Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного засобу для управління орендою житла

Виконав: студент IV курсу
групи 5ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Терешко Д. С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Бабюк Н. П.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Колесник І. С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Терешку Дмитру Сергійовичу

1. Тема роботи – розробка програмного засобу для управління орендою житла.

Керівник роботи: Бабюк Наталія Петрівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: тип програми – вебзастосунок; вхідні дані: параметри об'єктів житлової нерухомості, дані учасників договірних відносин, умови договорів, документи, відомості про платежі і використані комунальні послуги; середовище розробки – IntelliJ IDEA; мови програмування – Java, TypeScript; фреймворки – Spring (Boot, Data, Web MVC, Security), Angular; системи керування базами даних – PostgreSQL, MongoDB.

4. Зміст текстової частини: вступ; аналіз стану програмних засобів для управління орендою житла; порівняльний аналіз аналогів; постановка задач дослідження; аналіз вхідних даних програмного засобу; розробка моделі та алгоритмів роботи програмного засобу; обґрунтування вибору засобів для реалізації програмного забезпечення; розробка модулів програмного засобу; тестування програмного засобу; розробка інструкції користувача; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: титульний слайд; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність одержаних результатів; блок-схеми алгоритмів роботи застосунку; UML-діаграми компонентів програмного засобу; демонстрація графічного інтерфейсу програмного засобу; апробація результатів бакалаврської кваліфікаційної роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Бабюк Н. П., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку програмних засобів керування орендою житла та постановка задач дослідження	25.03.25 – 03.04.25	вик
2	Розробка методів, алгоритмів та моделей програмного засобу	04.04.25 – 15.04.25	вик
3	Розробка модулів програмного засобу	16.04.25 – 01.05.25	вик
4	Тестування програмного засобу	02.05.25 – 14.05.25	вик
5	Розробка інструкції користувача	15.05.25 – 21.05.25	вик
6	Оформлення матеріалів до захисту БКР	22.05.25 – 30.05.25	вик

Студент _____ **Д. С. Терешко**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Н. П. Бабюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.04

Терешко Д. С. Розробка програмного засобу для управління орендою житла: бакалаврська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 69 с.

На укр. мові. Бібліогр. : 41 назв.; рис. : 27; табл. : 1.

У бакалаврській кваліфікаційній роботі було розроблено програмний засіб для управління орендою житла. У ході роботи проведено аналіз предметної галузі та існуючих програмних засобів для управління орендою житла. Створено схеми моделі системи та алгоритми роботи деяких процесів.

Розроблено програмні модулі, що надають функції для документообігу, обліку платежів і комунальних послуг, керування об’єктами оренди, комунікації та керування договорами, для реалізації яких було використано мову програмування Java і набір фреймворків Spring (Boot, Data, Web MVC, Security). Для зберігання даних було застосовано PostgreSQL та MongoDB, для файлових даних – сховище MinIO.

Створено вебінтерфейс використовуючи мову програмування TypeScript, фреймворк Angular і бібліотеку елементів інтерфейсу Angular Material.

Отримані у бакалаврській кваліфікаційній роботі результати можна використати для покращення зручності процесу управління орендою житла.

Ключові слова: управління орендою житла, програмний засіб, вебзастосунок, Java, Spring Framework, PostgreSQL, MongoDB, Angular.

ABSTRACT

UDC 004.04

Tereshko D. S. Development of a software solution for residential rental management: bachelor's thesis on the specialty 121 Software engineering, educational program – Software engineering. Vinnytsia: VNTU, 2025. 69 p.

In Ukrainian language. Bibliography: 41 titles; fig.: 27; tabl.: 1.

In the bachelor's thesis, a software solution for residential rental management was developed. The work included analysis of the subject domain and existing software solutions for rental management. System model schemes and algorithms for certain processes were created.

Software modules were developed that provide functions for document management, payment and utility tracking, rental property management, communication, and contract management. For their implementation, Java programming language and Spring frameworks (Boot, Data, Web MVC, Security) were used. PostgreSQL and MongoDB were applied for data storage, while MinIO storage was used for file data.

A web interface was created using TypeScript programming language, Angular framework, and Angular Material UI component library.

The results obtained in the bachelor's thesis can be used to improve the convenience of the residential rental management process.

Keywords: residential rental management, software solution, web application, Java, Spring Framework, PostgreSQL, MongoDB, Angular.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Ринок житлової нерухомості в Україні становить один із ключових секторів національної економіки, який трансформувався із початком повномасштабної війни. Особливої актуальності в умовах воєнної економіки набуває сегмент орендних відносин, який стає критично важливим механізмом забезпечення житлових потреб громадян у період невизначеності [1].

Сучасний ринок оренди житла в Україні характеризується недостатньою прозорістю відносин між орендодавцями та орендарями, переважанням неформальних домовленостей над структурованими договірними відносинами, недосконалістю механізмів розв’язання конфліктних ситуацій та інструментів управління орендними платежами й супутніми витратами [1]. Ці фактори призводять до поширення тіньових схем та виникнення конфліктних ситуацій між учасниками орендних відносин.

Аналіз існуючих практик управління орендою житла виявляє три ключові задачі: паперовий документообіг, відсутність комплексного підходу до обліку платежів та обмежені можливості для оперативної комунікації між учасниками процесу [1].

Для розв’язання зазначених задач пропонується комплексний підхід до покращення процесу управління орендою житла з використанням інформаційно-комунікаційних технологій. Цей підхід включає цифрову трансформацію документообігу шляхом впровадження системи зберігання та версіонування договорів оренди чи інших пов’язаних документів, автоматизацію обліку фінансових операцій, покращення комунікації між учасниками процесу.

Ключовою перевагою пропонованої системи є її потенціал для виведення орендних відносин із тіньового сектору економіки шляхом спрощення юридичного оформлення відносин та створення прозорої системи фінансових взаємовідносин. Зважаючи на локальний характер ринку нерухомості, його чутливість до демографічних та економічних змін, а також зростаючі потреби в

зручному управлінні житловим фондом в умовах високої мобільності населення, впровадження спеціалізованих програмних систем набуває важливого значення. Тому, актуальною є розробка власного програмного засобу для управління орендою житла.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення зручності процесу управління орендою житла шляхом розробки програмного засобу, що матиме набір функцій, необхідний для забезпечення взаємодії учасників і прозорості процесу оренди.

Відповідно до поставленої мети потрібно вирішити такі завдання:

- провести аналіз предметної галузі та існуючих програмних засобів для управління орендою житла;

- розробити модель та алгоритми роботи програмного засобу;

- розробити модулі програмного засобу для управління орендою житла, що надають функції для документообігу, обліку платежів і комунальних послуг, керування об'єктами оренди, комунікації та керування договорами;

- розробити вебінтерфейс програмного засобу;

- провести тестування програмного засобу.

Об'єктом дослідження є процес розробки програмного засобу управління орендою житла.

Предметом дослідження є методи та засоби розробки програмного засобу для управління орендою житла.

Методи дослідження. Під час дослідження використовувалися наступні методи:

- методи розробки серверних застосунків засобами платформи Java і набору фреймворків Spring (Boot, Data, Web MVC, Security);

- методи розробки клієнтських вебзастосунків засобами мови програмування TypeScript і фреймворку Angular;

- методи організації баз даних для збереження та обробки даних, які стосуються процесу управління орендою житла;

- методи тестування, які використовуються для перевірки створених модулів програмного засобу для управління орендою житла.

Новизна отриманих результатів. Подальшого розвитку отримала модель програмного засобу для управління орендою житла, яка, на відміну від існуючих, інтегрує набір засобів для комунікації між учасниками процесу оренди, обліку платежів та комунальних послуг, збереження та керуваннями документами та покращує прозорість процесу через використання засобів аудиту дій учасників оренди.

Практична цінність отриманих результатів. Практична цінність отриманих результатів полягає у можливості використання інтегрованого програмного засобу, що покриває різні аспекти управління орендою житла. Програмний засіб може потенційно використовуватись як індивідуальними орендодавцями, так і окремими організаціями, які надають послуги оренди.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У роботах, опублікованих у співавторстві, автору належать такі результати: розробка моделі програмного засобу для управління орендою житла [1], аналіз систем керування базами даних для реалізації чату для обміну миттєвими повідомленнями [2].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення бакалаврської кваліфікаційної роботи були представлені на конференціях «Комп’ютерні ігри і мультимедіа, як інноваційний підхід до комунікації» (2024) та «Стан, досягнення і перспективи інформаційних систем і технологій» (2025) і були опубліковані у тезах конференцій [1-3].

Публікації. Результати дослідження були опубліковані у матеріалах конференцій «Стан, досягнення і перспективи інформаційних систем і технологій» (2025) [1-2] та «Комп`ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (2024) [3].

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ УПРАВЛІННЯ ОРЕНДОЮ ЖИТЛА ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз предметної галузі управління орендою житла

Ринок оренди житла в Україні характеризується високим рівнем фрагментації та недостатньою структурованістю. Частина орендних відносин залишається в межах тіньового сектору економіки, що ускладнює процеси державного регулювання та захисту прав учасників [1].

Поточний стан галузі вказує на необхідність впровадження інноваційних технологічних засобів для цифровізації процесів управління орендними відносинами. Аналіз існуючих практик демонструє наявність викликів, які можуть спричиняти конфліктні ситуації між суб’єктами орендних відносин [1]. По-перше, переважання паперового документообігу може бути причиною зайвих часових витрати та ризиків, пов’язаних з людським фактором. По-друге, відсутність комплексного підходу до обліку платежів та розрахунку комунальних послуг ускладнює фінансові взаємовідносини між орендодавцями та орендарями. По-третє, обмежені можливості для оперативної комунікації та розв’язання конфліктних ситуацій можуть знижувати загальну задоволеність учасників цих відносин.

Ключовою перевагою пропонованої системи є її потенціал для виведення частини орендних відносин з тіньового сектору економіки. Механізми електронного документообігу спрощуватимуть процес юридичного оформлення відносин між орендодавцем та орендарем. Автоматизація фінансових аспектів оренди створює прозору систему фінансових взаємовідносин, що спрощуватиме процес декларування доходів від оренди для орендодавців.

Міжнародна практика впровадження цифрових платформ для управління орендою житла, які надають можливості для документообігу, обліку платежів та комунікації, продемонструвала зниження операційних витрат на управління, покращення досвіду учасників процесу, спрощення ведення діяльності [4].

Цифровізація процесу управління орендою житла охоплює кілька ключових аспектів. Першим аспектом є електронний документообіг, який автоматизуватиме зберігання та аналіз договірної документації.

Другим важливим аспектом є управління фінансовими операціями, що включає в себе систематизацію обліку орендних платежів і комунальних витрат та формування аналітичної звітності. Система має надавати можливість створення структурованої історії фінансових операцій та забезпечує прозорість розрахунків Систематичний облік фінансових транзакцій сприятиме запобіганню виникненню спірних ситуацій та створює основу для формування статистичних даних щодо динаміки вартості оренди.

Третім аспектом функціонування системи є управління комунікацією між учасниками орендних відносин через структуровані звернення та інтегровані засоби миттєвого обміну повідомленнями. Даний функціонал забезпечує взаємодію між орендодавцями та орендарями, документування історії комунікації та оперативне розв’язання потенційних спірних питань.

Четвертим аспектом є комплексне управління об’єктами нерухомості, що включає систематизацію інформації про житлові приміщення, їх характеристики та наявне обладнання. Цей функціональний компонент має дозволяє створювати структуровані каталоги об’єктів нерухомості з детальним описом їх параметрів та здійснювати управління наявним житловим фондом для орендодавців.

Отже, аналіз стану систем управління орендою житла в Україні виявляє необхідність системного підходу до впровадження інноваційних технологічних засобів. Існуючі особливості ринку, пов’язані з низьким рівнем автоматизації та недостатньою прозорістю процесів, можуть бути покращенні за допомогою інтегрованих систем управління, що охоплюють усі аспекти орендних відносин: від обліку об’єктів нерухомості до формування фінансової звітності. Враховуючи зростаючий рівень цифровізації суспільства та розвиток електронних сервісів державних послуг, створення комплексних систем управління орендою є логічним кроком у напрямку модернізації ринку нерухомості в Україні.

1.2 Порівняльний аналіз аналогів

Впровадження інформаційних систем у процес управління орендою житла дозволяє покращити операційні процеси та, як наслідок, підвищити якість надання послуг оренди. Завдяки цифровізації таких операцій, як облік орендних платежів, обробка заявок на технічне обслуговування можуть скоротитися витрати часу на управління об’єктами оренди та мінімізуватимуться помилки, пов’язані з людським фактором. Використання інтегрованих систем сприяє прозорості фінансових операцій, оперативному опрацюванню запитів, спрощенню документообігу.

Ринок програмного забезпечення для управління орендою житла має низку програмних засобів із різним набором функцій:

- Rentster;
- Discussio;
- Pickspace;
- Rentroom.

Rentster – хмарна платформа для управління орендним бізнесом, що призначена для малих компаній, які надають послуги оренди різноманітного обладнання, транспортних засобів та приміщень (рис. 1.1). Система позиціонується як багатофункціональна цифрова платформа для управління відносинами з клієнтами, що забезпечує комплексну автоматизацію ключових бізнес-процесів, включаючи планування ресурсів, систему бронювання та резервування, а також безпроблемну інтеграцію з корпоративними вебсайтами компаній. Така інтеграція дозволяє створити єдиний екосистемний підхід до управління орендним бізнесом, покращуючи операційні процеси та підвищуючи загальну продуктивність роботи підприємства. Система управління договорами надає засоби для зберігання орендних контрактів. Особливістю Rentster є інтеграція з широким спектром технологій, включаючи розумні електронні замки, автоматизовані шлагбауми та бар'єри, системи відеоспостереження високої роздільної здатності, датчики руху та присутності, а також інші компоненти систем безпеки та моніторингу, що дозволяє створити повністю

автоматизоване середовище управління доступом, мінімізуючи людський фактор та підвищуючи рівень безпеки орендованих об'єктів [5].

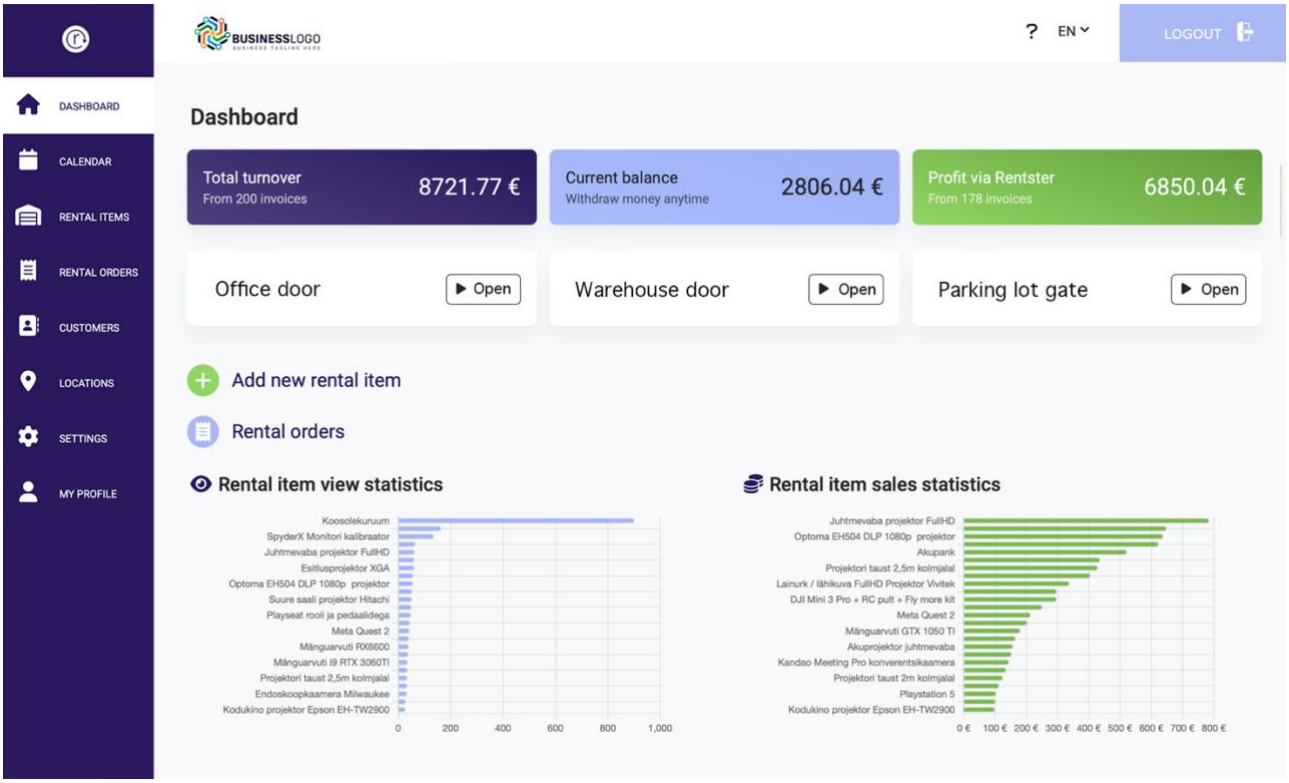

Рисунок 1.1 – Користувацький інтерфейс системи Rentster

Discussio – програмна платформа для управління орендною нерухомістю, що функціонує як екосистема для взаємодії орендодавців та орендарів у сфері житлової нерухомості (рис. 1.2). Система позиціонується як інтегрована соціальна мережа, що об’єднує функціональність пошуку житла, публікації оголошень та ведення орендних угод в єдиному середовищі. Застосунок має модуль управління оголошеннями для створення та публікації інформації про об’єкти оренди з можливістю фільтрації за різними параметрами, систему пошуку та підбору учасників орендних відносин з алгоритмами сповіщень про нові пропозиції, а також компонент для опрацювання орендних угод. Система управління об’єктами нерухомості дозволяє відстежувати стан приміщень, платежі та технічні роботи. Інтегрований підхід створює уніфікований досвід користування на всіх етапах орендного процесу та централізує дані в єдиному

сховищі, усуваючи необхідність експорту та імпорту інформації між різними сервісами [6].

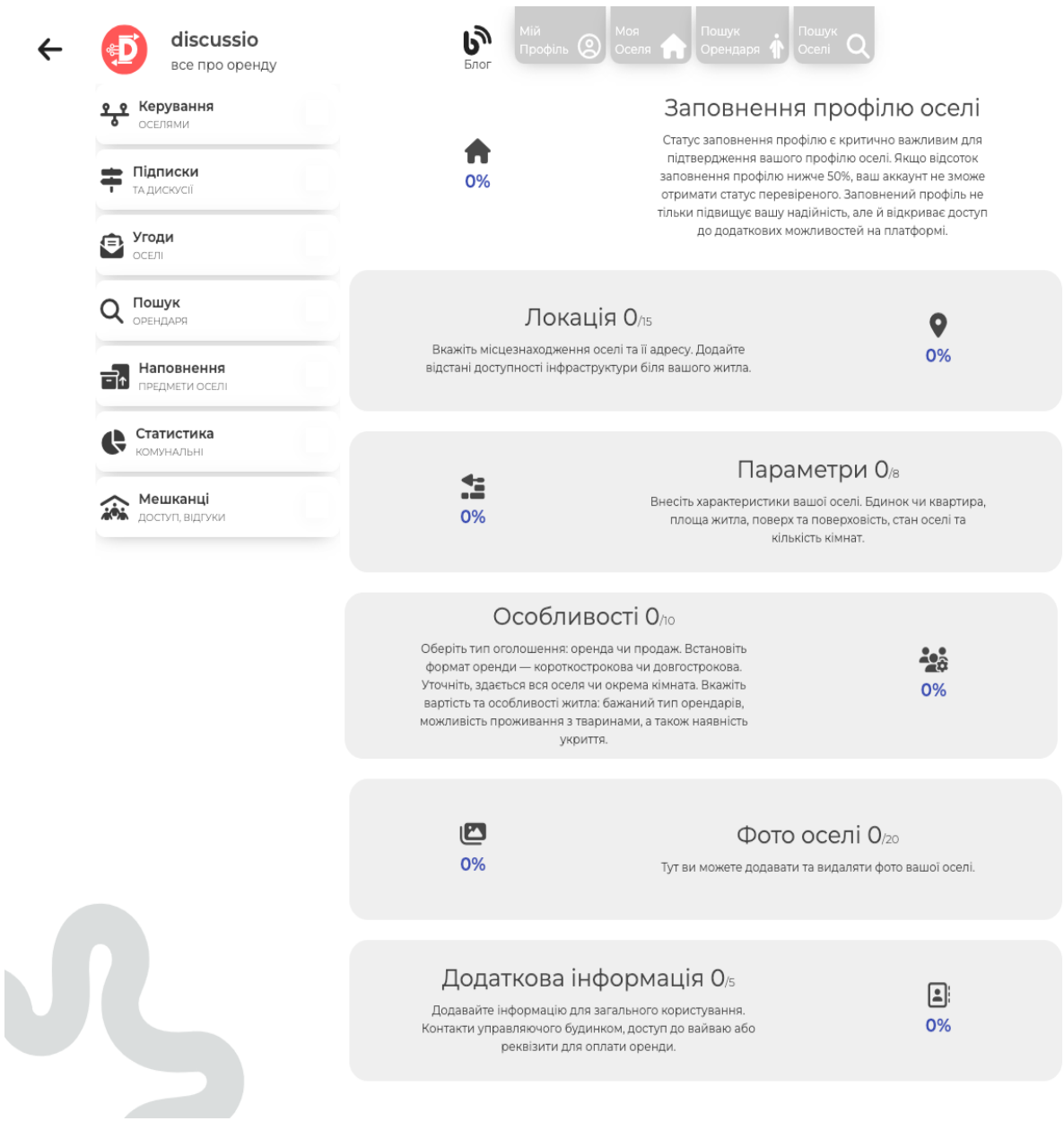

Рисунок 1.2 – Користувацький інтерфейс системи Discussio

Pickspace – це програмний застосунок для управління орендною нерухомістю (рис. 1.3). Функціонал управління об’єктами оренди в системі Pickspace розроблений з урахуванням ключових особливостей комерційної нерухомості. Центральним елементом системи є модуль конфігурації та імплементації складних тарифних планів, який дозволяє власникам нерухомості

розробляти диференційовані цінові моделі залежно від розташування, площі, терміну оренди та додаткових послуг. Модуль обліку платежів відзначається високим ступенем адаптивності до різноманітних комерційних тарифних моделей. Аналітичний компонент Pickspace дозволяє проводити комплексний аналіз використання орендних площ, включаючи моніторинг коефіцієнта заповнюваності, динаміки грошових потоків та рентабельності окремих об'єктів нерухомості [7].

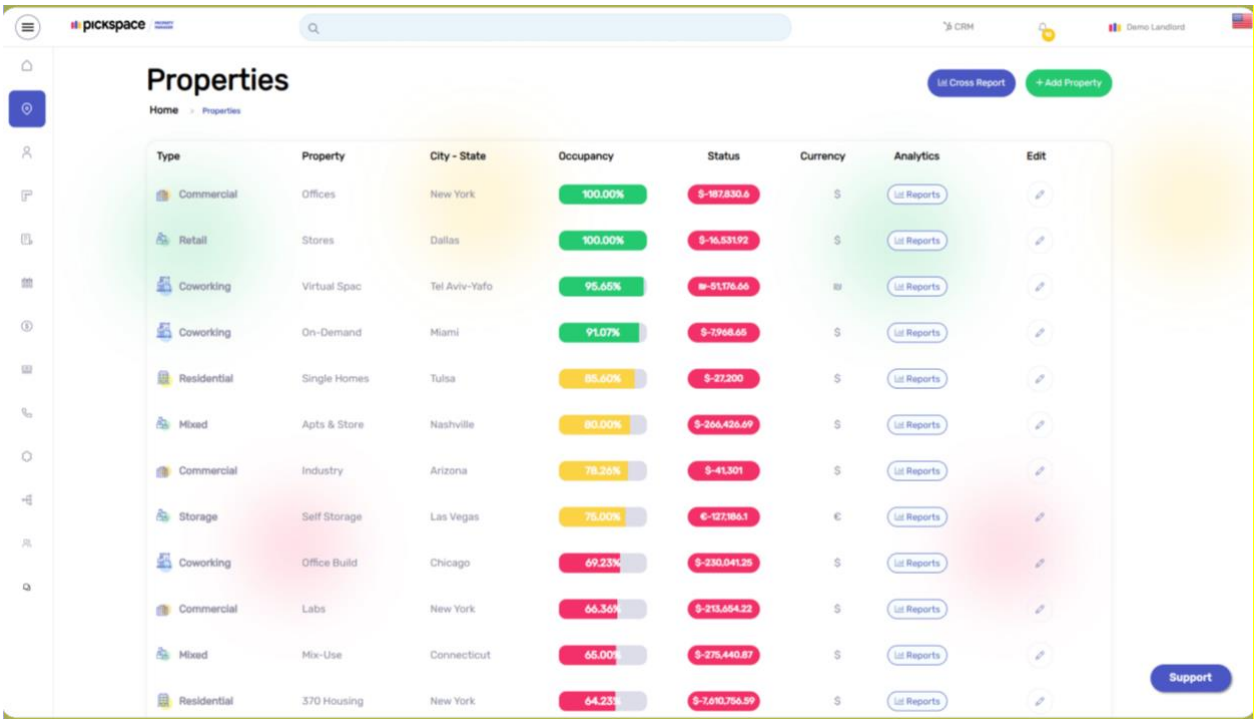

Рисунок 1.3 – Користувацький інтерфейс системи Pickspace

Rentroom – це хмарний сервіс для управління орендною нерухомістю, який надає набір інструментів для автоматизації процесів оренди (рис. 1.4). Ключовою особливістю комунікаційної складової системи є інтегрований підхід до організації інформаційної взаємодії між суб'єктами орендних відносин. Вона включає вбудований чат і систему структурованих звернень для оперативного зв'язку між орендодавцями та орендарями. Rentroom надає функції для детального обліку та аналізу платіжних транзакцій, пов'язаних з орендними відносинами. Це дозволяє управляти фінансовими потоками та забезпечує

прозорість фінансових взаємовідносин між орендарями та орендодавцями. Однак, система не надає функціоналу для обліку та моніторингу комунальних послуг [8].

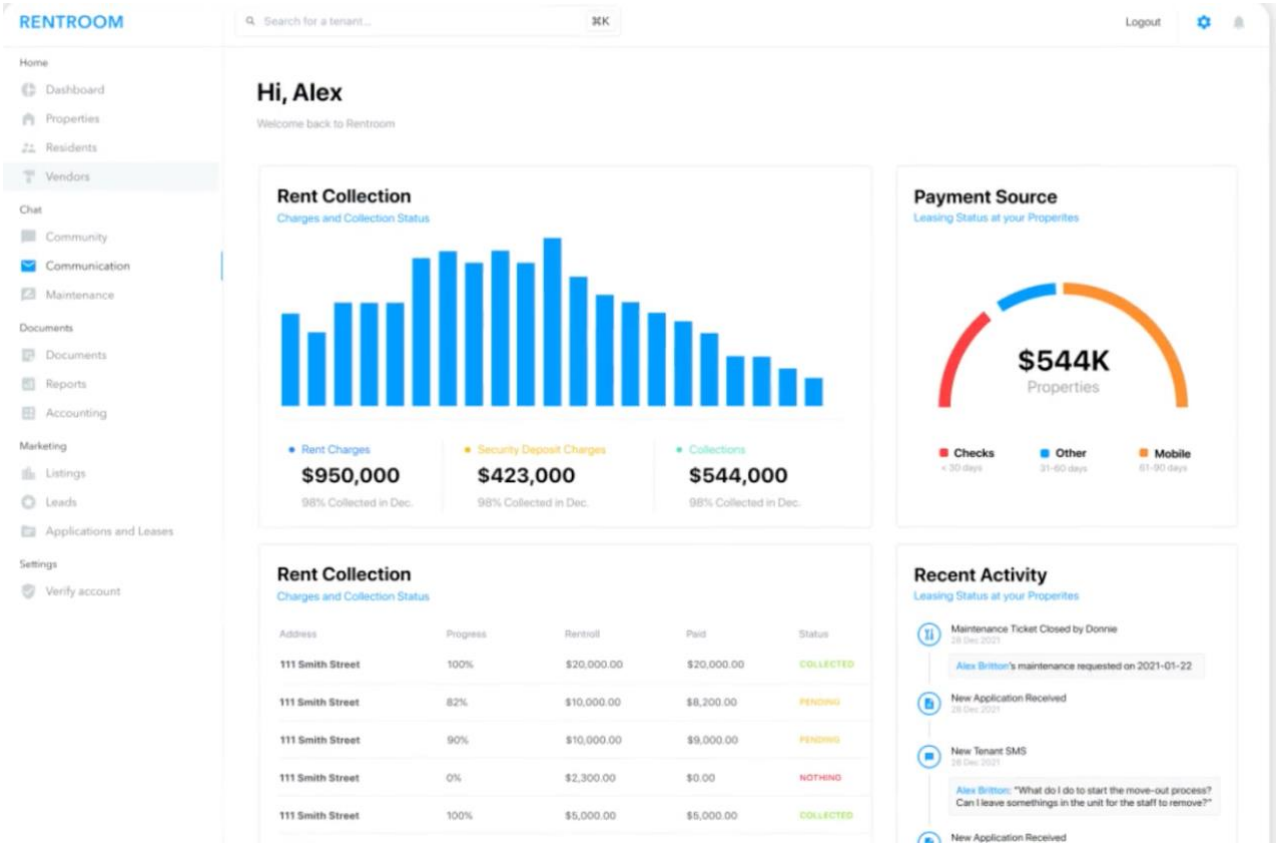

Рисунок 1.4 – Користувацький інтерфейс системи Rentroom

Розробка програмного засобу управління орендою житла полягає у створенні застосунку, здатного надати функції для спрощення керування об’єктами нерухомості і ведення договірних відносин. Комунікації між орендодавцями та орендарями мають бути забезпечені через систему на базі структурованих звернень та вбудованого чату для обміну миттєвими повідомленнями. Покращенню прозорості відносин сприятиме набір функцій для електронного документообігу, що надаватиме засоби для довготривалого збереження юридично вагомих документів, необхідних для оренди житла, та відстеження дій користувачів через механізм аудиту.

Результати порівняльного аналізу функціональних можливостей розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Rentster	Discussio	Pickspace	Rentroom	Власна розробка
Керування об’єктами оренди	+	+	+	+	+
Система збереження і керування документами	+	+	+	+	+
Вбудований чат для обміну миттєвими повідомленнями	-	+	-	+	+
Система структурованих звернень	-	-	+	+	+
Облік платежів	+	+	+	+	+
Облік комунальних послуг	-	+	-	-	+
Аудит дій користувачів	-	-	-	-	+
Загальна оцінка	42.8%	71.4%	57.1%	71.4%	100%

Зважаючи на дані у таблиці порівняльних характеристик, розробка власного програмного засобу є доцільною. Створювана система зможе об’єднати переваги існуючих програмних засобів, які функціонують на ринку, та покрити їх недоліки і цим надати розширені функції для управління орендою житла.

Отже, розробка власного програмного засобу управління орендою житла є доцільною, адже може сприяти трансформації ринку орендних відносин у галузі нерухомості. Така система потенційно матиме позитивний вплив на різні сфери соціально-економічної діяльності через детінізацію орендних відносин використовуючи електронний документообіг та автоматизацію обліку платежів, покращення комунікації між учасниками процесу за допомогою структурованих звернень та вбудованого чату, що, у свою чергу, сприятиме покращенню прозорості відносин між учасниками процесу.

1.3 Аналіз методів розв’язання завдань

Розробка програмного засобу для управління орендою житла вимагає комплексного підходу з урахуванням специфіки предметної галузі, вимог до масштабованості системи та особливостей взаємодії користувачів із системою.

Одним із ключових питань при розробці архітектури системи є вибір між монолітним та мікросервісним підходами. Мікросервісна архітектура полягає у декомпозиції системи на множину незалежних сервісів, кожен з яких відповідає за окрему функціональну область, має власну базу даних та розгортається незалежно від інших. Даний підхід забезпечує високий рівень масштабованості та гнучкості при розробці складних систем, проте характеризується складністю впровадження, зокрема через необхідність розв’язання задач міжсервісної комунікації, розподіленого моніторингу, обробки відмов та узгодженості даних. Однак, передчасне впровадження мікросервісної архітектури на початкових етапах розробки системи може призвести до суттєвого ускладнення проєкту без отримання відповідних переваг [9].

Альтернативним підходом є монолітна архітектура, у результаті використання якої буде створено єдиний застосунок, що містить всі функціональні модулі системи. Монолітна архітектура характеризується простотою розробки, тестування та розгортання, що особливо важливо на початкових етапах життєвого циклу програмного забезпечення. Водночас, за умови дотримання принципів високої згуртованості та низької зв’язаності між компонентами системи, монолітна архітектура не виключає можливості подальшої міграції до мікросервісного підходу при виникненні відповідних потреб.

З урахуванням проведеного аналізу, для розробки серверної частини системи управління орендою житла обрано монолітний підхід, що реалізує архітектурний патерн «багаторівнева архітектура» [10]. Даний патерн полягає у поділі системи на рівні контролерів, сервісів та репозиторіїв, що забезпечує чітке розмежування відповідальності між компонентами системи:

- рівень контролерів відповідає за обробку HTTP-запитів, валідацію вхідних даних та формування відповідей;

- рівень сервісів реалізує бізнес-логіку системи;

- рівень репозиторіїв забезпечує взаємодію з базою даних.

Для розробки клієнтської частини системи також обрано монолітний підхід, що реалізуватиме патерн «компонентно-орієнтована архітектура» [10]. Суть цього підходу полягає в організації програмного коду у вигляді компонентів, що інкапсулюють логіку, відображення та стилізацію, а також дозволяє реалізувати взаємодію між компонентами через механізм сервісів.

Обрані архітектурні та технологічні підходи дозволяють досягти балансу між простотою розробки та підтримки системи на початкових етапах її життєвого циклу та можливістю подальшого масштабування і еволюції архітектури відповідно до потреб.

Таким чином, для реалізації системи управління орендою житла обрано монолітний підхід з чітким розмежуванням відповідальності між компонентами на серверній через використання патерну «багаторівнева архітектура», та на клієнтській частині через використання патерну «компонентно-орієнтована архітектура».

1.4 Постановка задач дослідження

Проаналізувавши переваги та недоліки існуючих програмних систем для управління орендою житла, було визначено основні напрямки розробки власного програмного засобу:

1. Проаналізувати вхідні дані системи.

2. Розробити модель та алгоритми роботи програмного засобу.

3. Розробити метод аудиту дій користувачів.

3. Обрати засоби збереження даних відповідно до властивостей даних, якими вони оперуватимуть.

4. Розробити застосунок для управління орендою житла, що складається з:

- модуля керування об’єктами нерухомості, який призначений для керування даними про наявні в орендодавця житлові приміщення;

- модуля документообігу для збереження пов’язаних із орендними відносинами документів;

- модуля обліку платежів і комунальних послуг, який керує даними про здійснені фінансові операції та використані комунальні послуги;

- модуля комунікацій, який надає різнотипні методи взаємодії між учасниками процесу оренди житла;

- модуля керування договорами, який інтегрує інші модулі навколо договірних відносин;

- вебінтерфейсу для взаємодії користувачів із системою.

6. Провести тестування програмного засобу.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

Отже, було встановлено, що сучасний ринок оренди житла може бути покращений через впровадження програмних застосунків для цифровізації основних аспектів процесу оренди житлової нерухомості. Порівняльний аналіз існуючих програмних продуктів (Rentster, Discussio, Pickspace, Rentroom) продемонстрував наявність певних функціональних обмежень у кожній із систем. Таким чином, розробка власного програмного засобу є обґрунтованою і доцільною з огляду на можливість об’єднання переваг існуючих програмних систем та усунення виявлених функціональних обмежень.

На основі аналізу архітектурних підходів було обґрунтовано доцільність використання монолітної архітектури з чітким розмежуванням відповідальності між компонентами. Також, було визначено перелік завдань, які необхідно виконати для розробки власного програмного засобу для управління орендою житла.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ЗАСОБУ ДЛЯ УПРАВЛІННЯ ОРЕНДОЮ ЖИТЛА

2.1 Аналіз вхідних даних програмного засобу для управління орендою житла

Аналіз вхідних даних для програмної системи управління орендою житла потребує визначення основних інформаційних потоків, які формуються в процесі орендних відносин. Дані, що оброблятимуться системою, характеризуються різнорідністю, специфічною структурою та вимогами до обробки, що зумовлює необхідність комплексного підходу до їх організації та зберігання.

Ретельний аналіз характеристик вхідних даних є фундаментальним етапом проєктування архітектури системи, оскільки дозволяє обрати найдоцільнішу стратегію вибору та налаштування сховищ даних. Відмінності в частоті звернень, мінливості структури, обсягах та схемах доступу до різних категорій даних вимагають диференційованого підходу до їх зберігання. Невідповідність обраного сховища особливостям даних може призвести до істотних складнощів з покращенням швидкодії, розширюваністю та цілісністю даних у процесі використання системи. Коректний вибір сховищ даних відповідно до специфіки цих даних може сприяти кращому співвідношенню між експлуатаційними витратами, швидкодією та надійністю системи протягом її життєвого циклу.

Процес розробки системи управління орендою житла складається з створення п’ятьох основних функціональних модулів: керування об’єктами оренди, комунікації, документообігу, обліку платежів і комунальних витрат та керування договорами. Кожен із цих модулів оперує специфічними наборами даних, що потребують відповідних підходів до організації та зберігання.

Модуль керування об'єктами оренди оперує чітко структурованими даними, що характеризують житлові приміщення. Ключовими елементами даних цього модуля є параметри об'єктів нерухомості, що включають базову інформацію (такі, як адреса, площа, кількість кімнат), технічні характеристики та параметри оренди. Особливу категорію формують мультимедійні дані –

фотографії об'єктів, що зберігатимуться у спеціалізованому сховищі. Дані цього модуля забезпечують формування структурованого каталогу житлового фонду для управління наявними об'єктами нерухомості.

Модуль документообігу призначений для роботи з даними про документи, що характеризуються високими вимогами до цілісності. Основними даними цього модуля є метадані документів, такі як назва, дата створення тощо, та інформація про версії документів. Система забезпечує зберігання історії змін документів, при цьому файли документів розміщуються в окремому сховищі.

Модуль обліку платежів і комунальних послуг працює з фінансовими даними, що характеризуються регулярністю оновлення та необхідністю збереження хронологічної послідовності. Основні інформаційні потоки включають дані про фінансові операції у межах договірних відносин, інформацію про комунальні послуги (типи послуг, показники лічильників). Специфікою даних цього модуля є необхідність підтримки транзакційної цілісності та забезпечення можливості формування аналітичної звітності на основі історичних даних. При зростанні обсягів даних може застосовуватися шардинг за часовими періодами, що дозволить покращити продуктивність запитів до історичних даних без зниження швидкодії доступу до актуальної інформації [3].

Модуль комунікацій оперує даними двох основних типів: структурованими зверненнями та повідомленнями чату. Структуровані звернення містять дані про тип, пріоритет, статус та зміст запиту. Дані, якими оперує чат для обміну миттєвими повідомленнями, характеризуються високою інтенсивністю обміну та потребують механізмів забезпечення швидкого доступу до актуальних даних з одночасним збереженням історії комунікації.

Модуль керування договорами виконує інтеграційну роль, об'єднуючи дані з інших функціональних модулів системи. Основними елементами даних є параметри договірних відносин (сторони, дати), а інтеграція з іншими модулями здійснюється переважно через механізми зовнішніх ключів реляційної моделі даних. Такий підхід забезпечує встановлення стійких зв'язків між сутностями

різних модулів (об'єкти нерухомості, документи, платежі) з одночасним збереженням їх логічної незалежності. Організація даних у цьому модулі реалізує концепцію єдиної точки доступу до всієї інформації, пов'язаної з конкретними договірними відносинами, що потенційно спрощуватиме управління орендними процесами.

Аналіз вхідних даних системи управління орендою житла свідчить про необхідність застосування гібридного підходу до їх зберігання та обробки. Для структурованих даних з високими вимогами до транзакційної цілісності доцільним є використання реляційних баз даних із можливістю застосування шардингу у майбутньому для покращення продуктивності при зростанні обсягів інформації [3]. Для даних з гнучкою структурою та високою інтенсивністю обміну такими, як повідомлення чату, доцільним є застосування нереляційного сховища даних [2]. Файлові дані розміщуватимуться у спеціалізованому сховищі з можливістю масштабування при зростанні обсягів інформації. Такий підхід дозволяє досягти балансу між надійністю зберігання даних та продуктивністю системи в умовах різноманітних інформаційних потоків, що характеризують процес управління орендою житла.

2.2 Розробка ER-моделі предметної галузі та моделі системи

Для формалізації та структуризації даних у програмній системі управління орендою житла розроблено ER-модель (модель «сутність-зв'язок»), що відображає ключові сутності предметної області та взаємозв'язки між ними. Ця модель забезпечує концептуальне представлення інформаційної структури системи, визначаючи основні об'єкти даних та їх відношення, що є фундаментальною основою для подальшої реалізації бази даних.

Розроблена ER-модель (рис. 2.1) відображає систему взаємозв'язків між сутностями, що відповідають різним функціональним аспектам процесу управління орендою житла.

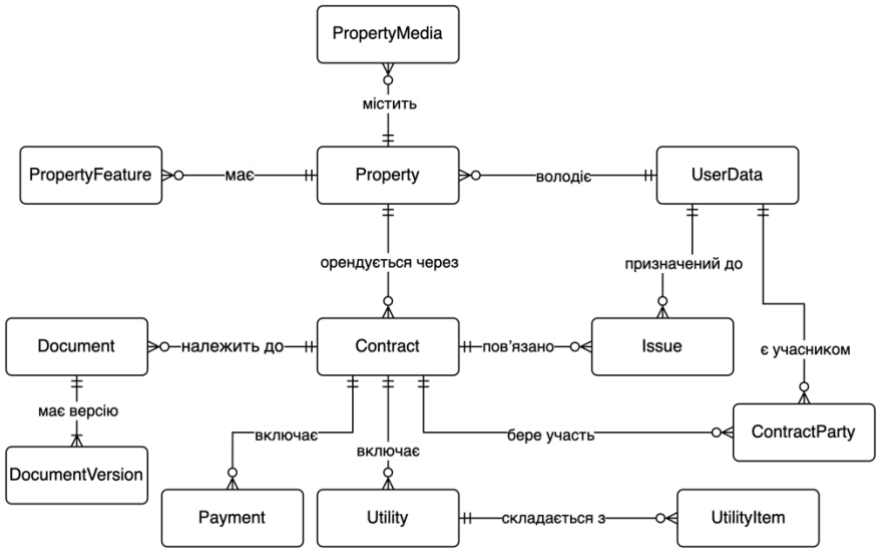

Рисунок 2.1 – ER-модель системи

Відношення між сутностями реалізовані через зв'язки типів один-до-багатьох з використанням також проміжних сутностей, які реалізують зв’язок багато-до-багатьох.

Центральними елементами розробленої ER-моделі є сутності, пов'язані з основними функціональними модулями системи. Сутність «Property» (об'єкт нерухомості) містить атрибути, що характеризують житлове приміщення, включаючи основну інформацію про місцезнаходження, параметри та статус об'єкта. Для деталізації характеристик об'єктів нерухомості введено допоміжні сутності «PropertyFeature» (особливості об'єкта) та «PropertyMedia» (медіа-файли об'єкта), що знаходяться у відношенні один-до-багатьох з основною сутністю.

Сутність «Contract» (договір) відображає договірні відносини між учасниками процесу оренди та містить інформацію про строки оренди, фінансові умови та статус договірних відносин. Ця сутність пов'язана з об'єктом нерухомості через зв'язок «орендується через», що забезпечує інтеграцію даних між модулями керування об'єктами оренди та договорами. Для визначення сторін договору введено сутність «ContractParty» (сторона договору), яка містить інформацію про роль учасника в договорі та знаходиться у відношенні багато-до-одного з договором через зв'язок «бере участь».

Фінансові аспекти оренди відображені в сутностях «Payment» (платіж) та «Utility» (комунальні послуги). Сутність «Payment» містить інформацію про платежі за оренду, а сутність «Utility» – про комунальні витрати за певний період. Деталізація комунальних послуг здійснюється через сутність «UtilityItem» (елемент комунальних послуг), що знаходиться у відношенні багато-до-одного з основною сутністю «Utility» через зв'язок «включає».

Модуль документообігу представлений сутностями «Document» (документ) та «DocumentVersion» (версія документа). Сутність «Document» містить метадані документа та інформацію про основний файл документа, а «DocumentVersion» забезпечує відстеження змін у документах та підтримку історії їх модифікацій. Між документами та договорами встановлений зв'язок «належить до», що дозволяє асоціювати документи з конкретними договірними відносинами.

Модуль комунікації представлений сутністю «Issue» (звернення), що містить інформацію про структуровані запити користувачів. Ця сутність є основою для організації формалізованої комунікації між учасниками орендних відносин та пов'язана з користувачами через зв'язки «призначено» і «створено» та із сутністю «Contract» – зв’язком «пов’язано». Сутності чату для обміну миттєвими повідомленнями є відділеними від основної ER-моделі, адже не мають строгих зв’язків з іншими сутностями системи.

Інформація про користувачів системи зберігається в сутності «UserData» (дані користувача), що містить особисті та контактні дані. Ця сутність є ключовою також для забезпечення авторизації та аутентифікації в системі і пов'язана з іншими сутностями через зв'язки, що визначають авторство дій та учасників орендних відносин.

Для забезпечення цілісності даних та відстеження змін усі основні сутності містять службові атрибути, що формують основу для механізмів аудиту та забезпечення цілісності даних у системі.

Для структурованої візуалізації архітектури програмної системи управління орендою житла розроблено модель системи з використанням UML-

діаграми компонентів (рис 2.2). Така модель слугує основою для подальшої детальної розробки кожного компонента системи, визначаючи їх функціональні можливості та механізми взаємодії між собою.

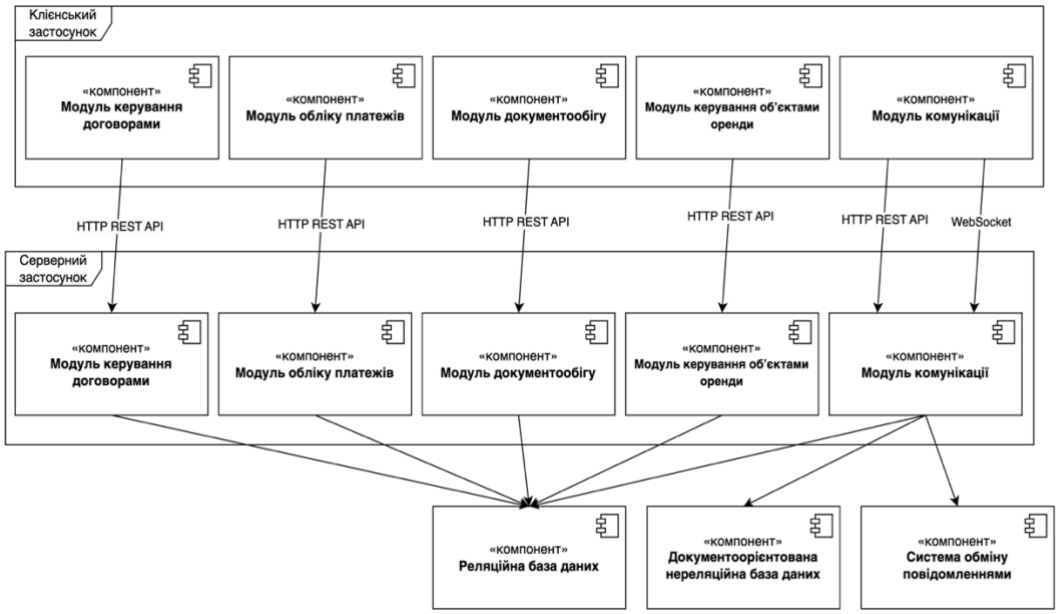

Рисунок 2.2 – UML-діаграма компонентів системи

Основу архітектури системи становлять серверна та клієнтська частини. Серверна частина складатиметься з п'яти ключових функціональних модулів: модуля керування об'єктами оренди, модуля комунікації, модуля документообігу, модуля обліку платежів і комунальних витрат та модуля керування договорами.

Модуль керування об'єктами оренди забезпечуватиме операції з даними житлових приміщень через сервіси управління об'єктами та роботи з мультимедійними даними, взаємодіючи з базою даних для зберігання інформації про нерухомість.

Модуль комунікації реалізуватиме взаємодію користувачів через чат та структуровані звернення. Сервіс для повідомлень використовуватиме документоорієнтовану нереляційну базу даних [2] для збереження повідомлень, для асинхронної комунікації в реальному часі буде інтегровано систему обміну повідомленнями. Сервіс управління зверненнями використовуватиме реляційну базу даних для збереження структурованої інформації про звернення.

Модуль документообігу оперуватиме метаданими і власне файлами документів, які використовуються під час орендних відносин, через взаємодію з базою даних для збереження інформації про них.

Модуль обліку платежів і комунальних витрат автоматизуватиме фінансові аспекти оренди через сервіси управління платежами та розрахунку комунальних послуг, зберігаючи дані у базі даних.

Модуль керування договорами об'єднує дані із модулів керування об'єктами оренди, комунікації, документообігу обліку платежів і комунальних витрат навколо окремого договору між учасниками процесу оренди. Для інтеграції цих даних використовуватимуться зв'язки між сутностями через зовнішні ключі у базі даних.

Клієнтська частина системи складатиметься з відповідних наборів компонентів користувацького інтерфейсу для кожного функціонального модуля.

Для забезпечення обміну даними між клієнтською та серверною частинами системи використовуватимуться два основні механізми: HTTP REST API [11] для стандартних операцій та WebSocket [12] для комунікації в реальному часі. REST API буде наданий контролерами, які оброблятимуть HTTP-запити для кожного функціонального модуля, а WebSocket використовуватиметься для обміну повідомленнями в чаті.

Для роботи з даними система використовуватиме два типи сховищ: реляційну базу даних для зберігання структурованих даних більшості модулів та документоорієнтовану нереляційну базу даних для реалізації чату.

2.3 Розробка алгоритмів роботи системи

Для коректного функціонування системи управління орендою житла необхідний аналіз і розробка алгоритмів роботи ключових процесів. Найбільш критичними для безпеки та продуктивності системи є алгоритм автентифікації та авторизації користувачів і алгоритм збереження повідомлень в режимі реального часу. Ці алгоритми були обрані для детальної розробки через їхню комплексність та вплив на загальну архітектуру системи.

Блок-схема (рис. 2.3) відображає принцип роботи алгоритму автентифікації та авторизації користувачів. Ключовою особливістю цього алгоритму є використання JWT-токенів, які використовуватимуться клієнтським вебзастосунком для авторизації взаємодій із сервером.

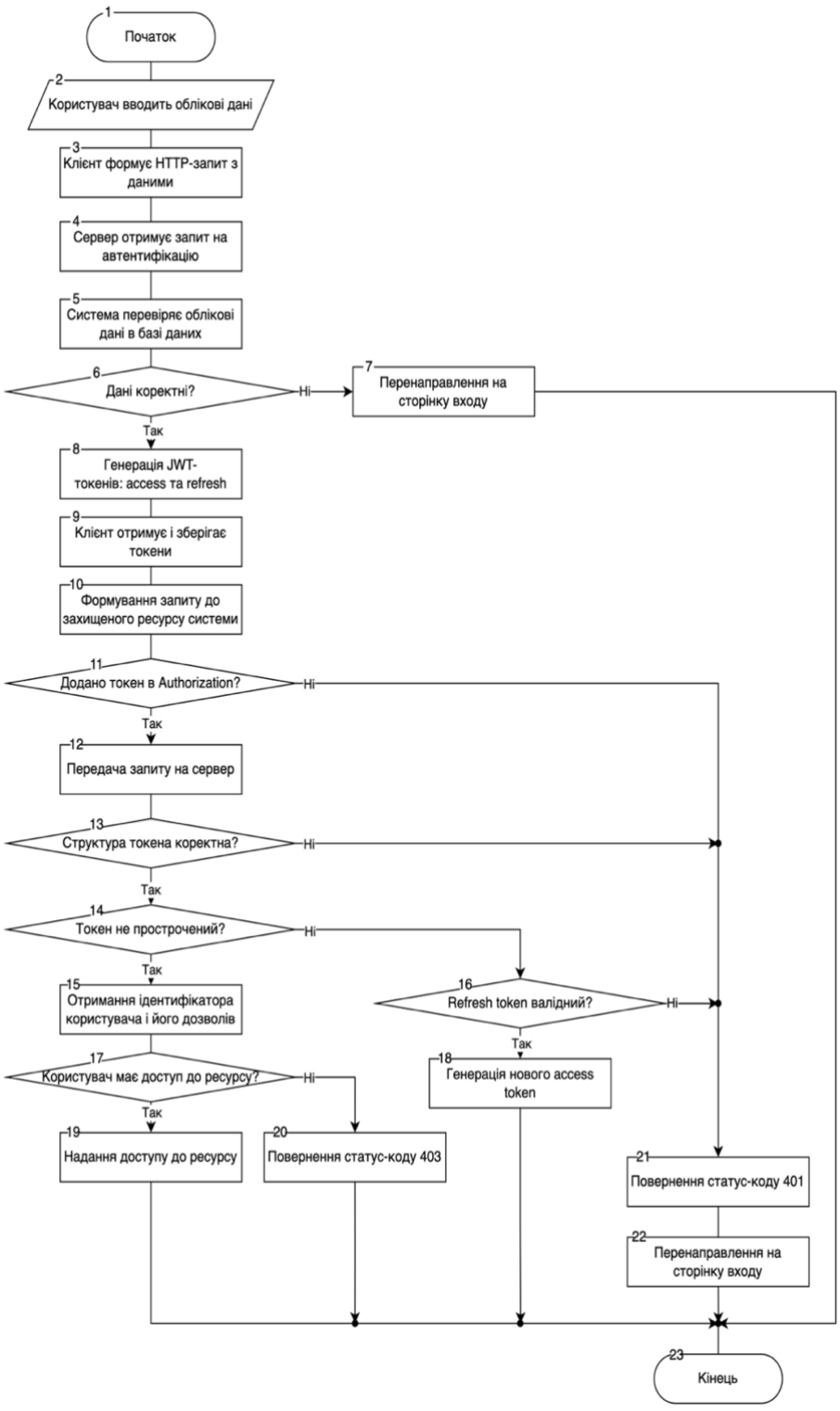

Рисунок 2.3 – Блок-схема алгоритму автентифікації та авторизації

JWT-токени (JSON Web Tokens) представляють собою компактний, самодостатній механізм безпечної передачі інформації між сторонами у вигляді JSON-об'єкта [13]. Структурно JWT-токен складається з трьох частин: заголовка, корисного навантаження та підпису, розділених крапками. Особливістю JWT-токенів є їх статичність, що дозволяє зберігати всю необхідну інформацію безпосередньо у токені, зменшуючи потребу в додаткових запитах до бази даних при автентифікації та авторизації. Криптографічний підпис забезпечує цілісність даних та запобігає несанкціонованій модифікації інформації. JWT-токени можуть містити довільні атрибути користувача, такі як ідентифікатор, роль, дозволи та термін дії, що дозволяє гнучко налаштовувати механізми контролю доступу в програмних системах.

Процес автентифікації розпочинається, коли користувач надає свої облікові дані через інтерфейс системи. Отримавши ці дані, клієнтська частина формує HTTP-запит до серверного компонента. Система здійснює перевірку наданих облікових даних, співставляючи їх із інформацією, збереженою в базі даних. При успішній верифікації генеруються два токени: короткостроковий access token для безпосереднього доступу до ресурсів системи та довгостроковий refresh token для автоматичного оновлення сесії без повторного введення облікових даних.

При кожному запиті до захищених ресурсів клієнтська частина автоматично додає access token до заголовків запиту. Серверний компонент перевіряє цілісність та термін дії токена, а також отримує з нього інформацію про ідентифікатор користувача, його роль та дозволи в системі. На основі цих даних перевіряється можливість доступу до запитуваного ресурсу.

Важливим аспектом алгоритму є механізм автоматичного оновлення токенів. У випадку закінчення терміну дії access token система перевіряє наявність та валідність refresh token. При наявності дійсного токена оновлення система автоматично генерує нову пару токенів, забезпечуючи безперервність сесії користувача без необхідності повторної автентифікації. Якщо refresh token недійсний або відсутній, користувач перенаправляється на сторінку входу для введення облікових даних.

Для забезпечення комунікації через використання чату для обміну миттєвими повідомленнями між користувачами системи розроблено алгоритм збереження повідомлень, блок-схема якого показана на рис. 2.4. Цей алгоритм базується на інтеграції трьох ключових технологій: WebSocket для двосторонньої комунікації з клієнтською частиною, системи асинхронного обміну повідомленнями для розподілу повідомлень та бази даних для постійного зберігання повідомлень.

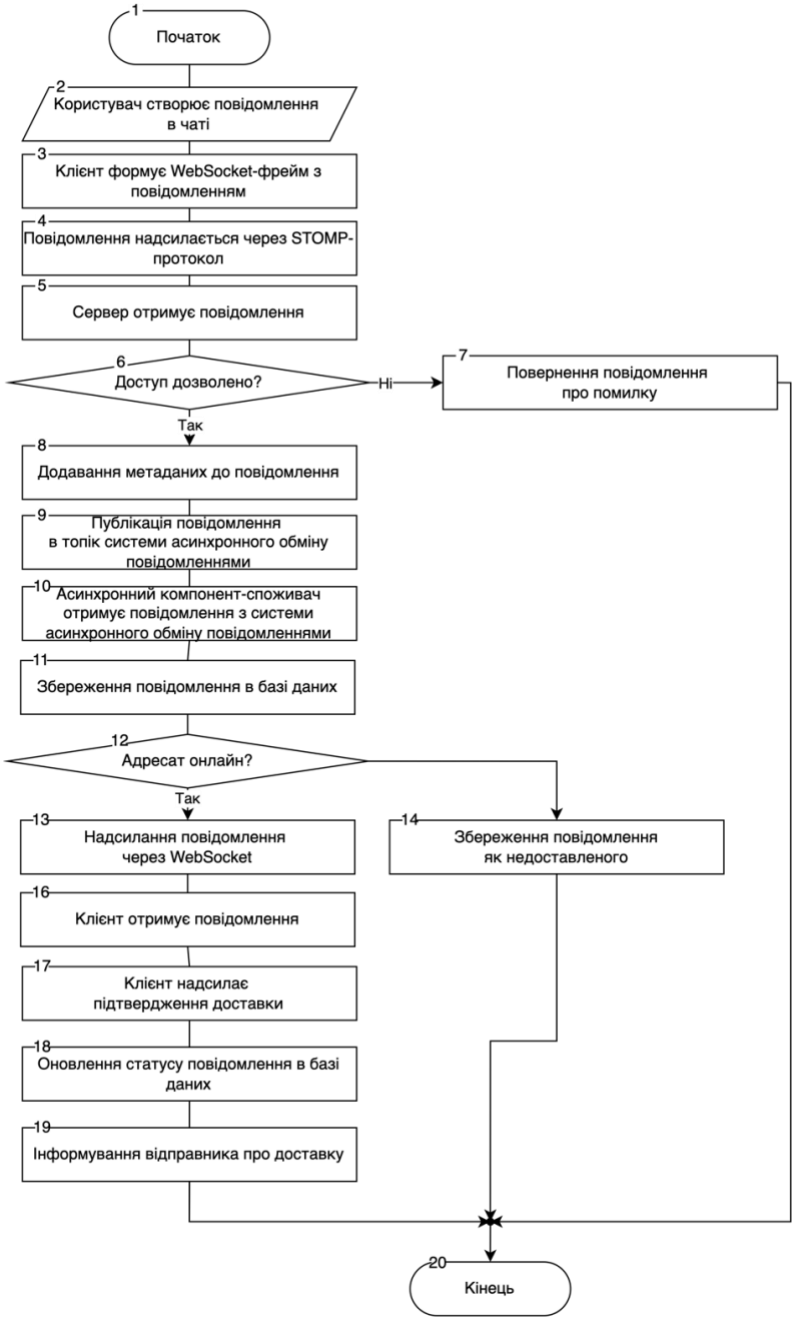

Рисунок 2.4 – Блок-схема алгоритму збереження повідомлень чату

Процес обміну повідомленнями розпочинається, коли користувач створює нове повідомлення у інтерфейсі чату. Клієнтська частина системи перетворює дані повідомлення у формат WebSocket-фрейму та надсилає його на сервер через STOMP-протокол. Отримавши повідомлення, серверний компонент здійснює перевірку прав доступу користувача до відповідного діалогу та, за умови успішної перевірки, доповнює повідомлення необхідними метаданими: унікальним ідентифікатором, часовою міткою та початковим статусом доставки.

Оброблене повідомлення публікується у відповідній черзі системи асинхронного обміну повідомленнями за ідентифікатором чату. Таким способом забезпечується збереження порядку повідомлень у межах одного діалогу та дозволятиме масштабувати систему при зростанні навантаження. Асинхронний компонент-споживач отримує повідомлення з черги, трансформує його у формат, призначений для зберігання, та записує у відповідну колекцію бази даних.

Після збереження повідомлення система визначає поточний статус адресатів. Для користувачів, які мають активне WebSocket-з’єднання, повідомлення негайно надсилається через відповідний канал підписки. Клієнтська частина, отримавши повідомлення, відображає його в інтерфейсі та надсилає підтвердження про доставку. На основі цього підтвердження система оновлює статус повідомлення в базі даних та інформує відправника про успішну доставку.

Для користувачів, які перебувають у статусі офлайн, система зберігає повідомлення як недоставлене. Такі повідомлення будуть доставлені при наступному підключенні користувача до системи, забезпечуючи безперервність комунікації.

Використання системи асинхронного обміну повідомленнями як проміжного шару між компонентами обробки WebSocket та бази даних забезпечує високу надійність процесу, гарантуючи збереження повідомлень навіть при тимчасових збоях окремих компонентів системи. Такий підхід також дозволяє розподілити навантаження між різними компонентами, забезпечуючи

стабільну роботу системи навіть при зростанні кількості користувачів та інтенсивності обміну повідомленнями.

Отже, алгоритми автентифікації, авторизації та обміну повідомленнями, розроблені для програмного засобу управління орендою житла, розв'язують основні задачі безпеки та комунікації через чат відповідно. Алгоритм автентифікації з використанням JWT-токенів надає надійний і прозорий захисний механізм, зберігаючи зручність для користувачів, а алгоритм обробки повідомлень, що інтегрує WebSocket, систему асинхронного обміну повідомленнями та базу даних, гарантуватиме стабільну та масштабовану комунікацію. Впровадження асинхронної обробки з проміжною ланкою забезпечуватиме стійкість до технічних збоїв та балансування навантаження.

2.4 Розробка методу аудиту дій користувачів

Аудит дій користувачів є одним з критично важливих компонентів системи управління орендою житла, адже він має надавати можливість відстеження змін даних, аналізу історії операцій та розслідування потенційних інцидентів безпеки. Для реалізації такого функціоналу необхідно розробити метод, який гарантуватиме повноту, точність та надійність збереження інформації про дії користувачів у системі.

У процесі проектування методу аудиту були проаналізовані різні підходи до відстеження дій користувачів, кожен з яких має специфічні характеристики, переваги та обмеження. Основними розглянутими методами були: імплементація на рівні прикладного програмного коду, використання аспектно-орієнтованого програмування та впровадження механізму тригерів бази даних.

Метод імплементації на рівні прикладного програмного коду полягає у внесенні логіки аудиту безпосередньо в бізнес-логіку системи. Цей підхід забезпечує високу гнучкість, дозволяючи реєструвати довільні події та параметри у процесі виконання операцій. Однак, його суттєвим недоліком є висока ймовірність виникнення помилок через необхідність дублювання коду в

різних частинах системи та потенційна неповнота охоплення всіх можливих шляхів модифікації даних.

Використання аспектно-орієнтованого програмування [14] дозволяє винести логіку аудиту в окремі аспекти, які перехоплюють виконання методів модифікації даних. Цей підхід забезпечує кращу модульність коду та знижує ймовірність пропуску операцій. Проте, він вимагає ретельного визначення точок з'єднання та може призвести до ускладнення процесу налагодження системи через неявне виконання коду.

Також одним із підходів для реалізації аудиту дій користувачів у системі управління орендою житла є використання тригерів бази даних [15]. Тригер – це спеціальна збережена процедура, яка автоматично виконується у відповідь на певні події в таблиці або представленні бази даних, такі як операції вставки, оновлення або видалення даних. Перевагами використання тригерів для реалізації аудиту є надійність, оскільки тригери активуються незалежно від того, яким чином виконується модифікація даних – через програмний код, збережені процедури чи прямі SQL-запити, що гарантує повноту охоплення всіх операцій. Тригери виконуються безпосередньо в середовищі бази даних, мінімізуючи додаткові витрати на мережеву взаємодію та перетворення даних, що забезпечує високу продуктивність. Операції аудиту виконуються в рамках тієї ж транзакції, що й основні зміни даних, забезпечуючи атомарність і цілісність журналу аудиту. Логіка аудиту зосереджена в одному місці, що спрощує підтримку та модифікацію системи.

Для реалізації аудиту через використання такого методу, у системі має бути створено тригер для кожної таблиці із даними, які підлягають аудиту, який реєструватиме інформацію про зміни даних в окремій таблиці для журналу аудиту. Журнал аудиту міститиме детальну інформацію про кожну операцію, включаючи ідентифікатор користувача, часову мітку, тип операції (вставка, оновлення, видалення), назву таблиці, ідентифікатор запису та значення атрибутів до та після операції.

Для ідентифікації користувача, який ініціював зміни в базі даних, система використовуватиме додаткові колонки в таблицях, які міститимуть інформацію про користувача-ініціатора. Серверний застосунок забезпечуватиме автоматичне заповнення цих полів при виконанні операцій модифікації даних. Таким чином, тригери зможуть отримувати інформацію про користувача безпосередньо з даних, що модифікуються.

Таким чином, використання тригерів бази даних для реалізації методу аудиту дій користувачів забезпечить надійне відстеження всіх операцій модифікації даних у системі управління орендою житла, що є важливою складовою забезпечення прозорості, безпеки та відповідності нормативним вимогам.

2.5 Розробка методу збереження файлів у системі

Збереження файлових даних у системі управління орендою житла становить одне з ключових завдань проєктування архітектури програмного застосунку. У результаті аналізу предметної області було виявлено, що система може оперувати різнотипними файловими даними, які стосуються різних аспектів процесу управління орендою житла, зокрема документами договірних відносин та фотоматеріалами об'єктів нерухомості. Для забезпечення надійного збереження цих даних необхідно визначити метод, який задовольнятиме вимоги щодо продуктивності, масштабованості та надійності.

У процесі розробки методу збереження файлів було проаналізовано три основні підходи: використання файлової системи сервера [16], інтеграція з реляційною базою даних [16] та впровадження спеціалізованого об’єктного сховища [17]. Кожен з цих підходів має власні характеристики, що визначають доцільність їх застосування в контексті системи управління орендою житла.

Використання файлової системи сервера [16] полягає у збереженні файлів безпосередньо на дисковому просторі сервера з відповідною організацією директорій та системою іменування файлів. Цей підхід характеризується простотою реалізації та високою швидкістю доступу до файлів у межах одного

сервера. Однак, він має обмеження щодо масштабованості, оскільки при збільшенні кількості серверів може стати необхідним розв’язання задачі синхронізації та узгодження доступу до файлів. Окрім того, даний підхід не забезпечує достатнього рівня надійності та відмовостійкості без впровадження додаткових механізмів реплікації та резервного копіювання.

Інтеграція з реляційною базою даних полягає у збереженні файлів у вигляді бінарних об'єктів (BLOB) безпосередньо в таблицях бази даних [16]. Такий підхід забезпечує узгодженість даних та можливість використання транзакційного механізму бази даних для забезпечення цілісності. Проте, збереження великих обсягів бінарних даних у реляційній базі даних може призвести до зниження продуктивності системи, ускладнення процедур резервного копіювання та відновлення даних, а також до надмірного навантаження на систему управління базами даних.

Третім розглянутим підходом є впровадження спеціалізованого об’єктного сховища для збереження файлів [17]. Сховище об’єктів – це розподілена система зберігання даних, безпосередньо призначена для управління бінарними об'єктами, що забезпечує високу продуктивність, масштабованість та надійність. Сучасні сховища об’єктів надають широкі можливості для організації даних, контролю доступу та інтеграції з іншими компонентами інформаційної системи.

Проведений аналіз показав, що для системи управління орендою житла найбільш доцільним є використання об’єктного сховища для збереження файлів. Цей вибір обумовлений такими перевагами, як: висока масштабованість, надійність та відмовостійкість завдяки автоматичній реплікації даних, вбудовані механізми безпеки та простота інтеграції з іншими компонентами системи завдяки стандартизованим API.

Для організації даних у об’єктному сховищі запропоновано створення окремих контейнерів (бакетів) для різних типів файлів. Зокрема, буде створено два основні бакети: «documents» для зберігання документів, пов'язаних з договірними відносинами, та «property-media» для зберігання фотоматеріалів об'єктів нерухомості. Такий поділ забезпечить логічну ізоляцію даних різних

категорій та дозволить застосовувати різні політики доступу та управління життєвим циклом для кожної категорії файлів.

У кожному бакеті файли будуть організовані за ієрархічною структурою, що відображає логічну структуру даних у системі. Для документів шлях до файлу формуватиметься за шаблоном «documents/{contract_id}/{document_id}/{version_id}», де contract_id – ідентифікатор договору, document_id – ідентифікатор документа, version_id – ідентифікатор версії документа. Для фотоматеріалів об'єктів нерухомості використовуватиметься шаблон «property-media/{property_id}/{media_id}», де property_id – ідентифікатор об'єкта нерухомості, media_id – унікальний ідентифікатор медіа-файлу.

Для можливості інтеграції об’єктного сховища з іншими компонентами системи буде розроблено сервіс-клієнт, що відповідатиме за взаємодію із сховищем. Цей сервіс надаватиме уніфікований інтерфейс для виконання операцій з файлами, абстрагуючи деталі взаємодії з конкретною реалізацією об’єктного сховища. Такий підхід забезпечить можливість зміни постачальника послуг об’єктного сховища без необхідності модифікації інших компонентів системи.

Таким чином, використання об’єктного сховища з організацією даних у вигляді окремих бакетів для різних типів файлів забезпечить надійне та безпечне збереження файлових даних у системі управління орендою житла. Цей підхід відповідає вимогам до масштабованості, надійності та продуктивності, створюючи міцний фундамент для функціонування системи в умовах зростаючих обсягів даних та кількості користувачів.

2.6 Висновки

Отже, було проведено аналіз вхідних даних програмної системи управління орендою житла та визначено їх структуру й характеристики. Розроблена ER-модель відображає систему взаємопов'язаних сутностей, що забезпечують функціонування п'яти ключових модулів: керування об'єктами

оренди, комунікації, документообігу, обліку платежів і комунальних витрат та керування договорами. Для візуалізації структури архітектури застосунку було розроблено UML-діаграму компонентів, що демонструє структурні елементи системи та взаємозв'язки між ними.

Створено алгоритми критичних для системи процесів – автентифікації та авторизації користувачів з використанням JWT-токенів, а також збереження повідомлень в чаті для обміну миттєвими повідомленнями. Розроблений механізм автентифікації забезпечує належний рівень безпеки при доступі до системи з одночасним збереженням зручності користування через автоматизовані механізми оновлення сесій. Алгоритм збереження повідомлень чату розв’язує задачу надійної комунікації через використання миттєвих повідомлень.

Для аудиту дій користувачів проаналізовано різні підходи та обґрунтовано вибір механізму тригерів бази даних. Розроблено метод збереження файлів з використанням об’єктного сховища та розділенням даних на окремі бакети для документів та фотоматеріалів об'єктів нерухомості.

3 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ УПРАВЛІННЯ ОРЕНДОЮ ЖИТЛА

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення для управління орендою житла

Розробка системи для управління орендою житла потребує ретельного підходу до вибору технологічних засобів, які забезпечать надійність, масштабованість та зручність у використанні. Процес вибору технологій для реалізації програмного забезпечення ґрунтувався на аналізі функціональних вимог до системи, оцінці переваг та обмежень різних технологічних стеків, а також врахуванні сучасних тенденцій у розробці програмного забезпечення.

Для реалізації серверної частини системи було обрано Java в поєднанні з фреймворком Spring Boot [18]. Цей вибір обумовлений високою надійністю та стабільністю Java-платформи [19], що є критичним для системи, яка працює з фінансовими даними та документацією. Набір фреймворків Spring є потужною екосистемою для розробки корпоративних застосунків, надаючі готові модулі для автентифікації, обробки HTTP-запитів, роботи з базами даних та інших аспектів, необхідних для розробки програмного забезпечення [20]. Також важливим аргументом на користь цього стеку стала наявність спеціалізованих модулів Spring Security [21] для реалізації механізмів захисту та наявність у Spring Web MVC [22] засобів для роботи з WebSocket для забезпечення комунікації в реальному часі.

Клієнтська частина розробляється з використанням TypeScript [23] та фреймворку Angular [24]. TypeScript забезпечує строгу типізацію, що зменшує кількість помилок на етапі розробки та полегшує подальшу підтримку коду. Angular, у свою чергу, надає методи для розробки модульних та масштабованих клієнтських застосунків. Для розробки користувацького інтерфейсу використовується набір компонентів Angular Material [25], за допомогою якого забезпечується цілісний дизайн елементів інтерфейсу і зменшується обсяг часу, необхідного для розробки, через використання готових елементів інтерфейсу.

Для забезпечення надійного зберігання даних застосовано гібридний підхід, що поєднує різні типи сховищ. PostgreSQL обрано для зберігання структурованих даних завдяки її можливостям з підтримки ACID-транзакцій та високому рівню надійності зберігання даних [26]. Це робить PostgreSQL доцільним вибором для модулів системи, які оперують даними з високими вимогами до цілісності та послідовності: документообіг, облік платежів і комунальних послуг та керування об'єктами оренди.

MongoDB використано для функціонування частини модуля комунікацій – чату для обміну миттєвими повідомленнями, де критичними вимогами є висока продуктивність при інтенсивному обміні повідомленнями та гнучкість схеми даних. Документоорієнтована природа MongoDB підходить для зберігання повідомлень, а вбудовані механізми шардингу забезпечуватимуть горизонтальне масштабування при зростанні навантаження [27].

Для зберігання файлів та документів було обрано MinIO як сховище об'єктів. MinIO надає S3-сумісний API, що дозволяє абстрагуватися від конкретної реалізації сховища та спрощує можливий перехід на хмарні середовища розгортання у майбутньому. Ключовими перевагами MinIO є легкість локальної розробки, висока продуктивність та можливість горизонтального масштабування для роботи з великими обсягами даних [28]. Використання MinIO забезпечує зберігання та швидкий доступ до файлів різних типів, що особливо важливо для модуля документообігу, де зберігаються договори оренди та інші документи.

Для забезпечення надійної асинхронної обробки повідомлень чату обрано Apache Kafka [29]. Цей інструмент виступає як проміжний шар між компонентами обробки WebSocket та MongoDB, гарантуючи збереження повідомлень навіть при тимчасових збоях окремих компонентів системи. Можливості Kafka з партиціонування та реплікації даних забезпечують високу продуктивність та надійність системи обміну повідомленнями, що є критичним для користувацького досвіду в модулі комунікації.

Важливим аспектом розробки є вибір середовища розробки. Після порівняльного аналізу IntelliJ IDEA [30], Visual Studio Code [31] та Eclipse [32] було обрано IntelliJ IDEA як основне середовище розробки. IntelliJ IDEA вирізняється потужними інструментами для роботи з Java та Spring, включаючи інтелектуальне автодоповнення коду, інтегровану підтримку для системи контролю версій Git, а також інтеграцію з інструментами збірки Maven та Gradle. Крім того, IntelliJ IDEA надає потужну підтримку для розробки клієнтської частини на TypeScript та Angular, що дозволяє працювати над повним стеком технологій в єдиному середовищі. На відміну від Eclipse, який має обмежені можливості для розробки Angular застосунків і вимагає більшої кількості плагінів для повноцінної роботи, та Visual Studio Code, який хоч і є легшим, але менш спеціалізованим для розробки на Java, IntelliJ IDEA забезпечує поєднання функціональності та зручності використання.

Для спрощення процесу розгортання та забезпечення стабільного середовища розробки, тестування та експлуатації системи використовуються технології контейнеризації Docker [33] та Docker Compose [34]. Docker дозволяє упакувати застосунок з усіма його залежностями в стандартизований модуль для розробки програмного забезпечення, а Docker Compose спрощує управління багатоконтейнерними застосунками. Це забезпечує ідентичність середовища розробки, тестування та виробництва, мінімізуючи ризики, пов'язані з різницею в конфігураціях.

Отже, вибір технологій для розробки системи управління орендою житла, що базується на Java зі Spring Boot, TypeScript з Angular, PostgreSQL, MongoDB, MinIO, Apache Kafka та контейнеризації з Docker, має важливе значення для створення надійної платформи. Поєднання цих інструментів надає комплексний підхід до розробки, що дозволяє створювати програмне забезпечення з високим рівнем масштабованості, безпеки, продуктивності та зручності використання для різних категорій користувачів. Використання IntelliJ IDEA як основного середовища розробки забезпечує підтримку для роботи з усіма обраними технологіями.

3.2 Розробка модуля документообігу

Модуль документообігу забезпечує управління юридичними документами, пов’язаними з орендними відносинами, такими як договори оренди, акти прийому-передачі та додаткові угоди. UML-діаграма компонентів цього модуля показана на рис. 3.1.

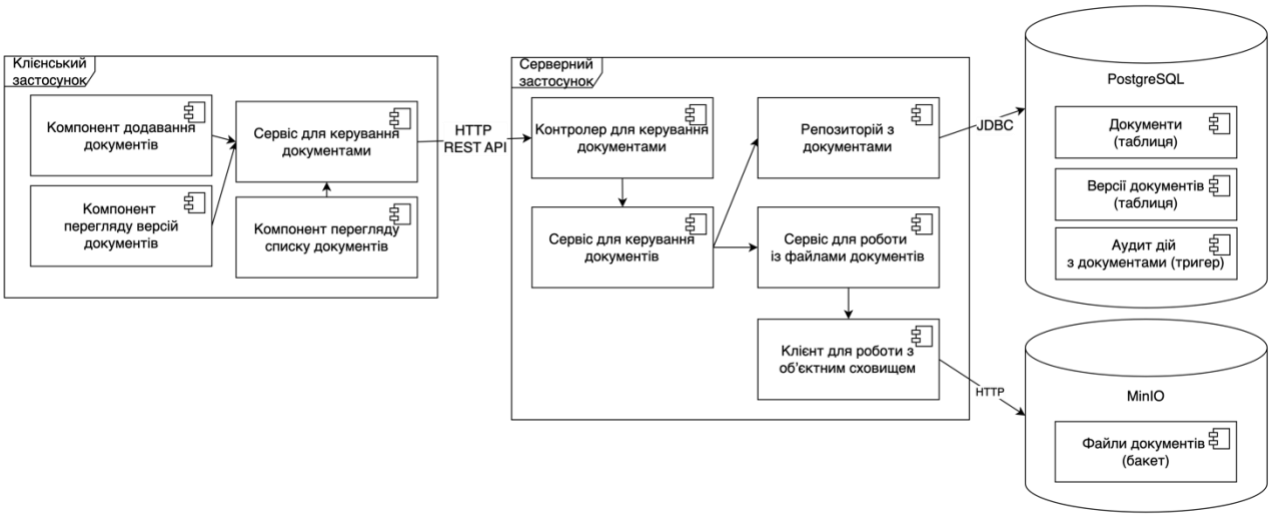

Рисунок 3.1 – UML-діаграма компонентів модуля документообігу

На клієнтській стороні модуль документообігу реалізується набором компонентів, які надають інтерфейс для створення та перегляду списку документів..

Серверна частина модуля документообігу відповідає за зберігання метаданих та файлів документів. Вона включає контролери для обробки HTTP-запитів, сервіси для реалізації бізнес-логіки обробки документів, репозиторії для взаємодії з базою даних. Особлива увага приділяється безпеці даних та захисту від несанкціонованого доступу до документів.

Для зберігання файлів документів впроваджено використання сховища MinIO з виділеним контейнером (бакетом) для файлів документів «documents». Така архітектура дозволяє відокремити метадані документів, які зберігаються в реляційній базі даних PostgreSQL, від власне файлів документів, які зберігаються в об'єктному сховищі. Взаємодія серверної частини з MinIO здійснюється через

S3-сумісний API, що забезпечує гнучкість у масштабуванні системи та можливість майбутньої міграції на хмарні сервіси без зміни програмного коду.

Система відображає всі доступні версії документа, що дозволяє відстежувати історію змін та бачити, які зміни були внесені на кожному етапі. Кожна версія документа містить інформацію про дату створення, користувача, який створив нову версію, та зберігається окремо для забезпечення повної аудиторської історії.

Взаємодія з базою даних реалізована через Spring Data JPA [35], що надає зручний інтерфейс для роботи з реляційними даними і використовує інтерфейс JDBC для взаємодії із PostgreSQL. Репозиторії модуля забезпечують операції збереження, пошуку та оновлення інформації про документи та їх версії.

3.3 Розробка модуля керування об’єктами оренди

Модуль керування об'єктами оренди є ключовим компонентом системи управління орендою житла, що забезпечує необхідний набір функцій для роботи з даними про нерухомість, їх характеристиками та мультимедійними матеріалами. Цей модуль реалізує повний цикл управління об'єктами від створення та редагування до пошуку та виведення детальної інформації. UML-діаграма компонентів цього модуля показана на рис. 3.2.

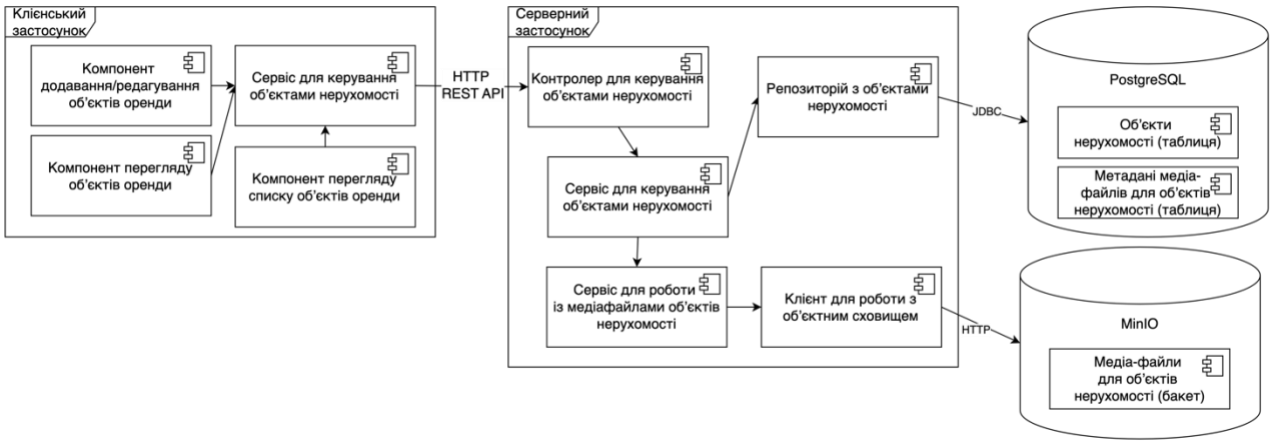

Рисунок 3.2 – UML-діаграма компонентів модуля керування об’єктами оренди

На клієнтській стороні модуль реалізується набором компонентів, які надають інтерактивний інтерфейс для взаємодії користувача з даними про об'єкти нерухомості. Він надає можливості додавання нових об'єктів, редагування існуючих, завантаження фотографій та інших медіа-файлів, а також перегляду детальної інформації. Інтерфейс реалізовано з використанням Angular.

Серверна частина модуля відповідає за бізнес-логіку та взаємодію з базою даних. Вона включає контролери для обробки HTTP-запитів, сервіси для реалізації бізнес-логіки та репозиторії для взаємодії з СКБД PostgreSQL. Особлива увага приділяється зберіганню та обробці мультимедійних даних, які оброблятимуться сховищем MinIO у спеціально виділеному контейнері (бакеті) «property-media».

Використання окремого сховища для зберігання медіа-файлів має переваги порівняно з традиційним підходом зберігання в базі даних. Воно дозволяє зменшити навантаження на базу даних, вилучивши з неї великі бінарні об'єкти. Також, S3-сумісний API MinIO забезпечує гнучкість у масштабуванні та можливість майбутньої міграції на хмарні сервіси без зміни програмного коду.

У базі даних PostgreSQL зберігаються лише метадані об'єктів нерухомості та посилання на медіа-файли в об’єктному сховищі, що зменшує навантаження на PostgreSQL при роботі з даними. Для організації зв'язків між об'єктами нерухомості та їх медіа-файлами використовується модель «один-до-багатьох», що дозволяє асоціювати з кожним об'єктом необмежену кількість зображень та інших медіа-матеріалів.

Взаємодія з базою даних реалізована через Spring Data JPA, який надає зручний інтерфейс для роботи з реляційними даними та дозволяє управляти зв'язками між сутностями. Репозиторії модуля забезпечують операції збереження, пошуку та оновлення інформації про об'єкти нерухомості та пов'язані з ними мультимедійні дані.

Взаємодія між клієнтською та серверною частинами здійснюється через HTTP REST API, що забезпечує стандартизований інтерфейс для виконання операцій створення, читання, оновлення та видалення над об'єктами

нерухомості. Серверна частина використовує з'єднання JDBC для взаємодії з базою даних PostgreSQL та S3-сумісний клієнт для комунікації з об’єктним сховищем MinIO.

3.4 Розробка модуля комунікації

Модуль комунікації надає можливість користувачам системи комунікувати через два основні механізми: чат для обміну миттєвими повідомленнями в реальному часі та структуровані звернення для більш формалізованої комунікації. UML-діаграма компонентів цього модуля показана на рис. 3.3.

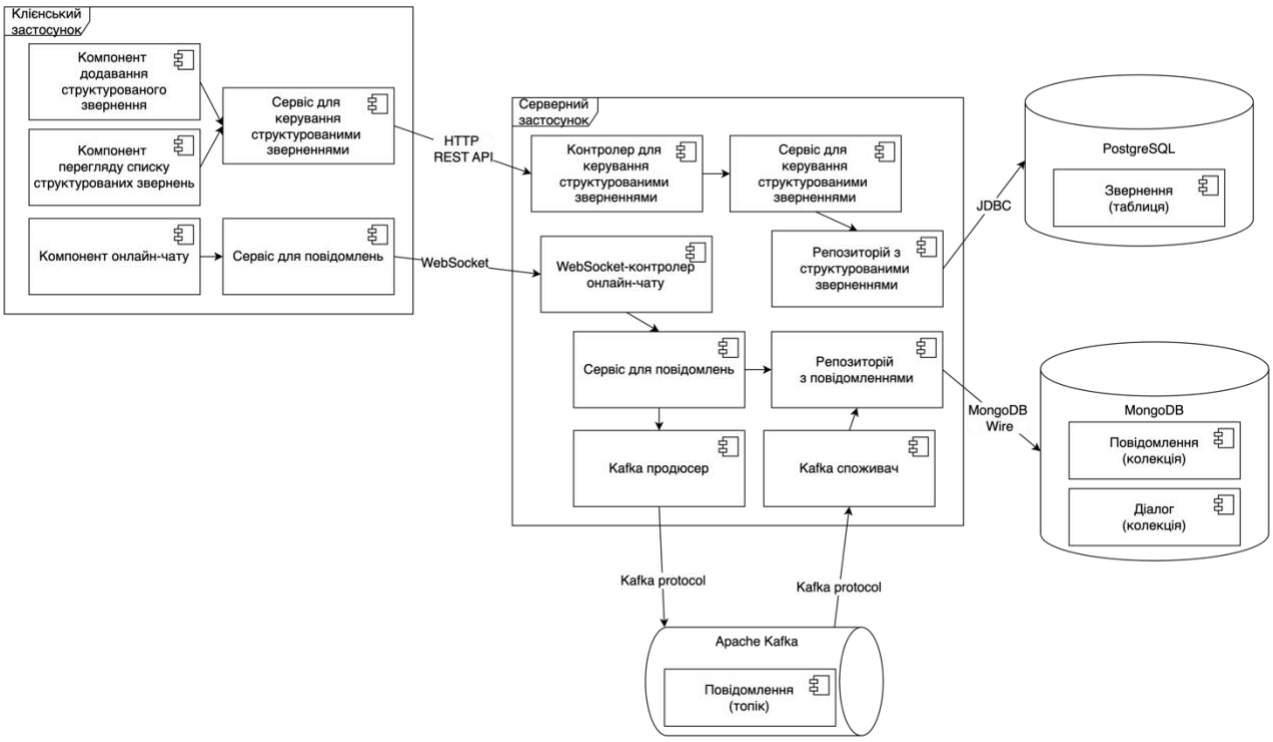

Рисунок 3.3 – UML-діаграма компонентів модуля комунікації

На клієнтській стороні модуль комунікації складається з двох основними частин: компонентом для чату, що надає інтерфейс для обміну миттєвими повідомленнями, та набором компонентів для керування структурованими зверненнями, який дозволяє створювати, переглядати та оновлювати формалізовані запити. Клієнтська частина чату використовує технологію

WebSocket для обміну даними в реальному часі, в той час як компонент структурованих звернень взаємодіє з сервером через REST API.

Серверна частина модуля комунікації реалізує обидва типи взаємодії користувачів. Для обміну миттєвими повідомленнями використовується Spring WebSocket, а для асинхронної обробки та гарантованої доставки повідомлень застосовується Apache Kafka.

Повідомлення чату зберігаються в MongoDB для забезпечення швидкого доступу до історії спілкування. Структуровані звернення, що мають чітку структуру та потребують надійності зберігання через використання механізмів транзакцій, зберігаються в PostgreSQL.

Для взаємодії з MongoDB використовується Spring Data MongoDB [36], що надає зручний доступ до документоорієнтованої бази даних через репозиторії. Аналогічно, для роботи з PostgreSQL застосовується Spring Data JPA.

Архітектурною особливістю модуля комунікації є використання асинхронних механізмів обробки повідомлень, що забезпечує високу надійність та масштабованість системи. Apache Kafka виступає як проміжний шар між WebSocket та MongoDB, гарантуючи збереження повідомлень навіть при тимчасових збоях окремих компонентів системи. Для роботи з Kafka використовується бібліотека Spring for Apache Kafka [37], що надає зручні абстракції для взаємодії з системою обміну повідомленнями та спрощує процес конфігурації продюсерів та споживачів.

3.5 Розробка модуля обліку платежів і комунальних послуг

Модуль обліку платежів і комунальних послуг надає необхідний функціонал для управління фінансовими аспектами орендних відносин, такими як облік орендних платежів та розрахунок комунальних послуг. Цей модуль є необхідним для забезпечення прозорості фінансових відносин між орендодавцями та орендарями. UML-діаграма компонентів цього модуля показана на рис. 3.4.

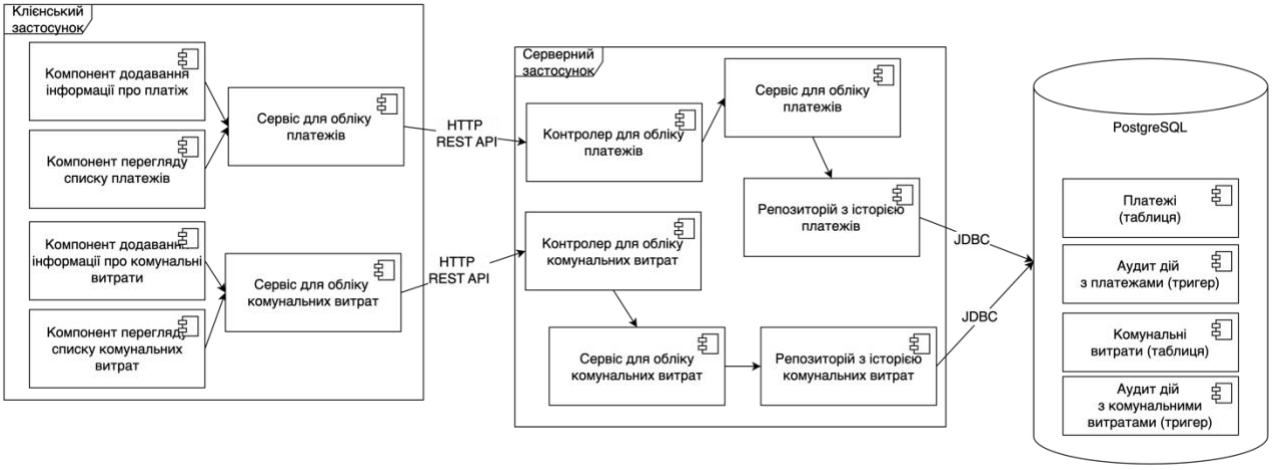

Рисунок 3.4 – UML-діаграма компонентів модуля обліку платежів і комунальних послуг

На клієнтській стороні модуль обліку платежів і комунальних послуг складається з набору компонентів, які надають інтерфейс для внесення інформації про платежі, перегляду історії транзакцій, ведення обліку використаних комунальних послуг.

Серверна частина цього модуля відповідає за бізнес-логіку фінансових операцій та забезпечує високий рівень цілісності та надійності даних. Вона включає контролери для обробки HTTP-запитів, сервіси для реалізації складних фінансових розрахунків, а також репозиторії для взаємодії з базою даних.

Взаємодія з базою даних реалізована також через Spring Data JPA. Застосунок взаємодіє У базі даних зберігаються дві основні таблиці: таблиця платежів, яка містить інформацію про всі фінансові транзакції, та набір таблиць для збереження даних про використані комунальні послуги.

3.6 Розробка модуля керування договорами

Модуль керування договорами є центральним інтеграційним компонентом системи управління орендою житла, що забезпечує зв’язок між окремими функціональними модулями та об’єднує їх навколо договірних відносин між учасниками процесу оренди. UML-діаграма компонентів показана на рис. 3.5.

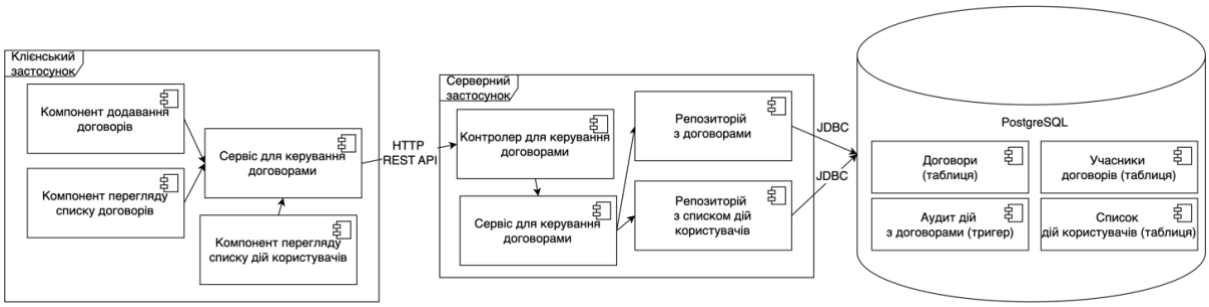

Рисунок 3.5 – UML-діаграма компонентів модуля керування договорами

На клієнтській стороні модуль керування договорами складається з набору компонентів, які надають інтерфейс для створення, редагування, перегляду та управління договорами оренди. Компонент додавання/редагування договорів, який дозволяє створювати нові договорів або редагувати існуючі. Ключовим у цьому модулі є компонент перегляду договорів, який надає інтегрує дані з інших модулів, такі як пов’язані документи, платежі, комунальні послуги.

Особливістю модуля керування договорами є його інтеграційна функція. Він встановлює та підтримує зв’язки між даними різних модулів, об’єднуючи їх навколо договірних відносин. Для реалізації цієї функції модуль використовує механізми зовнішніх ключів реляційної моделі даних, що дозволяє встановлювати та підтримувати стійкі зв’язки між сутностями різних модулів.

Важливим аспектом системи є реалізація механізму аудиту операцій користувачів через використання тригерів PostgreSQL у модулях обліку платежів і комунальних послуг та модулі документообігу. Розроблені тригери автоматично реєструють усі операції створення, оновлення та видалення записів у відповідних таблицях, записуючи детальну інформацію про зміни до спеціалізованої таблиці аудиту. Ця таблиця містить відомості про ідентифікатор користувача, тип операції, часову мітку та модифіковані дані, що забезпечує трасованість дій користувачів у системі. Дані з таблиці аудиту використовуються для формування журналу активності користувачів, що доступний у вебзастосунку, через компонент перегляду списку дій користувачів.

Взаємодія між клієнтською та серверною частинами модуля здійснюється через REST API, що забезпечує стандартизований інтерфейс для виконання

операцій над договорами. Серверна частина використовує з’єднання JDBC для взаємодії з базою даних PostgreSQL.

3.7 Розробка вебінтерфейсу програмного застосунку

Вебінтерфейс програмного застосунку для управління орендою житла розроблено з використанням фреймворку Angular та бібліотеки компонентів Angular Material.

Початковою точкою взаємодії користувача з системою є сторінка входу, яка відображає містить форму автентифікації з полями для введення логіну та паролю, а також кнопки «Увійти» та «Реєстрація» (рис. 3.6).

Рисунок 3.6 – Форма для авторизації

Після успішної автентифікації користувач потрапляє до головного інтерфейсу системи, який побудовано за принципом односторінкового застосунку (SPA) [38] з використанням Angular Router [24]. Навігаційна панель розташована у лівій частині екрану та містить основні пункти меню для переходу між основними частинами системи: «Звернення», «Чати», «Моя нерухомість» та «Мої оренди» (рис. 3.7). Кожен пункт меню супроводжується відповідною іконкою для полегшення візуальної ідентифікації. Активний пункт меню виділяється кольором та спеціальним маркером, що покращує навігаційний досвід користувача.

Рисунок 3.7 – Елементи бокового меню застосунку

Архітектурно навігаційна панель реалізована як окремий компонент, що взаємодіє з системою маршрутизації Angular Router. При виборі певного пункту меню відбувається зміна маршруту, що призводить до відображення відповідного компоненту в області контенту через директиву router-outlet. Це забезпечує модульність та незалежність компонентів інтерфейсу, а також покращує продуктивність застосунку, оскільки завантажуються лише ті компоненти, які необхідні для відображення поточного маршруту [24].

Пункт меню «Звернення» дозволяє перейти до інтерфейсу для роботи зі структурованими зверненнями користувачів. Основна сторінка модуля реалізована як компонент лістингу, що відображає список звернень з можливістю фільтрації, який показано на рис. 3.8. Верхня частина компонента містить рядок фільтрів, які дозволяють фільтрувати звернення за різними параметрами, такими як статус, пріоритет чи категорія.

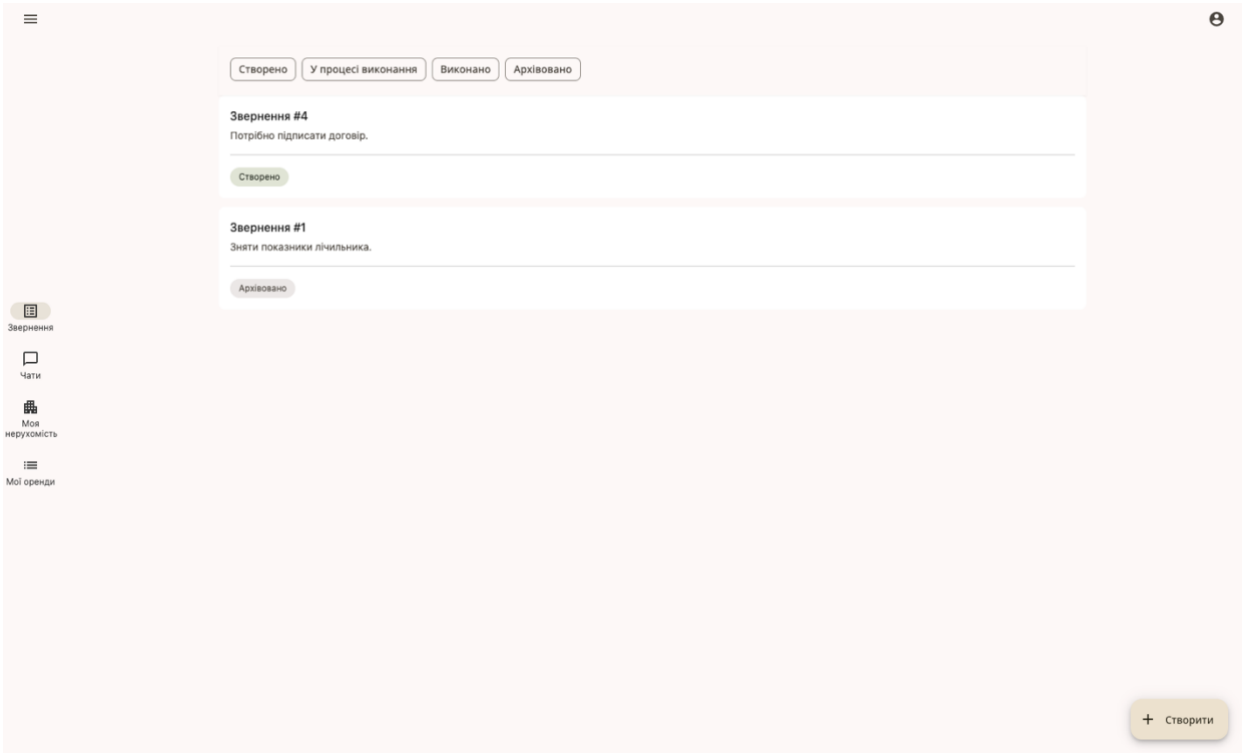

Рисунок 3.8 – Сторінка із списком звернень

Для забезпечення гнучкості відображення різних типів списків система використовує параметризовані ng-template, які передаються як вхідні параметри

до компоненту лістингу. Це дозволяє використовувати один і той самий компонент для відображення різних типів даних, змінюючи лише шаблон відображення елементів списку. Такий підхід спрощує підтримку та розширення функціональності системи, забезпечуючи єдиний інтерфейсний стиль для всіх списків.

Пункт меню «Чати» дозволяє перейти на сторінку, показану на рис. 3.9. Вона надає можливість обмінюватися миттєвими повідомленнями між користувачами системи, Сторінка розділена на дві основні частини: список діалогів у лівій частині та вікно активного діалогу справа. Список діалогів відображає співрозмовників з короткими відомостями про останнє повідомлення та час його надходження. Активний діалог вибирається зі списку та відображається у правій частині екрану, де користувач може переглядати історію повідомлень та надсилати нові повідомлення через поле введення в нижній частині.

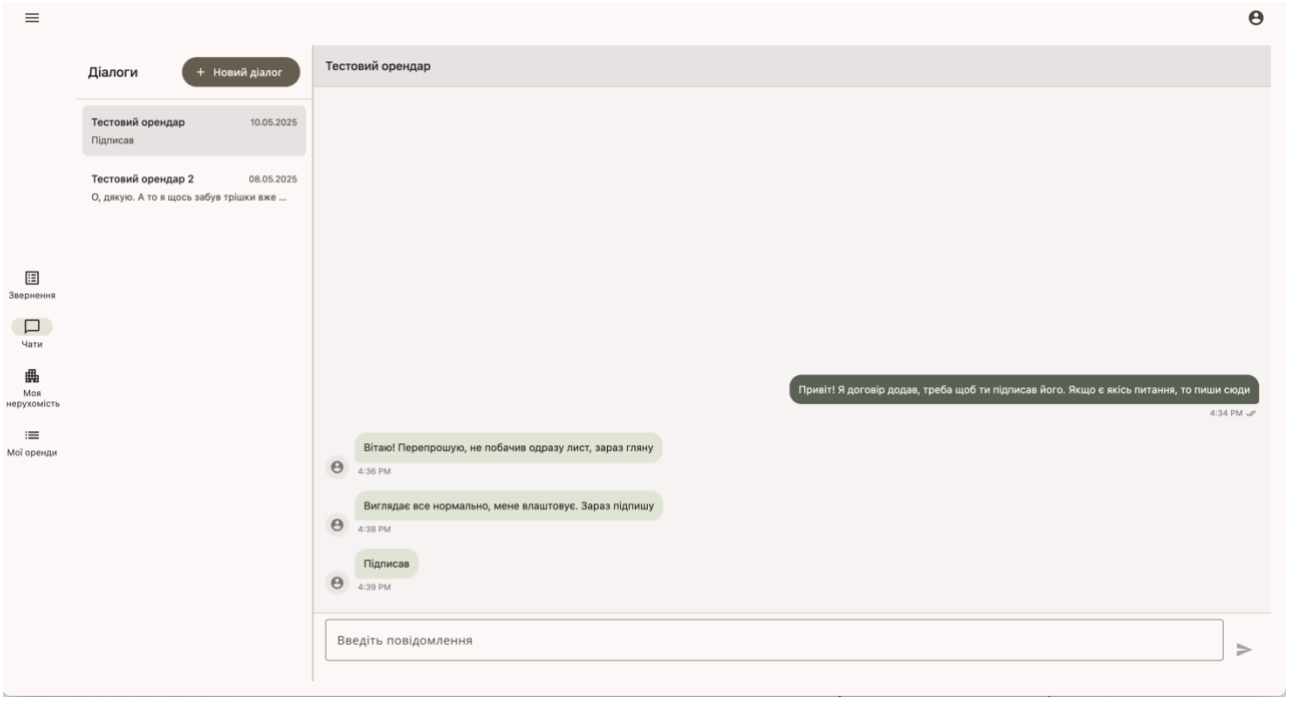

Рисунок 3.9 – Сторінка із елементами інтерфейсу чату

Пункт меню «Мої оренди» дозволяє перейти до інтерфейсу для управління орендними договорами та пов’язаними з ними процесами. Сторінка перегляду

орендного договору організована з використанням компонента з вкладками, що дозволяє структурувати різнотипну інформацію (рис. 3.10). У верхній частині сторінки відображається назва договору та адреса об’єкту нерухомості.

Ця сторінка містить три вкладки: «Документи», «Платежі» та «Комунальні послуги». Вкладка «Документи» містить список документів, пов’язаних з орендою, з можливістю їх перегляду та завантаження. Вкладка «Платежі» відображає історію платежів за орендою з детальною інформацією про кожен платіж, включаючи дату і суму. Вкладка «Комунальні послуги» надає інформацію про нарахування за комунальні послуги з можливістю перегляду історії показників лічильників чи розрахунків.

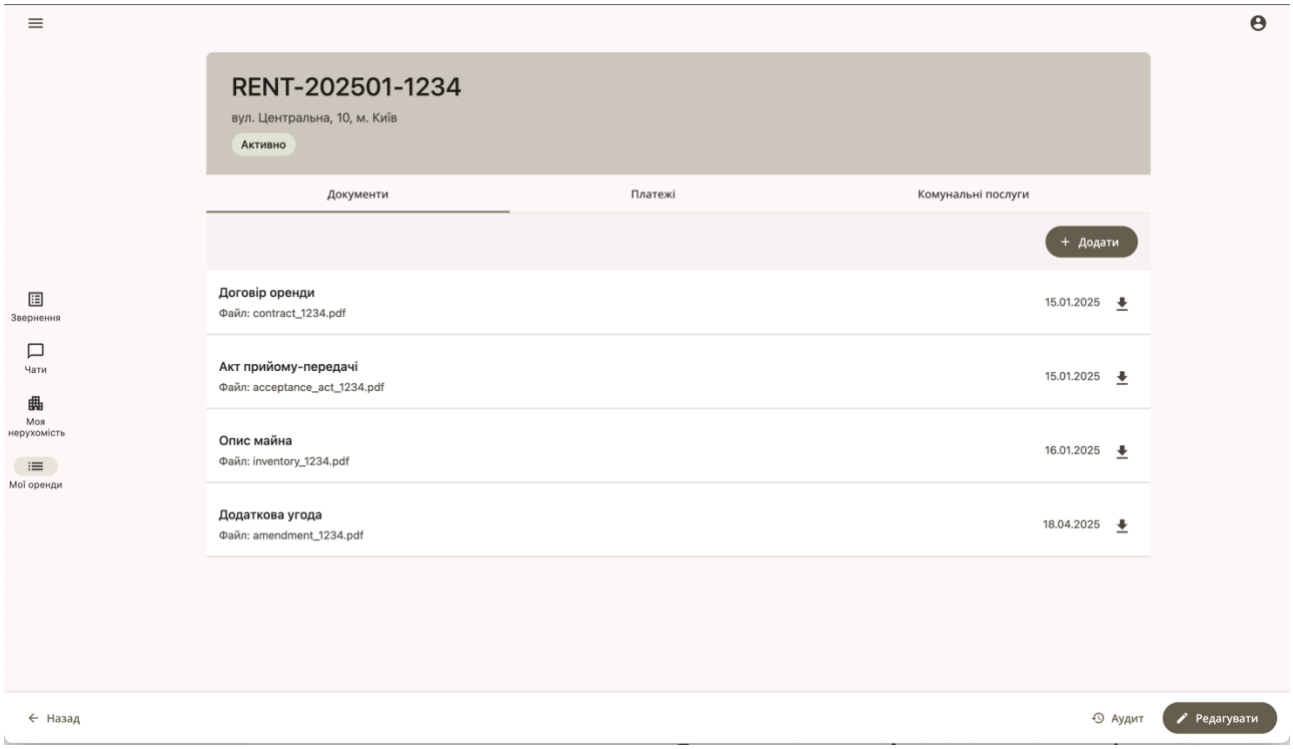

Рисунок 3.10 – Сторінка перегляду орендного договору

Отже, було детально описано розробку вебінтерфейсу програмного застосунку для управління орендою житла. Використання фреймворку Angular у поєднанні з бібліотекою компонентів Angular Material дозволило створити сучасний та адаптивний користувацький інтерфейс.

3.8 Висновки

Отже, було обґрунтовано вибір технологій та інструментів, що використовувались для розробки програмних модулів. Для реалізації серверної частини застосунку було обрано мову програмування Java і фреймворк Spring Boot, для клієнтської – мову TypeScript і фреймворк Angular. Для зручності розробки графічного інтерфейсу було використано бібліотеку елементів користувацького інтерфейсу Angular Material. Для зберігання та обробки даних було застосовано гібридний підхід з використанням PostgreSQL для структурованих даних з високими вимогами до цілісності та MongoDB для даних з гнучкою схемою, зокрема повідомлень чату.

Для зберігання файлових даних було впроваджено сховище MinIO з окремими бакетами «documents» для файлів документів та «property-media» для медіа-файлів об'єктів нерухомості, що дозволило зменшити навантаження на базу даних та забезпечити можливість масштабування у майбутньому. Асинхронна обробка повідомлень реалізована за допомогою Apache Kafka, що забезпечує надійність та масштабованість системи при високих навантаженнях.

У процесі вибору інтегрованого середовища розробки було проведено порівняльний аналіз IntelliJ IDEA, Visual Studio Code та Eclipse, в результаті якого обрано IntelliJ IDEA. Для спрощення процесу розгортання та забезпечення стабільного середовища використано технології контейнеризації Docker та Docker Compose.

У розділі детально описано розробку п'яти основних модулів системи: модуля керування об'єктами оренди, модуля комунікації, модуля документообігу, модуля обліку платежів і комунальних послуг та модуля керування договорами. Для кожного модуля наведено діаграми компонентів, що ілюструють взаємодію між клієнтською та серверною частинами, та описано основні особливості використаних інструментів.

Розроблений вебінтерфейс програмного застосунку реалізовано за принципом односторінкового застосунку (SPA) з використанням Angular Router та бібліотеки компонентів Angular Material. Використання засобів фреймворку

Angular, таких як параметризація компонентів через передавання ng-template із описом специфічної функціональності, спрощує підтримку і розширення функцій системи, а застосування Angular Material дозволяє уніфікувати стиль інтерфейсу користувача для всіх модулів.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ УПРАВЛІННЯ ОРЕНДОЮ ЖИТЛА

4.1 Аналіз методів тестування програмних продуктів

Тестування програмного забезпечення є невід'ємним етапом життєвого циклу розробки, що має на меті ідентифікацію невідповідностей між реальною поведінкою програмного продукту та його очікуваною поведінкою відповідно до вимог. Сучасні методології визначають тестування як процес виконання програми з метою виявлення помилок та підтвердження коректності функціонування системи відповідно до специфікацій [29].

У процесі розробки програмних систем виділяють різні методи тестування за рівнем доступу до внутрішньої структури програми. Основними з них є тестування за принципами «чорної скриньки», «білої скриньки» та «сірої скриньки».

Тестування за принципом «чорної скриньки» полягає у аналізі програмного забезпечення як закритої системи, внутрішня структура якої невідома або ігнорується. Тестування здійснюється через визначені інтерфейси програми та зосереджується на вхідних та вихідних даних без аналізу внутрішнього функціонування. Даний метод дозволяє оцінити програмний продукт з позиції кінцевого користувача, перевіряючи відповідність функціональності заявленим вимогам. Тестування «чорної скриньки» доцільне для виявлення помилок у вимогах, інтерфейсній взаємодії та функціональних аспектах системи.

Тестування за принципом «білої скриньки» базується на глибокому аналізі внутрішньої структури програми, включаючи код, архітектуру, потоки даних та алгоритми. Цей підхід дозволяє виявити помилки в алгоритмічній реалізації, ділянки коду з низькою продуктивністю, витоки пам'яті та інші низькорівневі дефекти. Тестування «білої скриньки» вимагає глибоких технічних знань і часто здійснюється самими розробниками або спеціалізованими тестувальниками з

навичками програмування. Методи тестування «білої скриньки» включають перевірку покриття операторів, розгалужень, шляхів та умов.

Тестування за принципом «сірої скриньки» є гібридним підходом, що поєднує елементи «чорної» та «білої» скриньок. При цьому методі тестувальник має обмежені знання про внутрішню структуру програми, такі як архітектурні діаграми, бази даних та алгоритми, але не має повного доступу до вихідного коду. Цей підхід дозволяє розробляти тестові сценарії, які спрямовані на перевірку найбільш критичних або схильних до помилок компонентів системи, при цьому зберігаючи перспективу кінцевого користувача. У процесі розробки програмних систем також виділяють два основні підходи до тестування з точки зору автоматизації: автоматизоване та ручне. Автоматизоване тестування полягає у використанні спеціалізованих інструментів та скриптів для автоматичного виконання тестових сценаріїв, що дозволяє забезпечити високу швидкість та повторюваність тестування. Ручне тестування, натомість, виконується безпосередньо тестувальником та дозволяє більш глибоко дослідити поведінку системи з точки зору користувача, виявити недоліки у зручності використання та візуального відображення [30].

З метою оцінки системи з позиції кінцевого користувача було застосовано ручне тестування за принципом «чорної скриньки». Для перевірки критичних компонентів, таких як механізми автентифікації та обліку платежів, було також застосовано елементи підходу «сіра скринька», що дозволило забезпечити більш глибоку перевірку цих функціональних модулів.

Ручне тестування також надало можливість більш гнучко адаптувати тестові сценарії до особливостей предметної області та швидко реагувати на виявлені недоліки. Слід зазначити, що для подальшого розвитку системи доцільним є впровадження автоматизованого тестування з елементами «білої скриньки» для можливості регресійного тестування та підвищення якості коду.

При розробці стратегії тестування системи управління орендою житла враховано специфіку системи з багатьма взаємопов'язаними модулями, що оперують критично важливими даними. План тестування було розроблено з

акцентом на перевірку цілісності даних, коректності бізнес-процесів та захисту інформації.

4.2 Тестування програмного засобу для управління орендою житла

Процес тестування системи управління орендою житла проводився у декілька послідовних етапів відповідно до розробленого плану. Тестування розпочалося з перевірки базових функцій і завершилося інтеграційними сценаріями, що охоплюють взаємодію між різними модулями системи. Для всіх тестових випадків було визначено набори вхідних даних, очікувані результати та критерії оцінки у відповідності до методології тестування на основі специфікації.

Першим етапом стала перевірка функціональності реєстрації користувачів. Під час тестування було проведено серію спроб реєстрації з різними наборами даних для перевірки коректності роботи системи валідації. Тестування включало перевірку обов’язкових полів та валідацію формату введених даних. На рис. 4.1 продемонстровано інтерфейс форми реєстрації.

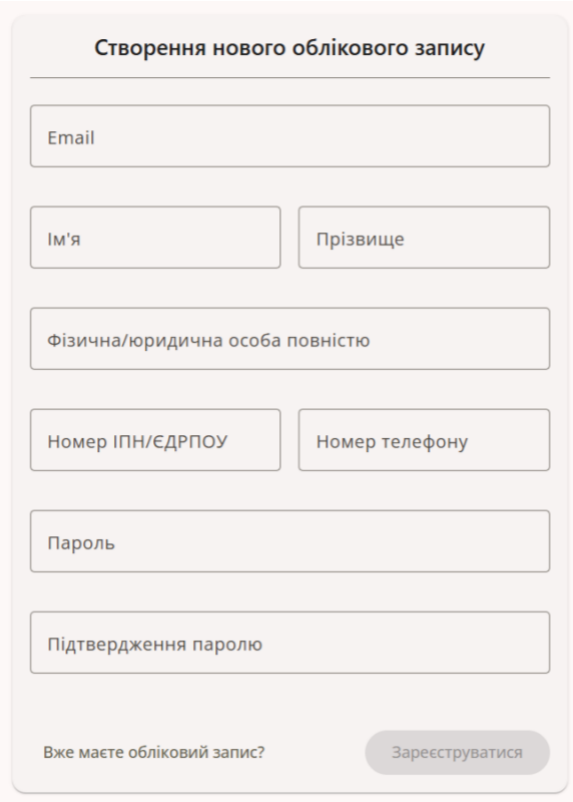

Рисунок 4.1 – Інтерфейс форми реєстрації нового користувача

Було підтверджено, що система коректно обробляє введені дані, перевіряє унікальність електронної адреси та відповідність паролю вимогам безпеки.

Другим етапом стала перевірка функціональності додавання об’єктів нерухомості. Було створено кілька тестових об’єктів з різними параметрами для визначення стабільності роботи системи. Тестування включало перевірку обов’язкових полів та можливість додавання фотографій. На рис. 4.2 показано інтерфейс форми додавання об’єкта нерухомості.

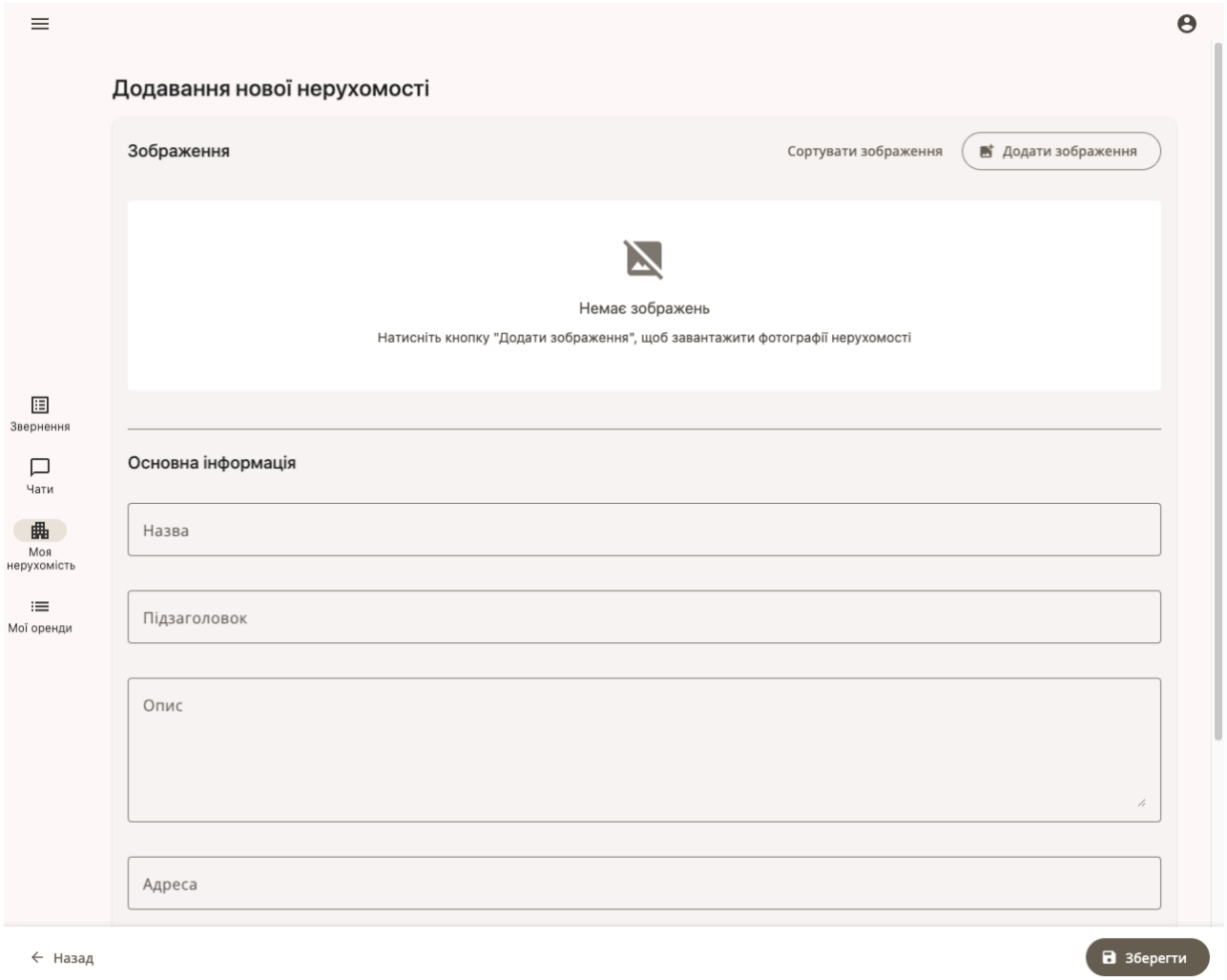

Рисунок 4.2 – Інтерфейс форми додавання нового об’єкта нерухомості

Аналіз результатів підтвердив коректну роботу системи при додаванні об’єктів з різними характеристиками. Система правильно зберігає та відображає інформацію, включаючи мультимедійні дані.

Третім етапом стала перевірка процесу створення договорів оренди. Для тестування використано раніше створений об’єкт нерухомості та акаунти користувачів з різними ролями. Форму для створення нового договору оренди показано на рис. 4.3.

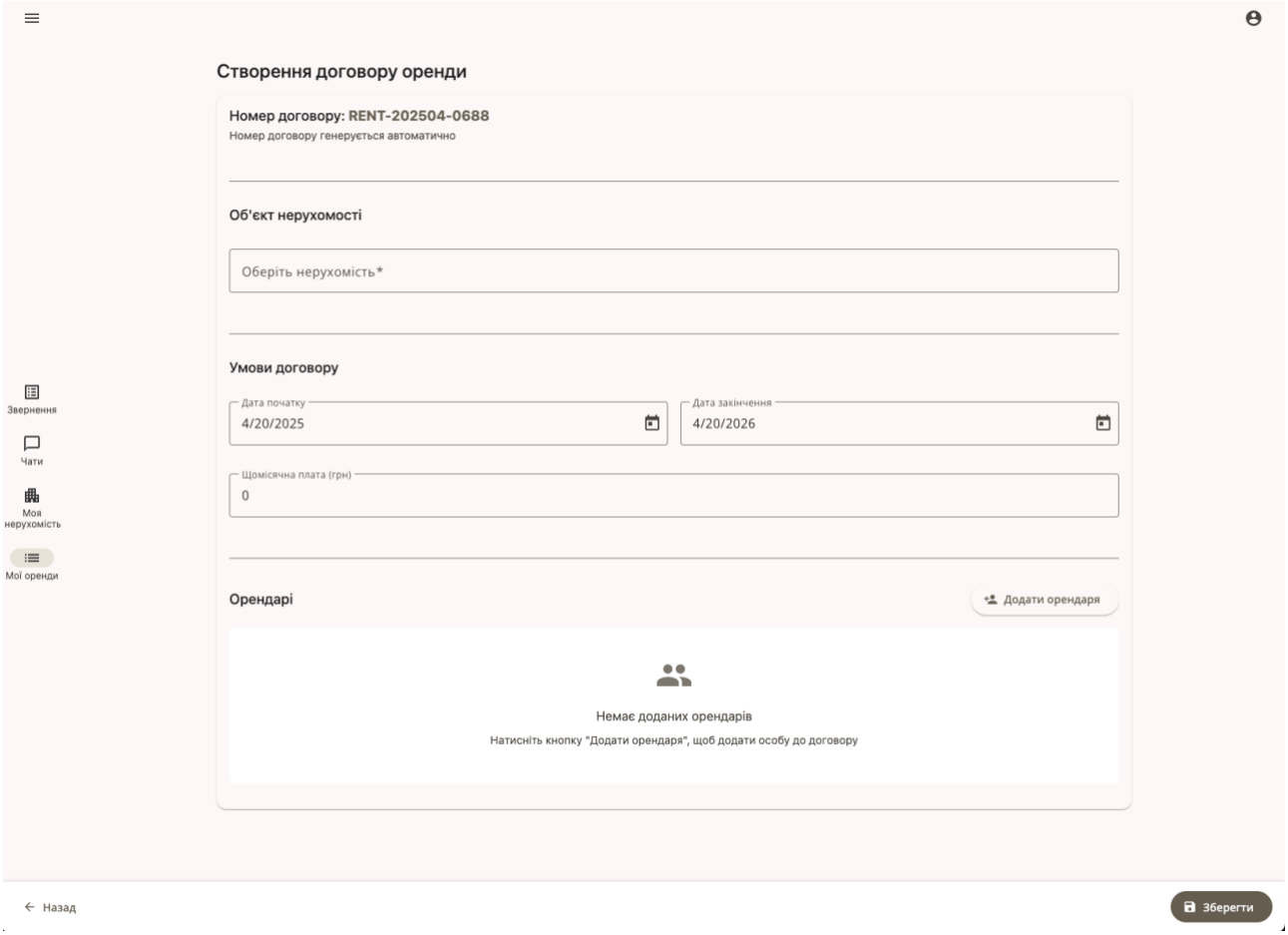

Рисунок 4.3 – Інтерфейс форми створення договору оренди

У результаті, було підтверджено, що система коректно зберігає дані про договори. Аналіз результатів підтвердив роботу системи валідації, яка запобігає створенню сутностей без заповнення обов’язкових полів. Система відображає зрозуміле повідомлення про помилку з поясненням причини відмови.

Четвертий етап був присвячений перевірці модуля структурованих звернень. Тестування включало створення звернень різних типів та перевірку процесу їх обробки. На рис. 4.4 продемонстровано інтерфейс створення структурованого звернення.

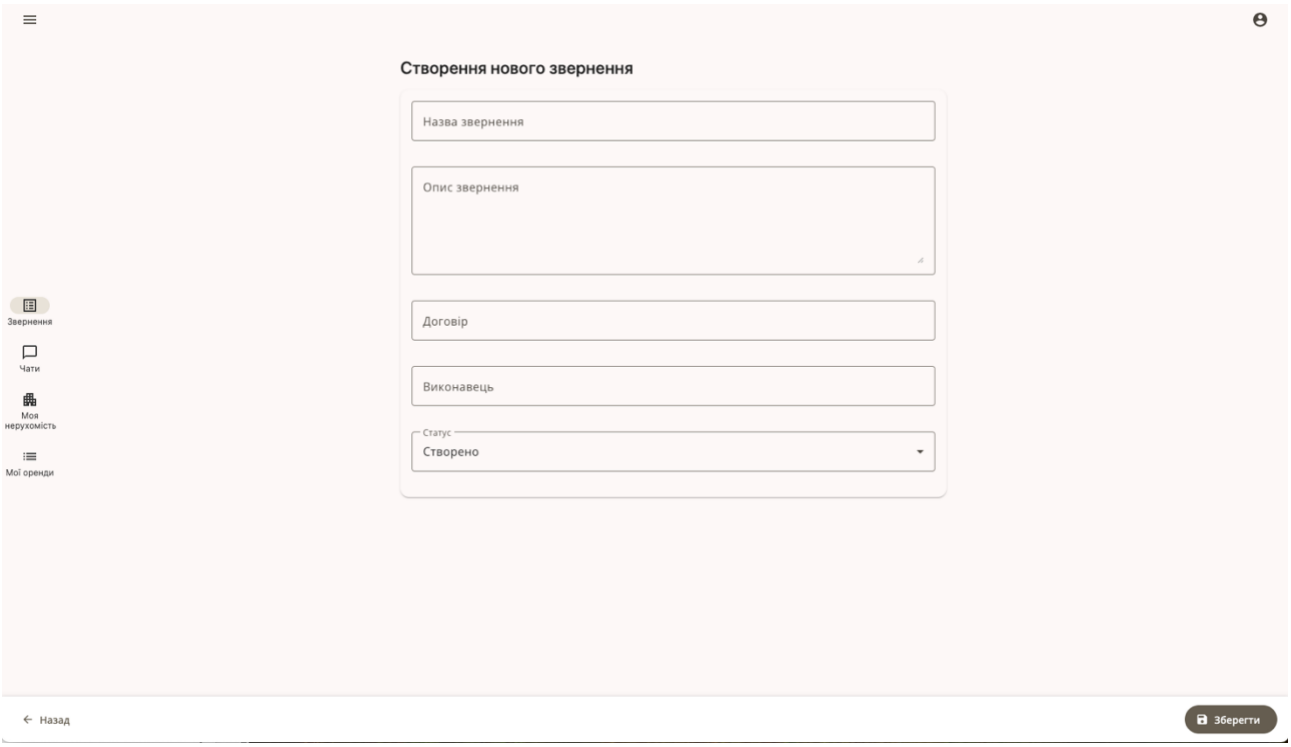

Рисунок 4.4 – Інтерфейс створення нового звернення

Результати підтвердили, що система здійснює валідацію полів форми і тоді зберігає дані про звернення.

П’ятим етапом стала перевірка функціональності фільтрації звернень. Інтерфейс для фільтрування списку показано рис. 4.5.

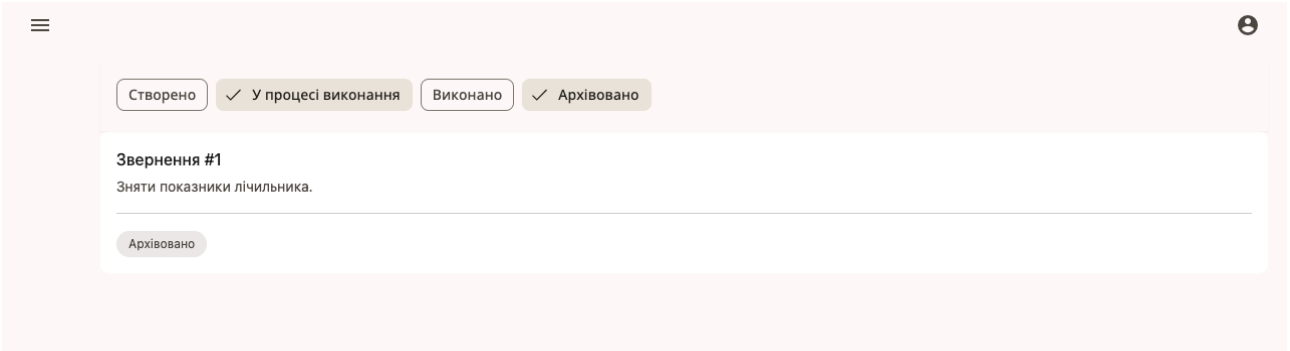

Рисунок 4.5 – Інтерфейс фільтрації звернень

Аналіз результатів підтвердив коректну роботу системи фільтрації. Система правильно відображає звернення за обраними критеріями та забезпечує швидкий доступ до інформації.

Шостим етапом стало тестування функціоналу сторінки керування договором оренди. Ця сторінка є ключовим інтеграційним компонентом системи, що об'єднує різні аспекти орендних відносин через систему вкладок. Тестування включало перевірку роботи трьох основних вкладок: «Документи», «Платежі» та «Комунальні витрати».

На вкладці «Документи» було протестовано можливість завантаження файлів формату PDF. Увага приділялася перевірці коректності створення нових версій документів після кожного завантаження нової версії документа. Система продемонструвала здатність зберігати всі версії документів та відображати їх у хронологічному порядку, що забезпечує повну історію змін.

На вкладці «Платежі» (рис. 4.6) тестувалося додавання нових платежів з різними параметрами: сума, дата, призначення платежу за допомогою модального вікна (рис. 4.7), яке з’являється після натиснення кнопки «Додати». Перевірялося також відображення історії оплат у хронологічному порядку.

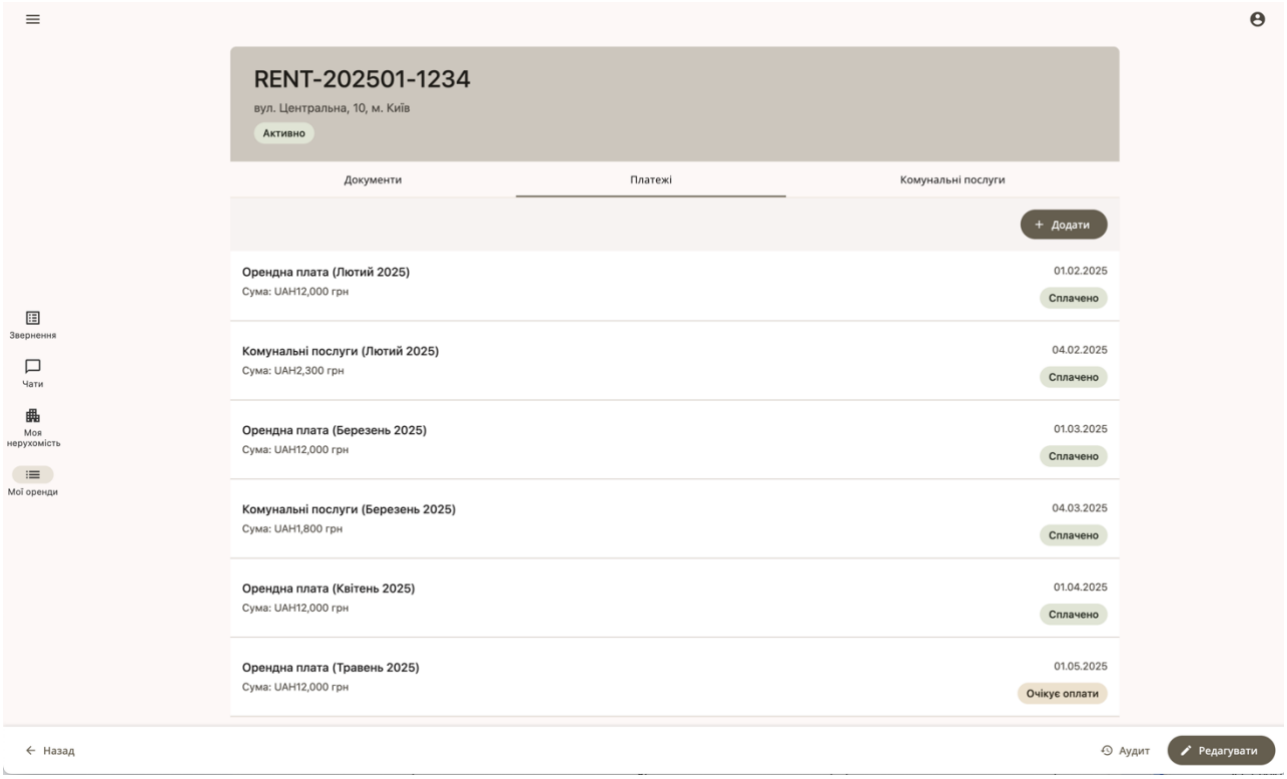

Рисунок 4.6 – Вкладка «Платежі»

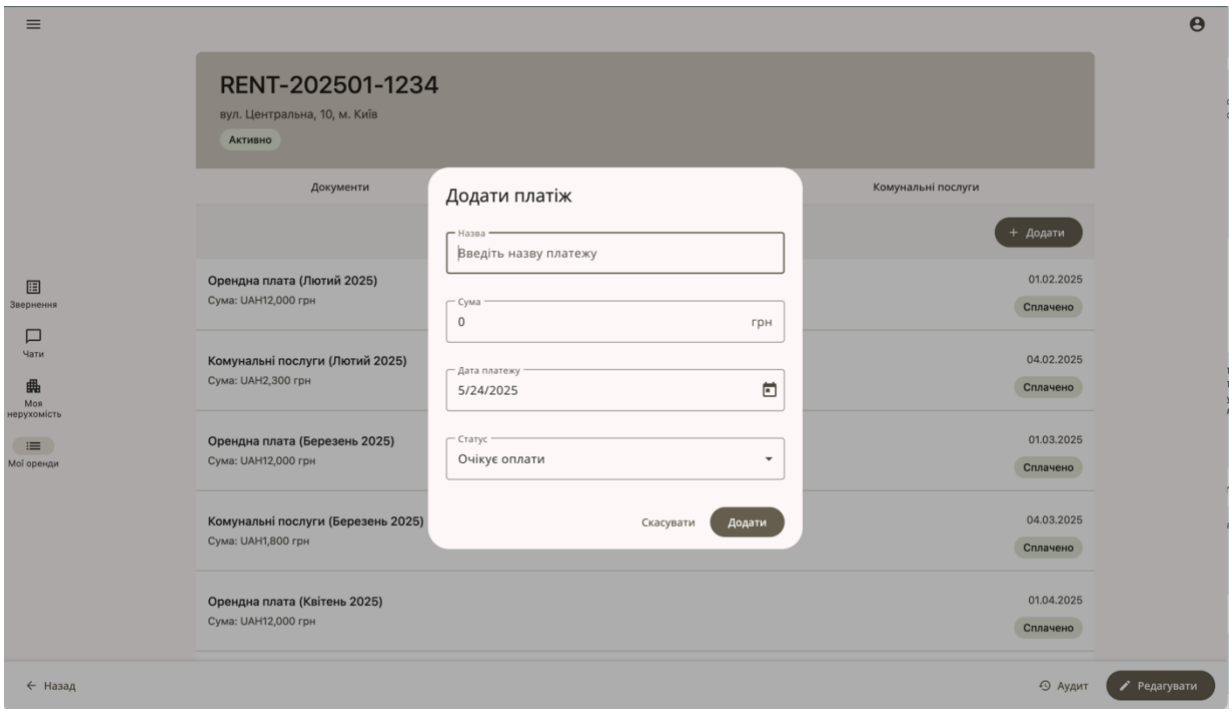

Рисунок 4.7 – Модальне вікно додавання платежів

На вкладці «Комунальні витрати» перевірялася можливість додавання витрат за різними типами комунальних послуг (електроенергія, вода, опалення тощо) з зазначенням дати, суми та показників лічильників за допомогою модального вікна (рис. 4.8). Система коректно зберігала внесені дані та відображала їх у структурованому вигляді.

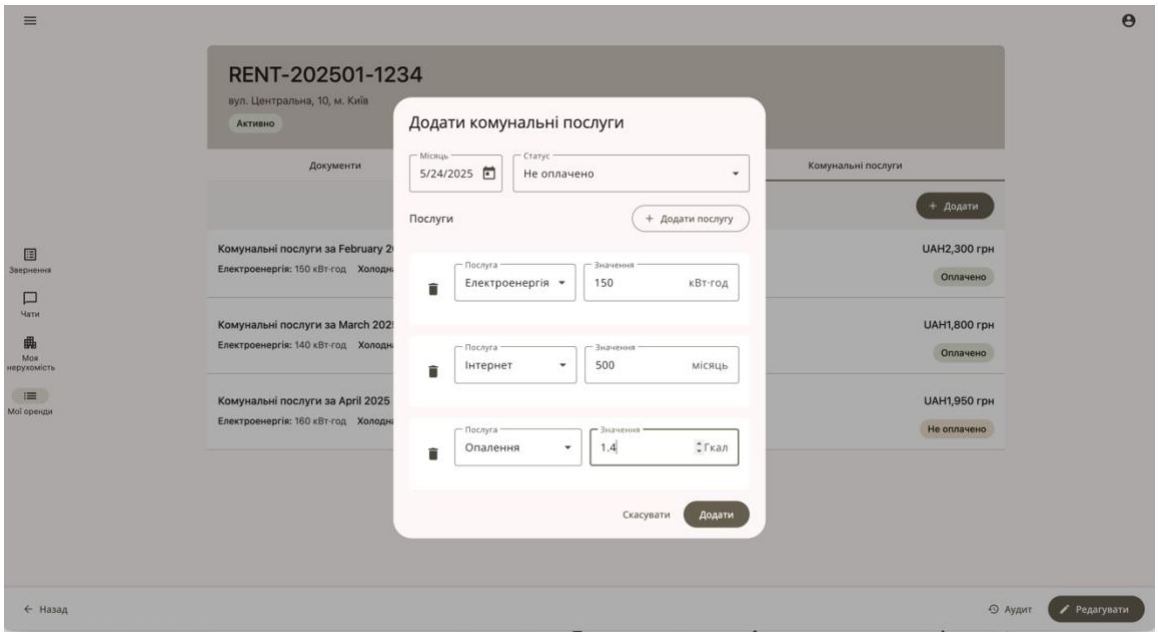

Рисунок 4.8 – Модальне вікно додавання комунальних витрат

Додатково було протестовано функціонал кнопки «Аудит», яка відкриває модальне вікно (рис. 4.9) зі списком усіх дій, здійснених користувачами із поточним договором. Система коректно реєструвала та відображала різні типи дій: додавання, редагування та видалення документів, платежів та комунальних витрат із зазначенням часу дії та користувача, який її виконав. Цей функціонал забезпечує повну прозорість усіх операцій та можливість відстеження історії змін, що особливо важливо для розв’язання потенційних спірних ситуацій між учасниками орендних відносин.

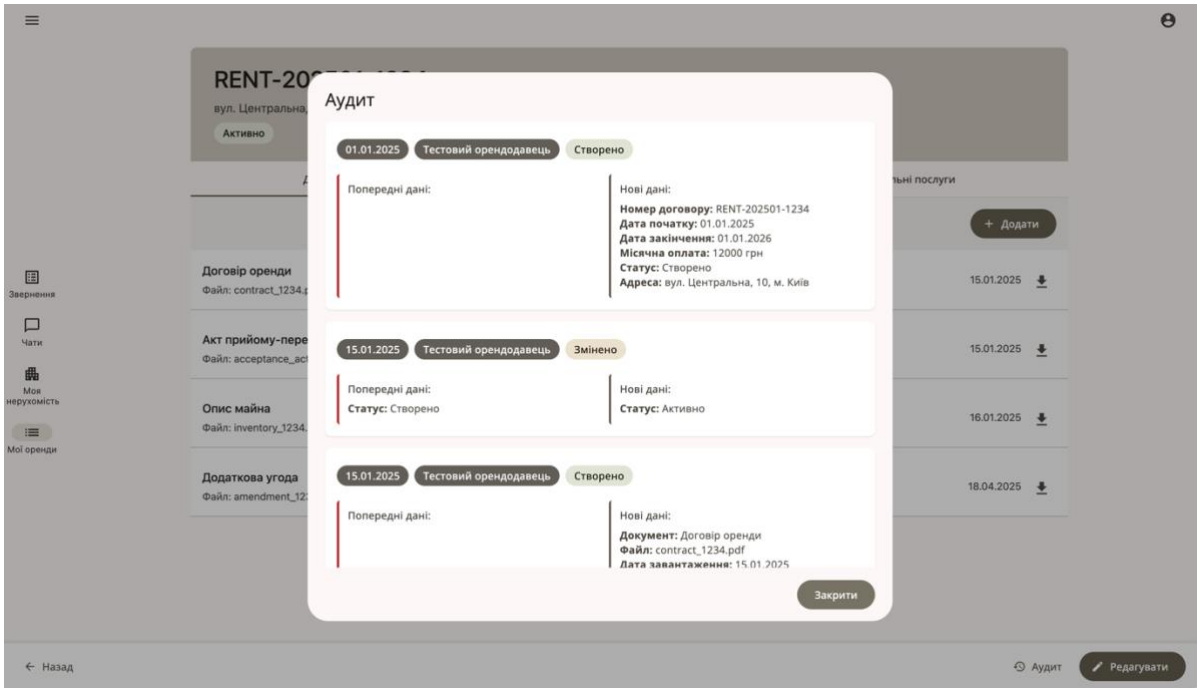

Рисунок 4.9 – Модальне вікно зі списком дій користувачів

Заключним етапом було тестування функціональності чату для обміну миттєвими повідомленням, який є одним із ключових елементів комунікації між користувачами системи. Під час тестування перевірялась коректність доставки повідомлень, формування історії листування, а також збереження структурованості даних при повторному входженні користувача в систему після виходу.

Результати тестування підтвердили надійність роботи системи чату та її відповідність функціональним вимогам. Повідомлення коректно доставляються

адресатам у режимі реального часу, при цьому зберігається логічна послідовність повідомлень.

Проведене ручне тестування системи управління орендою житла дозволило підтвердити її відповідність функціональним та нефункціональним вимогам. Використання методології ручного тестування надало можливість оцінити систему з позиції кінцевого користувача.

Тестування показало, що розроблена система успішно виконує всі покладені на неї функції. Усі виявлені в процесі тестування дефекти були належним чином задокументовані та усунені, що дозволило підвищити загальну якість та надійність системи.

За результатами тестування можна дійти висновку, що розроблений програмний засіб для управління орендою житла готовий до впровадження в реальних умовах експлуатації. Водночас, для подальшого розвитку системи рекомендується впровадження автоматизованого тестування.

4.3 Розробка інструкції користувача

Для належного функціонування програмної системи управління орендою житла для користувача необхідною є наявність комп’ютера з операційною системою Windows, macOS або Linux з встановленим сучасним веббраузером, що забезпечує повну підтримку HTML5, CSS3 та JavaScript (Google Chrome, Mozilla Firefox або Microsoft Edge останніх версій). Для комфортної роботи з системою рекомендовано використання монітора з роздільною здатністю не менше 1366×768 пікселів та стабільне інтернет-з’єднання.

Головний інтерфейс системи організовано за принципом односторінкового застосунку з боковим меню навігації, розташованим у лівій частині екрану. Меню навігації містить наступні пункти:

- «Звернення» – для доступу до списку звернень;
- «Чати» – для доступу до чату для обміну миттєвими повідомленнями;
- «Моя нерухомість» – для доступу до списку об’єктів нерухомості;
- «Мої оренди» – для доступу до списку договорів оренди.

Для переходу до відповідного розділу системи користувачу необхідно обрати відповідний пункт меню.

Сторінки зі списками звернень, об’єктів нерухомості та договорів оренди мають уніфіковану структуру, що включає:

- панель фільтрів у верхній частині сторінки для звуження вибірки відображуваних елементів;

- список елементів у центральній частині сторінки;

- кнопку додавання нового елемента у правому нижньому куті екрана.

Сторінки перегляду звернення, договору та об’єкта нерухомості відображають властивості цих сутностей у режимі «лише для читання» та містять панель у нижній частині екрана.

Для взаємодії з об’єктами користувач може виконати наступні дії:

1. Для повернення до списку об’єктів необхідно натиснути кнопку «Назад» у лівій частині панелі кнопок.

2. Для переходу до режиму редагування об’єкта необхідно натиснути кнопку «Редагувати» у правій частині панелі кнопок.

Сторінка перегляду договору також дозволяє оперувати пов’язаними сутностями і тому має власну структуру у вигляді вкладок «Документи», «Платежі», «Комунальні послуги». Кожна вкладка дозволяє переглядати та управляти відповідними сутностями. Для роботи з вкладками користувач може виконати наступні дії:

1. Для переходу між вкладками необхідно натиснути на відповідну назву вкладки.

2. Для додавання нового елемента на будь-якій вкладці необхідно натиснути кнопку «Додати» на панелі над списком, після чого заповнити поля у модальному вікні, що з'явиться, та натиснути кнопку «Зберегти».

3. Для перегляду детальної інформації про елемент на вкладці необхідно натиснути на відповідний елемент у списку.

На вкладці «Документи» користувач має можливість не лише додавати нові документи, але й додавати їх нові версії,. наприклад, після накладення КЕП

сторонніми сервісами. Система підтримує зберігання всіх версій документів, що дозволяє відстежувати повну історію змін. Користувач може переглядати всі попередні версії документа у хронологічному порядку через модальне вікно перегляду версій.

Інтерфейс чату розділено на дві основні частини: список діалогів у лівій частині та вікно активного діалогу у правій частині. Для роботи з чатом користувач може виконати наступні дії:

1. Для вибору діалогу необхідно натиснути на відповідний елемент у списку діалогів.

2. Для надсилання повідомлення необхідно ввести текст у поле введення в нижній частині вікна активного діалогу та натиснути кнопку «Надіслати» або клавішу Enter.

Історія повідомлень зберігається та доступна для перегляду при повторному відкритті діалогу.

4.4 Висновки

Отже, було здійснено тестування програмної системи управління орендою житла та розроблено інструкцію користувача. Проведено аналіз методів тестування програмних продуктів, який виявив доцільність застосування ручного тестування за принципом «чорної скриньки» з елементами «сірої скриньки» для критичних компонентів системи. Такий підхід дозволив оцінити систему з позиції кінцевого користувача та перевірити інтуїтивність інтерфейсу в реальних умовах використання.

Процес тестування охопив сім послідовних етапів: перевірку реєстрації користувачів, додавання об'єктів нерухомості, створення договорів оренди, роботу зі структурованими зверненнями, фільтрацію списків, тестування сторінки керування договором оренди з її основними вкладками («Документи», «Платежі», «Комунальні витрати») та функціонал аудиту, а також функціонування онлайн-чату. На вкладці «Документи» увагу було приділено тестуванню створення нових версій документів.

Результати тестування підтвердили, що система виконує усі покладені на неї функції. Система належним чином зберігає версії документів, коректно відображає історію змін через функціонал аудиту та забезпечує надійний обмін повідомленнями. Виявлені в процесі тестування дефекти були документовані та усунені, що підвищило загальну якість та надійність програмного засобу.

Розроблена інструкція користувача містить детальний опис роботи з системою, включаючи вимоги до технічного забезпечення, навігацію, роботу з основними модулями та управління договірними відносинами.

За результатами тестування можна зробити висновок, що розроблений програмний засіб для управління орендою житла готовий до впровадження в реальних умовах експлуатації. Водночас, для подальшого розвитку системи рекомендується впровадження автоматизованого тестування.

ВИСНОВКИ

Під час роботи над бакалаврською кваліфікаційною роботою було розроблено програмний засіб для управління орендою житла. Метою роботи є покращення зручності процесу управління орендою житла шляхом розробки програмного засобу, що матиме набір функцій, необхідний для забезпечення взаємодії учасників і прозорості процесу оренди.

У ході роботи було проведено аналіз предметної галузі та існуючих програмних засобів для управління орендою житла, таких як Rentster, Discussio, Pickspace і Rentroom. Використовуючи засоби візуаліації, такі як UML-діаграми та блок-схеми, було розроблено схеми моделі системи та алгоритми роботи деяких алгоритмів.

Було розроблено модулі програмного засобу для управління орендою житла, що надають функції для документообігу, обліку платежів і комунальних послуг, керування об’єктами оренди, комунікації та керування договорами. Для цього було використано мову програмування Java і набір фреймворків Spring (Boot, Data, Web MVC, Security). Для зберігання та обробки даних було застосовано гібридний підхід з використанням PostgreSQL та MongoDB. Для зберігання файлових даних було впроваджено сховище MinIO. Асинхронна обробка повідомлень реалізована за допомогою Apache Kafka. Розробка здійснювалася у середовищі IntelliJ IDEA. Для спрощення розгортання застосунку було використано засоби контейнеризації Docker і Docker Compose.

Вебінтерфейс застосунку було розроблено використовуючи мову програмування TypeScript, фреймворк Angular і бібліотеку елементів користувацького інтерфейсу Angular Material.

Було проведено тестування програмного засобу на предмет працездатності і відповідності до поставленого технічного завдання. Окрім цього, було розроблено інструкцію користувача.

Бакалаврську кваліфікаційну роботу було оформлено відповідно до методичних вказівок [41].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Терешко Д. С., Бабюк Н. П. Методи оптимізації процесу управління орендою житла із використанням інформаційно-комунікаційних технологій. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 158-160.

2. Терешко Д. С., Бабюк Н. П. Порівняльний аналіз СКБД PostgreSQL та MongoDB для реалізації онлайн-чату у системі управління орендою житла. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 56-58.

3. Терешко Д. С., Бабюк Н. П. Методи оптимізації зберігання великих масивів даних у базах даних. *Комп`ютерні ігри та мультимедіа як інноваційний підхід до комунікації* : матеріали IV Всеукр. наук.-тех. конф., 26-27 вер. 2024 р. Одеса : ОНТУ, 2024. С. 251-252.

4. The Role of Digital Platforms in Revolutionizing Property Management: вебсайт. URL: https://realtyboris.com/2024/06/02/digital-platforms-revolutionizing-property-management/ (дата звернення: 30.03.2025).

5. Rentster: вебсайт. URL: https://rentster.eu/en/software-landing-h5-2/ (дата звернення: 02.04.2025).

6. Discussio: вебсайт. URL: https://discussio.com.ua/ (дата звернення: 02.04.2025).

7. Pickspace: вебсайт. URL: https://pickspace.com/ (дата звернення: 02.04.2025).

8. Rentroom: вебсайт. URL: https://www.rentroom.com/ (дата звернення: 02.04.2025).

9. Newman Sam. Building Microservices, 2nd Edition. Sebastopol, CA: O'Reilly Media, Inc, 2021. – 278 p.

10. Bass Len, Clements Paul, Kazman Rick. Software Architecture in Practice, 4th Edition. Boston, Massachusetts: Addison-Wesley Professional, 2021. – 464 p.

11. What is REST?: вебсайт. URL: https://restfulapi.net/ (дата звернення: 10.04.2025).

12. The WebSocket API (WebSockets): вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 10.04.2025).

13. JSON Web Tokens: вебсайт. URL: https://jwt.io/ (дата звернення: 11.04.2025).

14. What Is Aspect-Oriented Programming (AOP)? Meaning, Working, and Frameworks: вебсайт. URL: https://www.spiceworks.com/tech/devops/articles/what-is-aop/ (дата звернення: 12.04.2025).

15. PostgreSQL audit logging using triggers: вебсайт. URL: https://vladmihalcea.com/postgresql-audit-logging-triggers/ (дата звернення: 12.04.2025).

16. Blob vs. File System in System Design: вебсайт. URL: https://www.geeksforgeeks.org/blob-vs-file-system-in-system-design/ (дата звернення: 13.04.2025).

17. Block, Object, and File Storage in System Design: вебсайт. URL: https://www.geeksforgeeks.org/block-object-and-file-storage-in-cloud-with-difference/ (дата звернення: 13.04.2025).

18. Spring Boot: вебсайт. URL: https://spring.io/projects/spring-boot (дата звернення: 18.04.2025).

19. Schildt Herbert, Coward Danny, Java: The complete reference, thirteenth edition, 13th ed. Columbus, OH: McGraw-Hill Education, 2024. – 1280 p.

20. Spring Framework Overview: вебсайт. URL: https://docs.spring.io/spring-framework/reference/overview.html (дата звернення: 18.04.2025).

21. Spring Security: вебсайт. URL: https://spring.io/projects/spring-security (дата звернення: 18.04.2025).

22. Spring Web MVC: вебсайт. URL: https://docs.spring.io/spring-framework/reference/web/webmvc.html (дата звернення: 18.04.2025).

23. What is TypeScript?: вебсайт. URL: https://www.typescriptlang.org/ (дата звернення: 19.04.2025).

24. Heckers Roberto. Effective Angular: Develop applications of any size by effectively using Angular with Nx, RxJS, NgRx, and Cypress. Birmingham, England: Packt Publishing, 2024. – 400 p.

25. Angular Material: вебсайт. URL: https://material.angular.io/ (дата звернення: 20.04.2025).

26. PostgreSQL 17.4 Documentation: вебсайт. URL: https://www.postgresql.org/docs/17/index.html (дата звернення: 20.04.2025).

27. MongoDB: вебсайт. URL: https://www.mongodb.com/ (дата звернення: 20.04.2025).

28. MinIO: вебсайт. URL: https://min.io/ (дата звернення: 20.04.2025).

29. Apache Kafka: вебсайт. URL: https://kafka.apache.org/ (дата звернення: 20.04.2025).

30. IntelliJ IDEA: вебсайт. URL: https://www.jetbrains.com/idea/ (дата звернення: 16.04.2025).

31. Visual Studio Code: вебсайт. URL: https://code.visualstudio.com/ (дата звернення: 16.04.2025).

32. Eclipse IDE: вебсайт. URL: https://eclipseide.org/ (дата звернення: 16.04.2025).

33. Docker: вебсайт. URL: https://www.docker.com/ (дата звернення: 20.04.2025).

34. Docker Compose: вебсайт. URL: https://docs.docker.com/compose/ (дата звернення: 20.04.2025).

35. Spring Data JPA: вебсайт. URL: https://spring.io/projects/spring-data-jpa (дата звернення: 21.04.2025).

36. Spring for Apache Kafka: вебсайт. URL: https://spring.io/projects/spring-kafka (дата звернення: 21.04.2025).

37. Spring Data MongoDB: вебсайт. URL: https://spring.io/projects/spring-data-mongodb (дата звернення: 21.04.2025).

38. SPA (Single-page application): вебсайт. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 22.04.2025).

39. Що таке тестування ПЗ, його етапи, види, інструменти: вебсайт. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 02.05.2025).

40. В чому різниця між автоматизованим і ручним тестуванням?: вебсайт. URL: https://university.sigma.software/manual-testing-vs-automation-testing/ (дата звернення: 02.05.2025).

41. Романюк О. Н., Черноволик Г. О. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення». Вінниця: ВНТУ, 2022. 52 с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., професор О. Н. Романюк
«25» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного засобу для управління орендою житла» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доцент Н. П. Бабюк
«24» березня 2025 р.
Виконав:
студент гр. 5ПІ-21б Д. С. Терешко
«24» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного засобу для управління орендою житла».

Галузь застосування – оренда житлової нерухомості.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є покращення процесу управління орендою житла шляхом розробки програмного засобу, що матиме набір функцій, необхідний для забезпечення взаємодії учасників і прозорості процесу оренди.

Призначення роботи – розробка вебзастосунку для управління орендою житла.

4. Вихідні дані для проведення НДР

1. Терешко Д. С., Бабюк Н. П. Методи оптимізації процесу управління орендою житла із використанням інформаційно-комунікаційних технологій. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 158-160.

2. The Role of Digital Platforms in Revolutionizing Property Management: вебсайт. URL: https://realtyboris.com/2024/06/02/digital-platforms-revolutionizing-property-management/ (дата звернення: 30.03.2025).

3. Schildt Herbert, Coward Danny, Java: The complete reference, thirteenth edition, 13th ed. Columbus, OH: McGraw-Hill Education, 2024. – 1280 p.

4. Spring Boot: вебсайт. URL: https://spring.io/projects/spring-boot (дата звернення: 18.04.2025).

5. Heckers Roberto. Effective Angular: Develop applications of any size by effectively using Angular with Nx, RxJS, NgRx, and Cypress. Birmingham, England: Packt Publishing, 2024. – 400 p.

6. PostgreSQL 17.4 Documentation: вебсайт. URL: https://www.postgresql.org/docs/17/index.html (дата звернення: 20.04.2025).

7. MongoDB: вебсайт. URL: https://www.mongodb.com/ (дата звернення: 20.04.2025).

5. Технічні вимоги

Тип програми – вебзастосунок; вхідні дані: параметри об’єктів житлової нерухомості, дані учасників договірних відносин, умови договорів, документи, відомості про платежі і використані комунальні послуги; середовище розробки – IntelliJ IDEA; мови програмування – Java, TypeScript; фреймворки – Spring (Boot, Data, Web MVC, Security), Angular; системи керування базами даних – PostgreSQL, MongoDB.

6. Конструктивні вимоги.

Вебзастосунок повинен відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програмних засобів керування орендою житла та постановка задач дослідження	25.03.25 – 03.04.25
2	Розробка методів, алгоритмів та моделей програмного засобу	04.04.25 – 15.04.25
3	Розробка модулів програмного засобу	16.04.25 – 01.05.25
4	Тестування програмного засобу	02.05.25 – 14.05.25
5	Розробка інструкції користувача	15.05.25 – 21.05.25
6	Оформлення матеріалів до захисту БКР	22.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного засобу для управління орендою житла

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Бабюк Н. П., к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Терешко Д. С
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

```
@Component({
	selector: 'app-audit-log',
	imports: [
		TableComponent,
		DatePipe,
		NgClass,
		MatButton,
		MatDialogActions,
		MatDialogTitle,
		MatDialogContent,
	],
	templateUrl: './audit-log.component.html',
	styleUrl: './audit-log.component.scss',
})
export class AuditLogComponent {
	public auditDataSource: DataSourceConfig<AuditLogItem>;

	protected readonly AuditActionType = AuditActionType;

	constructor(
		private contractService: ContractService,
		public dialogRef: MatDialogRef<AuditLogComponent>,
		@Inject(MAT_DIALOG_DATA) public data: AuditLogDialogData,
	) {
		this.auditDataSource = {
			source: (listRequest: ListRequest) => this.contractService.auditList(listRequest,
this.data?.contractId)
		};
	}

	public onClose(): void {
		this.dialogRef.close();
	}

	public getObjectKeys(obj: Record<string, string>): string[] {
		return obj ? Object.keys(obj) : [];
	}
}

@Component({
	selector: 'app-document-list',
	imports: [
		TableComponent,
		MatIconButton,
		MatIcon,
		DatePipe,
		ListActionsComponent,
	],
	templateUrl: './document-list.component.html',
	styleUrl: './document-list.component.scss',
})
export class DocumentListComponent {
	@Input() contractId = '';
```

```
	public documentDataSource: DataSourceConfig<Document> = {
		source: (listRequest: ListRequest) => this.documentService.list(listRequest, this.contractId)
	};
	public trigger = 0;

	constructor(
		private dialog: MatDialog,
		private documentService: DocumentService,
	) {}

	public onDocumentDownloadClick(document: { id: string, lastVersionFileUrl?: string }): void {
		window.open(document.lastVersionFileUrl, '_blank');
	}

	public onDocumentClick(document: { id: string, title?: string }): void {
		const dialogRef = this.dialog.open(DocumentVersionDialogComponent, {
			width: '800px',
			maxWidth: '90vw',
			data: { documentId: document.id, title: document.title },
		});

		dialogRef.afterClosed()
			.subscribe((result) => {
				if (result) {
					this.trigger++;
				}
			});
	}

	public openDocumentDialog(): void {
		const dialogRef = this.dialog.open(DocumentDialogComponent, {
			width: '500px',
			data: { contractId: this.contractId },
		});

		dialogRef.afterClosed()
			.pipe(switchMap(result => this.documentService.save(result as Document)))
			.subscribe((result) => {
				if (result) {
					this.trigger++;
				}
			});
	}
}

@Component({
	selector: 'app-contract-list',
	imports: [TableWithFiltersComponent, NgClass],
	templateUrl: './contract-list.component.html',
	styleUrl: './contract-list.component.scss',
})
export class ContractListComponent {
	public contractDataSource: DataSourceConfig<Contract> = {
		source: (listRequest: ListRequest, selectedFilters: string[]) =>
			this.contractService.list(listRequest, selectedFilters ?? []),
	};
	public filters: FilterOption[] = [];
```

```
	protected readonly ContractStatus = ContractStatus;

	constructor(
		private router: Router,
		private contractService: ContractService,
	) {
		this.filters = this.contractService.getFilters();
	}

	public onContractClick($event: TableItem): void {
		this.router.navigate([
			`/${AppLocation.CONTRACT}/${AppLocation.VIEW}/${$event.id}`,
		]);
	}

	public onCreateContract(): void {
		this.router.navigate([`/${AppLocation.CONTRACT}/${AppLocation.ADD}`]);
	}
}

@Component({
	selector: 'app-issue-add',
	imports: [
		MatProgressSpinner,
		MatIcon,
		MatCard,
		ReactiveFormsModule,
		MatCardContent,
		MatFormField,
		SearchDropdownComponent,
		MatSelect,
		MatOption,
		ActionBarComponent,
		MatInput,
		MatLabel,
		MatError
	],
	templateUrl: './issue-add.component.html',
	styleUrl: './issue-add.component.scss',
})
export class IssueAddComponent implements OnInit, OnDestroy {
	public issueForm: FormGroup;
	public isEditMode = false;
	public loading = false;
	public loadingData = false;
	public error: string | null = null;
	public statusOptions = ISSUE_STATUS_OPTIONS;
	public contracts: DropdownOption[] = [];
	public users: DropdownOption[] = [];

	public leftActions: ActionItem[] = [
		{
			icon: 'arrow_back',
			label: 'Назад',
			callback: () => this.navigateBack(),
		},
	];
```

```
public rightActions: ActionItem[] = [
	{
		icon: 'save',
		label: 'Зберегти',
		color: 'primary',
		callback: () => this.saveIssue(),
		flatButton: true,
	},
];

private destroy$ = new Subject<void>();
private issueId = '';

constructor(
	private fb: FormBuilder,
	private route: ActivatedRoute,
	private router: Router,
	private issueService: IssueService,
	private snackBar: MatSnackBar,
) {
	this.issueForm = this.createForm();
}

public ngOnInit(): void {
	this.loadingData = true;

	this.issueId = this.route.snapshot.paramMap.get('id');
	this.isEditMode = !!this.issueId;

	this.loadContracts();
	this.loadUsers();

	if (this.isEditMode && this.issueId) {
		this.loadIssue(this.issueId);
	} else {
		this.loadingData = false;
	}
}

public ngOnDestroy(): void {
	this.destroy$.next();
	this.destroy$.complete();
}

public saveIssue(): void {
	if (this.issueForm.invalid) {
		this.markFormGroupTouched(this.issueForm);
		return;
	}

	this.loading = true;

	const issueData: Issue = {
		...this.issueForm.value as Issue,
		id: this.issueId,
	};
```

```
        this.issueService
                .saveIssue(issueData)
                .pipe(
                        takeUntil(this.destroy$),
                        finalize(() => (this.loading = false)),
                )
                .subscribe({
                        next: (issue) => {
                                const message = this.isEditMode
                                        ? 'Звернення успішно оновлено'
                                        : 'Звернення успішно створено';

                                this.snackBar.open(message, 'Закрити', {
                                        duration: 3000,
                                });

                                this.navigateToView(issue.id);
                        },
                        error: (error) => {
                                this.error = `Не вдалося зберегти звернення: ${error.message}`;
                        },
                });
}

public onContractSelected(contractId: string | null): void {
        this.issueForm.get('contractId')?.setValue(contractId);
}

public onAssigneeSelected(assigneeId: string | null): void {
        this.issueForm.get('assigneeId')?.setValue(assigneeId);
}

private createForm(): FormGroup {
        return this.fb.group({
                title: ['', [Validators.required.bind(this), Validators.maxLength(100)]],
                description: ['', [Validators.required.bind(this)]],
                status: [IssueStatus.CREATED, [Validators.required.bind(this)]],
                contractId: [null],
                assigneeId: [null],
        });
}

private loadIssue(id: string): void {
        this.issueService
                .getIssueById(id)
                .pipe(
                        takeUntil(this.destroy$),
                        finalize(() => (this.loadingData = false)),
                )
                .subscribe({
                        next: (issue) => {
                                this.issueForm.patchValue({
                                        title: issue.title,
                                        description: issue.description,
                                        status: issue.status,
                                        contractId: issue.contractId,
```

```
                                        assigneeId: issue.assigneeId,
                                });
                        },
                        error: (error) => {
                                this.error = `Не вдалося завантажити звернення:
${error?.message}`;
                        },
                });
        }

        private loadContracts(): void {
                this.issueService
                        .getContracts()
                        .pipe(takeUntil(this.destroy$))
                        .subscribe((contracts) => {
                                this.contracts = contracts.map((contract) => ({
                                        id: contract.id,
                                        displayName: contract.displayName,
                                }));
                        });
        }

        private loadUsers(): void {
                this.issueService
                        .getUsers()
                        .pipe(takeUntil(this.destroy$))
                        .subscribe((users) => {
                                this.users = users.map((user) => ({
                                        id: user.id,
                                        displayName: user.displayName,
                                }));
                        });
        }

        private navigateBack(): void {
                if (this.isEditMode && this.issueId) {
                        this.router.navigate([
                                `/${AppLocation.ISSUE}/${AppLocation.VIEW}/${this.issueId}`,
                        ]);
                } else {
                        this.router.navigate([`/${AppLocation.ISSUE}/${AppLocation.LIST}`]);
                }
        }

        private navigateToView(id: string): void {
                this.router.navigate([
                        `/${AppLocation.ISSUE}/${AppLocation.VIEW}/${id}`,
                ]);
        }

        private markFormGroupTouched(formGroup: FormGroup): void {
                Object.values(formGroup.controls).forEach((control) => {
                        control.markAsTouched();
                        if ((control as any).controls) {
                                this.markFormGroupTouched(control as FormGroup);
                        }
                });
```

```
        }
}

@Component({
        selector: 'app-table',
        imports: [CdkScrollable, MatIcon, MatProgressSpinner, NgTemplateOutlet],
        templateUrl: './table.component.html',
        styleUrl: './table.component.scss',
})
export class TableComponent implements OnInit, OnChanges, OnDestroy {
        @ViewChild(CdkScrollable) scrollable!: CdkScrollable;
        @Input({ required: true }) itemTemplate: TemplateRef<any>;
        @Input({ required: true }) dataSource: DataSourceConfig;
        @Input() selectedFilters: FilterOption[] = [];
        @Input() triggerUpdate = 0;
        @Output() itemClick = new EventEmitter<TableItem>();

        public pageSize = 10;
        public visibleItems: TableItem[] = [];
        public loading = false;
        public noMoreItems = false;
        public currentPage = 0;

        private subscriptions = new Subscription();
        private loadingSubject = new BehaviorSubject<boolean>(false);

        public ngOnChanges(changes: SimpleChanges): void {
                if (changes['selectedFilters'] || changes['triggerUpdate']) {
                        this.resetAndReload();
                }
        }

        public ngOnInit(): void {
                this.subscriptions.add(
                        this.loadingSubject.subscribe((loading) => {
                                this.loading = loading;
                        }),
                );

                this.resetAndReload();
        }

        public ngOnDestroy(): void {
                this.subscriptions.unsubscribe();
        }

        public onScroll(event: Event): void {
                const element = event.target as HTMLElement;

                if (
                        !this.loading &&
                        !this.noMoreItems &&
                        element.scrollHeight - element.scrollTop < element.clientHeight * 1.5
                ) {
                        this.loadMoreItems();
                }
        }
```

```
	public onItemClick(item: TableItem): void {
		this.itemClick.emit(item);
	}

	public resetAndReload(): void {
		this.visibleItems = [];
		this.currentPage = 0;
		this.noMoreItems = false;
		this.loadMoreItems();
	}

	private loadMoreItems(): void {
		this.currentPage++;

		if (this.loading) return;

		const activeFilters = this.selectedFilters.map(f => f.value);

		this.loadingSubject.next(true);

		this.subscriptions.add(
			this.dataSource
				.source({ pageSize: this.pageSize, page: this.currentPage }, activeFilters)
				.pipe(finalize(() => this.loadingSubject.next(false)))
				.pipe(catchError(err => {
					console.error('Error fetching data:', err);
					this.noMoreItems = true;
					throw err;
				}))
				.subscribe(items => {
					if (items.length === 0) {
						this.noMoreItems = true;
					} else {
						this.visibleItems = [
							...this.visibleItems,
							...items,
						];
					}
				})
		);
	}
}

@Component({
	selector: 'app-table-with-filters',
	imports: [MatIcon, MatChipOption, MatFabButton, TableComponent],
	templateUrl: './table-with-filters.component.html',
	styleUrl: './table-with-filters.component.scss',
})
export class TableWithFiltersComponent {
	@Input({ required: true }) itemTemplate: TemplateRef<any>;
	@Input({ required: true }) dataSource: DataSourceConfig;
	@Input() filters: FilterOption[] = [];
	@Output() itemClick = new EventEmitter<TableItem>();
	@Output() createClick = new EventEmitter<void>();
```

```
	public selectedFilters: FilterOption[] = [];

	public toggleFilter(filter: FilterOption): void {
		filter.selected = !filter.selected;
		this.selectedFilters = this.filters.filter((f) => f.selected);
	}

	public onItemClick(item: TableItem): void {
		this.itemClick.emit(item);
	}

	public onCreateClick(): void {
		this.createClick.emit();
	}
}

@Injectable({ providedIn: 'root' })
export class AuthService {

	private apiUrl = `${environment.apiUrl}/auth`;
	private jwtHelper = new JwtHelperService();
	private currentUserSubject = new BehaviorSubject<User | null>(null);
	private refreshTokenTimeout: any;

	constructor(
		private http: HttpClient,
		private router: Router
	) {
		this.loadCurrentUser();
	}

	public get isAuthenticated(): boolean {
		const accessToken = localStorage.getItem('accessToken');
		return !!accessToken && !this.jwtHelper.isTokenExpired(accessToken);
	}

	public login(email: string, password: string): Observable<User> {
		return this.http.post<AuthResponse>(`${this.apiUrl}/login`, { email, password })
			.pipe(
				tap(response => this.handleAuthResponse(response)),
				map(response => response.user),
				catchError(this.handleError.bind(this))
			);
	}

	public register(userData: {
		email: string;
		firstName: string;
		lastName: string;
		fullOfficialName: string;
		taxNumber: string;
		phoneNumber: string;
		password: string;
	}): Observable<any> {
		return this.http.post(`${this.apiUrl}/register`, userData)
			.pipe(catchError(this.handleError.bind(this)));
	}
```

```
public logout(): void {
        this.http.post(`${this.apiUrl}/logout`, {})
                .pipe(catchError(() => of(null)))
                .subscribe();

        localStorage.removeItem('accessToken');
        localStorage.removeItem('refreshToken');
        this.currentUserSubject.next(null);
        this.stopRefreshTokenTimer();
        this.router.navigate(['/login']);
}

public refreshToken(): Observable<AuthResponse> {
        const refreshToken = localStorage.getItem('refreshToken');
        if (!refreshToken) {
                return throwError(() => new Error('No refresh token available'));
        }

        return this.http.post<AuthResponse>(`${this.apiUrl}/refresh-token`, { refreshToken })
                .pipe(
                        tap(response => this.handleAuthResponse(response)),
                        catchError(error => {
                                this.logout();
                                return throwError(() => error);
                        })
                );
}

public getAccessToken(): string | null {
        return localStorage.getItem('accessToken');
}

private loadCurrentUser(): void {
        const accessToken = localStorage.getItem('accessToken');
        if (accessToken && !this.jwtHelper.isTokenExpired(accessToken)) {
                const decodedToken = this.jwtHelper.decodeToken(accessToken);
                const user: User = {
                        id: decodedToken.sub,
                        email: decodedToken.email,
                        firstName: decodedToken.firstName,
                        lastName: decodedToken.lastName
                };
                this.currentUserSubject.next(user);
                this.startRefreshTokenTimer();
        }
}

private handleAuthResponse(response: AuthResponse): void {
        localStorage.setItem('accessToken', response.accessToken);
        localStorage.setItem('refreshToken', response.refreshToken);

        this.currentUserSubject.next(response.user);

        this.startRefreshTokenTimer();
}
```

```
	private startRefreshTokenTimer(): void {
		const accessToken = localStorage.getItem('accessToken');
		if (!accessToken) return;

		const expiresIn = this.jwtHelper.getTokenExpirationDate(accessToken).getTime() -
Date.now();
		const timeout = expiresIn - (60 * 1000);

		this.refreshTokenTimeout = setTimeout(() => {
			this.refreshToken().subscribe();
		}, Math.max(0, timeout));
	}

	private stopRefreshTokenTimer(): void {
		clearTimeout(this.refreshTokenTimeout);
	}

	private handleError(error: HttpErrorResponse): Observable<never> {
		let errorMessage = 'Сталася помилка під час обробки запиту.';

		if (error.error instanceof ErrorEvent) {
			errorMessage = `Помилка: ${error.error.message}`;
		} else if (error.status === 401) {
			errorMessage = 'Неправильний логін або пароль.';
		} else if (error.status === 409) {
			errorMessage = 'Користувач з таким email вже існує.';
		} else if (error.error?.message) {
			errorMessage = error?.error?.message as string;
		}

		return throwError(() => ({
			error: { message: errorMessage },
			status: error.status,
		}));
	}
}

@Component({
	selector: 'app-conversation-list',
	imports: [
		MatIcon,
		FormsModule,
		DatePipe,
		MatProgressSpinner,
		MatButton,
	],
	templateUrl: './conversation-list.component.html',
	styleUrl: './conversation-list.component.scss',
})
export class ConversationListComponent implements OnInit, OnDestroy {
	public conversations: Conversation[] = [];
	public loading = true;
	public activeConversationId: string | null = null;

	private subscriptions = new Subscription();

	constructor(
```

```
        private router: Router,
        private conversationService: ConversationService,
        private dialog: MatDialog,
) {}

public ngOnInit(): void {
        this.loadConversations();

        const url = this.router.url;
        const match = url.match(/\/chat\/([^\/]+)$/);
        if (match) {
                this.activeConversationId = match[1];
        }
}

public ngOnDestroy(): void {
        this.subscriptions.unsubscribe();
}

public onConversationClick(conversation: Conversation): void {
        this.activeConversationId = conversation.id;
        this.router.navigate(['/', AppLocation.CHAT, conversation.id]);
}

public onCreateConversation(): void {
        const dialogRef = this.dialog.open(NewConversationComponent, {
                width: '400px',
                data: { isIssue: false },
        });

        dialogRef.afterClosed().subscribe((result) => {
                if (result) {
                        this.conversationService
                                .createConversation(result)
                                .subscribe((newConversation) => {
                                        this.conversations.unshift(newConversation);
                                        this.router.navigate([
                                                '/',
                                                AppLocation.CHAT,
                                                newConversation.id,
                                        ]);
                                });
                }
        });
}

public isToday(date: Date): boolean {
        const today = new Date();
        const messageDate = new Date(date);

        return (
                today.getDate() === messageDate.getDate() &&
                today.getMonth() === messageDate.getMonth() &&
                today.getFullYear() === messageDate.getFullYear()
        );
}
```

```
	private loadConversations(): void {
		this.loading = true;

		this.subscriptions.add(
			this.conversationService.list().subscribe(
				(conversations) => {
					this.conversations = conversations;
					this.loading = false;
				},
				(error) => {
					console.error('Помилка завантаження діалогів:', error);
					this.loading = false;
				},
			),
		);
	}
}

@Component({
	selector: 'app-issue-list',
	imports: [TableWithFiltersComponent, NgClass],
	templateUrl: './issue-list.component.html',
	styleUrl: './issue-list.component.scss',
})
export class IssueListComponent {
	public issueDataSource: DataSourceConfig<Issue> = {
		source: (listRequest: ListRequest, selectedFilters: string[]) =>
this.issueService.list(listRequest, selectedFilters ?? [])
	};

	public filters = [
		{ label: 'Створено', value: IssueStatus.CREATED, selected: false },
		{ label: 'У процесі виконання', value: IssueStatus.IN_PROGRESS, selected: false },
		{ label: 'Виконано', value: IssueStatus.COMPLETED, selected: false },
		{ label: 'Архівовано', value: IssueStatus.ARCHIVED, selected: false },
	];
	protected readonly IssueStatus = IssueStatus;

	constructor(
		private router: Router,
		private issueService: IssueService
	) {}

	public onIssueClick($event: TableItem): void {
		this.router.navigate([`${AppLocation.ISSUE}/${AppLocation.VIEW}/${$event.id}`]);
	}

	public onCreateIssue(): void {
		this.router.navigate([`${AppLocation.ISSUE}/${AppLocation.ADD}`]);
	}
}

@Component({
	selector: 'app-document-version-dialog',
	imports: [
		MatDialogTitle,
		ReactiveFormsModule,
```

```
                MatFormField,
                MatInput,
                MatButton,
                TableComponent,
                MatDivider,
                MatDialogActions,
                DatePipe,
                MatIconButton,
                MatIcon,
                MatTooltip,
                MatDialogContent,
                MatLabel,
                MatError,
                CdkTrapFocus,
        ],
        templateUrl: './document-version-dialog.component.html',
        styleUrl: './document-version-dialog.component.scss',
})
export class DocumentVersionDialogComponent {
        public documentVersionDataSource: DataSourceConfig<DocumentVersion>;
        public versionForm: FormGroup;
        public selectedFile: File;
        public fileName = '';
        public trigger = 0;

        constructor(
                private fb: FormBuilder,
                private documentService: DocumentService,
                public dialogRef: MatDialogRef<DocumentVersionDialogComponent>,
                @Inject(MAT_DIALOG_DATA)
                public documentVersionData: DocumentVersionDialogData,
        ) {
                this.documentVersionDataSource = {
                        source: (listRequest) =>
                                this.documentService.listVersions(
                                        listRequest,
                                        this.documentVersionData.documentId,
                                ),
                };
                this.versionForm = this.createForm();
        }

        public onSubmit(): void {
                if (this.versionForm.invalid || !this.selectedFile) {
                        this.markFormGroupTouched(this.versionForm);
                        return;
                }

                const newVersion: Partial<DocumentVersion> = {
                        documentId: this.documentVersionData.documentId,
                        comment: this.versionForm.get('comment')?.value,
                        file: this.selectedFile,
                };

                const formData = new FormData();
                formData.append('documentId', this.documentVersionData.documentId);
                formData.append('comment', this.versionForm.get('comment')?.value);
```

```
                formData.append('file', this.selectedFile);

                this.documentService
                        .saveVersion(newVersion as DocumentVersion)
                        .subscribe({
                                next: (result) => {
                                        if (result) {
                                                this.versionForm.reset();
                                                this.selectedFile = null;

                                                this.trigger++;
                                        }
                                },
                                error: (error) => {
                                        console.error('Помилка:', error);
                                },
                        });
        }

        public onClose(): void {
                this.dialogRef.close();
        }

        public onFileSelected(event: Event): void {
                const input = event.target as HTMLInputElement;

                if (input.files?.length) {
                        this.selectedFile = input.files[0];
                        this.fileName = this.selectedFile?.name || '';
                }
        }

        public onVersionDownloadClick(version: {
                id: string;
                fileUrl?: string;
        }): void {
                if (version.fileUrl) {
                        window.open(version.fileUrl, '_blank');
                }
        }

        private createForm(): FormGroup {
                return this.fb.group({
                        comment: ['', [Validators.required.bind(this)]],
                });
        }

        private markFormGroupTouched(formGroup: FormGroup): void {
                Object.values(formGroup.controls).forEach((control) => {
                        control.markAsTouched();
                        if ((control as any).controls) {
                                this.markFormGroupTouched(control as FormGroup);
                        }
                });
        }
}
```

```
@Configuration
public class SecurityBeansConfiguration {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

@Configuration
@EnableWebSecurity
public class SecurityChainConfiguration {

    @Autowired
    private JwtAuthFilter jwtAuthFilter;

    @Autowired
    private AuthenticationProvider authenticationProvider;

    @Autowired
    private SecurityLogger securityLogger;

    @Value("${application.frontend.url}")
    private String frontendUrl;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/auth/**").permitAll()
                    .requestMatchers("/public/**").permitAll()
                    .requestMatchers(HttpMethod.OPTIONS, "/**").permitAll()
                    .requestMatchers("/ws/**").permitAll()
                    .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                    .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
            .exceptionHandling(exceptions -> exceptions
                    .authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))
            );

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of(frontendUrl));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        configuration.setAllowCredentials(true);
```

```
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws
Exception {
        return config.getAuthenticationManager();
    }

}

@Configuration
public class AuthProviderConfiguration {

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder);
        return authProvider;
    }
}

@Component
public class JwtAuthFilter extends OncePerRequestFilter {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Autowired
    private SecurityLogger securityLogger;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain) throws ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String userEmail;

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        jwt = authHeader.substring(7);
        userEmail = jwtService.extractUsername(jwt);
```

```
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);

            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request)
                );
                SecurityContextHolder.getContext().setAuthentication(authToken);
                securityLogger.logSecurityEvent("Token validation success", userEmail);
            } else {
                securityLogger.logSecurityEvent("Invalid token", userEmail);
            }

        }
        filterChain.doFilter(request, response);
    }
}

@Service
@Transactional
public class RefreshTokenService {

    @Autowired
    private RefreshTokenRepository refreshTokenRepository;

    @Autowired
    private IUserService userService;

    @Value("${application.security.jwt.refresh-token.expiration}")
    private long refreshTokenExpiration;

    public RefreshToken createRefreshToken(String username) {
        IUser user = userService.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));

        RefreshToken refreshToken = new RefreshToken();
        refreshToken.setUser(user);
        refreshToken.setToken(UUID.randomUUID().toString());
        refreshToken.setExpiryDate(DateUtil.plusMillis(refreshTokenExpiration));

        return refreshTokenRepository.save(refreshToken);
    }

    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.isExpired() || token.isRevoked()) {
            refreshTokenRepository.delete(token);
            throw new IllegalStateException("Refresh token was expired or revoked");
        }

        return token;
```

```
    }

    public Optional<RefreshToken> findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }

    public void revokeRefreshToken(String token) {
        RefreshToken refreshToken = refreshTokenRepository.findByToken(token)
                .orElse(null);

        if (refreshToken != null) {
            refreshToken.setRevoked(true);
            refreshTokenRepository.save(refreshToken);
        }
    }
}

@Service
public class JwtService {

    @Autowired
    private IUserService userService;

    @Value("${application.security.jwt.secret-key}")
    private String secretKey;

    @Value("${application.security.jwt.access-token.expiration}")
    private long jwtExpiration;

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        if (userDetails instanceof CustomUserDetails) {
            IUser user = userService.findByEmail(userDetails.getUsername())
                    .orElseThrow(() -> new UsernameNotFoundException("User not found"));

            claims.put("firstName", user.getFirstName());
            claims.put("lastName", user.getLastName());
            claims.put("email", user.getEmail());
        }

        return generateToken(claims, userDetails);

    }

    public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
        return buildToken(extraClaims, userDetails, jwtExpiration);
    }
```

```
    private String buildToken(
            Map<String, Object> extraClaims,
            UserDetails userDetails,
            long expiration
    ) {
        return Jwts
                .builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = secretKey.getBytes();
        return Keys.hmacShaKeyFor(keyBytes);
    }
}

@Service
@Transactional(rollbackOn = Exception.class)
public class AuthService {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private IUserService userService;

    @Autowired
```

```
    private CustomUserDetailsService userDetailsService;

    @Autowired
    private RefreshTokenService refreshTokenService;

    @Autowired
    private SecurityLogger securityLogger;

    public AuthResponseDTO register(RegisterDTO request) {
        if (userService.existsByEmail(request.getEmail())) {
            throw new RuntimeException("Email already exists");
        }

        IUser registeredUser = userService.registerUser(request);

        UserDetails user = userDetailsService.mapUserDetails(registeredUser);

        String accessToken = jwtService.generateToken(user);
        RefreshToken refreshTokenEntity =
refreshTokenService.createRefreshToken(registeredUser.getEmail());

        securityLogger.logRegistration(registeredUser.getEmail());

        return AuthResponseDTO.builder()
                .accessToken(accessToken)
                .refreshToken(refreshTokenEntity.getToken())
                .user(new LightUserDTO(registeredUser))
                .build();
    }

    public AuthResponseDTO authenticate(AuthRequestDTO request) {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        request.getEmail(),
                        request.getPassword()
                )
        );

        IUser user = userService.findByEmail(request.getEmail())
                .orElseThrow(() -> new RuntimeException("User not found"));

        UserDetails userDetails = userDetailsService.mapUserDetails(user);

        String accessToken = jwtService.generateToken(userDetails);
        RefreshToken refreshTokenEntity = refreshTokenService.createRefreshToken(user.getEmail());

        return AuthResponseDTO.builder()
                .accessToken(accessToken)
                .refreshToken(refreshTokenEntity.getToken())
                .user(new LightUserDTO(user))
                .build();
    }

    public AuthResponseDTO refreshToken(String refreshToken) {
        RefreshToken tokenEntity = refreshTokenService.findByToken(refreshToken)
                .orElseThrow(() -> new RuntimeException("Refresh token not found"));
```

```
        refreshTokenService.verifyExpiration(tokenEntity);

        IUser user = tokenEntity.getUser();
        UserDetails userDetails = userDetailsService.mapUserDetails(user);

        String accessToken = jwtService.generateToken(userDetails);

        refreshTokenService.revokeRefreshToken(refreshToken);
        RefreshToken newRefreshToken = refreshTokenService.createRefreshToken(user.getEmail());

        securityLogger.logTokenRefresh(user.getEmail());

        return AuthResponseDTO.builder()
                .accessToken(accessToken)
                .refreshToken(newRefreshToken.getToken())
                .user(new LightUserDTO(user))
                .build();
    }

    public void revokeRefreshToken(String refreshToken) {
        Optional<RefreshToken> tokenOpt = refreshTokenService.findByToken(refreshToken);
        if (tokenOpt.isPresent()) {
            IUser user = tokenOpt.get().getUser();
            securityLogger.logLogout(user.getEmail());
        }

        refreshTokenService.revokeRefreshToken(refreshToken);
    }

    public boolean validateToken(String token) {
        try {
            String username = jwtService.extractUsername(token);
            if (username == null) {
                return false;
            }

            UserDetails userDetails = userDetailsService.loadUserByUsername(username);

            return jwtService.isTokenValid(token, userDetails);
        } catch (Exception e) {
            return false;
        }
    }
}

@Transactional(rollbackOn = Exception.class)
public abstract class BaseCRUDService<M extends IBaseModel, D extends IBaseDTO>
    implements IBaseService<M, D> {

    @Autowired
    private ICurrentUserService currentUserService;

    @Override
    public boolean existsById(UUID id) {
        return getRepository().existsById(id);
    }
```

```
    @Override
    public D getById(UUID id) {
        M model = getRepository().findById(id).orElseThrow();

        return getMapper().convert(model);
    }

    @Override
    public M getModelById(UUID id) {
        return getRepository().getReferenceById(id);
    }

    @Override
    public D save(D dto) {
        M model = getMapper().convert(dto);

        IUser currentUser = currentUserService.getCurrentUser()
            .orElseThrow(() -> new EntityNotFoundException("Current user is null"));
        if (model.isTransient()) {
            setCreatedBy(currentUser, model);
        } else {
            setModifiedBy(currentUser, model);
        }

        M savedModel = getRepository().save(model);

        return getMapper().convert(savedModel);
    }

    @Override
    public void delete(UUID id) {
        M model = getModelById(id);
        model.setDeleted(true);
        getRepository().save(model);
    }

    @Override
    public Collection<D> list(ListRequestDTO listRequest) {
        PageRequest pageRequest = PageRequest.of(listRequest.pageNumber(), listRequest.pageSize());

        if (StringUtils.isNotBlank(getModelFilterProperty())) {
            Specification<M> filterSpec = ListRequestFilterSpecification.buildFilter(getModelFilterProperty(),
listRequest.selectedFilters());
            return getRepository().findAll(filterSpec, pageRequest).get()
                    .map(getMapper()::convert)
                    .toList();
        }

        return getRepository().findAll(pageRequest).get()
                .map(getMapper()::convert)
                .toList();
    }

    protected Collection<D> listByAttribute(ListRequestDTO listRequest, String attribute, String value) {
        PageRequest pageRequest = PageRequest.of(listRequest.pageNumber(), listRequest.pageSize());

        if (StringUtils.isNotBlank(attribute) && StringUtils.isNotBlank(value)) {
```

```
            Specification<M> filterSpec = ListRequestFilterSpecification.buildFilter(attribute,
Collections.singletonList(value));
            return getRepository().findAll(filterSpec, pageRequest).get()
                .map(getMapper()::convert)
                .toList();
        }

        return Collections.emptyList();
    }

    private void setCreatedBy(IUser currentUser, M model) {
        if (model instanceof IHasCreatedBy hasCreatedBy) {
            hasCreatedBy.setCreatedBy(currentUser);
        }
    }

    private void setModifiedBy(IUser currentUser, M model) {
        if (model instanceof IHasLastModifiedBy hasLastModifiedBy) {
            hasLastModifiedBy.setLastModifiedBy(currentUser);
        }
    }

    protected abstract IBaseRepository<M> getRepository();

    protected abstract IBaseModelDTOMapper<M, D> getMapper();

    protected abstract String getModelFilterProperty();

}

@NoRepositoryBean
public interface IBaseRepository<M extends IBaseModel> extends JpaRepository<M, UUID>,
JpaSpecificationExecutor<M> {
}

public abstract class BaseCRUDController<M extends IBaseModel, D extends IBaseDTO> {

    @GetMapping("/{id}")
    public ResponseEntity<D> getById(@PathVariable UUID id) {
        return ResponseEntity.ok(getService().getById(id));
    }

    @PostMapping("/create")
    public ResponseEntity<D> create(D dto) {
        return ResponseEntity.ok(getService().save(dto));
    }

    @PutMapping("/{id}")
    public ResponseEntity<D> update(@PathVariable UUID id, D dto) {
        if (!getService().existsById(id)) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(getService().save(dto));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable UUID id) {
```

```
        getService().delete(id);
        return ResponseEntity.noContent().build();
    }

    @PostMapping("/list")
    public ResponseEntity<Collection<D>> list(@RequestBody ListRequestDTO listRequest) {
        return ResponseEntity.ok(getService().list(listRequest));
    }

    protected abstract IBaseService<M, D> getService();
}

public interface IBaseService<M extends IBaseModel, D extends IBaseDTO> {

    boolean existsById(UUID id);

    D getById(UUID id);

    M getModelById(UUID id);

    D save(D dto);

    void delete(UUID id);

    Collection<D> list(ListRequestDTO listRequest);
}

@Entity
@Table(name = "contract")
@Getter
@Setter
public class Contract extends BaseAuditableModel implements IContract {

    @Column(name = "generated_id", length = 32, nullable = false)
    private String generatedId;

    @ManyToOne(targetEntity = Property.class, fetch = FetchType.LAZY)
    @JoinColumn(name = "property_id", nullable = false)
    private IProperty property;

    @Column(name = "status", nullable = false)
    @Enumerated(EnumType.STRING)
    private ContractStatus status;

    @Column(name = "monthly_rate", nullable = false)
    private BigDecimal monthlyRate;

    @Column(name = "start_date", nullable = false)
    private LocalDate startDate;

    @Column(name = "end_date", nullable = false)
    private LocalDate endDate;

    @OneToMany(targetEntity = ContractParty.class, fetch = FetchType.LAZY, mappedBy = "contract")
    private List<IContractParty> renters;

}
```

```
public interface IIssueService extends IBaseService<Issue, IssueDTO> {
}

@Service
public class IssueService extends BaseCRUDService<Issue, IssueDTO> implements IIssueService {

   @Autowired
   private IIssueRepository issueRepository;

   @Autowired
   private IIssueMapper issueMapper;

   @Override
   protected IBaseRepository<Issue> getRepository() {
      return issueRepository;
   }

   @Override
   protected IBaseModelDTOMapper<Issue, IssueDTO> getMapper() {
      return issueMapper;
   }

   @Override
   protected String getModelFilterProperty() {
      return "status";
   }
}

@RestController
@RequestMapping("/issue")
public class IssueController extends BaseCRUDController<Issue, IssueDTO> {

   @Autowired
   private IssueService issueService;

   @Override
   protected IBaseService<Issue, IssueDTO> getService() {
      return issueService;
   }
}

@Repository
public interface IIssueRepository extends IBaseRepository<Issue> {
}


@Document(collection = "messages")
@Getter
@Setter
public class Message {

   @Id
   private String id;

   @Field("conversation_id")
   private String conversationId;
```

```
@Field("sender_id")
private UUID senderId;

@Field("text")
private String text;

@Field("timestamp")
private LocalDateTime timestamp;

@Field("status")
private String status = "sent";

@Field("delivered_to")
private List<DeliveryInfo> deliveredTo = new ArrayList<>();

@Field("read_by")
private List<ReadInfo> readBy = new ArrayList<>();

@Field("undelivered_to")
private List<UUID> undeliveredTo = new ArrayList<>();

public Message() {
    this.timestamp = LocalDateTime.now();
}

public Message(String conversationId, UUID senderId, String text) {
    this();
    this.conversationId = conversationId;
    this.senderId = senderId;
    this.text = text;
}

public void markDeliveredTo(UUID userId) {
    DeliveryInfo delivery = new DeliveryInfo();
    delivery.setUserId(userId);
    delivery.setDeliveredAt(LocalDateTime.now());

    this.deliveredTo.removeIf(d -> d.getUserId().equals(userId));
    this.deliveredTo.add(delivery);

    this.undeliveredTo.remove(userId);

    updateStatus();
}

public void markReadBy(UUID userId) {
    ReadInfo read = new ReadInfo();
    read.setUserId(userId);
    read.setReadAt(LocalDateTime.now());

    this.readBy.removeIf(r -> r.getUserId().equals(userId));
    this.readBy.add(read);

    updateStatus();
}
```

```
    public void addUndeliveredTo(UUID userId) {
        if (!this.undeliveredTo.contains(userId)) {
            this.undeliveredTo.add(userId);
        }
        updateStatus();
    }

    public void removeUndeliveredTo(UUID userId) {
        this.undeliveredTo.remove(userId);
        updateStatus();
    }

    private void updateStatus() {
        if (!readBy.isEmpty()) {
            this.status = "read";
        } else if (!deliveredTo.isEmpty()) {
            this.status = "delivered";
        } else {
            this.status = "sent";
        }
    }

    public boolean isDeliveredTo(UUID userId) {
        return deliveredTo.stream().anyMatch(d -> d.getUserId().equals(userId));
    }

    public boolean isReadBy(UUID userId) {
        return readBy.stream().anyMatch(r -> r.getUserId().equals(userId));
    }

    public boolean isUndeliveredTo(UUID userId) {
        return undeliveredTo.contains(userId);
    }

    @Getter
    @Setter
    public static class DeliveryInfo {
        private UUID userId;
        private LocalDateTime deliveredAt;
    }

    @Getter
    @Setter
    public static class ReadInfo {
        private UUID userId;
        private LocalDateTime readAt;
    }
}

@Document(collection = "conversations")
@Getter
@Setter
public class Conversation {

    @Id
    private String id;
```

```
@Field("title")
private String title;

@Field("participants")
private List<UUID> participants;

@Field("last_message")
private String lastMessage;

@Field("last_message_date")
private LocalDateTime lastMessageDate;

@Field("created_at")
private LocalDateTime createdAt;

@Field("updated_at")
private LocalDateTime updatedAt;

@Field("online_participants")
private List<OnlineParticipant> onlineParticipants = new ArrayList<>();

public Conversation() {
    this.createdAt = LocalDateTime.now();
    this.updatedAt = LocalDateTime.now();
    this.onlineParticipants = new ArrayList<>();
}

public Conversation(List<UUID> participants) {
    this();
    this.participants = participants;
}

public Conversation(String title, List<UUID> participants) {
    this(participants);
    this.title = title;
}

public void updateLastMessage(String message) {
    this.lastMessage = message;
    this.lastMessageDate = LocalDateTime.now();
    this.updatedAt = LocalDateTime.now();
}

public void addOnlineParticipant(UUID userId, String sessionId) {
    removeOnlineParticipant(userId); // Remove if already exists
    OnlineParticipant participant = new OnlineParticipant();
    participant.setUserId(userId);
    participant.setSessionId(sessionId);
    participant.setConnectedAt(LocalDateTime.now());
    participant.setLastSeen(LocalDateTime.now());
    this.onlineParticipants.add(participant);
}

public void removeOnlineParticipant(UUID userId) {
    this.onlineParticipants.removeIf(p -> p.getUserId().equals(userId));
}
```

```
    public void updateParticipantLastSeen(UUID userId) {
        this.onlineParticipants.stream()
            .filter(p -> p.getUserId().equals(userId))
            .findFirst()
            .ifPresent(p -> p.setLastSeen(LocalDateTime.now()));
    }

    public boolean isParticipantOnline(UUID userId) {
        return this.onlineParticipants.stream()
            .anyMatch(p -> p.getUserId().equals(userId) &&
                p.getLastSeen().isAfter(LocalDateTime.now().minusMinutes(5)));
    }

    public List<UUID> getOnlineParticipantIds() {
        LocalDateTime cutoff = LocalDateTime.now().minusMinutes(5);
        return this.onlineParticipants.stream()
            .filter(p -> p.getLastSeen().isAfter(cutoff))
            .map(OnlineParticipant::getUserId)
            .toList();
    }

    public List<UUID> getOfflineParticipantIds() {
        List<UUID> onlineIds = getOnlineParticipantIds();
        return this.participants.stream()
            .filter(userId -> !onlineIds.contains(userId))
            .toList();
    }

    public void cleanupStaleConnections() {
        LocalDateTime cutoff = LocalDateTime.now().minusMinutes(5);
        this.onlineParticipants.removeIf(p -> p.getLastSeen().isBefore(cutoff));
    }

    @Getter
    @Setter
    public static class OnlineParticipant {
        private UUID userId;
        private String sessionId;
        private LocalDateTime connectedAt;
        private LocalDateTime lastSeen;
    }
}

@Service
@Slf4j
public class KafkaMessageProducer {

    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    @Value("${application.kafka.topics.messages}")
    private String messagesTopic;

    public void sendMessage(MessageResponse message) {
        try {
            log.debug("Sending message to Kafka - ConversationId: {}, MessageId: {}",
                message.getConversationId(), message.getId());
```

```
            CompletableFuture<SendResult<String, Object>> future =
                kafkaTemplate.send(messagesTopic, message.getConversationId(), message);

            future.whenComplete((result, ex) -> {
                if (ex == null) {
                    log.debug("Successfully sent message to Kafka topic [{}] with key [{}] at offset [{}]",
                        messagesTopic, message.getConversationId(), result.getRecordMetadata().offset());
                } else {
                    log.error("Failed to send message to Kafka topic [{}] with key [{}]: {}",
                        messagesTopic, message.getConversationId(), ex.getMessage(), ex);
                }
            });

        } catch (Exception e) {
            log.error("Error sending message to Kafka - ConversationId: {}, MessageId: {}, Error: {}",
                message.getConversationId(), message.getId(), e.getMessage(), e);
            throw new RuntimeException("Failed to send message to Kafka", e);
        }
    }
}

@Slf4j
@Component
public class KafkaMessageConsumer {

    @Autowired
    private ChatWebSocketHandler webSocketHandler;

    @Autowired
    private ConversationRepository conversationRepository;

    @KafkaListener(topics = "chat.messages")
    public void handleMessage(MessageResponse message) {
        log.info("Received message from Kafka: {}", message.getText());
        try {
            Optional<Conversation> conversationOpt =
conversationRepository.findById(message.getConversationId());

            if (conversationOpt.isPresent()) {
                Conversation conversation = conversationOpt.get();
                conversation.getParticipants().forEach(participantId -> {
                    webSocketHandler.sendMessageToUser(participantId, message);
                });

                log.info("Message broadcasted to {} participants", conversation.getParticipants()
                    .size());
            } else {
                log.error("Conversation not found: {}", message.getConversationId());
            }

        } catch (Exception e) {
            log.error("Error processing message: {}", e.getMessage());
        }
    }
}
```

Додаток Г – Ілюстративна частина

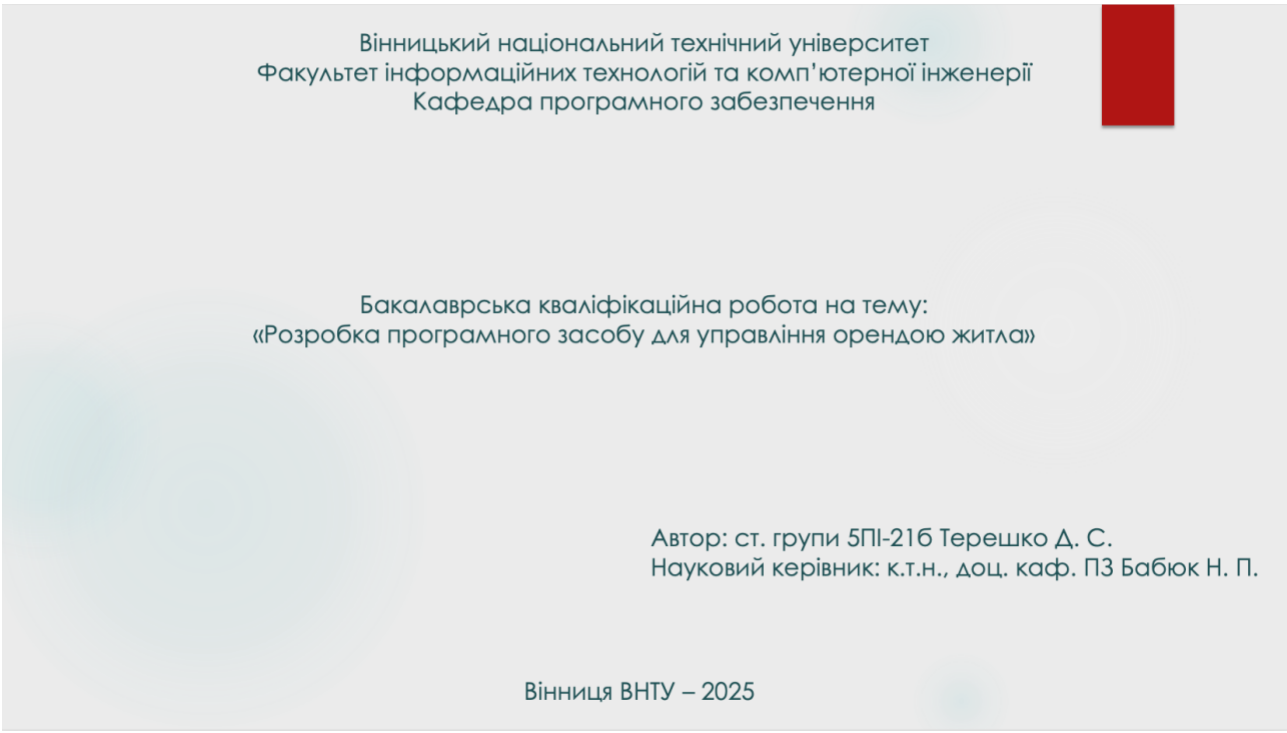

Рисунок Г.1 – Титульний слайд

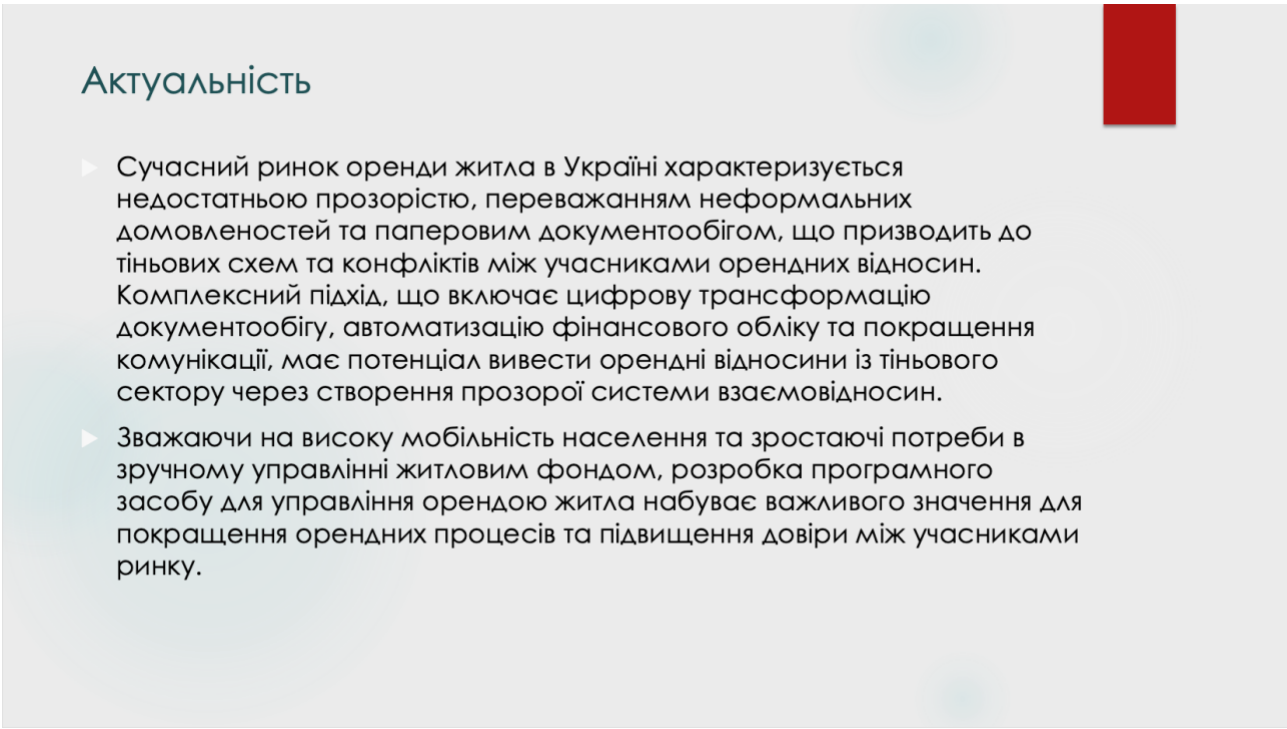

Рисунок Г.2 – Актуальність

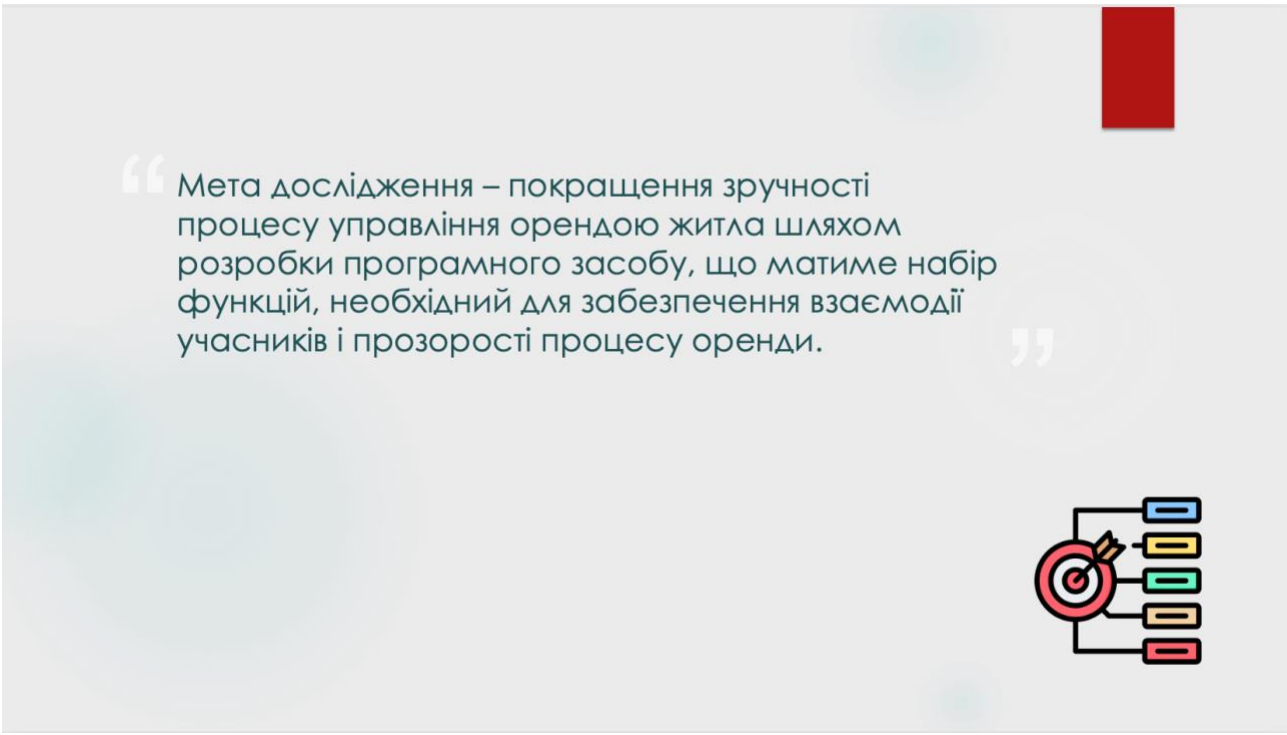

Рисунок Г.3 – Мета дослідження

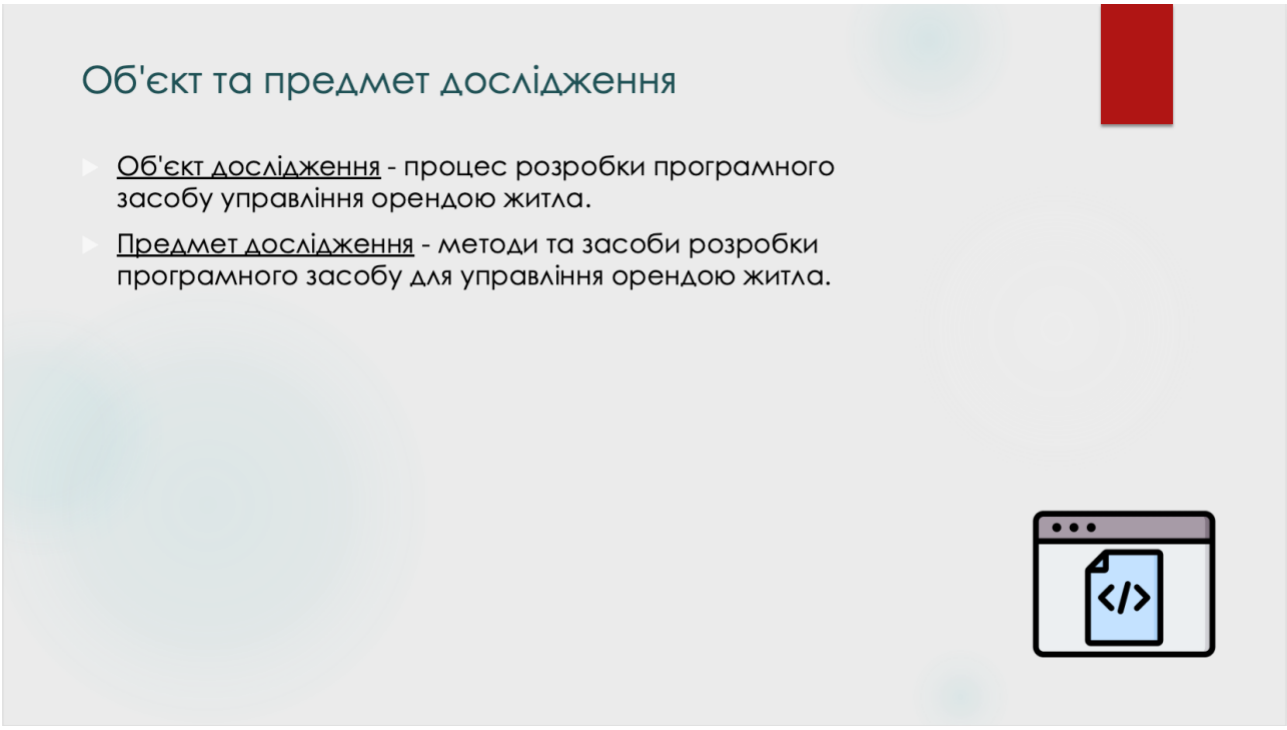

Рисунок Г.4 – Об’єкт та предмет дослідження

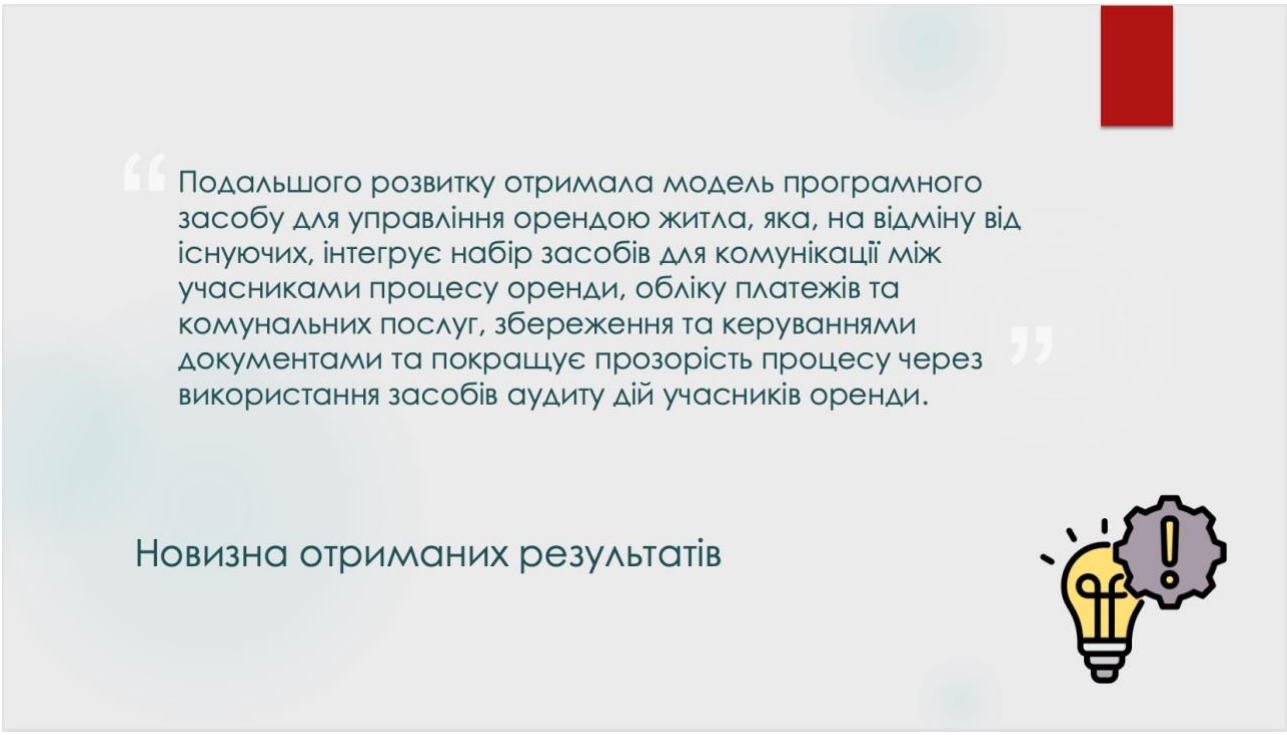

Рисунок Г.5 – Новизна отриманих результатів

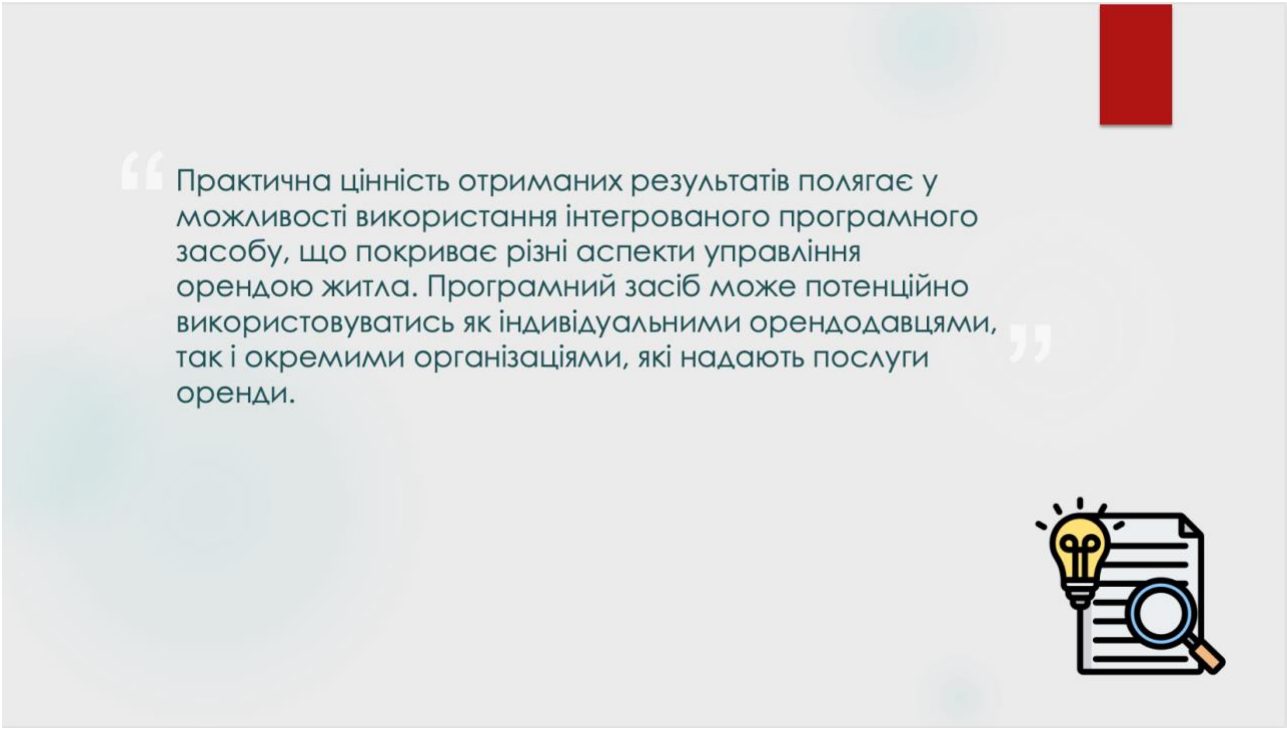

Рисунок Г.6 – Практична цінність отриманих результатів

Задачі дослідження

- - провести аналіз предметної галузі та існуючих програмних засобів для управління орендою житла;
- - розробити модель та алгоритми роботи програмного засобу;
- - розробити модулі програмного засобу для управління орендою житла, що надають функції для документообігу, обліку платежів і комунальних послуг, керування об'єктами оренди, комунікації та керування договорами;
- - розробити вебінтерфейс програмного засобу;
- - провести тестування програмного засобу.

Рисунок Г.7 – Задачі дослідження

Порівняння засобу з аналогами

Критерій	Rentster	Discussio	Pickspace	Rentroom	Власна розробка
Керування об'єктами оренди	+	+	+	+	+
Система збереження і керування документами	+	+	+	+	+
Вбудований чат	-	+	-	+	+
Система структурованих звернень	-	-	+	+	+
Облік платежів	+	+	+	+	+
Облік комунальних послуг	-	+	-	-	+
Аудит дій користувачів	-	-	-	-	+

Рисунок Г.8 – Порівняння засобу з аналогами

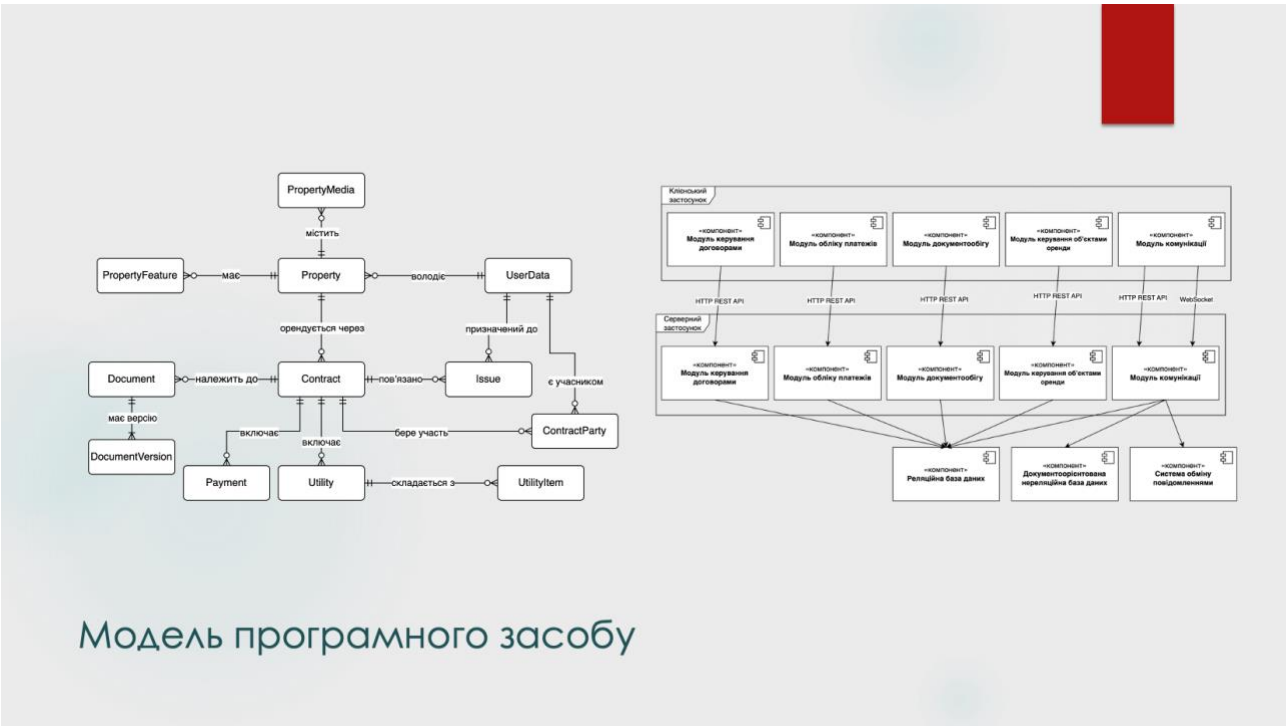

Рисунок Г.9 – Модель програмного засобу

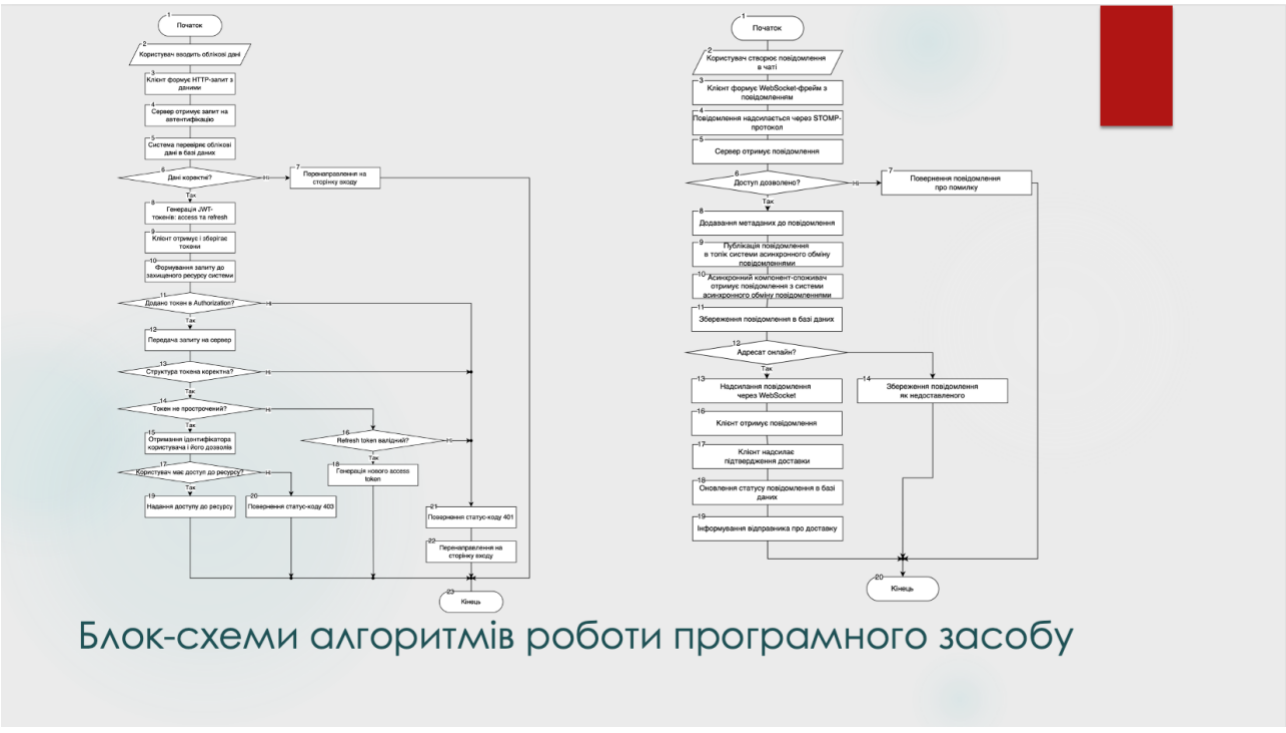

Рисунок Г.10 – Блок-схеми алгоритмів роботи програмного засобу

Рисунок Г.11 – Засоби, що були використані для реалізації програмного застосунку

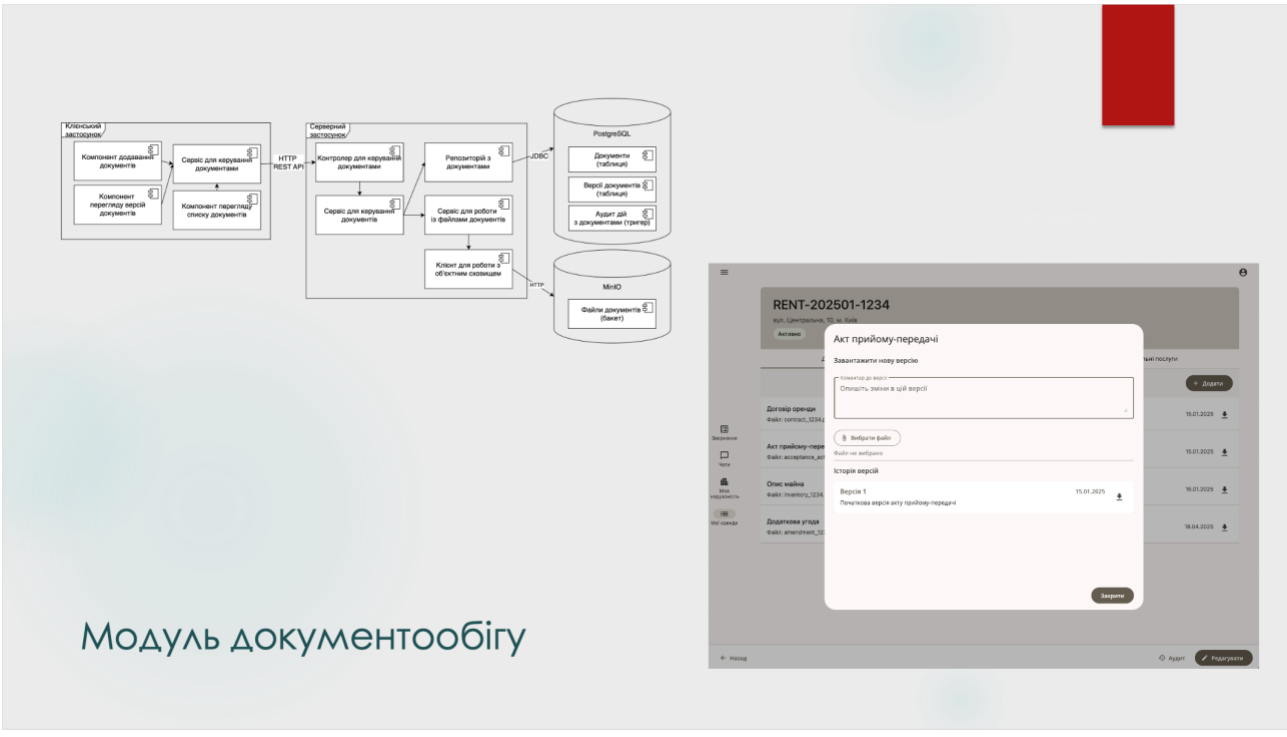

Рисунок Г.12 – Модуль документообігу

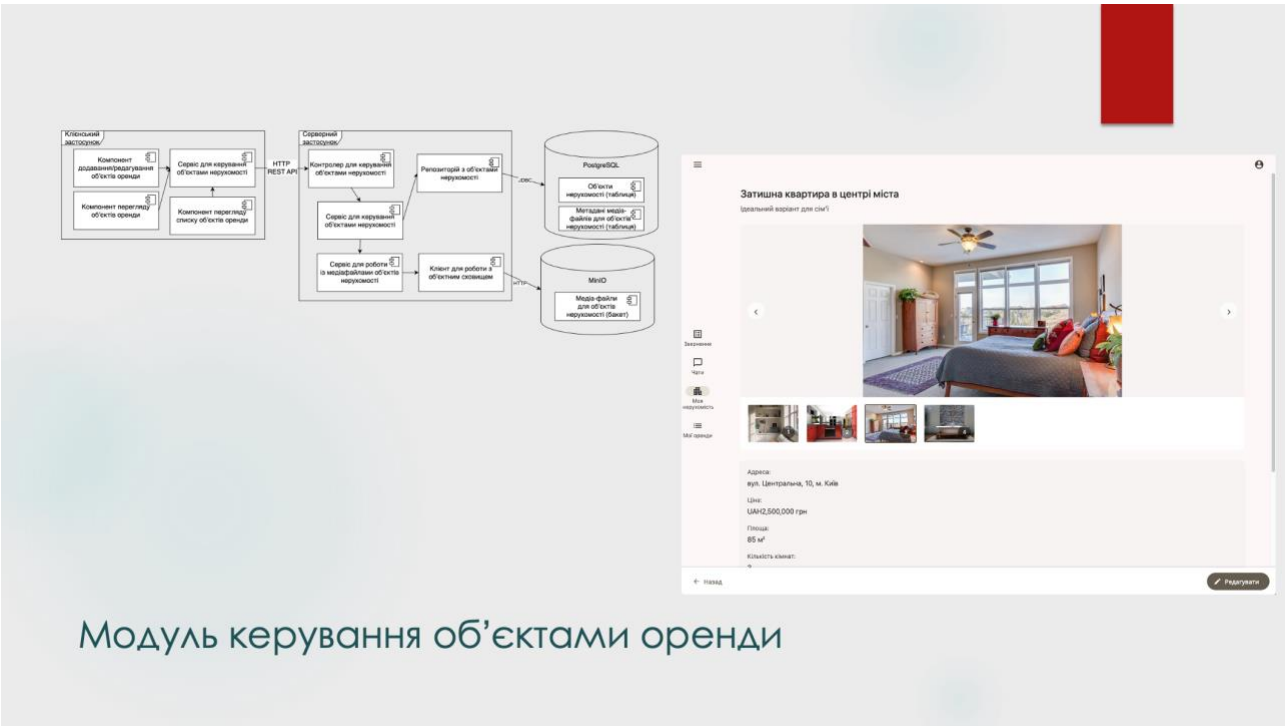

Рисунок Г.13 – Модуль керування об’єктами оренди

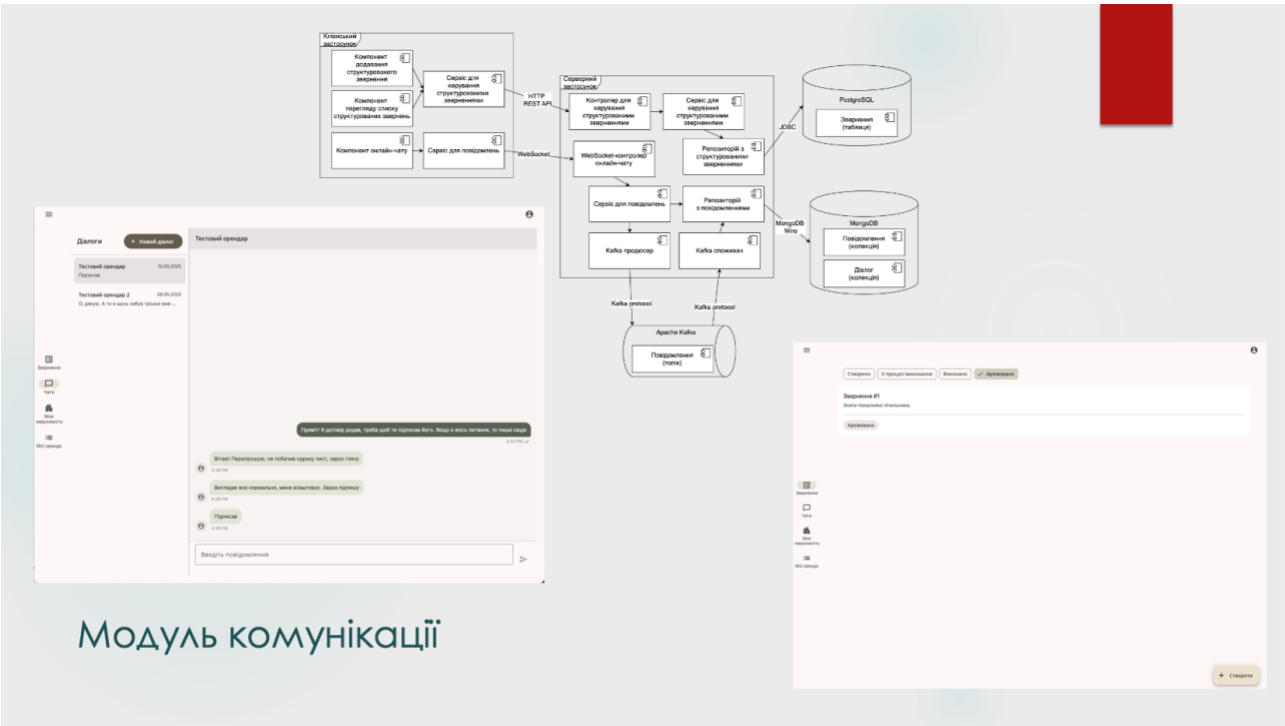

Рисунок Г.14 – Модуль комунікації

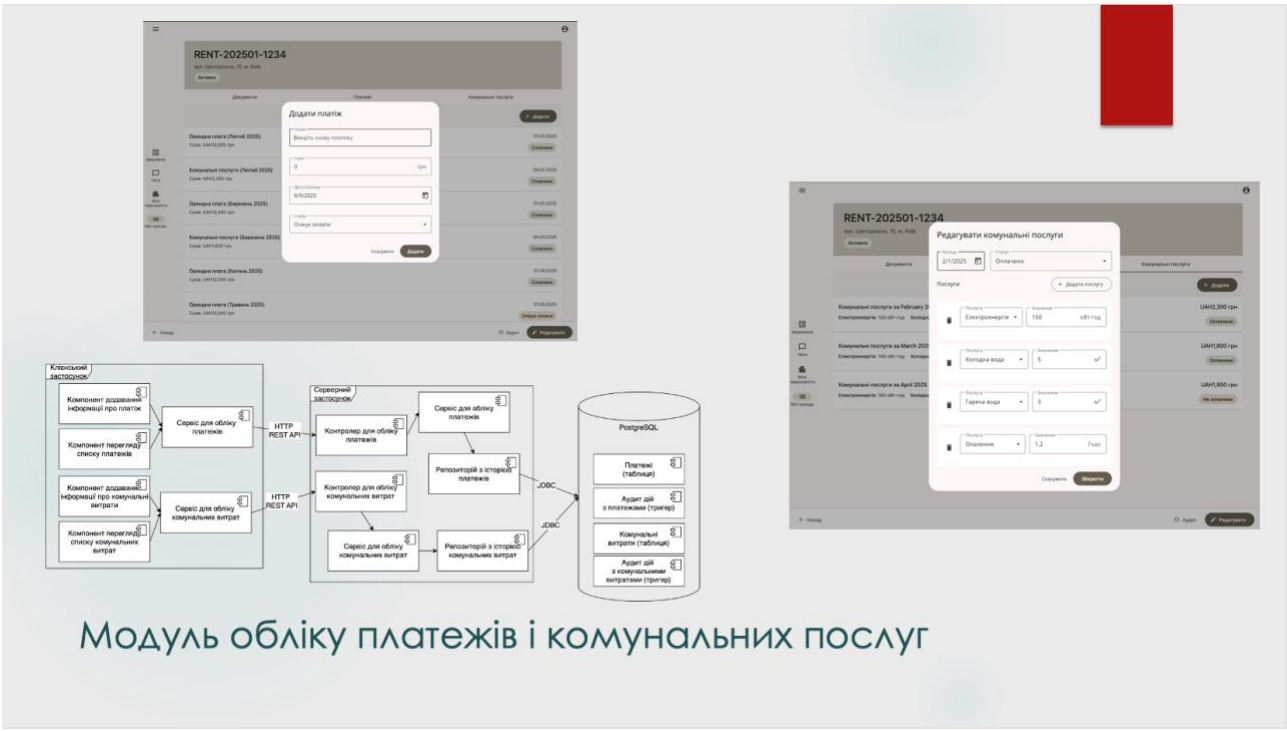

Рисунок Г.15 – Модуль обліку платежів і комунальних послуг

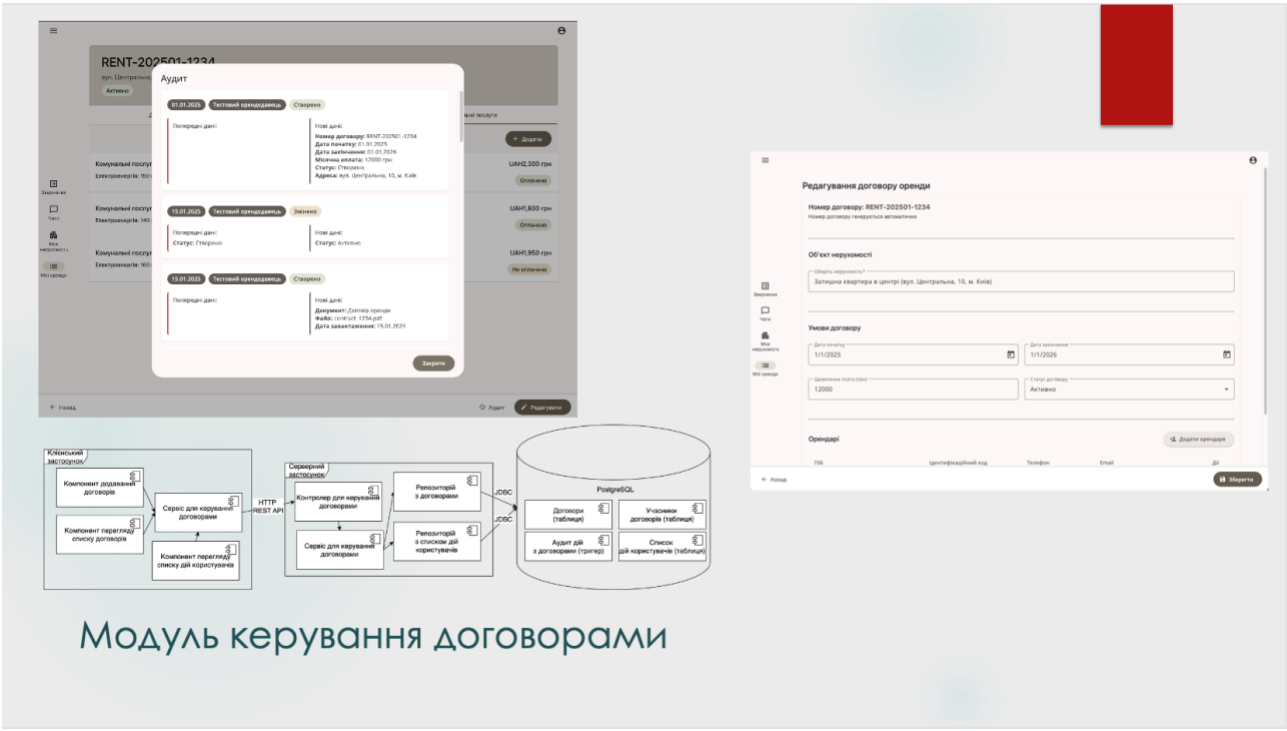

Рисунок Г.16 – Модуль керування договорами

Апробація матеріалів бакалаврської кваліфікаційної роботи

- Основні положення бакалаврської кваліфікаційної роботи були представлені на конференціях «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації» (2024) та «Стан, досягнення і перспективи інформаційних систем і технологій» (2025).


Міністерство освіти і науки України
Одеський національний технологічний університет
Вінницький національний технічний університет
Інститут комп'ютерної інженерії, автоматизації, робототехніки та програмування ім.Н.М.Платонова

МАТЕРІАЛИ

IV ВСЕУКРАЇНСЬКОЇ НАУКОВО – ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ МОЛОДИХ ВЧЕНИХ, АСПІРАНТІВ ТА СТУДЕНТІВ

«КОМП'ЮТЕРНІ ІГРИ І МУЛЬТИМЕДІА ЯК ІННОВАЦІЙНИЙ ПІДХІД ДО КОМУНІКАЦІЇ - 2024»

26-27 вересня 2024 р.
ОДЕСА


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
ННІ Комп'ютерної інженерії, автоматизації робототехніки та програмування ім. Н.М.Платонова
Національний технічний університет України «Київський політехнічний інститут»
Університет Інформатики і прикладних знань м. Лодзь, Польща

XXV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ І ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса
17-18 квітня 2025 р.

Рисунок Г.17 – Апробація матеріалів бакалаврської кваліфікаційної роботи

Висновки

- Під час роботи над бакалаврською кваліфікаційною роботою було розроблено програмний засіб для управління орендою житла.
- У ході роботи було проведено аналіз предметної галузі та існуючих програмних засобів для управління орендою житла, таких як Rentster, Discussio, Pickspace і Rentroom. Використовуючи засоби візуаліації, такі як UML-діаграми та блок-схеми, було розроблено схеми моделі системи та алгоритми роботи деяких алгоритмів.
- Було розроблено модулі програмного засобу для управління орендою житла, що надають функції для документообігу, обліку платежів і комунальних послуг, керування об'єктами оренди, комунікації та керування договорами.
- Було проведено тестування програмного засобу на предмет працездатності і відповідності до поставленого технічного завдання. Окрім цього, було розроблено інструкцію користувача.

Рисунок Г.18 – Висновки

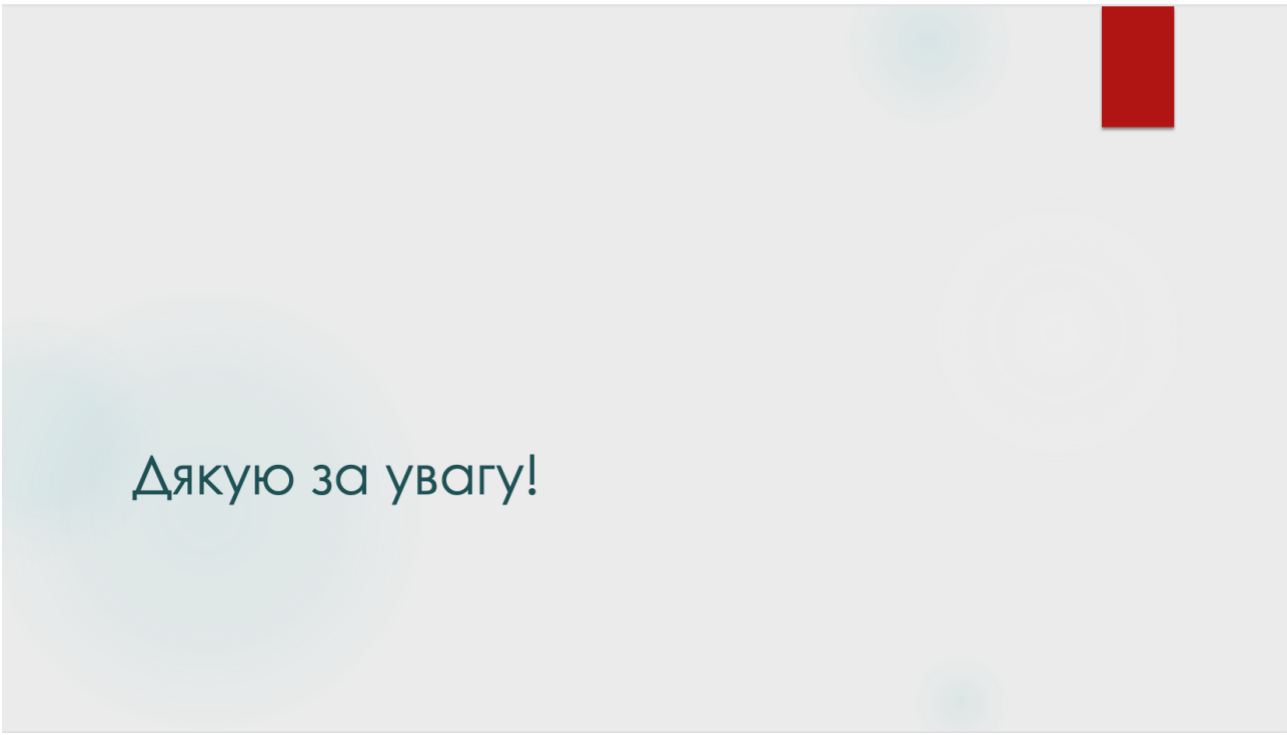

Рисунок Г.19 – Фінальний слайд