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 реалізації параметричного пошуку партнерів для домашніх улюбленців з метою розведення породистих тварин. Серверна частина
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-11

Ілля РОМАНОВСЬКИЙ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Гліб ТЕРЕЩЕНКО
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 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)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Романовському Іллє Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для реалізації параметричного пошуку партнерів для домашніх улюбленців з метою розведення породистих тварин. Серверна частина

Затверджена наказом по університету від 19.05.2025р. № 397 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 05.06.2025

3. Вихідні дані до роботи Розробити серверну частину вебзастосунку для параметричного підбору партнерів серед домашніх тварин з метою їх розведення. Реалізацію виконати мовою програмування Python з використанням вебфреймворку Django.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної області, виявлення та вирішення проблем, постановка задачі, формування вимог до програмної системи, архітектура та проєктування, опис прийнятих програмних рішень, реалізація основних компонентів, висновки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	22.05.2025	*виконано*
3	Проектування ПЗ	28.05.2025	*виконано*
4	Розробка ПЗ	30.05.2025	*виконано*
5	Тестування ПЗ	06.06.2025	*виконано*
6	Оформлення пояснювальної записки	07.06.2025	*виконано*
7	Підготовка презентації та доповіді	08.06.2025	*виконано*
8	Попередній захист	09.06.2025	*виконано*
9	Нормоконтроль, рецензування	11.06.2025	*виконано*
10	Здача роботи у електронний архів	11.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	12.06.2025	*виконано*

Дата видачі завдання «19» «травня» 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Гліб ТЕРЕЩЕНКО
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 40 стор., 6 рис., 16 джерел.

БАЗА ДАНИХ, ДОМАШНІ ТВАРИНИ, ПІДБІР ПАРТНЕРІВ, РОЗВЕДЕННЯ, API, DJANGO, MONGODB, PYTHON, REST, SPA

У кваліфікаційній роботі розроблено веб-застосунок, який дозволяє здійснювати параметричний підбір партнерів для домашніх тварин з урахуванням породи, віку, місцезнаходження та інших характеристик. Актуальність роботи зумовлена потребою власників породистих тварин у зручному й функціональному інструменті для пошуку відповідних пар з метою розведення.

Проєкт реалізовано як односторінковий застосунок (SPA), у якому фронтенд створено з використанням бібліотеки React, а серверну частину – за допомогою фреймворку Django. Для передачі даних між компонентами застосовано REST API, а для реалізації чату між користувачами – WebSocket-з’єднання. Застосунок підтримує реєстрацію та авторизацію користувачів, створення профілів тварин із фото, параметричний пошук, перегляд інформації про власників, а також обмін повідомленнями.

У процесі розробки окрема увага приділялася забезпеченню якості ПЗ. Проведено ручне тестування основного функціоналу, а також реалізовано набір автоматизованих тестів для перевірки стабільності інтерфейсу та обробки помилок.

Розроблену систему впроваджено на публічному GitHub-репозиторії. Її результати можуть бути використані як основа для подальшого розвитку спеціалізованих платформ у сфері зоотехнічних або клубних сервісів для тварин.

ABSTRACT

Explanatory note to the bachelor qualification thesis, 40 pages, 6 figures, 16 references.

DATABASE, DOMESTIC ANIMALS, PARTNER MATCHING, BREEDING, API, DJANGO, MONGODB, PYTHON, REST, SPA

This qualification thesis presents the development of a web application that enables parametric partner matching for domestic animals, considering breed, age, location, and other characteristics. The relevance of the work is determined by the need of pedigree pet owners for a convenient and functional tool to search for suitable breeding pairs.

The project is implemented as a single-page application (SPA), where the frontend is developed using the React library and the backend is built with the Django framework. Data exchange between components is performed via REST API, while user messaging is handled through WebSocket connections. The application supports user registration and authentication, creation of pet profiles with photos, parametric search, viewing information about pet owners, and message exchange.

During development, special attention was paid to software quality assurance. Manual testing of the main functionality was conducted, and a set of automated tests was implemented to verify interface stability and error handling.

The developed system is published on a public GitHub repository. Its results can serve as a basis for further development of specialized platforms in the field of zoological or club services for animals.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CSRF – Cross-Site Request Forgery

REST – Representational State Transfer

DRF – Django REST Framework

SPA – Single-Page Application

CORS – Cross-Origin Resource Sharing

API – Application Programming Interface

ORM – Object-Relational Mapping

HTTP – HyperText Transfer Protocol

WSGI – Web Server Gateway Interface

ASGI – Asynchronous Server Gateway Interface

CRUD – Create, Read, Update, Delete

JWT – JSON Web Token

UML – Unified Modeling Language

ВСТУП

У сучасному цифровому середовищі дедалі чіткіше простежується тенденція до створення вузькоспеціалізованих інформаційних систем, які вирішують конкретні прикладні задачі, забезпечуючи водночас гнучкість, захищеність і масштабованість. Однією з таких задач є організація контрольованої взаємодії між користувачами в умовах, коли критичне значення мають точність пошуку, достовірність даних, а також швидкість та надійність системи у високонавантаженому середовищі. У цій категорії особливу нішу займають платформи, призначені для пошуку партнерів з метою розведення домашніх тварин, адже цей процес передбачає не просто обмін контактами, а складну багатокрокову логіку взаємодії між власниками, контроль відповідності біологічних параметрів тварин, перевірку сумісності за кількома критеріями та збереження конфіденційності особистих даних.

У рамках цієї кваліфікаційної роботи об’єктом розробки стала серверна частина програмного комплексу PetBreedingApp – спеціалізованого веб-застосунку, який реалізує повноцінну взаємодію між зареєстрованими користувачами з метою підбору відповідних тварин для племінного розведення. Розроблена система дозволяє керувати анкетами тварин, забезпечує параметричний пошук за заданими критеріями (вік, порода, стать, колір шерсті тощо), реалізує механізми реєстрації, автентифікації та авторизації користувачів, а також інтегрується з модулем обміну повідомленнями в реальному часі на основі WebSocket-протоколу.

На відміну від більшості існуючих сервісів, що обмежуються статичним відображенням оголошень, дана система орієнтована на динамічну взаємодію, що передбачає як одноразові дії (наприклад, створення чи редагування анкети), так і тривалі процеси: встановлення симпатій, ведення діалогу, накопичення відгуків та подальше спостереження за історією взаємодій. Це зумовлює необхідність використання сучасних інструментів, які дозволяють реалізовувати обробку запитів у реальному часі, зберігати складні зв’язки між сутностями, забезпечувати цілісність даних навіть при паралельному доступі до ресурсів системи.

Серверна частина застосунку розроблялась на основі фреймворку Django, який було обрано за його стабільність, гнучкість, активну підтримку спільноти розробників і наявність вбудованих механізмів безпеки, автентифікації та управління сесіями. Для реалізації API-запитів використано модулі Django REST Framework (DRF), що дозволило створити повноцінну RESTful-архітектуру з підтримкою серіалізації, фільтрації, валідації даних і перевірки прав доступу. Крім того, до складу серверної частини було інтегровано Django Channels, що забезпечує підтримку асинхронної комунікації через WebSocket, необхідну для реалізації обміну повідомленнями між користувачами без перезавантаження сторінки [1].

Система побудована за модульним принципом, де кожен функціональний блок – реєстрація користувачів, робота з анкетами тварин, пошук за параметрами, обмін повідомленнями – реалізовано у вигляді окремого Django-додатку, що спрощує супровід, тестування та масштабування проєкту. Комунікація між клієнтською та серверною частиною здійснюється через стандартизовані HTTP-запити та WebSocket-повідомлення, що дозволяє підтримувати сучасний рівень інтерактивності та швидкодії.

У процесі реалізації проєкту також було приділено значну увагу захисту даних: впроваджено механізми CSRF-захисту, контроль за сесіями, перевірку прав доступу до ресурсів, обмеження доступу на рівні запитів, що потребують ідентифікації. Дані про користувачів, тварин і повідомлення зберігаються у структурованому вигляді з урахуванням вимог до логічної цілісності, а архітектура сховища адаптована для підтримки вкладених структур (наприклад, кількох фотографій тварини) та подальшого масштабування.

Таким чином, розроблена серверна частина програмного комплексу PetBreedingApp забезпечує надійний, захищений і масштабований бекенд для системи, орієнтованої на автоматизований підбір партнерів для домашніх тварин. Реалізований функціонал дозволяє знизити бар’єри для користувачів, мінімізувати ризики помилок при взаємодії, забезпечити цілісність даних і задовольнити вимоги до реального використання системи в умовах зростаючого навантаження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Цифрові технології стрімко змінюють способи взаємодії між людьми в найрізноманітніших сферах. Розведення домашніх тварин – одна з тих галузей, де традиційні підходи, засновані на особистих контактах, оголошеннях або випадкових публікаціях у соцмережах, дедалі більше поступаються сучасним, структурованим сервісам. В умовах потреби в контрольованому племінному розведенні це особливо помітно: недостатньо просто знайти іншу тварину тієї ж породи – важливо враховувати вік, стан здоров’я, генетичну сумісність і, за можливості, доступ до актуальних даних про партнера.

У центрі уваги цієї роботи – створення системи, яка дозволяє власникам тварин зручно та безпечно шукати відповідних партнерів для розведення. Йдеться не лише про реалізацію базового функціоналу типу створення чи редагування анкет. Платформа має забезпечувати повний цикл взаємодії – від реєстрації користувача до пошуку, фільтрації, спілкування і підтримки контролю якості даних. Це включає автентифікацію, розмежування доступу, зберігання зображень, обмін повідомленнями та розширений параметричний пошук за характеристиками тварин.

При проєктуванні системи надзвичайно важливо враховувати, що користувачі мають різні ролі та очікування. Звичайні зареєстровані користувачі створюють і редагують анкети тварин, надсилають повідомлення й взаємодіють з іншими власниками. Адміністратори, своєю чергою, керують усією інфраструктурою – модерують контент, реагують на порушення й можуть блокувати облікові записи. Є також категорія неавторизованих відвідувачів, які мають доступ виключно до функцій реєстрації та входу в систему. Тому реалізація серверної логіки має враховувати різні сценарії використання та відповідні права доступу.

З боку бекенду система має працювати з кількома взаємопов’язаними сутностями. Наприклад, кожна тварина асоціюється з конкретним користувачем і містить набір атрибутів: вид, породу, забарвлення, вік, стать, ілюстрації тощо.

Особливої уваги потребує коректна обробка ситуацій, коли об’єкти видаляються чи змінюються. Якщо користувача буде видалено, необхідно визначити, що робити з його анкетами, повідомленнями і пов’язаними зображеннями – автоматичне чи умовне каскадне видалення має налаштовуватись відповідно до обраної політики консистентності.

Ще один важливий аспект – здатність системи працювати зі зростаючим навантаженням. Якщо проєкт розрахований на велику кількість користувачів, необхідно оптимізувати зберігання даних і взаємодію з базою. Це передбачає використання select_related, prefetch_related, обмеження надлишкових запитів, а в майбутньому – підтримку горизонтального масштабування. На цьому рівні бекенд переходить із простої логіки до повноцінної архітектурної основи всього застосунку.

У системі, що зберігає персональні дані – включаючи контактну інформацію, особисті повідомлення та зображення – не може бути компромісів із безпекою. Захист має будуватись на кількох рівнях: CSRF-токени, правильна CORS-конфігурація, чітко налаштовані HTTP-заголовки, обмеження доступу до API-ендпоінтів за допомогою permission-класів. У разі навіть часткової компрометації будь-якого компонента, критичні дані повинні залишатись недоступними для сторонніх.

Окрему роль відіграє підтримка сучасних каналів комунікації. REST API – це необхідний базовий рівень, але в системі, де важлива жива взаємодія між користувачами, варто реалізовувати WebSocket-протокол для обміну повідомленнями в реальному часі. Це передбачає інший підхід до обробки запитів – асинхронну логіку, event loop, авторизацію на рівні WS-з'єднання та окрему обробку прав доступу до чату.

Узагальнюючи, можна сказати, що предметна область накладає комплексні вимоги до структури системи: вона має бути безпечною, масштабованою, підтримувати різні типи взаємодії та забезпечувати коректну роботу в багатокористувацькому режимі. Серверна частина тут – не просто інструмент, а ключовий елемент, який забезпечує узгодженість усіх компонентів системи. Саме

тому реалізація проєкту спирається на сучасні технології, перевірені архітектурні підходи й реальні виклики, з якими стикається галузь.

1.2 Виявлення та вирішення проблем

Сьогодні процес підбору партнерів для розведення тварин усе ще залишається складним і, що особливо важливо, слабо формалізованим у межах цифрової екосистеми. Попри помітну кількість доступних ресурсів – тематичних форумів, спільнот у соціальних мережах, платформ оголошень – користувачі не мають у своєму розпорядженні єдиного, зручного та безпечного інструменту, який дозволяв би проводити цей процес системно. Більше того, більшість таких рішень не передбачає централізованої перевірки користувачів, а інформація про самих тварин часто подається у вільній, ненормованій формі. У результаті потенційний власник має обмежені можливості відсіяти нерелевантні варіанти або переконатись у достовірності наданих даних.

На рівні архітектури такі платформи зазвичай обмежуються статичним відображенням анкет або оголошень, без інтерактивних функцій та повноцінного API. Поширеною є практика реалізації логіки у вигляді великого монолітного блоку, в якому важко розмежувати відповідальність окремих компонентів. У такій структурі будь-яка модифікація або масштабування вимагає значного втручання, що не тільки ускладнює підтримку, але й знижує загальну гнучкість системи. Крім того, відсутність чіткого розподілу ролей користувачів та прав доступу створює потенційні вразливості – наприклад, коли будь-який зареєстрований користувач має можливість змінювати або переглядати дані, до яких він не повинен мати доступ.

Ще одна типова проблема – це надмірна залежність від ручної модерації. У багатьох системах відсутні інструменти для оперативного реагування на скарги чи порушення. У випадку, коли користувач порушує правила або подає неправдиву інформацію, адміністрація не має достатньо швидких і зручних засобів втручання в реальному часі. Відповідальність за автентичність контенту фактично перекладається на самих користувачів, що знижує рівень довіри до платформи.

Щоб подолати ці обмеження, необхідно забезпечити централізацію процесів обробки та перевірки інформації. Це можливо лише за умови впровадження якісно спроєктованої серверної частини, яка дозволяє створювати, зберігати, оновлювати й перевіряти анкети тварин без перешкод і в режимі постійної доступності. Особливу увагу слід приділити забезпеченню цілісності даних – система має зберігати послідовність і несуперечність інформації навіть у разі некоректного запиту, перерваної сесії або конфлікту між паралельними змінами.

Значну проблему становить і слабка підтримка асинхронної взаємодії. У багатьох випадках обмін повідомленнями між користувачами реалізується через сторонні сервіси – email або месенджери – що унеможливлює формування цілісної комунікаційної моделі всередині самої платформи. Відсутність протоколів реального часу, таких як WebSocket, позбавляє систему ключової сучасної функції – миттєвої реакції. У підсумку користувач змушений або чекати відповіді поза межами сайту, або шукати альтернативні канали комунікації, що суперечить ідеї централізації.

З погляду захисту даних ситуація в багатьох реалізаціях викликає занепокоєння. Користувацькі й тваринні профілі часто зберігаються без належної авторизації, захисту від CSRF-атак або контролю доступу відповідно до ролі. Це відкриває шлях для серйозних зловживань, включно з несанкціонованим редагуванням чужих профілів або масовим вилученням інформації через погано налаштоване API. Безпечна система повинна реалізовувати перевірку сесій, накладати обмеження на HTTP-запити залежно від контексту, а також унеможливлювати взаємодію з об’єктами, якщо користувач не має відповідних прав.

Ще одна проблема, яку не можна ігнорувати – це слабка розширюваність. Багато платформ не враховують можливість розвитку: навіть додавання нового поля в анкету чи розширення списку фільтрів часто вимагає ручного втручання в код і структуру БД. У сучасній системі будь-яке масштабування – функціональне чи технічне – має реалізовуватись без зупинки роботи, без переробки всієї інфраструктури та без втрати зворотної сумісності.

Таким чином, проблематика, на яку спрямовано цю роботу, охоплює одразу кілька критично важливих напрямів: відсутність єдиного безпечного сервісу для взаємодії власників тварин; слабку підтримку асинхронної та паралельної взаємодії; проблеми з цілісністю, захищеністю й масштабованістю серверної частини. Саме ці виклики стали відправною точкою для формування вимог до нової системи, яка має поєднати функціональну повноту, гнучкість та сучасні механізми обробки й захисту даних.

1.3 Постановка задачі

Метою даної роботи є розробка серверної частини веб-застосунку для підбору партнерів з розведення домашніх тварин. Система повинна забезпечувати безпечну, масштабовану та зручну платформу для взаємодії користувачів, включаючи реєстрацію, авторизацію, створення та перегляд анкет тварин, а також обмін повідомленнями між користувачами.

Для досягнення цієї мети необхідно реалізувати наступні завдання:

- розробити RESTful API, що забезпечує повний цикл CRUD-операцій для основних сутностей системи: користувачів, анкет тварин та повідомлень;
- забезпечити аутентифікацію та авторизацію користувачів, включаючи захист від CSRF-атак та управління сесіями;
- реалізувати асинхронний обмін повідомленнями між користувачами в режимі реального часу за допомогою WebSocket;
- створити систему управління правами доступу, що розрізняє ролі користувачів (звичайний користувач, адміністратор) та обмежує доступ до відповідних ресурсів;
- забезпечити зберігання та обробку зображень, пов'язаних з анкетами тварин, включаючи завантаження, зберігання та відображення;
- реалізувати систему фільтрації та пошуку анкет тварин за різними параметрами (порода, вік, стать тощо);
- забезпечити логування та обробку помилок для моніторингу та підтримки стабільності системи [2].

Для реалізації зазначених завдань було обрано наступні технології та інструменти:

- мова програмування Python: обрана за її простоту, читабельність та широку екосистему бібліотек, що прискорюють розробку веб-застосунків;
- фреймворк Django: забезпечує швидку розробку веб-застосунків з використанням вбудованих інструментів для аутентифікації, управління сесіями та адміністративного інтерфейсу;
- DRF: розширення для Django, що дозволяє швидко створювати RESTful API з підтримкою серіалізації, пагінації, фільтрації та управління правами доступу [3];
- Django Channels: додає підтримку асинхронного програмування в Django, що необхідно для реалізації WebSocket-з’єднань та обміну повідомленнями в реальному часі;
- MongoDB: використовується як база даних для зберігання повідомлень, що дозволяє ефективно працювати з неструктурованими даними та забезпечує масштабованість [4];
- Git та GitHub: використовуються для контролю версій, спільної роботи над кодом та управління випусками [5].

Вибір зазначених технологій обумовлено їхньою популярністю, активною спільнотою розробників, наявністю детальної документації та підтримкою необхідного функціоналу для реалізації поставлених завдань. Django та DRF забезпечують швидку розробку та підтримку RESTful API, Django Channels дозволяє реалізувати асинхронний обмін повідомленнями, а використання MongoDB забезпечує гнучкість у роботі з різними типами даних [6].

Таким чином, обраний стек технологій дозволяє ефективно реалізувати серверну частину веб-застосунку для підбору партнерів з розведення домашніх тварин, забезпечуючи необхідний функціонал, безпеку та масштабованість системи.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Проєктована система має відповідати специфічним вимогам, які охоплюють функціональні, нефункціональні, інформаційні та технічні аспекти. Серверна частина, як центральний вузол обробки запитів і бізнес-логіки, повинна забезпечити коректне виконання всіх критичних операцій, відповідати принципам безпечного зберігання даних, підтримувати модульність та масштабованість.

Функціональні вимоги стосуються тих можливостей, які система повинна надавати користувачам. Передусім необхідно забезпечити аутентифікацію та авторизацію користувачів із перевіркою облікових даних, збереженням стану сесії та захистом від CSRF-атак. Також реалізовано функціонал реєстрації, верифікації паролів, відновлення доступу. Додатково має бути реалізовано повноцінний CRUD-функціонал для роботи з анкетами тварин – створення, перегляд, редагування, видалення – із валідацією даних та контролем прав доступу.

Одним із центральних функціональних блоків є фільтрація анкет за параметрами: вік, порода, стать, наявність зображень. Система повинна обробляти запити на пошук у реальному часі, з урахуванням складених умов, та повертати структуровані дані для подальшої обробки на клієнті. Також у складі функціональності бекенду реалізовано логіку обміну повідомленнями між користувачами, для чого створено окремий WebSocket-контролер, що забезпечує асинхронну передачу даних, а також обробку подій на підключення, надсилання й отримання повідомлень.

До нефункціональних вимог відноситься стабільність та безперервність роботи системи в умовах багатокористувацького середовища. Сервер має відповідати принципам fail-safe при обробці некоректних запитів, запобігати падінню застосунку при помилках взаємодії з БД, а також реалізовувати централізовану обробку виключень. У проєкті застосовано обробку помилок за допомогою DRF-обгорток, а також вбудованих middleware, що дозволяють забезпечити інформативну реакцію системи у разі виняткових ситуацій.

Окремо слід виділити вимоги до безпеки. Система не повинна дозволяти неавторизованим користувачам виконувати дії, які потребують ідентифікації,

зокрема перегляд або зміну анкет тварин, а також надсилання повідомлень. Контроль прав доступу реалізовано на рівні permissions-класів у DRF, а для захисту від міжсайтових атак використовується механізм перевірки CSRF-токенів [7, 8]. Передбачено обмеження на доступ до ресурсів, які належать іншим користувачам.

Інформаційні вимоги включають структуру даних, яка використовується для обміну між клієнтською і серверною частинами. Зокрема, кожна тварина описується набором атрибутів: ім’я, вид, порода, вік, стать, колір шерсті, а також список фото. Пов’язані з користувачем дані включають лише базову облікову інформацію, включаючи username, email, пароль, рейтинг і список тварин. Повідомлення містять час надсилання, відправника, одержувача та текст. Серіалізація і десеріалізація таких даних реалізована у відповідних DRF-серіалізаторах.

Щодо технічних вимог, система повинна бути розгорнута на сервері з підтримкою Python 3.10+, середовища Django 5+, та мати доступ до сховищ бази даних MongoDB. Усі залежності структуруються у файлі requirements.txt, що дозволяє спростити розгортання проєкту. API повинно бути побудоване за REST-парадигмою, що дозволяє легко інтегрувати серверну частину з будь-яким сучасним клієнтським фреймворком.

Таким чином, сформульовані вимоги забезпечують стабільне функціонування серверної частини проєкту та дозволяють досягнути узгодженості з потребами клієнтської частини, а також з перспективою подальшого масштабування функціональності.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ

3.1 UML проєктування ПЗ

Проєктування серверної частини системи розпочалося з побудови діаграм варіантів використання, які ілюструють основні сценарії взаємодії з сервісом з боку різних категорій користувачів. Це дозволило формалізувати функціональні вимоги до логіки обробки запитів і визначити ролі взаємодії. Зокрема, було розроблено три окремі діаграми прецедентів – для незареєстрованого користувача, зареєстрованого користувача та адміністратора. Вони охоплюють ключові сценарії: реєстрація, вхід у систему, перегляд анкет тварин, створення та редагування власних анкет, обмін повідомленнями, а також доступ адміністратора до перегляду та редагування даних (див. рис. 3.1–3.3).

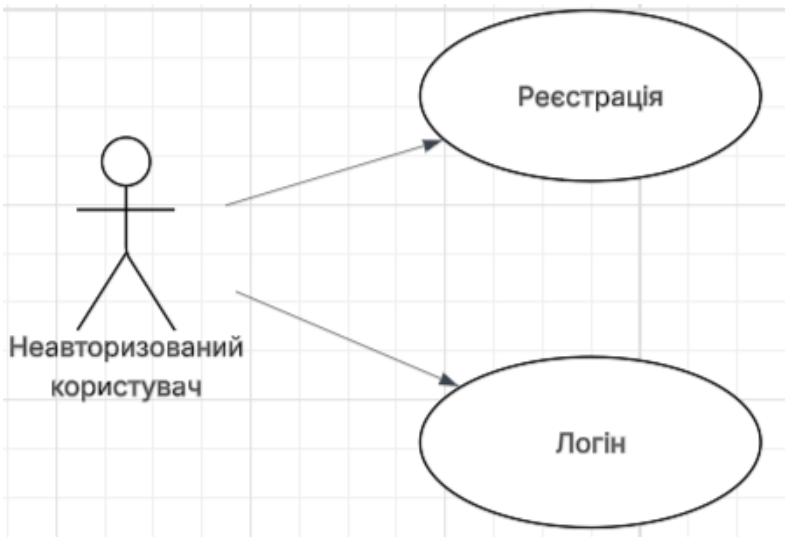


Рисунок 3.1 – Діаграма прецедентів для неавторизованого користувача (рисунок виконано самостійно)

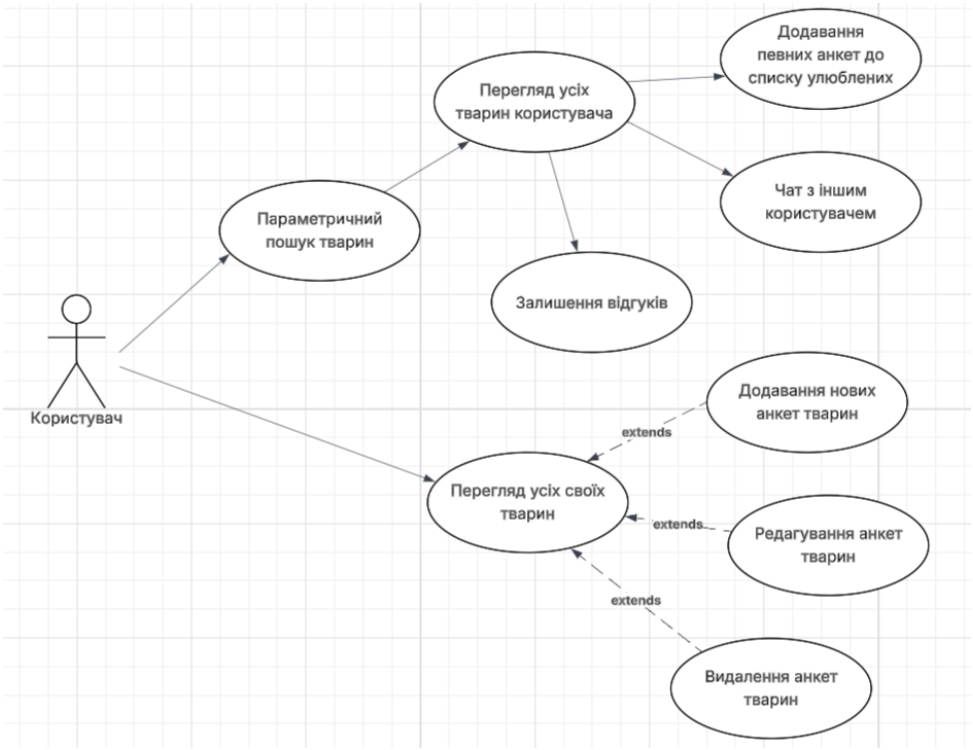


Рисунок 3.2 – Діаграма прецедентів для авторизованого користувача (рисунок виконано самостійно)

Рисунок 3.3 – Діаграма прецедентів адміністратора БД (рисунок виконано самостійно)

Для відображення взаємозв’язків між модулями системи було сформовано компонентну діаграму, що демонструє організацію серверної частини як сукупності окремих Django-додатків. Усі вони взаємодіють із центральним маршрутизатором API та базою даних MongoDB. Кожен модуль має власну структуру, що включає маршрути, обробники запитів, моделі та, де потрібно, WebSocket-консумери. Структура охоплює модулі для роботи з користувачами

(users), анкетами тварин (pets), логікою підбору (matching) та обміном повідомленнями (chats) (див. рис. 3.4).

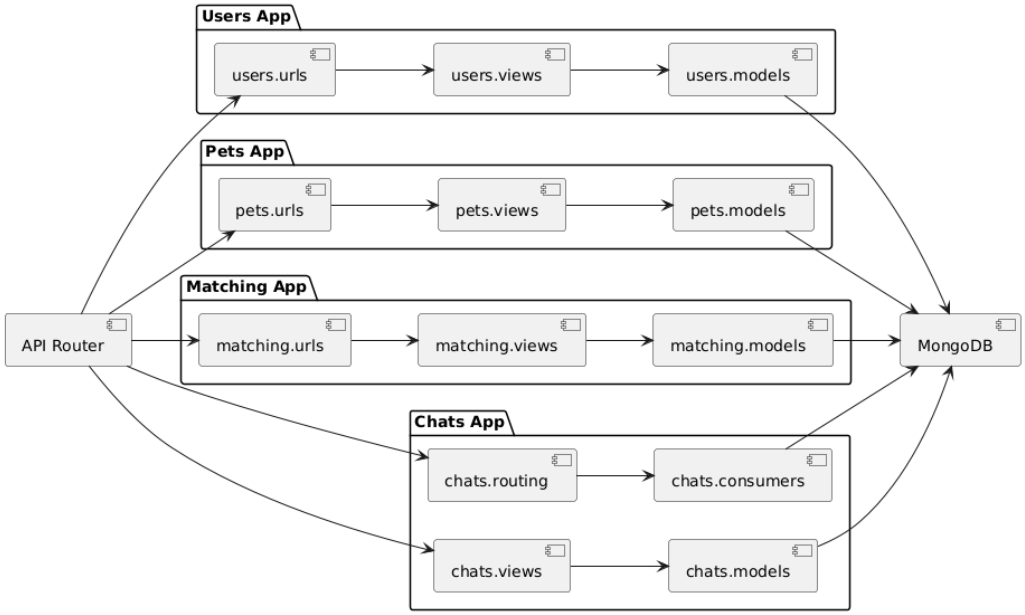


Рисунок 3.4 – Діаграма компонентів серверної частини (рисунок виконано самостійно)

Для опису фізичного розгортання системи була створена діаграма, яка демонструє основні вузли: клієнтський застосунок, сервер застосунку на основі Django з підтримкою WebSocket, та сервер бази даних MongoDB. Відображено основні канали взаємодії – HTTP-запити та запити до бази даних (див. рис. 3.5).

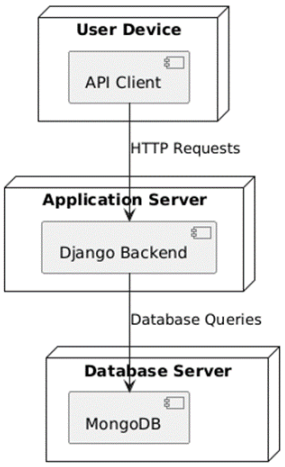


Рисунок 3.5 – Діаграма розгортання серверної частини (рисунок виконано самостійно)

Окремо було розроблено ER-діаграму структури бази даних MongoDB, яка відображає ключові сутності: користувач, тварина, повідомлення, відгук, платіж, фото тварин і обрані анкети. Визначено типи даних для кожного поля, а також зв’язки між сутностями, включаючи множинність і напрямок залежностей. Всі зв’язки моделюються у межах документно-орієнтованої схеми MongoDB з урахуванням принципів гнучкого зберігання даних (див. рис. 3.6).

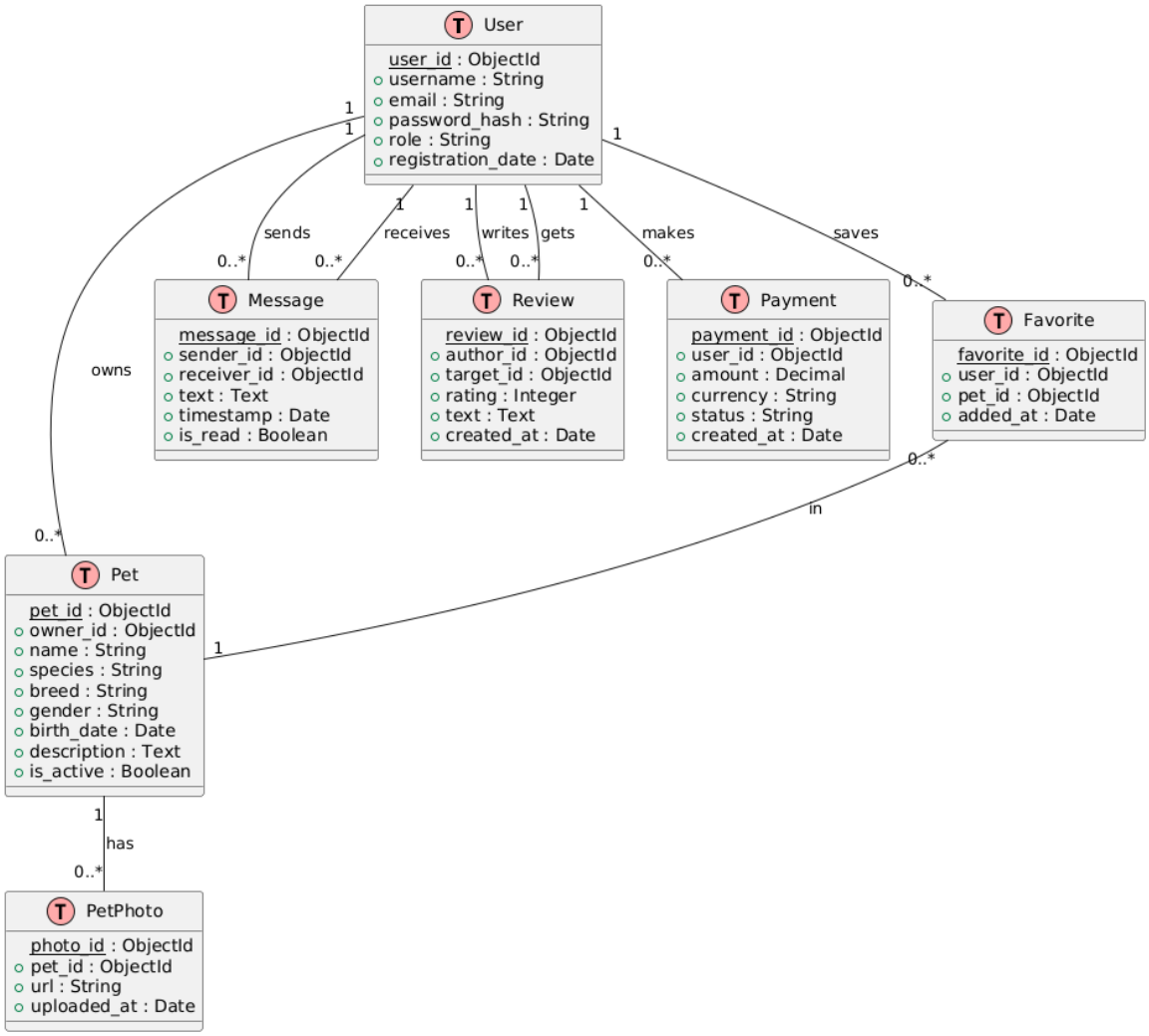


Рисунок 3.6 – ER-діаграма структури бази даних MongoDB (рисунок виконано самостійно)

Таким чином, UML-проєктування дозволило структурувати серверну логіку системи, визначити чіткі зони відповідальності між модулями та забезпечити узгодженість моделі з реальною реалізацією. Зв’язки між діаграмами відображають різні рівні абстракції: від сценаріїв використання до фізичної реалізації, що дозволяє впевнено переходити до подальших етапів розробки.

3.2 Проєктування архітектури ПЗ

Серверну частину системи побудовано за модульним принципом із чітким поділом на функціональні блоки. У центрі – Django, який слугує основою для реалізації REST API, обробки запитів, авторизації, перевірки прав доступу та взаємодії з базою даних.

Загалом архітектура поділена на чотири основні модулі, кожен з яких оформлений як окремий Django-додаток:

- users – відповідає за створення облікових записів, вхід у систему та зберігання даних користувачів;
- pets – реалізує логіку роботи з анкетами тварин: створення, перегляд, редагування, видалення;
- matching – містить механізм фільтрації анкет за вказаними параметрами;
- chats – обробляє обмін повідомленнями між користувачами.

Кожен із модулів має свій набір моделей (models.py), представлень (views.py) і маршрутів (urls.py). У модулі chats також є окремі файли для роботи з WebSocket (consumers.py та routing.py), що дає змогу передавати повідомлення в реальному часі.

Обмін даними між фронтендом і бекендом реалізовано через REST API, побудоване на Django REST Framework. Усі запити проходять через відповідні endpoint'и, де обробляються у view-функціях або класах. Для зручності й безпеки дані серіалізуються, а також проходять перевірку на відповідність очікуваному формату.

Захист запитів забезпечується через CSRF-механізм та сесійне зберігання інформації про користувача. В проєкті не використовується JWT, що спрощує реалізацію і знижує поріг входу для розробників. Усі важливі дії – такі як редагування або видалення анкет – дозволені лише тим користувачам, які їх створили, що контролюється через DRF-перевірки прав доступу [9].

Маршрутизація всередині проєкту організована просто й зручно: у кореневому urls.py головного проєкту підключені всі додатки зі своїми префіксами

– /api/users/, /api/pets/ тощо [10]. Це дозволяє чітко розділити функціональність і не змішувати логіку різних частин системи.

Окремо слід згадати реалізацію WebSocket-зв’язку. Коли користувач надсилає повідомлення в чаті, воно обробляється в ChatConsumer, який перевіряє автентичність і передає повідомлення адресату [11]. Все це відбувається без перезавантаження сторінки, у реальному часі. Повідомлення зберігаються в базі, щоб їх можна було переглядати пізніше.

Для зберігання даних використовується MongoDB. Це документно-орієнтована база, яка добре підходить для зберігання анкет із вкладеними фото, масивами полів тощо. Усі модулі взаємодіють із MongoDB через стандартні Django-моделі, адаптовані під відповідну схему. Кожна сутність має свій набір полів та зв’язків, який логічно узгоджений з форматом запитів і відповідей у клієнтській частині.

Загалом така архітектура вийшла гнучкою, логічно розбитою на частини й зручною для подальшої розробки. Кожен модуль можна тестувати, доповнювати або змінювати окремо, не чіпаючи інші частини. Це дозволяє підтримувати систему у робочому стані, адаптувати її до нових потреб і масштабувати, якщо зросте навантаження [12, 13].

3.3 Проєктування структури зберігання даних

Зберігання даних у системі реалізовано на основі документно-орієнтованої бази MongoDB, що дозволяє гнучко працювати з динамічною структурою документів і уникати жорсткого нормалізованого підходу, характерного для реляційних баз. Такий вибір обґрунтовано характером предметної області: дані мають різну складність, включають вкладені структури (наприклад, список фото тварини), часто модифікуються і мають бути легко масштабованими. Кожна модель, що описує окрему сутність системи, відображає логіку взаємодії користувача з платформою: створення анкет, відправка повідомлень, оцінка інших користувачів, взаємодія з тваринами тощо.

Базовою одиницею зберігання інформації про користувача є модель User, вбудована в Django [14]. Вона містить дані автентифікації: логін, пошту, пароль (у хешованому вигляді) та технічні поля. Для зберігання додаткової інформації, пов’язаної з функціональністю застосунку, створено пов’язану модель UserProfile. Вона зберігає контактний номер телефону користувача, а також статус преміум-доступу з датами початку й завершення, якщо така опція активна. Розділення інформації між двома моделями дозволяє залишити без змін внутрішню структуру Django, водночас доповнивши її прикладною логікою.

Дані про тварин організовано в моделі Pet. Кожен об’єкт тварини пов’язаний з користувачем, який її додав. Основні поля: ім’я, вид, порода, стать, колір шерсті, вік у місяцях, опис, вартість (якщо вона вказана), а також статус активності. Таким чином, система дозволяє створювати анкету навіть без точної інформації про всі параметри, що важливо для живого процесу оновлення даних. Кожна тварина може мати декілька зображень, які зберігаються окремо в моделі Photo, пов’язаній із Pet через зовнішній ключ. Це спрощує обробку фото та дозволяє, за потреби, масштабувати зберігання зображень окремо від основної анкети.

Соціальна взаємодія між користувачами реалізована через модель Review, яка дозволяє залишити оцінку іншому користувачу. Вона включає рейтинг (від 1 до 5), текстовий коментар, а також посилання на автора та адресата відгуку. За бажанням, коментар може бути порожнім, залишаючи лише числову оцінку. Це дозволяє користувачам оперативно оцінювати взаємодії без необхідності формулювати текстові рецензії.

У модулі matching реалізовано три окремі моделі для зберігання взаємодії між тваринами. Like зберігає односторонню симпатію (тобто власник однієї тварини позначив іншу як потенційного партнера). Dislike фіксує відсутність інтересу. Sympathy формується лише у випадку взаємного інтересу, тобто коли обидві сторони висловили бажання з’єднати своїх тварин. Такий підхід дозволяє легко реалізувати як базову логіку свайпів, так і візуалізацію взаємних відповідностей без дублювання даних. При цьому всі три моделі мають просту структуру з ідентифікаторами тварин та датою створення.

Для обміну повідомленнями між користувачами створено моделі Chat та Message. Кожен Chat пов’язує двох користувачів, незалежно від кількості тварин, і створюється автоматично в момент початку спілкування. Повідомлення (Message) містять посилання на чат, автора, текст, позначку часу, а також прапорець прочитано/непрочитано. Таким чином, зберігається історія переписки, яка доступна для обох учасників навіть після тривалого періоду неактивності.

Кожна модель містить часові поля (created_at, timestamp), що дає змогу реалізувати хронологічне сортування та відстеження активності. Усі зв’язки між сутностями реалізовані через ForeignKey з параметром on_delete=models.CASCADE, що гарантує очищення залежних об’єктів у разі видалення батьківського. Це дозволяє уникнути "висячих" записів і забезпечити логічну цілісність даних без складних транзакцій.

MongoDB, як база даних, добре справляється з цією структурою завдяки підтримці вкладених об’єктів та відсутності суворої схеми [15]. Наприклад, список фотографій не обов’язково має бути частиною самого об’єкта Pet, що дозволяє розширювати або змінювати модель без порушення цілісності основних даних. У майбутньому можливе запровадження індексації найбільш запитуваних полів (наприклад, вид, порода, вік), щоб оптимізувати фільтрацію при пошуку тварин.

Загалом, структура зберігання даних у системі розроблена таким чином, щоб бути водночас простою в реалізації, зручною для користування, гнучкою до змін і сумісною з реальними сценаріями використання. Вона дозволяє легко масштабувати обсяг даних, вводити нові поля або типи об’єктів і безболісно адаптувати систему під нові вимоги.

3.4 Алгоритми та методи обробки запитів

У процесі розробки серверної частини системи особливу увагу було приділено правильному структуруванню обробки HTTP та WebSocket-запитів. Кожен запит, що надходить від клієнтської частини, проходить через послідовність логічних перевірок, обробки та формування відповіді. Основна мета – забезпечити

стабільність, безпеку й передбачувану поведінку сервісу незалежно від навантаження або сценарію.

Одним із перших кроків у будь-якому запиті є перевірка автентичності користувача. Система побудована на сесійній автентифікації: після входу через /login/ користувач отримує сесію, що зберігається на стороні сервера. Завдяки цьому у всіх подальших запитах система має доступ до request.user, і може миттєво визначити, хто саме виконує дію. Додатково, кожен запит, що модифікує дані, перевіряється на наявність CSRF-токена. Такий підхід дозволяє уникнути типових атак типу Cross-Site Request Forgery без необхідності використання JWT або інших зовнішніх механізмів.

Під час виконання більшості запитів застосовується вбудована валідація даних. Наприклад, при створенні анкети тварини система перевіряє, чи введені обов’язкові поля (species, gender, age), чи вказана вартість, якщо це передбачено, і чи не дублюється запис. Усі перевірки реалізовано через DRF-серіалізатори. Помилки, знайдені під час перевірки, автоматично транслюються у форматовану JSON-відповідь із поясненням.

Окремий блок логіки присвячено фільтрації анкет тварин. Користувач має змогу підібрати тварину за рядом параметрів – видом, породою, віком, кольором шерсті. При цьому враховується стать: система автоматично підбирає анкети з протилежною статтю, оскільки пошук здійснюється для парування. Також реалізована перевірка на те, щоб уникати повторного показу вже оцінених анкет: у відповіді не з’являться тварини, яким користувач уже поставив «лайк» або «дизлайк». Алгоритм фільтрації виконується на основі агрегацій у запитах до MongoDB, із обмеженням вибірки до кількох результатів, щоб знизити навантаження.

Певна частина взаємодій не обмежується HTTP і реалізована через WebSocket-з’єднання. Це стосується чату між користувачами. Після встановлення з’єднання клієнт отримує унікальний канал (room), прив’язаний до конкретного чату. Усі повідомлення, які користувач надсилає, проходять через клас ChatConsumer, де перевіряється:

- чи автентифікований користувач;
- чи дійсно він є учасником чату;
- чи не намагається він підмінити отримувача.

Після перевірки повідомлення зберігається у базі даних, а потім транслюється іншому користувачу через той самий WebSocket-канал. Такий підхід дозволяє підтримувати чат у реальному часі з мінімальною затримкою.

У випадку, якщо користувач надсилає команду на видалення або редагування повідомлення, система перевіряє, чи є він автором цього повідомлення. Якщо перевірка проходить – зміни зберігаються, і новий стан повідомлення надсилається обом учасникам. Інакше – повертається повідомлення про помилку доступу.

Ще один напрям обробки запитів – залишення відгуків. Кожен користувач може оцінити іншого, із ким взаємодіяв. У відгуку зазначається оцінка (від 1 до 5) і, за бажанням, короткий коментар. Після надсилання, запис фіксується в моделі Review і доступний для перегляду з боку майбутніх партнерів. Завдяки цьому формується загальна картина надійності учасників спільноти. На рівні запиту система перевіряє, щоб не було повторного відгуку до того самого користувача, а також щоб сам користувач не залишав відгук собі.

Для кожної з описаних дій реалізовано обробку помилок: неправильний запит, спроба доступу без авторизації, відсутність необхідних полів або порушення прав – усе це призводить до чіткої, стандартизованої відповіді з поясненням причини.

Наприклад, алгоритм відбору анкет тварин, реалізований у функції find_matches, поєднує логіку фільтрації за видом, статтю, віком, породою та відкидає вже оцінені об'єкти. Нижче подано спрощений фрагмент цього алгоритму:

```
    def find_matches(request, pet_id):
        pet = get_object_or_404(Pet, id=pet_id)

        liked_pet_ids =
Like.objects.filter(petLikeFrom=pet).values_list('petLikeTo__id',
flat=True)
        disliked_pet_ids =
Dislike.objects.filter(petDislikeFrom=pet).values_list('petDislikeTo__id',
flat=True)
```

```
            matches = Pet.objects.filter(species=pet.species)

            opposite_gender = 'male' if pet.gender == 'female' else 'female'
            matches = matches.filter(gender=opposite_gender)

            min_age = max(0, pet.age - 6)
            max_age = pet.age + 6
            matches = matches.filter(age__gte=min_age, age__lte=max_age)

            if pet.breed:
                matches = matches.filter(breed=pet.breed)

            matches =
matches.exclude(id__in=liked_pet_ids).exclude(id__in=disliked_pet_ids)
```

У підсумку, реалізовані алгоритми охоплюють повний життєвий цикл взаємодії користувача із системою: від входу в обліковий запис до перегляду анкет, взаємодії з тваринами, обміну повідомленнями та оцінювання інших учасників. Обробка запитів побудована на реальних сценаріях і враховує всі основні винятки, що можуть виникнути в процесі використання системи. Це дозволяє гарантувати стабільність платформи та зручність для кінцевого користувача навіть у складних або нетипових випадках.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Реалізація WebSocket-комунікації

Одним з найважливіших аспектів функціонування розробленої системи є забезпечення можливості безперервного обміну повідомленнями між користувачами в режимі реального часу. Традиційні HTTP-запити не дозволяють досягти справжньої двосторонньої взаємодії: кожна нова дія вимагає відкриття нового з'єднання, що збільшує затримки і навантаження на сервер. У зв'язку з цим, для реалізації чату було обрано протокол WebSocket, який дозволяє клієнту та серверу підтримувати постійне з'єднання, не розриваючи його після кожного обміну повідомленнями.

На стороні сервера WebSocket-обробка реалізована за допомогою Django Channels – розширення до стандартного Django, що надає підтримку ASGI замість звичного WSGI [16]. Це дозволяє працювати з асинхронними з'єднаннями, необхідними для WebSocket.

У проєкті використовується спеціалізований клас ChatConsumer, який розташовується у модулі chats/consumers.py. Його основна роль – обробка усіх етапів життєвого циклу WebSocket-з'єднання: встановлення, отримання даних, обробка команд, відправка відповідей та закриття.

При підключенні до чату клієнт передає chat_id у URL з'єднання, що обробляється методом connect. Система зчитує ID чату, витягує дані з бази та перевіряє:

- чи є користувач авторизованим (request.user.is_authenticated);
- чи є він учасником цього чату (user_1 або user_2);
- чи існує вказаний чат взагалі.

У випадку успішної перевірки користувач підключається до відповідної групи каналу, через яку обмінюватиметься повідомленнями. У протилежному випадку з'єднання автоматично закривається:

```
async def connect(self):
    self.chat_id = self.scope['url_route']['kwargs']['chat_id']
    self.room_group_name = f"chat_{self.chat_id}"
```

```
        self.user = self.scope['user']

        if not self.user.is_authenticated:
            await self.close()
            return

        try:
            chat = await
database_sync_to_async(Chat.objects.select_related('user_1',
'user_2').get)(id=self.chat_id)
            self.chat = chat
            self.chat_user1_id = chat.user_1_id
            self.chat_user2_id = chat.user_2_id

            if self.user.id not in [self.chat_user1_id,
self.chat_user2_id]:
                await self.close()
                return

        except ObjectDoesNotExist:
            await self.close()
            return

        await self.channel_layer.group_add(self.room_group_name,
self.channel_name)
        await self.accept()
```

Завдяки такому підходу з’єднання зберігається активним тільки для дозволених учасників. Для зберігання повідомлень, а також для гарантування надійності, усі надіслані повідомлення фіксуються у базі даних до їх надсилання іншим користувачам. Це дозволяє підтримувати історію чату і забезпечувати надійність у випадку втрати з’єднання на стороні клієнта. Метод receive виконує диспетчеризацію вхідних повідомлень, передаючи їх на обробку до спеціалізованих методів (send_message, edit_message, delete_message), кожен із яких відповідає за збереження, редагування або видалення:

```
    async def receive(self, text_data):
        try:
            data = json.loads(text_data)
            action = data.get('action')
            if action == 'send':
                await self.send_message(data)
            elif action == 'edit':
                await self.edit_message(data)
            elif action == 'delete':
                await self.delete_message(data)
            else:
                await self.send_error("Unknown action.")
        except json.JSONDecodeError:
```

```
            await self.send_error("Invalid JSON.")
        except Exception as e:
            await self.send_error(f"Unexpected error: {str(e)}")
```

Цей фрагмент ілюструє сучасну архітектуру, у якій receive виступає диспетчером дій. Замість того, щоб безпосередньо створювати повідомлення, метод делегує обробку спеціалізованим функціям: send_message, edit_message, delete_message. Такий підхід значно покращує масштабованість і дозволяє в майбутньому легко додати нові дії (наприклад, реакції на повідомлення).

Метод chat_message, який відповідає за трансляцію повідомлень усередині групи, має такий вигляд:

```
async def chat_message(self, event):
    message = event['message']
    await self.send(text_data=message)
```

У цьому випадку message – це вже готовий серіалізований JSON-рядок, який формується на попередньому етапі у методі group_send. Завдяки цьому немає потреби у повторному перетворенні даних у JSON, а клієнт отримує готову структуру для відображення у своєму інтерфейсі. Такий підхід зменшує обчислювальне навантаження та пришвидшує відгук системи.

Особливу увагу було приділено обробці помилок і винятків. Якщо користувач намагається під’єднатися до чату, якого не існує, або до якого він не має доступу – система автоматично закриває з’єднання без надсилання повідомлення, що мінімізує ризики витоку інформації або перехоплення. Аналогічно, якщо при надсиланні повідомлення передано некоректні дані, або порожній рядок, повідомлення не буде оброблено.

На рівні інфраструктури використовується стандартний channel layer, який у мінімальній конфігурації реалізований через вбудовану пам’ять (InMemoryChannelLayer), але за потреби легко масштабується до Redis або інших брокерів повідомлень. Це дозволяє безперешкодно розгортати систему на кількох інстанціях серверів без втрати функціональності чату.

Загальна структура WebSocket-модуля дозволяє його ізольовано тестувати та масштабувати. Наприклад, у майбутньому можна додати:

- сповіщення про статус онлайн/офлайн;
- індикатори «користувач набирає повідомлення»;
- сповіщення про прочитане;
- обробку файлів або медіа.

Інтеграція ChatConsumer з моделями Chat та Message реалізована через стандартний Django ORM, а отже – уся логіка зберігання, зв’язків та авторизації є перевіреною, повторно використовуваною та масштабованою.

Таким чином, реалізація WebSocket-зв’язку у межах системи не тільки дозволяє досягти вимог до швидкодії та безперервності взаємодії, але й демонструє грамотне застосування асинхронної архітектури в рамках класичного Django-застосунку. Всі компоненти працюють злагоджено, легко модифікуються і дозволяють розширення функціоналу без порушення основної логіки.

4.2 Опис реалізації основних компонентів

Реалізація серверної частини системи виконувалась у рамках фреймворку Django, що забезпечує повноцінну логіку обробки запитів, керування базою даних, автентифікацію користувачів і модульну побудову структури. Вся логіка системи умовно розподілена між окремими функціональними модулями, кожен з яких відповідає за обробку своєї групи сутностей. У модулі users реалізовано обробку автентифікації, реєстрації, редагування профілю, перегляду інформації про користувачів та систему відгуків. У модулі pets зосереджена логіка створення, редагування, видалення анкет тварин та їх візуального представлення у форматі JSON. Модуль matching відповідає за пошук відповідних анкет на основі алгоритму підбору, з урахуванням симпатій, дизлайків і унікальності кожної рекомендації. Окремий модуль chats реалізує логіку обміну повідомленнями – як через WebSocket, так і REST-запити для створення чатів, перегляду історії тощо.

Приклади реалізації надають уявлення про загальний підхід до побудови системи. Наприклад, функція перегляду всіх чатів для поточного користувача

побудована з урахуванням перевірки автентичності та вибірки з використанням Django ORM:

```
    def all_chats(request):
        if request.method == "GET":
            if not request.user.is_authenticated:
                return JsonResponse({'error': 'User not authenticated'},
status=401)

            chats = Chat.objects.filter(Q(user_1=request.user) |
Q(user_2=request.user))

            chat_list = [
                {
                    "chat_id": chat.id,
                    "user_1": chat.user_1.username,
                    "user_2": chat.user_2.username,
                    "created_at": chat.created_at.strftime('%Y-%m-%d
%H:%M:%S')
                } for chat in chats
            ]

            return JsonResponse({"chats": chat_list})
        else:
            return JsonResponse({'error': 'Invalid request method'},
status=405)
```

У цьому фрагменті добре видно типову структуру більшості представлень: перевірка методу запиту, автентичності користувача, витяг даних з бази, формування списку словників для серіалізації у JSON-формат і відправлення відповіді. Такий патерн повторюється в усьому проєкті й дозволяє легко масштабувати функціональність.

Схожа логіка застосовується і для створення чатів між користувачами. При цьому перевіряється, чи існує вже створений чат між цими двома користувачами, щоб уникнути дублювання:

```
    def create_chat(request, user_id):
        if request.method != "POST":
            return JsonResponse({'error': 'Invalid request method'},
status=405)

        if not request.user.is_authenticated:
            return JsonResponse({'error': 'User not authenticated'},
status=401)

        try:
```

```
            other_user = User.objects.get(id=user_id)
        except User.DoesNotExist:
            return JsonResponse({'error': 'User not found'}, status=404)

        user1, user2 = sorted([request.user, other_user], key=lambda u:
u.id)

        chat = Chat.objects.filter(user_1=user1, user_2=user2).first()
        if chat:
            return JsonResponse({'chat_id': chat.id, 'message': 'Chat
already exists'})

        chat = Chat.objects.create(user_1=user1, user_2=user2)
        return JsonResponse({'chat_id': chat.id, 'message': 'Chat
created'})
```

У фрагменті використовується сортування учасників за ID, що дозволяє уникнути створення симетричних записів типу (user_1=A, user_2=B) і (user_1=B, user_2=A), які мають бути тотожними в логіці діалогу.

Важливою частиною реалізації є також модуль matching, у якому функція find_matches реалізує основний алгоритм добору. Вона витягує параметри поточної тварини (вид, стать, вік, порода), визначає протилежну стать, формує вікно вікової допустимості й виключає вже оцінені тварини з результату. Це дозволяє значно підвищити якість рекомендацій:

```
    def find_matches(request, pet_id):
        pet = get_object_or_404(Pet, id=pet_id)

        liked_pet_ids =
Like.objects.filter(petLikeFrom=pet).values_list('petLikeTo__id',
flat=True)
        disliked_pet_ids =
Dislike.objects.filter(petDislikeFrom=pet).values_list('petDislikeTo__id',
flat=True)

        matches = Pet.objects.filter(species=pet.species)

        opposite_gender = 'male' if pet.gender == 'female' else 'female'
        matches = matches.filter(gender=opposite_gender)

        min_age = max(0, pet.age - 6)
        max_age = pet.age + 6
        matches = matches.filter(age__gte=min_age, age__lte=max_age)

        if pet.breed:
            matches = matches.filter(breed=pet.breed)

        matches = matches.exclude(id__in=liked_pet_ids)
        matches = matches.exclude(id__in=disliked_pet_ids)
```

```
            results = [{
                "id": match.id,
                "coat_color": match.coat_color,
                "species": match.species,
                "gender": match.gender,
                "breed": match.breed,
                "price": match.price,
                "age": match.age,
                "photos": [photo.image.url for photo in
match.photo_set.all()]
            } for match in matches]

            return JsonResponse({"matches": results})
```

Цей код демонструє, як із використанням фільтрації, об'єднань і агрегації можна побудувати ефективний параметричний пошук. Така реалізація дозволяє інтегрувати персоналізовані рекомендації тварин для кожного користувача з урахуванням взаємної зацікавленості.

Кожен з компонентів системи реалізовано на основі схожого принципу: перевірка автентичності, перевірка правильності вхідних параметрів, обробка моделі через ORM, складання відповіді у форматі JSON. Завдяки цьому підтримується єдність стилю і мінімізується кількість помилок при розширенні логіки. Реалізація не використовує сторонніх REST-фреймворків, таких як Django REST Framework, що підкреслює повний ручний контроль над усіма процесами.

У цілому, цей підхід демонструє переваги чітко структурованої модульної архітектури, з можливістю швидкої інтеграції нового функціоналу й збереження читабельності та контрольованості коду. Кожна функція окремо тестується й може працювати незалежно від інших, що значно спрощує обслуговування та масштабування системи.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було реалізовано серверну частину програмної системи, яка дозволяє здійснювати параметричний підбір партнерів для домашніх улюбленців з метою їхнього подальшого розведення. Цей підпроєкт охоплює усі ключові аспекти бекенд-розробки, включаючи проєктування бази даних, реалізацію API для взаємодії з клієнтською частиною, а також побудову асинхронної системи обміну повідомленнями у реальному часі. В межах командної співпраці над проєктом саме бекенд-частина стала основою для функціонування всієї системи, оскільки через неї відбувається більшість логічних перевірок, взаємодія з базою даних та забезпечення стабільної роботи всіх модулів.

Першим етапом роботи було спроєктовано структуру зберігання даних, яка відображає логіку предметної області. Було визначено основні сутності: користувачі, їх профілі, анкети тварин, фотографії, оцінки, симпатії, чати та повідомлення. Для реалізації зв’язків між ними застосовано чітку схему референцій, яка дозволяє виконувати запити з урахуванням автентифікації та перевірки прав доступу. У якості бази даних використовувалась MongoDB, що надала гнучкість при зберіганні вкладених структур, таких як масиви фотографій до кожної анкети.

Другим напрямком стала реалізація механізмів автентифікації та безпеки. Усі запити користувача проходять через послідовність перевірок: наявність сесії, автентичність користувача, валідність CSRF-токена. Це дозволяє гарантувати, що лише авторизовані користувачі мають доступ до персоніфікованої інформації, а будь-які спроби несанкціонованої модифікації відхиляються системою. Було реалізовано гнучку валідацію даних, яка автоматично повертає уніфіковані повідомлення про помилки у форматі JSON, що спрощує обробку помилок на стороні клієнта.

Особливу увагу було приділено алгоритмам обробки даних. Один із найважливіших алгоритмів – пошук потенційних партнерів для тварин. Система підбирає відповідні анкети за критеріями: вид, стать, вік, порода, при цьому

виключаючи анкети, які вже були оцінені користувачем. Це дозволяє уникнути повторів і значно покращує користувацький досвід. У разі взаємної симпатії автоматично створюється чат, що дозволяє власникам спілкуватися безпосередньо. Реалізація подібної логіки вимагала узгодженої роботи запитів, транзакцій, а також детального тестування.

Окремий функціональний блок – обмін повідомленнями між користувачами – було реалізовано з використанням WebSocket-протоколу. На основі Django Channels було створено асинхронний обробник ChatConsumer, який відповідає за весь життєвий цикл з'єднання: встановлення, отримання та відправка повідомлень, обробка помилок, перевірка автентичності та доступу. Завдяки цьому було досягнуто високої швидкодії та низької затримки при взаємодії користувачів у реальному часі. Повідомлення зберігаються у базі даних одразу після надсилання, що дозволяє зберігати історію чату навіть у разі втрати з'єднання.

Система в цілому побудована таким чином, щоб кожен запит, який надходить до сервера, проходив повний цикл логічної обробки: ідентифікація користувача, перевірка прав доступу, валідація введених даних, обробка помилок та формування уніфікованої відповіді. Окремі маршрути відповідають за створення, редагування та видалення анкет, фото, повідомлень, а також за зчитування інформації про інших користувачів, перегляд відгуків тощо. Всі функції реалізовані без використання сторонніх бібліотек типу DRF, що дозволило зберегти повний контроль над логікою обробки.

Загалом, результат роботи демонструє успішну реалізацію складної серверної архітектури з великою кількістю взаємопов'язаних компонентів. Розроблений бекенд не тільки повністю виконує покладені на нього функції, але й є достатньо гнучким для подальшого розширення. Усі основні завдання, поставлені на початку, були виконані в повному обсязі. Робота не обмежувалася формальним написанням коду: доводилось ухвалювати архітектурні рішення, усувати конфлікти при обробці одночасних запитів, забезпечувати масштабованість і відповідність сучасним стандартам веброзробки.

Таким чином, виконана робота стала не лише практичним застосуванням знань, отриманих під час навчання, але й справжнім етапом професійного становлення розробника. Реалізована серверна частина – це стабільна, безпечна, масштабована основа для функціонування інформаційної системи, яка може бути з легкістю доповнена модулями аналітики, рекомендацій, розширеного пошуку або геолокаційного фільтру.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. State of WebSockets 2024. [Електронний ресурс]: https://ably.com/state-of-websockets (дата звернення: 01.06.2025).

2. Python Logging Facility Documentation. [Електронний ресурс]: https://docs.python.org/3/library/logging.html (дата звернення: 01.06.2025).

3. Django REST Framework. [Електронний ресурс]: https://www.django-rest-framework.org/ (дата звернення: 01.06.2025).

4. PyMongo Documentation. [Електронний ресурс]: https://pymongo.readthedocs.io/ (дата звернення: 01.06.2025).

5. GitHub Docs: About repositories. [Електронний ресурс]: https://docs.github.com/en/repositories (дата звернення: 01.06.2025).

6. “Data Modeling for MongoDB”. Sadalage, P. & Fowler, M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2012.

7. CSRF Protection in Django. [Електронний ресурс]: https://docs.djangoproject.com/en/stable/ref/csrf/ (дата звернення: 01.06.2025).

8. “OWASP Cheat Sheet Series: Cross-Site Request Forgery (CSRF)”. OWASP Foundation. [Електронний ресурс]: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (дата звернення: 01.06.2025).

9. Django Security Documentation. [Електронний ресурс]: https://docs.djangoproject.com/en/stable/topics/security/ (дата звернення: 01.06.2025).

10. RESTful API Design: Best Practices. [Електронний ресурс]: https://restfulapi.net/ (дата звернення: 01.06.2025).

11. Офіційна документація Django Channels. [Електронний ресурс]: https://channels.readthedocs.io/ (дата звернення: 01.06.2025).

12. Омельченко В., Романовський І., Лендел Я., Терещенко Г. Використання Параметричного Пошуку у Вебзастосунках для Підбору Тварин // Матеріали конференції MIT@AISs-2025. Харків: ХНУРЕ, 2025.

13. Django ORM Documentation. [Електронний ресурс]: https://docs.djangoproject.com/en/stable/topics/db/models/ (дата звернення: 01.06.2025).

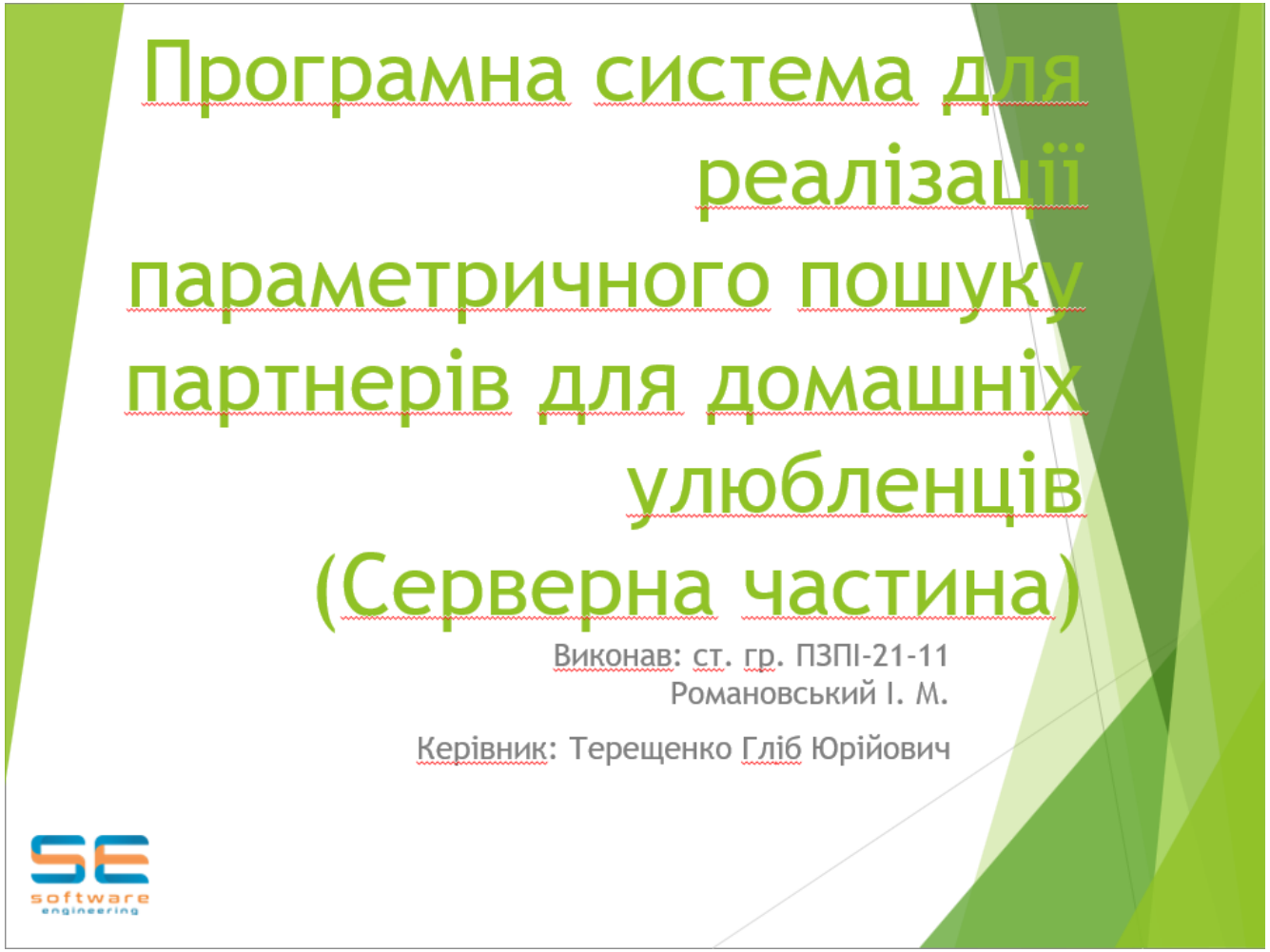
14. MongoDB Manual: Data Modeling Introduction. [Електронний ресурс]: https://www.mongodb.com/docs/manual/core/data-modeling-introduction/ (дата звернення: 01.06.2025).

15. API Security Best Practices (RapidAPI). [Електронний ресурс]: https://rapidapi.com/blog/api-glossary/what-is-api-security-api-security-best-practices/ (дата звернення: 01.06.2025).

16. GitHub-репозиторій з кодом проєкту. [Електронний ресурс]: https://github.com/NureRomanovskyiIllia/PI_PZPI-21-11_ROMANOVSKY_I_M (дата звернення: 01.06.2025).

ДОДАТОК А

Слайди презентації



Мета проєкту

- Реалізація REST API для керування користувачами, тваринами та симпатіями
- Забезпечення швидкої, зручної та безпечної взаємодії з базою даних
- Підтримка параметричного пошуку партнерів за породою, статтю, віком тощо
- Реалізація функціоналу обміну повідомленнями на основі WebSocket
- Створення масштабованої та модульної структури серверного застосунку на базі **Django**

Актуальність

- Соцмережі не дозволяють ефективно підбирати партнерів для тварин
- Відсутність фільтрації, безпечного середовища та перевірених профілів
- Потреба у спеціалізованій адаптивній SPA-системі з фокусом на UX

Аналіз аналогів

- На ринку відсутні спеціалізовані серверні рішення для автоматизованого підбору партнерів серед тварин з можливістю повноцінної взаємодії між користувачами.
- **Pets4You** - онлайн-каталог для продажу породистих тварин:
- Велика база даних тварин
  - Немає серверної логіки пошуку сумісних пар
  - Відсутній API для взаємодії між власниками
- **Petfinder** - сервіс для адопції тварин з притулків:
- Категоризація за місцем, видом
  - Відсутність серверної підтримки алгоритму сумісності
  - Немає бекенд-реалізації двосторонньої комунікації або оцінювання

Висновок: Існуючі сервіси не мають backend-архітектури, яка б підтримувала:

- авторизовану взаємодію між власниками тварин,
- алгоритмічний підбір за параметрами,
- обмін повідомленнями,
  що реалізовано в PetBreedingApp на основі Django.

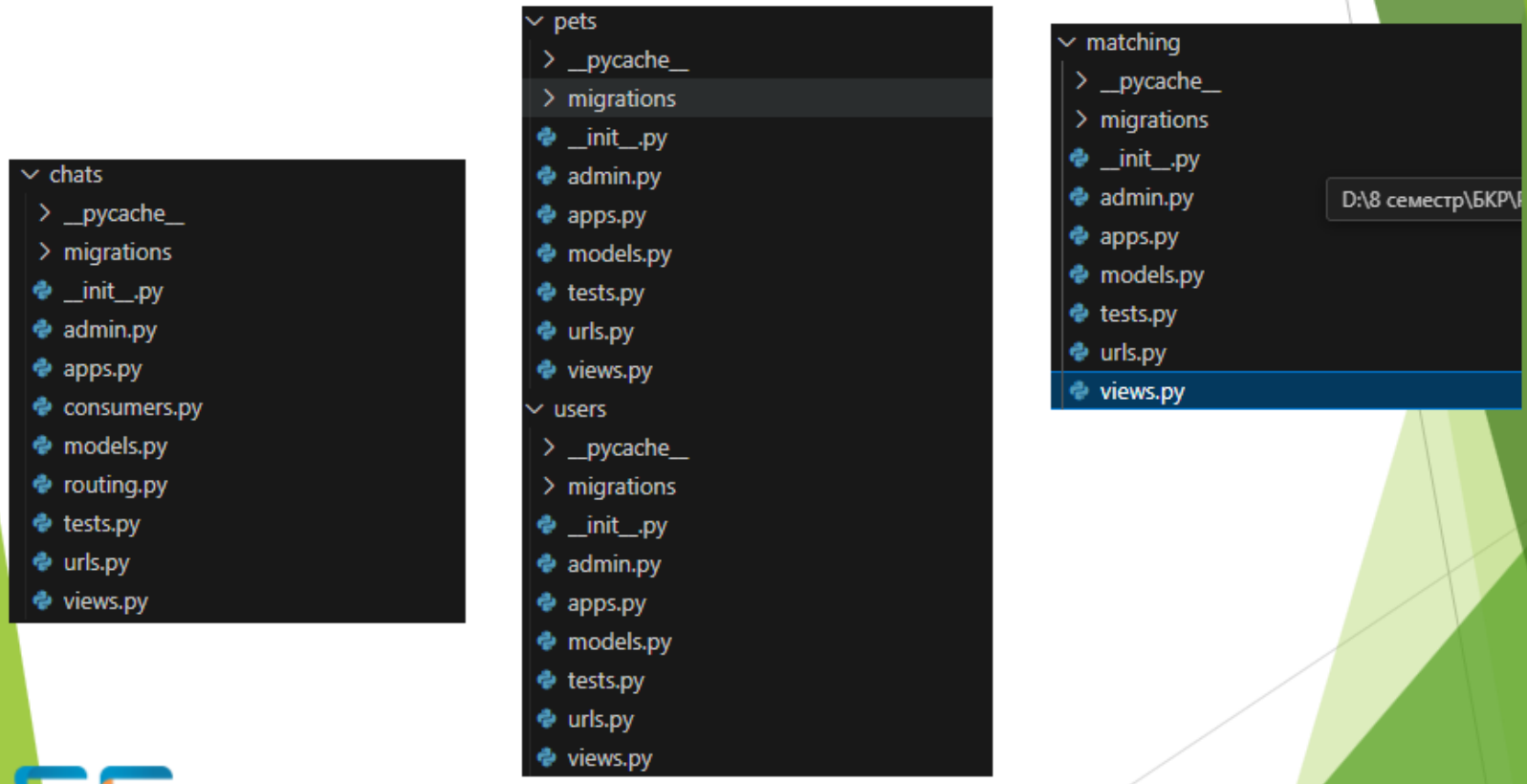
Постановка задачі (Backend)

- Реалізація API для **реєстрації**, **авторизації** та **аутентифікації користувачів**
- Створення моделей та ендпоінтів для **анкетування тварин**
- Підтримка **параметричного пошуку** за віком, статтю, породою та локацією
- Обробка запитів на **перегляд профілів** та улюблених анкет
- **WebSocket-підключення** для чату між користувачами
- Система **рейтингу та фіксації взаємних симпатій/антипатій**

Вибір технологій (Backend)

- **Django** - основа серверної логіки, реалізація REST API, авторизації, CRUD-операцій
- **Django** - основа серверної логіки, реалізація REST API, авторизації, CRUD-операцій
- React Router — маршрутизація
- **Django Channels** - підтримка WebSocket для чату в реальному часі
- **JWT/CSRF автентифікація** - захист API та сесій користувача
- **media storage** - збереження фото тварин

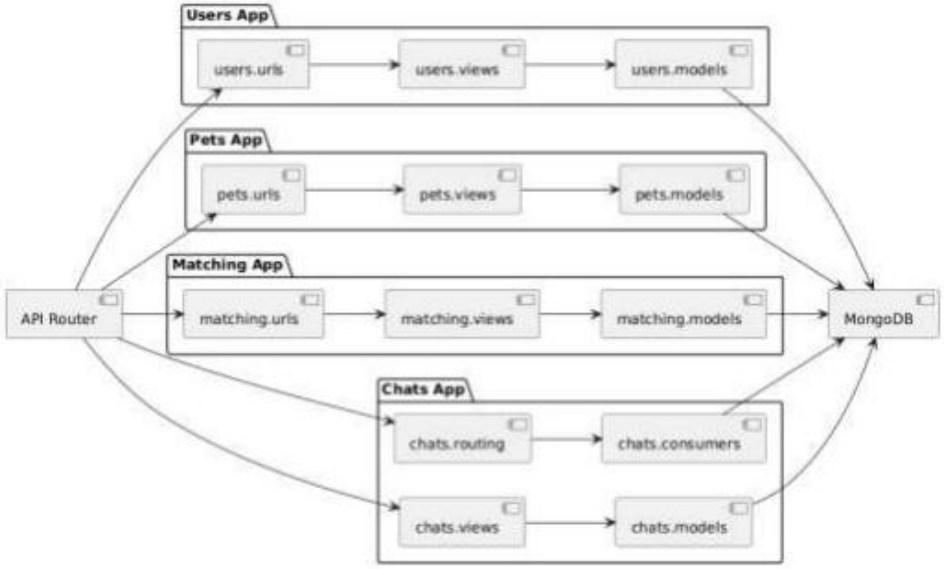
Архітектура створенного програмного забезпечення



Архітектура створенного програмного забезпечення

ER-діаграма структури бази даних

Архітектура створенного програмного забезпечення



Діаграма компонентів серверної частини

Тези

УДК 004.75

АРХІТЕКТУРА ОПЛАТИ ЗА ПРЕМІУМ-ПІДПИСКУ НА СЕРВІС З ВИКОРИСТАННЯМ КРИПТОГАМАНЦЯ ЧЕРЕЗ SOLIDITY І METAMASK

Олейченко В.В., Романовський І.М., Лендел Я.Р., Семенов О.О.
e-mail: vladyslav.oleichenko1@nure.ua, ilia.romanovskyi@nure.ua, yaroslav.lendel@nure.ua, oleksandr.semenov@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study explores the implementation of a blockchain-based payment system for premium subscriptions using Solidity smart contracts and Metamask integration. The proposed solution ensures security, transparency, and automation of transactions by eliminating intermediaries. The system consists of a smart contract, a Metamask-integrated web interface, and a backend for data synchronization. Testing in the Ethereum test network confirmed its reliability and efficiency. While challenges like user accessibility and crypto price volatility exist, this approach enhances trust and optimizes digital payment processes.

У сучасному цифровому світі, де технології блокчейн та криптовалюти стрімко розвиваються, інтеграція децентралізованих фінансових інструментів у вебсервіси стає все більш актуальною. Це відкриває нові можливості для бізнесу та користувачів, забезпечуючи прозорість, безпеку та автономність фінансових операцій [1]. Однією з перспективних областей застосування є впровадження оплати премум-підписок за допомогою смарт-контрактів та криптогаманців.

Метою даної роботи є розробка інноваційного механізму оплати премум-підписки для вебсервісу, який базується на використанні смарт-контрактів, написаних на мові програмування Solidity, та інтеграції з популярним криптогаманцем Metamask. Такий підхід дозволяє автоматизувати процеси оплати, забезпечити високий рівень безпеки транзакцій та надати користувачам зручний інтерфейс для взаємодії з блокчейн-технологіями [2].

Основні завдання, які ставляться в рамках цієї роботи, включають:

1. Розробка смарт-контракту для обробки платежів: створення безпечного та ефективного смарт-контракту, який буде відповідати за прийом платежів від користувачів, збереження інформації про статус їх підписок та управління доступом до преміум-функцій сервісу.

2. Реалізація механізму підтвердження підписки: забезпечення автоматичного оновлення статусу підписки користувача після здійснення оплати, а також розробка системи сповіщень про успішність транзакцій та терміни дії підписки.

3. Розробка клієнтського інтерфейсу для взаємодії з блокчейном: інтеграція Metamask у вебінтерфейс сервісу, що дозволить користувачам легко та безпечно здійснювати платежі, переглядати статус своїх підписок та керувати ними безпосередньо через браузер.

Застосування мови програмування Solidity для написання смарт-контрактів на платформі Ethereum надає можливість створювати децентралізовані додатки (dApps) з високим рівнем безпеки та прозорості [3]. Смарт-контракти дозволяють автоматизувати виконання умов угод, виключаючи потребу в посередниках та мінімізуючи ризики шахрайства. Інтеграція ж з Metamask забезпечує користувачам зручний доступ до їхніх криптовалютних активів та можливість взаємодії з блокчейном без необхідності встановлення додаткового програмного забезпечення.

Архітектура запропонованої системи складається з трьох основних компонентів:

1. Смарт-контракт, написаний на Solidity, який відповідає за обробку транзакцій, зберігання даних про підписки та управління доступом до преміум-функцій. Приклад смарт-контракту наведено на рисунку 1.

Рисунок 1 – Приклад Смарт-контракту

2. Веб-інтерфейс з інтеграцією Metamask, що надає користувачам можливість авторизації, здійснення платежів та перегляду статусу своїх підписок у реальному часі.

3. Бекенд-сервер, який забезпечує синхронізацію даних між блокчейном та вебінтерфейсом, обробляє запити користувачів та надає

Для оцінки ефективності запропонованого рішення було проведено тестування в тестовій мережі Ethereum (Rinkeby). Результати показали, що система здатна обробляти транзакції швидко та безпечно, забезпечуючи користувачам надійний доступ до преміум-функцій сервісу. Порівняно з традиційними методами оплати, використання блокчейн-технологій дозволяє знизити витрати на транзакції та підвищити довіру користувачів до сервісу.

Однак, впровадження таких технологій не позбавлене викликів. Однією з основних проблем є забезпечення зручності використання для користувачів, які не знайомі з криптовалютами та блокчейном. Для цього необхідно розробити інтуїтивно зрозумілий інтерфейс та надати детальні інструкції щодо використання Metamask та здійснення платежів. Крім того, варто враховувати можливі коливання вартості криптовалют, що можуть впливати на вартість підписок, та забезпечити механізми для мінімізації таких ризиків.

Запропонований підхід до оплати преміум-підписок за допомогою смарт-контрактів та інтеграції з Metamask відкриває нові горизонти для розвитку вебсервісів, надаючи користувачам безпечні, прозорі та автоматизовані способи здійснення платежів. Це сприяє підвищенню довіри до сервісу, розширенню його функціональних можливостей та залученню нових користувачів, які цінують сучасні технологічні рішення.

Список використаних джерел:

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf
2. Wood, G. (2014). Ethereum: A Secure Decentralised Generalised Transaction Ledger. https://ethereum.github.io/yellowpaper/paper.pdf
3. Dannen, C. (2017). Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. Apress. https://link.springer.com/book/10.1007/978-1-4842-3086-4

Підсумки

- **Реалістичність впровадження**

  Розроблена серверна частина на основі **Django** має модульну архітектуру, чітке розділення логіки по застосунках (users, pets, matching, chats). Система протестована в середовищі розробки, забезпечує підтримку: авторизації користувачів, створення та пошуку анкет та взаємодії через чат (WebSocket).

  Усе це дозволяє **інтегрувати backend до продакшен-середовища** без значних доопрацювань.

- **Переваги архітектури**

  Серверна частина побудована згідно з принципами RESTful API та асинхронної обробки повідомлень через Django Channels. Це забезпечує:

  масштабованість та гнучкість структури

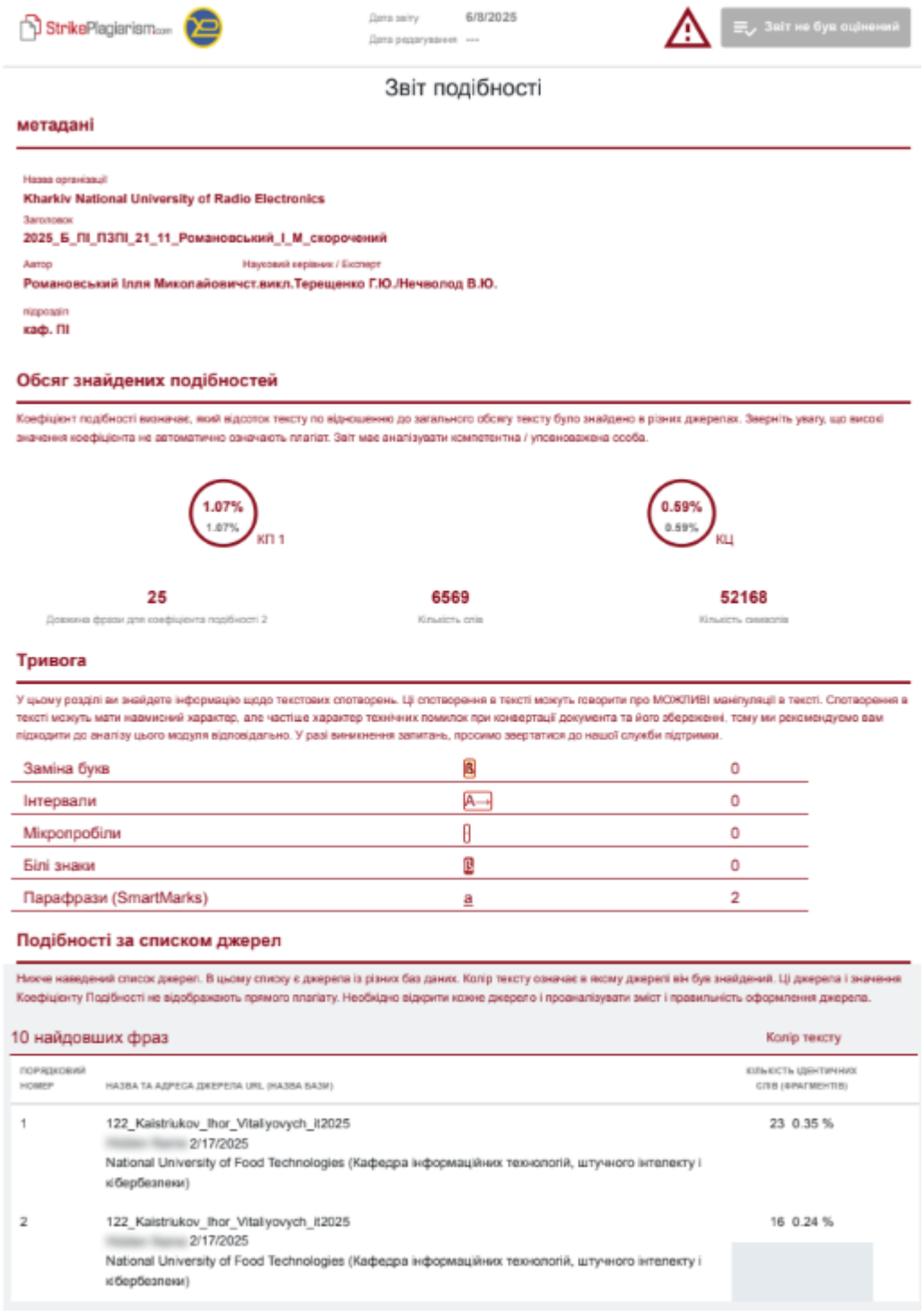
  чітке розділення відповідальностей

  (авторизація, дані тварин, симпатії, чат)

  легкість супроводу і доповнення функцій

ДОДАТОК В

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/8/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ_21_11_Романовський_І_М_скорочений
Автор
Романовський Ілля Миколайовичст.викл.Терещенко Г.Ю./Нечволод В.Ю.
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.07% 1.07% КП 1

0.59% 0.59% КЦ

25	6569	52168
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції з текстом. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		2

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	122_Kaistrukov_Ihor_Vitaliyovych_it2025 [redacted] 2/17/2025 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	23 0.35 %
2	122_Kaistrukov_Ihor_Vitaliyovych_it2025 [redacted] 2/17/2025 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	16 0.24 %

3	https://www.cnblogs.com/up-k/p/18464473	12 0.18 %
4	122_Kaistriukov_Ihor_Vitaliyovych_it2025 [redacted] 2/17/2025 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	9 0.14 %
5	122_Kaistriukov_Ihor_Vitaliyovych_it2025 [redacted] 2/17/2025 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	5 0.08 %
6	https://pantherax.com/how-to-build-a-chat-application-with-django-channels-and-websocket/	5 0.08 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.81 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	122_Kaistriukov_Ihor_Vitaliyovych_it2025 [redacted] 2/17/2025 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	53 (4) 0.81 %

з Інтернету (0.26 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://www.cnblogs.com/up-k/p/18464473	12 (1) 0.18 %
2	https://pantherax.com/how-to-build-a-chat-application-with-django-channels-and-websocket/	5 (1) 0.08 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)