пошуку харчових пропозицій

Функціональним ядром розробленого застосунку є реалізація механізму пошуку харчових пропозицій, які в межах системи репрезентовані у вигляді публікацій (постів). Запуск цього процесу ініціюється на рівні контролера, який наслідує базовий компонент VerticalLayout із фреймворку Vaadin, що забезпечує вертикальне компонування елементів інтерфейсу. На цьому етапі визначається структура контролера та маршрут доступу до нього через веб-браузер.

Однією з ключових функціональних складових контролера є метод configureGrid(), який відповідає за параметризацію відображення таблиці публікацій. Цей метод дозволяє задати розмір сітки, кількість та зміст комірок, визначаючи, які саме атрибути об'єктів будуть візуалізовані, що досягається шляхом передачі відповідних аргументів.

Подальший етап полягає у налаштуванні текстового поля для введення ключових слів з метою фільтрації публікацій за назвою або місцем розташування. Для зручності користувача передбачено плейсхолдер із текстовою підказкою щодо призначення поля. Крім того, активовано так званий "лінивий" режим обробки введення, який дозволяє зменшити навантаження на систему шляхом затримки реакції на зміну вмісту поля, замість миттєвого реагування на кожен введений символ.

Інформація, введена користувачем, передається до методу updateList(), який ініціює звернення до спеціалізованого сервісу обробки публікацій. Цей сервіс реалізує бізнес-логіку, пов’язану зі зберіганням, обробкою та пошуком публікацій.

З метою уточнення географічної прив’язки постів, сервіс публікацій делегує обробку адресного рядка окремому сервісу адрес, який реалізує взаємодію з відкритим API геокодування. Зокрема, здійснюється двостороннє перетворення: отримання координат за введеною адресою або, навпаки, визначення адреси за координатами. У відповідний метод передається текстовий опис місцезнаходження, введений користувачем.

Сервіс адрес формує HTTP-запит до стороннього API geocode.maps.co, передаючи туди адресний рядок і токен автентифікації користувача. У відповідь отримуються координати географічного об'єкта на мапі.

Отримані координати передаються назад до сервісу публікацій, звідки – до сервісу координат. Цей компонент забезпечує перетворення координат у формат, сумісний з вимогами бази даних.

На фінальному етапі агреговані та трансформовані дані передаються до репозиторію публікацій для виконання запиту пошуку постів за географічною близькістю. Завдяки використанню розширення PostGIS для PostgreSQL, пошук реалізується безпосередньо на рівні бази даних, із використанням просторових типів даних і функцій, наданих цим розширенням. Репозиторій здійснює запит, отримує релевантні результати та повертає їх сервісу публікацій.

На завершальному етапі відсортовані за відстанню публікації повертаються до контролера, де вони виводяться у графічному інтерфейсі для ознайомлення користувача з релевантними харчовими пропозиціями.

Послідовність викликів, пов’язана з пошуком харчових пропозицій, представлена у вигляді діаграми послідовності на рисунку 3.3.

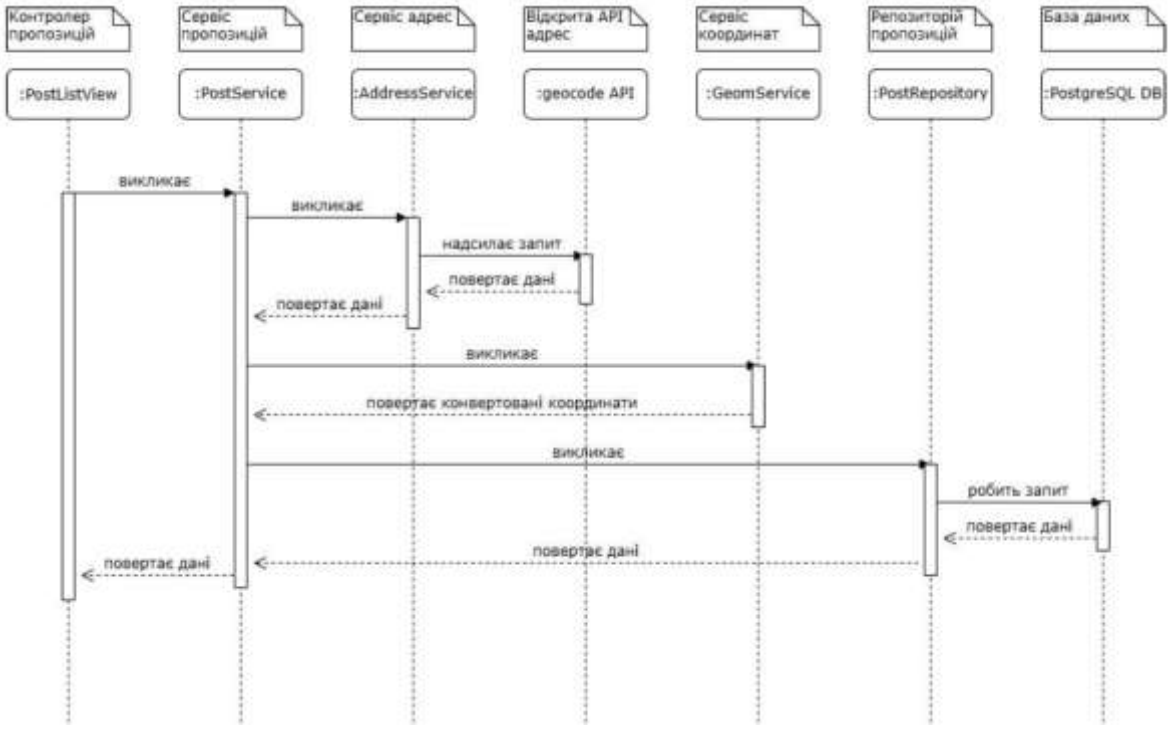


Рисунок 3.3 – UML-діаграма послідовності пошуком харчових пропозицій.

3.3. Реалізація інтерфейсу системи харчових пропозицій

Під час проєктування користувацького інтерфейсу системи пріоритетом є забезпечення його інтуїтивної зрозумілості та максимальної простоти взаємодії. Візуальна стилістика інтерфейсу ґрунтується на світлій колірній гамі з використанням спокійних, ненасичених тонів, що сприяє зосередженню уваги користувача на функціональній частині інтерфейсу, мінімізуючи вплив відволікаючих візуальних елементів.

Після переходу на веб-застосунок користувач потрапляє на стартову сторінку автентифікації, яка містить форму входу в систему. Інтерфейс логіну передбачає наявність полів для введення електронної пошти та пароля, а також текстову підказку з посиланням на форму реєстрації для нових користувачів, які ще не мають облікового запису.

Для підвищення надійності вводу реалізовано валідацію форми, яка передбачає обов’язкове заповнення всіх полів перед ініціацією процесу авторизації.

Візуальне представлення інтерфейсу форми входу наведено на рисунку 3.4.



Рисунок 3.4 – Інтерфейс форми входу

При натисканні на відповідне текстове посилання користувач перенаправляється на сторінку реєстрації, де йому пропонується заповнити всі необхідні поля для створення нового облікового запису. Інтерфейс реєстрації

також містить інструктивну підказку з можливістю переходу до форми входу, призначену для користувачів, які вже мають зареєстрований профіль у системі.

Візуальне відображення інтерфейс реєстрації наведено на рисунку 3.5.

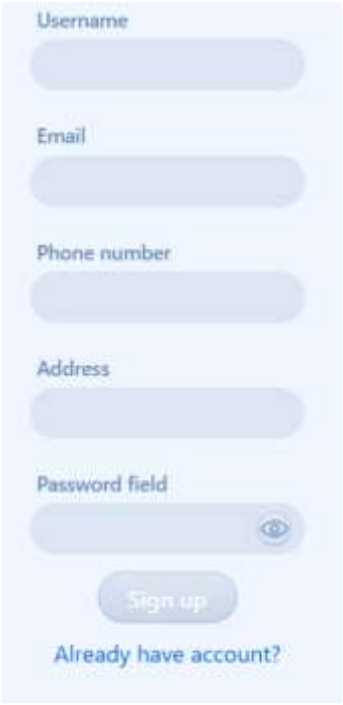


Рисунок 3.5 – Інтерфейс реєстрації

У формі реєстрації передбачено механізми валідації вхідних даних, що забезпечують контроль повноти заповнення всіх обов’язкових полів перед ініціацією процедури створення облікового запису. З метою підвищення безпеки персональних даних, зокрема при введенні паролю в умовах потенційного візуального доступу сторонніх осіб, символи у відповідному полі за замовчуванням маскуються. Для зручності користувача передбачено функціонал тимчасового відображення вмісту поля шляхом активації спеціальної кнопки, розташованої поряд із формою введення паролю.

Після успішної автентифікації користувач автоматично перенаправляється на головну сторінку застосунку, яка реалізує базову функціональність системи – пошук публікацій із пропозиціями щодо купівлі або обміну харчових ресурсів.

На даній сторінці розміщене поле для введення ключових слів, що дозволяє здійснювати пошук постів за назвою або географічною адресою.

Повне введення текстового запиту не є обов’язковим: система підтримує часткове співставлення введених даних із записами в базі. Крім того, реалізовано механізм сортування результатів за такими атрибутами публікацій, як назва, опис, місцезнаходження та вартість.

Візуальне представлення інтерфейсу пошуку публікацій наведено на рисунку 3.6.



Рисунок 3.6 – Інтерфейс пошуку публікацій

Однією з ключових функціональних можливостей системи є створення нових публікацій (постів) користувачами. З цією метою реалізовано окремий інтерфейс, який забезпечує зручний механізм введення усіх необхідних даних для формування нового запису. Структура екрану містить стандартизований набір форм для введення релевантної інформації, що гарантує коректне збереження об’єкта в системі.

Після завершення процесу створення посту та підтвердження введених даних, користувач автоматично перенаправляється на головну сторінку пошуку публікацій, де має можливість переглянути створений запис або ознайомитися з іншими пропозиціями, що відповідають його запитам.

Інтерфейс створення повідомлень представлено на рисунку 3.7.

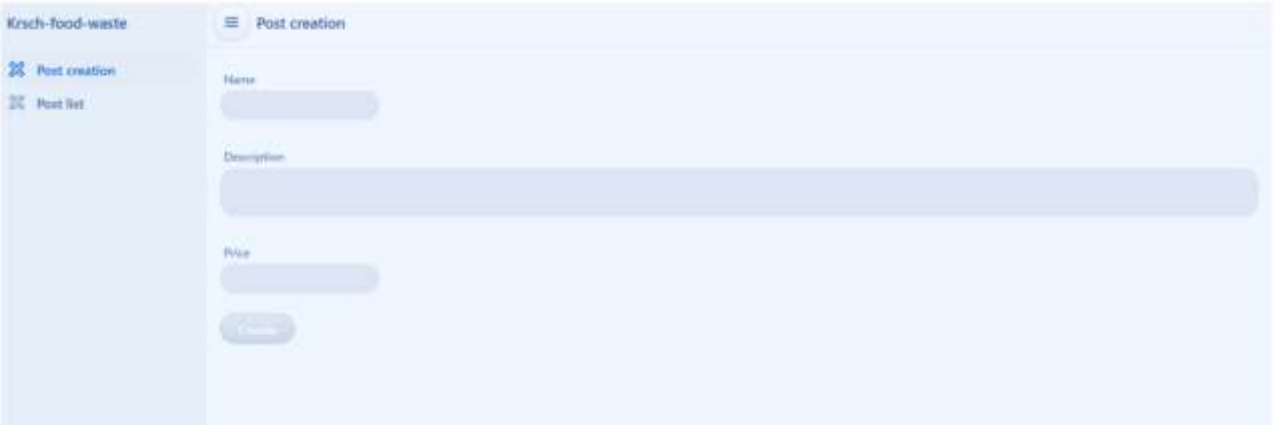


Рисунок 3.7 – Інтерфейс створення повідомлень

З метою підвищення зручності користувацької взаємодії з інтерфейсом системи було реалізовано бокову панель навігації, яка забезпечує швидкий доступ до основних функціональних розділів застосунку. Меню дозволяє здійснювати перемикання між ключовими екранами, а також містить опцію виходу з облікового запису. Ця функціональність забезпечує можливість повторної автентифікації – зокрема, для створення нового профілю або входу за допомогою вже існуючого облікового запису.

Візуальне представлення навігаційної панелі подано на рисунку 3.8.



Рисунок 3.8 – Візуальне представлення навігаційної панелі

Проєктування графічного інтерфейсу веб-платформи здійснювалося в інтегрованому середовищі розробки IntelliJ IDEA із застосуванням фреймворку Vaadin, що забезпечує побудову динамічних вебінтерфейсів на основі Java. У межах реалізації функціональних компонентів було створено програмні модулі для авторизації користувачів та пошуку харчових пропозицій, які є складовими частинами системи, орієнтованої на підвищення ефективності обігу харчових

продуктів. Також було розроблено повноцінний користувацький інтерфейс, що забезпечує взаємодію з основними сервісами платформи.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра було здійснено розробку веб-застосунку, спрямованого на оптимізацію процесів обміну та реалізації харчових продуктів шляхом інтеграції клієнтської та серверної логіки в єдиному програмному середовищі. На початковому етапі було проведено порівняльний аналіз архітектурних рішень для побудови веб-систем, зокрема розглянуто традиційні підходи з розділеним фронтендом і бекендом, а також альтернативні рішення, що передбачають об’єднання обох компонентів у межах одного фреймворку. У результаті аналітичного етапу було обґрунтовано доцільність використання монолітної архітектури на основі Vaadin – сучасного Java-фреймворку, який забезпечує гнучку побудову інтерфейсів та ефективне керування серверною логікою без необхідності розділення застосунку на окремі технологічні шари.

Обраний архітектурний підхід дозволив суттєво знизити загальну складність розробки, спростити процес інтеграції компонентів та зменшити ймовірність помилок, які зазвичай виникають на стику між клієнтською та серверною частинами. Крім того, уніфікація середовища розробки позитивно вплинула на швидкість реалізації та подальшу підтримку застосунку, що є особливо актуальним для невеликих та середніх за масштабом систем. Незважаючи на певну обмеженість у масштабуванні такого підходу порівняно з мікросервісною або клієнт-серверною архітектурою, отриманий результат виявився цілком придатним для вирішення задач, поставлених у рамках цього проєкту.

У процесі реалізації було визначено ключову функціональність інформаційної системи, що охоплює створення і пошук харчових пропозицій (постів), авторизацію користувачів, збереження даних у базі, а також інтеракцію з геолокаційними сервісами для визначення місця розташування постів. Було спроєктовано структуру бази даних із дотриманням принципів нормалізації, що дозволило забезпечити цілісність і консистентність

інформації. Серверна частина системи реалізована з урахуванням модульності та підтримки основних принципів об'єктно-орієнтованого програмування, що гарантує масштабованість логіки застосунку в майбутньому.

Графічний інтерфейс користувача було побудовано відповідно до принципів UX-дизайну з акцентом на простоту, інтуїтивну зрозумілість і адаптивність. Було враховано вимоги до валідації введених даних, безпеки паролів, зручності навігації та швидкості доступу до основних функцій. Усі компоненти системи інтегруються в межах єдиного застосунку, що забезпечує плавний користувацький досвід і дозволяє ефективно працювати з даними в режимі реального часу.

У підсумку, результати дипломної роботи підтверджують доцільність використання фреймворків із єдиною архітектурною моделлю для побудови спеціалізованих веб-застосунків. Розроблена система демонструє приклад успішної реалізації такого підходу, поєднуючи функціональність, ефективність, зручність використання та придатність до подальшого вдосконалення. Отримані напрацювання можуть бути використані для розширення функціоналу системи, інтеграції з іншими сервісами або масштабування на рівень регіональної чи національної платформи з обліку та розподілу харчових ресурсів.

СПИСОК ЛІТЕРАТУРИ

1. Food Waste. Харчові відходи [Електронний ресурс] – Режим доступу до ресурсу: https://slowfood.in.ua/global-projects/food-waste/
2. Харчові відходи – один з найбільших викликів сьогодення [Електронний ресурс] – Режим доступу до ресурсу: https://sd4ua.org/harchovi-vidhody-odyn-z-najbilshyh-vyklykiv-sogodennya/
3. Food Waste | The Nutrition Source [Електронний ресурс] – Режим доступу до ресурсу: https://www.hsph.harvard.edu/nutritionsource/sustainability/food-waste/
4. Головачьов М.О. Investigation Of PostgreSQL Extensions For Work With Coordinates Of Objects On The Map // Матеріали конференції «Стан, досягнення та перспективи інформаційних систем і технологій» (2024), Одеса, Видавництво ОНТУ, 2024 р. – 498 с.
5. MVC Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/mvc-design-pattern/
6. ResQ Club – Leave no meal behind [Електронний ресурс] – Режим доступу до ресурсу: https://www.resq-club.com/
7. Too Good To Go: Save Food - Help The Planet [Електронний ресурс] – Режим доступу до ресурсу: https://www.toogoodtogo.com/
8. Olio - Your Local Sharing App [Електронний ресурс] – Режим доступу до ресурсу: https://olioapp.com/en/
9. foodsharing | Rette mit! [Електронний ресурс] – Режим доступу до ресурсу: https://foodsharing.de/
10. Karma App - Save food with a tap [Електронний ресурс] – Режим доступу до ресурсу: https://save.karma.life/
11. Client-Server Architecture - Advantages and Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://kitrum.com/blog/client-server-architecture-advantages-and-disadvantages/

12. What is MVC? Advantages and Disadvantages of MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/
13. What Is Java? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/java
14. Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-framework
15. Intellij Idea - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/intellij_idea/intellij_idea_introduction.htm
16. Get Started with Vaadin Flow [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@rejaluo/get-started-with-vaadin-flow-76026dd9e902
17. What is Liquibase: A Comprehensive Guide | by YD [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/nerd-for-tech/what-is-liquibase-a-comprehensive-guide-db01807235d3
18. Beginning Java Objects: From Concepts to Code, 3rd edition навч. посіб. / Beginning Java Objects.2023 – 22 ст.
19. Vaadin Flow Quick Start [Електронний ресурс] – Режим доступу до ресурсу: https://vaadin.com/docs/latest/guide/quick-start
20. What is bcrypt and how does it work? | by ValeVPN [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/@valevpn/what-is-bcrypt-and-how-does-it-work-bef43ee8762d
21. Guide to Web Application Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/web-application-testing
22. Bloch J. Effective Java. – 3rd ed. – Boston : Addison-Wesley, 2018. – 416 с.
23. Sierra K., Bates B. Head First Java. – 3rd ed. – Sebastopol : O'Reilly Media, 2022. – 720 с.

24. Horstmann C. Core Java. Volume I–Fundamentals. – 12th ed. – Upper Saddle River : Prentice Hall, 2022. – 928 с.
25. Eck D. Introduction to Programming Using Java [Електронний ресурс]. – Режим доступу: https://math.hws.edu/javanotes – Дата звернення: 10.05.2025.
26. Liguori D. Java Pocket Guide. – 4th ed. – Sebastopol : O'Reilly Media, 2017. – 210 с.
27. Goetz B. Java Concurrency in Practice. – Boston : Addison-Wesley, 2006. – 384 с.
28. Cornell G., Horstmann C. Core Java 2: Advanced Features. – 7th ed. – Prentice Hall, 2005. – 1056 с.
29. Deitel P., Deitel H. Java: How to Program. – 11th ed. – Pearson, 2018. – 1248 с.
30. Schildt H. Java: The Complete Reference. – 12th ed. – New York : McGraw-Hill, 2021. – 1248 с.
31. Downey A. Think Java: How to Think Like a Computer Scientist. – 2nd ed. – Sebastopol : O'Reilly Media, 2019. – 252 с.
32. Johnson R., Hoeller J. Professional Java Development with the Spring Framework. – Indianapolis : Wrox, 2005. – 768 с.
33. Walls C. Spring in Action. – 6th ed. – Shelter Island : Manning Publications, 2022. – 520 с.
34. Mak D. Spring Boot in Practice. – Shelter Island : Manning Publications, 2021. – 400 с.
35. Gutierrez R. Learning Spring Boot 3.0. – 4th ed. – Birmingham : Packt Publishing, 2022. – 372 с.
36. Long J. Reactive Spring. – Sebastopol : O'Reilly Media, 2020. – 340 с.
37. Turnquist G. Learning Spring Boot 2.0. – Birmingham : Packt Publishing, 2017. – 370 с.

38. JetBrains. IntelliJ IDEA Documentation [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/idea/documentation – Дата звернення: 10.05.2025.
39. Vasilev D. IntelliJ IDEA Essentials. – Birmingham : Packt Publishing, 2015. – 180 с.
40. Yeremeyev V. Getting Started with IntelliJ IDEA. – Independently published, 2020. – 156 с.
41. DiMarzio J. Android Studio 4.2 Development Essentials. – 2021. – 765 с.
42. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. – 7th ed. – New York : McGraw-Hill, 2019. – 1376 с.
43. Elmasri R., Navathe S. Fundamentals of Database Systems. – 7th ed. – Pearson, 2016. – 1232 с.
44. Date C. J. An Introduction to Database Systems. – 8th ed. – Boston : Addison-Wesley, 2004. – 1024 с.
45. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs – Дата звернення: 10.05.2025.

ДОДАТКИ