НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ Голуб Б.Л.
(підпис) (ПІБ)

"__" _____ 20__ р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмне забезпечення для c2c-торгівлі вторинною сировиною»
Спеціальність 121 – «Інженерія програмного забезпечення»
ОПП «Інженерія програмного забезпечення»

Гарант освітньої програми

К.т.н., доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Лендєл Т.І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Подунай А.О.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ Голуб Б.Л.
(підпис) (ПІБ)

"16" грудня 2024 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Подунай Андрію Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення для с2с-торгівлі вторинною сировиною.
затверджена наказом ректора НУБіП України від "16" грудня 2024 р.
№ 2248"С"
Термін подання завершеної роботи на кафедру 2025.05.25
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи: опис організації та змісту роботи амбулаторно-поліклінічних закладів

Перелік питань, які потрібно розробити:

1. Районна поліклініка як об'єкт дослідження
2. Проєктування інформаційного забезпечення
3. Розробка програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "16" грудня 2024 р.

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Лендєл Т.І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Подунай А.О.

_

(підпис) (ПІБ студента)

АНОТАЦІЯ

У цій бакалаврській роботі представлено розробку C2C вебплатформи, призначеної для полегшення купівлі та продажу вторинної сировини безпосередньо між користувачами. Головною метою системи є створення зручного та ефективного онлайн-простору, що сприяє повторному використанню ресурсів та зменшенню відходів.

Платформу розроблено з використанням JavaScript-бібліотеки React JS для створення інтерактивного користувацького інтерфейсу, фреймворку Fastify на базі Node.js для забезпечення серверної логіки та NoSQL бази даних MongoDB для гнучкого зберігання даних про користувачів та оголошення. Для автентифікації користувачів застосовано Auth0-verify, а для зберігання зображень оголошень використано хмарний сервіс AWS S3.

Дипломна робота охоплює аналіз існуючих рішень у сфері C2C торгівлі та вторинної сировини, проєктування архітектури платформи, розробку ключових функцій, таких як реєстрація користувачів, розміщення та перегляд оголошень з можливістю пошуку за різними критеріями. Отриманий вебзастосунок демонструє можливості сучасних веб-технологій для створення платформи, що сприяє екологічно свідомому споживанню та повторному використанню матеріалів.

ABSTRACT

This bachelor's thesis presents the development of a C2C web platform designed to facilitate the buying and selling of secondary raw materials directly between users. The main goal of the system is to create a convenient and efficient online space that promotes the reuse of resources and waste reduction.

The platform is developed using the React JS JavaScript library for building an interactive user interface, the Fastify framework based on Node.js for providing server-side logic, and the MongoDB NoSQL database for flexible storage of user and listing data. Auth0-verify is used for user authentication, and the AWS S3 cloud service is used for storing listing images.

The thesis covers an analysis of existing solutions in the field of C2C commerce and secondary raw materials, the design of the platform architecture, the development of key features such as user registration, listing creation and viewing with search capabilities based on various criteria. The resulting web application demonstrates the capabilities of modern web technologies to create a platform that promotes environmentally conscious consumption and material reuse.

ЗМІСТ

ВСТУП

У сучасних реаліях глобального розвитку людства та загострення екологічних проблем, питання забезпечення екологічної безпеки, ефективного використання

природних ресурсів та скорочення обсягів твердих і рідких відходів постають надзвичайно актуальними. Одним із перспективних напрямів вирішення зазначених проблем є активне впровадження концепції повторного використання матеріалів, що вже були у вжитку. У цьому контексті особливого значення набуває розвиток C2C-платформ (customer-to-customer), які забезпечують зручний функціонал для безпосередньої взаємодії між приватними особами з метою купівлі-продажу вторинної сировини без залучення посередників чи сторонніх організацій.

У зв’язку з посиленням екологічної свідомості серед населення та підвищенням рівня зацікавленості в зменшенні витрат на нову продукцію, зростає потреба у цифрових сервісах, які б максимально спрощували обмін використаними матеріалами. Проте поточний стан ринку таких рішень свідчить про недостатній рівень розвитку. Наявні сервіси не завжди відповідають вимогам саме щодо роботи з вторинною сировиною, оскільки часто ігноруються аспекти сортування, особливості логістики, обліку обсягів і типів матеріалів, а також інші критично важливі функціональні можливості.

Створення інноваційної, інтуїтивно зрозумілої та функціонально багатої інформаційної системи, яка б сприяла ефективній комунікації між покупцями та продавцями в контексті вторсировини, є важливим кроком у напрямку оптимізації процесів торгівлі такими матеріалами. Запровадження подібної системи дозволить підвищити прозорість операцій, забезпечити їх контрольованість і надати додаткові інструменти моніторингу, що в сукупності

сприятиме зменшенню негативного впливу людської діяльності на навколишнє середовище.

Основне завдання автоматизації таких процесів полягає у створенні ефективної вебплатформи, здатної забезпечити швидкий, простий, зручний та безпечний обмін між зацікавленими сторонами. Така система повинна надавати змогу користувачам реєструватися, розміщувати оголошення, переглядати наявні пропозиції, фільтрувати товари за різноманітними критеріями, а також вести комунікацію між учасниками платформи. Крім того, надзвичайно важливим є реалізація модулів аналітики, які допоможуть відстежувати активність та інші параметри використання системи.

Один із центральних елементів цього проєкту — застосування сучасних інструментів для моделювання, проектування та реалізації веборієнтованого програмного забезпечення, що функціонує у зв’язці з базою даних. Така інтеграція дозволяє якісно та структуровано зберігати відомості про користувачів, їх активність, транзакції, властивості товарів тощо, що є основою ефективної роботи всієї системи.

У зв’язку з вищезазначеним, головна мета цієї бакалаврської кваліфікаційної роботи — створити програмний продукт, який буде виступати C2C платформою для організації процесів купівлі-продажу вторсировини.

Для досягнення окресленої мети необхідно реалізувати низку послідовних завдань:

- здійснити поглиблений аналіз предметної області, визначити специфіку ринку та узагальнити ключові вимоги до інформаційної системи;
- дослідити існуючі на ринку рішення, оцінити їх функціональні можливості та виявити недоліки;
- сформулювати як функціональні, так і нефункціональні вимоги до майбутньої вебплатформи;
- побудувати концептуальні моделі бізнес-процесів за допомогою сучасних UML-діаграм (діаграми варіантів використання, послідовності, активності тощо);
- розробити архітектуру майбутньої системи, передбачити загальну логіку її функціонування;
- обрати відповідні інструменти та технології, що забезпечать надійність та ефективність роботи платформи;
- створити зручний, інтуїтивно зрозумілий інтерфейс користувача;

- спроєктувати і впровадити базу даних з урахуванням масштабованості та безпеки;
- реалізувати вебдодаток відповідно до обраної архітектури;
- провести комплексне тестування функціональних можливостей і зручності інтерфейсу користувача.

У підсумку очікується створення ефективного інструменту — вебплатформи, яка сприятиме популяризації обміну вторинною сировиною, підвищенню екологічної свідомості суспільства та інтеграції цифрових екологічних ініціатив у повсякденне життя.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛСТІ

1.1 Постановка задачі

Програмна система, що розробляється, має на меті забезпечити ефективну та зручну взаємодію між фізичними особами, які мають потребу або бажання продати чи купити вторинну сировину. Така система орієнтована на користувачів, які прагнуть до сталого розвитку, піклуються про довкілля та бажають зробити внесок у повторне використання ресурсів.

Основні завдання, які має виконувати вебплатформа:

- забезпечити просту та швидку публікацію оголошень про продаж чи купівлю різних видів вторинної сировини;
- реалізувати функціонал реєстрації та авторизації користувачів із захистом персональних даних;
- надати можливість створення та управління оголошеннями, редагування та видалення інформації про товари;
- сформувати повноцінну базу даних, у якій зберігатимуться відомості про користувачів, товари, замовлення, історію дій;
- забезпечити стабільну та захищену роботу платформи, яка легко масштабується залежно від навантаження.

З огляду на тенденції цифровізації торгівлі та зростання значення екологічних практик у житті суспільства, створення подібного ресурсу є не лише доцільним, а й стратегічно важливим для сталого розвитку.

Функціональні модулі платформи передбачають:

- створення, редагування та публікацію оголошень;
- перегляд товарів за допомогою інтерактивного каталогу;
- можливість збереження обраних товарів у список «Улюблене» або «Відстежуване»;
- пошук і фільтрацію за параметрами (категорія, ціна, місце розташування,

стан тощо);

- підтримку облікових записів користувачів з персональними кабінетами;
- адміністрування контенту через спеціалізовану панель управління.

Незважаючи на відсутність модулів онлайн-платежів та внутрішньої системи повідомлень, платформа надає базовий набір можливостей, що задовольняє основні потреби користувачів у сфері C2C-торгівлі вторсировиною. Інтерфейс системи максимально спрощений для зручності сприйняття та роботи, не вимагає спеціальних технічних знань і є інтуїтивно зрозумілим.

Уся інформація в системі зберігається у документно-орієнтованій базі даних MongoDB, що забезпечує оперативний доступ до даних, легке масштабування та можливість адаптації до змінних умов експлуатації.

Серед основних даних, що мають бути передбачені у системі:

- облікова інформація про користувачів (ПІБ, контактні дані, профільні налаштування);
- характеристика вторсировини (тип, категорія, опис, фізичний стан, вартість);
- історія взаємодій, опублікованих оголошень і замовлень.

Система повинна забезпечити:

- повноцінну реєстрацію та автентифікацію користувачів з можливістю відновлення пароля;
- додавання, редагування та видалення товарів з урахуванням бізнес-логіки; •

ефективні механізми фільтрації та пошуку за різними параметрами; •

належний рівень захисту персональних даних користувачів; • адаптивність

вебінтерфейсу до різних типів пристроїв (мобільні телефони, планшети, ПК);

- дружній, інтуїтивно зрозумілий користувацький досвід.

Загалом, платформа повинна поєднувати в собі простоту експлуатації, мінімальні вимоги до технічної підготовки користувача та високий рівень

доступності. Головною метою її створення є забезпечення комфортного

цифрового середовища для громадян, зацікавлених у сталому споживанні та повторному використанні ресурсів.

1.2 Моделювання предметної області

Процес створення якісного, масштабованого та функціонального програмного забезпечення потребує попереднього глибокого аналізу предметної області, у межах якої функціонуватиме майбутня система. У випадку розробки вебплатформи для C2C-торгівлі вторинною сировиною це моделювання стало основою для визначення ключових сутностей, сценаріїв використання, а також взаємозв’язків між елементами системи.

З метою точного опису структури, логіки функціонування, а також взаємодії між об’єктами програмного середовища було застосовано сучасні підходи об’єктно-орієнтованого аналізу, зокрема візуалізацію за допомогою уніфікованої мови моделювання UML (Unified Modeling Language). Цей підхід дає змогу створити формалізовану модель, яка слугує одночасно технічною специфікацією для розробників і засобом комунікації між усіма учасниками проєкту.

Діаграма варіантів використання (Use Case Diagram)

Одним із ключових етапів стало створення діаграми варіантів використання (див. рис. 1.1), яка є своєрідним «каркасом» взаємодії користувачів із системою. У цій діаграмі виокремлено основні ролі — **Користувач** та **Адміністратор**. Користувач має можливість здійснювати базові операції, такі як створення нових оголошень про продаж сировини, їх редагування, видалення, а також перегляд усіх наявних пропозицій. Адміністратор, своєю чергою, несе відповідальність за перевірку, модерацію та затвердження оголошень, забезпечуючи дотримання політик платформи.

Таке моделювання дозволяє не лише структурувати функціонал за ролями, а й виявити потенційні області розширення функціоналу або ризики, пов’язані з небажаними діями користувачів.

Діаграма варіантів використання відображає загальну картину взаємодії користувачів із системою (див. рис. 1.1). У ній окреслено основні ролі — користувач та адміністратор. користувач може створювати, редагувати й видаляти оголошення про продаж сировини, переглядати доступні пропозиції. Адміністратор відповідає за модерацію оголошень.

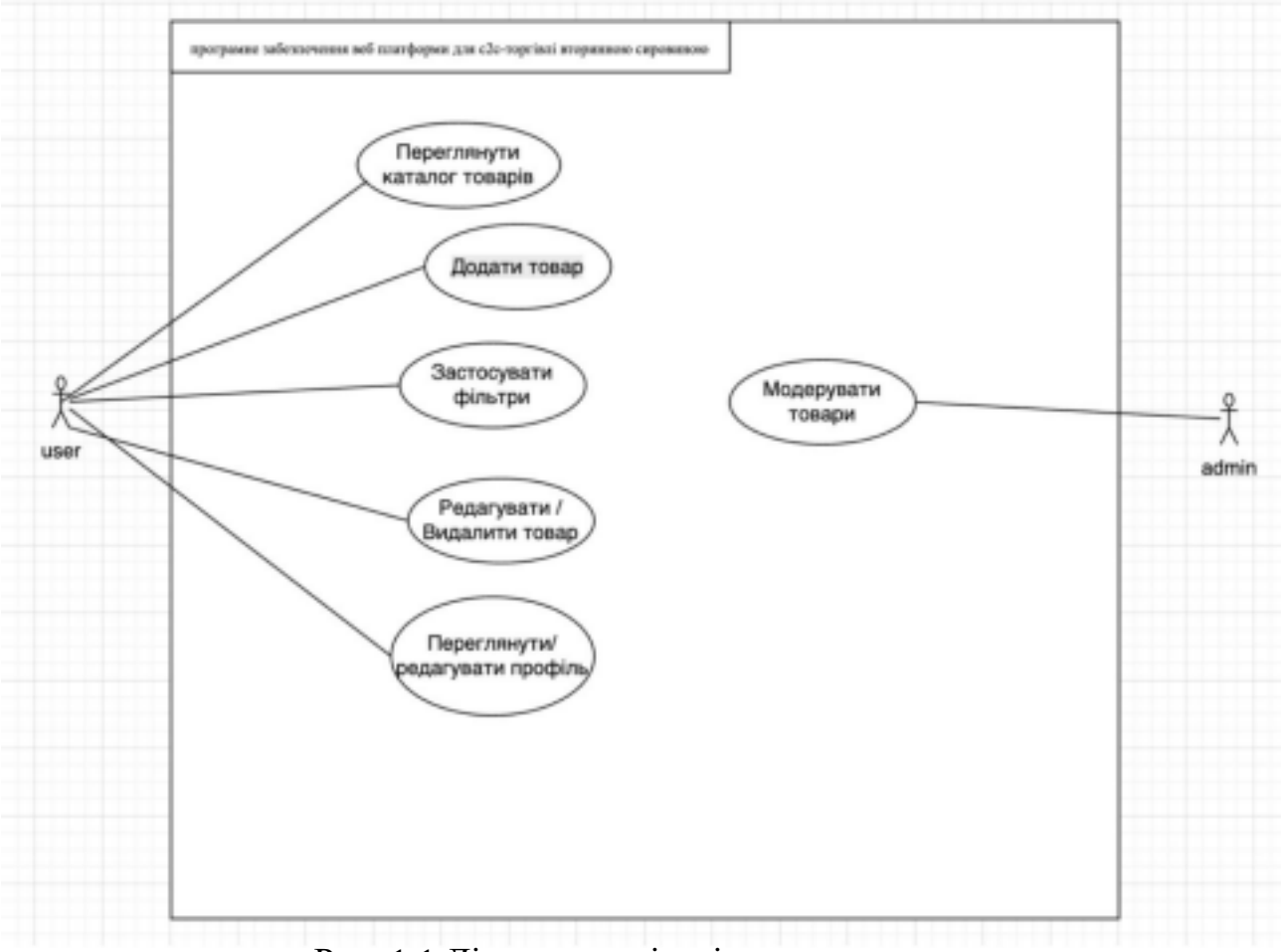


Рис. 1.1 Діаграма варіантів використання

Діаграма послідовності (Sequence Diagram)

Наступним кроком стало створення діаграми послідовності (рис. 1.2), яка дозволяє детально відстежити часову послідовність повідомлень між об’єктами під час виконання конкретного сценарію — наприклад, створення нового оголошення. Цей процес охоплює кілька етапів: спочатку користувач заповнює форму оголошення, далі система проводить перевірку коректності введених

даних, після чого оголошення зберігається у базі даних із початковим статусом "очікує модерації". Адміністратор отримує сповіщення про нове оголошення, переглядає його, та у разі схвалення змінює статус на "активне", після чого оголошення стає видимим для інших користувачів.

Цей тип діаграм особливо корисний для виявлення прихованих залежностей і вузьких місць у логіці обміну даними, що дозволяє уникнути помилок під час реалізації програмного коду.

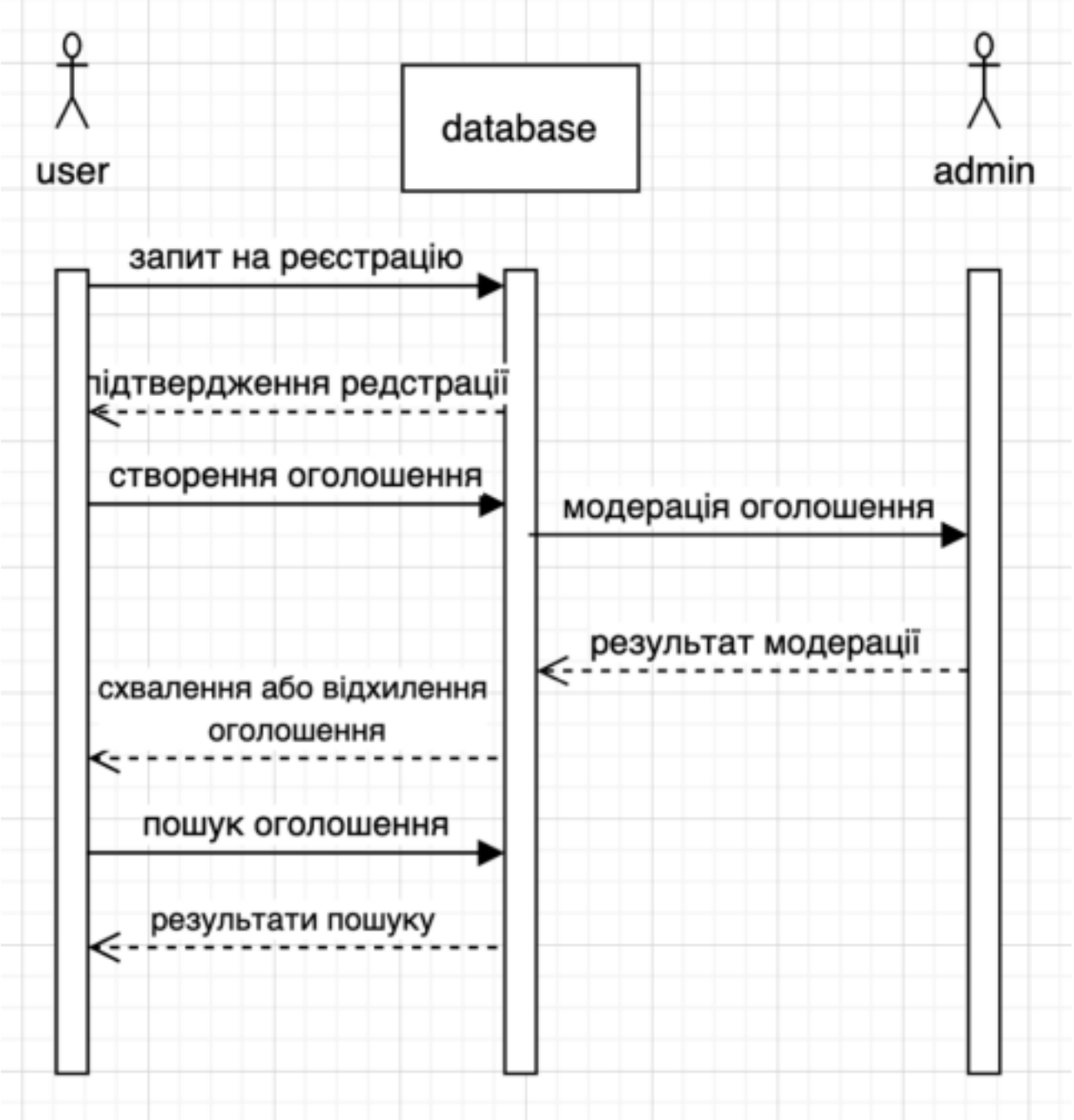


Рис. 1.2 Діаграма послідовності

Діаграма активності (Activity Diagram)

Діаграма активності (рис. 1.3) демонструє алгоритм дій користувача після авторизації у системі. Вона охоплює такі ключові шляхи, як перегляд наявних оголошень, створення нового оголошення, а також обробку випадків відхилення оголошення та повторного подання на модерацію. Кожна гілка активності відображає окремий варіант поведінки: у разі успішної авторизації користувач переходить до вибору доступних дій. Залежно від обраного шляху система виконує відповідні дії, такі як запит до бази даних, перевірку статусу або відображення повідомлень.

Завдяки візуалізації активностей стає можливим ідентифікувати потенційні точки прийняття рішень, моменти, де можливе розгалуження сценаріїв або виникнення паралельних процесів. Це критично важливо для забезпечення стабільності й узгодженості роботи всієї системи.

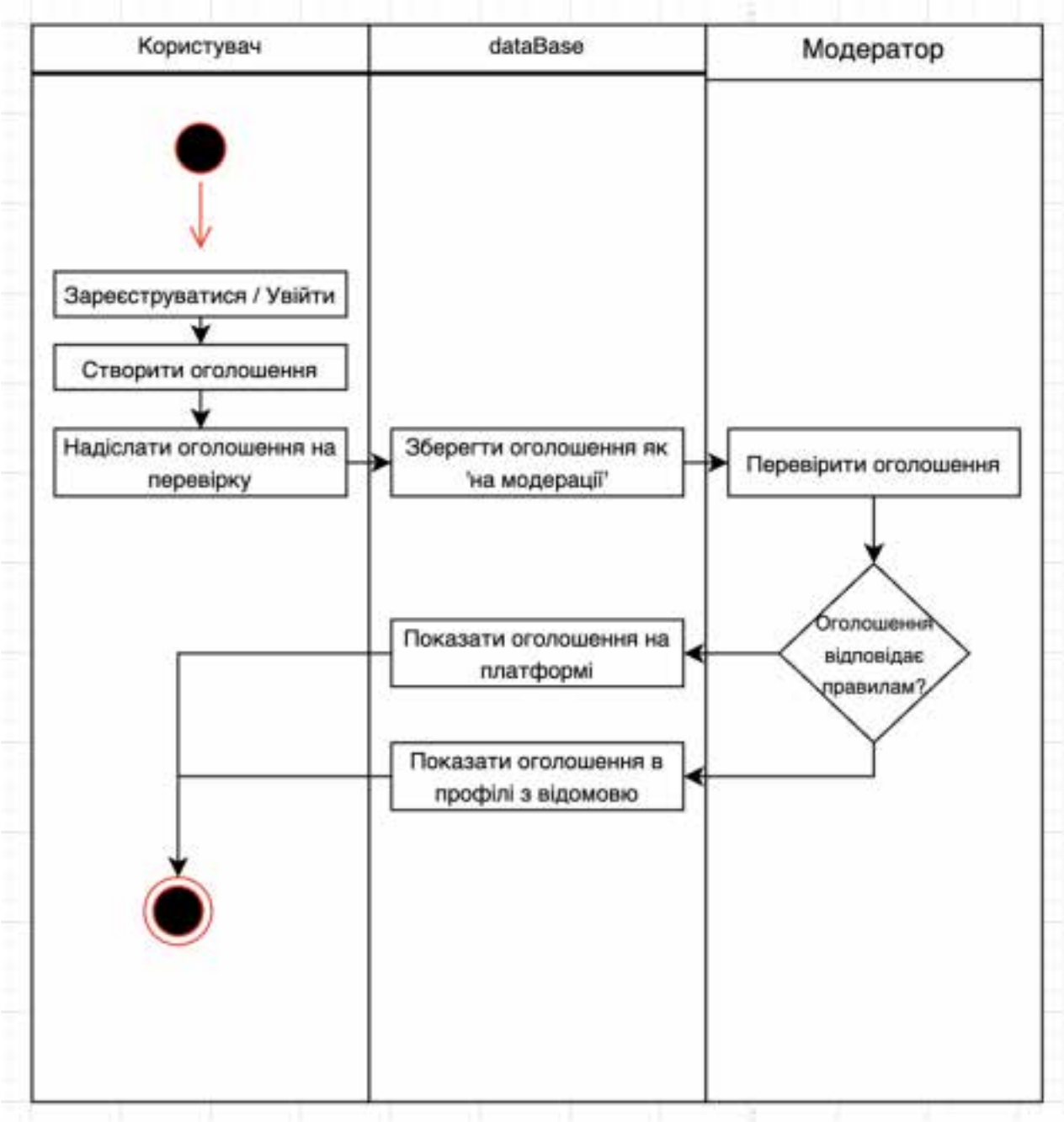


Рис. 1.3 Діаграма активності

Кожна гілка активності відображає окремий варіант поведінки: у разі успішної авторизації користувач переходить до вибору дій, після чого система реагує залежно від обраної опції. Візуалізована структура активностей дозволяє виявити можливі точки прийняття рішень, розгалуження і повернення до головного меню. Діаграма активності також дає змогу описати паралельність або послідовність виконання процесів, що є важливим для програмної реалізації.

Модель абстракцій (рис. 1.4) відображає сутності, що формують предметну область, та взаємозв’язки між ними. До основних абстракцій належать: Користувач, Оголошення, Замовлення, Система Фільтрів, Панель Адміністратора. Кожен користувач може створювати необмежену кількість оголошень, здійснювати замовлення на основі оголошень інших користувачів за допомогою контактів. Адміністратор має доступ до всіх оголошень і може змінювати їхній статус. Модель забезпечує високий рівень повторного використання об'єктів і гнучке розширення функціоналу.

Рис. 1.4 Модель абстракції

Модель абстракцій

У межах аналізу предметної області було створено також модель абстракцій (рис. 1.4), яка відображає основні сутності та зв’язки між ними. До ключових абстрактних об’єктів належать: **Користувач**, **Оголошення**, **Замовлення**, **Система Фільтрації**, **Адміністративна Панель**. Кожна сутність має набір атрибутів та методів, що описують її поведінку у системі. Наприклад, користувач може створювати, редагувати та видаляти оголошення, а також здійснювати замовлення через контактну інформацію, вказану в оголошеннях інших користувачів.

Модель забезпечує гнучкість у розширенні, можливість повторного

використання компонентів та реалізацію принципів інкапсуляції, що підвищує надійність системи та знижує складність підтримки коду в майбутньому.

Рис. 1.5 Діаграма класів

Діаграма пакетів подає організаційну структуру програмного забезпечення на логічному рівні (див. рис. 1.8). Програмне забезпечення було розділено на модулі. Така модульність забезпечує гнучкість, масштабованість та зручність супроводу. Відокремлення функціональних блоків у пакети дозволяє незалежно тестувати, розробляти та оновлювати окремі частини системи без ризику впливу на інші компоненти також полегшує масштабування та обслуговування коду

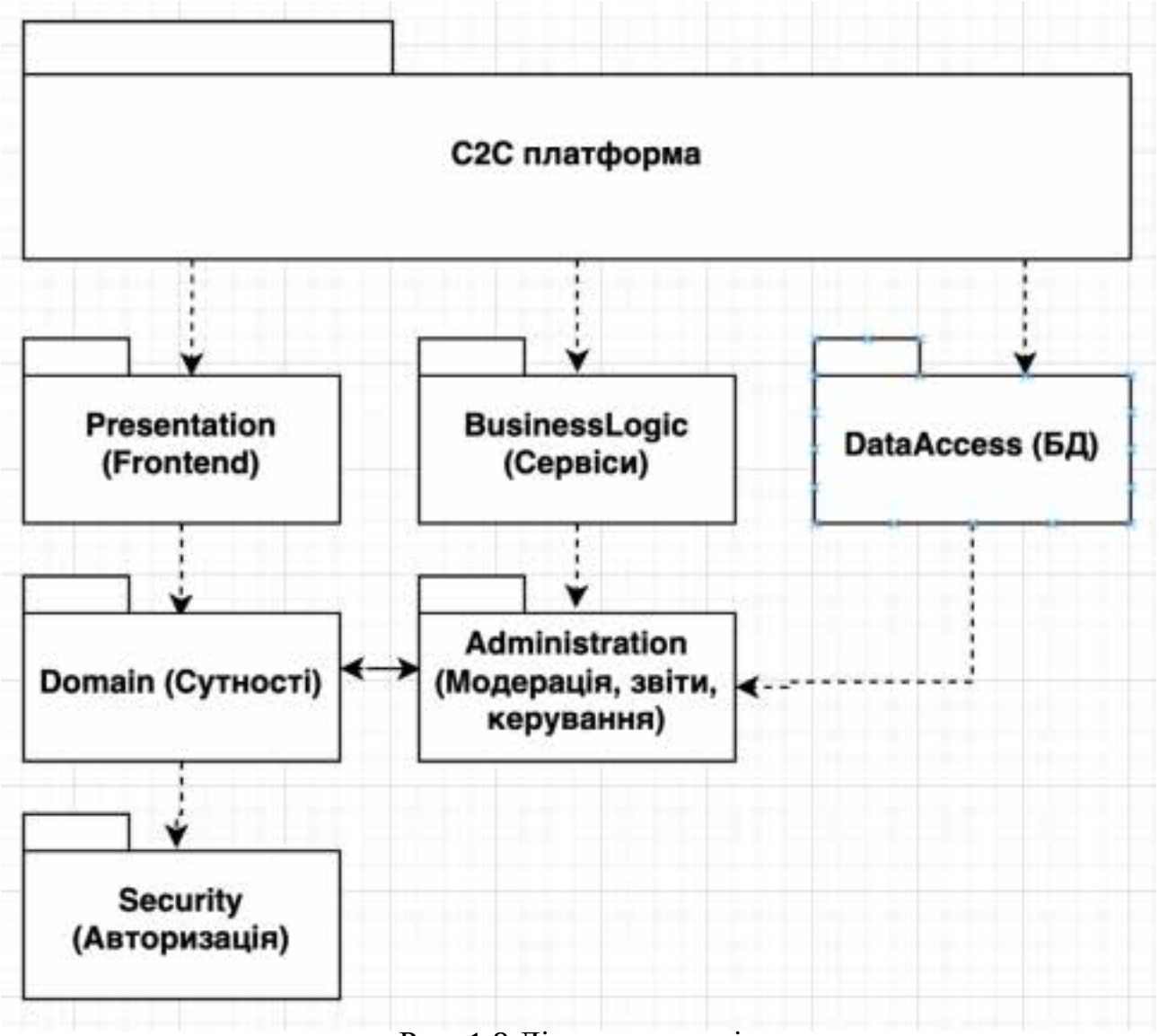


Рис. 1.8 Діаграма пакетів

Модель даних фізичного рівня реалізована у вигляді реляційної структури з нормалізованими таблицями, які зберігають дані про працівників, графіки, зміни, ролі користувачів, повідомлення, історію змін і звіти (див. рис. 1.9). Усі таблиці пов’язані зовнішніми ключами, що забезпечує цілісність даних і логіку відношень. Модель дотримується вимог третьої нормальної форми, що зменшує надмірність даних і мінімізує ймовірність аномалій при оновленні. Реалізація такої моделі у СУБД забезпечує високу продуктивність при виконанні запитів, надійне зберігання інформації та підтримку масштабованості.

Рис. 1.9 Модель даних фізичного рівня

Діаграма компонентів моделює структуру вихідного коду системи, відображаючи взаємозв'язки між основними модулями платформи для купівлі та продажу вторсировини. Компоненти реєстрації користувачів, управління товарами, модерації контенту представлені як окремі частини, що взаємодіють з базою даних (MongoDB) та між собою через визначені інтерфейси. Це дозволяє реалізувати принципи слабкого зв'язування та високої когезії між модулями. У діаграмі також враховано процес компіляції та створення виконуваного файлу, який об'єднує всі компоненти в єдину платформу.

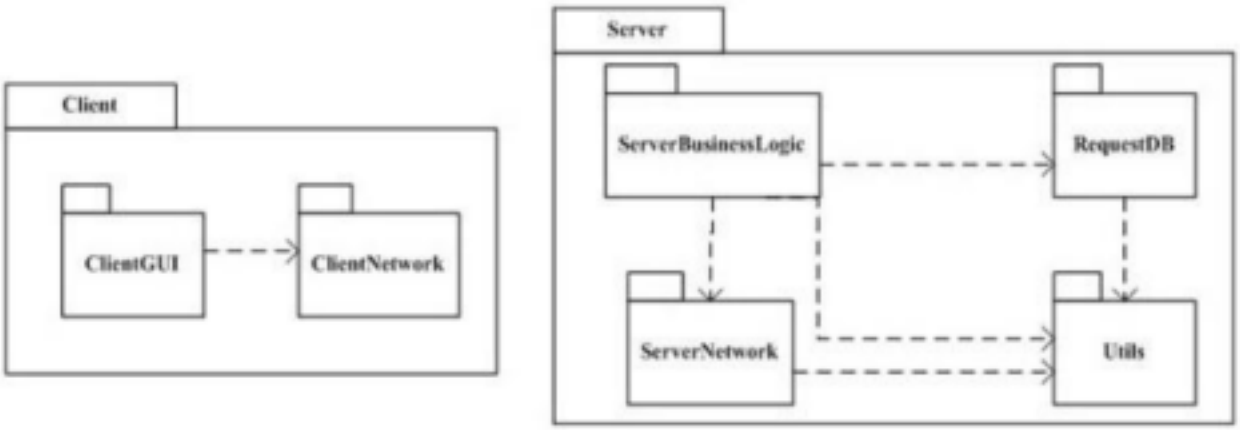


Рис. 1.10 Діаграма компонентів

Діаграма розгортання моделює фізичне середовище функціонування

системи. Вона демонструє, що платформа складається з клієнтської частини, розгорнутої на комп’ютерах користувачів, та серверної частини, розміщеної на сервері, який містить базу даних та логіку обробки запитів. Клієнтська частина взаємодіє з сервером через протокол HTTP, що забезпечує масштабованість і можливість використання системи у мережевому середовищі. Така структура дозволяє розподіляти навантаження, централізовано адмініструвати систему та забезпечити безпеку даних, враховуючи специфіку онлайн платформ для продажу вторсировини.

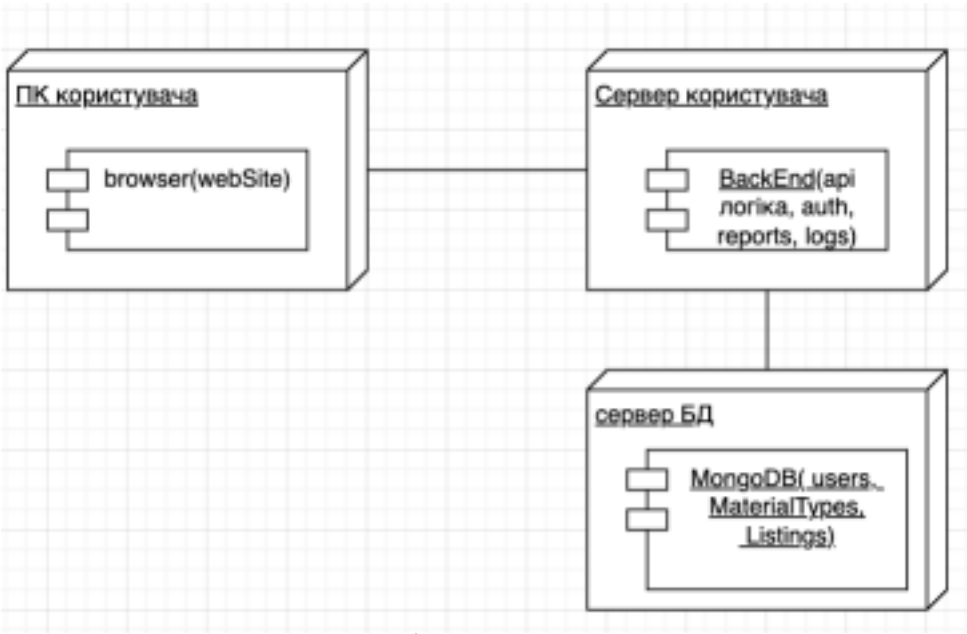


Рис. 1.11 Діаграма розгортання

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Логічна модель даних

Логічна модель даних для C2C платформи з продажу вторинної сировини відображає структуру зберігання інформації про користувачів, оголошення та їхні взаємозв'язки. Враховуючи використання NoSQL бази даних MongoDB, модель фокусується на колекціях документів, які є гнучкими та можуть містити вкладені структури, що добре підходить для представлення різноманітних типів вторинної сировини та їхніх характеристик.

Основні колекції, які передбачаються в базі даних MongoDB для даної платформи:

1. **Користувачі (users):** Ця колекція зберігатиме інформацію про зареєстрованих користувачів платформи. Кожен документ у цій колекції представлятиме одного користувача та може містити наступні поля:

   - _id: Унікальний ідентифікатор користувача (ObjectID у MongoDB).
   - username: Логін користувача (рядок, унікальний).
   - email: Електронна адреса користувача (рядок, унікальна).
   - password_hash: Хешований пароль користувача (рядок).
   - role: Роль користувача в системі (рядок, можливі значення: "user", "moderator").
   - registration_date: Дата реєстрації користувача (дата).
   - contact_phone: Контактний номер телефону користувача (рядок).
   - personal_info: Додаткова інформація про користувача (вкладений документ, може містити ім'я, прізвище, місцезнаходження тощо).
2. **Оголошення (listings):** Ця колекція зберігатиме інформацію про оголошення, розміщені користувачами для продажу вторинної сировини. Кожен документ представлятиме одне оголошення та може включати:
   - _id: Унікальний ідентифікатор оголошення (ObjectID у MongoDB).
   - user_id: Ідентифікатор користувача, який розмістив оголошення (ObjectID, посилання на колекцію "users").
   - category: Категорія вторинної сировини (рядок, наприклад, "метал", "пластик", "папір" тощо).
   - title: Заголовок оголошення (рядок).
   - description: Детальний опис вторинної сировини (рядок).
   - images: Масив ідентифікаторів файлів зображень, пов'язаних з оголошенням (масив рядків або ObjectIDs, якщо використовується GridFS).
   - price: Ціна за одиницю (число, необов'язково).
   - unit: Одиниця виміру (рядок, наприклад, "кг", "тонна", "шт.", необов'язково).
   - location: Місцезнаходження вторинної сировини (вкладений

документ з полями для широти та довготи або текстовим описом).

- publication_date: Дата публікації оголошення (дата).

- is_active: Статус оголошення (булеве значення, наприклад, true - активне, false - неактивне).
- contact_phone: Контактний номер телефону продавця (дублюється для зручності, рядок).

У цій моделі дані про користувачів та оголошення зберігаються в окремих колекціях. Зв'язок між оголошеннями та користувачами здійснюється за допомогою поля user_id в колекції "listings", яке містить ідентифікатор користувача з колекції "users". MongoDB дозволяє ефективно здійснювати пошук оголошень за різними критеріями, такими як категорія, місцезнаходження та ключові слова в описі.

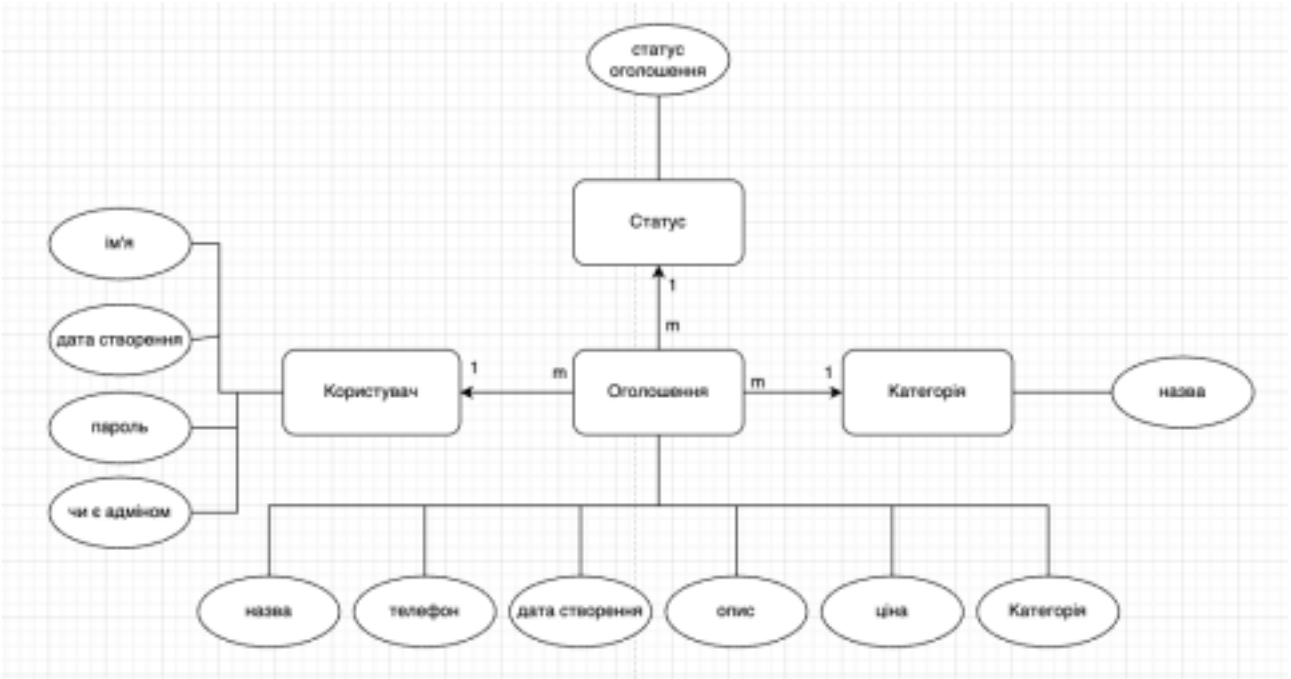


Рис. 2.1 ER Diagram

2.2 Вибір системи управління інформаційною базою

У сучасних веб-проєктах, особливо тих, що пов’язані з обробкою великих обсягів користувацької інформації та динамічних оголошень, надзвичайно

важливим є вибір надійної та гнучкої системи управління базами даних (СУБД). Саме СУБД визначає, як будуть зберігатися, організовуватися, оброблятися та надаватися дані, що, у свою чергу, безпосередньо впливає на

ефективність роботи платформи, швидкість доступу до інформації, масштабованість проєкту та загальну якість користувацького досвіду. Це особливо критично для C2C платформи, яка забезпечує взаємодію між численними користувачами та дозволяє розміщувати широкий спектр оголошень про продаж різних видів вторинної сировини.

У рамках даного проєкту було прийнято рішення на користь використання MongoDB — документо-орієнтованої NoSQL бази даних, яка стала одним із провідних рішень у сфері розробки масштабованих веб-додатків. Цей вибір був зумовлений рядом переваг, які роблять MongoDB ідеальним рішенням для платформ із гнучкою структурою даних і постійно змінюваними вимогами.

MongoDB зберігає дані у форматі BSON (розширений JSON), що дозволяє кожному документу мати власну унікальну структуру. Такий підхід кардинально відрізняється від реляційних баз даних, де кожен запис повинен відповідати строго визначеній схемі таблиці. Для проєкту з торгівлі вторинною сировиною це відкриває широкі можливості: різні категорії матеріалів, наприклад, метал, пластик, скло або папір, можуть мати абсолютно різні властивості, і ці особливості можна легко враховувати без перебудови всієї структури бази. Наприклад, для металів важливими можуть бути марка сплаву, маса або рівень забруднення, тоді як для пластику — тип, колір або об’єм.

Крім цього, MongoDB забезпечує можливість горизонтального масштабування завдяки функціоналу шардінгу (sharding), що дозволяє розподіляти дані між кількома серверами або вузлами. Це особливо корисно при обробці великих обсягів інформації або в разі зростання кількості користувачів та активності на платформі. У майбутньому це забезпечить стабільну роботу системи навіть під час пікового навантаження, що є критично важливим для платформ із високим рівнем взаємодії користувачів.

Ще однією важливою характеристикою MongoDB є її висока продуктивність при виконанні запитів, особливо коли йдеться про пошук і фільтрацію даних. Завдяки використанню індексів, можливості створення складених і часткових індексів, MongoDB дозволяє значно скоротити час доступу до інформації, що позитивно позначається на швидкості відгуку платформи. Наприклад, пошук оголошень за категорією, ціною або місцезнаходженням виконується дуже швидко навіть при наявності великої кількості записів.

Важливо також відзначити, що MongoDB має потужну екосистему інструментів, включаючи офіційні драйвери для більшості популярних мов

програмування (JavaScript, Python, Java тощо), інструменти адміністрування (наприклад, MongoDB Compass), а також активну спільноту, що постійно

підтримує і вдосконалює продукт. Це значно полегшує процес розробки, розгортання та обслуговування проєкту, а також дозволяє швидко знаходити рішення типових проблем завдяки наявності численних прикладів і документації.

З точки зору безпеки, MongoDB підтримує механізми автентифікації та авторизації, які дозволяють визначати, хто і які саме дії може виконувати з базою даних. Також реалізовано підтримку реплікації, що гарантує збереження даних у разі збоїв та дозволяє забезпечити високу доступність системи. Реплікаційні набори MongoDB дозволяють мати резервні копії даних, які автоматично синхронізуються з основною базою.

І нарешті, важливою перевагою MongoDB є її гнучкість в інтеграції з сучасними хмарними рішеннями, такими як AWS, Google Cloud або Azure, а також можливість використання в середовищах контейнеризації, таких як Docker. Це створює додаткові переваги для автоматизованого розгортання, масштабування та резервного копіювання бази даних.

Отже, беручи до уваги специфіку та потреби C2C платформи з торгівлі вторинною сировиною, MongoDB виявляється оптимальним вибором. Вона поєднує в собі гнучкість, масштабованість, продуктивність і простоту інтеграції, що забезпечує надійне та ефективне управління інформацією. Її використання дозволяє адаптувати систему до постійно змінюваних потреб користувачів, підтримувати високий рівень доступності та забезпечити швидку реакцію на запити — усе це критично важливо для забезпечення стабільної роботи та розвитку платформи.

2.3 Створення інформаційної бази

Для реалізації інформаційної бази було обрано NoSQL СУБД MongoDB – гнучку та масштабовану платформу, яка використовує документо-орієнтовану модель даних, забезпечує високу продуктивність при роботі з різноманітними структурами даних та спрощене горизонтальне масштабування [посилання на джерело про MongoDB. У даному випадку логіка створення бази полягає у визначенні колекцій документів, які відповідають основним сутностям

предметної області – користувачам та оголошенням про продаж вторинної сировини.

Основною логічною одиницею зберігання даних у MongoDB є колекція. Для даної платформи передбачається створення двох основних колекцій: **users** та **listings**.

Колекція **users** слугуватиме для зберігання інформації про зареєстрованих користувачів платформи. Кожен користувач буде представлений окремим документом у цій колекції. Структура документа може включати наступні поля:

- _id: Унікальний ідентифікатор користувача (ObjectID, автоматично генерується MongoDB).
- username: Логін користувача (рядок, унікальний).
- email: Електронна адреса користувача (рядок, унікальна).
- password_hash: Хешований пароль користувача (рядок).
- role: Роль користувача в системі (рядок, можливі значення: "user", "moderator").
- registration_date: Дата реєстрації користувача (дата).
- contact_phone: Контактний номер телефону користувача (рядок).
- personal_info: Вкладений документ, що може містити додаткову інформацію про користувача, таку як ім'я, прізвище, місцезнаходження тощо.

Для забезпечення унікальності логінів та електронних адрес у колекції users будуть створені унікальні індекси для полів username та email. Поле _id за замовчуванням є індексованим.

Колекція **listings** призначена для зберігання інформації про оголошення, розміщені користувачами для продажу вторинної сировини. Кожне оголошення буде представлене окремим документом з наступною можливою структурою:

- _id: Унікальний ідентифікатор оголошення (ObjectID, автоматично генерується MongoDB).
- user_id: Ідентифікатор користувача, який розмістив оголошення (ObjectID, посилання на документ у колекції users).

- category: Категорія вторинної сировини (рядок, наприклад, "метал", "пластик", "папір").
- title: Заголовок оголошення (рядок).
- description: Детальний опис вторинної сировини (рядок).
- images: Масив ідентифікаторів файлів зображень, пов'язаних з оголошенням (масив рядків або ObjectIDs, якщо використовується GridFS).
- price: Ціна за одиницю (число, необов'язково).
- unit: Одиниця виміру (рядок, наприклад, "кг", "тонна", "шт.", необов'язково).
- location: Вкладений документ з інформацією про місцезнаходження вторинної сировини (наприклад, поля latitude, longitude або текстовий опис address).
- publication_date: Дата публікації оголошення (дата).
- is_active: Статус оголошення (булеве значення, наприклад, true - активне, false - неактивне).
- contact_phone: Контактний номер телефону продавця (рядок).

Зв'язок між оголошеннями та користувачами встановлюється за допомогою поля user_id, яке містить ObjectID відповідного користувача з колекції users. MongoDB дозволяє виконувати операції об'єднання (lookup) для отримання інформації про користувача разом з його оголошеннями, коли це необхідно.

Гнучкість документо-орієнтованої моделі MongoDB дозволяє легко додавати нові поля до документів у колекціях users та listings без необхідності змінювати схему всієї бази даних. Наприклад, якщо в майбутньому знадобиться додати інформацію про рейтинги користувачів або додаткові характеристики для певних категорій вторинної сировини, це можна буде зробити, просто додавши відповідні поля до документів.

Використання MongoDB замість реляційної СУБД, як MySQL, у даному проєкті зумовлено необхідністю обробки потенційно різнорідних даних про вторинну сировину та забезпечення гнучкості при масштабуванні платформи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕПЕЧЕННЯ

3.1 Вибір інструментарію для розробки програмного забезпечення

Розробка сучасної C2C-платформи для торгівлі вторинною сировиною вимагає застосування продуманого технологічного стеку, який забезпечить високу продуктивність, надійність, масштабованість, зручність для користувача та швидкість розробки. Оскільки така платформа має на меті з'єднати приватних користувачів у межах цифрового середовища для обміну товарами, критично важливо забезпечити стабільну роботу системи навіть за умови великої кількості одночасних запитів. У цьому розділі докладно розглядається вибір технологій, інструментів та середовищ розробки, які використовувалися під час створення вебплатформи.

3.1.1 Фронтенд розробка: React JS та Ant Design

Для розробки клієнтської частини платформи обрано бібліотеку React JS. React є однією з найпопулярніших JavaScript-бібліотек для створення інтерактивних та динамічних веб-інтерфейсів.

Переваги React JS:

- **Компонентний підхід:** React дозволяє розбивати інтерфейс на незалежні багаторазові компоненти, що полегшує розробку, тестування та підтримку коду.
- **Віртуальний DOM:** React використовує віртуальне DOM для оптимізації оновлення інтерфейсу, що забезпечує високу продуктивність та швидку реакцію на дії користувача.
- **Велика спільнота та екосистема:** React має велику та активну спільноту розробників, що забезпечує доступ до численних бібліотек, інструментів та навчальних матеріалів.

- **Декларативний синтаксис:** React дозволяє описувати бажаний стан інтерфейсу, а бібліотека сама оновлює DOM, що спрощує розуміння та написання коду.

Для швидкої розробки користувацького інтерфейсу та забезпечення його сучасності та зручності використання обрано UI-бібліотеку Ant Design. Ant Design надає великий набір готових до використання компонентів (кнопки, форми, таблиці, навігація тощо) з сучасним дизайном та високою якістю.

Переваги Ant Design:

- **Готові компоненти:** Ant Design значно прискорює процес розробки, надаючи широкий спектр попередньо розроблених та стилізованих компонентів.
- **Адаптивність:** Компоненти Ant Design є адаптивними та добре виглядають на різних пристроях.
- **Зручність використання:** Бібліотека має зрозумілу документацію та API, що полегшує її інтеграцію в проєкт.
- **Консистентний дизайн:** Ant Design забезпечує єдиний стиль для всіх компонентів, що сприяє створенню професійного та гармонійного інтерфейсу.

Вибір React JS у поєднанні з Ant Design дозволить створити сучасний, інтерактивний та зручний інтерфейс для користувачів платформи з продажу вторинної сировини.

3.1.2 Бекенд розробка: Fastify та Node.js

Для розробки серверної частини платформи обрано фреймворк Fastify, який працює на базі Node.js. Node.js є JavaScript-середовищем виконання на сервері, що дозволяє використовувати одну мову програмування як для фронтенду, так і для бекенду, що спрощує розробку та покращує взаємодію між командами.

Переваги Fastify:

- **Висока продуктивність:** Fastify є одним з найшвидших Node.js фреймворків, що забезпечує швидку обробку запитів та низьку затримку. •

Проста структура: Fastify має зрозумілий та лаконічний API, що полегшує розробку та підтримку коду.

- **Розширюваність:** Fastify підтримує плагіни, що дозволяє легко додавати нову функціональність та інтегрувати сторонні бібліотеки.
- **Підтримка TypeScript:** Fastify має вбудовану підтримку TypeScript, що сприяє написанню більш надійного та підтримуваного коду.

Використання Fastify на базі Node.js дозволить створити потужний та продуктивний бекенд для обробки запитів клієнтської частини, керування даними та забезпечення функціональності платформи.

3.1.3 База даних: MongoDB Оскільки система має справу з гнучкими структурами даних (користувачі, оголошення, повідомлення, транзакції), було вирішено використати документоорієнтовану СУБД — MongoDB. На відміну від реляційних баз, MongoDB дозволяє зберігати дані у форматі BSON (Binary JSON), що забезпечує більшу гнучкість у структурі документів та кращу продуктивність при роботі з неструктурованими або слабо структурованими даними.

MongoDB дозволяє ефективно масштабувати систему горизонтально, що важливо для майбутнього зростання проєкту. Вбудована підтримка реплікації та автоматичного шардингу дозволяє досягти високої доступності та відмовостійкості.

Крім того, можливість створення індексів, включно з геопросторовими та текстовими, дає змогу реалізувати пошук оголошень за категоріями, місцем розташування чи ключовими словами з високою швидкістю.

3.1.4 Автентифікація та авторизація: Auth0-verify

Для забезпечення безпеки та керування доступом користувачів до платформи буде використана бібліотека Auth0-verify. Auth0 є платформою для автентифікації та авторизації, яка надає зручні інструменти для інтеграції безпечних механізмів аутентифікації у веб-додатки. Бібліотека Auth0-verify допоможе верифікувати токени, видані Auth0, на бекенді.

Переваги Auth0:

- **Безпека:** Auth0 забезпечує надійні механізми автентифікації та авторизації, включаючи підтримку різних протоколів (OAuth 2.0, OpenID Connect).
- **Простота інтеграції:** Auth0 легко інтегрується з різними фреймворками та платформами, включаючи React та Fastify.
- **Гнучкість:** Auth0 підтримує різні сценарії автентифікації, включаючи соціальні мережі, корпоративні облікові записи тощо.

3.1.5 Хмарна інфраструктура: AWS-sdk

Для взаємодії з хмарними сервісами Amazon Web Services (AWS), такими як зберігання зображень оголошень (наприклад, AWS S3), буде використано AWS sdk для Node.js. AWS надає широкий спектр масштабованих та надійних хмарних сервісів.

Переваги AWS:

- **Масштабованість:** AWS дозволяє легко масштабувати інфраструктуру залежно від потреб платформи.
- **Надійність:** AWS забезпечує високу доступність та відмовостійкість сервісів.
- **Широкий спектр сервісів:** AWS надає безліч сервісів для зберігання даних, обчислень, мереж та іншого.

3.1.6 Середовище розробки: Microsoft Visual Studio

Для розробки програмного забезпечення буде використано інтегроване середовище розробки (IDE) Microsoft Visual Studio. Хоча Visual Studio традиційно асоціюється з розробкою на C# та .NET, воно також забезпечує відмінну підтримку JavaScript, TypeScript та Node.js завдяки різноманітним розширенням та вбудованим інструментам.

Переваги Microsoft Visual Studio для JavaScript/Node.js розробки: ·

Потужний редактор коду: Visual Studio надає розширені можливості редагування коду, включаючи підсвічування синтаксису, автодоповнення

(IntelliSense), рефакторинг та навігацію по коду.

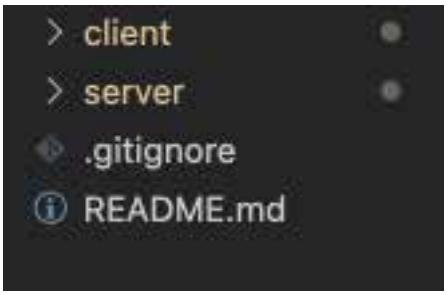
- **Інтегрований термінал:** Вбудований термінал дозволяє виконувати команди Node.js та інші інструменти безпосередньо з IDE.

- **Інструменти відладки:** Visual Studio надає потужні інструменти для відладки JavaScript та Node.js коду як на клієнтській, так і на серверній стороні.
- **Підтримка TypeScript:** Відмінна підтримка TypeScript з інтелектуальними підказками та перевіркою типів.
- **Інтеграція з системами контролю версій:** Вбудована підтримка Git та інших систем контролю версій.
- **Розширюваність:** Visual Studio має велику кількість розширень, які можуть покращити процес розробки для різних технологій, включаючи React та Node.js.

Вибір Microsoft Visual Studio як IDE забезпечить зручне та функціональне середовище для розробки всіх компонентів платформи, включаючи фронтенд (React), бекенд (Fastify/Node.js) та інтеграцію з іншими сервісами (AWS).

3.2 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення для C2C платформи з продажу вторинної сировини (див. рис. 3.1, відображає структуру вашого проєкту) чітко розділена на клієнтську та серверну частини, що сприяє кращій організації коду, полегшує розробку, тестування та майбутню підтримку. Така структура дозволяє незалежно розробляти та масштабувати різні частини застосунку.


client
server
.gitignore
README.md


client
flow-typed
node_modules
public ~/Desktop/fe
src
components
images
pages
rest
store
App.js
App.test.js
index.js M
index.scss
reportWebVitals.js
setupTests.js
variables.js M
env.development.l
env.local
.eslintrc
.flowconfig
.gitignore
.prettierignore
.prettierrc

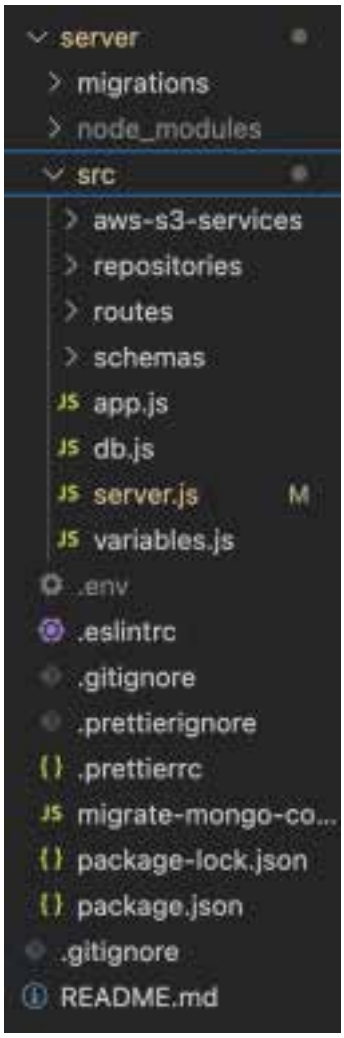


Рис. 3.1 Структура програмного забезпечення

Основна директорія проєкту містить дві головні піддиректорії: client та server, які відповідають за клієнтську та серверну частини платформи відповідно.

Директорія client:

Ця директорія містить весь код, що відповідає за користувацький інтерфейс, розроблений за допомогою бібліотеки React JS.

- flow-typed/: Містить файли визначень типів для Flow, якщо він використовується для статичної типізації JavaScript коду клієнта.
- node_modules/: Містить встановлені Node.js пакети та залежності, необхідні для роботи React, Ant Design та інших бібліотек клієнтської частини.
- public/: Містить статичні файли, такі як зображення та інші публічні ресурси.
- src/: Основна директорія з вихідним кодом клієнтської частини.

- components/: Містить багаторазові UI-компоненти, розроблені з використанням React та Ant Design.
- images/: Містить статичні зображення, що використовуються в інтерфейсі.
- pages/: Містить компоненти, які представляють окремі сторінки веб сайту (наприклад, головна сторінка, сторінка оголошень, особистий кабінет).
- rest/: Може містити код для взаємодії з API серверної частини. o store/: Містить код для управління станом застосунку (наприклад, за допомогою Redux або Context API).
- Основні файли застосунку (App.js), точка входу (index.js), файли стилів (index.scss, variables.js) та інші службові файли.

• Файли конфігурації (.env.development.local, .env.local), файли лінтера (.eslintrc), форматування коду (.prettierrc, .prettierignore), система контролю версій (.gitignore) та менеджер пакунків (package.json, package-lock.json).

Директорія server:

Ця директорія містить весь код серверної частини платформи, розроблений за допомогою фреймворку Fastify на базі Node.js.

• migrations/: містить файли міграцій бази даних для управління змінами схеми MongoDB.

• node_modules/: Містить встановлені Node.js пакети та залежності, необхідні для роботи Fastify, MongoDB драйвера, Auth0-verify, AWS-sdk та інших серверних бібліотек.

• src/: Основна директорія з вихідним кодом серверної частини. o aws-s3-services/: Містить код для взаємодії з AWS S3 для завантаження та зберігання зображень.
- repositories/: Містить код для взаємодії з базою даних MongoDB (операції CRUD).
- routes/: Містить визначення API-маршрутів, які обробляють запити від клієнтської частини.

- schemas/: містить схеми даних для валідації запитів та відповідей.
- Основні файли серверного застосунку (app.js, server.js), файл підключення до бази даних (db.js), файли конфігурації (variables.js) та інші службові файли.

- Файли конфігурації (.env), файли лінтера (.eslintrc), форматування коду (.prettierrc, .prettierignore), система контролю версій (.gitignore), конфігурація міграцій (migrate-mongo-config.js) та менеджер пакунків (package.json, package-lock.json).

Корінь проєкту:

У корені проєкту знаходяться загальні файли, такі як .gitignore для визначення файлів, які не потрібно відстежувати системою контролю версій Git, та README.md з описом проєкту.

Така організаційна структура забезпечує чітке розділення відповідальності між клієнтською та серверною частинами, полегшує навігацію по коду, сприяє паралельній розробці та спрощує розгортання та масштабування окремих компонентів платформи. Використання окремих директорій для клієнта та сервера є стандартною практикою для сучасних веб-застосунків, побудованих з використанням React на фронтенді та Node.js на бекенді.

4. ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування системи

Відповідно до міжнародного стандарту **IEEE Std 829-1983**, процес тестування визначається як ретельне і послідовне дослідження програмного забезпечення, що має на меті не лише виявлення відмінностей між фактично реалізованими властивостями програмного продукту та тими, які були задані на етапі проектування, але й оцінку його якості, стабільності та функціональної відповідності. Такі невідповідності прийнято називати **дефектами** або помилками.

Основні цілі тестування:

• виявлення, фіксація та подальший аналіз дефектів у програмному продукті;

• формування обґрунтованих висновків щодо поточного рівня якості програмного забезпечення;

• перевірка реалізації функціональних і нефункціональних вимог у рамках конкретних сценаріїв використання;

• забезпечення відповідності фактичної поведінки системи до очікуваної;

• підтвердження того, що продукт реалізований згідно з технічним завданням та вимогами замовника.

Суттєвою складовою тестування є підвищення загальної **якості ПЗ**, що є головною метою як для розробників, так і для зацікавлених сторін, зокрема замовників. Одним із ключових показників якості програмного забезпечення вважається **надійність**, особливо коли йдеться про системи з підвищеними вимогами до стабільності — такі як системи безпеки, системи реального часу, банківське або медичне програмне забезпечення.

Верифікація, атестація, аудит

• **Верифікація** — це процес, який визначає, чи відповідає створене програмне забезпечення специфікаціям, технічним вимогам та архітектурі, визначеним у попередніх етапах розробки. До процесу верифікації можуть входити як формальні методи аналізу, так і практичні — тестування, перевірка документації та огляди коду.

• **Атестація** — це визначення відповідності фактичної реалізації ПЗ його призначенню. Інакше кажучи, мова йде про те, наскільки ефективно

продукт виконує свої основні функції.

• **Оцінка проекту** — це більш комплексний процес, який дозволяє визначити стан розробки, якість менеджменту, використання ресурсів та дотримання графіку.

• **Аудит** — перевірка дотримання вимог, планів, договорів, а також технічної документації, що супроводжує програмний продукт.

Усі перераховані вище процедури в сукупності формують поняття **тестування програмного забезпечення** як багатоступеневий процес перевірки, який охоплює і технічну, і організаційну складову. Перед початком тестування важливо чітко визначити, за якими саме стандартами (наприклад, ISO/IEC 25010, IEEE 1012 тощо) буде проводитися тестування та які критерії відповідності застосовуватимуться.

Важливість етапу тестування

Тестування ПЗ — це одна з ключових фаз життєвого циклу розробки програмного забезпечення. Воно дозволяє перевірити не тільки **працездатність і стабільність** системи, а й знайти критичні помилки, які могли бути допущені під час розробки.

Завдання тестування полягає в тому, щоб отримати конкретні результати при визначених вхідних даних, тим самим перевірити коректність функціонування програмного продукту і переконатися в його **надійності, безпечності та відповідності** очікуванням кінцевого користувача.

На сьогодні існує безліч методів тестування, але жоден з них не гарантує **повного виявлення усіх помилок або дефектів**. Це особливо актуально при тестуванні закритого (пропрієтарного) коду, де тестувальники обмежені лише доступним функціоналом без можливості аналізу внутрішньої логіки. У таких випадках застосовується **формалізований процес верифікації**, який дозволяє встановити, що система відповідає вимогам лише в рамках конкретного підходу до тестування. Таким чином, **абсолютна відсутність помилок у системі не може бути гарантована** жодним з існуючих методів.

Тестування має включати перевірку всіх гілок програми та має включати мінімальний набір вимог.

Під час тестування необхідно виконати перевірку:

- **Тестування розміщення оголошень:** Перевірка коректності завантаження зображень через AWS S3, правильності відображення інформації про оголошення.

- **Тестування пошуку:** Перевірка точності та релевантності результатів пошуку за різними критеріями.
- **Тестування модерації:** Перевірка роботи інструментів адміністративної панелі для модерації оголошень та користувачів. · **Тестування безпеки:** Особлива увага має бути приділена захисту персональних даних користувачів та запобіганню несанкціонованому доступу. Перевірка інтеграції з Auth0-verify для автентифікації API.

Методи тестування:

- **Тестування "чорної скриньки":** Тестування функціональності системи без доступу до вихідного коду. Тестувальник взаємодіє з платформою через користувацький інтерфейс або API, перевіряючи, чи відповідає її поведінка очікуваній на основі вимог та специфікацій. Приклади: перевірка роботи форм реєстрації, розміщення оголошень, пошуку.
- **Тестування "білої скриньки":** Тестування внутрішньої структури та коду окремих модулів. Розробники можуть писати юніт-тести для перевірки правильності роботи окремих функцій та компонентів React та Fastify.

Існують також спеціальні методи для тестування тих аспектів програм, які є функціональними, тобто таких, які не впливають на працездатність самих програм.

Це такі види тестування як:

1. **Тестування користувацького інтерфейсу (UI/UX тестування):** Перевірка зручності, інтуїтивності та візуальної привабливості інтерфейсу для користувачів. Оцінка адаптивності дизайну на різних пристроях.

2. **Тестування продуктивності:** Оцінка швидкості завантаження сторінок, часу відгуку сервера при різних рівнях навантаження. Перевірка масштабованості системи.

3. **Тестування безпеки:** Перевірка платформи на наявність вразливостей, таких як SQL-ін'єкції (хоча у випадку MongoDB це менш актуально, але важливі інші аспекти безпеки), XSS-атаки, захист даних користувачів. Перевірка безпеки API.

4.2 Апаратні та технічі засоби

Для успішного розгортання та функціонування C2C вебплатформи з продажу вторинної сировини необхідно врахувати апаратні та програмні вимоги як для серверної, так і для клієнтської частини, а також для інфраструктури зберігання даних та файлів.

Апаратні вимоги для сервера

Таблиця 2.

Ресурс Мінімальні вимоги		Рекомендовані вимоги
Процесор	1 ГГц	2 ГГц і вище
Оперативна пам'ять	1 ГБ	2 ГБ і вище
Жорсткий диск	20 ГБ (для ОС, застосунку та логів)	50 ГБ і вище
Мережа	100 Мбіт/с	1 Гбіт/с і вище

Програмні вимоги для сервера

Таблиця 3

Ресурс Мінімальні вимоги Рекомендовані вимоги		
Операційна система	Linux (Ubuntu, CentOS), Windows Server	Linux (сучасна версія)
Node.js	Версія 16 і вище	Остання стабільна версія
npm/yarn	Остання версія	Остання версія
MongoDB	Будь-яка підтримувана версія	Остання стабільна версія

Програмне забезпечення

4.2.1 Серверна частина (Backend):

Серверна частина, розроблена з використанням Node.js та фреймворку

Fastify, а також бази даних MongoDB.

4.2.2 Клієнтська частина (Frontend):

Клієнтська частина, розроблена з використанням React JS та UI бібліотеки Ant Design, є вебзастосунком, який виконується у веббраузері користувача. Тому особливих апаратних та програмних вимог до клієнтських пристроїв немає, окрім наявності сучасного веббраузера з підтримкою JavaScript.

Рекомендовані вимоги для клієнтських пристроїв:

- Сучасний персональний комп'ютер, ноутбук, планшет або смартфон.
- Стабільне підключення до Інтернету.
- Сучасний веббраузер (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge).

4.2.3 Мережеві вимоги:

Для забезпечення стабільної роботи платформи необхідне надійне інтернет з'єднання як для серверної частини, так і для користувачів.

- **Серверна частина:** Рекомендована швидкість інтернет-з'єднання - не менше 1 Гбіт/с для забезпечення швидкої обробки запитів та передачі даних.
- **Клієнтська частина:** Мінімальна рекомендована швидкість інтернет з'єднання для користувачів - 1 Мбіт/с, проте для комфортної роботи з завантаженням зображень рекомендується 5 Мбіт/с і вище.

4.3 Опис роботи програми

На початку взаємодії з вебплатформою C2C-торгівлі вторинною сировиною, користувач потрапляє на головний екран системи. Цей екран є початковою точкою входу до платформи і надає базову інформацію та

функціональність для нових і зареєстрованих користувачів. Вікно головного екрану платформи зображено на рисунку 9. У цьому вікні користувачеві пропонується виконати одну з двох ключових дій — авторизуватися за допомогою логіну та пароля, якщо вже є зареєстрований обліковий запис, або створити нову реєстрацію, якщо користувач вперше взаємодіє з платформою.

Також на головному екрані наявна кнопка для створення оголошення про продаж чи купівлю вторинної сировини. Однак варто зазначити, що при спробі створення оголошення без попередньої авторизації або реєстрації система автоматично перенаправляє користувача на сторінку входу (рис. 10). Це зроблено для забезпечення контролю доступу, а також для безпеки публікацій, аби лише верифіковані користувачі мали змогу додавати інформацію до бази оголошень.

Процес авторизації передбачає введення унікального логіну та пароля. Для різних типів користувачів — звичайних «Користувачів» та адміністративних «Адміністраторів» — облікові записи різняться, як за рівнем доступу, так і за призначенням функціоналу. Наприклад, користувач має доступ до особистого кабінету, створення та редагування оголошень,

перегляду історії взаємодій, тоді як адміністратор може модератувати оголошення, керувати списком користувачів, переглядати статистику активності на платформі тощо.

Подальші етапи роботи з платформою детально продемонстровані на рисунках 9–15. Ці кроки ілюструють основні сценарії використання системи, починаючи від входу та реєстрації, створення оголошення, перегляду й пошуку пропозицій, до управління особистими повідомленнями, а також модерації та адміністративного контролю. Таким чином, користувачі отримують цілісне уявлення про функціональні можливості системи, а також розуміння того, як платформа підтримує зручну, безпечну та ефективну взаємодію між продавцями та покупцями вторинної сировини.

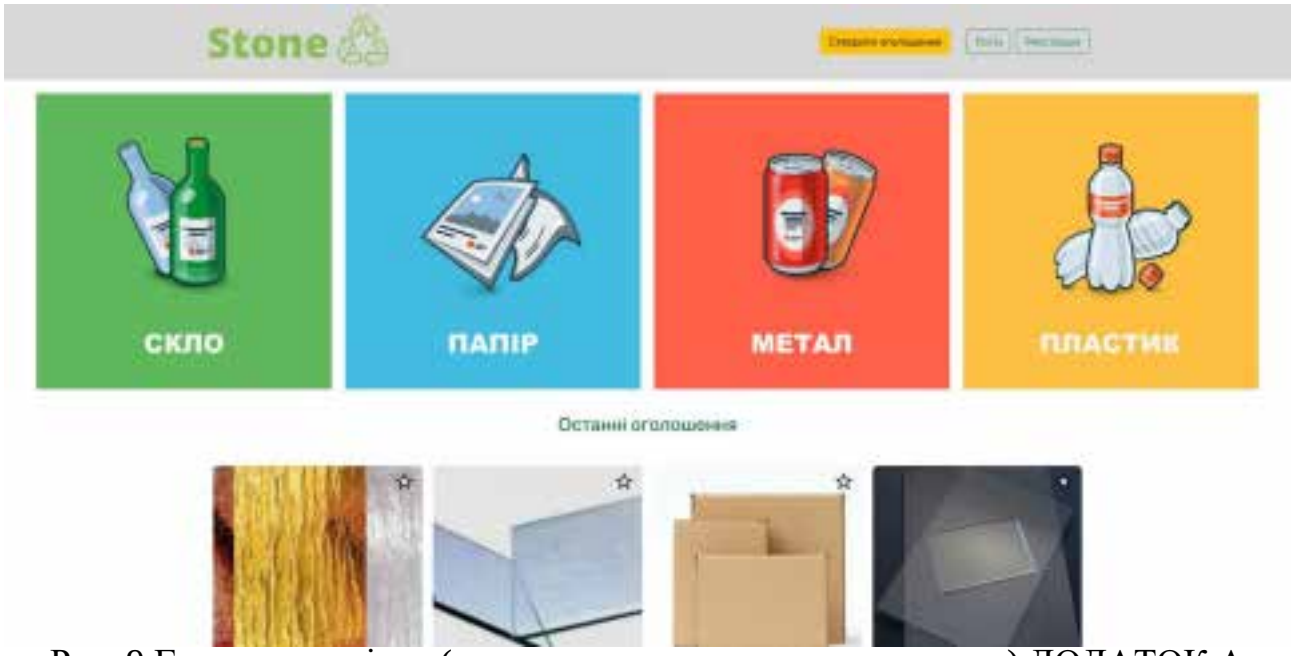


Рис. 9 Головна сторінка (для неавторизованого користувача) ДОДАТОК А

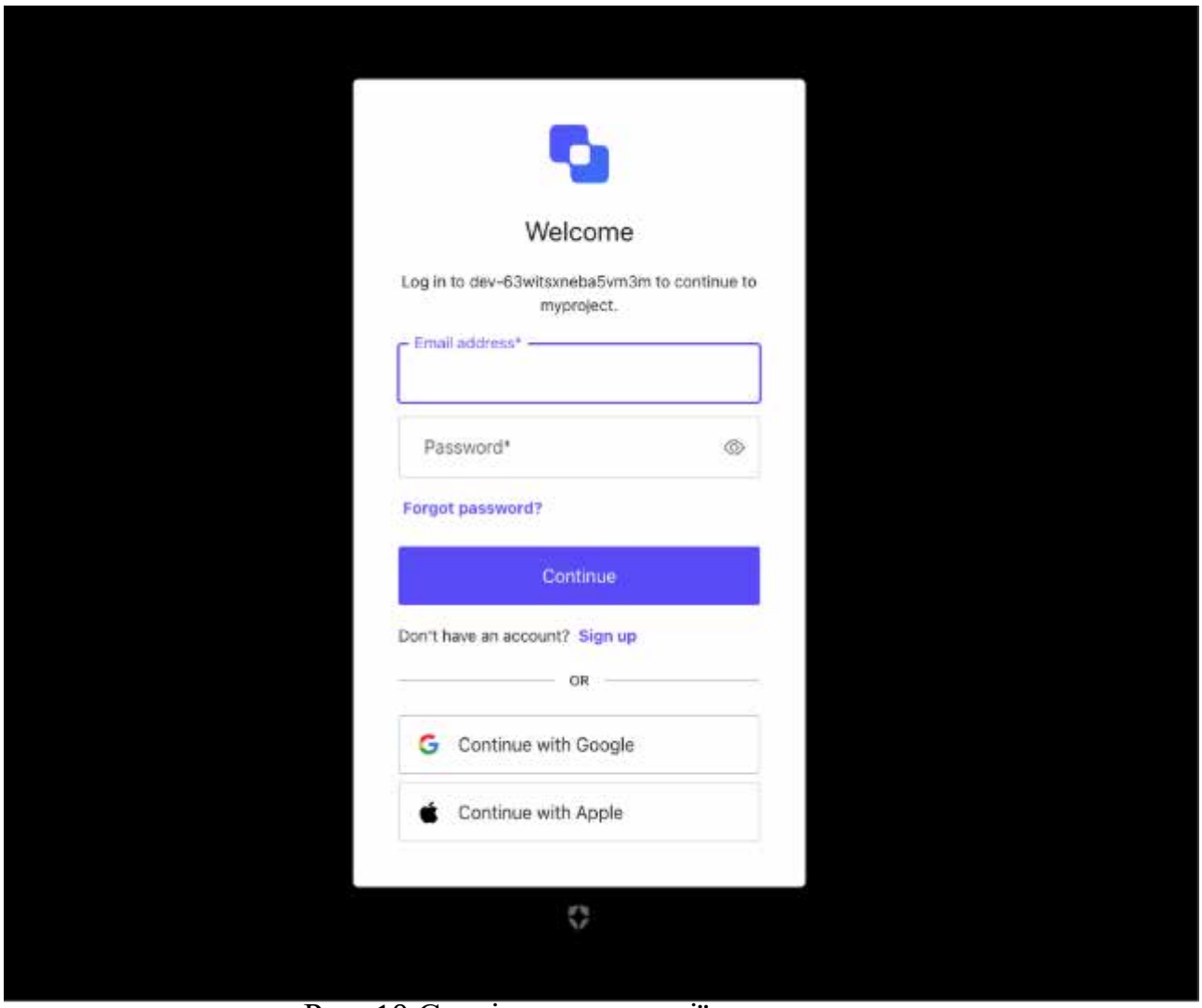


Рис. 10 Сторінка реєстрації користувача

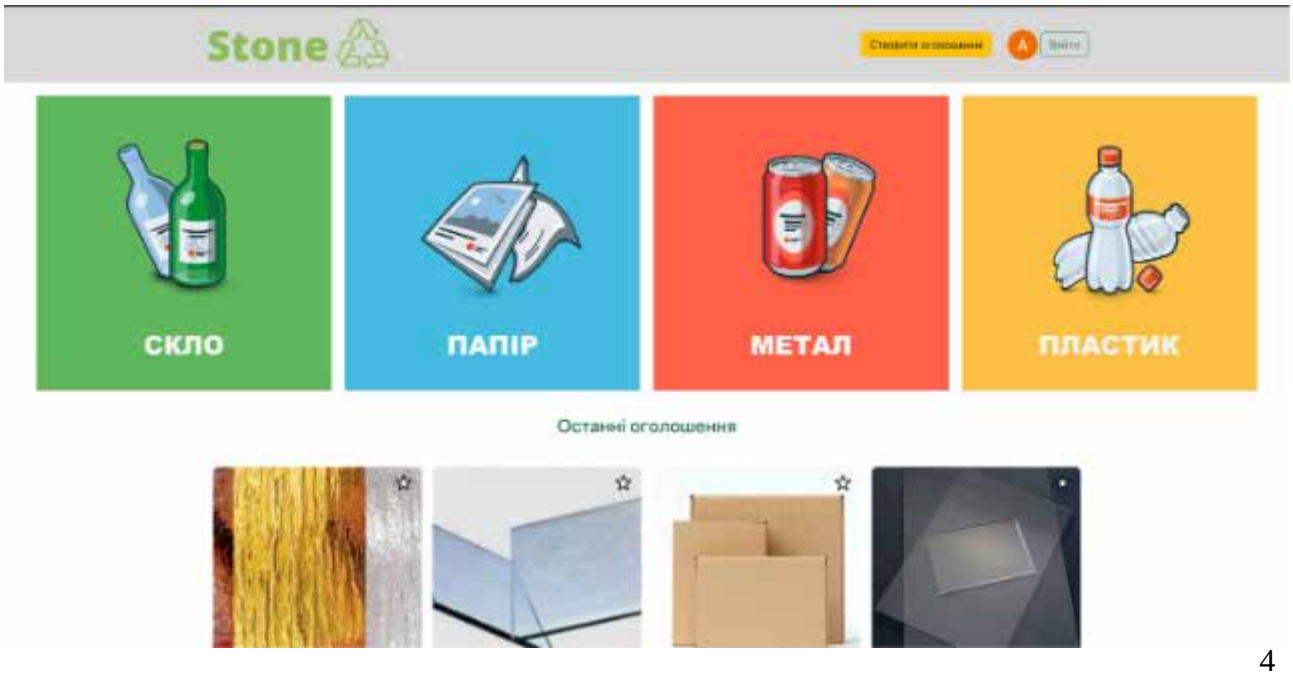


Рис. 11 Головна сторінка (для авторизованого користувача)

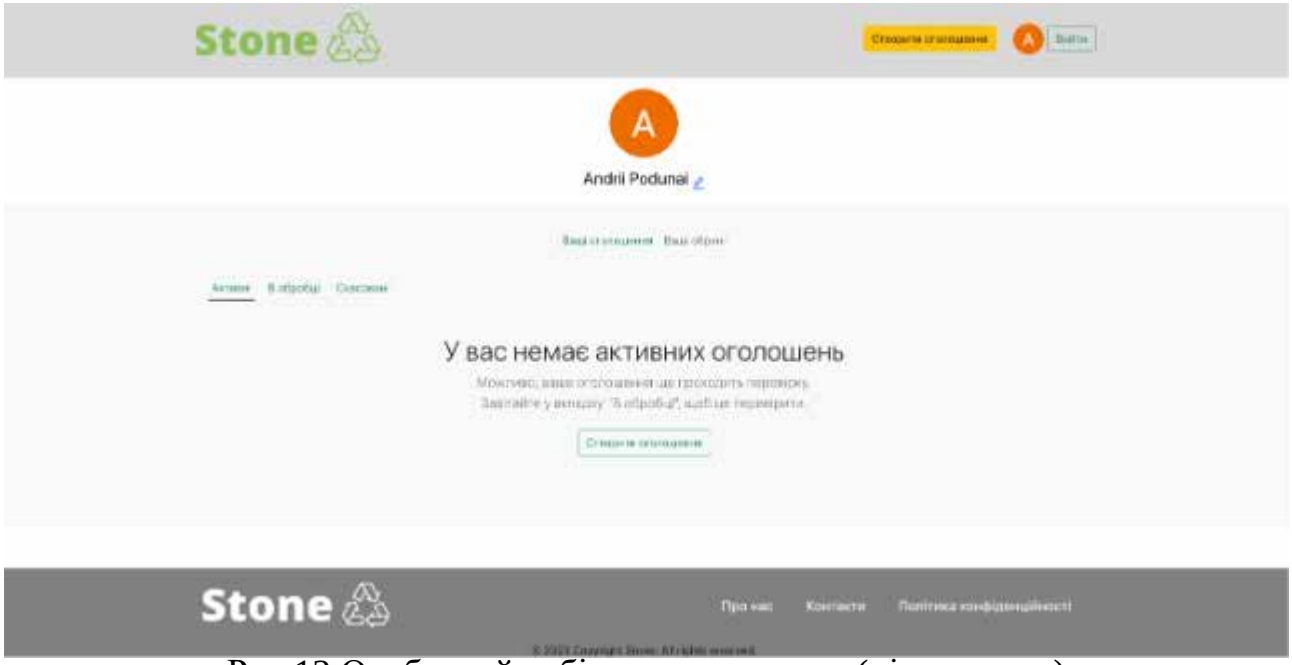


Рис.12 Особистий кабінет користувача (після входу)

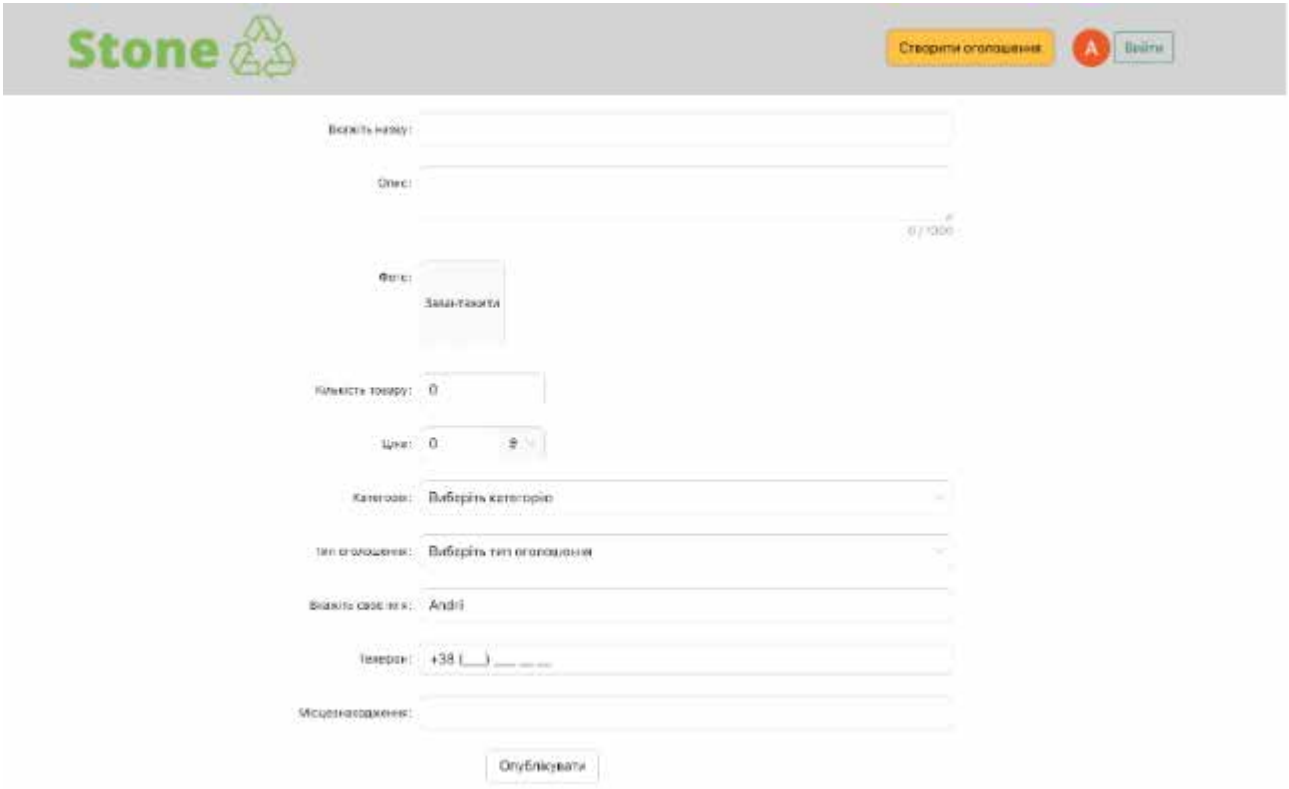


Рис 13. Форма додавання нового оголошення (ДОДАТОК Б)

Рис. 14 Сторінка перегляду оголошення (детальна інформація)

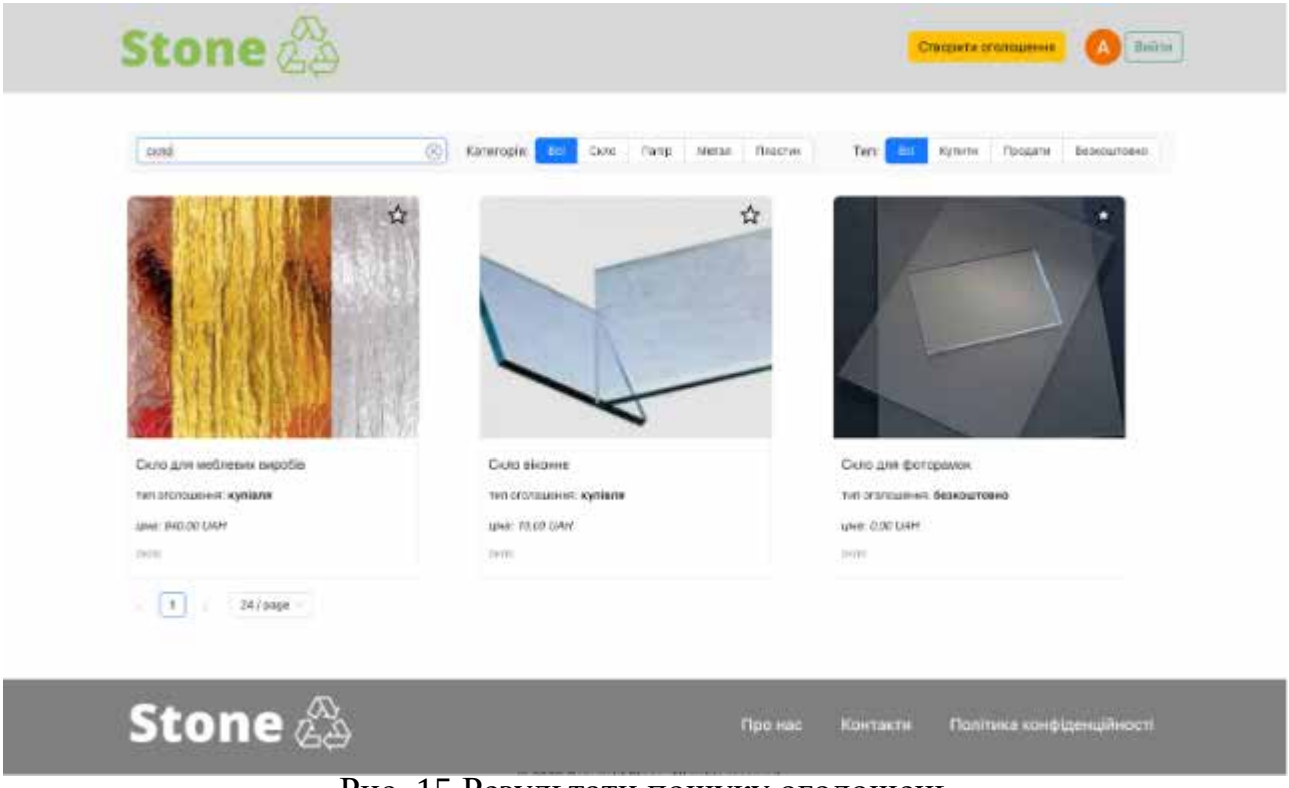


Рис. 15 Результати пошуку оголошень

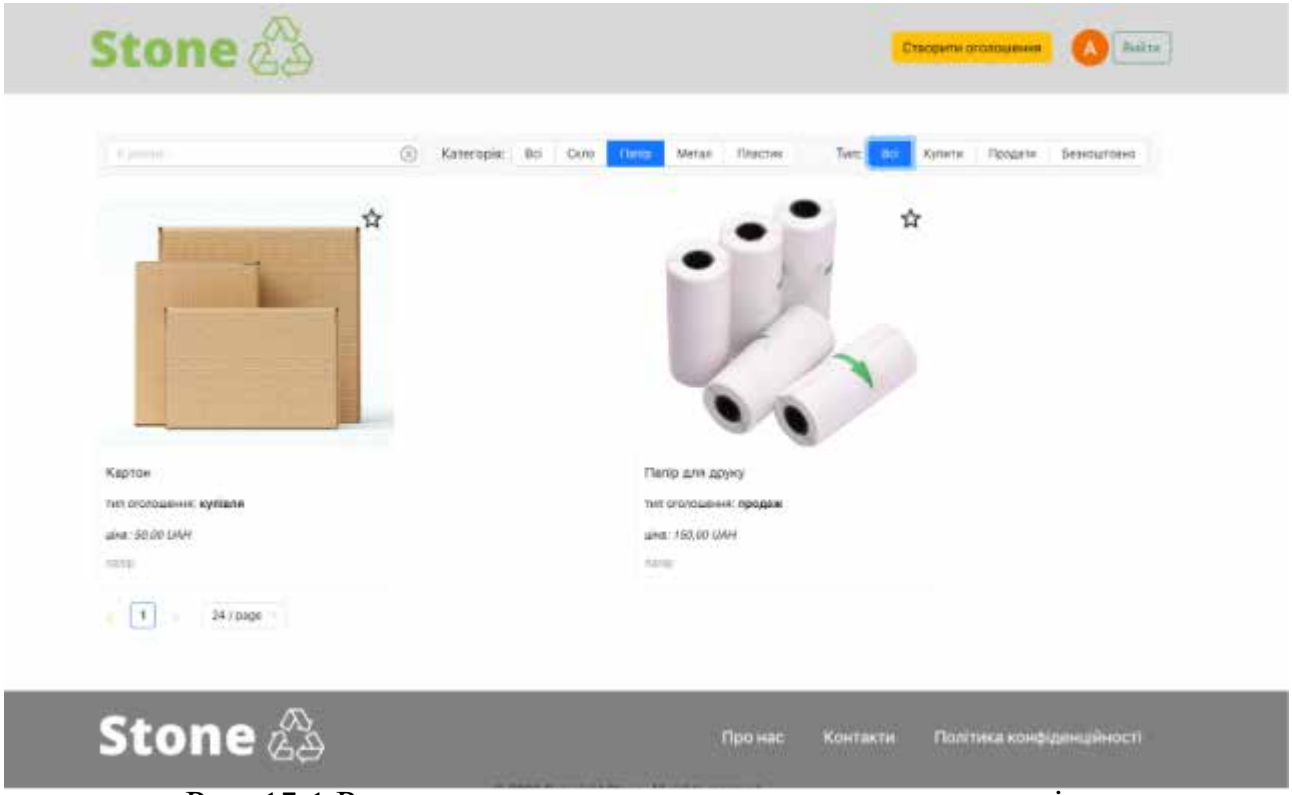


Рис. 15.1 Результати сортування оголошень за категорією

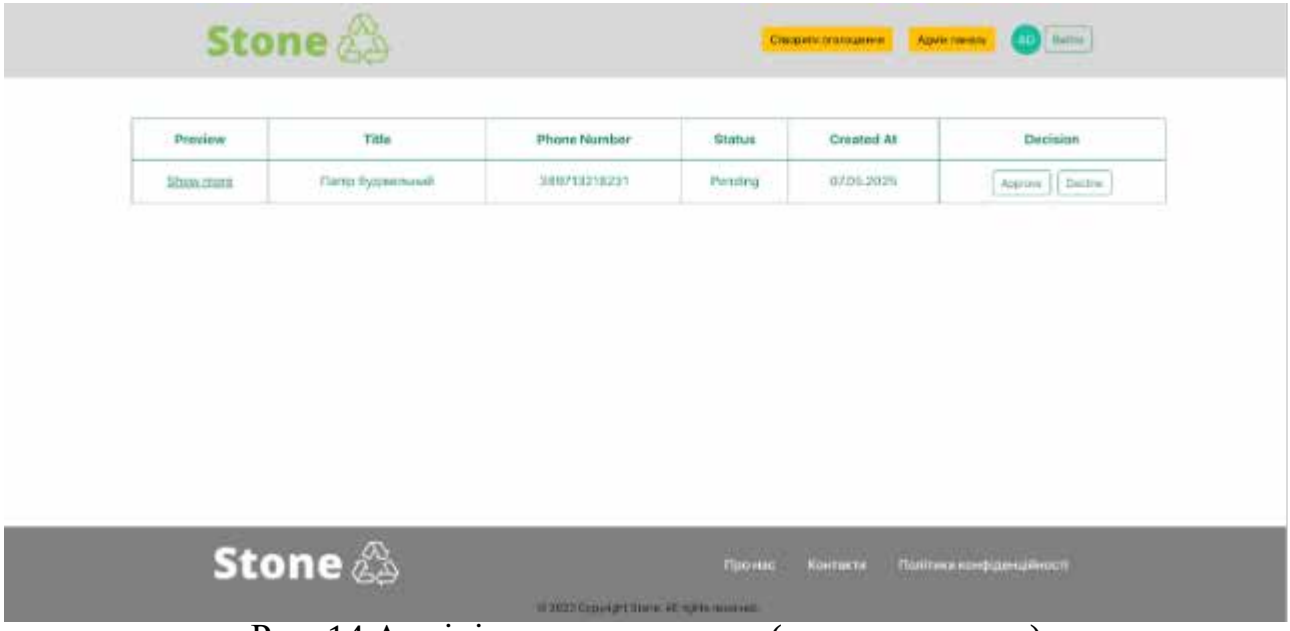


Рис. 14 Адміністративна панель (для модератора)

Рис.15 Процес редагування оголошення (для автора оголошення)

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було розроблено вебплатформу для C2C продажу вторинної сировини. Програмне забезпечення розроблялося у відповідності до поставлених завдань. Була спроєктована гнучка та масштабована інформаційна база даних, що відповідає особливостям

предметної області.

Розроблена вебплатформа дозволяє користувачам зручно розміщувати оголошення про продаж вторинної сировини, знаходити необхідні матеріали, а також безпосередньо зв'язуватися з продавцями для здійснення угод. Платформа забезпечує базовий функціонал для C2C взаємодії у сфері відновлюваних ресурсів.

Для виконання роботи було обрано наступний стек технологій: 1. Для розробки клієнтської частини було використано бібліотеку **React JS** та UI-бібліотеку **Ant Design**. React забезпечив компонентний підхід до розробки інтерфейсу, а Ant Design надав набір готових стилізованих компонентів для швидкого створення зручного та сучасного користувацького інтерфейсу.

2. Для розробки серверної частини було використано фреймворк **Fastify** на базі **Node.js**. Fastify забезпечив високу продуктивність та ефективну обробку запитів, а Node.js дозволив використовувати JavaScript на сервері.
3. Для зберігання даних було обрано NoSQL базу даних **MongoDB**, яка завдяки своїй гнучкій схемі добре підходить для зберігання різноманітних оголошень з різними характеристиками.
4. Для автентифікації та авторизації користувачів використовувалася бібліотека **Auth0-verify**, що забезпечило безпечний доступ до функціоналу платформи.
5. Для інтеграції з хмарним сервісом зберігання файлів **AWS S3** використовувався **AWS-sdk**, що дозволило ефективно керувати завантаженням та зберіганням зображень оголошень.
6. Середовищем розробки було обрано **Microsoft Visual Studio**, яке забезпечило зручні інструменти для розробки, відладки та керування проєктом на всіх етапах.

У першому розділі бакалаврської кваліфікаційної роботи було визначено мету та завдання проєкту, проведено аналіз предметної області C2C торгівлі

вторинною сировиною, а також сформульовано функціональні та нефункціональні вимоги до розроблюваної платформи.

Другий розділ був присвячений вибору інформаційного забезпечення. Було обґрунтовано вибір NoSQL бази даних MongoDB, враховуючи гнучкість схеми, масштабованість та продуктивність, що є важливими для даного типу платформи.

У третьому розділі було здійснено вибір інструментарію для розробки програмного забезпечення. Було обґрунтовано вибір React JS для фронтенду, Fastify/Node.js для бекенду, MongoDB для бази даних, Auth0-verify для автентифікації, AWS-sdk для роботи з хмарним сховищем та Microsoft Visual Studio як IDE. Також було описано організаційну структуру розробленого програмного забезпечення, що включає клієнтську та серверну частини.

У четвертому розділі було розглянуто принципи тестування програмного забезпечення та описано підходи до тестування розробленої C2C платформи. Було визначено основні сценарії тестування, включаючи функціональне тестування, тестування користувацького інтерфейсу, тестування продуктивності та безпеки. Також було описано апаратні та технічні засоби, необхідні для розгортання та функціонування платформи.

Результатом виконаної роботи стала розроблена вебплатформа для C2C продажу вторинної сировини, яка реалізує основні функції, необхідні для взаємодії користувачів у цій сфері. Розроблена система є базою для подальшого розвитку та вдосконалення функціоналу відповідно до потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React. The JavaScript library for building user interfaces:

https://react.dev/. (дата звернення: 07.04.2025)
2. Ant Design: The world's second most popular React UI framework: https://ant.design/. (дата звернення: 10.04.2025)
3. Fastify: Fast and low overhead web framework, for Node.js: https://www.fastify.io/. (дата звернення: 11.04.2025)
4. Node.js: https://nodejs.org/. (дата звернення: 10.04.2025)
5. MongoDB: The developer data platform: https://www.mongodb.com/. • MongoDB Documentation: https://www.mongodb.com/docs/. • Auth0: Identity, authentication, and authorization built for developers: https://auth0.com/. (дата звернення: 18.04.2025)
6. AWS SDK for JavaScript: https://aws.amazon.com/sdk-for-javascript/. (дата звернення: 20.04.2025)
7. Amazon Simple Storage Service (S3): https://aws.amazon.com/s3/. (дата звернення: 21.04.2025)
8. Microsoft Visual Studio: https://visualstudio.microsoft.com/. (дата звернення: 09.04.2025)

Тестування вебзастосунків. (дата звернення: 13.05.2025)
   a. https://www.google.com/search?q=https://react.dev/learn/testing
   b. **Книга:** Kent C. Dodds. Testing JavaScript: Applications.
9. Тестування ПЗ (види тестування) – [Електронний ресурс] – режим доступу: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1 (дата звернення: 15.05.2025)
10. Hardware and software requirements: Overview, definition, and example – режим доступу: https://www.cobrief.app/resources/legal-glossary/hardware-and-software-requirements-overview-definition-and-example/ (дата звернення: 17.05.2025)

ДОДАТОК А

```
import { Link } from 'react-router-dom';
import { Col, Row } from 'antd';
import emptyImg from 'images/emptyImage.png';
import './style.scss';
```



```
import ProductCard from 'components/ProductCard';
import { useUserToken, useFavorites, useCards } from 'rest';

function Home() {
 const [token] = useUserToken();
 const { favorites } = useFavorites(token);
 const { cards, errorCards } = useCards(1, 12);

 return (
 <div className="container-main">
 <Row style={{ marginTop: '10px' }}>
 <Col span={6}>
 <Link to="/products?category=glass">
 <img src="./img/glass.jpg" alt="Glass" className="Category"></img>
 </Link>
 </Col>
```



```
 <Col span={6}>
 <Link to="/products?category=paper">
 <img src="./img/paper.jpg" alt="Paper" className="Category"></img>
 </Link>
 </Col>
 <Col span={6}>
 <Link to="/products?category=metal">
 <img src="./img/metal.jpg" alt="Metal" className="Category"></img>
 </Link>
 </Col>
 <Col span={6}>
 <Link to="/products?category=plastic">
 <img src="./img/plastic.jpg" alt="Plastic" className="Category"></img>
 </Link>
 </Col>
 </Row>
 <h3 className="PageHeader">Останні оголошення</h3>
 <ul className="container row mx-auto gap-3 list-unstyled py-3">
 {!errorCards &&
 cards.map((elem) => {
 const image =
 elem.images.length > 0 && elem.images[0].url ? elem.images[0].url : emptyImg;
 let favorite = false;
 if (favorites !== false) {
 favorite = favorites.includes(elem._id);
 }
 return (
 <li key={elem._id} className="col p-0 d-flex justify-content-center">
 <Link className="w-100 text-decoration-none" to={`/products/${elem._id}`}>
```

```
<ProductCard
id={elem._id}
price={elem.price}
category={elem.category}
image={image}
```



```
title={elem.title}
currency={elem.currency}
type={elem.type}
favorite={favorite}
userToken={token}
/>
</Link>
</li>
);
})}
</ul>
<Link to={'/products?category=all'}>
<p className="link-success">До всіх оголошень</p>
</Link>
</div>
);
}

export default Home
```

ДОДАТОК Б

Сторінка 1

```
import React, { useState } from 'react';
import { Form, Input, Button, Select, Upload, message } from 'antd';
import { Formik } from 'formik';
import schema from './validationSchema';
import { PatternFormat } from 'react-number-format';
import { SERVER_URL } from 'variables';
import convertNumber from './numberConverter';
import { useDeleteImage } from 'rest';

const { TextArea } = Input;
const { Option } = Select;

function ProductForm({ initialValues, submit }) {
 const { reRenderDelImage } = useDeleteImage();
 const [fileList, setFileList] = useState(
 initialValues.images.map((el) => {
 return {
 response: [el],
 url: el.url,
 status: 'done',
 };
```

```
})
);

function onChangeUpload(info) {

if (info.file.status === 'removed') {
if (info.file.response[0]) {
reRenderDelImage([info.file.response[0].key]);
}
}
if (info.file.status === 'error') {
message.error(`${info.file.key} file upload failed.`);
}
setFileList((prevFileList) => {
return info.fileList.map((file) => {
const mistake = prevFileList.find(
(prevFile) =>
prevFile.uid === file.uid && prevFile.status === 'done' && file.status === 'uploading'
);
if (mistake) {
return { ...file, status: 'done', response: mistake.response };
}
```



```
return file;
});
});
}

async function onPreviewUpload(file) {
let src = file.url;
if (!src) {
src = await new Promise((resolve) => {
const reader = new FileReader();
reader.readAsDataURL(file.originFileObj);
reader.onload = () => resolve(reader.result);
});
}
const image = new Image();
image.src = src;
const imgWindow = window.open(src);
imgWindow?.document.write(image.outerHTML);
}

function onSubmitForm(value) {
if (typeof value.phoneNumber === 'string') {
value.phoneNumber = convertNumber(value.phoneNumber);
}
const images = fileList.filter(({ status }) => status === 'done');
value.images = images.map((value) => value.response[0]);
submit(value);
}
```

```
return (
<Formik
initialValues={initialValues}
validationSchema={schema}

onSubmit={(value) => onSubmitForm(value)}>
{({ handleSubmit, touched, errors, getFieldProps, values, setFieldValue }) => (
<Form
className="ps-2 pe-2"
labelCol={{ span: 9 }}
wrapperCol={{ span: 8 }}
onFinish={handleSubmit}
layout="horizontal"
size="large"
autoComplete="off"
style={{ marginTop: 20 }}>
<Form.Item
label="Вкажіть назву"
htmlFor="title"
help={touched.title && errors.title ? errors.title : ''}
validateStatus={touched.title && errors.title ? 'error' : undefined}>
<Input {...getFieldProps('title')} />
</Form.Item>
```



```
<Form.Item
label="Опис"

help={touched.description && errors.description ? errors.description : ''}
validateStatus={touched.description && errors.description ? 'error' : undefined}>
<TextArea showCount maxLength={1000} {...getFieldProps('description')} />
</Form.Item>

<Form.Item
label="Фото"
valuePropName="fileList"
htmlFor="images"
help={touched.images && errors.images ? errors.images : ''}
validateStatus={touched.images && errors.images ? 'error' : undefined}>
<Upload
{...getFieldProps('images')}
fileList={fileList}
name="files[]"
action={SERVER_URL + '/upload'}
accept="image/*"
listType="picture-card"
multiple
onChange={(info) => onChangeUpload(info)}
onPreview={(file) => onPreviewUpload(file)}>
{values.images.length < 12 && 'Завантажити'}
</Upload>
</Form.Item>
```

```
<Form.Item
label="Кількість товару"
htmlFor="count"
help={touched.count && errors.count ? errors.count : ''}

validateStatus={touched.count && errors.count ? 'error' : undefined}>
<Input type="number" min="0" {...getFieldProps('count')} style={{ width: 150 }} />
</Form.Item>

<Form.Item
label="Ціна"
htmlFor="price"
help={touched.price && errors.price ? errors.price : ''}
validateStatus={touched.price && errors.price ? 'error' : undefined}>
<Input
addonAfter={
<Select
{...getFieldProps('currency')}
onChange={(value) => setFieldValue('currency', value)}>
<Option value="USD">$</Option>
<Option value="EUR">€</Option>
<Option value="UAH">₴</Option>
</Select>
}
```



```
type="number"
min="0"
{...getFieldProps('price')}
style={{ width: 150 }}
/>
</Form.Item>

<Form.Item
label="Категорія"
htmlFor="category"
help={touched.category && errors.category ? errors.category : ''}
validateStatus={touched.category && errors.category ? 'error' : undefined}>
<Select
{...getFieldProps('category')}
onChange={(value) => setFieldValue('category', value)}>
<Option disabled value="">
Виберіть категорію
</Option>
<Option value="plastic">Пластик</Option>
<Option value="metal">Метал</Option>
<Option value="paper">Папір</Option>
<Option value="glass">Скло</Option>
</Select>
</Form.Item>

<Form.Item
label="Тип оголошення"
htmlFor="type"
```

```
help={touched.type && errors.type ? errors.type : ''}
validateStatus={touched.type && errors.type ? 'error' : undefined}>
<Select {...getFieldProps('type')} onChange={(value) => setFieldValue('type', value)}>
<Option disabled value="">
Виберіть тип оголошення
</Option>
<Option value="buy">Купити</Option>
<Option value="sell">Продати</Option>
<Option value="free">Безкоштовно</Option>
</Select>
</Form.Item>

<Form.Item
label="Вкажіть своє ім'я"
htmlFor="name"
help={touched.name && errors.name ? errors.name : ''}
validateStatus={touched.name && errors.name ? 'error' : undefined}>
<Input {...getFieldProps('name')} />
</Form.Item>

<Form.Item
label="Телефон"
htmlFor="phoneNumber"
help={touched.phoneNumber && errors.phoneNumber ? errors.phoneNumber : ''}
```



```
validateStatus={touched.phoneNumber && errors.phoneNumber ? 'error' : undefined}>
<PatternFormat
type="tel"
className={`form-control${touched.phone && errors.phone ? ' is-invalid' : ''}`}
{...getFieldProps('phoneNumber')}
format="+38 (###) ### ## ##"
mask="_"
/>
</Form.Item>

<Form.Item
label="Місцезнаходження"
htmlFor="location"
help={touched.location && errors.location ? errors.location : ''}
validateStatus={touched.location && errors.location ? 'error' : undefined}>
<Input {...getFieldProps('location')} />
</Form.Item>

<Form.Item wrapperCol={{ offset: 10, span: 16 }}>
<Button type="text" htmlType="submit" style={{ border: '1px solid #d9d9d9' }}>
Опублікувати
</Button>
</Form.Item>
</Form>
)}
</Formik>
);
```

```
}

export default ProductForm;
```