Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для обміну речами. Back-end.

(тема)

Виконала:
студентка 4 курсу, групи ПЗПІ-20-2

Сорокіна Д. Є.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доцент. кафедри ПІ Ворочек О. Г.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З. В. Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Сорокіній Дар’ї Євгенівні
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для обміну речами. Back-end.
Затверджена наказом по університету від 20.05.2024 р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2024

3. Вихідні дані до роботи Розробити серверну частину програмної системи для обміну речами з використанням мови C# та технології ASP.NET Core 8.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.04.2024	*виконано*
2	Створення специфікації ПЗ	22.04.2024	*виконано*
3	Проектування ПЗ	29.04.2024	*виконано*
4	Розробка ПЗ	06.04.2024	*виконано*
5	Тестування ПЗ	13.05.2024	*виконано*
6	Написання пояснювальної записки	07.06.2024	*виконано*
7	Підготовка презентації та доповіді	10.06.2024	*виконано*
8	Нормоконтроль, рецензування	14.06.2024	*виконано*
9	Здача роботи у електронний архів	14.05.2024	*виконано*
10	Попередній захист	16.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	19.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка) _____ Сорокіна Д. Є.
(підпис)

Керівник роботи _____ доцент кафедри ПІ Ворочек О. Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 61 с., 37 рис., 1 табл., 5 додатків, 10 джерел.

БАРТЕР, КОРИСТУВАЧ, ОБМІН, ПРОГРАМНА СИСТЕМА, ТОВАР, УГОДА, ASP.NET, C#, VISUAL STUDIO

Об’єкт розробки – серверна частина для програмної системи для обміну речами.

Мета розробки – створення зручної та ефективної платформи, яка спростить процес пошуку речей та допоможе позбутися речей, які не потрібні, шляхом обміну.

Метод рішення – середовище розробки Visual Studio, мова програмування C#, фреймворк ASP.NET Core 8.

У результаті роботи було спроектовано та розроблено бек-енд програмної системи для обміну речами, яка допомагає з’єднати людей, які потребують обміну, надаючи їм зручні інструменти для цього.

BARTER, USER, EXCHANGE, SOFTWARE SYSTEM, ITEM, DEAL, ASP.NET, C#, VISUAL STUDIO

The object of development is the server part for a software system for exchanging things.

The development purpose is to create a convenient and efficient platform that will simplify the process of finding things and help you get rid of things you don’t need by exchanging them.

The solution method was Visual Studio development environment, C# programming language, ASP.NET Core 8 framework.

As a result, I designed and developed the back-end of a software system for exchanging things, which helps connect people who need to exchange things by providing

them with convenient tools for doing so.

Я, Сорокіна Дар’я Євгенівна, студентка гр. ПЗПІ-20-2, здобувачка вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для обміну речами. Бек-енд.», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі, зростаюча увага до екологічних питань та усвідомленого використання ресурсів призвела до того, що люди заміняють покупку нових товарів придбанням, обміном та використанням вживаних речей. Проте, на поточному ринку майже немає платформ, які спеціалізуються на бартері різними товарами, а тим паче, які роблять даний процес зручним і головне – простим.

Темою кваліфікаційної роботи є серверна частина для програмної системи для обміну речами. Основне завдання програмної системи – надати зручний та інтуїтивно простий інструмент для бартеру речами.

Основною метою роботи є розробка бек-енду для платформи для обміну речами, яка діятиме як ефективний посередник між людьми, надаючи зручний, легкий та безпечний обмін речами. Система забезпечуватиме простий процес оформлення оголошень про речі для обміну, пошуку та фільтрації необхідних предметів, а також організації зручного процесу обміну між користувачами.

Завдання даної роботи полягають в:

- аналізі існуючих аналогів та технологій, що використовуються для реалізації подібних рішень;
- проектуванні архітектури серверної частини програмної системи з використанням принципів чистої архітектури;
- реалізації бек-енду за допомогою C# та фреймворку ASP.NET Core 8;
- тестуванні роботи даної частини та перевірці на відповідність вимогам.

Дана платформа має потенціал спростити та оптимізувати процес обміну між користувачами. Вона сприятиме формуванню більш стійких, економічно ефективних та екологічно збалансованих спільнот. Це сприятиме зменшенню обсягів споживання та подальшому розвитку культури обміну. Програмна система для обміну речами стане не просто технічним інструментом, але і стратегічним компонентом, що реагуватиме на зростаючі потреби користувачів [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

У сучасному світі все більше людей надають перевагу екологічному способу життя та усвідомлюють важливість вторинного використання та збереження ресурсів. Обмін речами – це процес, що набирає популярність, який передбачає передачу речей між особами без використання грошей. Обмін речами надає можливість продовжити життєвий цикл речей, зменшити відходи та зберегти наявні ресурси, що є надзвичайно актуальним та важливим. Також обмін є економічно вигідним рішенням як для індивідуальних споживачів, так і для суспільства – він допомагає економити гроші, які могли бути витрачені на нові покупки, та спричиняє зміну споживчих звичок.

З економічної точки зору обмін речами надає суттєву перевагу у вигляді отримання необхідних речей без витрати грошей, це особливо важливо та актуально для людей, які мають низькі доходи, наприклад студенти або багатодітні сім’ї. До того ж, обмін речами призводить до зменшення виробництва нових товарів тож зменшується використання ресурсів, енергії, які б пішли на їхнє виготовлення. У соціальному плані плюсом обміну речами є формування спільнот та соціальних зв’язків – люди, які беруть участь в обміні, взаємодіють один з одним, що в свою чергу будує взаємну довіру.

В свою чергу технологічний прогрес тільки допомагає цьому процесу розвиватися – завдяки появі та розвитку різних інтернет платформ та мобільних застосунків, обмін речами може стати зручнішим та доступнішим для більшості людей. Такі платформи надають можливість легко розміщувати оголошення про речі, які лежать без діла, шукати потрібні товари, спілкуватися з іншими учасниками та організовувати зустрічі для передачі речей.

Таким чином дана програмна система для обміну різноманітними товарами спрямована на створення середовища для легкого, безпечного та комфортного співробітництва між людьми, яке сприяє економії грошей, ресурсів та поліпшенню стану нашого навколишнього середовища.

1.2 Аналіз існуючих аналогів

Для успішного аналізу предметної галузі важливо знайти та розглянути існуючі аналоги програмної системи для обміну речами.

Представлені аналоги пропонують різні підходи та можливості, а також мають власні переваги та недоліки. Було знайдено три рішення, які вирішують ту саму проблему, що і наш проект і два, які містять подібне рішення, нижче описані їхні переваги та недоліки.

HaveNeed – мобільний застосунок для обміну товарами та послугами (див. рис. 1.1):

- представлений тільки у мобільному застосунку як для Android, так і iOS систем;
- має мінімалістичний, але не дуже сучасний дизайн;
- є можливість чату між користувачами;
- є пошук товарів, але відсутня фільтрація;
- одночасно для одного товару може бути багато пропозицій обміну;
- є оцінювання угод;
- обмежений локально, не доступний в Україні.

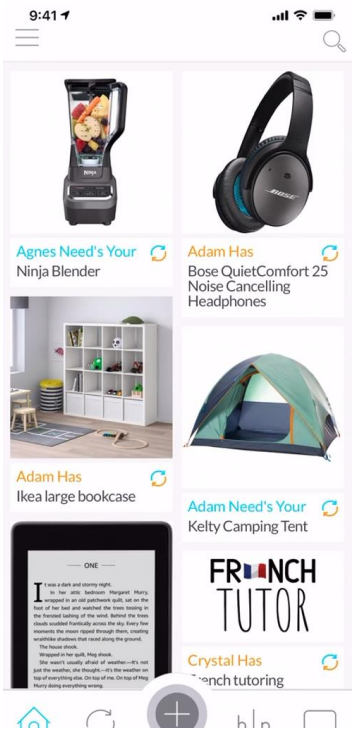

Рисунок 1.1 – Застосунок «HaveNeed»

TradeMade – мобільний застосунок для iOS системи для обміну речами та послугами (див. рис. 1.2):

– представлений тільки у мобільному застосунку тільки для iOS системи;
– має мінімалістичний сучасний дизайн;
– є і пошук товарів, і фільтрація;
– є чат між користувачами;
– є можливість обміну як товарами, так і послугами;
– є оцінка користувачів.

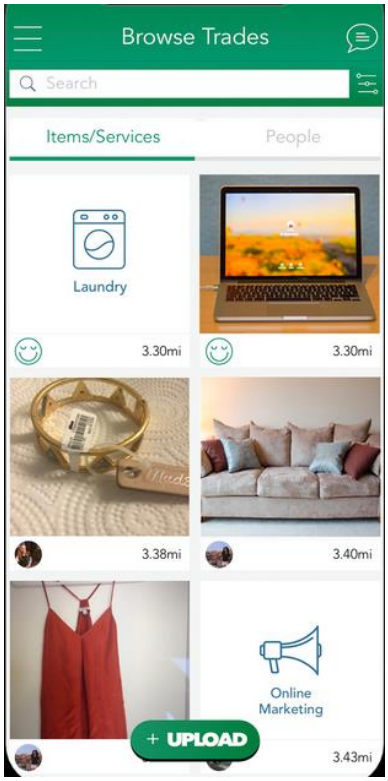

Рисунок 1.2 – Застосунок «TradeMade»

GoSwap – веб-платформа для обміну нерухомістю та транспортними засобами (див. рис. 1.3):

– представлений тільки у вигляді веб-сайту;
– має не сучасний не зручний дизайн;
– можливий обмін тільки нерухомістю та транспортними засобами;
– немає чату між користувачами.

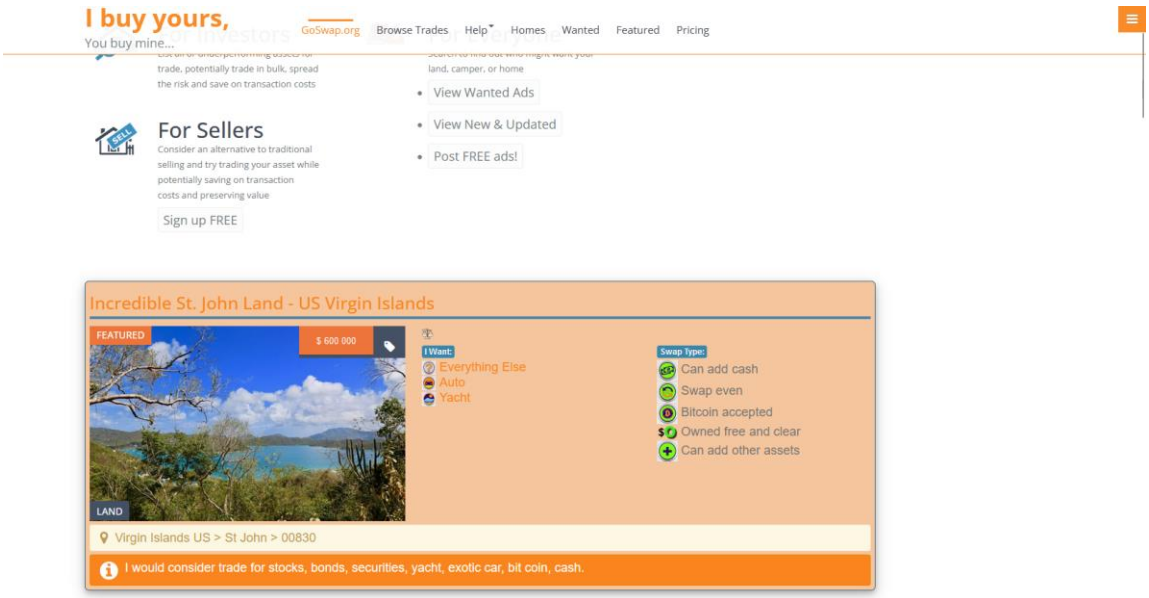

Рисунок 1.3 – Веб-платформа «GoSwap»

BarterQuest – веб-платформа для обміну всіма видами товарів, послуг та нерухомості (див. рис. 1.4) :

– представлений тільки у вигляді веб-сайту;
– має можливість обміну і послугами, і звичайними товарами, і нерухомістю;
– має не сучасний дизайн;
– є оцінка користувачів;
– є пошук, фільтрація та розділення за категоріями.

Рисунок 1.4 – Веб-майданчик «BarterQuest»

OLX – популярний торгівельний веб-майданчик, який спеціалізується на більш на торгівлі товарами, ніж на обміні (див. рис. 1.5):

– представлений як у вигляді мобільного застосунку, так і веб-сайту;
– май сучасний зручний дизайн;
– є пошук, фільтрація та розділення за категоріями;
– зосереджений більш на торгівлі товарами, ніж на обміні;
– є чат між користувачами.

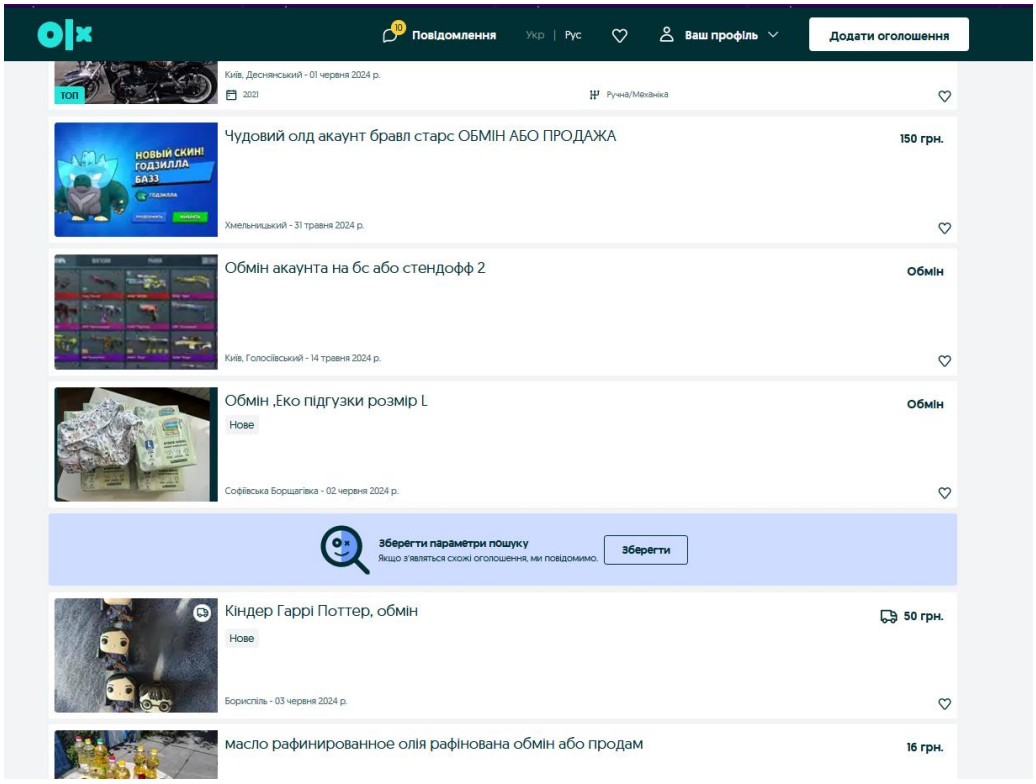

Рисунок 1.5 – Торгівельна веб-платформа «OLX»

Аналіз існуючих аналогів для системи для обміну речами показав, що представлені платформи пропонують великий спектр функцій, включаючи розміщення оголошень, пошук та фільтрацію товарів, комунікацію між користувачами та системи рейтингу для підвищення довіри. Кожна з платформ має свої особливості, орієнтована на певні категорії товарів та на певні цільові аудиторії. Це доводить важливість розробки гнучкої, зручної та інтуїтивно простої програмної системи, яка могла б задовольнити потреби різних користувачів.

1.3 Цілі та ризики проєкту

Цілі даного проєкту полягають у створенні програмної системи, яка відповідає потребам користувачів і сприяє отриманню досвіду безпечних та простих обмінів. Отже, основними цілями проєкту є:

- розробка інтуїтивно зрозумілої, зручної та безпечної системи, яка полегшить процес обміну речами між користувачами та зробить його максимально ефективним;
- створення платформи, яка дозволить користувачам створювати оголошення про товари для обміну, легко знаходити потрібні товари та влаштовувати обміни;
- підвищення свідомості щодо екологічних питань, зменшення впливу на навколишнє середовище шляхом збільшення вторинного використання речей та зменшення споживання нових товарів.

Розробка та впровадження програмної системи для обміну речами пов’язані з рядом певних ризиків, від яких залежить успішність створення даної системи. Отже, ризики проєкту полягають в:

- виникненні складнощів під час розробки та впровадженні системи, наприклад, з архітектурою, сумісністю з іншими програмними продуктами;
- можливості, що програмна система не здобуде достатньої популярності серед потенційних користувачів, що може спричинити обмеження обсягу обмінів та активності учасників;
- складнощах у забезпеченні повної безпеки та захисту особистих даних користувачів;
- виникненні проблем з комунікацією та взаємодією між учасниками обміну, що спричинить збільшення кількості неуспішних угод та незадоволеності користувачів.

Успішне управління ризиками передбачає їхню ідентифікацію, аналіз та постійний моніторинг. Врахування описаних ризиків на всіх етапах проєкту

допоможе забезпечити стабільну роботу системи та її популярність у користувачів.

1.4 Постановка задачі

Розробка програмної системи для обміну речами вимагає чіткої постановки задачі, яка охоплює всі складові, які потрібні для проєктування, створення, впровадження та використання системи.

Однією з основних задач серверної частини є забезпечення високої продуктивності для ефективної обробки великого потоку запитів від користувачів, а також здатність масштабуватися для вирішення зростаючих потреб в обслуговуванні користувачів та обробці даних. Додатково, безпека є критично важливим аспектом і система повинна мати вбудовані механізми захисту даних від несанкціонованого доступу та зловживань. Моніторинг та аналіз роботи системи допоможуть вчасно виявляти та вирішувати проблеми ефективності та надійності. Крім того, надання документації та підтримки для інших розробників є важливим аспектом для забезпечення продуктивної та ефективної роботи з системою.

Далі, треба спроектувати та реалізувати можливість реєстрації, авторизації та аутентифікації користувачів, можливість відновлення паролю, доступ до редагування особистої інформації. Також у користувачів має бути можливість створювати оголошення про товари для обміну, редагувати їхню інформацію, переглядати та видаляти власні оголошення. Крім того, мають бути реалізовані пошук та фільтрація товарів за певними параметрами. До того ж для розуміння, чи є надійним їхній партнер по обміну, користувачі мають мати можливість оцінювати один одного. Однією з головних функцій, також, має бути можливість обмінюватись повідомленнями, тобто наявність чату.

Таким чином, окреслення цілей та задач проєкту забезпечить їх вдале втілення та створення надійної та придатної програмної системи для обміну речами.

2 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Постановка мети

Провідною метою даної роботи є створення серверної частини для програмної системи для обміну.

Вона буде розроблена за допомогою мови C# та фреймворку ASP.NET Core 8 [2]. Також у якості допоміжних ресурсів плануються використовуватись такі бібліотеки як MediatR, FluentResults, FluentValidation.

Архітектура даної програмної системи базується на шаблоні проектування – чистій архітектурі, мета якого створити структуру, яка дозволяє легко керувати та підтримувати додаток, коли він росте та змінюється з часом [3]. Система буде поділена на чотири основні шари: рівень домену, який відповідає за бізнес-логіку та сутності програми, рівень програми, який діє як посередник між рівнем домену та зовнішніми інтерфейсами програми, рівень інфраструктури, який реалізує технічні деталі програми, та презентаційний рівень, який відповідає за надсилання відповідей і отримання запитів користувачів.

2.2 Загальний опис системи

Програмна система для обміну речами – це інтернет-платформа, яка надає можливість бажаючим легко та зручно обмінюватись предметами з іншими, в яких більше немає потреби і при цьому отримувати бажані речі без витрати грошей.

Серверна частина є основою системи, яка обробляє запити від клієнтської частини, забезпечує роботу з базою даних та відповідає за реалізацію бізнес-логіки. Метою бекенду програмної системи для обміну речами є забезпечення стабільної та надійної роботи цієї системи, підтримуючи продуктивність та безпеку даних користувачів.

Платформа надаватиме можливість створення оголошень з товарам, якими користувач бажає обмінятися з іншими користувачами системи, пошуку потрібних товарів і додаткової фільтрації для більш ефективного пошуку, можливість створення обмінних угод, а також чат для спілкування з іншими користувачами.

2.3 Основна функціональність системи

Система налічуватиме дві ролі користувачів – звичайний користувач (user) та адміністратор (admin).

Функціонал, доступний звичайному користувачу:

- реєстрація у системі;
- авторизація та автентифікація;
- редагування особистих даних;
- додавання оголошень про товари;
- редагування інформації товарів;
- видалення оголошень;
- пошук товарів;
- фільтрація товарів за певними параметрами;
- просувати товар;
- чат з іншими користувачами;
- створення обмінних угод;
- оцінювання обмінних угод.

Функціонал, доступний адміністратору:

- авторизація та автентифікація;
- створення категорій;
- видалення категорій;
- додавання кольорів;
- видалення кольорів.

Серверна частина буде відповідати за захист даних, обробку дій користувачів. Для цього будуть використовуватись сучасні методи шифрування даних, а також надійні механізми автентифікації та авторизації для запобігання несанкціонованого доступу.

Загалом, система забезпечить достатню кількість функцій, зручність, продуктивність та надійність, що дозволить їй задовільнити потреби великої кількості користувачів.

2.4 Загальні обмеження

Для забезпечення стабільної та безпечної роботи програмної системи для обміну речами, користувачі мають дотримуватися певних обмежень та правил. Тож серверна частина даної системи має такі обмеження:

- користувачі можуть завантажити до 5 зображень для свого товару;
- певні поля, наприклад, назва товару має містити мінімум 10 та максимум 70 символів;
- користувач має пройти процес авторизації та аутентифікації, щоб отримати доступ до майже усіх функцій;
- до функцій, які пов’язані з категоріями та кольорами, має доступ тільки користувач з роллю адміністратора;
- деякі поля є обов’язковими для заповнення.

Дотримання цих обмежень є необхідним для стабільного та ефективного користування системою. Це забезпечить збереження високих стандартів обслуговування та безпеки на платформі, що є важливим для її успішного існування та розвитку.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 UML-проєктування програмної системи

Діаграма прецедентів [4] була створена під час проектування програмної системи та її детального аналізу. Вона описує взаємодію користувача з системою, які він має можливості та функції, які надає система (див. рис. 4.1).

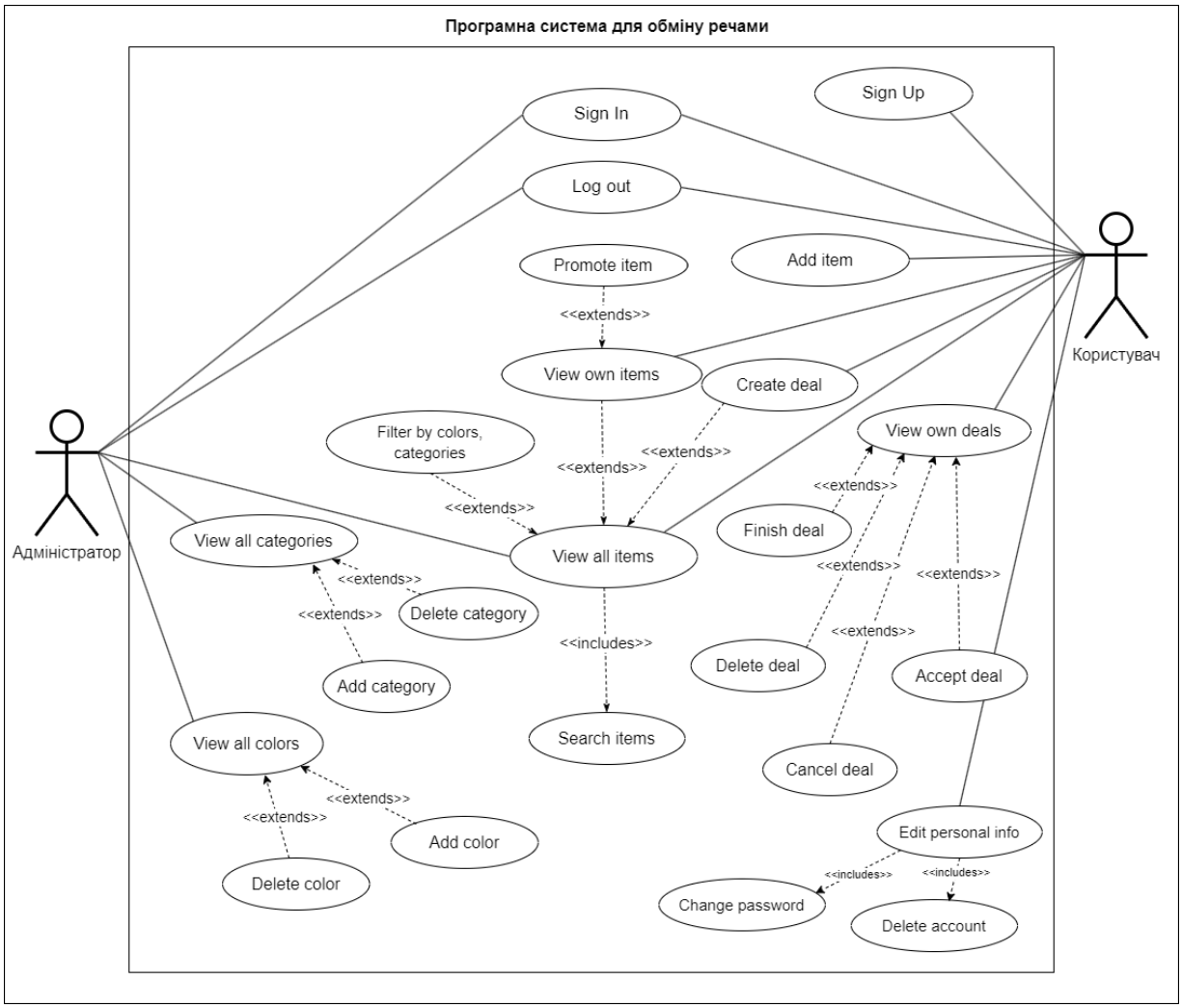

Рисунок 3.1 – Діаграма прецедентів для програмної системи

У рамках програмної системи для обміну речами діаграма включає двох акторів – користувач та адміністратор.

Користувач – основна роль в системі, він може реєструватися в системі, авторизуватися, додавати оголошення про товари, редагувати їх та видаляти, створювати обмінні угоди з іншими користувачами, а також обмінюватися з ними повідомленнями.

Діаграма розгортання [5] (див. рис. 4.2) системи ілюструє фізичну структуру програмної системи для обміну речами. Система встановлюється на сервері, що використовує технології ASP.NET Core 8 для серверної частини.

Основною метою діаграми розгортання є візуалізація та розуміння того, як фізичні компоненти програмної системи взаємодіють між собою та з іншими системами в середовищі мережі.

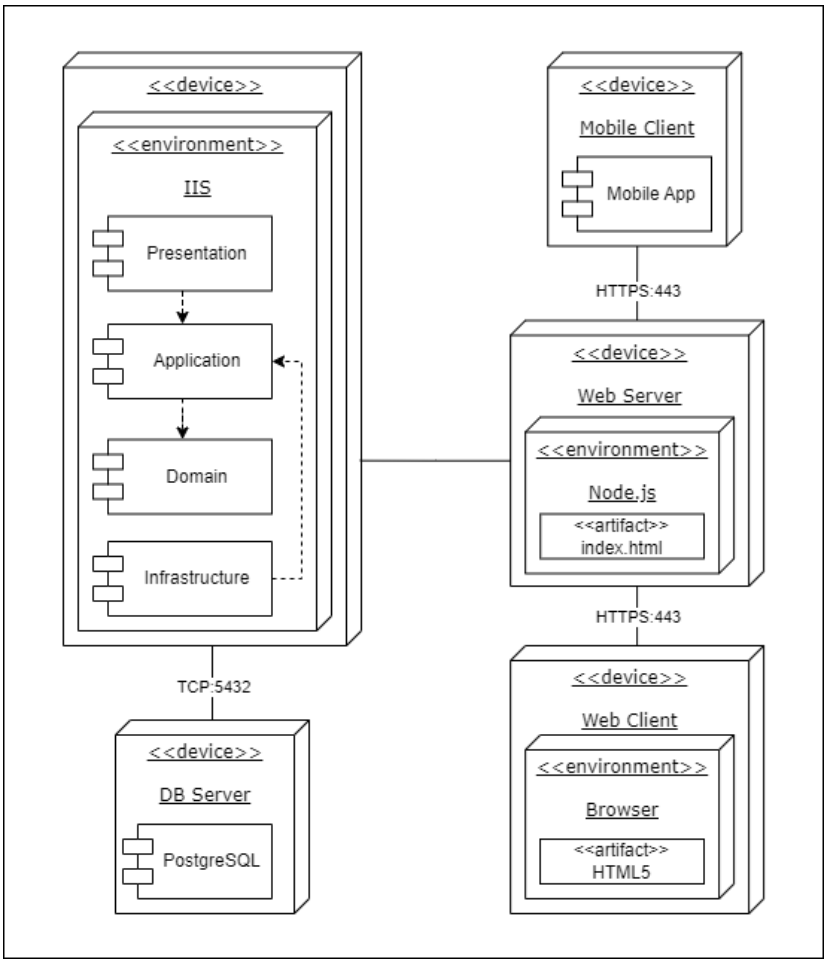

Рисунок 3.2 – Діаграма розгортання для програмної системи

Компонентами діаграми є: серверна частина, яка поділяється на рівні презентації, програми, домену та інфраструктури, база даних, клієнтська частина, яка складається з мобільного застосунку та веб-клієнту, та веб-серверу.

Система буде побудована на основі багаторівневої архітектури, де кожен рівень виконує свою задачу, забезпечуючи гнучкість та масштабованість. Крім того, взаємодія між компонентами здійснюється через стандартні протоколи (HTTPS, TCP), які забезпечують безпеку та сумісність.

На рисунку 4.3 зображена діаграма класів [6], яка демонструє структуру програми на рівні класів та їх взаємодію.

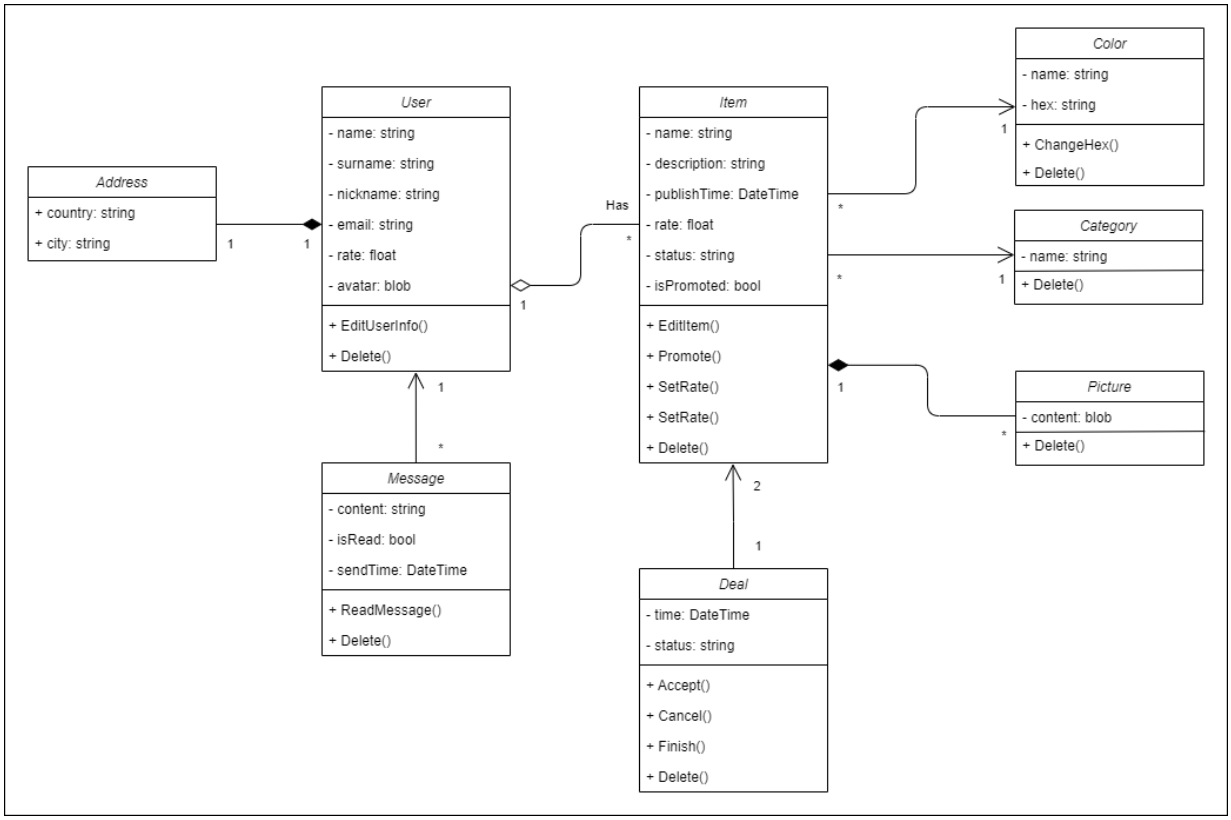

Рисунок 3.3 – Діаграма класів

Складається система з таких класів, як User, Item, Deal, Message, Color, Category, Picture, Address, де:

– User представляє користувачів системи;
– Item представляє товари, які користувачі можуть обмінювати;
– Deal представляє процес обміну товарами між користувачами;
– Message забезпечує обмін повідомленнями;
– Color, Category, Picture представляють додаткові характеристики для товарів;
– Address надає можливість вказати користувачами адресу проживання.

Головна мета діаграми класів – допомогти розуміти структуру програми та її компонентів. Вона дозволяє отримати уявлення про класи, які складають систему, їх властивості та зв’язки.

3.2 Проєктування архітектури системи

Так як у якості архітектури системи було обрано чисту архітектуру, система буде розділена на декілька основних рівнів: доменний (Domain), програмний (Application), інфраструктурний (Infrastructure) та презентаційний (Presentation). На рисунку 4.3 зображена схема архітектури системи.

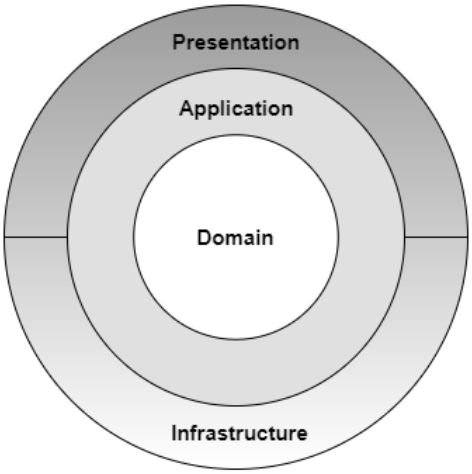

Рисунок 3.4 – Схема архітектури системи (рисунок виконаний самостійно)

Доменний рівень визначає сутності та бізнес-правила, які складають ядро програми. Програмний рівень координує взаємодію між доменним рівнем та зовнішніми компонентами та перетворює дані між різними рівнями. Рівень інфраструктури взаємодіє з зовнішніми системами та технологіями, такими як бази даних, API або хмарними службами. Презентаційний рівень є зовнішнім рівнем програми, який безпосередньо взаємодіє з кінцевим користувачем.

Додавання нових функцій, виправлення помилок та внесення змін в існуючу функціональність не буде впливати на решту програми завдяки розділенню системи на окремі шари. Також поділ на рівні допоможе при тестування системи, так як це гарантує, що при написанні автоматизованих тестів, зміни на одному рівні не порушать функціональність решти додатку.

Принципи чистої архітектури будуються за основі принципів SOLID [7]:

– Принцип єдиної відповідальності (SRP): клас повинен мати лише одну причину для зміни;

– Принцип відкритості/закритості (OCP): клас має бути відкритим для розширення, але закритим для змін;
– Принцип заміщення Лісков (LSP): підтипи повинні бути замінюваними для своїх базових типів;
– Принцип розділення інтерфейсу (ISP): клієнти не мають залежати від методів, які вони не використовують;
– Принцип інверсії залежності (DIP): високорівневі модулі не повинні залежати від низькорівневих, обидва повинні залежати від абстракцій.

Отже, чиста архітектура дозволяє створювати надійні, гнучкі та легко підтримувані системи. Завдяки чіткій структурі та інверсії залежностей, система залишається адаптивною до змін та забезпечує високу якість коду.

3.3 Проєктування структури збереження даних системи

Під час проєктування системи було розроблено ER-діаграму [8], або діаграму зв’язків між сутностями, для бази даних, яка використовується для моделювання відношень між різними сутностями та їх атрибутами (рисунок 4.4).

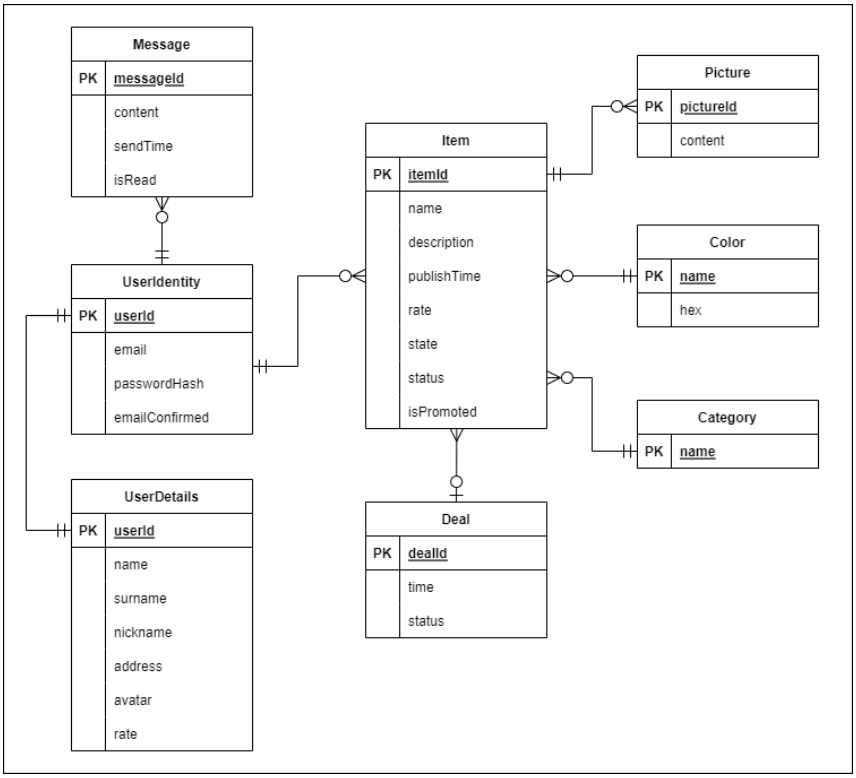

Рисунок 3.5 – Діаграма зв’язків між сутностями

База даних даної програмної системи буде зберігатися в окремому контейнері. Уся інформація, яка буде поступати у систему, така як дані користувачів, інформація про товари, обміни, буде зберігатися у базі даних, що гарантує стабільність та надійність даних.

Складатися база даних буде з 8 сутностей:

- Item – містить інформацію про товар (унікальний ключ, назва, опис, час публікації, оцінка, стан, статус, значення рекламується чи ні);
- Color – містить інформацію про колір товара (унікальна назва, код кольору);
- Category – містить інформацію про категорію товара (унікальна назва);
- Deal – відповідає за обмін користувача (час створення, статус);
- Picture – відповідає за фото товара (унікальний ключ, контент);
- Message – відповідає за повідомлення користувача (унікальний ключ, час відправки, значення прочитане чи ні);
- UserDetails – зберігає основну інформацію користувача, окрім адреси пошти та пароля (унікальний ключ, ім’я, прізвище, прізвисько, адреса, фото профілю, оцінка користувача, яка формується на основі оцінок товарів користувача);
- UserIdentity – зберігає адресу пошти та пароль користувача.

Між сутностями наявні відношення:

- Сутність UserIdentity пов’язана з сутностями Item та Message зв’язком один до багатьох;
- Сутність Item зв’язана з сутностями Deal, Category, Color зв’язком багато до одного;
- Сутність UserDetails пов’язана з сутністю UserIdentity зв’язком один до одного;
- Сутність Picture пов’язана з сутністю Item зв’язком багато до одного.

Планується реалізування бази даних за допомогою реляційної моделі, СУБД PostgreSQL [9] та технології Docker, яка допоможе запускати базу даних в окремому контейнері. Також для ефективної та зручної роботи з базою даних

планується використання ORM Entity Framework Core 8.

3.4 Проєктування UI / UX або іншого дизайну системи

Бекенд програмної системи для обміну речами складається з 41 ендпоїнтів і реалізований у вигляді RESTful API. Дана архітектура використовує HTTP-протокол для створення, читання, оновлення та видалення ресурсів, що робить взаємодію між клієнтом та сервером зручною та інтуїтивно зрозумілою.

REST надає доступ до ресурсів за допомогою HTTP-методів та унікальних ідентифікаторів.

Доступ до ресурсів здійснюється через стандартні HTTP-методи, а саме:

- POST – створює нові ресурси;
- GET – отримує дані про ресурси;
- PUT – оновлює дані про ресурси;
- DELETE – видаляє наявні ресурси.

Також для повідомлення про результати обробки запитів використовуються HTTP-статус-коди, наприклад, код 200 означає, що запит був успішний, код 201 – запит був успішно виконаний і було створено новий ресурс, код 404 означає, що ресурс не було знайдено, а код 500 говорить про внутрішню помилку на сервері.

Для забезпечення якісної документації та полегшення використання API, було використано інструмент для опису та документування RESTful API – Swagger.

Нижче наведено REST-специфікацію на прикладі ендпоінтів для товару (див. табл. 3.1).

Таблиця 3.1 – Маршрут /api/Items

Дія	Метод	Тіло	Відповіді
Add	POST	{ “name”: “string”, “description”: “string”, “color”: “string”, “category”: “string”,	201 – успішно створено, 400 – помилка валідації

		“state”: 0, “wantedCategory”: [“string”], “pictureIds”: [“<uuid>”, “<uuid>”] }	

Edit	PUT	{
		“name”: “<string>”,
		“description”: “<string>”,
		“color”: “<string>”,
		“state”: 0,
		“wantedCategory”: [
		“<string>”,
		“<string>”
		],
		“pictureIds”: [
		“<uuid>”,
		“<uuid>”
		]
		}
			200 – успішно змінено, 400 – помилка валідації, 404 – товар не знайдено

Delete	DELETE	{}	200 – успішно видалено, 404 – товар не знайдено
Get	GET	{}	200 – успішно знайдено, 404 – товар не знайдено

Крім того, у системі існують рівні доступу, які визначають, у кого є доступ до даних та функцій. Таких рівнів двоє, а саме:

- «user», який надається користувачам, які авторизовані у системі, у них є доступ до усіх основних функцій системи;
- «admin», який надається певному користувачу, який вже створений у системі, йому доступні функції адміністрування у системі;

Це розділення допомагає захистити систему від несанкціонованого доступу, забезпечує належне управління доступом, мінімізує ризики витоку конфіденційної інформації та і, загалом, забезпечує її стабільну та безпечну роботу.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис обраної платформи

Для реалізації серверної частини програмної системи для обміну речами було обрано платформу ASP.NET Core 8, що працює на мові програмування C#. Це рішення було обумовлене певними технічними та стратегічними перевагами, які забезпечують надійність, масштабованість, безпеку та зручність для розробки системи. ASP.NET Core 8 підтримує розробку та розгортання на різних операційних системах, таких як Windows, Linux та macOS, що дозволяє забезпечити гнучкість у виборі серверного середовища та зменшити витрати на інфраструктуру. Висока продуктивність даної платформи забезпечує швидкий відгук системи та ефективне використання ресурсів сервера, що є критично важливим для обробки великих обсягів даних, а масштабованість ASP.NET Core 8 дозволяє легко адаптувати систему до зростаючих вимог користувачів. Безпека також є ключовим аспектом цієї платформи, оскільки вона має вбудовані засоби захисту, такі як автентифікація та авторизація, захист від CSRF-атак, та підтримка HTTPS, що забезпечує високий рівень захисту даних користувачів.

Серед стратегічних переваг можна відзначити підтримку та розвиток даної платформи компанією Microsoft, це гарантує регулярні оновлення, виправлення помилок та нові можливості. Крім того, ASP.NET Core легко інтегрується з іншими продуктами та сервісами Microsoft, такими як SQL Server, Azure та Visual Studio, що забезпечує зручний та ефективний процес розробки, тестування та розгортання. Використання відкритої платформи з відкритим вихідним кодом зменшує витрати на ліцензування та дозволяє використовувати безкоштовні інструменти та бібліотеки.

Таким чином, вибір ASP.NET Core 8 для серверної частини був обумовлений його технічними перевагами – продуктивністю, масштабованістю та безпекою, а також стратегічними перевагами, такими як активна підтримка та інтеграція з іншими продуктами. Це забезпечило створення надійної, продуктивної та зручної у підтримці системи, яка відповідає потребам користувачів і бізнесу.

4.2 Опис обраного архітектурного рішення

Для розробки серверної частини програмної системи для обміну речами було використано архітектурний підхід на основі чистої архітектури (Clean Architecture). Використання чистої архітектури дозволяє поділити код на окремі, незалежні модулі, це сприяє зменшенню змішування між компонентами, що робить код більш зрозумілим і легким для підтримування. Завдяки чітко визначеним межам між модулями, можна легко вносити зміни в одну частину системи без ризику порушення роботи в інших частинах. Дотримання принципів чистої архітектури забезпечує високий рівень безпеки системи. Завдяки взаємодії зовнішніх систем і сервісів з бізнес-логікою через чітко визначені інтерфейси, знижується ризик вразливостей, які пов’язані з неправильною інтеграцією або неконтрольованим доступом до критичних частин системи.

Таким чином, вибір чистої архітектури для серверної частини програмної системи для обміну речами забезпечує створення стабільного, гнучкого та легко підтримуваного програмного забезпечення.

У додатку Г зображена структура проєкту на основі чистої архітектури.

Доменний шар (Domain Layer) є основним компонентом чистої архітектури, він представляє бізнес-логіку та сутності системи. Даний шар містить усі бізнес-правила та знання про застосунок і є незалежним від будь-яких конкретних деталей реалізації або технологій. Доменний шар визначає сутності, об’єкти значень та бізнес-правила, які складають ядро застосунку. У контексті програмної системи для обміну речами даний рівень містить усі сутності, а саме: Category, Color, Deal, Item, Message, Picture, UserDetails, методи з бізнес-правилами в середині кожної сутності, а також об’єкт Address. Частина коду сутності Deal:

```
public Deal(Guid id, Item offeredItem, Item requestedItem, DateTime
creationTime) : this()
{
    Id = id;
    OfferedItem = offeredItem;
    RequestedItem = requestedItem;
    CreationTime = creationTime;
}
```

```
public DateTime CreationTime { get; set; }
public Item OfferedItem { get; set; } = null!;
public Item RequestedItem { get; set; } = null!;
public DealStatus Status { get; set; } = DealStatus.Requested;

public void Accept()
{
    if (Status is not DealStatus.Requested)
        throw new InvalidOperationException("You cannot accept a deal
that is not requested!");

    Status = DealStatus.Started;
    OfferedItem.Status = ItemStatus.InDeal;
    RequestedItem.Status = ItemStatus.InDeal;
}
```

Прикладний рівень (Application Layer) діє як міст між доменним рівнем і зовнішніми інтерфейсами програми, такими як рівень представлення (Presentation Layer). Цей рівень координує взаємодію між доменним рівнем і зовнішніми компонентами та перетворює дані між цими шарами. У контексті даної системи основою даного шару є команди з обробниками команд (Command Handler) та запити з обробниками запитів. Також тут знаходяться інтерфейси для репозиторіїв. Команди відповідають за передачу даних, які необхідні для виконання певних дій. Вони є контейнерами для даних і не містять логіки. Обробники для команд ж відповідають за виконання логіки команд, які обробляють команди та перевіряють їхню валідність. Те саме стосується й запитів, тільки їхня відмінність полягає в тому, що запити відповідають за методи отримання. Код команди для додавання товара:

```
public record AddItemCommand(string Name, string Description, string
Color, string Category, State State,
                             string[] WantedCategory,
IEnumerable<Guid> PictureIds) : IRequest<Result<ItemViewModel>>;

public class AddItemCommandHandler(IItemRepository repository, IUser
user) : IRequestHandler<AddItemCommand, Result<ItemViewModel>>
{
    public async Task<Result<ItemViewModel>> Handle(AddItemCommand
request, CancellationToken cancellationToken)
    {
        var id = Guid.NewGuid();
        var item = new Item(id, request.Name, request.Description,
request.Color, request.Category, request.State,
```

```
                                request.WantedCategory, DateTime.UtcNow,
request.PictureIds, user.Id!.Value);
        await repository.AddAsync(item, cancellationToken);
        return Result.Ok();
    }
}

public class AddItemCommandValidator :
AbstractValidator<AddItemCommand>
{
    public AddItemCommandValidator(ICategoryRepository
categoryRepository, IColorRepository colorRepository,
IPictureRepository pictureRepository)
    {
        RuleFor(i =>
i.Name).NotEmpty().MinimumLength(16).MaximumLength(70);
        RuleFor(i =>
i.Description).NotEmpty().MinimumLength(40).MaximumLength(500);
        RuleFor(i =>
i.Color).NotEmpty().MustAsync(colorRepository.ContainsAsync);
        RuleFor(i =>
i.Category).NotEmpty().MustAsync(categoryRepository.ContainsAsync);
        RuleForEach(i =>
i.WantedCategory).MustAsync(categoryRepository.ContainsAsync);
        RuleForEach(i =>
i.PictureIds).MustAsync(pictureRepository.ContainsAsync);
    }
     }
```

Інфраструктурний шар (Infrastructure Layer) відповідає за технічні деталі програми, такі як доступ до даних, електронна пошта. Інфраструктурний рівень взаємодіє із зовнішніми системами і технологіями, такими як бази даних, API або хмарні сервіси. Цей рівень використовує абстракції та інтерфейси для взаємодії з прикладним рівнем, що дозволяє змінювати реалізацію певного компонента, не впливаючи на решту програми. Цей шар містить міграції для управління змінами у структурі базі даних, репозиторії, які відповідають за операції створення, читання, оновлення та видалення (CRUD) сутностей, та репозиторії для читання, які орієнтовані тільки на операції читання, а також Application Context, який координує взаємодію з базою даних. Код репозиторія:

```
public class Repository<TAggregate, TEntity, TKey>(ApplicationContext
context, IMapper mapper) : IRepository<TAggregate, TKey>
    where TAggregate : Aggregate<TKey>
    where TEntity : class, IEntity<TKey>
    where TKey : IEquatable<TKey>
{
    protected DbSet<TEntity> Entities => context.Set<TEntity>();
```

```
    public virtual Task AddAsync(TAggregate aggregate,
CancellationToken cancellationToken = default)
    {
        var entity =
mapper.From(aggregate).EntityFromContext(context).AdaptToType<TEntity
>();
        Entities.Add(entity);
        return context.SaveChangesAsync(cancellationToken);
    }
    public virtual Task DeleteAsync(TAggregate aggregate,
CancellationToken cancellationToken = default)
    {
        var entity =
mapper.From(aggregate).EntityFromContext(context).AdaptToType<TEntity
>();
        Entities.Remove(entity);
        return context.SaveChangesAsync(cancellationToken);
    }
    public virtual async Task<TAggregate?> FindAsync(TKey id,
CancellationToken cancellationToken = default)
    {
        var entity = await
Entities.AsNoTracking().FirstOrDefaultAsync(x => Equals(x.Id, id),
cancellationToken);
        return entity.Adapt<TAggregate>();
    }
    public virtual Task<bool> ContainsAsync(TKey key,
CancellationToken cancellationToken = default)
    {
        return Entities.AnyAsync(x => x.Id.Equals(key),
cancellationToken);
    }
    public virtual Task UpdateAsync(TAggregate aggregate,
CancellationToken cancellationToken = default)
    {
        var entity =
mapper.From(aggregate).EntityFromContext(context).AdaptToType<TEntity
>();
        Entities.Update(entity);
        return context.SaveChangesAsync(cancellationToken);
    }
}
```

Рівень представлення (Presentation Layer) відповідає за надсилання відповідей та отримання запитів користувачів. Це зовнішній рівень додатку, який безпосередньо взаємодіє з кінцевим користувачем. Даний шар не містить бізнес-логіки або знань про предметну область. Головними елементами цього шару є запити, які відповідають за отримання даних без внесення змін до стану системи, і ендпоінти. Частина коду ендпоінтів для кольору:

```
public static class ColorEndpoints
{
    public static RouteGroupBuilder MapColors(this
IEndpointRouteBuilder app)
    {
        var groupName = "Colors";
        var group = app.MapGroup($"/api/{groupName}")
                       .WithTags(groupName)
                       .WithOpenApi();
        group.MapGet(string.Empty, GetColors);
        group.MapPost(string.Empty, AddColor);
        group.MapPatch("{name}", EditColor);
        group.MapDelete("{name}", DeleteColor);
        return group;
    }

    private static async Task<PaginationList<ColorViewModel>>
GetColors([AsParameters] GetColorsRequest request, ISender mediator)
    {
        var query = new GetColorsQuery
        {
            Page = request.Page,
            PerPage = request.PerPage,
            Search = request.Search,
            Sort = nameof(Color.Name),
            SortBy = Sorting.Descending
        };
        return await mediator.Send(query);
    }

    private static async Task<IResult> AddColor(AddColorRequest
request, ISender mediator)
    {
        var command = new AddColorCommand(request.Name, request.Hex);
        var result = await mediator.Send(command);
        return result.ResultToResponse(StatusCodes.Status201Created);
}
```

В ендпоінтах використовується новий підхід по маршрутизації – RouteGroupBuilder, який надає можливість групувати ендпоінти, спрощуючи їхнє налаштування та управління.

4.3 Опис обраної моделі зберігання даних

PostgreSQL було обрано як основну систему для збереження даних для програмної системи для обміну речами з урахуванням численних переваг, які ця технологія надає. PostgreSQL – потужна та відкрита реляційна СКБД, вона відзначається своєю стабільністю та широким набором функцій, які сприяють якісному зберіганню та ефективній обробці даних. Однією з ключових переваг

PostgreSQL є її здатність підтримувати складні запити та транзакції. Це є ключовим аспектом для даного проекту, оскільки забезпечення цілісності даних та виконання бізнес-логіки на рівні бази даних є важливою складовою стабільної роботи системи обміну речами. Крім того, ця база даних легко масштабується та підтримує велике навантаження завдяки своїй архітектурі та можливості горизонтального масштабування. Це дає змогу не переживати за зростання обсягу даних та запитів у майбутньому, зберігаючи при цьому стабільну роботу системи. Щодо безпеки, PostgreSQL також відзначається високим рівнем функціональності. Вбудовані механізми безпеки, такі як аутентифікація, шифрування та керування ролями та правами доступу, гарантують захист даних від несанкціонованого доступу. Немало важливою перевагою PostgreSQL є її доступність та активна спільнота розробників, що забезпечує постійне вдосконалення та оновлення технології.

Як було вказано раніше, у системи існує клас з назвою ApplicationContext, який був створений за допомогою бібліотеки EntityFrameworkCore. Даний клас є головним інструментом для взаємодії з базою даних. ApplicationContext з’єднує додаток з базою даних, а також керує моделями даних та відповідає за виконання запитів. Цей клас допомагає зручно та ефективно працювати з базою даних через код, що полегшує розробку і підтримку системи.

Фрагмент коду класу ApplicationContext:

```
public class ApplicationContext(DbContextOptions<ApplicationContext>
options)
    : IdentityDbContext<UserIdentity, IdentityRole<Guid>,
Guid>(options)
{
    public DbSet<Category> Categories { get; set; } = null!;
    public DbSet<Color> Colors { get; set; } = null!;
    public DbSet<Deal> Deals { get; set; } = null!;
    public DbSet<Item> Items { get; set; } = null!;
    public DbSet<PictureInItem> PicturesInItems { get; set; } =
null!;
    public DbSet<Picture> Pictures { get; set; } = null!;
    public DbSet<UserDetails> UserDetails { get; set; } = null!;
    public DbSet<WantedCategory> WantedCategories { get; set; } =
null!;

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) => optionsBuilder.UseLazyLoadingProxies();
```

```
        protected override void OnModelCreating(ModelBuilder builder)
        {

    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly
    ());

            builder.Entity<Category>()
                   .HasMany(c => c.Items)
                   .WithOne(i => i.Category)
                   .HasForeignKey(i => i.CategoryName)
                   .OnDelete(DeleteBehavior.Restrict);

            builder.Entity<Category>()
                   .Property(c => c.Name)
                   .ValueGeneratedNever();
```

У цьому класі визначені властивості DbSet для кожної сутності, яка представлена в базі даних. Кожна властивість має значення «null!», щоб уникнути потенційної помилки компіляції.

Метод OnModelCreating визначає конфігурацію моделі даних. Він встановлює зв’язки між таблицями, налаштовує первинні та зовнішні ключі, а також задає варіант видалення даних при видаленні батьківського запису.

4.4 Опис обраної технології розгортання

Для більш якісного та зручнішого управління та користування програмною системою для обміну речами було використано технологію розгортання Docker [10]. Ця технологія представляє стабільне середовище, що знижує ймовірність помилок, які викликані відмінностями у налаштуваннях середовищ, та забезпечує передбачувану поведінку проєкту на всіх етапах його життєвого циклу. Контейнери Docker ізолюють додатки та їхні залежності, запобігаючи конфліктам між різними версіями бібліотек і забезпечують безпеку та стабільність роботи системи. Дана технологія дозволяє легко масштабувати серверні компоненти як вертикально, так і горизонтально, завдяки чому можна ефективно управляти навантаженням і забезпечувати високу продуктивність.

У даній програмній системі також база даних PostgreSQL розміщується у контейнері Docker, це забезпечує стабільність даних, створюючи ізольоване середовище, що дозволяє уникнути впливу на інші компоненти системи під час

тестування та розгортання нових версій бази даних. Крім того, це полегшує створення резервних копій та відновлення бази даних у разі потреби, що значно спрощує управління даними.

Контейнери Docker надають можливість легко масштабувати базу даних, забезпечуючи їхню високу продуктивність навіть під великим навантаженням. Дана технологія також гарантує високий рівень безпеки завдяки ізоляції мереж, контролю доступу та управлінню налаштуваннями безпеки, які допомагають захистити базу даних від несанкціонованого доступу.

Отже, використання Docker для розгортання серверу та бази даних дозволило створити гнучку та безпечну інфраструктуру. Це було важливе рішення для створення надійного середовища для роботи даної програмної системи обміну речами.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є важливим етапом у процесі розробки, що дозволяє оцінити його якість, функціональність та відповідність вимогам. Воно складається з різноманітних підходів та методів, які забезпечують всебічну перевірку функціональності, продуктивності, безпеки та зручності використання системи.

Функціональне тестування зосереджене на перевірці функцій програмного забезпечення відповідно до визначених вимог. Воно поділяється на:

- модульне тестування, де окремі компоненти тестуються ізольовано;
- інтеграційне тестування, яке перевіряє взаємодію між компонентами;
- системне тестування, що охоплює повну перевірку системи;
- тестування прийняття, яке забезпечує відповідність вимогам користувачів чи замовників.

Нефункціональне тестування спрямоване на оцінку не функціональних аспектів програмного забезпечення. Це включає:

- тестування продуктивності для оцінки швидкості та ефективності роботи;
- тестування навантаження для перевірки системи під високим навантаженням;
- тестування на міцність для оцінки роботи під граничними умовами;
- тестування безпеки для виявлення вразливостей;
- тестування надійності для перевірки стійкості системи до відмов;
- тестування зручності користування для оцінки інтуїтивності та легкості використання.

Тестування на етапах розробки включає методи білого ящика, чорного ящика та сірого ящика. Тестування білого ящика здійснюється з урахуванням внутрішньої структури системи, чорного ящика – без знання внутрішньої реалізації, орієнтуючись на вхідні та вихідні дані, а сірого ящика поєднує елементи обох підходів.

Спеціалізоване тестування включає:

- регресійне тестування для перевірки, що нові зміни не вплинули на існуючу функціональність;
- повторне тестування для підтвердження виправлення помилок;
- тестування сумісності для перевірки роботи на різних платформах;
- тестування конфігурації для перевірки з різними налаштуваннями;

Описані види тестування використовуються на різних етапах життєвого циклу розробки програмного забезпечення і допомагають забезпечити його високу якість, відповідність вимогам користувачів і замовників та надійність у роботі.

Під час розробки серверної частини програмної системи для обміну речами використовувалось гібридне тестування методами білої та чорної скриньок, це допомогло отримати повне уявлення про якість програмного забезпечення та забезпечити його надійність та ефективність.

Серверна частина в основному була протестована функціональним мануальним способом, але також були створені автоматизовані тести для перевірки функціональності системи за допомогою Postman. Тести для перевірки введених значень у Postman забезпечують перевірку коректності даних, що відправляються на сервер. Це включає перевірку на обов’язкові поля, правильність формату даних, межі числових значень тощо. Такі тести допомагають гарантувати, що сервер отримує правильні дані, і допомагають виявити потенційні проблеми.

На рисунку 5.1 зображений приклад тестів, які написані для перевірки валідності даних та успішності запиту при створенні оголошення про товар.

```
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Check if 'name' field is present", function () {
    var jsonData = pm.request.body.raw;
    var requestBody = JSON.parse(jsonData);
    pm.expect(requestBody.name).not.empty;
});

pm.test("Check if 'name' length is at least 10 characters and maximum 40 characters", function () {
    var jsonData = pm.request.body.raw;
    var requestBody = JSON.parse(jsonData);
    pm.expect(requestBody.name.length).to.be.at.least(10).and.to.be.below(40);
});

pm.test("Check if 'description' field is present", function () {
    var jsonData = pm.request.body.raw;
    var requestBody = JSON.parse(jsonData);
    pm.expect(requestBody.description).not.empty;
});

pm.test("Check if 'name' length is at least 40 characters and maximum 500 characters", function () {
    var jsonData = pm.request.body.raw;
    var requestBody = JSON.parse(jsonData);
```

Рисунок 5.1 – Написання автоматизованих тестів

Дані тести написані за допомогою JavaScript, вони перевіряють статус відповіді сервера, обов’язкові поля та їх довжину.

Дані введені в тіло запиту:

```
{
  "name": "Blue ball with few scratches",
  "description": "Big blue ball with few scratches, made of plastic",
  "color": "blue",
  "category": "toys",
  "state": 0,
  "wantedCategory": [
    "toys"
  ],
  "pictureIds": [
    "2168abf4-19e1-414a-8fad-6d734ff662e9"
  ]
}
```

На рисунку 5.2 зображені результати тестів.

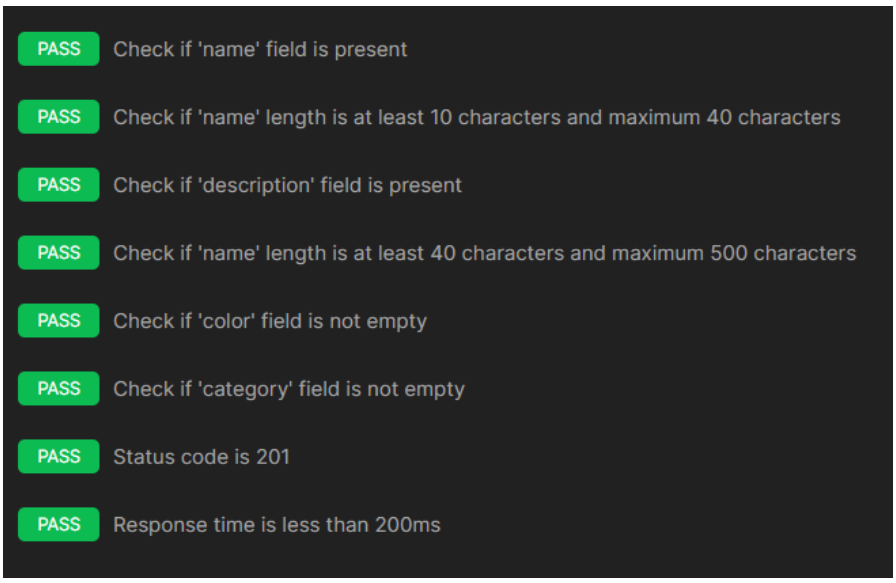

Рисунок 5.2 – Результати виконання тестів

Тестування є невід’ємної частиною розробки програмного забезпечення, без нього неможливо повноцінно визначити чи працює система правильно, ефективно та стабільно.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботі було розроблено серверну частину програмної системи для обміну речами, використовуючи архітектурні принципи та сучасні технології, що забезпечують надійність, масштабованість та ефективність. На основі аналізу предметної галузі та існуючих аналогів було визначено цілі та ризики проекту, це дозволило чітко сформулювати вимоги до системи та завдання, які вона повинна виконувати.

Проектування архітектури системи проводилося з використанням принципів чистої архітектури, що дозволило створити модульну та гнучку структуру, яка легко піддається розширенню та підтримці. Для зберігання даних було обрано реляційну базу даних PostgreSQL, яка забезпечує високу продуктивність та надійність збереження інформації. Розгортання серверної частини було виконано за допомогою Docker, що дозволило створити ізольоване середовище для запуску додатку та забезпечило його незалежність від інфраструктури.

Серверна частина розроблялася на мові програмування C# у середовищі розробки Visual Studio. Використання Swagger для документування API забезпечило легкість інтеграції з іншими системами та зручність для розробників клієнтської частини. Тестування розробленої системи підтвердило її відповідність вимогам та достатній рівень функціональності.

Розроблена система є ефективним рішенням для організації обміну речами між користувачами, забезпечуючи високу якість обслуговування та безпеку даних. Робота може бути продовжена з метою подальшого вдосконалення функціональності та розширення можливостей системи. Додатково, планується інтеграція з іншими платформами для розширення аудиторії користувачів. Важливою задачею на майбутнє є покращення алгоритмів пошуку та рекомендацій для користувачів, що підвищить зручність та ефективність використання системи. Подальші дослідження та впровадження нових технологій дозволять зробити систему ще більш надійною та масштабованою.

Результати було представлено на XXVIII Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (додаток Д).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Куценко А., Колесник О. Розробка компоненту системи веб-застосунку «каталог одягу». Комп'ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві: Всеукр. наук.-практ. конф. здобувачів вищ. освіти і молодих вчених, 22 листопада 2023 р. Харків. нац. автомоб.-дор. ун-т., Харків, 2023. С. 198–201.

2. ASP.NET | open-source web framework for .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 25.04.2024).

3. Hengkyawan J. Clean Architecture in ASP .NET Core Web API. Medium. URL: https://juldhais.net/clean-architecture-in-asp-net-core-web-api-4e5ef0b96f99 (дата звернення: 22.04.2024).

4. Use Case Diagram. Visual Paradigm - Online Productivity Suite. URL: https://online.visual-paradigm.com/diagrams/tutorials/use-case-diagram-tutorial/ (дата звернення: 05.05.2024).

5. Deployment diagram. What is Deployment Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ (дата звернення 05.05.2024)

6. What is Class Diagram? Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/ (дата звернення: 05.05.2024).

7. Diogo Ribeiro da Silva. Clean Architecture: Principles and Practices for Sustainable Code. LinkedIn. URL: https://www.linkedin.com/pulse/clean-architecture-principles-practices-sustainable-ribeiro-da-silva-retvf (дата звернення: 22.04.2024).

8. What is an Entity Relationship Diagram (ERD)? Lucidchart. URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 05.05.2024).

9. PostgreSQL: About. PostgreSQL: The world's most advanced open-source database. URL: https://www.postgresql.org/about/ (дата звернення: 07.05.2024).

10. Manuals. Docker Documentation. URL: https://docs.docker.com/manuals/ (дата звернення: 08.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

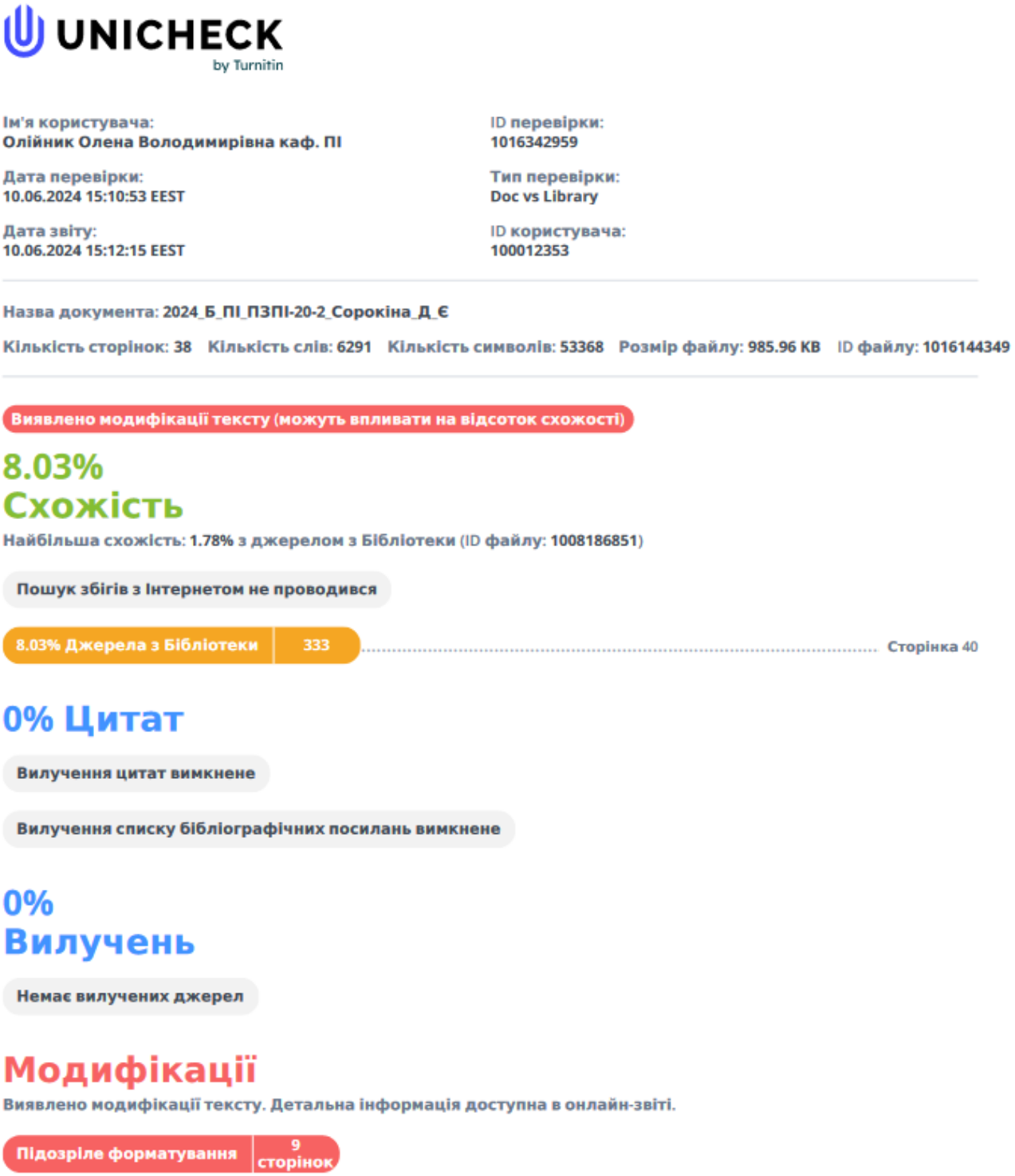

Рисунок А.1 – Звіт Unicheck

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Перший слайд

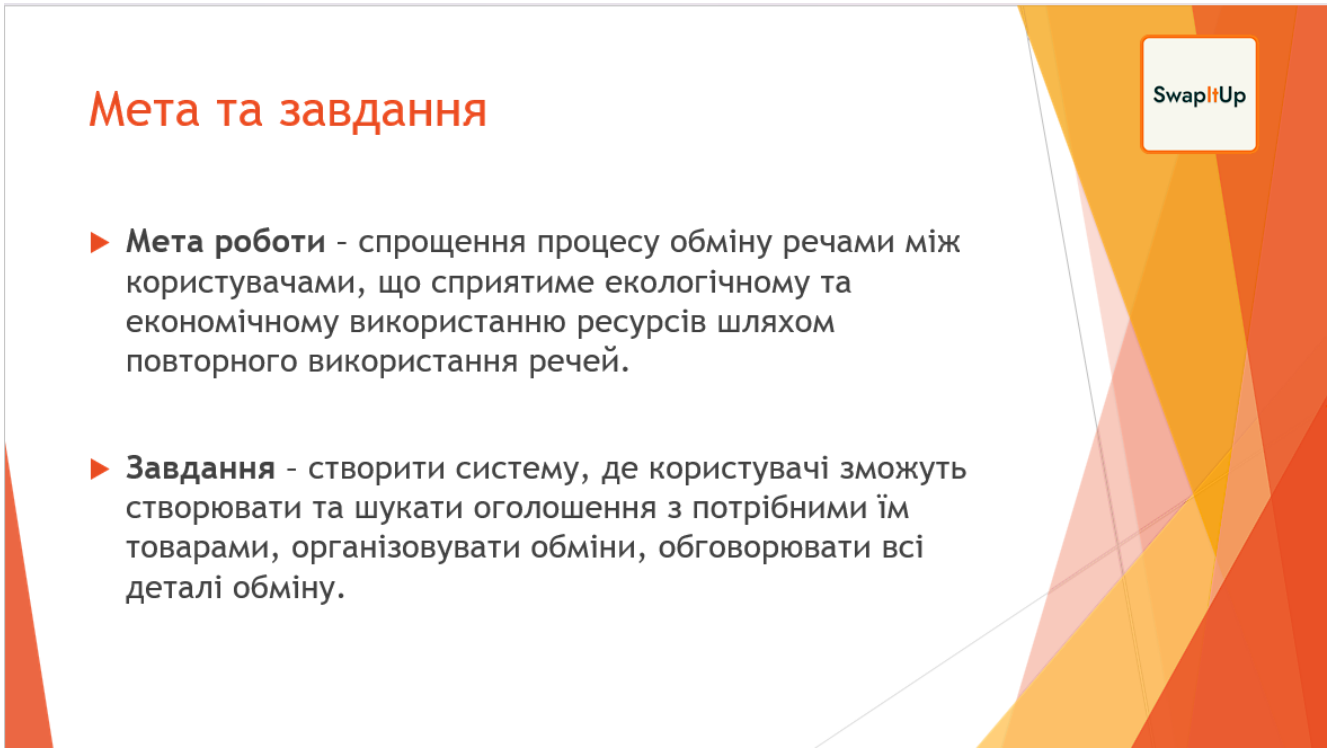

Рисунок Б.2 – Другий слайд

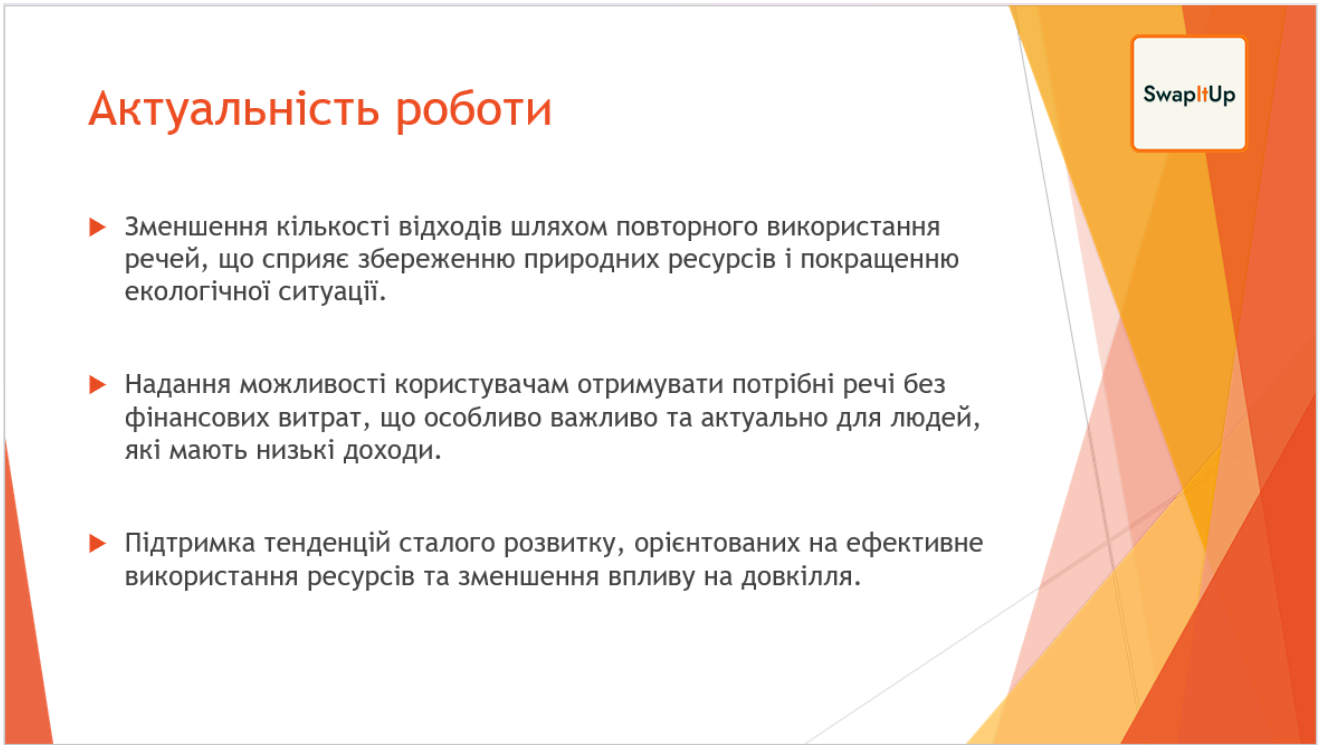

Рисунок Б.3 – Третій слайд

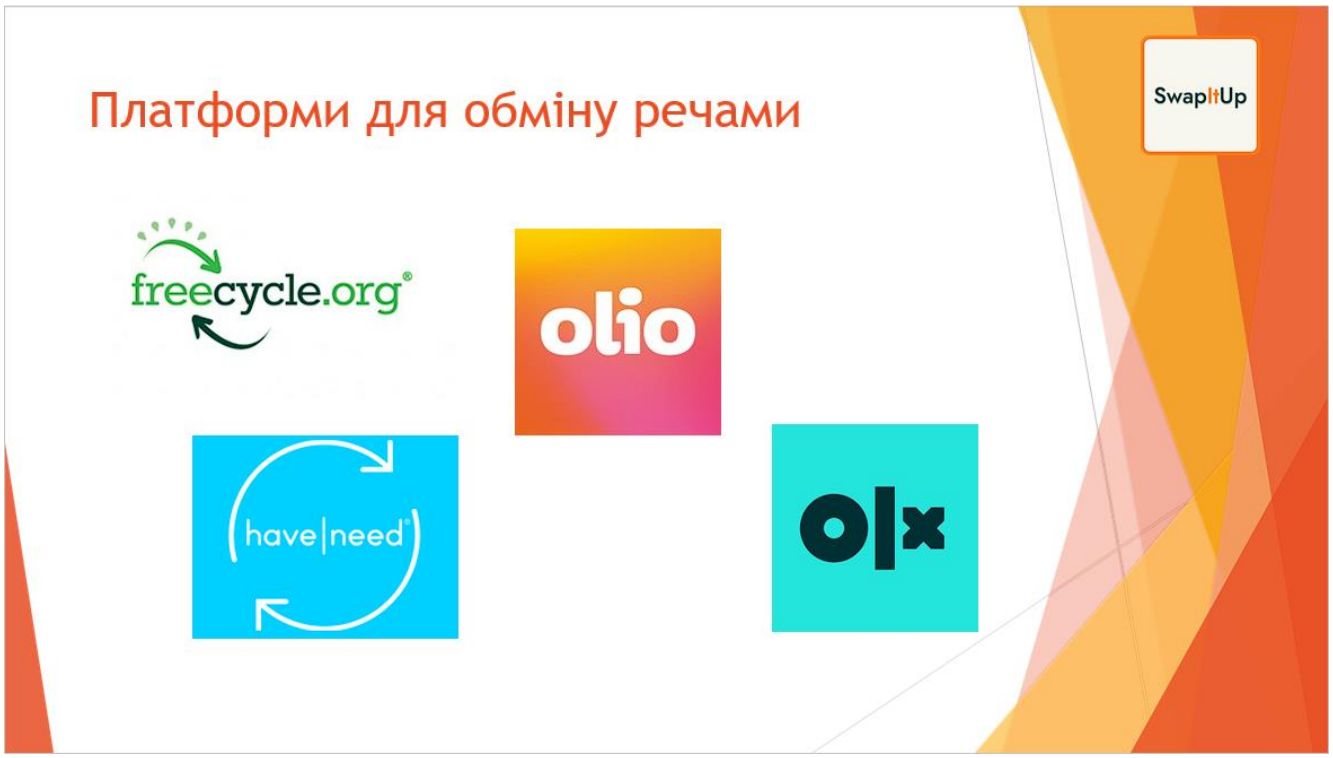

Рисунок Б.4 – Четвертий слайд

Рисунок Б.5 – П’ятий слайд

Рисунок Б.6 – Шостий слайд

Рисунок Б.7 – Сьомий слайд

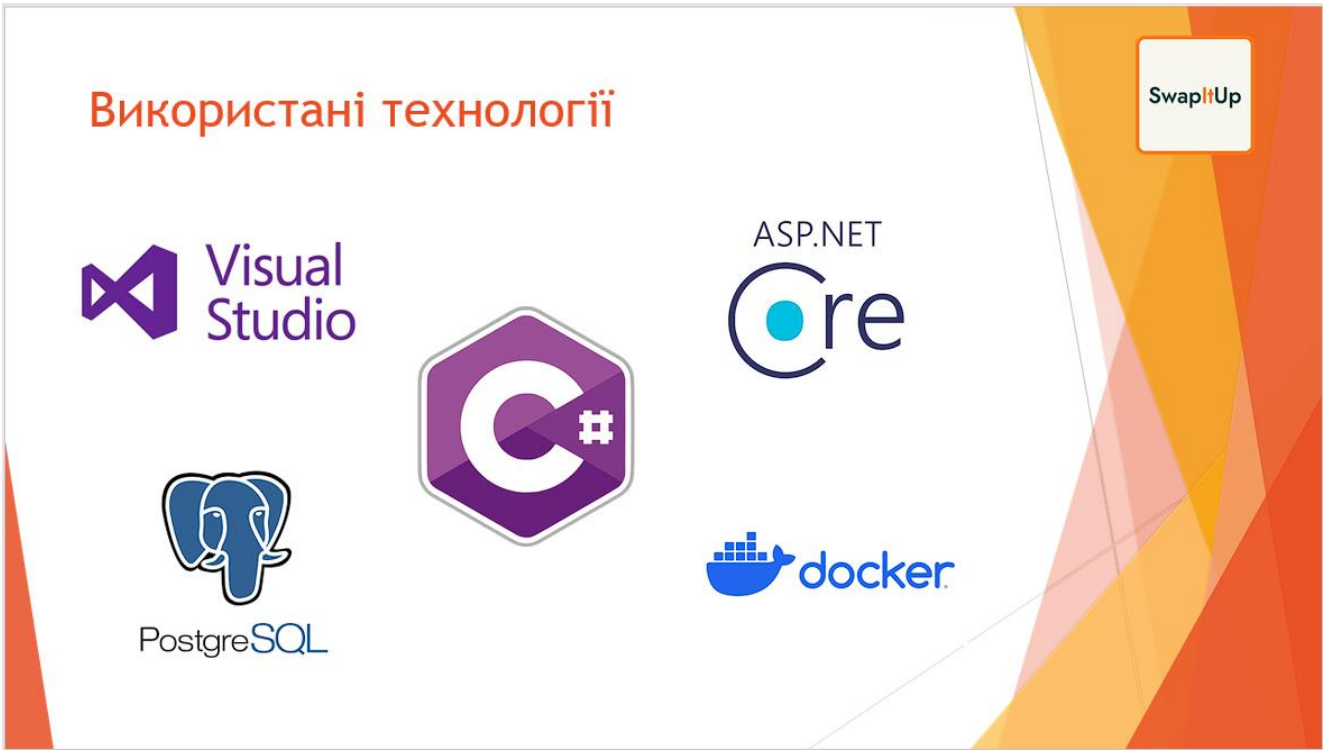

Рисунок Б.8 – Восьмий слайд

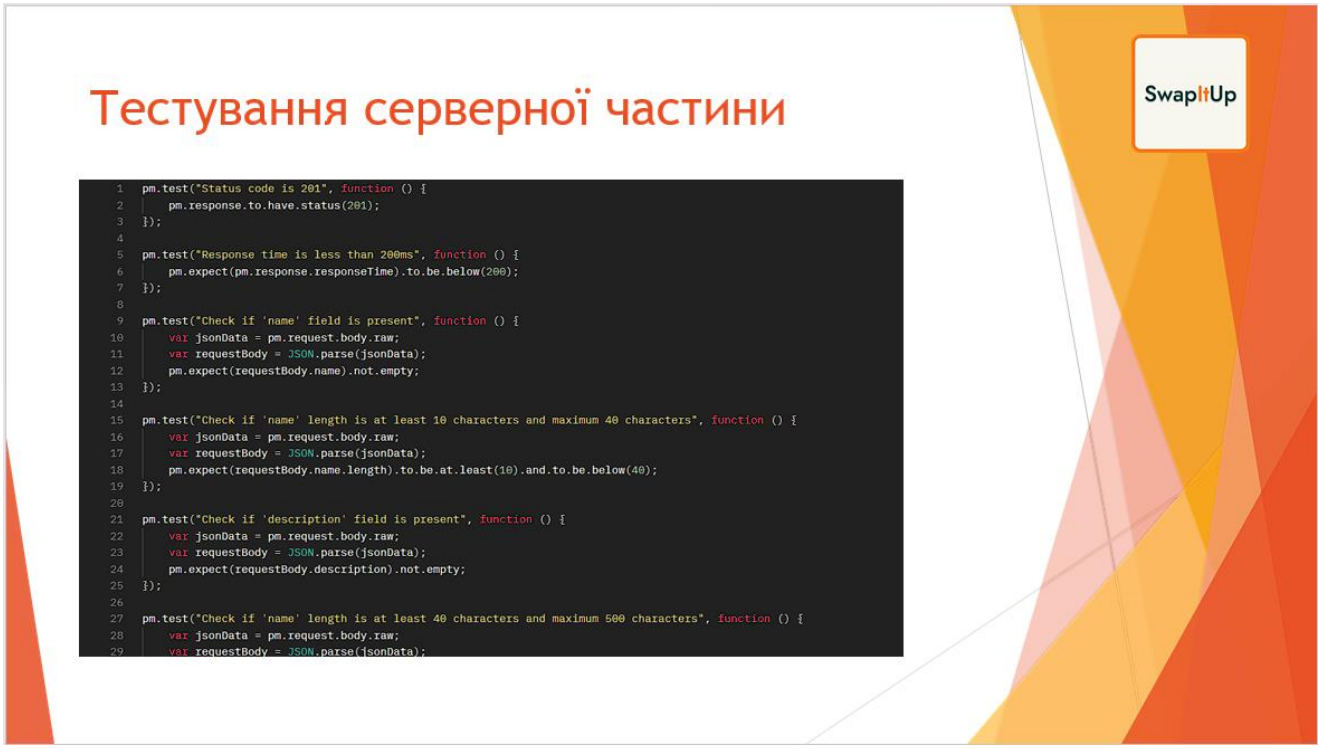

Рисунок Б.9 – Дев’ятий слайд

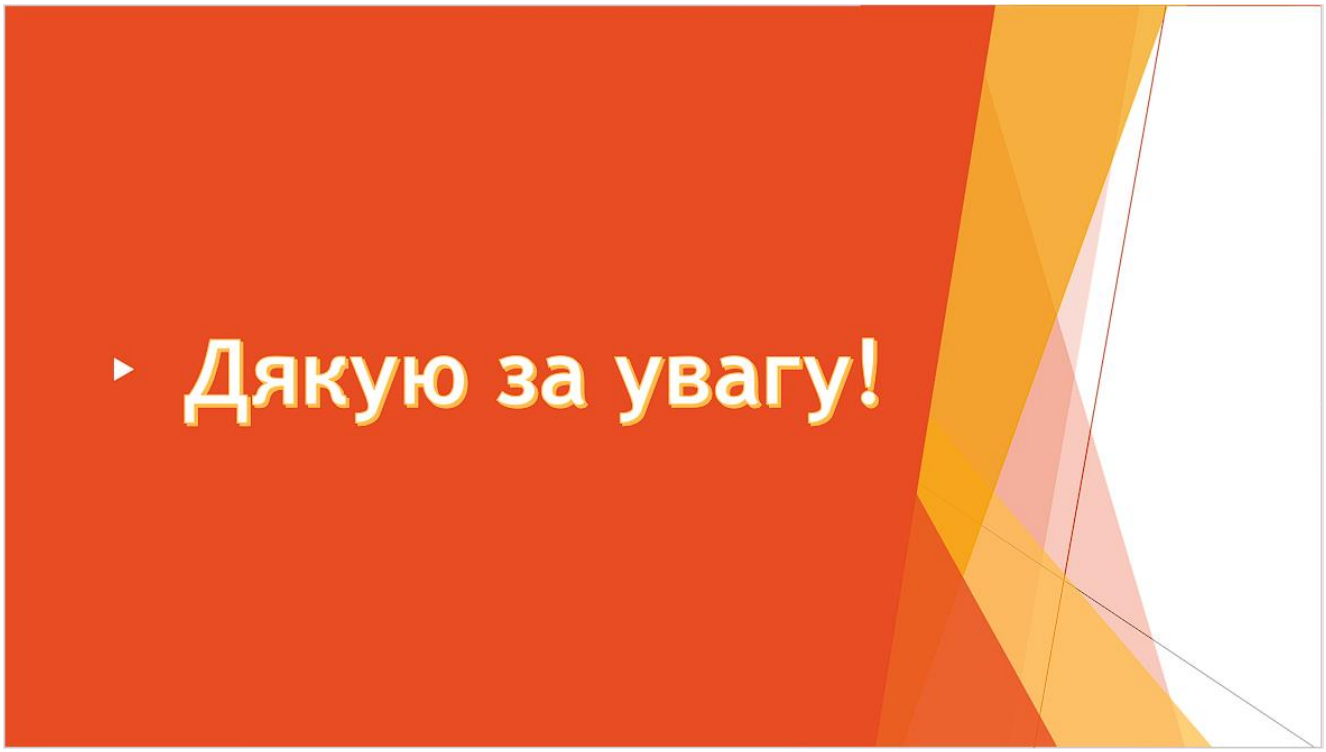

Рисунок Б.10 – Десятий слайд

ДОДАТОК В

Специфікація програмного продукту

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет комп'ютерних наук

Кафедра програмної інженерії

СПЕЦИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
для проєкту з темою
«Програмна система для обміну речами»

Виконала ст. гр. ПЗПІ-20-2 Сорокіна Д. Є.
Керівник роботи доцент кафедри ПІ Ворочек О. Г.

2024

Рисунок В.1 – Сторінка 1 специфікації ПЗ

1 ВСТУП

1.1 Огляд продукту

Програмна система для обміну речами – це інтернет-платформа, що дозволяє користувачам обмінюватися предметами, які їм більше не потрібні, на речі, які вони бажають отримати. Ця система забезпечує безпечний, ефективний та зручний процес обміну речами між користувачами.

1.2 Мета

Створення зручної та ефективної платформи, яка спростить процес пошуку речей та допоможе позбутися речей, які не потрібні, шляхом обміну.

1.3 Межі

Система обмежена функціоналом обміну речами, управлінням обліковими записами користувачів, створенням оголошень товарів та спілкуванням між користувачами. Вона не включає функціонал продажу речей за гроші.

1.4 Посилання

1. Tidwell J., Brewer C., Valencia A. Designing interfaces: patterns for effective interaction design. O'Reilly Media, Incorporated, 2020. 500 с.
2. Progressive web apps | web.dev. *web.dev*. URL: https://web.dev/explore/progressive-web-apps.
3. Hengkyawan J. Clean Architecture in ASP .NET Core Web API. Medium. URL: https://juldhais.net/clean-architecture-in-asp-net-core-web-api-4e5ef0b96f99.
4. Norman D. A. Design of everyday things. MIT Press, 1998. 270 с.

1.5 Означення та абревіатури

PWA (Progressive Web Application) – прогресивний веб-додаток, що поєднує в собі кращі властивості веб-сайтів та мобільних додатків.

Рисунок В.2 – Сторінка 2 специфікації ПЗ

UI (User Interface) – інтерфейс користувача, система засобів, що забезпечують взаємодію користувача з програмою.

UX (User Experience) – досвід користувача, загальне враження та досвід, який отримує користувач під час взаємодії з продуктом.

API (Application Programming Interface) – інтерфейс програмування додатків, набір правил та механізмів, що дозволяють взаємодіяти різним програмним компонентам.

REST (Representational State Transfer) – стиль архітектури для створення веб-сервісів, що дозволяє взаємодіяти з ресурсами через стандартні HTTP-методи.

CRUD (Create, Read, Update, Delete) – базові операції для управління даними в інформаційних системах.

SPA (Single Page Application) – односторінковий додаток, що завантажує одну HTML-сторінку та динамічно оновлює контент на ній, не перезавантажуючи сторінку повністю.

Рисунок В.3 – Сторінка 3 специфікації ПЗ

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Програмна система для обміну речами має значний потенціал розвитку та впливу на різні сфери життя суспільства. Завдяки актуальності теми обміну речами, кількість користувачів системи може суттєво зрости, адже люди все частіше шукають способи зменшити свій екологічний слід та зекономити гроші. Люди більше спілкуватимуться та взаємодіятимуть між собою, що зміцнить соціальні зв'язки в межах окремих громад та міст. Екологічна стійкість платформи полягає в зменшенні кількості відходів, оскільки речі будуть повторно використовуватися, а не викидатися, що допоможе зменшити навантаження на сміттєзвалища та зберегти природні ресурси.

Використання сучасних технологій забезпечує високий рівень продуктивності та зручності користування, відкриваючи можливості для подальшого розвитку та впровадження нових функцій, таких як інтеграція з соціальними мережами. Програмна система також може залучити до співпраці різні організації, включаючи благодійні фонди, екологічні рухи, місцеві уряди та бізнеси, що підсилить вплив та розширить можливості платформи.

2.2 Функції продукту

Функціонал, доступний звичайному користувачу:

- реєстрація у системі;
- авторизація та автентифікація;
- редагування особистих даних;
- додавання оголошень про товари;
- редагування інформації товарів;
- видалення оголошень;
- пошук товарів;
- фільтрація товарів за певними параметрами;
- чат з іншими користувачами;

Рисунок В.4 – Сторінка 4 специфікації ПЗ

– створення обмінних угод;
– оцінювання обмінних угод.

Функціонал, доступний адміністратору:

– авторизація та автентифікація;
– створення категорій;
– видалення категорій;
– додавання кольорів;
– видалення кольорів.

2.3 Характеристики користувачів

У системі існують рівні доступу, або користувачів, які визначають, у кого є доступ до даних та функцій. Таких рівнів двоє, а саме:

– «user», який надається користувачам, які авторизовані у системі, у них є доступ до усіх основних функцій системи;
– «admin», який надається певному користувачу, який вже створений у системі, йому доступні функції адміністрування у системі;

Це розділення допомагає захистити систему від несанкціонованого доступу, забезпечує належне управління доступом, мінімізує ризики витоку конфіденційної інформації та і, загалом, забезпечує її стабільну та безпечну роботу.

2.4 Загальні обмеження

Для забезпечення стабільної та безпечної роботи програмної системи для обміну речами, користувачі мають дотримуватися певних обмежень та правил. Тож серверна частина даної системи має такі обмеження:

– користувачі можуть завантажити до 5 зображень для свого товару;
– певні поля, наприклад, назва товару має містити мінімум 10 та максимум 70 символів;
– користувач має пройти процес авторизації та аутентифікації, щоб отримати доступ до майже усіх функцій;

Рисунок В.5 – Сторінка 5 специфікації ПЗ

– до функцій, які пов'язані з категоріями та кольорами, має доступ тільки користувач з роллю адміністратора;
– деякі поля є обов'язковими для заповнення.

Клієнтська частина має такі обмеження:

– для роботи PWA у користувача має бути будь-який браузер, окрім Safari та Firefox;
– у користувача має бути стабільний інтернет;
– користувач має знати або українську, або англійську мову;
– у користувача буде можливість завантажити обмежену кількість фото для товару.

2.5 Припущення та залежності

Припущення:

– передбачається, що користувачі будуть зацікавлені в обміні речами як екологічно дружньому та економічно вигідному способі отримання потрібних речей;
– передбачається, що користувачі матимуть стабільний доступ до інтернету, необхідний для роботи веб-додатку та PWA;
– передбачається, що користувачі будуть активні у взаємодії один з одним;
– припускається, що команда розробників матиме достатні технічні ресурси, знання та час для розробки, впровадження та підтримки системи.

Залежності:

– система повинна відповідати законодавчим вимогам щодо зберігання та обробки даних, захисту персональної інформації та електронної комерції;
– розвиток системи залежить від активного зворотного зв'язку користувачів, що дозволяє визначати необхідні покращення та нові функції, які відповідають потребам користувачів.

Рисунок В.6 – Сторінка 6 специфікації ПЗ

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Повинен бути інтуїтивно зрозумілим, зручним та сучасним. Користувачі повинні мати можливість реєструватися та входити у систему, переглядати, шукати та фільтрувати доступні речі для обміну. Інтерфейс повинен дозволяти додавання нових речей для обміну з можливістю додавання фото та опису, створення обмінних угод з можливістю прийняття або відхилення їх.

3.1.2 Апаратний інтерфейс

Має підтримувати роботу на мобільних пристроях (смартфонах і планшетах) та персональних комп'ютерах. Він повинен бути сумісним з різними операційними системами.

3.1.3 Програмний інтерфейс

Повинен бути реалізований RESTful API для доступу до усіх функцій. Серверна частина повинна взаємодіяти з базою даних PostgreSQL.

3.1.4 Комунікаційний інтерфейс

Взаємодія між частинами програмної системи здійснюється завдяки підтримки REST API з використанням HTTP запитів та даних формату JSON. Кожна частина повинна реалізувати можливість обробки HTTP запитів завдяки відповідних бібліотек.

Взаємодія між частинами програмної системи здійснюється за допомогою REST API з використанням HTTP запитів та даних формату JSON. Кожна частина повинна реалізувати можливість обробки HTTP запитів.

Рисунок В.7 – Сторінка 7 специфікації ПЗ

3.1.5 Обмеження пам’яті

Система повинна ефективно використовувати доступну пам'ять, забезпечуючи стабільну роботу навіть під високими навантаженнями. Це включає оптимізацію запитів до бази даних та використання кешування для зменшення навантаження на сервер.

3.1.6 Операції

Основні операції включають реєстрацію та авторизацію користувачів, створення та управління оголошеннями, пошук та фільтрацію товарів, організацію обмінів, обмін повідомленнями між користувачами та підтримку адміністративних функцій системи.

3.1.7 Функції продукту

Функціональність системи включає всі основні операції, необхідні для повноцінного обміну речами між користувачами, з урахуванням безпеки, надійності та продуктивності.

3.2 Властивості програмного продукту

Програмне забезпечення повинно мати такі властивості, як надійність та захищеність даних, мати можливість масштабуватись, доносити необхідну інформацію до майбутніх користувачів, відповідати усім нормам користування ПЗ.

3.3 Атрибути програмного продукту

3.3.1 Надійність

Система повинна бути надійною та стійкою до збоїв, забезпечуючи безперервну роботу і доступність функцій для користувачів у будь-який час.

Рисунок В.8 – Сторінка 8 специфікації ПЗ

3.3.2 Доступність

Система повинна бути доступною для користувачів у будь-який час, мінімізуючи час простою та забезпечуючи безперебійну роботу.

3.3.3 Безпека

Система повинна забезпечувати захист даних користувачів та запобігати несанкціонованому доступу, включаючи використання шифрування даних та механізмів автентифікації.

3.3.4 Супроводжуванність

Код системи повинен бути зрозумілим та легким для підтримки, оновлення та розширення функціональності. Це включає дотримання принципів модульності та використання документованих API.

3.3.5 Переносимість

Система повинна підтримувати роботу на різних платформах та пристроях, забезпечуючи однаковий користувацький досвід незалежно від пристрою.

3.3.6 Продуктивність

Система повинна забезпечувати високу продуктивність, здатну обробляти велику кількість одночасних запитів без значних затримок або зниження якості обслуговування.

3.4 Вимоги бази даних

База даних повинна бути масштабованою та бути готовою до великого об’єму даних, який слід обробити у доволі швидкий проміжок часу.

3.5 Інші вимоги

Інші вимоги включають дотримання всіх стандартів та норм, необхідних

Рисунок В.9 – Сторінка 9 специфікації ПЗ

для забезпечення коректної роботи системи, включаючи вимоги до безпеки, конфіденційності даних та відповідність законодавчим нормам.

Рисунок В.10 – Сторінка 10 специфікації ПЗ

ДОДАТОК Г

Структура проєкту

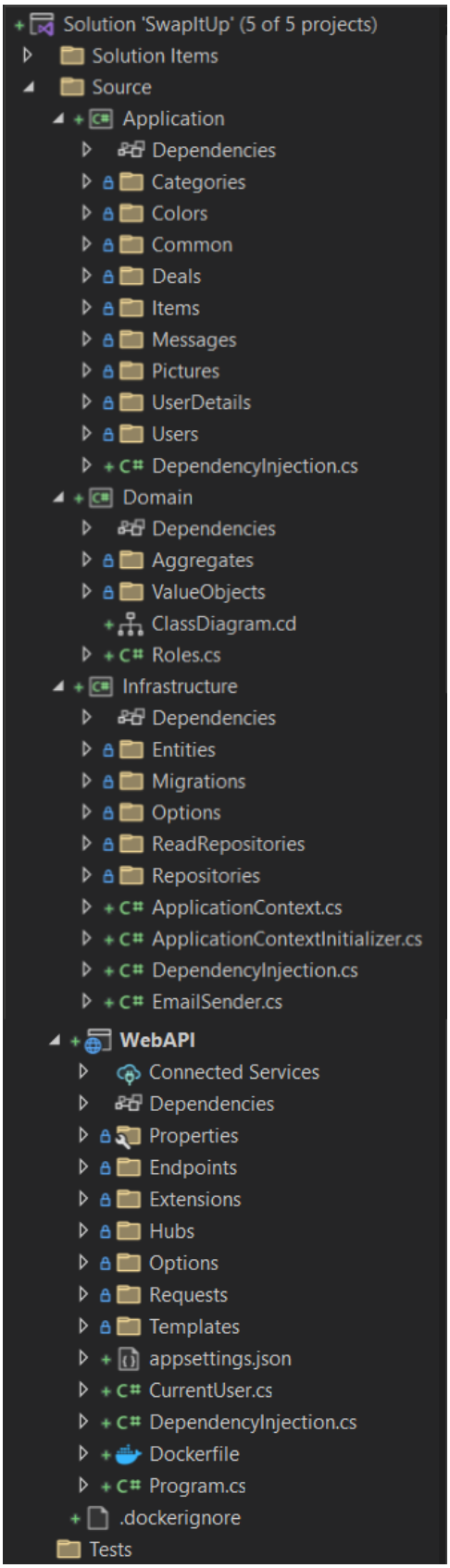

Рисунок Г.1 – Структура проєкту серверної частини

ДОДАТОК Д

Тези за темою кваліфікаційної роботи

УДК 004.9:504 DOI: https://doi.org/10.30837/IYF.IIS.2024.656

ПРОГРАМНА СИСТЕМА ДЛЯ ОБМІНУ РЕЧАМИ ЯК ЗАСІБ ПІДТРИМКИ 3R-ПІДХОДУ РОЗУМНОГО СПОЖИВАННЯ

Сорокіна Д. Є., Кащенко Ю. Є.
Науковий керівник – к.т.н., доцент Ворочек О. Г.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
e-mail: daria.sorokina@nure.ua, yukhym.kashchenko@nure.ua

This work describes a current problem of modern society – the inefficient use of resources due to insufficient exchange of things between people. The focus is on the need to create a software system to facilitate this process. This work justifies the need for this system due to the increase in consumption and waste, which has a negative impact on the environment. The software system offers an effective mechanism for sharing things between users to reduce waste and encourage a culture of resource use. The report also discusses the possible benefits to society, such as cost savings, promotion of resource recovery and support for social mutual assistance. It highlights the key features and benefits of the proposed software system aimed at optimizing the exchange of things and preserving the environment.

У сучасному світі все більше людей цінують екологічний спосіб життя та усвідомлюють необхідність вторинного використання товарів для збереження ресурсів і зменшення кількості відходів. Програмна система для обміну речами призначена для створення зручного та ефективного середовища для обміну різноманітними предметами між користувачами. Вона надає можливість людям обмінюватись з іншими користувачами речами, які їм більше не потрібні, що допомагає зменшити виробництво нових товарів і споживання ресурсів. Дана програма є не просто технічним інструментом, але і стратегічним компонентом, що реагує на зростаючі потреби користувачів [1]. Основна ідея даної системи полягає у створенні онлайн-платформи, на якій користувачі зможуть розміщувати оголошення про речі, якими вони хочуть обмінятися або які шукають. Система допомагає з'єднати людей, які потребують обміну, надаючи їм зручні інструменти для пошуку, перегляду та контакту з іншими користувачами. Домовленості про обмін можуть відбуватися у мобільному додатку або у веб-застосунку.

З огляду на зростаючу увагу до екологічних проблем, потрібність створення системи для обміну речами є стратегічним кроком до шляху сталого споживання. Система сприяє переходу від традиційної моделі споживання, заснованої на безперервному придбанні нових товарів, до більш сталої моделі, яка базується на збереженні та використанні наявних ресурсів. Чим більше товарів обмінюється між користувачами, тим менше товарів потрапляє на сміттєзвалище. Це допомагає зменшити обсяги

Рисунок Д.1 – Перша сторінка тез

відходів та викидів, пов'язаних зі звичайним викиданням непотрібних речей.

У поточних реаліях, коли бренди прагнуть якомога швидше продати свій одяг і у якомога більших кількостях, випускаючи нові колекції наче конвеєр, індустрія моди є одним з головних причин забруднення нашого довкілля [2]. На швейну промисловість припадає 10% від загальних викидів вуглекислого газу щороку, вона також є другим найбільшим споживачем прісної води, на її частку припадає 20% усіх промислових забруднень води [3]. Таким чином розробка системи для обміну речами не тільки дозволяє уникнути надмірного споживання, а разом з цим зменшує споживання природних ресурсів та викиди вуглецю, що пов'язані з виробництвом та транспортуванням.

Ще одним популярним явищем на сьогодні є «буккросинг», процес звільнення книг, що рятує книги від безцільного знаходження на полицях [4]. Програмна система для обміну речами допоможе буккросингу розширити географічно область своїх дій, що призведе до збільшення кількості книг, якими можна обмінюватися та розширення меж обміну. Крім того, вона полегшить відстеження руху книг та також прибере необхідність фізичних зустрічей для учасників. Це зробить процес обміну більш ефективним та зручним.

Наша система пропонує можливість використання як мобільного застосунку, так і веб-застосунку для пошуку товарів та проведення обмінів. Для розробки серверної частини програмної системи використовуються мови C# та технології ASP.NET Core 7, у якості бази даних обрано PostgreSQL. Клієнтська частина реалізується за допомогою технології React на основі мови JavaScript. Для мобільної версії системи доцільне використання фреймворку React Native.

Аналогічні програмні рішення пропонують різні підходи та можливості, відзначаються власними перевагами та недоліками. Представником такого типу систем є відома платформа для торгівлі OLX. Її перевагами є: широка популярність даної платформи і доступність її у багатьох країнах, але разом з цим головним недоліком є те, що OLX не спеціалізується на обміні товарами, а, в першу чергу, є торговельним майданчиком. Іншим доступним аналогом є Facebook Marketplace. Його головною перевагою є те, що дана платформа є частиною соціальної мережі Facebook, що надає велике охоплення та зручну комунікацію. Разом з цим це є і недоліком даного аналогу, так як багато людей не мають акаунту у Facebook, а створювати його лише для можливості доступу до Marketplace буде для когось недоречним. Також ще одним недоліком є те, що на даній платформі більш розповсюджена торгівля, а не обмін.

Усе вищеозначене зумовлює доцільність запропонованої програмної системи задля вирішення задач формування більш зручного та

Рисунок Д.2 – Друга сторінка тез

сприятливого для людини середовища, в якому панує ефективне використання та споживання ресурсів.

Список використаних джерел:

1. Куценко А., Колесник О. Розробка компоненту системи веб-застосунку «каталог одягу». Комп'ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві: Всеукр. наук.-практ. конф. здобувачів вищ. освіти і молодих вчених, 22 листопада 2023 р. Харків. нац. автомоб.-дор. ун-т., Харків, 2023. С. 198–201.

2. Вплив швидкої моди на екологію та цінність сприйняття одягу. Dubhumans. URL: https://dubhumans.com/vplyv-shvydkoi-mody-na-ekolohiiu-ta-tsinnist-spryiniattia-odiahu/ (дата звернення: 29.02.2024).

3. Industrial water usage to produce these items | the71percent. The 71 Percent. URL: https://www.the71percent.org/industrial-water-usage/ (дата звернення: 29.02.2024).

4. Буккросинг. URL: https://pulyny.school.org.ua/bukkrosing-09-38-29-05-01-2022/ (дата звернення: 29.02.2024).

Рисунок Д.3 – Третя сторінка тез