МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Зав. кафедри комп'ютерних наук та
прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Проєктування та розробка автоматизованої системи моніторингу використання земельних ресурсів громади

Виконав: Мельничук Олександр Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

група ІПЗ-41

Керівник: к.т.н., доцент Мічута Ольга Романівна _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 57 с., 19 рисунків, 6 таблиць, 12 джерел.

Мета роботи: є підвищення ефективності управління комунальним майном територіальної громади шляхом розробки веб-орієнтованої автоматизованої інформаційної системи.

Предмет дослідження: є моделі, архітектура, алгоритми та програмні засоби для автоматизації обліку, моніторингу та управління комунальним майном.

Об’єкт дослідження: процеси управління комунальним майном у органах місцевого самоврядування.

Методи вивчення: Для вирішення поставлених задач використовувалися: методи системного та порівняльного аналізу для дослідження предметної області; теорія реляційних баз даних та ER-моделювання для проєктування структури даних; об'єктно-орієнтоване проєктування з використанням мови UML для розробки архітектури та моделей системи; методи вебпрограмування для реалізації програмного продукту.

У роботі проведено аналіз існуючих програмних рішень, обґрунтовано вибір технологічного стеку, спроєктовано архітектуру системи та реалізовано ключові функціональні модулі.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗОВАНА СИСТЕМА, ЗЕМЕЛЬНІ РЕСУРСИ, УПРАВЛІННЯ, ASP.NET CORE, POSTGRESQL, REACT, БАЗА ДАНИХ, ДОГОВІРИ ОРЕНДИ, ЗВІТНІСТЬ, ТЕРИТОРІАЛЬНА ГРОМАДА.

ВСТУП

Актуальність теми. В умовах цифрової трансформації та реформи децентралізації в Україні суттєво зростає роль і відповідальність органів місцевого самоврядування (ОМС). Територіальні громади отримали у своє розпорядження значну кількість активів, що включають земельні ділянки, будівлі та споруди, об'єкти інфраструктури, комунальні підприємства та транспортні засоби. Ефективне управління цим різнорідним майном є ключовим фактором для сталого економічного розвитку громади та наповнення місцевого бюджету.

На сьогодні у багатьох громадах процеси обліку, моніторингу договорів оренди та контролю за використанням ресурсів ведуться вручну або за допомогою розрізнених електронних таблиць. Такий підхід призводить до низки проблем: фрагментація та дублювання даних, висока ймовірність людських помилок, відсутність оперативної аналітики та, як наслідок, втрати потенційних надходжень. Тому розробка централізованої автоматизованої системи, що забезпечує комплексний облік, контроль та аналіз використання комунального майна, є надзвичайно актуальним та важливим завданням.

Метою кваліфікаційної роботи є підвищення ефективності процесів управління комунальним майном територіальної громади шляхом проєктування та розробки веб-орієнтованої автоматизованої інформаційної системи.

Для досягнення поставленої мети було визначено наступні задачі дослідження:

1. Провести системний аналіз предметної області управління комунальним майном та чинної нормативно-правової бази.
2. Дослідити існуючі на ринку програмні рішення-аналоги та виконати їх порівняльний аналіз.
3. Обґрунтувати вибір стеку технологій, архітектури та інструментальних засобів для реалізації проєкту.

4. Спроєктувати архітектуру системи, розробити моделі використання та детальну структуру бази даних для зберігання інформації про різнотипні об'єкти майна.
5. Реалізувати ключові програмні модулі системи, включаючи управління даними, пошук та фільтрацію, генерацію документів та інтеграцію з інтерактивною картою.
6. Провести тестування розробленого функціоналу для підтвердження його працездатності та відповідності поставленим вимогам.

Об’єкт дослідження – процеси обліку, моніторингу та управління комунальним майном територіальної громади.

Предмет дослідження – моделі, архітектура, алгоритми та програмні засоби автоматизованої системи управління комунальним майном.

Методи дослідження. Для вирішення поставлених задач використовувалися: методи системного та порівняльного аналізу для дослідження предметної області; теорія реляційних баз даних та ER-моделювання для проєктування структури даних; об'єктно-орієнтоване проєктування з використанням мови UML для розробки архітектури та моделей системи; методи вебпрограмування для реалізації програмного продукту.

Наукова новизна одержаних результатів полягає в удосконаленні підходу до комплексного управління муніципальними активами шляхом розробки єдиної моделі даних, що дозволяє інтегрувати та структурувати інформацію про земельні об'єкти комунальної власності.

Практичне значення одержаних результатів. Розроблений програмний продукт є готовим до впровадження рішенням для відділів комунального майна ОМС. Його використання дозволить автоматизувати рутинні операції, підвищити прозорість обліку, посилити контроль за своєчасністю орендних платежів та строками дії договорів.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз процесів управління комунальним майном та їх правове регулювання

Управління комунальним майном є одним із ключових напрямків діяльності органів місцевого самоврядування (ОМС), що безпосередньо впливає на економічний стан та розвиток інфраструктури територіальної громади. Цей процес є складним, багатоетапним та суворо регламентованим, поєднуючи в собі адміністративні, юридичні та фінансові аспекти. Для ефективного проєктування автоматизованої системи необхідно провести глибокий аналіз існуючих бізнес-процесів («as-is») та нормативно-правової бази, що їх регулює.

1.1.1 Детальний опис бізнес-процесів "як є" в органах місцевого самоврядування (ОМС)

На сьогодні у більшості територіальних громад, зокрема у Здолбунівській ТГ, на основі даних якої проводився аналіз, процеси управління комунальним майном характеризуються значною часткою ручних операцій та використанням розрізнених інструментів обліку. Основними учасниками цих процесів є:

- **Спеціалісти відділу комунального майна/земельних ресурсів:** основні виконавці, відповідальні за ведення реєстрів, підготовку документів та моніторинг.
- **Юридичний відділ:** здійснює правову експертизу договорів та рішень.
- **Бухгалтерія:** відповідає за облік надходжень від орендної плати та контроль заборгованості.
- **Керівництво ОМС та депутати місцевої ради:** приймають фінальні рішення щодо розпорядження майном.
- **Громадяни та юридичні особи:** виступають у ролі заявників, орендарів тощо.

Типовий життєвий цикл управління об'єктом комунальної власності (на прикладі передачі в оренду) можна поділити на такі етапи:

1. **Ініціація та реєстрація звернення.** Фізична або юридична особа подає до ОМС заяву про намір отримати в оренду об'єкт майна. Спеціаліст відділу реєструє заяву в паперовому журналі та перевіряє статус об'єкта у наявних реєстрах (зазвичай, це таблиці MS Excel, окремі для земельних ділянок, будівель, рекламних конструкцій тощо). На цьому етапі виникає ризик втрати даних та ускладнюється пошук актуальної інформації через її фрагментованість.
2. **Обробка та погодження.** Спеціаліст готує проєкт рішення для розгляду на сесії місцевої ради. Документ проходить погодження у юридичному відділі та інших дотичних структурах. Комунікація між відділами часто відбувається через паперовий документообіг, що сповільнює процес та створює залежність від фізичної присутності відповідальних осіб.
3. **Укладення договору.** На підставі позитивного рішення ради укладається договір оренди. Спеціаліст вносить дані договору (терміни, сума оренди, реквізити орендаря) до відповідного Excel-файлу. Відсутність єдиної бази даних призводить до того, що інформація про один і той самий об'єкт може бути внесена з помилками або не повною мірою в різні реєстри (наприклад, у земельному відділі та в бухгалтерії).
4. **Операційний моніторинг.** Це найбільш проблемний етап. Контроль за термінами дії договорів здійснюється спеціалістами вручну. Вони змушені періодично переглядати сотні записів у таблицях, щоб виявити прострочені договори. Це неефективно, призводить до значних затримок у реагуванні та, як наслідок, до прямих фінансових втрат бюджету громади.
5. **Контроль та звітність.** Підготовка комплексних аналітичних звітів для керівництва (наприклад, "Загальна площа земель в оренді", "Сума заборгованості по всіх договорах") є вкрай трудомістким процесом, що вимагає ручного зведення даних з численних файлів.

Як ми зрозуміли з цього життєвого циклу, процес є надмірно бюрократизованим та залежним від ручних операцій, що підтверджує гостру необхідність його автоматизації.

1.1.2 Огляд нормативно-правової бази

Діяльність ОМС у сфері управління комунальним майном суворо регламентована чинним законодавством України. Розроблювана автоматизована система повинна враховувати та відповідати вимогам ключових нормативно-правових актів.

Основними документами, що регулюють цю сферу, є:

- **Закон України «Про місцеве самоврядування в Україні».** Цей закон є основоположним, оскільки він визначає статус та повноваження місцевих рад. Згідно зі **статтею 60**, до об'єктів права комунальної власності належить рухоме і нерухоме майно, доходи місцевих бюджетів, земля, природні ресурси та інші активи. **Стаття 26** закріплює виключне право сільських, селищних та міських рад приймати на сесіях рішення щодо розпорядження об'єктами комунальної власності, встановлення ставок земельного податку та орендної плати. [2]
- **Земельний кодекс України.** Регулює правовідносини, пов'язані з використанням та охороною земель. **Стаття 83** визначає, які землі можуть перебувати у комунальній власності. Ключовими для даного проєкту є **статті 93** та **124**, які регламентують порядок передачі земельних ділянок в оренду, зокрема на конкурентних засадах (земельних торгах), а також визначають істотні умови договору оренди землі. [1]
- **Закон України «Про оренду державного та комунального майна».** Цей закон встановлює правові та організаційні засади передачі в оренду майна, що перебуває в комунальній власності. Він визначає об'єкти та суб'єкти оренди, порядок проведення аукціонів через електронну торгову систему (Prozorro.Продажі), а також істотні умови договору оренди (об'єкт, строк,

орендна плата, порядок її індексації тощо), які є обов'язковими для фіксації в автоматизованій системі. [3]

- **Цивільний та Господарський кодекси України.** Вони встановлюють загальні засади укладання договорів, визначають поняття зобов'язання, відповідальності за його порушення, а також регламентують строки позовної давності, що важливо для коректної реалізації функціоналу відстеження заборгованості.

Таким чином, аналіз нормативно-правової бази дозволяє сформувати чіткі вимоги до атрибутивного складу сутностей в базі даних майбутньої системи (наприклад, які поля повинні бути в картці договору) та до логіки роботи її модулів (наприклад, алгоритми розрахунку пені, правила формування звітів).

1.1.3 Класифікація об'єктів комунального майна та специфіка управління ними

Комунальне майно територіальної громади є неоднорідним за своєю природою, складом та призначенням. Аналіз реєстрів, наданих Здолбунівською ТГ, дозволяє класифікувати наявні активи на декілька ключових груп, кожна з яких має унікальні атрибути та вимагає специфічного підходу до управління.

1. **Нерухоме майно.** Це найбільш капіталомістка та значуща частина активів громади.
    - **Земельні ділянки:** Основні атрибути – кадастровий номер, площа, місцезнаходження, цільове призначення (напр., для сільського господарства, житлової забудови, комерції). Управління ними полягає в укладанні договорів оренди, контролі за сплатою орендної плати та земельного податку, а також моніторингу дотримання умов цільового використання.
    - **Будівлі та споруди:** Характеризуються адресою, площею, поверховістю, технічним станом, балансовою вартістю. Управління включає передачу в оренду цілих об'єктів або їх частин (приміщень),

облік витрат на утримання та ремонт, контроль за комунальними платежами.

- **Об'єкти культурної спадщини:** Особлива категорія нерухомого майна, що вимагає додаткового контролю. Окрім стандартних атрибутів, для них є обов'язковим ведення охоронних договорів, що накладають на орендаря значні обмеження щодо використання та реконструкції об'єкта.

2. **Інфраструктурні та інші об'єкти.**
    - **Об'єкти зовнішньої реклами:** Це рекламні щити, білборди, сітілайти. Ключовими параметрами є місце розташування, тип конструкції, площа рекламної поверхні. Управління полягає не в оренді, а у видачі дозволів на розміщення та зборі відповідної плати до місцевого бюджету.
    - **Майданчики для паркування:** Обліковуються за адресою та площею. Управління може здійснюватися як напряму комунальним підприємством, так і через передачу в експлуатацію приватним операторам.
3. **Рухоме майно.**
    - **Транспортні засоби:** Характеризуються маркою, моделлю, роком випуску, номерним знаком, призначенням. Управління включає закріплення за конкретним комунальним підприємством чи установою, контроль технічного стану, страхування, облік паливно-мастильних матеріалів.
4. **Цілісні майнові комплекси.**
    - **Комунальні підприємства, установи та організації:** Це юридичні особи, засновані громадою. Управління ними полягає не в прямому розпорядженні їх майном, а в корпоративному контролі: призначенні керівників, аналізі фінансових планів та звітів, контролі за ефективністю використання майна, переданого їм на баланс.
5. **Допоміжні об'єкти обліку.**

- **Лічильники:** Не є самостійними об'єктами управління, але є критично важливими для обліку. Вони прив'язуються до об'єктів нерухомості (будівель, приміщень) для контролю за споживанням ресурсів (вода, електроенергія, тепло), що необхідно для розрахунків з орендарями або обліку витрат на утримання.

Така диверсифікація об'єктів доводить недієвість уніфікованих підходів до обліку на базі електронних таблиць. Ефективне управління вимагає гнучкої реляційної моделі даних, здатної відобразити всі ці об'єкти та складні зв'язки між ними.

1.1.4 Виявлення та опис ключових проблем і "вузьких місць" ручного обліку

Аналіз бізнес-процесів "як є" дозволяє ідентифікувати низку системних проблем, що безпосередньо випливають з ручного або напівавтоматизованого способу ведення обліку. Ці проблеми створюють ризики як фінансового, так і управлінського характеру.

- **Фрагментація та ризик втрати даних.** Інформація про комунальне майно розпорошена між різними відділами (земельний, юридичний, бухгалтерія), зберігається у різних форматах (MS Excel, MS Word, паперові архіви) та на локальних комп'ютерах співробітників. Це створює ситуацію, коли не існує єдиного, достовірного джерела даних (“single source of truth”). Оновлення інформації в одному місці не гарантує її оновлення в іншому, а відсутність централізованого резервного копіювання створює постійний ризик безповоротної втрати даних через збій обладнання чи людську помилку.
- **Людський фактор та неминучість помилок.** Ручне введення даних є головним джерелом помилок. Одруківки в кадастровому номері, прізвищі орендаря, даті укладення чи закінчення договору можуть призвести до серйозних юридичних та фінансових наслідків. Наприклад, помилка в даті закінчення договору може стати

причиною того, що договір не буде вчасно пролонговано або розірвано, що призведе до втрати контролю над активом. Електронні таблиці, на відміну від спеціалізованої бази даних, не мають вбудованих механізмів валідації даних, що лише посилює цю проблему.

- **Складність та низька оперативність у підготовці звітів.** Формування будь-якого комплексного звіту (напр., про загальну суму заборгованості по орендній платі або перелік договорів, що закінчуються у наступному кварталі) перетворюється на довготривалий та трудомісткий процес. Спеціаліст змушений вручну збирати, зводити та перевіряти дані з десятків різних файлів. Це може займати від кількох годин до кількох днів, тоді як в автоматизованій системі такий звіт генерується за лічені секунди.
- **Відсутність прозорості та складнощі контролю.** Існуюча система не дозволяє керівництву громади оперативно отримувати цілісну картину стану комунального майна. Неможливо швидко відповісти на питання: "Яка загальна площа комерційних приміщень, що зараз вільні?" або "Яка динаміка надходжень від оренди землі за останній рік?". Крім того, відсутність аудиторського сліду (хто, коли і які зміни вносив до запису) робить систему непрозорою та створює умови для зловживань чи приховування помилок.

Сукупність цих проблем доводить, що існуюча система обліку є неефективною, ризикованою та не відповідає сучасним вимогам до управління. Впровадження централізованої автоматизованої системи є єдиним шляхом для їх комплексного вирішення.

1.2. Огляд та порівняльний аналіз існуючих програмних рішень

Для обґрунтування доцільності розробки нової автоматизованої системи необхідно провести аналіз існуючих на ринку програмних продуктів, призначених для вирішення подібних завдань. Ринок пропонує низку рішень, що

відрізняються за функціональністю, технологічною платформою, вартістю та ступенем адаптації до українського законодавства. Розглянемо декілька характерних прикладів, що представляють різні сегменти ринку.

1.2.1 Дослідження існуючих систем

1. Програмний комплекс «Реєстр територіальної громади» (Компанія «СофтПро», Україна)

Це комплексна геоінформаційна система (ГІС), розроблена для створення єдиного інформаційного простору та муніципального кадастру територіальної громади. «СофтПро» є одним з провідних українських розробників у сфері земельного кадастру та містобудування.

- **Основні можливості:**
  - Ведення реєстру земельних ділянок, об'єктів нерухомості, інженерних мереж.
  - Інтеграція з Публічною кадастровою картою України та іншими державними реєстрами.
  - Потужні інструменти для просторового аналізу та візуалізації даних на карті.
  - Формування різноманітної технічної документації та звітів.
- **Технологічна платформа:** Зазвичай реалізується як клієнт-серверне або веб-орієнтоване рішення.
- **Позиціонування:** Це потужна, ентерпрайз-рівня система, орієнтована на великі міста та громади з високими вимогами до ГІС-функціоналу. Її впровадження та підтримка вимагають значних фінансових та кадрових ресурсів.

2. Програмний продукт «Майно комунальної громади» (Компанія «ІТС», Україна)

Дане рішення орієнтоване в першу чергу на бухгалтерський та фінансовий облік комунальних активів. Продукт тісно інтегрується з популярними системами для ведення бухгалтерії, такими як BAS.

- **Основні можливості:**
    - Детальний облік основних засобів: балансова вартість, нарахування амортизації, інвентаризація.
    - Ведення договорів оренди, автоматичне нарахування орендної плати та пені.
    - Формування фінансової та бухгалтерської звітності.
    - Облік рухомого та нерухомого майна, що знаходиться на балансі комунальних підприємств.
- **Технологічна платформа:** Переважно є десктопним клієнт-серверним додатком.
- **Позиціонування:** Продукт є ефективним інструментом для бухгалтерії та фінансових відділів. Однак, його ГІС-можливості є обмеженими або реалізуються через інтеграцію зі сторонніми системами, а інтерфейс може бути менш гнучким у порівнянні з сучасними веб-додатками.

3. Accela Civic Platform (Компанія Accela, США)

Це одна з провідних світових хмарних платформ для автоматизації діяльності державних установ. Модуль *Land Management* (Управління земельними ресурсами) пропонує комплексний підхід до управління територіальным розвитком.

- **Основні можливості:**
    - Повний життєвий цикл управління дозволами, ліцензіями та інспекціями.
    - Публічні портали для громадян та бізнесу для подачі онлайн-заявок.
    - Мобільні додатки для інспекторів, що працюють "у полі".
    - Гнучка система налаштування робочих процесів (workflows).
    - Хмаркова архітектура (SaaS), що забезпечує високу доступність та надійність.
- **Технологічна платформа:** Сучасна хмарна SaaS-платформа.
- **Позиціонування:** Є надзвичайно потужним та гнучким рішенням, але має два суттєві недоліки для українського ринку: висока вартість ліцензування

та повна невідповідність українським законодавчим та адміністративним реаліям "з коробки", що вимагало б дороговартісної кастомізації.

Проведений огляд показує, що на ринку існують як комплексні дорогі системи, так і вузькоспеціалізовані продукти. Це створює нішу для розробки гнучкого, сучасного веб-рішення, що було б адаптоване до потреб українських громад та мало б збалансовану вартість.

1.2.2 Створення детальної порівняльної таблиці (матриці)

Для наочного порівняння розглянутих програмних рішень та визначення їх сильних і слабких сторін відносно потреб проєкту, зведемо ключові характеристики у порівняльну таблицю (табл. 1.1).

Таблиця 1.1

Порівняльний аналіз існуючих програмних рішень

Критерій	Комплекс "Реєстр ТГ" (СофтПро)	ПП "Майно ком. громади" (ІТС)	Accela Civic Platform	QGIS
Функціональні можливості	Ведення реєстрів (земля, нерухомість) Управління договорами Дуже сильний ГІС-модуль Формування звітів	Дуже сильний бух. Облік Управління договорами Слабкий або відсутній ГІС-модуль Фінансова звітність	Комплексне управління (дозволи, ліцензії) Сучасний ГІС та мобільний доступ Публічні портали для громадян Гнучкі звіти та аналітика	Лише ГІС-функціонал Відсутня логіка управління договорами, фінансами, звітами "з коробки"
Стек технологій	Переважно клієнт-сервер, частково веб	Переважно десктопний клієнт-сервер	Сучасна хмарна платформа (SaaS)	Десктопний додаток, відкритий код (C++, Python)
Модель розповсюдження	Коробкове рішення,	Коробкове рішення, ліцензування	SaaS (програмне забезпечення як	FOSS (вільне ПЗ з

	впровадження "під ключ"		послуга), підписка	відкритим кодом)
Орієнтовна вартість	Висока (десятки тисяч доларів за впровадження)	Середня / Висока	Дуже висока (корпоративний сегмент)	Безкоштовно (вартість розробки та підтримки)
Переваги	Комплексність Адаптація до законодавства України Потужна геоаналітика	Глибока інтеграція з бухгалтерією Сильний фінансовий модуль	Сучасність, гнучкість, масштабованість Найширший функціонал	Нульова вартість ліцензії Абсолютна гнучкість Потужні ГІС-інструменти
Недоліки	• Висока вартість та складність Монолітність, повільне оновлення	Застаріла архітектура Слабкий ГІС Орієнтація на бухгалтерів, а не на управлінців	Неадаптованість до реалій України Надзвичайно висока вартість Залежність від постачальника	Не є готовою системою Вимагає розробки всієї бізнес-логіки з нуля

Як видно з таблиці, жодне з розглянутих рішень не є оптимальним для потреб середніх та малих громад України, які потребують збалансованого, сучасного та економічно доступного інструменту.

1.2.3 Висновок про доцільність розробки власної системи

Проведений аналіз ринку програмних рішень для управління комунальним майном демонструє, що існуючі пропозиції є поляризованими. З одного боку, на ринку присутні потужні, але дорогі та складні комерційні системи. З іншого – вільні інструменти, які не є готовими рішеннями і вимагають значних ресурсів на розробку та адаптацію.

Українські комерційні продукти («Реєстр ТГ», «Майно комунальної громади») хоч і адаптовані до місцевого законодавства, часто базуються на застарілих технологічних платформах (десктоп, клієнт-сервер), є недостатньо гнучкими та мають високий поріг входження через значну вартість ліцензій та впровадження. Міжнародні аналоги, як Accela, попри свою технологічну

досконаліть, є неприйнятними через надвисоку вартість та фундаментальну невідповідність адміністративним процесам і правовому полю України.

Використання таких інструментів, як QGIS, є безкоштовною альтернативою лише на перший погляд, оскільки вартість розробки повноцінної системи з нуля силами кваліфікованих спеціалістів може бути співмірною з вартістю готового комерційного продукту.

Отже, існує чітко окреслена незайнята ніша для програмного продукту, який би поєднав у собі ключові переваги різних підходів:

- **Функціональна повнота:** інтеграція найважливіших модулів (реєстр об'єктів, договори, фінанси, звіти) з сучасним ГІС-інтерфейсом.
- **Сучасна архітектура:** розробка на гнучкому та масштабованому веб-стеку (ASP.NET Core), що забезпечить доступність через браузер, простоту оновлення та інтеграції.
- **Адаптованість:** повна відповідність вимогам українського законодавства та реальним бізнес-процесам ОМС.
- **Економічна доступність:** потенційно значно нижча вартість володіння у порівнянні з існуючими комерційними монополістами.

Таким чином, розробка власної автоматизованої системи є не тільки доцільною, але й стратегічно виправданою. Вона дозволяє створити цільовий, ефективний та економічно доступний інструмент, що повністю відповідає потребам сучасних українських громад та позбавлений недоліків, притаманних існуючим на ринку рішенням.

Сучасні багатокористувацькі ігри вимагають ефективних і надійних мережевих рішень, щоб забезпечити стабільний ігровий процес і якісний досвід для гравців. Різні інструменти та платформи надають різноманітні можливості для розробників ігор, дозволяючи їм обирати найбільш підходящі рішення залежно від специфіки проєкту. Розглянемо деякі з найбільш популярних і широко використовуваних мережевих рішень для багатокористувацьких ігор.

1.3 Обґрунтування вибору технологій та архітектурних рішень

Вибір правильного стеку технологій є фундаментальним етапом проєктування, що безпосередньо впливає на швидкість розробки, надійність, масштабованість та вартість підтримки майбутньої системи. Рішення приймалися на основі аналізу альтернатив та з урахуванням специфічних вимог до системи управління комунальним майном.

1.3.1 Вибір технології для розробки серверної частини (Backend)

Серверна частина (бекенд) відповідає за всю бізнес-логіку, роботу з базою даних та безпеку. Розглядалися три провідні платформи: ASP.NET Core, Node.js та Python (Django).

- **Node.js** – це середовище виконання JavaScript, відоме своєю асинхронною моделлю та високою продуктивністю в I/O-операціях. Це робить його чудовим вибором для чатів та застосунків реального часу, однак для системи з великою кількістю складної бізнес-логіки динамічна типізація JavaScript може призводити до помилок на етапі виконання.
    - **Python** (з фреймворками Django/Flask) – надзвичайно популярний завдяки простоті синтаксису та величезній кількості бібліотек. Django пропонує підхід "все включено" для швидкої розробки, але може поступатися у чистій продуктивності більш компільованим платформам.
    - **ASP.NET Core** – це сучасна, кросплатформна та високопродуктивна платформа від Microsoft. Саме її було обрано для реалізації проєкту з наступних причин:
        - **Продуктивність:** ASP.NET Core є однією з найшвидших веб-платформ у світі, що критично важливо для системи, яка буде обробляти великі обсяги даних та запитів.
        - **Строга типізація мови C#:** На відміну від JavaScript чи Python, C# є мовою зі строгою статичною типізацією. Це дозволяє виявляти велику кількість помилок ще на етапі компіляції, значно

підвищує надійність коду та спрощує його підтримку і рефакторинг у довгостроковій перспективі.

- **Інтегрована екосистема:** Платформа .NET надає потужний та перевірений набір інструментів "з коробки": Entity Framework Core для роботи з базами даних, вбудовані механізми автентифікації та авторизації, система впровадження залежностей (Dependency Injection). Це створює цілісне та надійне середовище для розробки складних систем.

Для розробки надійного та масштабованого корпоративного застосунку, яким є система управління майном, поєднання продуктивності, надійності та потужної екосистеми робить ASP.NET Core оптимальним вибором.

1.3.2 Вибір архітектури клієнтської частини (Frontend)

Для реалізації інтерфейсу користувача розглядалися два основні підходи: створення односторінкового застосунку (SPA) та використання класичної архітектури Model-View-Controller (MVC).

- **SPA (Single-Page Application)**, реалізований на фреймворках як React, Angular чи Blazor WebAssembly, забезпечує високу інтерактивність, подібну до десктопних додатків. Сервер у такому випадку віддає лише дані через API, а вся логіка відображення виконується у браузері.
- **Архітектура MVC** передбачає, що сервер генерує готові HTML-сторінки та відправляє їх клієнту. Цей підхід є більш традиційним, але залишається надзвичайно ефективним для певного класу систем.

Для даного проєкту було обрано архітектуру MVC (з використанням Razor Pages) з наступних причин:

- **Перевага серверної логіки:** Проєкт містить значну кількість бізнес-логіки (перевірка умов договорів, розрахунки, валідація), яка має виконуватися на сервері. Архітектура MVC дозволяє тримати цю логіку впорядкованою та під повним контролем.

- **Генерація документів:** Однією з ключових функцій системи є створення PDF-документів (повідомлень, звітів). Ця операція є суто серверною. В архітектурі MVC контролер може легко зібрати необхідні дані, згенерувати файл і віддати його користувачу єдиним потоком, що значно простіше, ніж координувати цей процес у SPA.
- **Простота розробки та підтримки:** Для системи, що орієнтована на відображення даних та форм, а не на складні анімації, підхід MVC є простішим та швидшим в реалізації.

Таким чином, архітектура MVC є надійним та прагматичним вибором, що ідеально відповідає вимогам системи з інтенсивною обробкою даних та генерацією документів на сервері.

1.3.3 Вибір системи управління базами даних (СУБД)

База даних є ядром системи, тому вибір СУБД має критичне значення. Аналізувалися три популярні реляційні СУБД: MySQL, MS SQL Server та PostgreSQL.

- **MySQL** – найпопулярніша у світі СУБД з відкритим кодом, відома своєю швидкістю та надійністю, є гарним вибором для багатьох веб-проєктів.
- **MS SQL Server** – потужна комерційна СУБД від Microsoft, що має чудову продуктивність та тісну інтеграцію з екосистемою .NET. Однак вартість ліцензування може бути перешкодою.
- **PostgreSQL** – це об'єктно-реляційна СУБД з відкритим кодом, яка здобула репутацію найнадійнішої та найбільш функціональної серед безкоштовних аналогів.

Вибір було зроблено на користь PostgreSQL з таких причин:

- **Відкритий код та вартість:** PostgreSQL є повністю безкоштовною, що виключає будь-які витрати на ліцензування для громади.
- **Надійність та відповідність стандартам:** Вона відома своєю винятковою надійністю, транзакційною цілісністю та суворим дотриманням стандарту SQL.

Поєднання нульової вартості, високої надійності та унікальних геопросторових можливостей робить PostgreSQL ідеальним фундаментом для розроблюваної системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ МОНІТОРИНГУ ВИКОРИСТАННЯ ЗЕМЕЛЬНИХ РЕСУРСІВ ГРОМАДИ

2.1. Формування вимог до системи на основі проведеного аналізу

Розробка функціональних вимог до системи базується на двох основних джерелах: аналізі бізнес-процесів ОМС («as-is») та висновках, зроблених під час огляду існуючих програмних рішень. Мета полягає не в копіюванні функціоналу аналогів, а в створенні збалансованого продукту, який поєднує в собі сильні сторони ринкових лідерів, але при цьому є доступним, гнучким та адаптованим до реалій українських громад.

В системі передбачено дві основні ролі з чітким розмежуванням повноважень:

- **Адміністратор** — роль з повними правами на створення та модифікацію даних. Відповідає за наповнення системи, актуальність та цілісність інформації.
- **Користувач** — роль з правом перегляду даних, їх аналізу та формування звітності. Відповідає за моніторинг, контроль та використання інформації для прийняття управлінських рішень.

Вимоги сформульовані у форматі User Stories, що дозволяє чітко визначити цінність кожної функції для кожної ролі.

2.1.1 Функціональні вимоги до системи

Роль: Адміністратор (Управління даними)

- Як Адміністратор, я хочу мати повні права на створення, редагування та архівування всіх об'єктів майна (землі, будівель, транспорту), щоб централізовано та одноосібно відповідати за якість даних у системі.
- Як Адміністратор, я хочу створювати та редагувати договори, прив'язуючи їх до об'єктів та орендарів, щоб вести повний та достовірний облік всіх договірних відносин громади.
- Як Адміністратор, я хочу вносити та оновлювати інформацію про фінансові операції (надходження орендної плати), щоб система містила актуальні дані для фінансового моніторингу, який буде проводити Користувач.

- Як Адміністратор, я хочу керувати обліковими записами Користувачів (створення, блокування), щоб контролювати доступ до інформації в системі.
- Як Адміністратор, я хочу завантажувати та керувати шаблонами документів, щоб забезпечити єдиний стандарт для всіх звітів та повідомлень, які генерують Користувачі.

Роль: Користувач (Аналіз та моніторинг)

- Як Користувач, я хочу мати доступ до перегляду всіх даних про майно та договори у режимі "тільки читання", щоб бути впевненим, що я не зможу випадково пошкодити чи змінити офіційні дані.
- Як Користувач, я хочу використовувати потужну систему пошуку та фільтрації, щоб швидко знаходити потрібні об'єкти чи договори за будь-якими параметрами (статус, тип, дата, орендар).
- Як Користувач, я хочу бачити на головній панелі інструментів (Dashboard) окремий віджет, що автоматично показує список договорів, термін дії яких закінчується у найближчі 30/60/90 днів, щоб проактивно реагувати та ініціювати процедуру їх пролонгації чи розірвання.
- Як Користувач, я хочу мати можливість відфільтрувати всіх боржників одним кліком та згенерувати для них стандартні листи-повідомлення, щоб ефективно вести претензійну роботу.
- Як Користувач, я хочу генерувати стандартні аналітичні звіти ("Надходження за період", "Вільні об'єкти для оренди", "Структура майна громади"), щоб готувати інформацію для керівництва та сесій ради.
- Як Користувач, я хочу мати можливість експортувати результати пошуку та звіти у формат CSV/Excel, щоб проводити поглиблений аналіз даних у зовнішніх програмах.
- Як Користувач, я хочу мати можливість додавати власні коментарі чи нотатки до об'єктів або договорів (наприклад, "Орендар обіцяв сплатити

до 15.06"), які будуть видимі іншим Користувачам, щоб полегшити внутрішню комунікацію без зміни основних даних.

2.1.2 Нефункціональні вимоги до системи

Якщо функціональні вимоги визначають, що система має робити, то нефункціональні вимоги визначають, як вона повинна це робити. Вони встановлюють критерії якості, які є не менш важливими для успішного впровадження та експлуатації системи, особливо при роботі з офіційними даними в державному секторі.

- **Продуктивність**. Система повинна забезпечувати швидку та комфортну взаємодію для користувачів, щоб уникнути простоїв у роботі.
    - Час відповіді інтерфейсу: Завантаження ключових сторінок, таких як реєстр об'єктів або список договорів, при типовому навантаженні (до 10 000 записів) не повинно перевищувати 2 секунди.
    - Швидкість обробки запитів: Виконання операцій пошуку та фільтрації за кількома критеріями має надавати результат протягом 1-3 секунд.
    - Генерація звітів: Створення стандартних аналітичних звітів або експорт даних у CSV/Excel обсягом до 500 записів має виконуватися не довше 15 секунд.
- **Надійність.** Система має бути стабільною, доступною протягом усього робочого дня та гарантувати цілісність даних.
    - Рівень доступності (Uptime): Система повинна бути доступною не менше 99.5% часу протягом робочих годин (з 8:00 до 18:00). Планові оновлення та технічне обслуговування мають проводитися у неробочий час.
    - Транзакційна цілісність: Всі операції, що модифікують дані (створення, редагування, видалення), які виконує Адміністратор, повинні бути атомарними та виконуватися в рамках транзакцій. Це

гарантує, що у разі збою часткові зміни не потраплять до бази даних.

- Резервне копіювання: Повинна бути налаштована система щоденного автоматичного резервного копіювання бази даних для можливості швидкого відновлення у разі апаратного збою чи іншого форс-мажору.

- **Безпека.** Захист даних є найвищим пріоритетом, оскільки система оперує офіційною інформацією громади.
  - Автентифікація та авторизація: Доступ до системи має здійснюватися виключно за логіном та паролем. Всі паролі користувачів повинні зберігатися в базі даних тільки у хешованому вигляді з використанням криптографічної солі (наприклад, за допомогою сучасного алгоритму BCrypt або Identity Framework), що унеможливлює їх відновлення навіть у разі витоку даних.
  - Розмежування доступу: Система повинна суворо дотримуватися дворольової моделі (Адміністратор, Користувач). Користувач не повинен мати жодної технічної можливості виконати дію, призначену для Адміністратора.
  - Захист від веб-вразливостей: Система має бути захищена від основних векторів атак. Захист від SQL-ін'єкцій забезпечується використанням ORM Entity Framework Core, яка автоматично параметризує всі запити до бази даних. Захист від міжсайтового скриптингу (XSS) забезпечується механізмами фреймворку ASP.NET Core, який за замовчуванням кодує всі дані, що виводяться у HTML.
  - Захищений канал зв'язку: Взаємодія між браузером користувача та сервером має відбуватися виключно через захищений протокол HTTPS з використанням SSL/TLS сертифіката.
- **Масштабованість.** Архітектура системи повинна бути розрахована на поступове зростання обсягів даних та кількості користувачів.

- Система має ефективно працювати з базою даних, що містить до 100 000 записів про об'єкти майна та до 200 000 записів про пов'язані договори та операції.
- Архітектура має дозволяти одночасну роботу до 20-30 користувачів без помітної деградації продуктивності.

- **Юзабіліті (Зручність використання).** Інтерфейс має бути орієнтований на користувачів, які не є IT-спеціалістами.
  - Інтуїтивність: Інтерфейс має бути логічним, послідовним та передбачуваним. Елементи керування (кнопки, фільтри) повинні розташовуватися на очікуваних місцях на всіх сторінках.
  - Простота освоєння: Новий співробітник повинен мати змогу освоїти основні функції своєї ролі (наприклад, пошук та генерація звіту для Користувача) протягом одного робочого дня.
  - Адаптивність: Інтерфейс системи має коректно відображатися на стандартних моніторах з різною роздільною здатністю (Full HD, WQHD), забезпечуючи читабельність та доступність всіх елементів керування без горизонтальної прокрутки.

2.2 Розробка моделей та діаграм системи

На цьому етапі проєктування ми переходимо від текстового опису вимог до їх формалізованого візуального представлення. Це дозволяє краще зрозуміти структуру майбутньої системи, взаємозв'язки між її компонентами та логіку взаємодії з користувачем. Основними інструментами для цього є діаграми мови UML (Unified Modeling Language).

2.2.1 Створення Use Case діаграми для всієї системи

Діаграма варіантів використання (Use Case Diagram) є високорівневим описом функціональності системи. Вона моделює взаємодію між зовнішніми акторами (користувачами) та системою, показуючи, які цілі (варіанти використання) кожен актор може досягти.

На основі визначених ролей (**Адміністратор** та **Користувач**) та сформульованих функціональних вимог, було розроблено загальну Use Case діаграму для автоматизованої системи «Майно громади» (рис. 2.1.1).

Актори:

- **Адміністратор:** Користувач з максимальними повноваженнями, відповідальний за цілісність та актуальність даних у системі.
- **Користувач:** Користувач, що використовує дані для аналізу, моніторингу та звітності.

Діаграма наочно демонструє розподіл обов'язків: Адміністратор виконує всі функції Користувача, а також має ексклюзивні права на управління даними та налаштуваннями системи. Кожен варіант використання, окрім авторизації, неявно передбачає, що актор вже увійшов до системи.

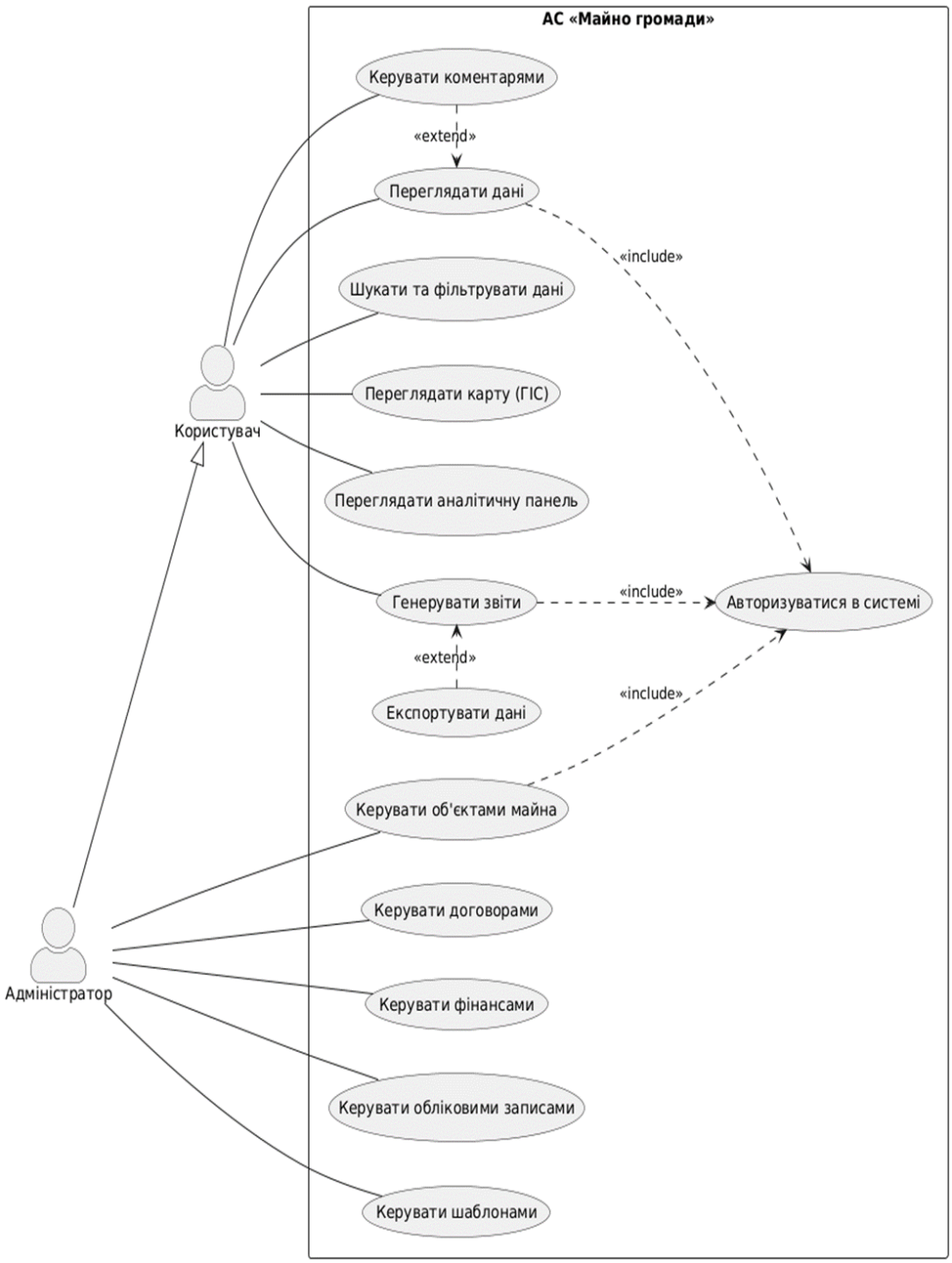

Рисунок 2.1 Use-case діаграма

2.2.2 Написання текстових описів для ключових варіантів використання

Для глибшого розуміння логіки системи та взаємодії користувача з нею, детально опишемо два ключових варіанти використання у розширеному форматі.

Варіант використання 1: Додавання нового договору оренди

- **Назва:** Додавання нового договору оренди.
- **Актор:** Адміністратор.
- **Передумови:**
    1. Адміністратор успішно пройшов автентифікацію в системі.
    2. Об'єкт майна, що передається в оренду, вже існує в реєстрі.
    3. Дані про орендаря (фізичну чи юридичну особу) присутні у відповідному довіднику системи.
- **Основний сценарій успіху:**
    1. Адміністратор у головному меню обирає розділ "Договори" та натискає кнопку "Створити новий".
    2. Система відображає форму для введення даних нового договору.
    3. Адміністратор за допомогою поля пошуку знаходить та обирає потрібний об'єкт майна.
    4. Адміністратор аналогічно знаходить та обирає орендаря.
    5. Адміністратор заповнює реквізити договору: номер, дата підписання, дата початку та закінчення дії, сума орендної плати, періодичність платежів.
    6. Адміністратор натискає кнопку "Зберегти".
    7. Система проводить валідацію введених даних (наприклад, перевіряє коректність дат, заповнення обов'язкових полів).
    8. Система зберігає новий запис про договір у базі даних, встановлюючи зв'язки з відповідним об'єктом майна та орендарем.

9. Система перенаправляє Адміністратора на сторінку перегляду щойно створеного договору та виводить повідомлення про успішне збереження.

- **Альтернативні потоки:**
    1. **7а. Помилка валідації:** Якщо на кроці 7 система виявляє некоректні дані (наприклад, дата закінчення раніше за дату початку), вона не зберігає договір, а повторно відображає форму, зберігаючи введені користувачем дані та підсвічуючи поля з помилками разом із текстовими підказками.
    2. **3а. Об'єкт майна не знайдено:** Якщо Адміністратор не може знайти потрібний об'єкт, він змушений перервати поточний процес, перейти до розділу "Об'єкти майна", додати новий об'єкт, і після цього повернутися до створення договору.

Варіант використання 2: Пошук боржників та генерація повідомлень

- **Назва:** Пошук боржників та генерація повідомлень.
- **Актор:** Користувач.
- **Передумови:**
    1. Користувач успішно пройшов автентифікацію в системі.
    2. В системі наявна інформація про договори та історію платежів, внесена Адміністратором.
- **Основний сценарій успіху:**
    1. Користувач переходить до розділу "Моніторинг" або "Звіти".
    2. Користувач активує фільтр "Показати боржників".
    3. Система виконує запит до бази даних: знаходить усі активні договори, для яких сума фактичних платежів менша за суму планових нарахувань на поточну дату.
    4. Система відображає результат у вигляді таблиці з даними боржників (ПІБ/назва орендаря, об'єкт, номер договору, сума боргу).
    5. Користувач за допомогою прапорців (checkbox) обирає одного чи кількох боржників зі списку.

6. Користувач натискає кнопку "Сформувати повідомлення".
7. Система для кожного обраного запису заповнює даними (ПІБ, сума боргу тощо) попередньо налаштований Адміністратором шаблон документа.
8. Система генерує єдиний PDF-файл, де кожна сторінка є персоналізованим листом-повідомленням для одного боржника.
9. Система надає Користувачу посилання для завантаження згенерованого PDF-файлу на його комп'ютер.

- **Альтернативні потоки:**
    1. **4а. Боржники відсутні:** Якщо в результаті запиту на кроці 4 система не знаходить жодного боржника, вона виводить інформаційне повідомлення "Заборгованість по договорах відсутня".
    2. **6а. Не обрано жодного запису:** Якщо Користувач натискає кнопку "Сформувати повідомлення", не обравши жодного боржника, система виводить підказку "Будь ласка, оберіть хоча б один запис для генерації документів".

2.2.3 Побудова діаграми компонентів для візуалізації архітектури

Діаграма компонентів (Component Diagram) показує високорівневу фізичну структуру системи. Вона візуалізує, з яких основних блоків складається програмний продукт та як ці блоки взаємодіють між собою. Архітектура системи спроєктована за класичною трирівневою моделлю (Three-Tier Architecture), що є стандартом для сучасних веб-додатків на ASP. NET Core.

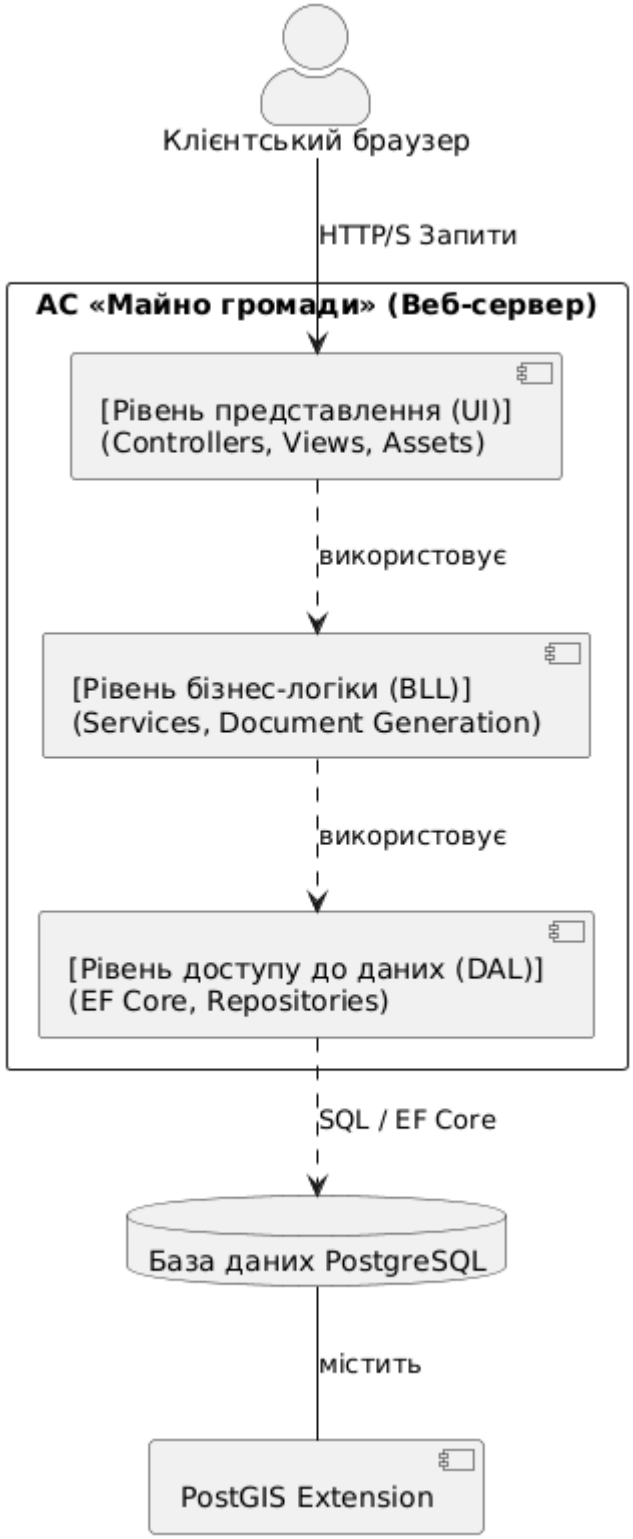

Рисунок 2.2 Діаграма компонентів системи

2.3. Проєктування структури бази даних

Основою будь-якої інформаційної системи є її база даних. Для розробленого додатку було спроєктовано нормалізовану реляційну модель, яка забезпечує цілісність даних, усуває їх надлишковість та надає гнучкість для майбутніх розширень. Структура бази даних розроблялася з урахуванням функціональних вимог та аналізу наданих файлів з реальними даними.

2.3.1. Фінальна ER-діаграма бази даних з усіма таблицями та зв'язками

Логічна модель бази даних складається з п'яти ключових сутностей, кожна з яких відповідає за свою предметну область:

- **LandPlots**: центральна таблиця, що зберігає вичерпну інформацію про кожну унікальну земельну ділянку.
- **Lessees**: довідник орендарів (фізичних та юридичних осіб), що дозволяє уникнути дублювання їхніх даних.
- **Contracts**: таблиця договорів, яка є зв'язуючою ланкою і встановлює відношення "один-до-багатьох" між ділянками та орендарями.
- **Payments**: таблиця для обліку історії фінансових надходжень по кожному конкретному договору.
- **Users**: таблиця для зберігання даних користувачів самої системи (Адміністратор, Користувач).

На рисунку 2.3 наведено фінальну ER-діаграму, що візуалізує ці сутності та встановлені між ними зв'язки.

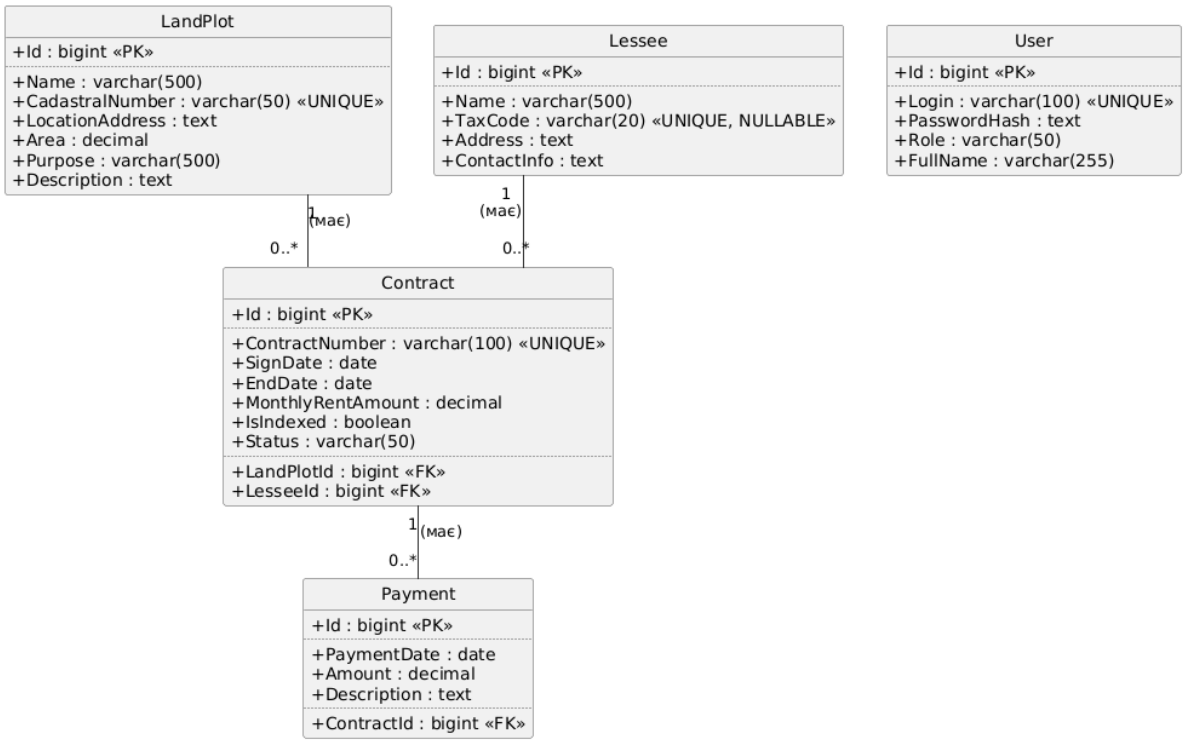

Рисунок 2.3 – ER-діаграма бази даних системи

2.3.2. Створення словника даних

Словник даних є детальним текстовим описом кожної таблиці та її полів, що формалізує спроєктовану схему.

2.3.2 Створення словника даних

Словник даних є детальним текстовим описом кожного елемента ER-діаграми. Він формалізує структуру таблиць, типи даних кожного поля та їх обмеження, що є необхідним для подальшої розробки та супроводу системи. Нижче наведено опис ключових таблиць. (табл 2.1-2.5)

Таблиця 2.1

Структура таблиці LandPlots

Назва поля	Тип даних	Обмеження	Призначення
Id	bigint	PK (Primary Key)	Унікальний ідентифікатор ділянки

Name	varchar(500)		Назва об'єкта / виду призначення
CadastralNumber	varchar(50)	NOT NULL, UNIQUE	Кадастровий номер земельної ділянки
LocationAddress	text	NOT NULL	Місцезнаходження земельної ділянки
Area	decimal(18, 4)	NOT NULL	Площа, га
Purpose	varchar(500)		Цільове призначення / категорія земель
Description	text		Додаткові відомості, примітки

Таблиця 2.2

Структура таблиці Users

Назва поля	Тип даних	Обмеження	Призначення
Id	SERIAL	PK(PrimaryKey)	Унікальнийідентифікатор користувача
Login	VARCHAR(100)	NOT NULL, UNIQUE	Логін користувача для входу в систему
PasswordHash	TEXT	NOT NULL	Хеш пароля користувача з сіллю
Role	VARCHAR(50)	NOT NULL	Роль користувача в системі ("Адміністратор", "Користувач")
FullName	VARCHAR(255)	NOT NULL	Повне ім'я та прізвище користувача

Таблиця 2.3

Структура таблиці Lessees

Назва поля	Тип даних	Обмеження	Призначення
Id	SERIAL	PK	Унікальний ідентифікатор орендаря
Name	VARCHAR(500)	NOT NULL	ПІБ фізичної особи або назва юридичної особи
TaxCode	VARCHAR(20)	UNIQUE	ІПН або ЄДРПОУ орендаря
Address	TEXT		Адреса реєстрації орендаря
ContactInfo	TEXT		Контактна інформація (телефон, email, адреса)

Таблиця 2.4

Структура таблиці Contracts

Назва поля	Тип даних	Обмеження	Призначення
Id	SERIAL	PK	Унікальний ідентифікатор договору
ContractNumber	VARCHAR(100)	NOT NULL, UNIQUE	Номер договору
SignDate	DATE	NOT NULL	Дата укладення договору
EndDate	DATE	NOT NULL	Дата закінчення терміну дії договору
AnnualRentAmount	DECIMAL(18, 2)	NOT NULL	Розмір річної орендної плати, грн
Status	VARCHAR(50)	NOT NULL	Поточний статус ("Активний", "Неактивний")

LeaseableObjectId	BIGINT	FK до LandPlots.Id	Посилання на об'єкт, що орендується
LesseeId	BIGINT	FK до Lessees.Id	Посилання на орендаря

Таблиця 2.5

Структура таблиці Payments

Назва поля	Тип даних	Обмеження	Призначення
Id	SERIAL	PK	Унікальний ідентифікатор платежу
PaymentDate	DATE	NOT NULL	Дата фактичної оплати
Amount	DECIMAL(18, 2)	NOT NULL	Сума платежу
ContractId	BIGINT	FK до Contracts.Id	Посилання на договір, по якому здійснено оплату

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Налаштування середовища та реалізація шару доступу до даних

3.1.1. Опис інструментальних засобів та архітектури проєкту

Для розробки автоматизованої системи було обрано набір сучасних та поширених у галузі інструментів, що забезпечують високу продуктивність, надійність та гнучкість процесу розробки.

В якості основного інтегрованого середовища розробки (IDE) було використано Microsoft Visual Studio 2022. Дане середовище надає потужний функціонал для роботи з кодом на мові C#, включаючи інтелектуальне автодоповнення (IntelliSense), вбудований відладчик (debugger) для покрокового аналізу виконання програми, а також зручні інструменти для управління NuGet-пакетами та системою контролю версій Git.

Основою для додатку стала платформа .NET 8, яка є останньою версією з довготривалою підтримкою (LTS — Long-Term Support). Це гарантує стабільність, регулярні оновлення безпеки та актуальність технології на найближчі роки. Ключовими перевагами .NET 8 є висока продуктивність, кросплатформеність та підтримка сучасних можливостей мови C#, що дозволило побудувати ефективну архітектуру. [6]

Для зберігання даних було обрано об'єктно-реляційну систему управління базами даних (СУБД) PostgreSQL. Це потужна, безкоштовна СУБД з відкритим кодом, відома своєю надійністю та суворим дотриманням стандартів SQL. [9] Для візуального адміністрування бази даних використовувався інструмент pgAdmin4.

3.1.2. Опис реалізованої трирівневої архітектури

Проєкт було спроєктовано з використанням класичної трирівневої архітектури з метою чіткого розділення відповідальності між компонентами

системи. Такий поділ відповідає принципам чистої архітектури, що дозволяє забезпечити незалежність бізнес-логіки від способів представлення та зберігання даних [5]. Рішення (Solution) у Visual Studio складається з трьох окремих проєктів, кожен з яких реалізує свій архітектурний шар (рис. 3.1).

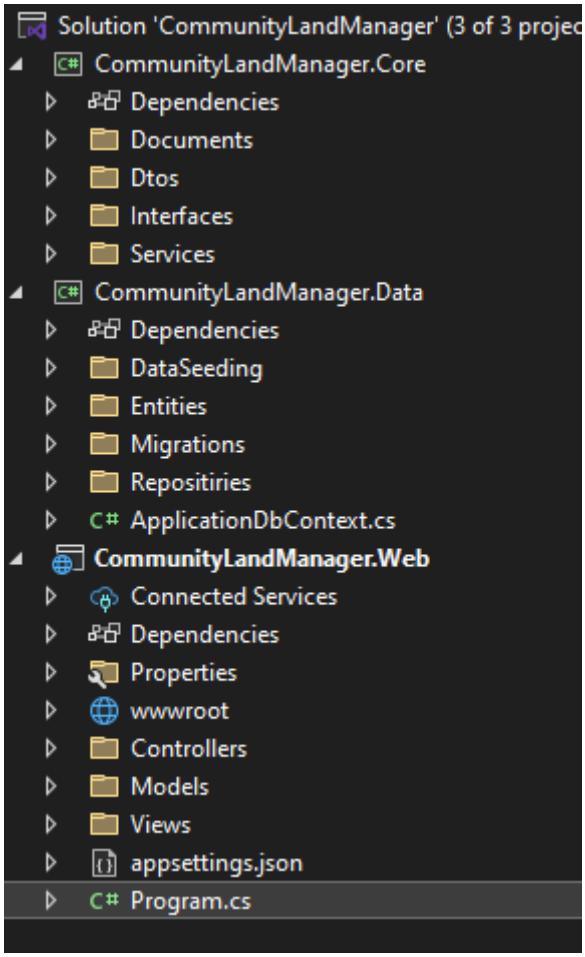

Рисунок 3.1 – Структура проєктів у рішенні

Призначення архітектурних шарів:

- **CommunityLandManager.Data (Рівень доступу до даних – DAL):** Цей проєкт є найнижчим рівнем архітектури і відповідає виключно за взаємодію з базою даних. Він містить C#-класи сутностей (Entities), що точно відображають структуру таблиць, та основний клас ApplicationDbContext, який реалізує логіку роботи з даними за допомогою Entity Framework Core. [8]
- **CommunityLandManager.Core (Рівень бізнес-логіки – BLL):** Це "мозок" додатку. Він містить всю основну логіку: класи-сервіси, що реалізують бізнес-правила та DTO (Data Transfer Objects) – об'єкти для безпечної

передачі даних між шарами. Цей шар не залежить від деталей реалізації інтерфейсу чи бази даних.

- **CommunityLandManager.Web (Рівень представлення – UI):** Це основний веб-проєкт, створений за шаблоном ASP.NET Core MVC. Він відповідає за взаємодію з користувачем. Контролери цього шару приймають HTTP-запити, викликають відповідні методи сервісів з рівня бізнес-логіки, отримують від них дані у вигляді DTO та передають їх у Razor-представлення (Views) для генерації фінальної HTML-сторінки.

3.1.3. Розробка модуля початкового завантаження даних (Seeding)

Для наповнення системи початковими даними та для цілей тестування було розроблено спеціалізований модуль початкового завантаження, або "сідінгу". Основна задача цього модуля — імпорт реальних даних з наданих CSV-файлів, які мають складну, денормалізовану структуру та особливості форматування. Для реалізації цього функціоналу було створено статичний клас DataSeeder.

Процес імпорту зіткнувся з кількома практичними проблемами, для вирішення яких було розроблено відповідні алгоритми.

1. Обробка форматів та кодувань CSV-файлів

Початковий аналіз вихідних файлів показав, що вони збережені з використанням крапки з комою (;) в якості роздільника стовпців, а також у застарілому кодуванні windows-1251, що призводило до некоректного відображення кириличних символів. Для вирішення цієї проблеми було використано бібліотеку CsvHelper зі спеціальною конфігурацією. [12] (рис. 3.2).

```
Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
var windows1251 = Encoding.GetEncoding("windows-1251");
var config = new CsvConfiguration(new CultureInfo("uk-UA")) { Delimiter = ";" };

var records = new List<LandPlotCsvRecord>();
using (var reader = new StreamReader(csvFilePath, windows1251))
using (var csv = new CsvReader(reader, config))
{
    records = csv.GetRecords<LandPlotCsvRecord>().ToList();
}
```

Рисунок 3.2 – Конфігурація CsvHelper для обробки локалізованих файлів

Такий підхід дозволив зробити процес імпорту стійким до регіональних особливостей форматування файлів. Для представлення рядка з CSV-файлу у вигляді об'єкта було створено спеціальні класи-маппери (наприклад, RegistryCsvRecord) з атрибутами [Name("...")], що пов'язують властивості класу з назвами стовпців у файлі.

2. Реалізація патерну "Знайти або створити" (Find-or-Create)

Основною проблемою вихідних даних була їх денормалізація: в одному рядку містилася інформація про земельну ділянку, орендаря та договір. Прямий імпорт таких даних призвів би до масового дублювання записів. Для вирішення цієї проблеми було реалізовано патерн "Знайти або створити" з використанням кешування в пам'яті (Dictionary). (рис. 3.3).

```
foreach (var record in records)
{
    if (string.IsNullOrWhiteSpace(record.LesseeName) || string.IsNullOrWhiteSpace(record.CadastralNumber))
    {
        logger.LogWarning("Skipping row {RowNumber} due to missing LesseeName or CadastralNumber.", record.Number);
        continue;
    }

    string lesseeKey;
    // Якщо є код - він є ключем. Якщо немає - ключем стає ім'я.
    if (!string.IsNullOrWhiteSpace(record.LesseeTaxCode))
    {
        lesseeKey = record.LesseeTaxCode.Trim();
    }
    else
    {
        lesseeKey = record.LesseeName.Trim().ToUpper();
    }
```

Рисунок 3.3 – Фрагмент реалізації патерну "знайти або створити" для орендарів

Цей алгоритм гарантує, що для кожного унікального орендаря (ідентифікованого за кодом або, у випадку його відсутності, за іменем) буде створено лише один запис у таблиці Lessees. Аналогічний підхід було застосовано і для сутності LandPlot з використанням кадастрового номера як унікального ключа. Після знаходження або створення пов'язаних сутностей Lessee та LandPlot створюється об'єкт Contract, який встановлює зв'язок між ними. Таким чином, денормалізовані дані з одного рядка CSV-файлу коректно розподіляються по трьох пов'язаних таблицях у базі даних.

3.2. Реалізація шару бізнес-логіки (BLL)

Шар бізнес-логіки, реалізований у проєкті CommunityLandManager.Core, є центральним елементом архітектури, що інкапсулює основні бізнес-правила та процеси системи. Він виступає посередником між шаром представлення та шаром доступу до даних, забезпечуючи їх слабку зв'язність.

3.2.1. Застосування патерну "Сервіс" та об'єктів для передачі даних (DTO)

Для структурування бізнес-логіки було застосовано патерн "Сервіс". Кожен сервіс є класом, що відповідає за певну функціональну область системи (наприклад, ContractService відповідає за всі операції з договорами). Для забезпечення гнучкості та можливості тестування, кожен сервіс реалізує відповідний інтерфейс (напр., IContractService), який визначає його "контракт".

Ключовим архітектурним рішенням стало використання DTO (Data Transfer Objects). Це прості класи, призначені виключно для передачі даних між шарами. Такий підхід дозволяє повністю відділити структуру бази даних (сутності Entities) від моделей, які використовуються для відображення на сторінках (Views). Це підвищує безпеку системи (захист від over-posting атак) та дозволяє створювати оптимізовані, "пласкі" моделі для складних представлень.

Процес перетворення даних із сутності на DTO, що називається мапінгом, відбувається всередині сервісного шару. (рис. 3.4).

```
7 references
public async Task<ContractDetailsDto?> GetContractDetailsAsync(long id)
{
    var contract = await _context.Contracts
        .Include(c => c.Lessee)
        .Include(c => c.LandPlot)
        .AsNoTracking()
        .FirstOrDefaultAsync(c => c.Id == id);

    if (contract == null) return null;

    return new ContractDetailsDto
    {
        Id = contract.Id,
        ContractNumber = contract.ContractNumber,
        SignDate = contract.SignDate,
        EndDate = contract.EndDate,
        AnnualRentAmount = contract.AnnualRentAmount,
        Status = contract.Status,
        LesseeName = contract.Lessee.Name,
        LesseeTaxCode = contract.Lessee.TaxCode,
        LandPlotAddress = contract.LandPlot.LocationAddress,
        LandPlotCadastralNumber = contract.LandPlot.CadastralNumber,
        LandPlotArea = contract.LandPlot.Area
    };
```

Рисунок 3.4 – Приклад мапінгу сутності на DTO в ContractService

3.2.2. Реалізація ключових бізнес-правил та автоматизації

Сервісний шар є ідеальним місцем для реалізації автоматизованих бізнес-процесів. Як приклад, було розроблено механізм для автоматичного оновлення статусів договорів, термін дії яких закінчився.

Для цього в ContractService було створено метод UpdateAllExpiredContractsStatusAsync. (рис. 3.5).

```
2 references
public async Task UpdateAllExpiredContractsStatusAsync()
{
    var today = DateOnly.FromDateTime(DateTime.Now);

    var expiredContracts = await _context.Contracts
        .Where(c => c.Status == "Активний" && c.EndDate < today)
        .ToListAsync();

    if (expiredContracts.Any())
    {
        foreach (var contract in expiredContracts)
        {
            contract.Status = "Неактивний";
        }
        await _context.SaveChangesAsync();
    }
}
```

Рисунок 3.5 – Метод для автоматичного оновлення статусів договорів

Цей метод викликається кожного разу при відкритті сторінки зі списком договорів у ContractsController. Такий підхід гарантує, що користувач завжди бачить актуальні статуси договорів без необхідності ручного оновлення або запуску фонових задач, що є простим та надійним рішенням для системи даного масштабу.

3.2.3. Реалізація модуля генерації PDF-документів

Однією з ключових вимог до системи є автоматизація документарного супроводу договорів оренди. Для вирішення цієї задачі було розроблено модуль, що дозволяє генерувати офіційні документи у форматі PDF на основі даних, що зберігаються в базі.

Для програмного створення PDF-документів було обрано сучасну .NET-бібліотеку **QuestPDF**. Її перевагами є висока продуктивність, відсутність зовнішніх залежностей та зручний Fluent API для верстки документів за допомогою C# коду. [10]

Вся логіка генерації була інкапсульована у спеціалізованому сервісі DocumentService, що реалізує інтерфейс IDocumentService та знаходиться у проєкті CommunityLandManager.Core. Такий підхід дозволяє відділити логіку створення документів від бізнес-логіки контролерів. Контролер викликає метод сервісу, передаючи йому необхідні дані у вигляді DTO, а сервіс, у свою чергу, викликає відповідний клас-шаблон документа і повертає готовий файл у вигляді масиву байтів. (рис. 3.6).

```
2 references
public byte[] GenerateLeaseCertificate(ContractDetailsDto contractDetails)
{
    var document = new LeaseCertificateDocument(contractDetails);
    return document.GeneratePdf();
}
```

Рисунок 3.6 – Приклад методу в DocumentService

Для кожного типу документа було створено окремий клас у папці CommunityLandManager.Core/Documents. Кожен такий клас реалізує інтерфейс

IDocument з бібліотеки QuestPDF і містить метод Compose, в якому програмно описується структура та вміст документа. Дані для заповнення шаблону передаються в клас через конструктор.

Було реалізовано три основних типи документів:

1. **Довідка про оренду:** Офіційний документ для підтвердження факту наявності діючого договору оренди між громадою та орендарем. (рис. 3.7).
2. **Повідомлення про закінчення терміну дії договору:** Документ для завчасного інформування орендаря про необхідність вжити заходів для поновлення договору.
3. **Вимога про звільнення земельної ділянки:** Офіційний документ, що генерується для неактивних договорів і містить юридично обґрунтовану вимогу повернути ділянку громаді.

```
oid ComposeContent(IContainer container)

    container.PaddingVertical(40).Column(column =>
    {
        column.Spacing(20);

        column.Item().AlignCenter().Text("ДОВІДКА").Bold().FontSize(16);

        column.Item().Text(text =>
        {
            text.DefaultTextStyle(x => x.LineHeight(1.5f));
            text.Span("Видана ");
            text.Span($"{_model.LesseeName}").Bold();
            text.Span($" (РНОКПП/код ЄДРПОУ: {_model.LesseeTaxCode ?? "не вказано"}) про те, що особа є орендарем земельної ділянки
            text.Span($"% {_model.ContractNumber}").Bold();
            text.Span($" від {_model.SignDate:dd.MM.yyyy} р.");
        });

        column.Item().Text("Відомості про земельну ділянку:");
        column.Item().PaddingLeft(25).Text(text =>
        {
            text.Line($"- Кадастровий номер: {_model.LandPlotCadastralNumber}").Bold();
            text.Line($"- Адреса: {_model.LandPlotAddress}");
        });

        column.Item().Text($"Договір є чинним. Термін дії договору до {_model.EndDate:dd.MM.yyyy} р.");
        column.Item().PaddingTop(15).Text("Довідка видана для пред'явлення за місцем вимоги.");
    });
```

Рисунок 3.7 – Фрагмент коду класу-шаблону LeaseCertificateDocument.cs

Для підвищення ефективності роботи користувачів було реалізовано функціонал масової генерації документів для певної групи договорів (наприклад, для всіх неактивних). Щоб не змушувати користувача завантажувати десятки окремих файлів, система автоматично генерує всі PDF-документи в пам'яті сервера, після чого пакує їх в єдиний ZIP-архів і автоматично завантажує їх.

Ця логіка реалізована в окремих методах DocumentService, таких як GenerateVacateDemandsForInactiveContractsAsZipAsync. (рис. 3.8).

```
2 references
public async Task<byte[]> GenerateVacateDemandsForInactiveContractsAsZipAsync()
{
    var inactiveContractsDetails = await _contractService.GetDetailsForAllInactiveContractsAsync();

    if (!inactiveContractsDetails.Any())
    {
        return Array.Empty<byte>();
    }

    using (var memoryStream = new MemoryStream())
    {
        using (var archive = new ZipArchive(memoryStream, ZipArchiveMode.Create, true))
        {
            foreach (var contractDetails in inactiveContractsDetails)
            {
                var document = new VacateDemandDocument(contractDetails);
                var pdfBytes = document.GeneratePdf();

                string safeContractNumber = (contractDetails.ContractNumber ?? $"ID_{contractDetails.Id}").Replace('/', '_');

                var zipEntry = archive.CreateEntry($"Вимога_{safeContractNumber}.pdf", CompressionLevel.Fastest);

                using (var zipStream = zipEntry.Open())
                {
                    await zipStream.WriteAsync(pdfBytes, 0, pdfBytes.Length);
                }
            }
        }
        return memoryStream.ToArray();
    }
}
```

Рисунок 3.8 – Фрагмент методу для генерації ZIP-архіву

Цей підхід дозволяє швидко та зручно обробляти велику кількість документів, що є значною перевагою розробленої системи.

3.3. Реалізація шару представлення (UI)

Шар представлення, реалізований у проєкті CommunityLandManager.Web, відповідає за всю взаємодію з користувачем. Він побудований за архітектурним патерном MVC (Model-View-Controller), що дозволяє чітко розділити логіку обробки запитів, представлення даних та самі дані.

3.3.1. Розробка MVC-контролерів та їх взаємодія з сервісами

Контролери є центральною частиною шару представлення. Вони не містять складної бізнес-логіки, а виступають у ролі координаторів, що приймають HTTP-запити від користувача, викликають відповідні методи сервісів з шару бізнес-логіки та повертають результат у вигляді представлення (View).

Для забезпечення слабкої зв'язності та високої тестовності було застосовано механізм впровадження залежностей (Dependency Injection). Контролер не створює екземпляри сервісів сам, а отримує їх через свій конструктор.

```
public ContractsController(IContractService contractService,
                           ILandPlotService landPlotService,
                           ILesseeService lesseeService,
                           IDocumentService documentService)
{
    _contractService = contractService;
    _landPlotService = landPlotService;
    _lesseeService = lesseeService;
    _documentService = documentService;
}
```

Рисунок 3.8 – Приклад конструктора ContractsController з впровадженням залежностей

Методи-обробники (Actions) у контролері є асинхронними та використовують DTO для передачі даних у представлення, як показано у прикладі методу Details. [7] (рис. 3.9).

```
public async Task<IActionResult> Details(long? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var contractDetails = await _contractService.GetContractDetailsAsync(id.Value);

    if (contractDetails == null)
    {
        return NotFound();
    }
    return View(contractDetails);
}
```

Рисунок 3.9 – Приклад методу-обробника в ContractsController

3.3.2. Створення користувацького інтерфейсу з розширеною функціональністю

Для підвищення зручності користування системою, окрім стандартних CRUD-сторінок, було реалізовано додатковий інтерактивний функціонал.

Комбінована фільтрація та пошук: На сторінці зі списком договорів було створено єдину HTML-форму, що поєднує випадаючий список для фільтрації за статусом договору ("Активні", "Неактивні", "Закінчуються скоро") та текстове поле для повнотекстового пошуку по кількох атрибутах одночасно (номер договору, ім'я орендаря, кадастровий номер). Обидва елементи керування відправляють свої значення як параметри GET-запиту, що дозволяє контролеру передати їх у сервісний шар для виконання складного комбінованого запиту до бази даних.

Стандартний елемент <select> є незручним при роботі з великою кількістю записів (сотні орендарів чи ділянок). Для вирішення цієї проблеми було інтегровано клієнтську JavaScript-бібліотеку **Select2**. На сторінках створення та редагування договорів ця бібліотека застосовується до випадаючих списків вибору орендаря та земельної ділянки, перетворюючи їх на зручні поля з вбудованим пошуком. Це дозволяє користувачу швидко знайти потрібний запис, просто почавши вводити його назву або номер.

3.3.3. Налаштування локалізації для коректної обробки даних

Під час реалізації форм створення та редагування було виявлено проблему з валідацією даних. Механізм прив'язки моделей ASP.NET Core за замовчуванням очікував дати у форматі США (MM/dd/yyyy) та числа з десятковою крапкою. Це призводило до помилок валідації, коли користувач вводив дані у звичному для України форматі (dd.MM.yyyy та десяткова кома).

Проблему було вирішено шляхом налаштування локалізації на рівні всього додатку. У файл Program.cs було додано проміжне програмне забезпечення RequestLocalizationMiddleware, яке встановлює українську культуру (uk-UA) як основну для обробки всіх запитів. (рис. 3.10).

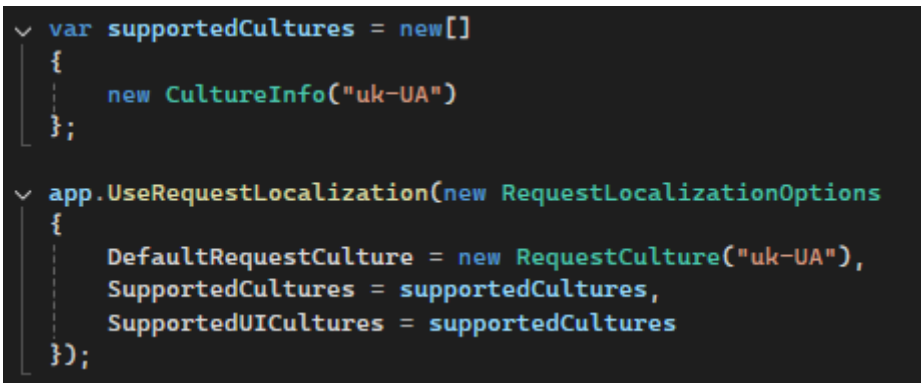

Рисунок 3.10 – Конфігурація локалізації у Program.cs

Це налаштування дозволило системі коректно розпізнавати та валідувати дати й числа, введені користувачем у національному форматі.

3.4. Реалізація розмежування доступу на основі ролей

Для забезпечення безпеки даних та відповідності бізнес-вимогам, у системі було реалізовано механізм розмежування доступу. Архітектура передбачає дві ролі: **Користувач** (з правами тільки на читання та генерацію документів) та **Адміністратор** (з повними правами на зміну даних).

3.4.1. Налаштування Cookie-автентифікації та політик авторизації

В якості механізму автентифікації було обрано стандартний для ASP.NET Core підхід на основі Cookie. При успішному вході система створює для користувача зашифрований автентифікаційний cookie-файл, який браузер надсилає з кожним наступним запитом.

Для розмежування прав було створено спеціальну політику авторизації з назвою "AdminOnly". Ця політика перевіряє, чи містить cookie користувача спеціальну мітку (Claim) IsAdmin=true. Налаштування сервісів автентифікації та політики авторизації виконується у файлі Program.cs. (рис. 3.11).

```
23
24  builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
25      .AddCookie(options =>
26      {
27          options.LoginPath = "/Home/Index";
28          options.AccessDeniedPath = "/Home/Index";
29      });
30
31  builder.Services.AddAuthorization(options =>
32  {
33      options.AddPolicy("AdminOnly", policy => policy.RequireClaim("IsAdmin", "true"));
34  });
35
91
92  app.UseAuthentication();
93  app.UseAuthorization();
```

Рисунок 3.11 – Конфігурація сервісів автентифікації та авторизації у Program.cs

3.4.2. Реалізація механізму входу та виходу для адміністратора

Для простоти системи було реалізовано механізм "розблокування" прав адміністратора за допомогою єдиного пароля, що зберігається у файлі конфігурації appsettings.json. Логіка обробки пароля та створення сесії інкапсульована у HomeController. (рис. 3.12).

```
[HttpPost]
0 references
public async Task<IActionResult> LoginAsAdmin(string password)
{
    string adminPassword = _configuration["AdminPassword"] ?? "default_password";

    if (password == adminPassword)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, "Admin"),
            new Claim("IsAdmin", "true")
        };

        var claimsIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
        var authProperties = new AuthenticationProperties { IsPersistent = true };

        await HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme,
            new ClaimsPrincipal(claimsIdentity),
            authProperties);

        return RedirectToAction("Index", "Home");
    }

    TempData["ErrorMessage"] = "Неправильний пароль адміністратора";
    return RedirectToAction("Index", "Home");
}
```

Рисунок 3.12 – Метод-обробник LoginAsAdmin у HomeController

Метод Logout виконує зворотну дію, викликаючи HttpContext.SignOutAsync для видалення cookie.

3.4.3. Захист методів контролерів та приховування елементів інтерфейсу

Авторизація реалізована на двох рівнях: на сервері (захист методів) та на клієнті (приховування елементів UI).

- **Захист на сервері:** Усі методи-обробники, що відповідають за зміну даних (Create, Edit, Delete), у всіх контролерах (ContractsController, LandPlotsController тощо) захищено атрибутом [Authorize(Policy = "AdminOnly")]. Цей атрибут змушує middleware ASP.NET Core перевіряти політику доступу перед виконанням методу.
- **Приховування в інтерфейсі:** Для покращення досвіду користувача кнопки та посилання на захищені дії приховуються від звичайних користувачів. Це реалізовано у Razor-представленнях за допомогою сервісу IAuthorizationService. (рис. 3.13).

```
<a asp-action="Details" asp-route-id="@item.Id" class="btn btn-sm btn-outline-secondary">Деталі</a>
@if ((await AuthorizationService.AuthorizeAsync(User, "AdminOnly")).Succeeded)
{
    <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-sm btn-outline-primary">Редагувати</a>
    <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-sm btn-outline-danger">Видалити</a>
}
```

Рисунок 3.13 – Приклад умовного відображення елементів у Index.cshtml

Цей підхід гарантує, що користувачі не тільки не можуть виконати дії, на які не мають прав, але й навіть не бачать відповідних елементів керування в інтерфейсі.

3.5. Розробка та візуальне оформлення користувацького інтерфейсу

Окрім розробки основного функціоналу, було приділено увагу створенню зручного та візуально привабливого інтерфейсу користувача.

3.5.1. Дизайн головної сторінки та навігаційних елементів

Стандартна стартова сторінка проєкту була перетворена на інформаційний дашборд. (рис. 3.14). Вона містить стислий опис призначення системи та три великі навігаційні картки з іконками (з бібліотеки Font Awesome), що забезпечують швидкий доступ до основних розділів: "Договори", "Земельні ділянки" та "Орендарі". Глобальні стилі додатку були оновлені для використання

приємного світло-сірого градієнтного фону. Структура головного шаблону _Layout.cshtml була перебудована з використанням Flexbox-верстки для реалізації "липкого" футера, що коректно відображається на сторінках з будь-якою кількістю контенту.

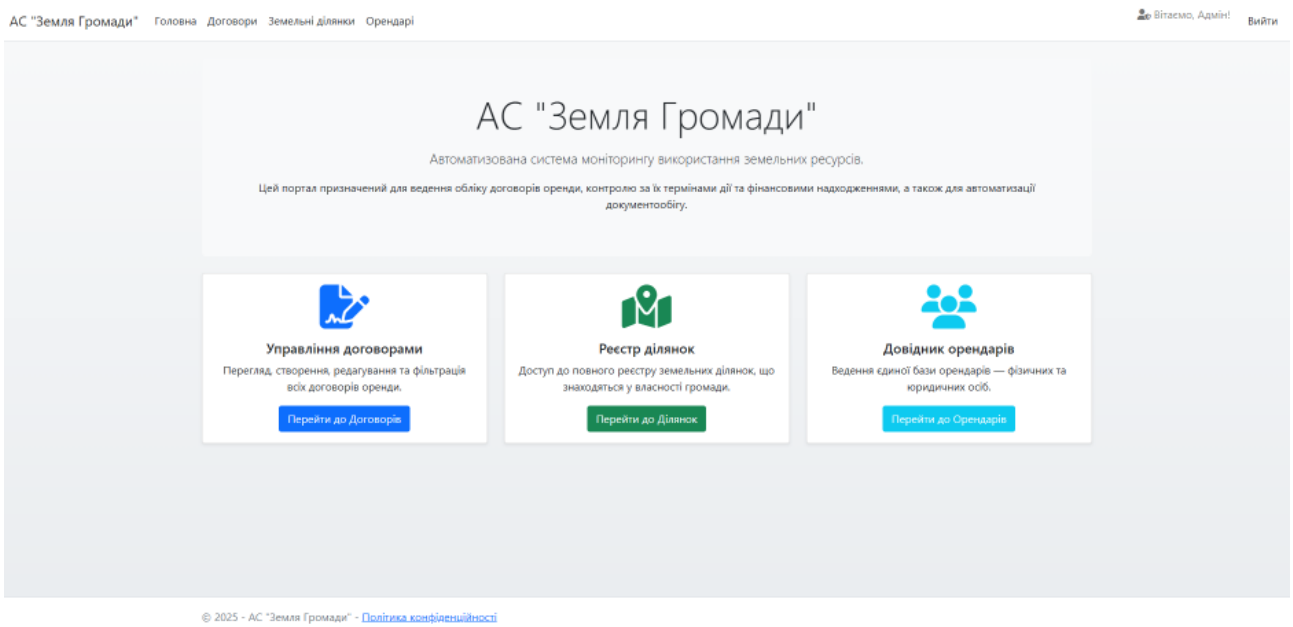

Рисунок 3.14 – Скріншот головної сторінки додатку

3.5.2. Реалізація карток для відображення списків даних

Для підвищення інформативності та покращення візуального сприйняття даних, стандартні таблиці на сторінках-списках (Index.cshtml) були замінені на сучасний макет на основі карток. (рис. 3.10). Кожен запис (договір, ділянка, орендар) відображається в окремому блоці-картці з використанням компонентів Bootstrap 5. Це дозволило логічно згрупувати інформацію, додати іконки для швидкої ідентифікації типу даних та візуальні акценти, наприклад, кольорові індикатори для статусів договорів.

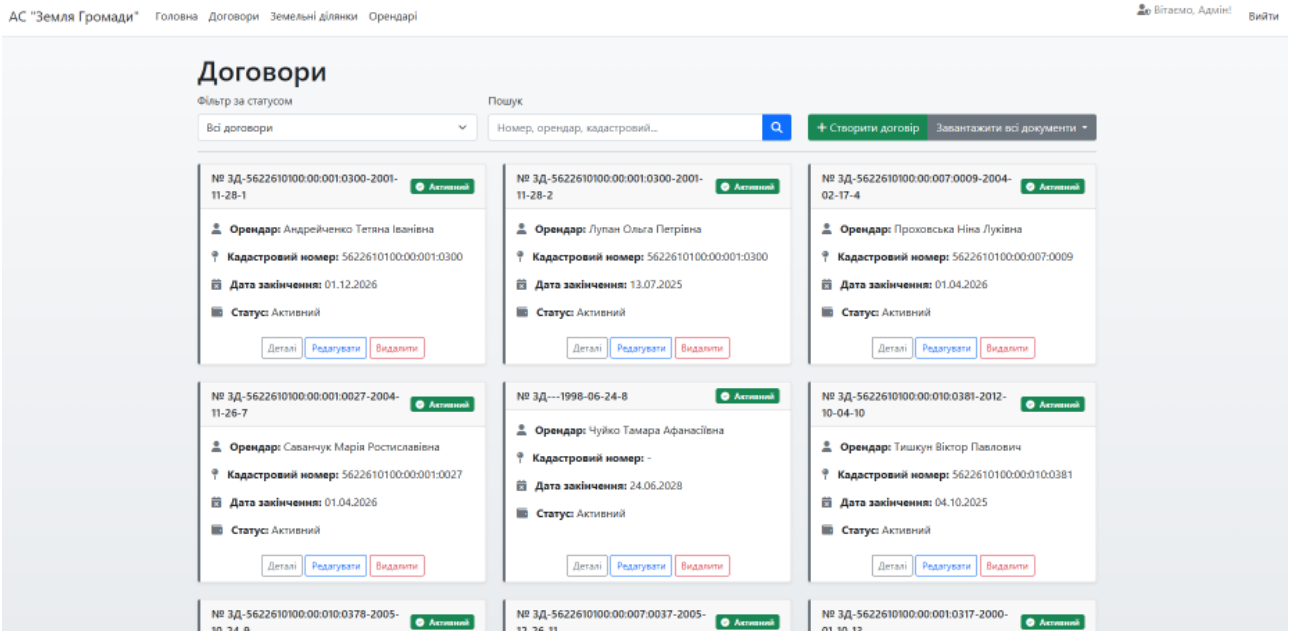

Рисунок 3.15 – Приклад сторінки зі списком договорів

ВИСНОВКИ

У кваліфікаційній роботі було вирішено актуальну науково-практичну задачу розробки автоматизованої системи для моніторингу та управління земельними ресурсами територіальної громади. В ході виконання роботи було отримано наступні результати.

1. Проведено комплексний аналіз предметної області управління земельними ресурсами в органах місцевого самоврядування. Виявлено ключові недоліки існуючих ручних процесів, серед яких: фрагментація даних, високий ризик людських помилок, неефективний моніторинг термінів дії договорів та складність у підготовці документації. Досліджено ринок існуючих програмних рішень, що дозволило обґрунтувати доцільність розробки власної системи, адаптованої до потреб українських громад та розробленої на сучасному, гнучкому стеку технологій (.NET 8, PostgreSQL).
2. Спроєктовано трирівневу архітектуру програмного додатку, що забезпечує чітке розділення відповідальності між шаром доступу до даних (Data), шаром бізнес-логіки (Core) та шаром представлення (Web). Розроблено нормалізовану реляційну модель бази даних, що складається з **5** основних таблиць (LandPlots, Lessees, Contracts, Payments, Users) та забезпечує цілісність даних. Для візуалізації архітектури та сценаріїв взаємодії користувача з системою було побудовано відповідні UML-діаграми (Use Case, діаграма компонентів).
3. Реалізовано програмний продукт у вигляді веб-додатку за технологією ASP.NET Core MVC. Створено модульний механізм початкового завантаження даних (DataSeeder), здатний обробляти складні денормалізовані CSV-файли, вирішуючи проблеми з кодуванням та дублюванням даних. Реалізовано бізнес-логіку з використанням патерну "Сервіс" та об'єктів для передачі даних (DTO). Розроблено модуль генерації **3** типів PDF-документів ("Довідка про оренду", "Повідомлення про закінчення терміну дії", "Вимога про звільнення") та реалізовано

функцію їх масового створення з пакуванням у ZIP-архів. Розроблено користувацький інтерфейс з розширеними можливостями, зокрема, комбінованою фільтрацією та повнотекстовим пошуком, а також інтерактивними випадаючими списками.

4. Розроблена система є готовим до впровадження програмним продуктом, що має високе практичне значення. Її використання дозволить автоматизувати значну частину рутинних операцій працівників відділів земельних ресурсів, скоротити час на пошук інформації та підготовку документів, мінімізувати кількість помилок та посилити контроль за своєчасністю поновлення договорів оренди. Це, в свою чергу, може сприяти підвищенню ефективності управління комунальним майном та збільшенню надходжень до місцевого бюджету.

Таким чином, у ході виконання кваліфікаційної роботи було досягнуто поставленої мети. Створено програмне рішення, яке, на відміну від багатьох існуючих комерційних аналогів, є гнучким, економічно доступним та сфокусованим на ключових потребах українських територіальних громад.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Земельний кодекс України: Закон України від 25.10.2001 № 2768-III. URL: https://zakon.rada.gov.ua/laws/show/2768-14.
2. Про місцеве самоврядування в Україні: Закон України від 21.05.1997 № 280/97-ВР. URL: https://zakon.rada.gov.ua/laws/show/280/97-вр.
3. Про оренду державного та комунального майна: Закон України від 03.10.2019 № 157-IX. URL: https://zakon.rada.gov.ua/laws/show/157-20.
4. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p.
5. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
6. Troelsen A., Japikse P. Pro C# 12 and .NET 8. 12th ed. Apress, 2024. 1435 p.
7. Introduction to ASP.NET Core MVC / Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview.
8. Create and configure a model - EF Core / Microsoft Docs. URL: https://docs.microsoft.com/en-us/ef/core/modeling/.
9. PostgreSQL: Documentation. URL: https://www.postgresql.org/docs/.
10. QuestPDF: Documentation. URL: https://www.questpdf.com/documentation/.
11. The Repository Pattern / Microsoft Docs. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design-repository-pattern.
12. CsvHelper: Documentation. URL: https://joshclose.github.io/CsvHelper/.