ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Клієнт–серверна платформа для автоматизації процесів оренди фінансової компанії з використанням екосистем .NET та React/Next.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо–професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав МЕЗЕЦЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ТЦР–43
Владислав МЕЗЕЦЬКИЙ

Керівник: Ждан ЧЕРНЯВСЬКИЙ
ст. викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо–професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мезецькому Владислав Олеговичу

1. Тема кваліфікаційної роботи: «Клієнт–серверна платформа для автоматизації процесів оренди фінансової
компанії з використанням екосистем .NET та React/Next.js»
керівник кваліфікаційної роботи Ждан ЧЕРНЯВСЬКИЙ
затверджені наказом Державного університету інформаційно–комунікаційних технологій від « 20 » березня 2026 року № 51.
2. Строк подання кваліфікаційної роботи « 01 » травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку мікросервісних веб–застосунків, принципи об'єктно–орієнтованого програмування, документація до платформи ASP.NET Core, засоби розробки клієнтської частини на основі React / Next.js, технічна документація мови програмування C#, документація до СУБД PostgreSQL, засоби автоматизації договірного документообігу
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити):
   1. Огляд та аналіз існуючих програмних рішень для автоматизації договірного документообігу у сфері оренди нерухомості.

2. Аналіз підходів до генерації документів та обґрунтування технічних рішень системи.
3. Проектування бази даних та архітектури системи автоматизації договірного документообігу.
4. Програмна реалізація веб–застосунку з використанням ASP.NET Core, React та патернів проектування Factory Method і Builder.
5. Тестування та розгортання розробленої системи.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз існуючих рішень для автоматизації документообігу.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Архітектура мікросервісної системи.
5. Алгоритм генерації договірних документів
6. Схема бази даних .
7. Екранні форми інтерфейсу користувача.
8. Апробація результатів дослідження

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково–технічної літератури	20.02.2026 – 28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03.2026 – 07.03.2026	
3	Вибір та обґрунтування технологічного стеку	08.03.2026 – 15.03.2026	
4	Проєктування мікросервісної архітектури та бази даних	16.03.2026 – 25.03.2026	
5	Програмна реалізація застосунку	26.03.2026 – 20.04.2026	
6	Тестування застосунку	21.04.2026 – 29.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	30.04.2026 – 11.05.2026	
8	Розробка демонстраційних матеріалів	12.05.2026 – 17.05.2026	
9	Підготовка презентації та попередній захист	18.05.2026 – 27.05.2026	

Здобувач(ка) вищої освіти _____ Владислав МЕЗЕЦЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Ждан ЧЕРНЯВСЬКИЙ
(підпис)

1

РЕФЕРАТ

Мета роботи – автоматизація процесу формування договірної документації шляхом розробки web–додатку для генерації договорів оренди та суборенди на основі шаблонів.

Обєкт дослідження – процес документообігу при укладанні договорів оренди та суборенди нерухомості.

Предмет дослідження – web–додаток для автоматизованого формування, зберігання та управління договорами оренди та суборенди нерухомості.

Методи дослідження – аналіз предметної області, моделювання бізнес–процесів, об'єктно–орієнтоване проєктування, розробка програмного забезпечення з використанням мікросервісної архітектури.

Короткий зміст роботи – у роботі проведено аналіз існуючих підходів до автоматизації договірного документообігу, зокрема систем керування документами та шаблонної генерації документів. Розроблено web–додаток за допомогою Next.js/React та ASP.NET Core Web API на основі мікросервісної архітектури з використанням API–шлюзу Ocelot. Додаток зберігає дані про контрагентів, відділення та умови договорів у базі даних PostgreSQL і використовує їх для автоматичної генерації документів у форматі Word на основі заздалегідь підготовлених шаблонів: договорів оренди та суборенди, додаткових угод, актів приймання–передачі та повернення майна.

Галузь використання – фінансові компанії та мережі пунктів обміну валют, що орендують комерційні приміщення для розміщення розгалуженої мережі відділень і потребують автоматизованого формування договірної документації.

ЗМІСТ

РЕФЕРАТ 6
ЗМІСТ 7
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 9
ВСТУП 10
1. ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ДОГОВІРНОГО ДОКУМЕНТООБІГУ 13
1.1 Сутність та особливості договірного документообігу у сфері оренди нерухомості 13
1.2 Огляд існуючих підходів та інструментів автоматизації документообігу . 13
1.3 Огляд існуючих технологій розробки web–додатків 17
1.3.1 Засоби розробки клієнтської сторони 17
1.3.2 Інструменти серверної частини 18
1.3.3 Архітектурні підходи до побудови веб–застосунків 19
1.3.4 Система управління базами даних 20
1.4. Обґрунтування вибору інструментарію 20
1.4.1 Клієнтська частина 21
1.4.2 Серверна частина 21
1.4.3 Архітектурний підхід 22
1.4.4 Система управління базами даних 23
1.4.5 Генерація документів 24
2. МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ ВЕБ–ЗАСТОСУНКУ АВТОМАТИЗАЦІЇ ДОГОВІРНОГО ДОКУМЕНТООБІГУ 25
2.1 Аналіз підходів до генерації документів 25
2.1.1 Вбудовування тексту договору у програмний код 25
2.1.2 Використання Placeholders 25
2.1.3 Використання Content Control 26
2.2 Обґрунтування патернів проектування 26
2.2.1 Пряма ініціалізація в контролері або сервісі 27
2.2.2 Шаблонний метод (Template Method) 27
2.2.3 Будівельник (Builder) 28
2.2.4 Розгалуження через if/switch 29
2.2.5 Стратегія (Strategy) 29

2.2.6 Фабричний метод 29
2.3 Методика проектування бази даних 31
2.4 Методика побудови мікросервісної архітектури 44
2.4.1 Сервіс авторизація та автентифікації 45
2.4.2 Сервіс контрагентів 45
2.4.3 Сервіс груп 46
2.4.4 Сервіс оренди або суборенди 46
2.4.5 Сервіс охорони 46
2.4.6 Сервіс шлюзу 47
2.5 Методика розробки клієнтської частини 49
2.5.1 Структура та маршрутизація 49
2.5.2 Організація маршрутів та захист сторінок 49
2.5.3 Навігація між розділами 50
2.5.4 Компоненти інтерфейсу 50
2.5.5 Стилізація компонентів 51
2.5.6 Інтерфейс користувача 51
2.5.7 Управління станом даних 55
2.5.8 Безпека на стороні клієнта 56
3. РЕАЛІЗАЦІЯ ВЕБ–ЗАСТОСУНКУ ТА РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ 57
3.1 Архітектура розробленої системи 57
3.2 Реалізація алгоритму генерації документів 63
3.2.1 Отримання запиту 63
3.2.2 Збір даних через Builder 64
3.2.3 Визначення класу документа через Factory 64
3.2.4 Заповнення шаблону та збереження 66
3.3 Розгортання системи 67
3.4 Рекомендації щодо впровадження 69
ВИСНОВКИ 72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 77
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ТОВ – Товариство з обмеженою відповідальністю

ФОП – Фізична особа–підприємець

ORM – Object–Relational Mapping (об'єктно–реляційне відображення)

БД – База даних

API Gateway – Application Programming Interface Gateway (шлюз прикладного програмного інтерфейсу)

LINQ – Language Integrated Query (мова інтегрованих запитів)

API – Application Programming Interface (прикладний програмний інтерфейс)

SQL – Structured Query Language (мова структурованих запитів)

JWT – JSON Web Token (токен автентифікації у форматі JSON)

CRUD – Create, Read, Update, Delete (базові операції з даними)

REST – Representational State Transfer (архітектурний стиль побудови API)

SSR – Server–Side Rendering (серверний рендеринг)

SSG – Static Site Generation (статична генерація сторінок)

TPH – Table Per Hierarchy (стратегія відображення ієрархії в одній таблиці)

TPT – Table Per Type (стратегія відображення ієрархії по одній таблиці на тип)

TPC – Table Per Concrete Type (стратегія відображення ієрархії по одній таблиці на конкретний клас)

CORS – Cross–Origin Resource Sharing (механізм управління міжсайтовими запитами)

XSS – Cross–Site Scripting (міжсайтовий скриптинг)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

SRP – Single Responsibility Principle (принцип єдиної відповідальності)

OCP – Open/Closed Principle (принцип відкритості/закритості)

SOLID – абревіатура п'яти принципів об'єктно–орієнтованого проектування

DDD – Domain–Driven Design (предметно–орієнтоване проектування)

ВСТУП

Актуальність теми. Процеси цифровізації фінансового сектору та стрімке розширення мереж пунктів обміну валют ставлять перед бізнесом нові виклики у сфері юридичного супроводу. Підприємства, що управляють десятками та сотнями відділень, стикаються з необхідністю масового оформлення договорів оренди та суборенди.[1, 2] Кожна локація потребує окремого пакету документів (договори, акти, додаткові угоди), що в умовах ручного заповнення шаблонів призводить до значних часових витрат, помилок через людський фактор та уповільнення масштабування бізнесу.

Використання сучасних стеків розробки, таких як ASP.NET Core та React [3, 4] , дозволяє створювати високонавантажені системи автоматизації документообігу. Проте існуючі універсальні рішення є занадто дорогими для середнього бізнесу, а спеціалізованих інструментів, що враховують специфіку мережевих обмінників (різні типи контрагентів ФОП/ТОВ та ієрархію відділень), на ринку бракує. Створення спеціалізованого веб–додатку для автоматизованої генерації договірної документації є актуальним завданням, що дозволить забезпечити точність даних та високу швидкість операційних процесів.

Об'єкт дослідження – процес документообігу при укладанні та супроводі договорів оренди й суборенди комерційної нерухомості.

Предмет дослідження – методи, архітектурні підходи та технології розробки веб–орієнтованих систем для автоматизованого формування та зберігання договірної документації.

Мета роботи – проєктування та розробка спеціалізованого веб–додатку для автоматизації процесу формування пакету договорів оренди на основі шаблонів та централізованого управління даними контрагентів.

Для досягнення мети у кваліфікаційній роботі виконано наступні завдання:

– аналіз існуючих теоретичних підходів та програмних засобів автоматизації договірного документообігу для виявлення їх переваг та

недоліків.

- сформувати перелік функціональних та нефункціональних вимог до веб–додатку, враховуючи специфіку мережі пунктів обміну валют.
- спроєктувати архітектуру системи на базі мікросервісного підходу та розробити реляційну модель бази даних для зберігання відомостей про контрагентів та об’єкти оренди.
- розробити серверний компонент системи на базі ASP.NET Core Web API із впровадженням API–шлюзу Ocelot [5] для забезпечення маршрутизації та безпеки.
- реалізувати користувацький інтерфейс веб–додатку за допомогою фреймворків Next.js та React, забезпечивши адаптивність та зручність взаємодії.
- створити модуль автоматизованої генерації документів у форматі Word на основі динамічних шаблонів та механізму Content Controls [6].
- виконати тестування розробленого програмного забезпечення та оцінити практичну ефективність його впровадження в бізнес–процеси підприємства.

Методи дослідження. Під час виконання роботи використано методи об'єктно–орієнтованого програмування, системного аналізу предметної області, архітектурного моделювання (UML), реляційного моделювання баз даних та шаблони проєктування програмного забезпечення.

Наукова новизна одержаних результатів. Запропоновано комплексний підхід до автоматизації нішевого документообігу, що поєднує:

- гібридну модель збереження даних контрагентів різних правових форм (ФОП та ТОВ) на основі паттерну Table–per–Hierarchy (TPH) [7] у межах єдиної схеми PostgreSQL;
- архітектурне рішення з реалізацією централізованої JWT–авторизації [8] на рівні API–шлюзу Ocelot, що виключає дублювання логіки

безпеки в мікросервісах.

– модифікований механізм генерації документів через поєднання патернів Builder та Factory Method, що дозволяє юридичному відділу редагувати шаблони через Content Controls без залучення розробників.

Практичне значення одержаних результатів. Розроблений веб–додаток забезпечує ефективне рішення для фінансових організацій та мереж обміну валют. Впровадження системи дозволяє мінімізувати кількість помилок у реквізитах, суттєво скоротити час на підготовку документів та забезпечити єдине сховище актуальних даних про орендовані приміщення та партнерів.

Робота пройшла апробацію на науково–технічних конференціях. За результатами апробації опубліковано тези доповідей:

1. VII Всеукраїнська науково–технічна конференція «Застосування програмного забезпечення в інформаційно–комунікаційних технологіях». Визначення вимог до клієнт–серверної вебплатформи для автоматизації процесів оренди фінансової компанії.
2. VII Всеукраїнська науково–технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Автоматизація документообігу фінансової компанії.

За результатами апробації опубліковано тези доповідей [9,10].

1. ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ДОГОВІРНОГО ДОКУМЕНТООБІГУ

1.1 Сутність та особливості договірного документообігу у сфері оренди нерухомості

Документообіг є невід'ємною складовою діяльності будь–якого підприємства, що здійснює орендні відносини. Під договірним документообігом розуміють сукупність процесів створення, реєстрації, погодження, зберігання та контролю виконання договірної документації.

У сфері оренди та суборенди нерухомості документообіг має низку особливостей. Кожен договір супроводжується пакетом супровідних документів: актом приймання–передачі, актом повернення, додатковими угодами щодо змін умов або продовження строку дії договору. Дані про сторони та об'єкти оренди часто повторюються, тому одні й ті самі реквізити використовуються у багатьох документах. Помилки у договірній документації можуть призвести до юридичних наслідків, тому точність заповнення є особливо важливою.

Традиційний підхід, коли менеджер вручну заповнює шаблон документа у Word, є неефективним. Це призводить до високої ймовірності помилок при копіюванні реквізитів, значних витрат часу, відсутності централізованого обліку договорів і ускладнює швидкий пошук та аналіз документів.

1.2 Огляд існуючих підходів та інструментів автоматизації документообігу

На сьогодні на ринку існує кілька категорій рішень для автоматизації договірного документообігу, кожна з яких має свої переваги та обмеження.

PandaDoc – популярна хмарна платформа для автоматизації документів, що дозволяє створювати шаблони з динамічними полями та автоматично заповнювати їх даними з підключених CRM–систем (HubSpot, Salesforce, Zoho) [11]. Підтримує

електронний підпис, відстеження статусу документів та аналітику. Платформа також підтримує Content Controls, що дозволяє структурувати шаблони з керованими полями введення.

Однак використання сервісу є платним: після 14–денного безкоштовного пробного періоду необхідно придбати преміум–підписку. При цьому створення шаблонів доступне лише в рамках платного тарифу – безкоштовна версія не надає такої можливості. Сервіс також не має можливості адаптації під специфічну предметну область, таку як облік відділень контрагентів обмінника валют.

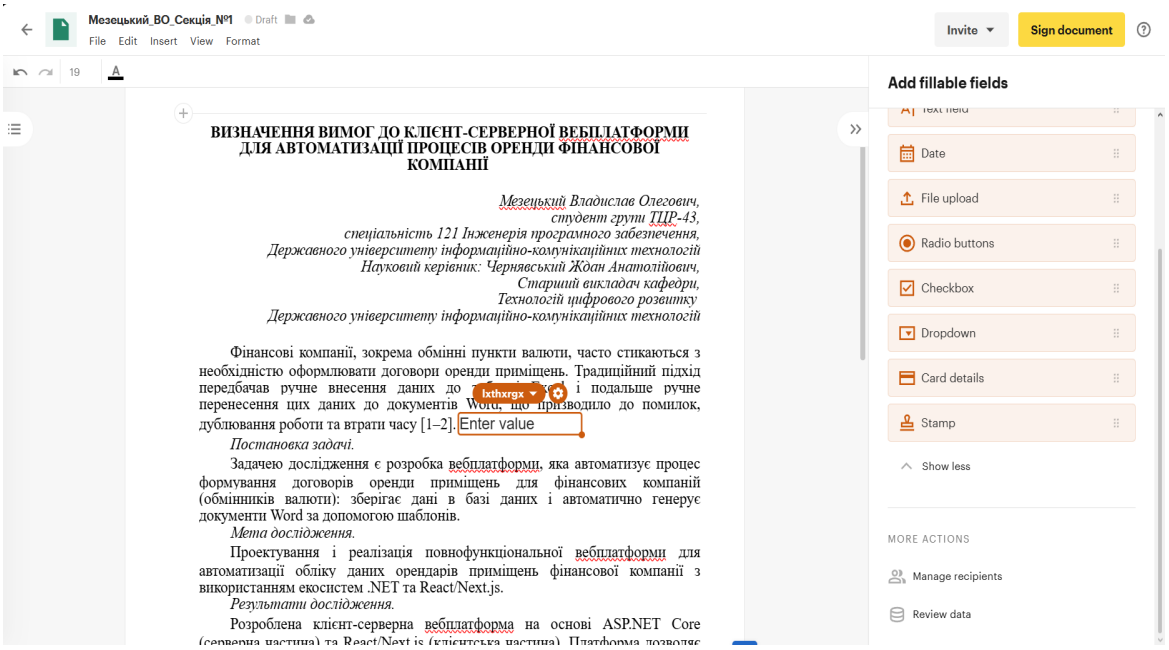

Рис. 1.1. інтерфейс PandaDoc

DocuSign – одна з найпоширеніших платформ для роботи з договорами та електронним підписом [12]. Інтегрується з Microsoft Word через окремий додаток, що дозволяє надсилати документи на підписання безпосередньо з редактора. Основний формат збереження документів – PDF, що забезпечує незмінність підписаних файлів.

Однак платформа орієнтована насамперед на процес підписання, а не на генерацію документів із бази даних. Автоматична підстановка даних контрагентів у шаблон потребує додаткової інтеграції з CRM–системою, що суттєво ускладнює впровадження. Робота з Word–шаблонами та Content Controls не підтримується в повній мірі – документи обробляються і зберігаються у форматі PDF. Крім того, сервіс є відносно дорогим для малого бізнесу та не має можливості адаптації під специфічну предметну область, таку як облік відділень контрагентів обмінника валют.

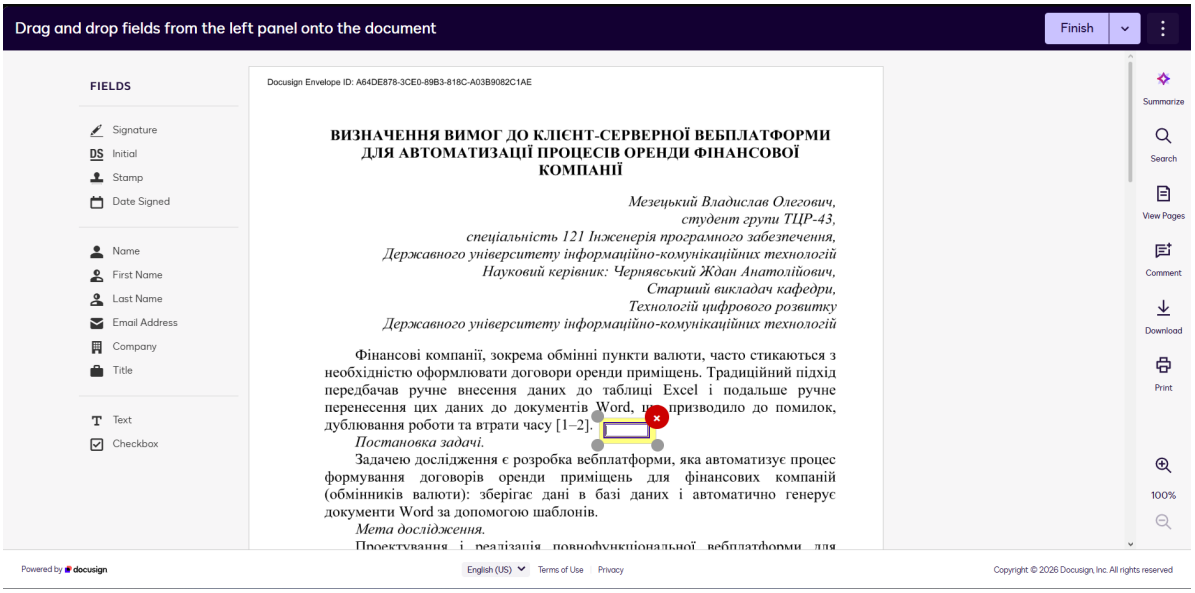

Рис. 1.2. інтерфейс DocuSign

Zoho Contracts – комплексна платформа управління життєвим циклом договорів із бібліотекою шаблонів, відстеженням зобов'язань та інструментами для переговорів з контрагентами [13]. Підтримує понад 35 типів звітів та електронний підпис. Веб–редактор платформи функціонує за принципом, аналогічним до Microsoft Word, та використовує Content Controls для створення і заповнення шаблонів договорів.

Однак інтерфейс платформи є перевантаженим і потребує суттєвого часу на освоєння. Як і інші універсальні платформи, Zoho Contracts не передбачає прив'язки шаблонів до специфічної структури даних підприємства – наприклад, до типів контрагентів (ФОП/ТОВ) або типів оренди відділень. Сервіс є платним, а значне налаштування під конкретні бізнес–процеси вимагає додаткових ресурсів.

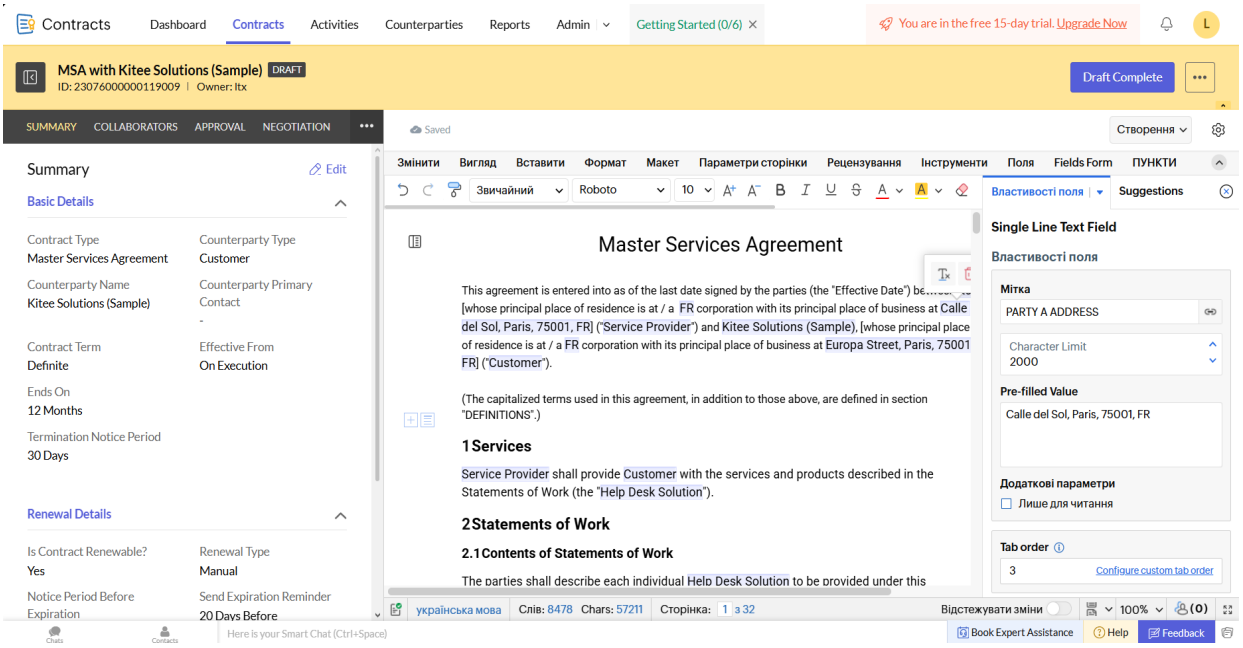

Рис. 1.3. інтерфейс Zoho Contracts

Порівняльний аналіз показує, що жодне з розглянутих рішень не поєднує в собі одночасно: використання стандартних Word–шаблонів з Content Controls без прив'язки до власного редактора, спеціалізовану модель даних для обліку контрагентів різних типів та їх відділень з різними типами оренди, просте веб–орієнтоване розгортання без дорогого впровадження. Саме ця незайнята ніша і визначає практичну цінність розроблюваного застосунку.

1.3 Огляд існуючих технологій розробки web–додатків

1.3.1 Засоби розробки клієнтської сторони

Vue.js – це JavaScript–фреймворк з відкритим вихідним кодом, призначений для побудови користувацьких інтерфейсів.

Фреймворк використовує компонентний підхід до організації інтерфейсу та підтримує двосторонє зв'язування даних між компонентами. Vue.js має власну екосистему інструментів, зокрема Vue Router для маршрутизації та Pinia для управління станом застосунку. Однак порівняно з React екосистема Vue.js є меншою за обсягом, а кількість доступних сторонніх бібліотек та готових рішень є суттєво нижчою. Це ускладнює вирішення нестандартних задач та збільшує обсяг власної розробки у проектах з розширеними вимогами до функціоналу [14].

React – це бібліотека JavaScript з відкритим вихідним кодом, розроблена компанією Meta, що призначена для побудови користувацьких інтерфейсів. Головною концепцією React є компонентний підхід – інтерфейс розбивається на незалежні повторно використовувані компоненти, кожен з яких керує власним станом. Завдяки використанню віртуального DOM React ефективно оновлює лише ті частини сторінки, які змінились, що забезпечує високу продуктивність навіть у складних інтерфейсах. Це робить бібліотеку одним з найпопулярніших інструментів сучасної веб–розробки.

Angular – це фреймворк від Google на основі TypeScript, що надає повний набір вбудованих інструментів для розробки великих застосунків. Ключовими особливостями є двостороннє зв'язування даних, власна система модулів та dependency injection. Angular орієнтований на корпоративні рішення та має вищий поріг входження порівняно з альтернативами, що зумовлює його надлишковість для проектів середнього масштабу [15].

Next.js – це фреймворк на основі React, розроблений компанією Vercel, що розширює можливості бібліотеки за рахунок низки вбудованих механізмів. Ключовою перевагою Next.js є підтримка серверного рендерингу (SSR) та

статичної генерації сторінок (SSG), що суттєво покращує швидкість завантаження та SEO–показники застосунку. Фреймворк також надає файлову систему маршрутизації, що спрощує організацію сторінок, вбудовану оптимізацію зображень та шрифтів, а також зручний механізм API Routes для створення серверних ендпоінтів безпосередньо у межах проекту [16].

1.3.2 Інструменти серверної частини

PHP – це серверна мова програмування з відкритим вихідним кодом, що широко використовується для розробки веб–застосунків. Мова має низький поріг входження та велику кількість готових рішень, однак поступається сучасним фреймворкам за продуктивністю та типовою безпекою. Відсутність строгої типізації за замовчуванням ускладнює підтримку великих кодових баз та підвищує ризик помилок у складних системах [17].

ASP.NET Core Web API – це кросплатформений фреймворк від Microsoft для побудови HTTP–сервісів, що працює на базі середовища виконання .NET. Він забезпечує високу продуктивність завдяки асинхронній моделі обробки запитів, має вбудовану підтримку dependency injection, що спрощує управління залежностями між компонентами, та зручно інтегрується з Entity Framework Core для роботи з базами даних. ASP.NET Core є основою для кожного мікросервісу у розроблюваній системі.

Go – це компільована мова програмування від Google, орієнтована на розробку високонавантажених серверних застосунків. Забезпечує високу продуктивність та ефективну роботу з конкурентністю через вбудований механізм горутин. Однак екосистема Go є менш розвиненою порівняно з .NET, а відсутність узагальненого ORM–фреймворку рівня Entity Framework Core ускладнює роботу з реляційними базами даних у проектах зі складною доменною моделлю [18].

Node.js з TypeScript – це середовище виконання JavaScript на сервері з додаванням статичної типізації через TypeScript. Дозволяє використовувати єдину мову програмування на клієнтській та серверній частинах застосунку.[19, 20]

Проте асинхронна модель на основі однопотокового циклу подій має обмеження при обробці обчислювально інтенсивних задач, а зрілість екосистеми серверних фреймворків поступається платформі .NET у контексті побудови мікросервісних архітектур із розвиненою підтримкою dependency injection та ORM.

1.3.3 Архітектурні підходи до побудови веб–застосунків

Монолітна архітектура – це класичний підхід до побудови веб–застосунків, при якому всі компоненти системи – інтерфейс користувача, бізнес–логіка та робота з базою даних – зібрані в єдиному застосунку і розгортаються як одне ціле. Увесь код знаходиться в одному репозиторії, а всі модулі тісно пов'язані між собою та виконуються в рамках одного процесу [21].

Мікросервісна архітектура – це підхід до проектування програмного забезпечення, при якому система розбивається на набір невеликих незалежних сервісів, кожен з яких відповідає за окрему бізнес–функцію та може розгортатися, масштабуватися і підтримуватися незалежно від інших. На відміну від монолітної архітектури, де всі модулі об'єднані в одному застосунку, мікросервісний підхід дозволяє вносити зміни в окремий сервіс без ризику порушити роботу всієї системи [22].

Модульний моноліт – це проміжний варіант між класичним монолітом і мікросервісами. Застосунок розробляється як єдиний блок з одним репозиторієм і однією базою даних, проте всередині чітко розділений на модулі з визначеними межами та контрактами взаємодії між ними. Такий підхід значно простіший у розробці та підтримці порівняно з мікросервісами, оскільки не потребує складної інфраструктури для міжсервісної комунікації [23].

Сервісно–орієнтована архітектура – більш старий і широкий підхід порівняно з мікросервісами, що набув поширення в корпоративному середовищі. Сервіси взаємодіють між собою через спільну шину даних, яка централізовано керує маршрутизацією та перетворенням даних та бізнес–логікою [24].

1.3.4 Система управління базами даних

PostgreSQL – це об'єктно–реляційна система управління базами даних з відкритим вихідним кодом, що вирізняється надійністю, підтримкою складних SQL–запитів, транзакцій та широкими можливостями масштабування [25]. PostgreSQL підтримує зберігання даних у форматі JSON, що дає додаткову гнучкість при роботі з неструктурованими даними. У розроблюваному застосунку PostgreSQL використовується як єдине сховище даних для всіх мікросервісів – зберігає інформацію про користувачів, контрагентів, їх відділення, групи та договори. Взаємодія з базою даних здійснюється через Entity Framework Core – ORM–фреймворк для .NET, що дозволяє працювати з таблицями бази даних як зі звичайними C#–об'єктами, не пишучи SQL–запити вручну.

MongoDB – це документно–орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (бінарний JSON) [26]. На відміну від реляційних баз даних, MongoDB не вимагає фіксованої схеми таблиць – кожен документ у колекції може мати власну структуру. Це робить її особливо зручною для роботи з даними, що часто змінюють свою форму, або для зберігання складних вкладених об'єктів. MongoDB добре масштабується горизонтально та широко використовується у застосунках із великим обсягом неструктурованих або напівструктурованих даних.

SQLite – це легковагова вбудована реляційна база даних, яка не потребує окремого серверного процесу і зберігає всі дані в єдиному файлі на диску [27]. Завдяки своїй простоті SQLite є стандартним вибором для мобільних застосунків, браузерів та невеликих десктопних програм. Вона повністю підтримує SQL і не вимагає налаштування або адміністрування, проте не призначена для багатокористувацьких систем із високим одночасним навантаженням.

1.4. Обґрунтування вибору інструментарію

Вибір технологічного стеку для розробки web–застосунку автоматизації договірного документообігу здійснювався на основі аналізу вимог до системи,

досвіду розробника та порівняння доступних інструментів.

1.4.1 Клієнтська частина

Для реалізації клієнтської частини обрано бібліотеку React та фреймворк Next.js. Серед альтернативних рішень розглядались Vue.js та Angular. Vue.js, попри низький поріг входження, має меншу екосистему та поступається React за кількістю доступних бібліотек і готових рішень. Angular є потужним фреймворком, однак його складність та об’ємний boilerplate–код роблять його надлишковим для проектів середнього масштабу. React натомість забезпечує достатню гнучкість, має велику спільноту та широку екосистему пакетів. Next.js доповнює React серверним рендерингом та зручною файловою маршрутизацією, що дозволяє швидко організувати навігацію між розділами застосунку – контрагентами, відділеннями, групами та договорами.

1.4.2 Серверна частина

Для реалізації серверної частини обрано ASP.NET Core Web API. Цей фреймворк забезпечує високу продуктивність завдяки асинхронній моделі обробки запитів, має вбудовану підтримку dependency injection та зручно інтегрується з Entity Framework Core для роботи з базою даних. Кросплатформеність .NET дозволяє розгортати застосунок як на Windows, так і на Linux–серверах. ASP.NET Core є природним вибором для побудови мікросервісів з чітко визначеним REST API.

При виборі технології для бекенду розглядались й інші альтернативи. PHP (Laravel) попри широке розповсюдження поступається ASP.NET Core за продуктивністю та типобезпечністю – відсутність строгої типізації за замовчуванням ускладнює підтримку великих проєктів зі складною бізнес–логікою. Node.js дозволяє використовувати JavaScript як на фронтенді, так і на бекенді, однак динамічна типізація знижує надійність при роботі з великою

кількістю взаємопов'язаних сутностей, що є критичним для даного проєкту. Go демонструє відмінні показники швидкості, проте має значно меншу екосистему та не має зрілого ORM–інструменту рівня Entity Framework Core, що суттєво збільшило б обсяг ручного коду для роботи з базою даних.

ASP.NET Core у поєднанні з мовою C# забезпечує строгу типізацію, розвинену екосистему бібліотек, зрілі інструменти для тестування та офіційну підтримку від Microsoft – що в сукупності робить його найбільш обґрунтованим вибором для реалізації надійного та масштабованого серверного застосунку.

1.4.3 Архітектурний підхід

Рішення про застосування мікросервісної архітектури було прийнято розробником самостійно, виходячи з логічної структури предметної області. Система природно розбивається на чотири незалежні модулі – авторизація, контрагенти, групи та договори – кожен з яких має власну зону відповідальності та може розвиватись незалежно. Як API–шлюз обрано бібліотеку Ocelot, яка забезпечує централізовану маршрутизацію запитів від клієнтської частини до відповідних мікросервісів та централізовану перевірку JWT–токенів, що спрощує реалізацію авторизації в межах всієї системи.

При виборі архітектурного підходу розглядались й інші варіанти. Монолітна архітектура була відхилена через надмірну зв'язність між модулями, яку вона створює – будь–яка зміна в одному модулі потенційно впливає на решту системи, а масштабування можливе лише цілим блоком, що є неефективним, коли навантаження зосереджено на конкретному сервісі. Модульний моноліт розглядався як проміжний варіант і має свої переваги – простіша інфраструктура та єдиний процес розгортання. Однак чіткий поділ предметної області на самостійні функціональні блоки з різними зонами відповідальності робив повноцінне розділення на сервіси більш природним і обґрунтованим рішенням з самого початку. Сервісно–орієнтована архітектура передбачає централізовану шину даних та орієнтована на масштаби великих корпоративних систем – така складність є

надлишковою для даного проєкту і лише ускладнила б розробку без відчутної користі.

Таким чином, мікросервісна архітектура забезпечує оптимальний баланс між гнучкістю, незалежністю модулів та складністю реалізації в межах даного проєкту.

1.4.4 Система управління базами даних

Для зберігання даних обрано PostgreSQL. Ця об'єктно–реляційна система управління базами даних з відкритим вихідним кодом вирізняється надійністю, підтримкою складних SQL–запитів, транзакцій та широкими можливостями масштабування. PostgreSQL підтримує зберігання даних у форматі JSON, що дає додаткову гнучкість при роботі з неструктурованими даними. У розроблюваному застосунку PostgreSQLвикористовується як єдине сховище даних для всіх мікросервісів – зберігає інформацію про користувачів, контрагентів, їх відділення, групи та договори.

При виборі бази даних розглядались й інші альтернативи. SQLite була відхилена через свою природу вбудованої однофайлової бази даних – вона не призначена для багатокористувацьких систем із одночасними запитами від кількох мікросервісів і не забезпечує необхідного рівня продуктивності та надійності під навантаженням. MongoDB як документно–орієнтована NoSQL база даних не є оптимальним вибором для даного проєкту, оскільки дані про контрагентів, їх відділення та договори мають чітко визначену реляційну структуру зі зв'язками між сутностями – така структура природно представляється у вигляді реляційних таблиць, а не документів із довільною схемою.

Таким чином, PostgreSQL є найбільш обґрунтованим вибором для проєкту з чітко визначеною реляційною моделлю даних та потребою у надійній роботі в багатосервісному середовищі.

1.4.5 Генерація документів

Для автоматичного формування договорів обрано підхід на основі Word–шаблонів із використанням механізму Content Control . Цей підхід дозволяє юридичному відділу самостійно створювати та редагувати шаблони договорів у звичному середовищі Microsoft Word без залучення розробників. Система отримує шаблон, знаходить у ньому визначені поля та підставляє актуальні дані контрагента та його відділення з бази даних, формуючи готовий документ. Такий підхід забезпечує гнучкість у роботі з різними типами договорів та не прив'язує користувача до інтерфейсу застосунку при редагуванні шаблонів.

Для програмної взаємодії з Content Controls у .NET–середовищі обрано бібліотеку XDoc [26]. Бібліотека надає API для відкриття OOXML–документів, пошуку елементів керування вмістом за тегами та підстановки значень без необхідності встановлення Microsoft Word на сервері. Остання властивість є визначальною для розгортання системи у Docker–контейнерах, де наявність офісного програмного забезпечення не передбачена.

2. МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ ВЕБ–ЗАСТОСУНКУ АВТОМАТИЗАЦІЇ ДОГОВІРНОГО ДОКУМЕНТООБІГУ

2.1 Аналіз підходів до генерації документів

Сьогодні існує кілька підходів до автоматичного заповнення договірних документів, кожен з яких має свої переваги та недоліки.

2.1.1 Вбудовування тексту договору у програмний код

Суть методу полягає в тому, що текст договору або його окремі частини зберігаються безпосередньо у вихідному коді застосунку, а змінні дані підставляються програмно при формуванні документа. Такий підхід є найпростішим з точки зору реалізації, однак має суттєві практичні недоліки. Будь–яка зміна тексту договору – навіть виправлення кількох слів або редагування окремого абзацу – потребує участі розробника: юрист формулює список змін, розробник знаходить відповідний фрагмент у коді, вносить правки та розгортає оновлену версію застосунку на production–середовищі. У великих компаніях, де договори регулярно оновлюються відповідно до змін законодавства або внутрішньої політики, такий процес є неприйнятно повільним та витратним.

2.1.2 Використання Placeholders

Placeholder – це спеціально позначений фрагмент у шаблоні, який програма замінює реальними даними під час генерації. Для цього використовують різні символи–обмежувачі, наприклад {ПІБ}, [[АДРЕСА]], %РНОКПП%. Цей спосіб зручніший, оскільки шаблон зберігається окремим файлом, який юрист або менеджер може редагувати без залучення розробника. Однак основний недолік – placeholders є звичайним текстом. Якщо під час редагування шаблону випадково змінити або видалити символ–обмежувач, наприклад одну дужку у {ПІБ}, програма не розпізнає поле і може залишити його незаповненим або працювати

некоректно. Також цей підхід вимагає знання синтаксису placeholders і уважності при редагуванні.

2.1.3 Використання Content Control

Content Controls – це структуровані елементи керування вмістом, вбудовані у формат Microsoft Word (OOXML).

Вони є окремими XML–елементами з унікальними тегами, за якими програма знаходить потрібні поля. Content Controls підтримують різні типи вмісту: простий і форматований текст, дати, списки. Основна перевага – шаблон залишається звичайним Word–документом, який юрист може редагувати у знайомому середовищі, а поля для підстановки даних структурно відокремлені від основного тексту. Програма підставляє значення з бази даних за відповідними тегами через XML–структуру документа.

2.2 Обґрунтування патернів проектування

Проектування підсистеми генерації договірних документів вимагало вирішення двох принципово різних задач. Перша – збір та підготовка даних для документа, об'єднання різнорідних моделей та формування єдиного плоского об'єкта, придатного для передачі в шаблон. Друга – вибір конкретного класу документа залежно від комбінації параметрів: типу оренди, типу контрагента та виду договірного документа.

Кожна з цих задач є самостійною в архітектурному сенсі та вимагає окремого шаблону проектування. Для першої задачі було обрано патерн Builder, для другої – патерн Factory Method. Нижче наведено аналіз альтернативних підходів та обґрунтування зробленого вибору.

Перед генерацією будь–якого документа система має сформувати об'єкт – плоску структуру, що містить усі поля, необхідні для підстановки у Word–шаблон. Джерелами даних є три незалежні записи в базі даних: сутність договору

(SubleaseModel), відділення (GroupModel) з відповідною інформацією про оренду (SubleaseRentInfo, RentType1Info або RentType2Info) і контрагент (CounterpartyModel у вигляді CounterpartyFop або CounterpartyLLC). Процес збору включає перевірку типу контрагента, вибір потрібного підкласу орендної інформації та заповнення відповідних полів об'єкта.

2.2.1 Пряма ініціалізація в контролері або сервісі

Найпростіший підхід полягає у тому, щоб розмістити логіку збору даних безпосередньо в методі контролера або сервісного класу. Усі звернення до репозиторіїв, перевірки типів і присвоєння полів виконуються послідовно в одному методі. Такий підхід є прийнятним для тривіальних випадків, однак суперечить принципу єдиної відповідальності (SRP) з SOLID: контролер або сервіс виявляється відповідальним одночасно за координацію HTTP–запиту та за складну логіку агрегації даних. Зі зростанням кількості типів документів і джерел даних такий метод швидко перетворюється на важко тестовану функцію з сотень рядків.

2.2.2 Шаблонний метод (Template Method)

Патерн Шаблонний метод передбачає визначення скелета алгоритму в базовому класі та делегування окремих кроків підкласам [29, с. 325–330]. Застосовуючи цей патерн до задачі збору даних, можна було б створити абстрактний клас із захищеними методами (LoadSublease, LoadCounterparty та BuildResult) і конкретними підкласами для кожного типу документа. Недолік підходу в тому, що він нав'язує успадкування там, де достатньо композиції. Підкласи вимушені знати про весь алгоритм у цілому, навіть якщо відрізняються лише одним кроком. Крім того, у системі Actum процес збору даних є спільним для всіх типів документів – різниця полягає лише у тому, як інтерпретуються зібрані дані на етапі генерації. Тому дублювання алгоритму в підкласах не виправдане.

2.2.3 Будівельник (Builder)

Патерн проектування Будівельник відокремлює процес побудови складного об'єкта від його представлення [29, с. 97–106]. У контексті задачі клас (ContractDataBuilder) інкапсулює всю логіку поетапного збору даних: завантаження договору, завантаження відділення та орендної інформації, визначення типу контрагента та завантаження відповідного підкласу, заповнення полів ContractData. Директор у класичному розумінні відсутній – роль ззовні виконує метод BuildAsync(subleaseId), який є єдиною точкою входу. Такий підхід забезпечує ізольованість логіки агрегації, легке тестування (кожен крок можна перевірити незалежно), а також можливість розширення без зміни зовнішнього інтерфейсу.

Таблиця 2.1

Порівняння патернів проектування

Критерій	Пряма ініціалізація	Template Method	Builder
Ізоляція логіки збору	Відсутня	Часткова	Повна
Відповідність SRP	Порушується	Часткова	Дотримується
Тестованість	Низька	Середня	Висока
Розширюваність	Низька	Середня	Висока
Зв'язність з типом документа	Висока	Висока	Відсутня

За сукупністю критеріїв патерн Будівельник є оптимальним вибором для задачі побудови ContractData.

Після того як ContractData сформовано, система має визначити конкретний клас документа, що відповідає поточній комбінації параметрів. У системі Actum таких параметрів три: тип оренди (LeaseType: Sublease, Rent1, Rent2), тип контрагента (ContractorType: Tov, Fop) і вид документа (ContractDocumentType: основний договір з актом і додатком, додаткова угода, акт повернення, угода про пролонгацію). Кількість можливих комбінацій складає 3 × 2 × 4 = 24. Кожна

комбінація відповідає окремому класу документа з власною логікою заповнення Content Controls .

2.2.4 Розгалуження через if/switch

Найпрямолінійніший підхід – послідовний ланцюг умовних операторів у методі–фабриці або сервісі. Вибір класу реалізується через вкладені switch або if–else if з перевіркою кожного з трьох параметрів. Недоліки очевидні: зі зростанням кількості підтримуваних комбінацій метод розростається у важко читаний і схильний до помилок код. Додавання нового типу документа вимагає модифікації існуючого методу, що суперечить принципу відкритості/закритості (OCP) [30].

2.2.5 Стратегія (Strategy)

Патерн Стратегія передбачає визначення сімейства алгоритмів, інкапсуляцію кожного з них та їх взаємозамінність [29, с. 315–324]. Застосований до задачі вибору документа, він означав би наявність інтерфейсу та реєстрацію конкретних стратегій для кожної комбінації параметрів. Вибір стратегії здійснювався б через словник або контейнер залежностей. Підхід є розширюваним, проте надмірним для даної задачі: Strategy зазвичай застосовується тоді, коли алгоритм підмінюється у runtime на підставі зовнішнього стану. У системі Actum набір можливих документів є статично відомим на момент компіляції, а вибір здійснюється одноразово на підставі вхідних параметрів запиту. Strategy вносить зайву складність без реального виграшу.

2.2.6 Фабричний метод

Патерн Фабричний метод делегує створення об'єктів підкласам або спеціалізованим фабричним методам, приховуючи конкретні типи за спільним інтерфейсом [29, с.107–116]. У реалізації ContractDocumentFactory метод Create

приймає три параметри–дискримінатори та повертає екземпляр відповідного класу документа через switch–вираз. Усі 14 класів реалізують спільний інтерфейс, що дозволяє клієнтському коду (контролеру) працювати з будь–яким документом однаково, не знаючи його конкретного типу. Логіка вибору зосереджена в одному місці і легко розширюється без зміни зовнішнього інтерфейсу.

Таблиця 2.2

Порівняння підходів до вибору класу документа

Критерій	if/switch у сервісі	Strategy	Factory Method
Відповідність OCP	Порушується	Дотримується	Дотримується
Ізоляція логіки вибору	Відсутня	Часткова	Повна
Складність реалізації	Низька	Висока	Середня
Придатність для статичного набору	Надмірна	Надмірна	Оптимальна
Читабельність клієнтського коду	Низька	Висока	Висока

Factory Method забезпечує оптимальне співвідношення між ізоляцією логіки вибору, відповідністю принципу відкритості\закритості (OCP) та складністю реалізації.

Обидва патерни утворюють чіткий конвеєр обробки запиту на генерацію документа. На першому етапі ContractDataBuilder.BuildAsync(subleaseId) збирає дані з бази й повертає заповнений ContractData. На другому ContractDocumentFactory.Create(leaseType, contractorType, documentType) миттєво повертає потрібний клас документа. На третьому обраний клас отримує ContractData і виконує підстановку значень у Content Controls Word–шаблону через бібліотеку XDoc. Такий поділ відповідальності відповідає принципу єдиної відповідальності: Builder відповідає за дані, Factory – за вибір типу, конкретний клас документа – за логіку заповнення шаблону.

Комбінація Builder та Factory є обґрунтованим і мінімально достатнім рішенням для підсистеми генерації договірних документів системи Actum. Builder усуває проблему розпорошеної логіки агрегації даних, Factory усуває проблему розгалуженої логіки вибору типу документа. Разом вони формують розширювану, тестовану та легко супроводжувану архітектуру генерації документів.

2.3 Методика проектування бази даних

База даних системи Actum є централізованим сховищем усіх даних: контрагентів, відділень, договорів та записів охорони. Проектування схеми вимагало вирішення кількох нетривіальних задач: представлення ієрархій успадкування в реляційній моделі, збереження поліморфних підтипів орендної інформації та забезпечення цілісності зв'язків між сутностями різних мікросервісів. У цьому підрозділі описано підходи, що розглядалися, обґрунтовано зроблений вибір та наведено структуру таблиць і зв'язків між ними.

Реляційна модель не має нативного механізму для представлення успадкування об'єктно–орієнтованих класів. Entity Framework Core підтримує три стратегії відображення ієрархій, кожна з яких має відмінні характеристики щодо нормалізації, продуктивності та складності запитів.

Table Per Hierarchy (TPH) зберігає всі підкласи в одній таблиці. Рядки розрізняються за допомогою додаткового стовпця–дискримінатора (Discriminator), що містить назву конкретного підкласу. Поля, специфічні для підкласу, присутні в таблиці для всіх рядків, але для рядків інших підкласів містять NULL. Підхід мінімізує кількість таблиць і JOIN–операцій, проте вносить розрідженість даних.

Table Per Type (TPT) виділяє для кожного підкласу окрему таблицю, що містить лише його специфічні поля. Базова таблиця зберігає спільні поля, таблиці підкласів пов'язані з нею зовнішнім ключем. Схема є повністю нормалізованою, проте вибірка конкретного підкласу вимагає JOIN між базовою таблицею і таблицею підкласу.

Table Per Concrete Type (TPC) створює окрему таблицю для кожного

конкретного класу, включаючи в неї всі поля – як успадковані, так і власні. Базовий клас таблиці не має. Підхід усуває JOIN при читанні, але дублює визначення успадкованих стовпців і ускладнює запити, що охоплюють увесь ієрархічний тип.

Таблиця 2.3

Порівняння стратегій відображення ієрархій в Entity Framework Core

Критерій	TPH	TPT	TPC
Кількість таблиць	Мінімальна	Середня	Максимальна
Нормалізація	Низька	Висока	Середня
JOIN при вибірці підкласу	Відсутній	1 JOIN	Відсутній
NULL–поля в таблиці	Присутні	Відсутні	Відсутні
Складність міграцій	Низька	Середня	Висока
Підтримка поліморфних запитів	Висока	Середня	Низька

У системі Actum стратегія TPH (Table Per Hierarchy) застосована для двох ієрархій: контрагентів (CounterpartyModel з підкласами CounterpartyFop та CounterpartyLLC) та орендної інформації (RentInfo з підкласами SubleaseRentInfo, RentType1Info та RentType2Info). В обох випадках кількість підкласів є малою та стабільною, специфічних полів у кожному підкласі небагато, а запити до ієрархії переважно є поліморфними – система завантажує запис без попереднього знання його підтипу. TPH забезпечує отримання такого запису єдиним SELECT без JOIN; накладні витрати від NULL–полів є незначними з огляду на малу кількість специфічних атрибутів.

Схема бази даних складається з п'яти таблиць.

Counterparties – таблиця призначена для зберігання даних контрагентів обмінного пункту. Відповідно до стратегії успадкування Table–Per–Hierarchy (TPH), усі підкласи зберігаються в єдиній таблиці разом із базовим класом. Таблиця

містить як загальні поля базового класу CounterpartyModel, так і специфічні поля підкласів: Director та ShortNameDirector для юридичних осіб (CounterpartyLLC), а також Edryofop для фізичних осіб–підприємців (CounterpartyFop). Розрізнення типів під час матеріалізації об'єктів забезпечується службовим стовпцем Discriminator, який автоматично заповнюється значенням «LLC» або «Fop» засобами EF Core.

Таблиця 2.4

Таблиця Counterparties

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Id	GUID	PK, NOT NULL, за замовчуванням – новий GUID	Унікальний ідентифікатор запису контрагента
Fullaname	string	NOT NULL	Повне офіційне найменування контрагента
ShortName	string	NOT NULL	Скорочене найменування контрагента для відображення в інтерфейсі
GroupName	string	NOT NULL	Назва групи, до якої належить контрагент
Address	string	NOT NULL	Юридична або фактична адреса контрагента
BankAccount	string	NOT NULL	Банківські реквізити контрагента
ResPerson	string	NOT NULL	Відповідальна особа з боку контрагента
Fullaname	string	NOT NULL	Повне офіційне найменування контрагента
ShortName	string	NOT NULL	Скорочене найменування контрагента для відображення в інтерфейсі

Продовження таблиці 2.4

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
BankAccount	string	NOT NULL	Банківські реквізити контрагента
ResPerson	string	NOT NULL	Відповідальна особа з боку контрагента
Fullaname	string	NOT NULL	Повне офіційне найменування контрагента
ShortName	string	NOT NULL	Скорочене найменування контрагента для відображення в інтерфейсі
GroupName	string	NOT NULL	Назва групи, до якої належить контрагент
Address	string	NOT NULL	Юридична або фактична адреса контрагента
BankAccount	string	NOT NULL	Банківські реквізити контрагента
ResPerson	string	NOT NULL	Відповідальна особа з боку контрагента
Phone	string	NOT NULL	Контактний номер телефону
Email	string	NOT NULL	Адреса електронної пошти для комунікації
Rnokpp	string	NOT NULL	Реєстраційний номер облікової картки платника податків
Status	CounterpartyStatus (enum)	NOT NULL	Поточний статус контрагента в системі
Director	string	NULL	Повне ім'я директора – заповнюється лише для підкласу CounterpartyLLC
ShortNameDirector	string	NULL	Скорочене ім'я директора – заповнюється лише для підкласу CounterpartyLLC
Edryofop	string	NULL	Код ЄДРЮОФОП – заповнюється лише для підкласу CounterpartyFop
Discriminator	string	NOT NULL	Службовий стовпець EF Core для визначення типу підкласу («LLC» / «Fop»)

Groups – таблиця призначена для зберігання інформації про відділення обмінного пункту. Тип оренди виступає невід'ємним атрибутом відділення, а не окремого договору, оскільки він визначає юридичний режим використання приміщення незалежно від конкретного орендного зобов'язання. Це забезпечує коректне відображення правового статусу об'єкта на рівні структурної одиниці.

Таблиця 2.5

Таблиця Groups

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Id	GUID	PK, NOT NULL, за замовчуванням – новий GUID	Унікальний ідентифікатор групи
NumberGroup	int	NOT NULL	Номер відділення
CounterpartyId	GUID	FK → CounterpartyModel, NOT NULL	Ідентифікатор контрагента
Address	string	NOT NULL	Адреса відділення
Area	double	NOT NULL, за замовчуванням – 1.0	Площа приміщення
IsAlert	bool	NOT NULL, за замовчуванням – false	Ознака закритого відділення
DateCloseDepartment	DateOnly?	NULL	Дата закриття відділення

Продовження таблиці 2.5

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
RentType	GroupRentType (enum)	NOT NULL, за замовчуванням – None	Тип оренди групи

RentInfos – таблиця реалізує ієрархію орендної інформації на основі стратегії Table–Per–Hierarchy (TPH). Базовий абстрактний клас GroupRentInfo містить загальні ідентифікаційні поля, тоді як кожен підклас розширює його специфічними полями відповідно до типу оренди: SubleaseRentInfo – для суборенди, RentType1Info – для першого типу правового режиму, RentType2Info – для другого. Зв'язок із таблицею Groups є логічно одно–до–одного: обмеження унікальності на рівні одного активного запису забезпечується сервісним шаром застосунку, а не засобами бази даних.

Таблиця 2.6

Таблиця RentInfos

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Id	GUID	PK, NOT NULL, за замовчуванням – новий GUID	Унікальний ідентифікатор запису орендної інформації
GroupId	GUID	FK → Groups, NOT NULL	Зовнішній ключ, що визначає відділення, якому належить орендна інформація

Продовження таблиці 2.6

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Discriminator	string	NOT NULL	Службовий стовпець EF Core для визначення типу підкласу («Sublease» / «Type1» / «Type2»)
RentNumber	string	NULL	Номер договору суборенди – заповнюється лише для SubleaseRentInfo
StartDate	DateOnly	NULL	Дата початку дії договору суборенди – заповнюється лише для SubleaseRentInfo
Person	string	NULL	Найменування або ПІБ орендодавця – заповнюється лише для SubleaseRentInfo
Rnokpp	string	NULL	Реєстраційний номер облікової картки платника податків орендодавця – заповнюється лише для SubleaseRentInfo
Edrpou	string	NULL	Код ЄДРПОУ орендодавця – заповнюється лише для SubleaseRentInfo

Продовження таблиці 2.6

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
CertNumber	string	NULL	Номер правовстановлюючого документа – заповнюється лише для RentType1Info
SeriesCert	string	NULL	Серія правовстановлюючого документа – заповнюється лише для RentType1Info
Issued	string	NULL	Орган, що видав документ – заповнюється лише для RentType1Info
Date	DateOnly	NULL	Дата укладення або реєстрації документа – заповнюється лише для RentType2Info
Num	string	NULL	Номер реєстраційного запису або документа – заповнюється лише для RentType2Info

Subleases – таблиця призначена для зберігання договорів суборенди, укладених у межах відділень обмінного пункту. Кожен запис відповідає окремому договору та містить реквізити документа, часові межі його дії, а також фінансові умови оренди. Зв'язок із таблицею Groups є багато–до–одного: одне відділення може мати кілька договорів суборенди протягом свого існування. Додаткові відомості та файлові вкладення зберігаються у пов'язаних таблицях

Таблиця 2.7

Таблиця Subleases

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Id	GUID	PK, NOT NULL, за замовчуванням – новий GUID	Унікальний ідентифікатор запису договору суборенди
GroupId	GUID	FK → Groups, NOT NULL	Зовнішній ключ, що визначає групу, якій належить договір
ContractNumber	string	NOT NULL	Номер договору суборенди
ContractSigningDate	DateOnly	NOT NULL	Дата підписання договору
AktDate	DateOnly	NOT NULL	Дата акту прийому–передачі об'єкта
ContractEndDate	DateOnly	NOT NULL	Основна дата закінчення дії договору
ContractEndDate2	DateOnly	NULL	Додаткова (уточнена) дата закінчення договору

Продовження таблиці 2.7

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
RentalFee	double	NOT NULL	Основна орендна ставка за договором
RentalFee2	double	NOT NULL	Додаткова орендна ставка
IsContinuation	bool	NOT NULL	Ознака того, що договір є продовженням попереднього
Done	bool	NULL, за замовчуванням – false	Ознака завершеності договору

Guards – таблиця призначена для зберігання записів про охорону відділень обмінного пункту. Кожен запис містить реквізити охоронної компанії, умови договору та контактні дані відповідальної особи. Зв'язок із таблицею Groups є багато–до–одного: одне відділення може мати кілька записів охорони протягом свого існування. Каскадне видалення не застосовується з метою збереження історичних даних про охорону після закриття відділення.

Таблиця 2.8

Таблиця Guards

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Id	GUID	PK, NOT NULL, за замовчуванням – новий GUID	Унікальний ідентифікатор запису договору охорони
GroupId	GUID	FK → Groups, NOT NULL	Зовнішній ключ, що визначає групу, якій належить договір
Address	string	NOT NULL	Адреса об'єкта, що охороняється
OhronnaComp	string	NOT NULL	Найменування охоронної компанії
NumDog	string	NOT NULL	Номер основного договору охорони
NumDog2	string	NOT NULL	Номер додаткового договору охорони
StrokDii	DateOnly	NOT NULL	Основна дата закінчення дії договору
StrokDii2	DateOnly	NULL	Додаткова дата закінчення дії договору
ResPerson	string	NULL	Відповідальна особа

Продовження таблиці 2.8

Назва поля	Тип даних	Обмеження та властивості	Опис призначення
Phone	string	NULL	Контактний номер телефону відповідальної особи
ResPerson	string	NULL	Відповідальна особа за договором охорони
Phone	string	NULL	Контактний номер телефону відповідальної особи
Email	string	NULL	Електронна адреса відповідальної особи

Кожен мікросервіс працює з визначеною підмножиною схеми: Counterparty керує таблицею Counterparties; Groups – таблицями Groups та RentInfos; Sublease – таблицею Subleases з правом читання з Groups і Counterparties для агрегації даних під час генерації документів; Guard – таблицею Guards. Зв'язки між таблицями наведено на рисунку 2.3.1.

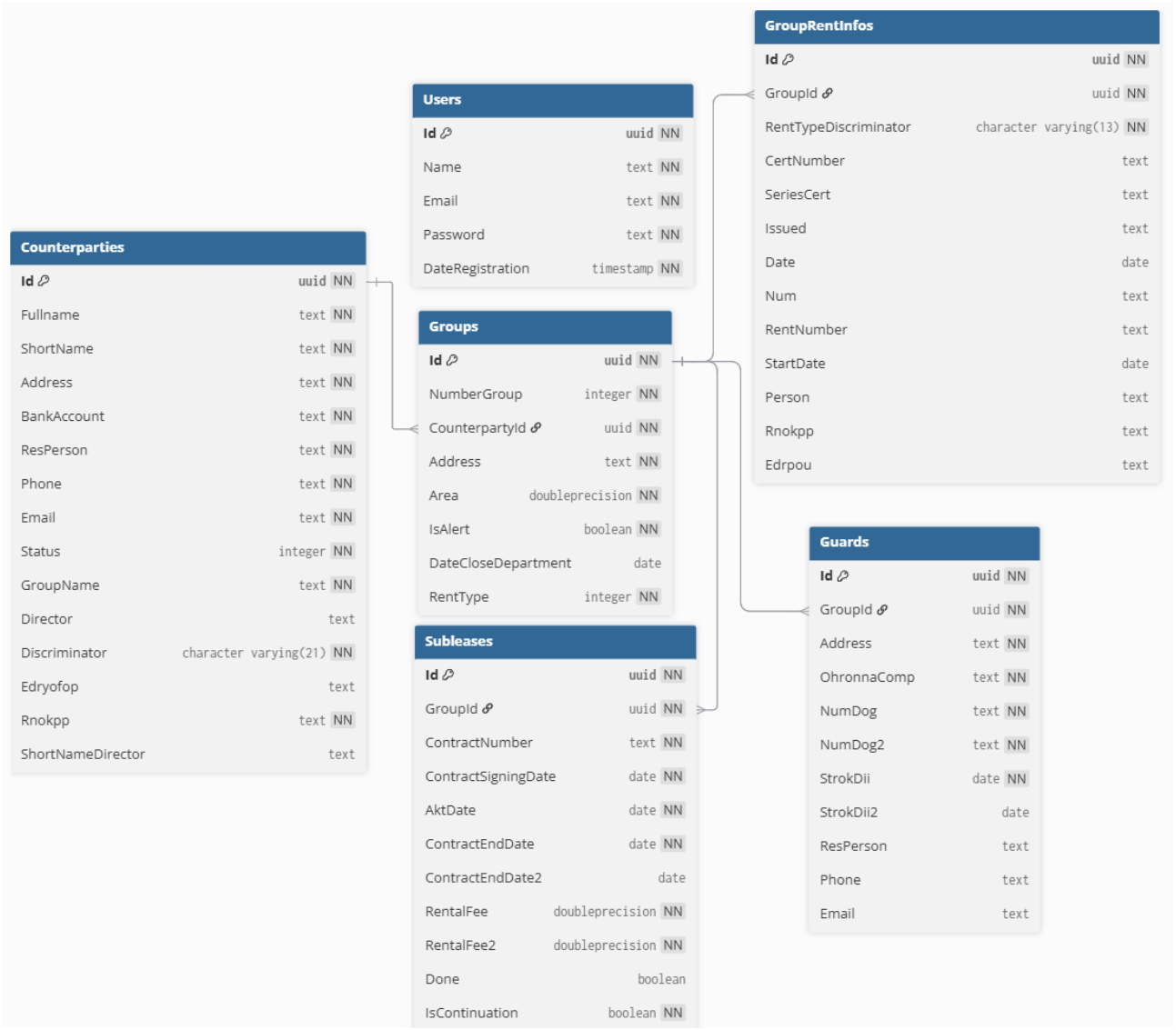

Рис. 2.1. ER–діаграма зв’язків таблиці

У поточній реалізації всі сервіси використовують спільну базу даних PostgreSQL, що є прийнятним рішенням для MVP–стадії проекту.

Спільна база спрощує транзакційну узгодженість і усуває потребу в міжсервісній реплікації даних при генерації документів.

Описана методика проектування бази даних забезпечує відповідність схеми предметній моделі, мінімізує складність запитів для найбільш частих операцій та зберігає можливість еволюції схеми у міру розвитку системи.

2.4 Методика побудови мікросервісної архітектури

Вибір мікросервісної архітектури як основи системи Actum зумовлений характером предметної області: обміннику валют притаманні чітко розмежовані функціональні домени – управління контрагентами, облік відділень, ведення договорів, контроль охорони – кожен з яких має власний життєвий цикл змін і власні вимоги до розгортання. У цьому підрозділі описано принципи декомпозиції системи на сервіси, роль API–шлюзу та механізм автентифікації.

Перед прийняттям архітектурного рішення було розглянуто два основних підходи до побудови серверної частини системи. Моноліт передбачає реалізацію всієї бізнес–логіки в межах одного застосунку, що розгортається єдиним артефактом. Перевагами монолітного підходу є простота розробки на початковому етапі, відсутність мережевих затримок між модулями та єдиний контекст транзакцій. Недоліки проявляються зі зростанням системи: зміна в одному модулі вимагає перерозгортання всього застосунку, масштабування є можливим лише цілісно, а розмір кодової бази ускладнює навігацію та підвищує ризик ненавмисних залежностей між модулями. Мікросервісна архітектура декомпозує систему на автономні сервіси, кожен з яких відповідає за окремий бізнес–домен [2], має власний процес розгортання та взаємодіє з іншими сервісами виключно через мережевий API. Такий підхід забезпечує незалежне масштабування окремих сервісів, ізоляцію збоїв та можливість використання різних технологій для різних компонентів. Ціною є підвищена складність інфраструктури: необхідні механізми маршрутизації, автентифікації на рівні шлюзу та моніторингу розподіленої системи.

Таблиця 2.9

Порівняння підходів архітектури

Критерій	Моноліт	Мікросервіси
Складність початкової розробки	Низька	Середня
Незалежне розгортання сервісів	Відсутнє	Наявне
Ізоляція збоїв	Відсутня	Наявна
Масштабування окремих компонентів	Неможливе	Можливе
Складність інфраструктури	Низька	Висока
Чіткість меж між доменами	Низька	Висока

Декомпозиція системи виконувалась за принципом обмеженого контексту (Bounded Context) з предметно–орієнтованого проектування (Domain–Driven Design) [31]. Кожен сервіс відповідає за один зв'язний домен і є єдиним авторитетним джерелом для операцій модифікації даних у своїй зоні відповідальності. На поточному етапі сервіси використовують спільну базу даних, що є свідомим компромісом між ізоляцією доменів та простотою реалізації.

2.4.1 Сервіс авторизація та автентифікації

Сервіс Auth відповідає виключно за ідентифікацію користувачів: реєстрацію, вхід і видачу JWT–токенів. Виділення автентифікації в окремий сервіс є загальноприйнятою практикою, оскільки вимоги безпеки та алгоритми хешування паролів змінюються незалежно від бізнес–логіки інших доменів. Жоден інший сервіс не зберігає облікові дані користувачів і не перевіряє паролі безпосередньо.

2.4.2 Сервіс контрагентів

Сервіс Counterparty є авторитетним джерелом даних про контрагентів. Він реалізує повний CRUD для обох підтипів – CounterpartyFop та CounterpartyLLC – і

є єдиним сервісом, що має право запису до таблиці Counterparties. Інші сервіси, яким потрібні дані контрагента в режимі читання (наприклад, Sublease при генерації документів), звертаються до таблиці Counterparties безпосередньо через спільний DbContext. Право на запис – створення, оновлення та видалення записів – залишається виключно за сервісом Counterparty.

2.4.3 Сервіс груп

Сервіс Groups керує відділеннями та їх орендною інформацією. Сервіс реалізує CRUD відділень, управління типом оренди та пов'язаною орендною інформацією. Поєднання відділень і орендної інформації в одному сервісі виправдане їх тісним семантичним зв'язком: тип оренди є атрибутом відділення, а орендна інформація завжди існує в контексті конкретного відділення.

2.4.4 Сервіс оренди або суборенди

Сервіс Sublease керує договорами та виконує генерацію документів. Він є найскладнішим за логікою: при генерації документа сервіс координує дані з трьох інших сервісів – Counterparty, Groups та власного сховища. Відповідальність за генерацію документів покладено саме на цей сервіс, оскільки договір є центральним об'єктом, що агрегує всі необхідні дані.

2.4.5 Сервіс охорони

Сервіс Guard веде записи охорони відділень. Його виокремлення в самостійний сервіс пояснюється функціональною незалежністю: управління охороною має власний бізнес–процес, власний цикл оновлення даних і не впливає на логіку.

2.4.6 Сервіс шлюзу

Усі зовнішні запити від клієнтської частини надходять не безпосередньо до сервісів, а через єдину точку входу – API–шлюз, реалізований на основі бібліотеки Ocelot у рамках окремого ASP.NET Core застосунку. Застосування шлюзу вирішує кілька інфраструктурних задач одночасно.

Шлюз забезпечує маршрутизацію запитів: кожен вхідний маршрут описується в конфігураційному файлі ocelot.json через відповідність між зовнішнім шляхом і внутрішньою адресою сервісу. Наприклад, запит GET /api/counterparty/{id} перенаправляється на http://localhost:5282/api/counterparty/{id}. Клієнт не знає і не повинен знати про реальні адреси та порти внутрішніх сервісів.

Також шлюз є єдиним місцем перевірки JWT–токенів. Middleware автентифікації налаштовується на рівні шлюзу і застосовується до всіх захищених маршрутів. Внутрішні сервіси довіряють запитам, що пройшли через шлюз, і не виконують повторну перевірку токена. Такий підхід усуває дублювання логіки автентифікації та гарантує, що жоден запит не може досягти внутрішнього сервісу, обминувши перевірку.

Крім того, шлюз є природним місцем для реалізації наскрізних функцій – логування запитів, обмеження частоти звернень (rate limiting) та CORS–політик. Їх централізація на рівні шлюзу знімає відповідальність за ці аспекти з окремих сервісів.

Структура конфігурації ocelot.json описує кожен маршрут через три основні параметри: UpstreamPathTemplate (шлях, який надходить від клієнта), DownstreamPathTemplate (шлях на внутрішньому сервісі) та DownstreamHostAndPorts (адреса і порт цільового сервісу). Додатково для кожного маршруту вказується перелік дозволених HTTP–методів та, за необхідності, вимога наявності автентифікаційного токена.

Автентифікація в системі Actum реалізована на основі JSON Web Tokens (JWT) [16]. При успішному вході користувача сервіс Auth формує токен, що

містить ідентифікатор користувача та термін дії. Токен підписується секретним ключем з використанням алгоритму HMAC–SHA256, що унеможливлює його підробку без знання ключа.

Клієнтський застосунок зберігає отриманий токен і передає його в заголовку Authorization: Bearer {token} з кожним наступним запитом до шлюзу. Middleware шлюзу перевіряє підпис токена, термін дії та наявність обов'язкових полів. Запити з відсутнім або недійсним токеном отримують відповідь зі статусом 401 і не досягають внутрішніх сервісів.

Перевага JWT перед сесійною автентифікацією в контексті мікросервісної архітектури полягає у відсутності стану на стороні сервера: токен є самодостатнім і містить всю необхідну інформацію для перевірки. Це означає, що шлюз може перевірити токен без звернення до сервісу Auth, що знижує латентність і усуває сервіс Auth з критичного шляху кожного запиту.

Варто зазначити, що у поточній реалізації системи Actum міжсервісна взаємодія при генерації документів здійснюється не через HTTP–запити між сервісами, а через спільну базу даних PostgreSQL. Сервіс Sublease має власний DbContext, що включає доступ до таблиць Counterparties, Groups та RentInfos, і читає необхідні дані безпосередньо з бази під час виконання ContractDataBuilder.BuildAsync(). Такий підхід є прагматичним рішенням для поточної стадії розвитку проекту: він усуває мережеву латентність між сервісами та спрощує транзакційну узгодженість при зборі даних для генерації документа.

Межі між сервісами дотримуються на рівні права запису: кожен сервіс є єдиним авторитетним джерелом для операцій створення, оновлення та видалення у своєму домені. Сервіс Sublease виконує лише читання з таблиць суміжних доменів і не модифікує їх. Це забезпечує консистентність даних і зберігає можливість майбутнього виділення сервісів у повністю ізольовані процеси з окремими базами даних – за умови додавання міжсервісних HTTP–клієнтів або механізму подій.

2.5 Методика розробки клієнтської частини

Клієнтська частина системи Actum реалізована у вигляді веб–застосунку на основі Next.js з використанням App Router і є єдиною точкою взаємодії користувача з системою. Усі запити до бекенду виконуються через API–шлюз, що унеможливлює пряме звернення клієнта до внутрішніх мікросервісів. У цьому підрозділі описано архітектурні принципи побудови клієнтської частини, підходи до організації компонентів, управління станом, стилізації та автентифікації.

2.5.1 Структура та маршрутизація

Next.js є фреймворком поверх React, що розширює його можливостями серверного рендерингу, файлової маршрутизації та вбудованої оптимізації. Починаючи з версії 13, Next.js пропонує два підходи до маршрутизації: класичний Pages Router на основі директорії pages/ та новий App Router на основі директорії app/. Для системи Actum обрано App Router як більш сучасний підхід, що відповідає актуальним рекомендаціям фреймворку. App Router базується на концепції React Server Components [12]: компоненти за замовчуванням є серверними і виконуються на сервері Next.js без передачі JavaScript–коду клієнту. Клієнтські компоненти, що потребують інтерактивності або доступу до браузерних API, позначаються директивою 'use client'. Такий поділ дозволяє мінімізувати обсяг JavaScript, що завантажується браузером, і чітко розмежувати компоненти за їх природою. Кожна директорія в app/ відповідає сегменту URL, а спеціальні файли page.tsx та layout.tsx визначають вміст і обгортку відповідного сегмента.

2.5.2 Організація маршрутів та захист сторінок

Клієнтська частина системи Actum поділена на окремі сторінки відповідно до функціональних доменів: контрагенти, відділення, договори та охорона. Захищені сторінки розташовані в групі маршрутів (app)/ і містять власний layout.tsx зі

спільною навігаційною обгорткою. Публічні сторінки входу та реєстрації винесені в окрему групу (auth)/ з власним макетом. Такий поділ на route groups є механізмом розмежування захищеної та публічної частин застосунку на рівні файлової структури. Захист маршрутів від неавтентифікованих користувачів реалізований через Next.js Middleware: файл middleware.ts перехоплює всі вхідні запити і перевіряє наявність дійсного JWT [16] у cookie. У разі відсутності токена користувач перенаправляється на сторінку входу до рендерингу будь–якого захищеного компонента.

2.5.3 Навігація між розділами

Навігація між розділами реалізована через компонент Link з пакета next/link, що забезпечує клієнтські переходи між сторінками без повного перезавантаження. Спільні елементи інтерфейсу – навігаційна панель (NavMenu.tsx), обгортка сторінки (LayoutWrapper.tsx) – винесені в layout.tsx групи (app)/ і є батьківськими для всіх захищених сторінок. Такий підхід унеможливлює дублювання розмітки та гарантує консистентний вигляд інтерфейсу на всіх маршрутах.

2.5.4 Компоненти інтерфейсу

Інтерфейс застосунку побудований за компонентним принципом React: кожен елемент UI є окремим компонентом з чітко визначеними пропсами та зоною відповідальності. Компоненти організовані в директорії components/ з розбивкою за призначенням: table/ містить компоненти таблиці з підтримкою зміни ширини колонок (useColumnResize.ts) та конфігурацією стовпців (columnConfig.tsx); modalwindow/ містить модальні вікна для редагування та перегляду сутностей кожного домену; navmenu/ – компоненти навігації; card/ – компоненти карток; theme/ – перемикач теми оформлення.

2.5.5 Стилізація компонентів

Стилізація компонентів реалізована через поєднання CSS Modules та глобальних CSS–стилів. CSS Modules забезпечують локальну область видимості класів: файли з розширенням *.module.css генерують унікальні імена класів під час збірки, що виключає конфлікти між стилями різних компонентів. Глобальні стилі визначені в app/globals.css та окремих файлах директорії css/ і містять базові налаштування типографіки, CSS–змінні для кольорів і відступів та стилі спільних елементів. Підтримка світлої та темної теми реалізована через компонент ThemeToggle.tsx із відповідними змінними в theme.css. Відсутність зовнішніх UI–бібліотек забезпечує повний контроль над зовнішнім виглядом і незалежність від сторонніх пакетів.

2.5.6 Інтерфейс користувача

Головна сторінка застосунку є інформаційною панеллю, що надає користувачу узагальнений огляд поточного стану системи. У верхній частині сторінки відображається статус системи — індикатор «Система активна» підтверджує коректну роботу всіх сервісів. Основна область сторінки містить чотири статистичні показники: загальну кількість груп у системі з приміткою про приріст за поточний місяць, кількість активних груп із відображенням їх частки від загальної кількості у відсотках, кількість груп із договорами, що завершуються впродовж найближчих 30 днів, а також загальну кількість завершених договорів за весь час роботи системи. Головна сторінка зображена на рисунку 2.2

Рис. 2.2. Головна сторінка

Усі сторінки застосунку, окрім головної, побудовані за єдиним принципом: кожна містить табличний інтерфейс з полем пошуку для миттєвої фільтрації записів, а також панель керування з кнопками створення, збереження та видалення записів і лічильником змін поточного сеансу редагування.

Сторінка контрагентів відображає перелік усіх контрагентів системи. Подвійне натискання на рядок таблиці переводить запис у режим редагування безпосередньо в таблиці, без переходу на окрему сторінку. Інтерфейс сторінки контрагентів наведено на рисунку 2.3, режим редагування — на рисунку 2.4.

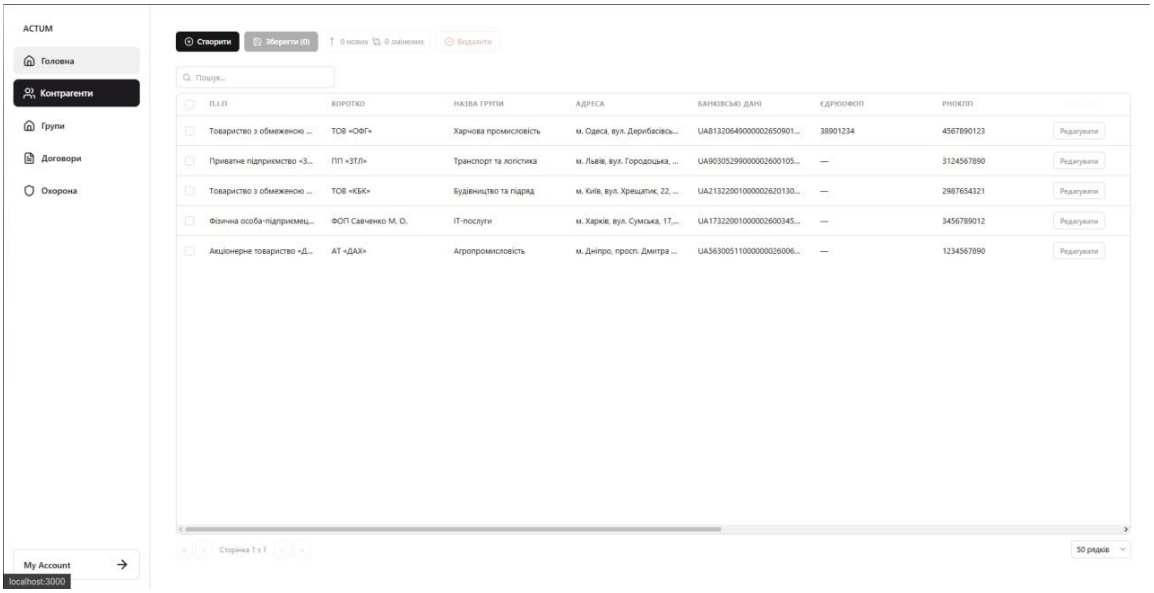

Рис. 2.3. сторінка контрагентів

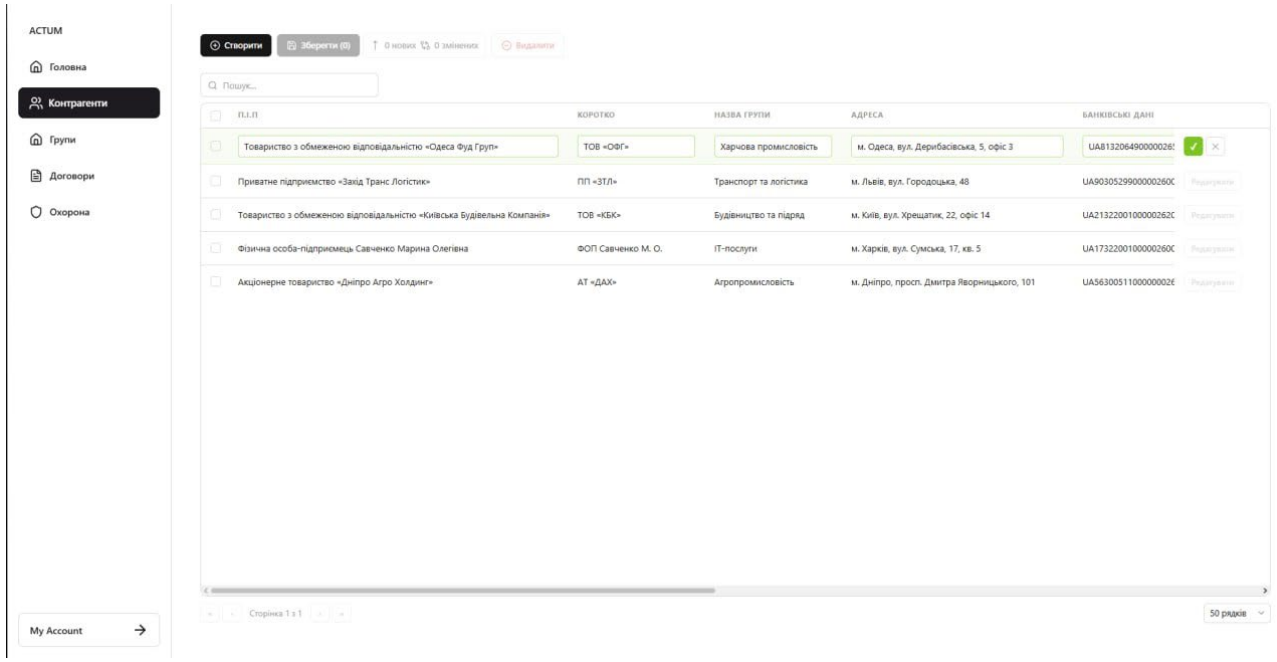

Рис. 2.4. Режим редагування запису

Сторінка груп призначена для управління записами відділень контрагентів. Додатково для кожного запису передбачено окреме модальне вікно, що дозволяє вказати тип оренди відділення. Вікно містить випадаючий список з трьома варіантами: суборенда, оренда за свідоцтвом та оренда за договором. Залежно від обраного типу форма модального вікна змінює свій склад полів — приклади форм для різних типів оренди наведено на рисунках 2.5. Інтерфейс модального вікна вибору типу оренди наведено на рисунку 2.6.

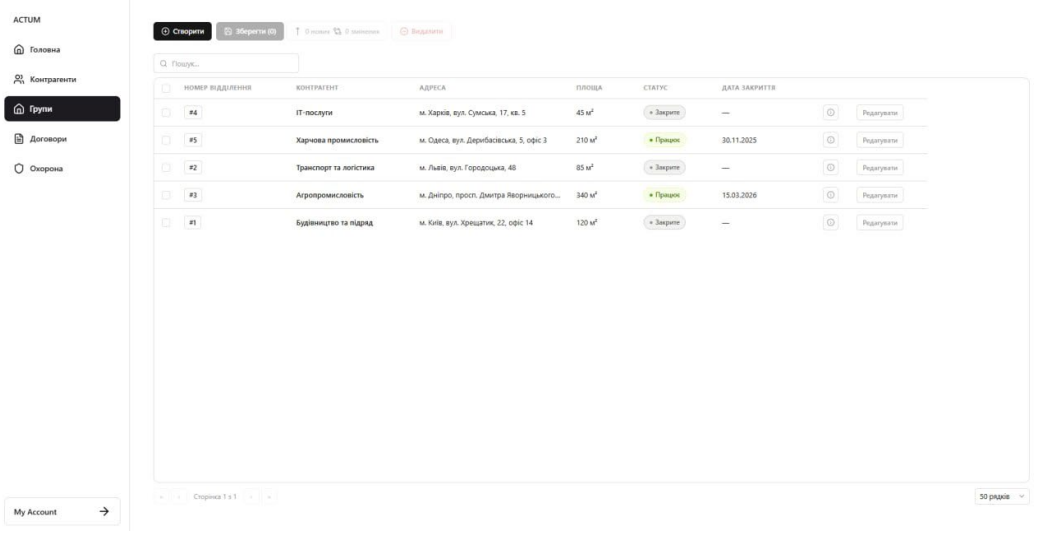

Рис. 2.5. Сторінка груп

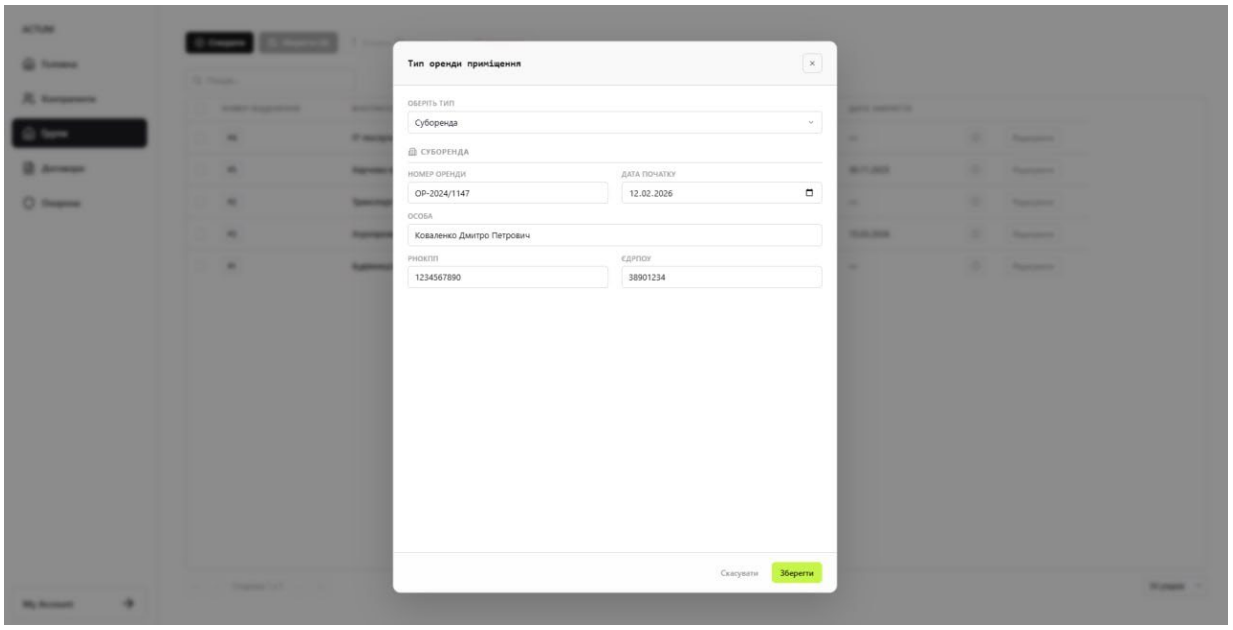

Рис. 2.6. Модальне вікно типу оренди

Сторінка договорів забезпечує управління договірними записами. Генерація договірного документа виконується через модальне вікно, що відкривається для обраного запису. У модальному вікні відображається узагальнена інформація про контрагента, відділення та умови договору, а також надається можливість вибору способу отримання згенерованого документа: завантаження на стороні клієнта або збереження у файловій системі сервера. Інтерфейс сторінки договорів та модальне вікно генерації наведено на рисунках 2.7 та 2.8 відповідно

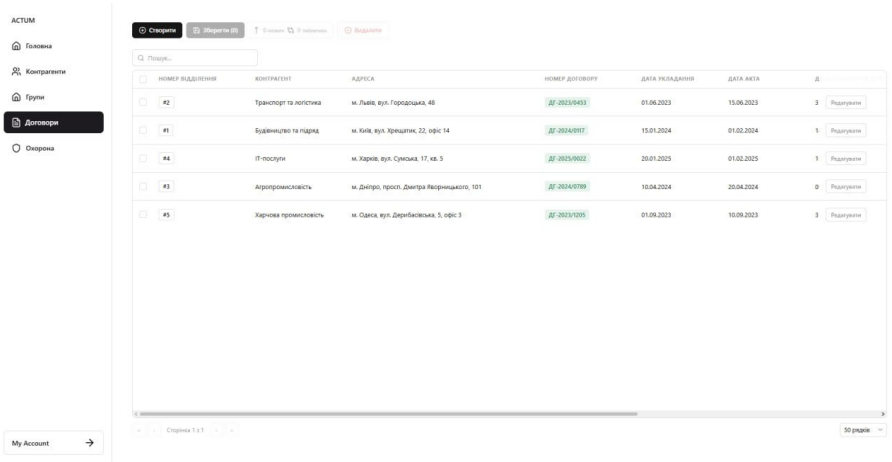

Рис. 2.7. Сторінка договорів

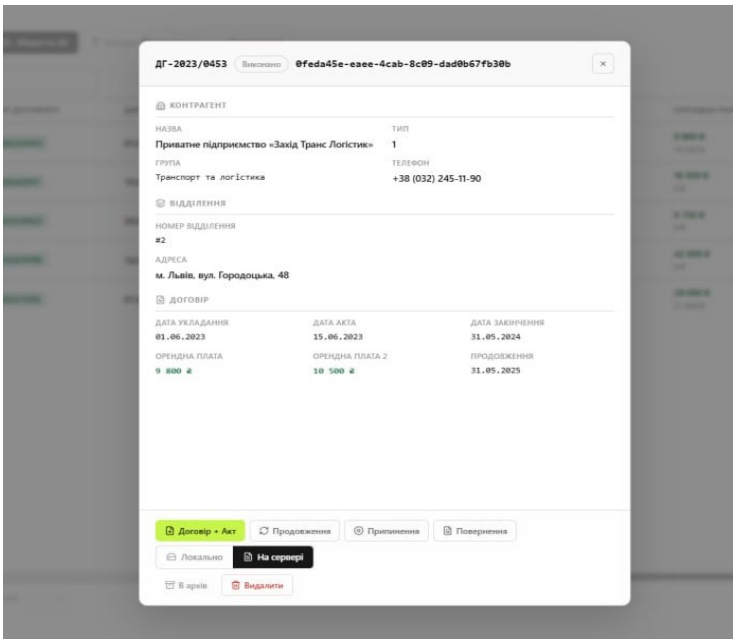

Рис. 2.8. Модальне вікно договору

Сторінка охорони призначена для обліку охоронних компаній, що обслуговують відділення контрагентів. Записи прив'язуються до конкретних відділень, що забезпечує централізований контроль стану охорони всієї мережі в межах єдиної системи. Інтерфейс сторінки охорони наведено на рисунку 2.9.

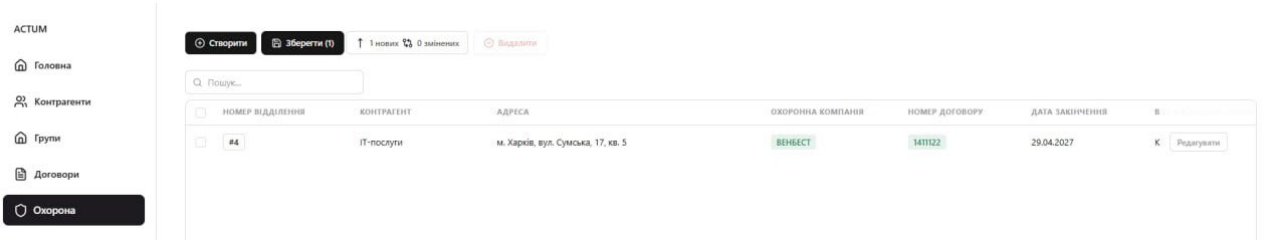

Рис. 2.9. Сторінка записів охорони

2.5.7 Управління станом даних

Управління станом у клієнтській частині реалізоване за рівневим принципом залежно від природи та зони видимості стану. Серверний стан – дані контрагентів, відділень і договорів – управляється через Context API у поєднанні з власним сервісним шаром. Для кожного домену реалізовано окремий провайдер: CounterpartyProvider.tsx (провайдер даних контрагентів), GroupProvider.tsx

(провайдер даних груп та відділень) та UserProvider.tsx(провайдер даних користувача), що розташовані в директорії Context/. Провайдери інкапсулюють логіку завантаження даних через відповідні функції сервісного шару – Api/Counterparty.ts (функції запитів до сервісу контрагентів), Api/Group.ts (функції запитів до сервісу груп), Api/sublease.ts (функції запитів щодо суборенди), Api/guard.ts (функції запитів щодо охорони) – і надають стан дочірнім компонентам через відповідні контексти. Локальний стан компонентів – значення форм, стан відкриття модальних вікон, тимчасові UI–стани – управляється через хук useState безпосередньо в компоненті. HTTP–запити до API–шлюзу виконуються через Fetch API, інкапсульований у функції сервісного шару, що містять усю логіку формування URL, заголовків та обробки відповідей. Компоненти звертаються до цих функцій через провайдери контексту, не знаючи деталей HTTP–взаємодії.

2.5.8 Безпека на стороні клієнта

JWT–токен [16], отриманий від сервісу Auth після успішного входу, зберігається у httpOnly cookie. Вибір цього механізму є обґрунтованим рішенням з точки зору безпеки: httpOnly cookie недоступні для JavaScript–коду на сторінці, що унеможливлює їх викрадення через XSS–атаку. На відміну від localStorage, де токен може бути прочитаний будь–яким скриптом на сторінці, httpOnly cookie передається браузером автоматично з кожним HTTP–запитом. Встановлення cookie виконується на стороні сервера у відповідь на успішний запит автентифікації через заголовок Set–Cookie з атрибутами HttpOnly, Secure та SameSite=Strict. Вихід з системи реалізований через видалення cookie на сервері, після чого Middleware перенаправляє всі наступні запити на сторінку входу [32].

Описана методика розробки клієнтської частини забезпечує чітке розмежування відповідальності між компонентами, управління серверним станом через доменні провайдери Context API, безпечне зберігання токена автентифікації в httpOnly cookie та природну відповідність файлової структури навігаційній

ієрархії застосунку.

3. РЕАЛІЗАЦІЯ ВЕБ–ЗАСТОСУНКУ ТА РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ

3.1 Архітектура розробленої системи

Розроблена система має трирівневу архітектуру, що складається з рівня представлення (клієнтська частина на Next.js), рівня бізнес–логіки (мікросервіси на ASP.NET Core) та рівня даних (PostgreSQL). Загальну схему взаємодії компонентів системи наведено на рисунку 3.1.

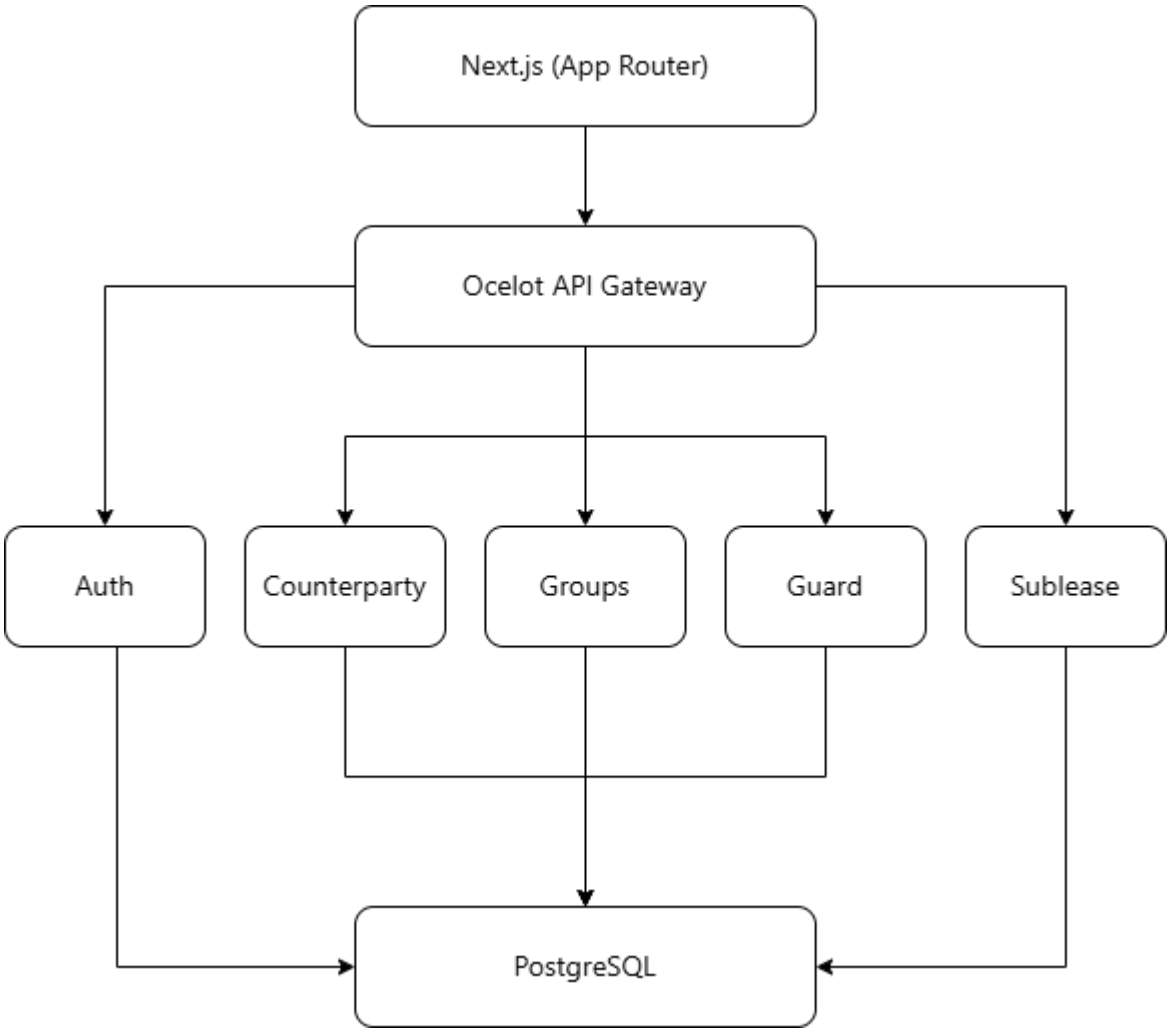

Рис. 3.1. Трирівнева архітектура системи

Як видно з рисунку 3.1, клієнтська частина взаємодіє виключно з API–шлюзом Ocelot Gateway, який маршрутизує запити до п'яти мікросервісів: Auth, Counterparty, Groups, Guard та Sublease. Кожен мікросервіс має власне підключення до спільної бази даних PostgreSQL через Entity Framework Core [9].

Gateway є центральним вузлом серверної архітектури та реалізований на базі бібліотеки Ocelot. При запуску Gateway завантажує конфігурацію маршрутів з файлу ocelot.json, налаштовує JWT–автентифікацію та реєструє два власних middleware. CookieTokenMiddleware відповідає за витяг JWT–токена з HTTP–cookie та додавання його до заголовка Authorization кожного вхідного запиту, що дозволяє фронтенду зберігати токен у cookie замість localStorage. UserIdMiddleware зчитує ідентифікатор користувача з валідованого токена та передає його до мікросервісів через власний заголовок. CORS налаштований на дозвіл запитів з http://localhost:3000 з підтримкою credentials. Після обробки власними middleware та автентифікації запит передається до Ocelot, який виконує фінальну маршрутизацію до цільового мікросервісу.

Таблиця 3.1

Маршрутизація запитів через Ocelot Gateway

URL–шаблон на шлюзі	Мікросервіс	Порт
/api/auth/*	Auth	5237
/api/counterparty/*	Counterparty	5282
/api/groups/*, /api/groupInfo/*	Groups	5189
/api/sublease/*, /api/Document/*	Sublease	5274
/api/guard/*	Guard	5138

Auth – сервіс автентифікації та авторизації. Містить два контролери: AuthController для реєстрації та входу користувачів і JWTController для операцій з токенами. Сервісний шар складається з чотирьох класів: SignUp,SignIn, jwt та httpOnly. AuthController зображений на рисунку 3.2.

Рис. 3.2. Контролер Auth

Counterparty – сервіс управління контрагентами. Містить контролер counterpartyController та сервісний клас CounterpartyService, що реалізує повний CRUD для двох типів контрагентів – ФОП та ТОВ. Всі операції зміни даних виконуються в межах транзакцій бази даних. CounterpartyController зображений на рисунку 3.3.

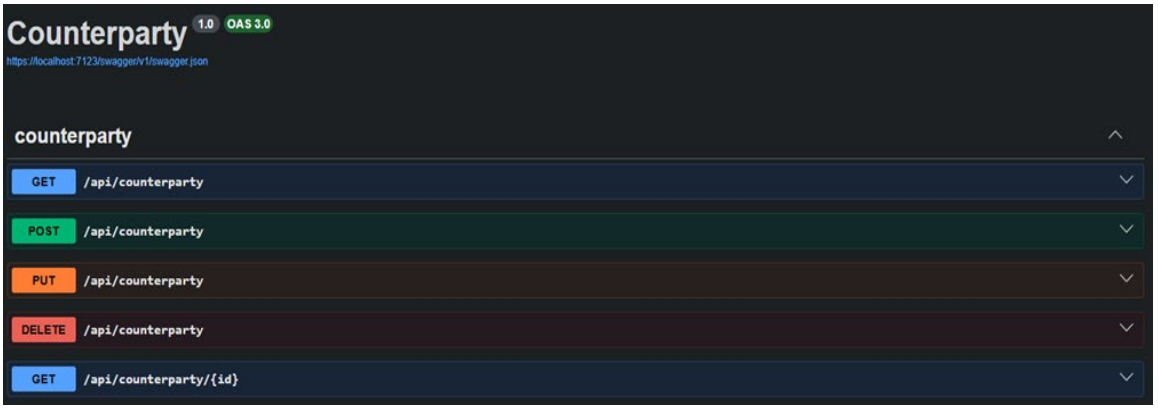

Рис. 3.3. Контролер Counterparty

Groups – сервіс управління відділеннями контрагентів. Містить два контролери: groupsController та groupInfoController для управління інформацією про тип оренди. Має два відповідних сервісних класи: groupsService та GroupInfoService. Groups Controller зображений на рисунку 3.4.

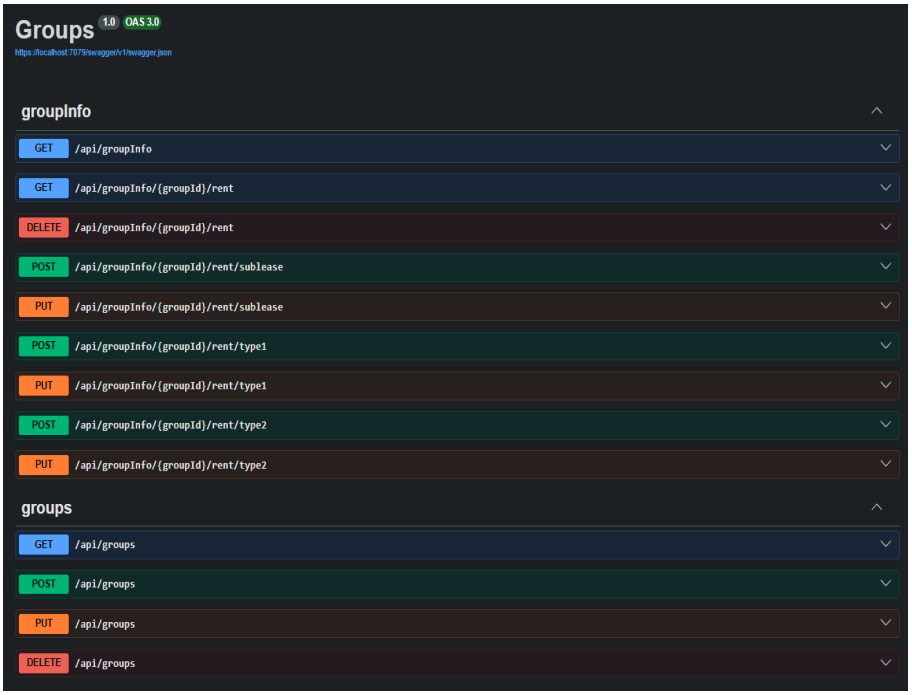

Рис. 3.4. Контролер Groups

Sublease – основний сервіс системи, що реалізує зберігання договорів та генерацію документів. Містить два контролери: subleaseController для CRUD–операцій з договорами та DocumentController для генерації Word–документів. Sublease Controller зображений на рисунку 3.5.

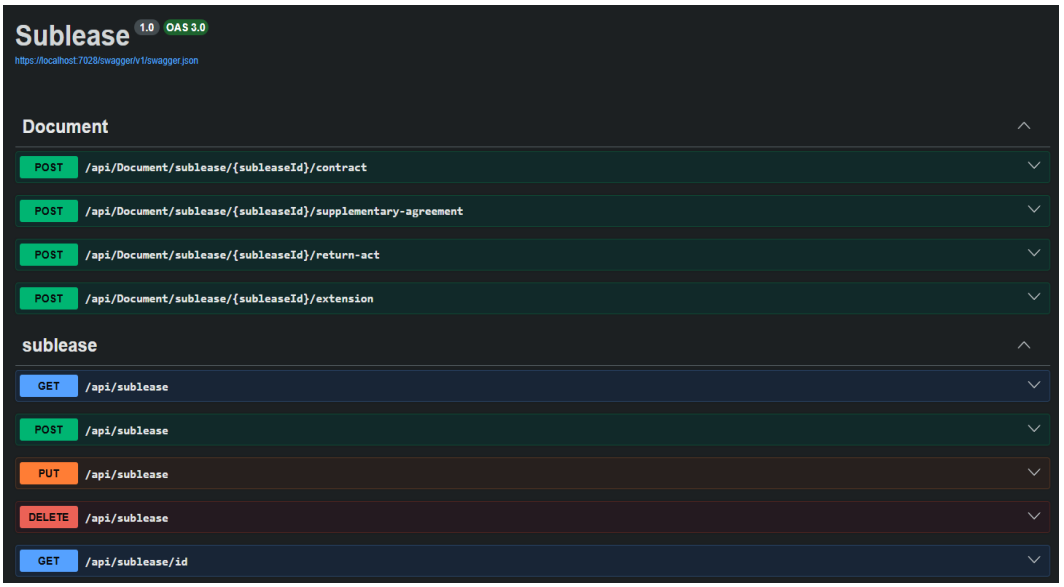

Рис. 3.5. Контролер Sublease

Ієрархія класів генерації документів організована відповідно до діаграми класів, наведеної на рисунку 3.6.

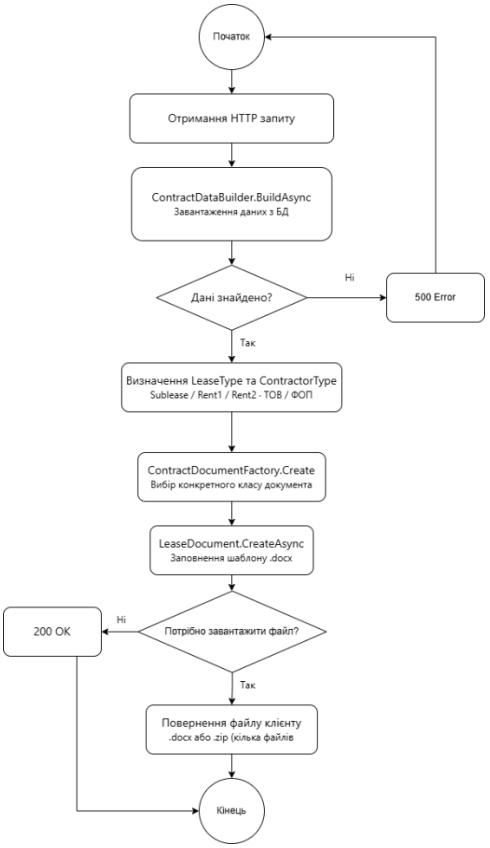

Рис. 3.6 Діаграма класів модуля генерації документів

Як видно з рисунку 3.6, ієрархія класів побудована наступним чином. Абстрактний клас контракту документа (ContractDocument) визначає інтерфейс для всіх типів документів через абстрактний метод асинхронного створення (CreateAsync). Клас орендного документа (LeaseDocument) успадковує його та містить спільну логіку – метод отримання даних (GetDataAsync) для завантаження інформації через будівельника даних контракту (ContractDataBuilder), а також метод збереження (Save) для запису готового файлу. Клас даних контракту (ContractData) є плоским знімком усіх даних, необхідних для заповнення шаблону документа. Конкретні класи документів розділені за двома вимірами: тип оренди (LeaseType: Sublease, Rent1, Rent2) та тип контрагента (ContractorType: Tov, Fop). Статичний клас фабрики (ContractDocumentFactory) через зіставлення зі зразком (pattern matching) на основі комбінації трьох параметрів створює відповідний конкретний клас документа.

Guard – сервіс управління записами охорони відділень. Містить контролер guardController та сервісний клас GuardService. Guard Controller на рисунку 3.7.

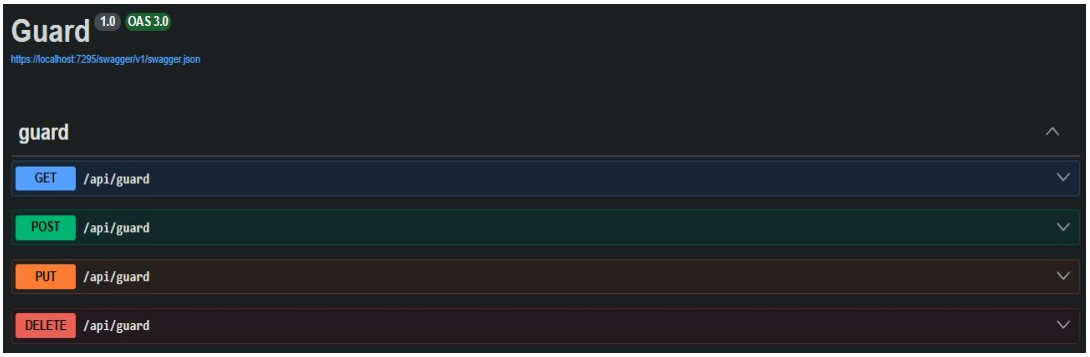

Рис. 3.7. Контролер Guard

SharedLibraries – спільна бібліотека, що підключається до кожного мікросервісу як залежність. Містить моделі даних, контекст бази даних DatabaseModel та допоміжні класи.

Клієнтська частина реалізована на Next.js з використанням файлової маршрутизації через директорію app. Структура проекту організована за функціональним принципом та поділена на кілька логічних шарів.

Маршрутизація сторінок реалізована через дві групи маршрутів. Група (auth) містить сторінки входу та реєстрації з власним layout без навігаційного меню. Група (app) містить захищені сторінки: головна, counterparty, group, sublease та guard.

Шар взаємодії з API організований у папці Api та містить окремі модулі для кожного мікросервісу: Counterparty.ts, Group.ts, guard.ts, sublease.ts та auth/auth.ts.

Управління станом застосунку реалізовано через React Context API [13]. Папка Context містить три пари контекст/провайдер: CounterpartyContext/CounterpartyProvider, GroupContext/GroupProvider та UserContext/UserProvider. Провайдери реєструються у файлі providers.tsx на рівні кореневого layout.

Папка components містить перевикористовувані UI–компоненти: NavMenu, table з підтримкою зміни ширини стовпців через хук useColumnResize, modalwindow для створення та редагування записів кожної сутності, card та theme для перемикача теми оформлення.

3.2 Реалізація алгоритму генерації документів

Алгоритм генерації договірних документів є ключовою функцією розробленої системи та реалізований у мікросервісі Sublease. Процес генерації запускається HTTP–запитом до DocumentController та проходить чотири послідовні етапи: отримання запиту та визначення типу документа, збір даних з бази даних через Builder, визначення конкретного класу документа через Factory, заповнення Word–шаблону та збереження файлу.

3.2.1 Отримання запиту

Контролер документів надає чотири HTTP–ендпоінти з методом POST, кожен з яких відповідає окремому типу договірного документа: основний договір з актом та додатком, додаткова угода, акт повернення та угода про пролонгацію. Усі ендпоінти приймають ідентифікатор договору як параметр шляху та не потребують тіла запиту.

```
[HttpPost("sublease/{subleaseId}/contract")]
public async Task<IActionResult> SubleaseContract(Guid subleaseId)
    => await GenerateAsync(subleaseId, ContractDocumentType.ContractWithActAndAnnex);
[HttpPost("sublease/{subleaseId}/supplementary–agreement")]
public async Task<IActionResult> SubleaseSupplementaryAgreement(Guid subleaseId)
    => await GenerateAsync(subleaseId, ContractDocumentType.SupplementaryAgreement);

[HttpPost("sublease/{subleaseId}/return–act")]
public async Task<IActionResult> SubleaseReturnAct(Guid subleaseId)
    => await GenerateAsync(subleaseId, ContractDocumentType.ReturnAct);

[HttpPost("sublease/{subleaseId}/extension")]
public async Task<IActionResult> SubleaseExtension(Guid subleaseId)
    => await GenerateAsync(subleaseId, ContractDocumentType.ExtensionAgreement);
```

Кожен ендпоінт делегує виконання спільному внутрішньому обробнику,

якому передається ідентифікатор договору та відповідний тип документа. Такий підхід усуває дублювання логіки обробки запитів і централізує обробку помилок в одному місці, забезпечуючи однакову поведінку системи незалежно від типу запитуваного документа.

3.2.2 Збір даних через Builder

На першому етапі генерації документа виконується завантаження всіх необхідних даних з бази даних. Для цього використовується будівельник даних, якому передається ідентифікатор договору. Завантаження здійснюється єдиним запитом до бази даних із застосуванням механізму eager loading [9], що дозволяє отримати пов'язані сутності — групу, інформацію про тип оренди та контрагента — без додаткових звернень до бази даних. У разі відсутності запису з вказаним ідентифікатором система генерує виняток, що зупиняє подальше виконання операції.

Після завантаження даних визначається організаційно–правова форма контрагента на основі типу завантаженого об'єкта. На підставі цього формується плоска модель даних, що містить окремі поля для товариства з обмеженою відповідальністю та для фізичної особи–підприємця. Така структура усуває необхідність повторного визначення типу контрагента на наступних етапах обробки та спрощує доступ до даних у процесі заповнення шаблону.

3.2.3 Визначення класу документа через Factory

На основі вхідних даних запиту система визначає два додаткових параметри, необхідних для вибору відповідного класу документа: тип контрагента та тип оренди. Тип контрагента визначається на підставі ознаки організаційно–правової

форми – фізична особа–підприємець або товариство з обмеженою відповідальністю. Тип оренди встановлюється відповідно до значення, збереженого в моделі групи: суборенда, оренда першого або другого типу. Якщо значення типу оренди не відповідає жодному з очікуваних варіантів, система генерує виняток, що унеможливлює подальше виконання операції з некоректними даними.

Три параметри – тип оренди, тип контрагента та тип документа – передаються до фабричного класу, який на основі їх комбінації за допомогою механізму зіставлення зі зразком повертає відповідний конкретний клас документа. Така архітектура централізує логіку вибору типу документа в одному компоненті та спрощує розширення системи за рахунок додавання нових комбінацій без зміни суміжних модулів.

Фабричний клас підтримує 14 комбінацій параметрів, перелік яких наведено в таблиці 3.2.

Таблиця 3.2

Комбінації параметрів генерації документів

LeaseType	ContractorType	ContractDocumentType	Клас
Sublease	Tov	ContractWithActAndAnnex	SubleaseContractWithActAndAnnexTov
Sublease	Tov	SupplementaryAgreement	SubleaseSupplementaryAgreementTov
Sublease	Tov	ReturnAct	SubleaseReturnActTov
Sublease	Tov	ExtensionAgreement	SubleaseExtensionAgreementTov
Sublease	Fop	ContractWithActAndAnnex	SubleaseContractWithActAndAnnexFop
Sublease	Fop	SupplementaryAgreement	SubleaseSupplementaryAgreementFop
Sublease	Fop	ReturnAct	SubleaseReturnActFop

Продовження таблиці 3.2

LeaseType	ContractorType	ContractDocumentType	Клас
Sublease	Fop	ExtensionAgreement	SubleaseExtensionAgreementFop
Rent1	Tov	ContractWithActAndAnnex	Rent1ContractWithActAndAnnexTov
Rent1	Tov	SupplementaryAgreement	Rent1SupplementaryAgreementTov
Rent1	Tov	ReturnAct	Rent1ReturnActTov
Rent1	Fop	ContractWithActAndAnnex	Rent1ContractWithActAndAnnexFop
Rent2	Tov	ContractWithActAndAnnex	Rent2ContractWithActAndAnnexTov
Rent2	Fop	ContractWithActAndAnnex	Rent2ContractWithActAndAnnexFop

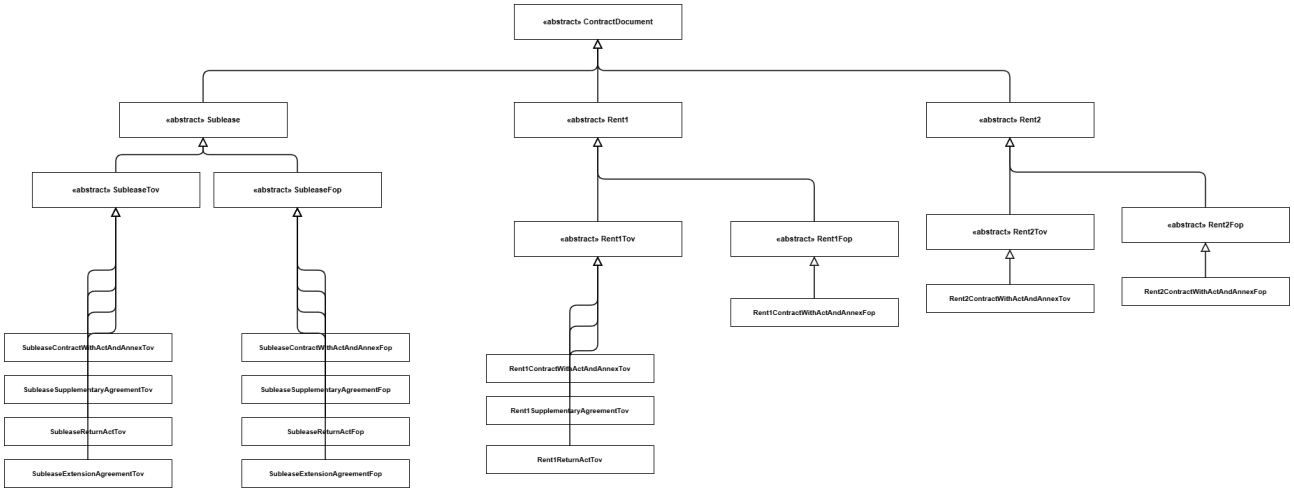

Рис. 3.8. Діаграма комбінацій генерації документів

3.2.4 Заповнення шаблону та збереження

Кожен клас документа реалізує логіку заповнення Word–шаблону шляхом послідовної підстановки значень у відповідні поля. Для кожного документа створюється екземпляр процесора, якому передається ключ конфігурації, що визначає шлях до відповідного шаблону. Процесор знаходить кожен Content

Control у структурі документа за його тегом та замінює його на відповідне значення з бізнес–моделі. Для складених типів документів, що передбачають одночасне формування кількох файлів, наприклад договору, акту та додатку, процес заповнення виконується окремо для кожного з них у межах одного запиту.

Перед підстановкою значення проходять форматування відповідно до вимог документа. Числові значення орендної плати відображаються з двома знаками після коми та комою як роздільником дробової частини. Ідентифікатор орендодавця формується залежно від наявних реєстраційних даних: за наявності індивідуального податкового номера використовується позначення РНОКПП, за наявності коду юридичної особи – ЄДРПОУ. Підстава права власності формується відповідно до типу оренди: для першого типу використовуються серія та номер свідоцтва, для другого – дата та індексний номер витягу з Єдиного державного реєстру.

Сформований документ зберігається у структурованій директорії файлової системи сервера. Шлях до файлу будується за ієрархічним принципом: на першому рівні розташована директорія групи, що містить її номер та назву, на другому – безпосередньо файл документа, ідентифікований номером договору, назвою групи та типом документа. Назва групи попередньо очищується від символів, що є недопустимими в іменах файлів і директорій операційної системи Windows, що забезпечує коректне збереження незалежно від найменування контрагента.

У разі успішного виконання всіх етапів сервер повертає відповідь з HTTP–статусом 200, що підтверджує створення документа. У разі виникнення помилки на будь–якому з етапів повертається статус 500 з описом помилки, що забезпечує прозорість діагностики в процесі експлуатації системи.

3.3 Розгортання системи

Розгортання розробленої системи реалізовано засобами контейнеризації Docker. Кожен мікросервіс упакований у власний Docker–контейнер на основі відповідного Dockerfile, що забезпечує ізольоване середовище виконання та

незалежність від конфігурації хост–машини. Оркестрація всіх контейнерів здійснюється через docker–compose, що дозволяє запустити повну систему однією командою [33].

Таблиця 3.3

Склад контейнерів системи

Контейнер	Образ/ Runtime	Порт (internal)	Порт (external)	Залежності
next–app	node:20 –alpine	3000	3000	gateway
gateway	.NET 10 / Ocelot	5000	5000	auth, counterparty, groups, guard, sublease
auth	.NET 10	5001	–	postgres
counterpar ty	.NET 10	5002	–	postgres
groups	.NET 10	5003	–	postgres
guard	.NET 10	5004	–	postgres
sublease	.NET 10	5005	–	postgres
postgres	postgres :16– alpine	5432	–	volume: pg– data

Зовні доступні лише два контейнери – next–app на порту 3000 та gateway на порту 5000. Всі мікросервіси (auth, counterparty, groups, guard, sublease) працюють

у внутрішній Docker–мережі та не мають прямого доступу ззовні, що відповідає принципу мінімальних привілеїв та підвищує безпеку системи. База даних PostgreSQL також доступна лише всередині мережі контейнерів.

Для збереження даних між перезапусками контейнерів використовується named volume pg–data, що монтується до директорії даних PostgreSQL. Це гарантує що дані бази даних не будуть втрачені при оновленні або перезапуску контейнера postgres.

Порядок запуску контейнерів визначається директивою depends_on у конфігурації docker–compose : спочатку запускається postgres, потім мікросервіси, далі gateway і останнім next–app. Такий порядок гарантує що кожен сервіс при старті знайде свої залежності вже доступними.прав між менеджерами, юристами та адміністраторами системи.

3.4 Рекомендації щодо впровадження

Перед розгортанням системи у виробничому середовищі обмінника валют необхідно врахувати низку ризиків та вжити відповідних заходів для забезпечення надійної та безпечної роботи.

Таблиця 3.4

Ризики впровадження та рекомендовані заходи

Ризик	Рівень	Рекомендований захід
Втрата даних БД при оновленні контейнера	Високий	Налаштувати named volume pg–data, регулярні pg_dump резервні копії
Витік JWT токена через незахищений транспорт	Високий	Обов'язково використовувати HTTPS (TLS), налаштувати HttpOnly cookie з Secure–прапором

Продовження таблиці 3.4

Ризик	Рівень	Рекомендований захід
Недоступність одного мікросервісу блокує інші	Середній	Додати health–check у docker–compose, розглянути retry–policy через Polly (вже є в Gateway)
Шаблони документів відсутні або пошкоджені	Середній	Монтувати шаблони як окремий volume, перевіряти при старті сервісу Sublease
Конфліктуючі міграції БД між сервісами	Середній	Всі сервіси використовують SharedLibraries.dll – міграції застосовувати лише через один сервіс–мігратор
Збільшення навантаження при масовій генерації документів	Низький	Розглянути чергу задач (RabbitMQ / Hangfire) для асинхронної генерації
Помилка типізації контрагента (FOP/TOV)	Низький	EF Core TPH–дискримінатор гарантує правильний тип – перевірити конфігурацію DbContext

Окрім усунення виявлених ризиків, для подальшого розвитку системи рекомендується розглянути наступні напрямки вдосконалення. По–перше, інтеграція електронного підпису – додавання підтримки кваліфікованого електронного підпису (КЕП) до згенерованих договорів дозволить повністю відмовитись від паперового документообігу та надасть документам повну юридичну силу відповідно до законодавства України. По–друге, система сповіщень – автоматичне сповіщення відповідальних осіб про наближення дати закінчення

договору на основі поля ContractEndDate, що дозволить вчасно ініціювати процедуру продовження або повернення об'єкта. По–третє, розмежування прав доступу – впровадження рольової моделі (RBAC) з розподілом прав між менеджерами, юристами та адміністраторами системи.

Загалом розроблений застосунок є готовим до впровадження у середовищі обмінника валют з розгалуженою мережею відділень та великою кількістю контрагентів. Мікросервісна архітектура забезпечує можливість незалежного масштабування окремих компонентів системи у міру зростання навантаження, а використання стандартних Word–шаблонів з Content Controls гарантує що юридичний відділ зможе самостійно оновлювати форми договорів без залучення розробників.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено web–застосунок для автоматизації договірного документообігу в обміннику валют, що вирішує практичну проблему ручного заповнення договорів оренди та суборенди нерухомості. За результатами роботи можна зробити наступні висновки.

1. Проведено аналіз існуючих рішень для автоматизації документообігу. Встановлено, що універсальні продукти, наявні на ринку, є надмірно дорогими для середнього бізнесу та не враховують специфіку мережевих обмінників — різні правові форми контрагентів та розгалужену ієрархію відділень. Це підтвердило необхідність розробки спеціалізованого рішення.
2. Сформовано перелік функціональних та нефункціональних вимог до системи з урахуванням реальних операційних потреб мережі пунктів обміну валют: швидкість оформлення документів, точність даних, розмежування прав доступу персоналу та підтримка різних типів орендних відносин.
3. Спроектовано масштабовану архітектуру системи, що дозволяє підключати нові відділення та розширювати функціонал без зупинки роботи та переписування існуючих компонентів. Розроблено модель бази даних, що коректно відображає різні правові форми контрагентів та типи орендних об'єктів.
4. Реалізовано серверну частину системи з централізованим управлінням доступом, що забезпечує безпечну роботу персоналу різних відділень у єдиному середовищі без ризику несанкціонованого доступу до даних.
5. Розроблено зручний веб–інтерфейс, що не потребує технічних знань від персоналу та забезпечує швидке введення даних контрагентів і об'єктів оренди.
6. Реалізовано модуль автоматичної генерації договорів у форматі Word, який підтримує чотирнадцять комбінацій типів документів. Юридичний

відділ може самостійно редагувати шаблони без залучення розробників, що усуває операційну залежність бізнесу від IT–команди.

7. Проведено тестування системи та оцінено практичну ефективність її впровадження. Визначено сім ризиків впровадження та надано рекомендації щодо їх мінімізації. Система є повністю готовою до експлуатації.

Практичне значення розробленого застосунку полягає у вирішенні реальної операційної проблеми бізнесу: скороченні часу на підготовку договірної документації, усуненні помилок через людський фактор та забезпеченні єдиного актуального сховища даних про контрагентів та орендовані об'єкти. Перспективами подальшого розвитку є інтеграція кваліфікованого електронного підпису та реалізація автоматичних сповіщень про закінчення термінів договорів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про електронні документи та електронний документообіг : Закон України від 22.05.2003 р. № 851–IV. URL: https://zakon.rada.gov.ua/laws/show/851–15 (дата звернення: 17.03.2026).
2. Про оренду державного та комунального майна : Закон України від 03.10.2019 р. № 157–IX. URL: https://zakon.rada.gov.ua/laws/show/157–20 (дата звернення: 17.03.2026).
3. Microsoft. ASP.NET Core documentation. URL: https://learn.microsoft.com/en–us/aspnet/core (дата звернення: 10.03.2026).
4. React documentation. URL: https://react.dev (дата звернення: 11.03.2026).
5. Ocelot API Gateway documentation. URL: https://ocelot.readthedocs.io (дата звернення: 12.03.2026).
6. Microsoft. Open XML SDK documentation. URL: https://learn.microsoft.com/en–us/office/open–xml/open–xml–sdk (дата звернення: 16.03.2026).
7. Microsoft. Entity Framework Core – Inheritance (TPH, TPT, TPC). URL: https://learn.microsoft.com/en–us/ef/core/modeling/inheritance (дата звернення: 10.03.2026).
8. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 15.03.2026).
9. VII Всеукраїнська науково–технічна конференція “Застосування програмного забезпечення в інформаційно–комунікаційних технологіях”. Визначення вимог до клієнт–серверної вебплатформи для автоматизації процесів оренди фінансової компанії.
10. Всеукраїнська науково–технічна конференція “Технології цифрового розвитку: програмні системи та децентралізація”. Клієнт–серверна вебплатформа для автоматизації процесів оренди фінансової компанії з використанням екосистем .NET та React/Next.js.

11. PandaDoc : веб–сайт. URL: https://www.pandadoc.com (дата звернення: 19.03.2026).
12. DocuSign : веб–сайт. URL: https://www.docusign.com (дата звернення: 19.03.2026).
13. Zoho Contracts : веб–сайт. URL: https://www.zoho.com/contracts (дата звернення: 19.03.2026).
14. Vue.js Documentation. URL: https://vuejs.org/guide/introduction (дата звернення: 29.04.2026).
15. Angular Documentation. URL: https://angular.dev/overview (дата звернення: 29.04.2026).
16. Vercel. Next.js documentation. URL: https://nextjs.org/docs (дата звернення: 11.03.2026).
17. PHP Documentation. URL: https://www.php.net/docs.php (дата звернення: 12.03.2026).
18. Go Documentation. URL: https://go.dev/doc/ (дата звернення: 12.03.2026).
19. Node.js Documentation. URL: https://nodejs.org/en/docs (дата звернення: 12.03.2026).
20. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 12.03.2026).
21. Monolithic Architecture. IBM. URL: https://www.ibm.com/topics/monolithic–architecture (дата звернення: 12.03.2026).
22. Microservices Architecture. IBM. URL: https://www.ibm.com/topics/microservices (дата звернення: 13.03.2026).
23. Modular Monolith. Martin Fowler. URL: https://martinfowler.com/bliki/MonolithFirst.html (дата звернення: 13.03.2026).
24. Service–Oriented Architecture. IBM. URL: https://www.ibm.com/topics/soa (дата звернення: 13.03.2026).
25. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16 (дата звернення: 13.03.2026).

26. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 14.03.2026).
27. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 17.03.2026).
28. XDoc : бібліотека для роботи з Word–шаблонами на платформі .NET. GitHub. URL: https://github.com/lxthxrgx/XDoc (дата звернення: 22.03.2026).
29. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object–Oriented Software. New York : Addison–Wesley, 1994. 395 p.
30. Мартін Р. К. Чиста архітектура. Посібник майстра з програмної структури та дизайну. Prentice Hall, 2017. 432 с.
31. Evans E. Domain–Driven Design: Tackling Complexity in the Heart of Software. Addison–Wesley, 2003. 560 p.
32. ASP.NET Core Middleware. Microsoft : веб–сайт. URL: https://learn.microsoft.com/en–us/aspnet/core/fundamentals/middleware/ (дата звернення: 21.04.2026).
33. Docker documentation. URL: https://docs.docker.com (дата звернення: 22.03.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

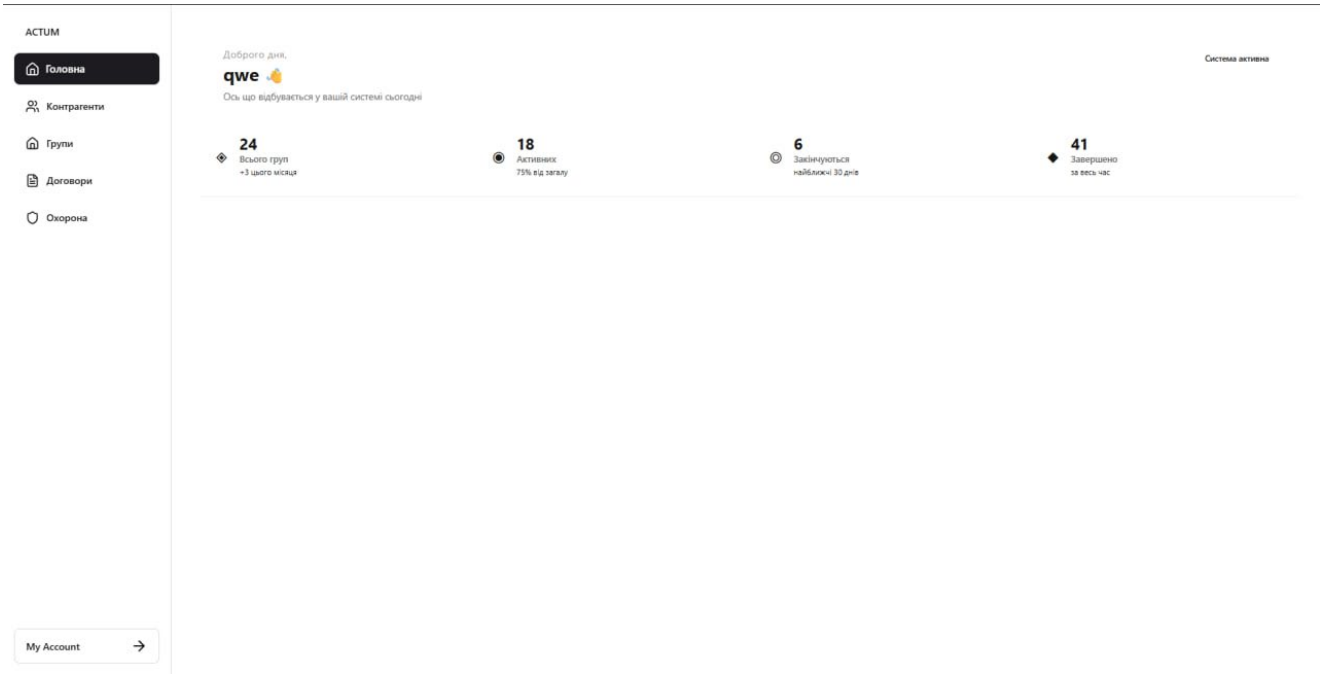

Рис. А.1. Головна сторінка

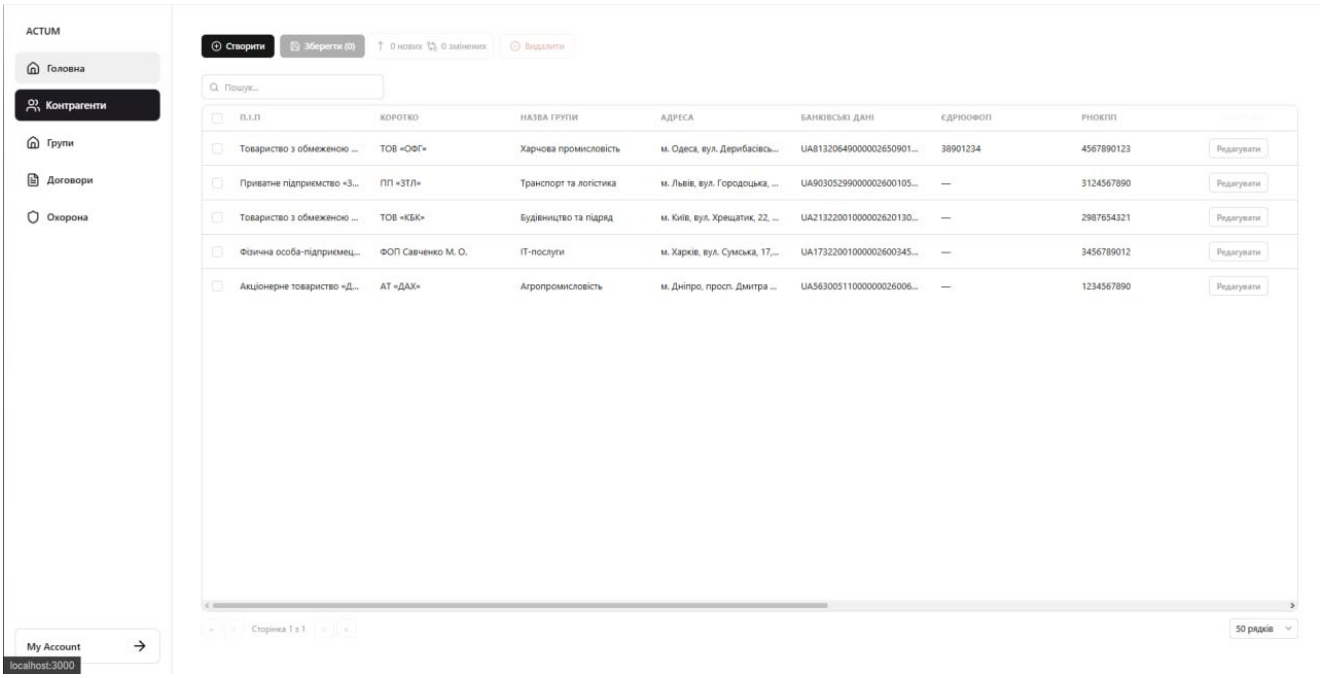

Рис. А.2. Сторінка контрагентів

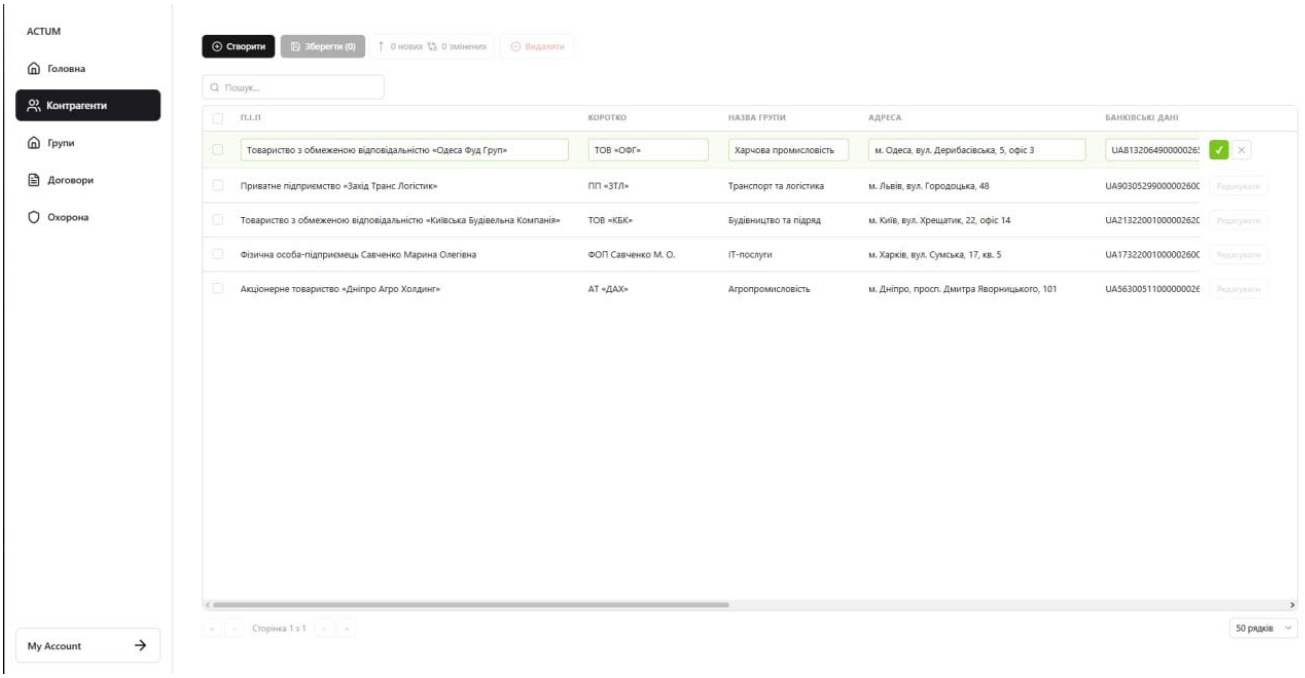

Рис. А.3. Режим редагування запису

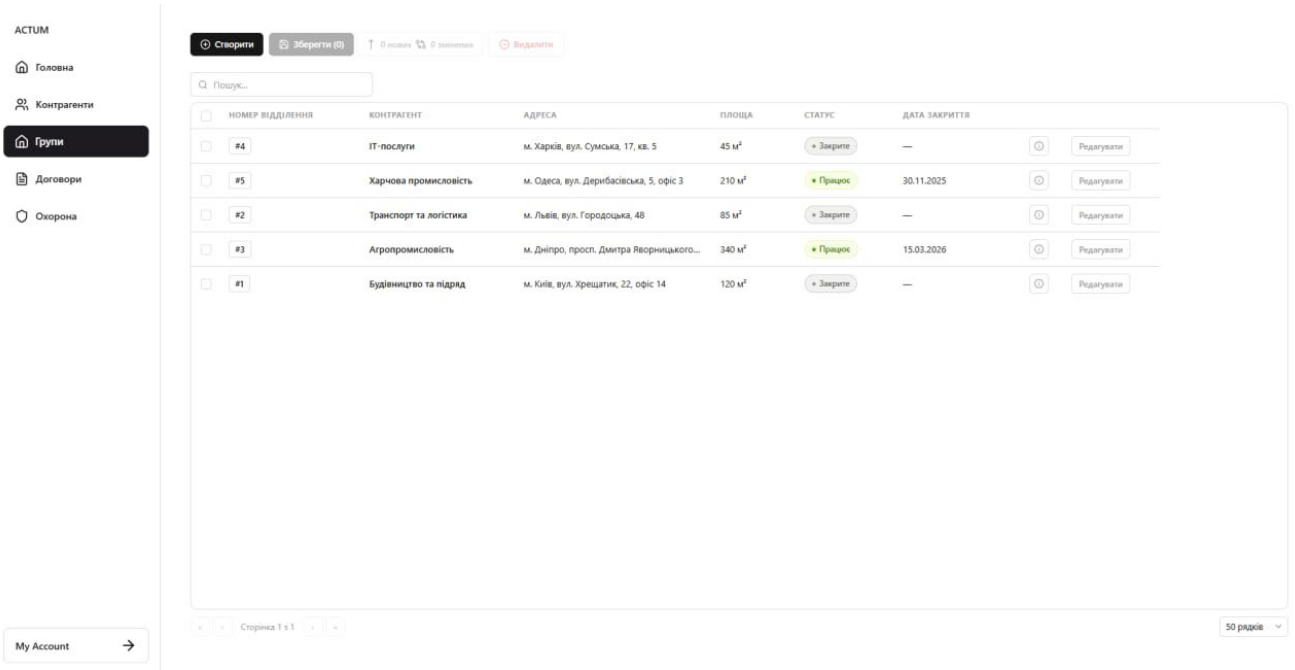

Рис. А.4. Сторінка груп

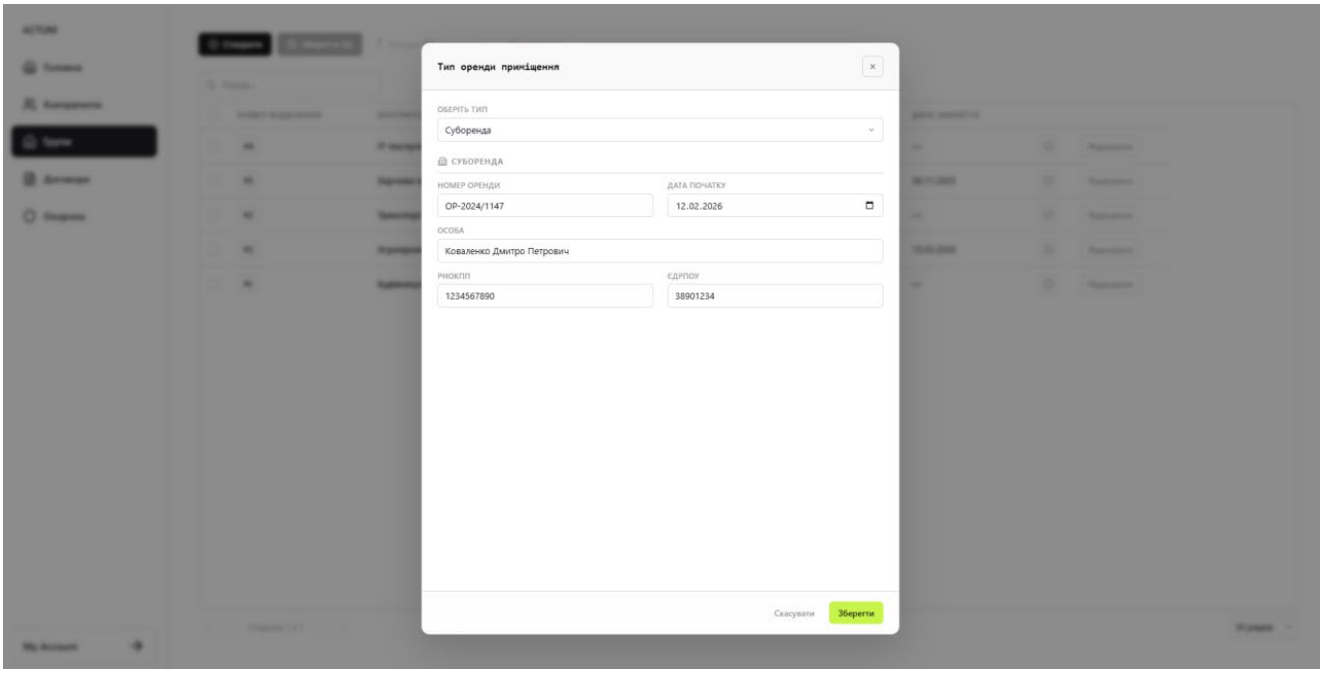

Рис. А.5. Модальне вікно типу оренди

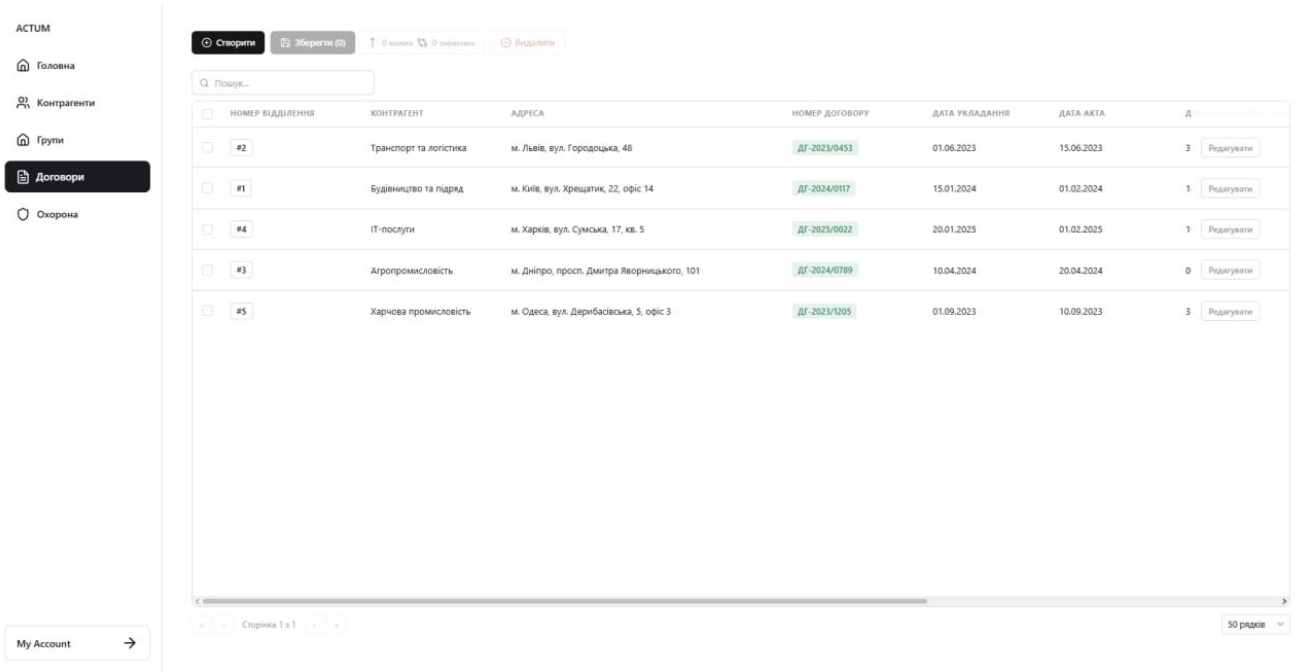

Рис. А.6. Сторінка договорів

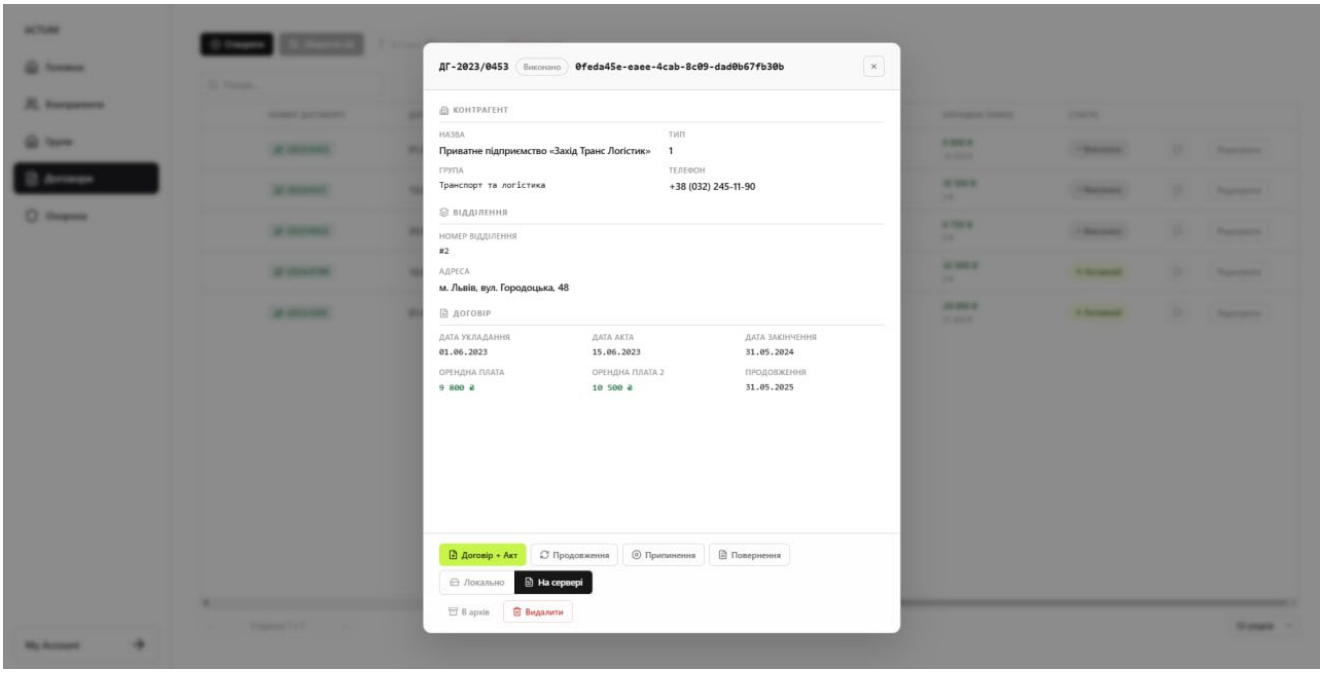

Рис. А.7. Модальне вікно договору

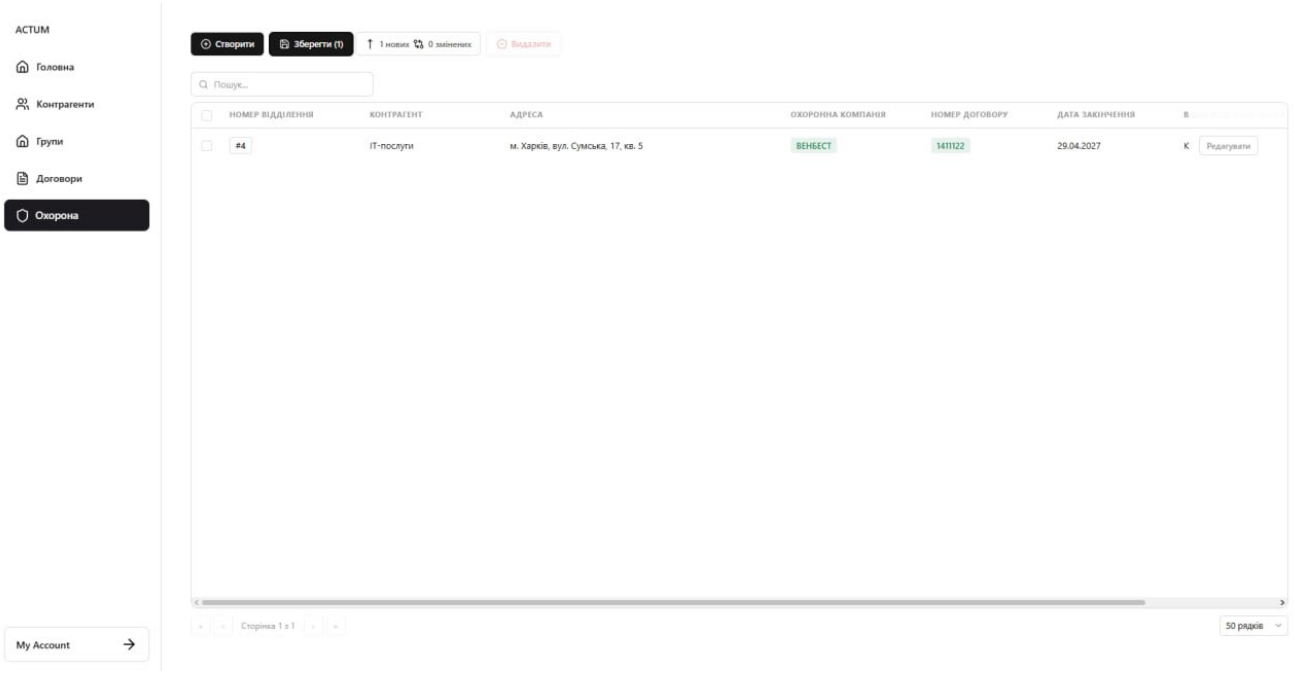

Рис. А.8. Сторінка договорів охорони

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Лістинг Б.1 — DocFactory.cs

```
using Microsoft.EntityFrameworkCore;
using SharedLibraries.components;
using SharedLibraries.Database;
using SharedLibraries.model;
using System.Text.RegularExpressions;
using XDoc;

namespace Lease.Factory
{

   public enum LeaseType { Sublease, Rent1, Rent2 }
   public enum ContractorType { Tov, Fop }
   public enum ContractDocumentType
   {
      ContractWithActAndAnnex,
      SupplementaryAgreement,
      ReturnAct,
      ExtensionAgreement
   }

   public class ContractData
   {
      public GroupModel Group { get; set; } = null!;
      public SubleaseModel Sublease { get; set; } = null!;
      public CounterpartyLLC? CounterpartyLLC { get; set; }
      public CounterpartyFop? CounterpartyFop { get; set; }

      public bool IsFop => CounterpartyFop != null;

      public GroupRentInfo? RentInfo => Group.RentInfo;
   }

   public class ContractDataBuilder
   {
      private readonly DatabaseModel _context;
      public ContractDataBuilder(DatabaseModel context) => _context = context;

      public async Task<ContractData> BuildAsync(Guid subleaseId)
      {
         var sublease = await _context.Subleases
            .Include(s => s.Group)
               .ThenInclude(g => g.RentInfo)
            .Include(s => s.Group)
               .ThenInclude(g => g.Counterparty)
            .FirstOrDefaultAsync(s => s.Id == subleaseId)
            ?? throw new InvalidOperationException($"Sublease {subleaseId} not found.");

         if (sublease.Group.Counterparty == null)
         {
            await _context.Entry(sublease.Group)
               .Reference(g => g.Counterparty)
               .LoadAsync();
         }

         var data = new ContractData
         {
```

```
            Group = sublease.Group,
            Sublease = sublease
        };

        switch (sublease.Group.Counterparty)
        {
            case CounterpartyLLC llc: data.CounterpartyLLC = llc; break;
            case CounterpartyFop fop: data.CounterpartyFop = fop; break;
            default:
                throw new InvalidOperationException(
                    $"Unknown counterparty type: {sublease.Group.Counterparty?.GetType().Name}");
        }

        return data;
    }
}

internal static class DocHelper
{
    public static string Doublezero(double num) =>
        $"{num:N2}".Replace('.', ',');

    public static string SanitizeName(string name) =>
        new string(name.Where(c => !Path.GetInvalidFileNameChars().Contains(c)).ToArray());

    public static string RenameGog(string name)
    {
        var invalid = new string(Path.GetInvalidFileNameChars());
        return Regex.Replace(name, "[" + Regex.Escape(invalid) + "]", "_");
    }

    public static string GetSublessorId(SubleaseRentInfo info)
    {
        if (!string.IsNullOrEmpty(info.Rnokpp)) return "РНОКПП " + info.Rnokpp;
        if (!string.IsNullOrEmpty(info.Edrpou)) return "ЄДРПОУ " + info.Edrpou;
        return "_____";
    }

    public static string BuildNaPidType1(RentType1Info info) =>
        $"свідоцтва про право власності серії {info.SeriesCert} " +
        $"номер {info.CertNumber}, виданого {info.Issued}.";

    public static string BuildNaPidType2(RentType2Info info) =>
        $"Витягу з ЄДР речових прав на нерухоме майно про реєстрацію прав та їх обтяжень " +
        $"від {info.Date:dd/MM/yyyy} індексний номер {info.Num}";

    public static string SavePath(GroupModel group, string groupName, string fileName) =>
        $"{group.NumberGroup}–{SanitizeName(groupName)}\\{RenameGog(fileName)}";
}


public abstract class ContractDocument
{
    protected readonly DatabaseModel _context;
    protected ContractDocument(DatabaseModel context) => _context = context;
    public abstract Task CreateAsync(Guid subleaseId);
}

public abstract class LeaseDocument : ContractDocument
{
```

```
        protected LeaseDocument(DatabaseModel context) : base(context) { }

        protected async Task<ContractData> GetDataAsync(Guid subleaseId)
        {
            var data = await new ContractDataBuilder(_context).BuildAsync(subleaseId);
            return data;
        }

        protected XPathProcessor CreateProcessor(string configKey) =>
            new XPathProcessor(ConfigHelper.Configuration[configKey],
ConfigHelper.Configuration["CustomSettings:path"]);

        protected void Save(XPathProcessor p, ContractData data, string fileName)
        {
            var groupName = data.CounterpartyLLC?.GroupName
                    ?? data.CounterpartyFop?.GroupName
                    ?? "unknown";
            p.Save(DocHelper.SavePath(data.Group, groupName, fileName));
        }
    }

    public class SubleaseContractWithActAndAnnexTov : LeaseDocument
    {
        public SubleaseContractWithActAndAnnexTov(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");
            var rentInfo = data.RentInfo as SubleaseRentInfo;

            var dog = CreateProcessor("sublease–tov:sublease–agreement–tov");
            dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dog.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dog.WriteXmlTree("PIB", llc.Fullname);
            dog.WriteXmlTree("rnokpp", llc.Rnokpp);
            dog.WriteXmlTree("area", data.Group.Area.ToString());
            dog.WriteXmlTree("address_p", data.Group.Address);
            dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate);
            dog.WriteXmlTree("address", llc.Address);
            dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
            dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            dog.WriteXmlTree("Director", llc.Director);
            dog.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            dog.WriteXmlTree("subleaseDopNum", rentInfo?.RentNumber ?? "____");
            dog.WriteXmlTree("subleaseDopDate", rentInfo?.StartDate.ToString("dd/MM/yyyy") ?? "____");
            dog.WriteXmlTree("subleaseDopName", rentInfo?.Person ?? "____");
            dog.WriteXmlTree("subleaseDopRnokpp", rentInfo != null ? DocHelper.GetSublessorId(rentInfo) :
"____");
            dog.WriteXmlTree("subleaseDopStatus", rentInfo?.Edrpou ?? "____");
            dog.Save($"{data.Sublease.ContractNumber}–{llc.GroupName}–договір");

            var akt = CreateProcessor("sublease–tov:sublease–act–tov");
            akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            akt.WriteXmlTree("PIB", llc.Fullname);
```

```
            akt.WriteXmlTree("rnokpp", llc.Rnokpp);
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("address", llc.Address);
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            akt.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            akt.WriteXmlTree("Director", llc.Director);
            akt.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            akt.Save($"{data.Sublease.ContractNumber}–{llc.GroupName}–акт");

            var dod = CreateProcessor("sublease–tov:supplement–tov");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", llc.Fullname);
            dod.WriteXmlTree("address", llc.Address);
            dod.WriteXmlTree("rnokpp", llc.Rnokpp);
            dod.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            dod.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            dod.Save($"{data.Sublease.ContractNumber}–{llc.GroupName}–додаток");
        }
    }

    public class SubleaseSupplementaryAgreementTov : LeaseDocument
    {
        public SubleaseSupplementaryAgreementTov(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");

            var p = CreateProcessor("sublease–tov:sublease–termination–tov");
            p.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.AddDays(–1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("PIB", llc.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", llc.Rnokpp);
            p.WriteXmlTree("address", llc.Address);
            p.WriteXmlTree("Director", llc.Director);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            p.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(p, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–припинення–ТОВ");
        }
    }

    public class SubleaseReturnActTov : LeaseDocument
    {
        public SubleaseReturnActTov(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
```

```
            var data = await GetDataAsync(subleaseId);
            var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");

            var p = CreateProcessor("sublease–tov:sublease–return–act–tov");
            p.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.ContractEndDate.ToString("dd/MM/yyyy")));
            p.WriteXmlTree("PIB", llc.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", llc.Rnokpp);
            p.WriteXmlTree("address", llc.Address);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            p.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            p.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            p.WriteXmlTree("Director", llc.Director);
            p.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(p, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–повернення–ТОВ");
        }
    }

    public class SubleaseExtensionAgreementTov : LeaseDocument
    {
        public SubleaseExtensionAgreementTov(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");
            var rentInfo = data.RentInfo as SubleaseRentInfo;

            var p = CreateProcessor("sublease–tov:extension–contract–tov");
            p.WriteXmlTree("ContractNumber", data.Sublease.ContractNumber);
            p.WriteXmlTree("CreationContractDate", data.Sublease.ContractSigningDate.ToString("dd.MM.yyyy"));
            p.WriteXmlTree("CreationDate", data.Sublease.ContractEndDate.ToString("dd.MM.yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate2?.ToString("dd.MM.yyyy") ?? "____");
            p.WriteXmlTree("PipSublessor", llc.Fullname);
            p.WriteXmlTree("rnokppSublessor", llc.Rnokpp);
            p.WriteXmlTree("addressSublessor", llc.Address);
            p.WriteXmlTree("PipDirector", llc.Director);
            p.WriteXmlTree("PipsDirector", llc.ShortNameDirector);
            p.WriteXmlTree("RoomArea", data.Group.Area.ToString());
            p.WriteXmlTree("RoomAreaText", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("RoomAreaAddress", data.Group.Address);
            p.WriteXmlTree("subleaseDopContractNumber", rentInfo?.RentNumber ?? "____");
            p.WriteXmlTree("subleaseDopStartDate", rentInfo?.StartDate.ToString("dd.MM.yyyy") ?? "____");
            p.WriteXmlTree("subleaseDopName", rentInfo?.Person ?? "____");
            p.WriteXmlTree("rnokppSublessorDop", rentInfo != null ? DocHelper.GetSublessorId(rentInfo) : "____");
            p.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            Save(p, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–продовження");
        }
    }

    public class SubleaseContractWithActAndAnnexFop : LeaseDocument
    {
```

```
        public SubleaseContractWithActAndAnnexFop(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");
            var rentInfo = data.RentInfo as SubleaseRentInfo;

            var dog = CreateProcessor("sublease:sublease–agreement");
            dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dog.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dog.WriteXmlTree("PIB", fop.Fullname);
            dog.WriteXmlTree("edruofop_Data", fop.Edryofop);
            dog.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            dog.WriteXmlTree("area", data.Group.Area.ToString());
            dog.WriteXmlTree("address_p", data.Group.Address);
            dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            dog.WriteXmlTree("address", fop.Address);
            dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
            dog.WriteXmlTree("PIBS", fop.ResPerson);
            dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            dog.WriteXmlTree("subleaseDopNum", rentInfo?.RentNumber ?? "____");
            dog.WriteXmlTree("subleaseDopDate", rentInfo?.StartDate.ToString("dd/MM/yyyy") ?? "____");
            dog.WriteXmlTree("subleaseDopName", rentInfo?.Person ?? "____");
            dog.WriteXmlTree("subleaseDopRnokpp", rentInfo != null ? DocHelper.GetSublessorId(rentInfo) :
"____");
            dog.WriteXmlTree("subleaseDopStatus", rentInfo?.Edrpou ?? "____");
            Save(dog, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–договір–ФОП");

            var akt = CreateProcessor("sublease:sublease–act");
            akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            akt.WriteXmlTree("PIB", fop.Fullname);
            akt.WriteXmlTree("edruofop_Data", fop.Edryofop);
            akt.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("address", fop.Address);
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
            akt.WriteXmlTree("PIBS", fop.ResPerson);
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            akt.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            Save(akt, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–акт–ФОП");

            var dod = CreateProcessor("sublease:supplement–fop");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", fop.Fullname);
            dod.WriteXmlTree("address", fop.Address);
            dod.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            dod.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            dod.WriteXmlTree("PIBS", fop.ResPerson);
            Save(dod, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–додаток–ФОП");
```

```
        }
    }

    public class SubleaseSupplementaryAgreementFop : LeaseDocument
    {
        public SubleaseSupplementaryAgreementFop(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");

            var p = CreateProcessor("sublease:sublease–termination");
            p.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.AddDays(–1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("PIB", fop.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            p.WriteXmlTree("address", fop.Address);
            p.WriteXmlTree("edruofop_Data", fop.Edryofop);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            p.WriteXmlTree("PIBS", fop.ResPerson);
            Save(p, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–припинення–ФОП");
        }
    }

    public class SubleaseReturnActFop : LeaseDocument
    {
        public SubleaseReturnActFop(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");

            var p = CreateProcessor("sublease:sublease–return–act");
            p.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.ContractEndDate.ToString("dd/MM/yyyy")));
            p.WriteXmlTree("PIB", fop.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            p.WriteXmlTree("address", fop.Address);
            p.WriteXmlTree("edruofop_Data", fop.Edryofop);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            p.WriteXmlTree("PIBS", fop.ResPerson);
            p.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            p.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            Save(p, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–повернення–ФОП");
```

```
    }
}

public class SubleaseExtensionAgreementFop : LeaseDocument
{
    public SubleaseExtensionAgreementFop(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");
        var rentInfo = data.RentInfo as SubleaseRentInfo;

        var p = CreateProcessor("sublease:extension–contract–fop");
        p.WriteXmlTree("ContractNumber", data.Sublease.ContractNumber);
        p.WriteXmlTree("CreationContractDate", data.Sublease.ContractSigningDate.ToString("dd.MM.yyyy"));
        p.WriteXmlTree("CreationDate", data.Sublease.ContractEndDate.ToString("dd.MM.yyyy"));
        p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate2?.ToString("dd.MM.yyyy") ?? "____");
        p.WriteXmlTree("PipSublessor", fop.Fullname);
        p.WriteXmlTree("rnokppSublessor", fop.Rnokpp ?? "");
        p.WriteXmlTree("addressSublessor", fop.Address);
        p.WriteXmlTree("Edruofop", fop.Edryofop);
        p.WriteXmlTree("PipsSublessor", fop.ResPerson);
        p.WriteXmlTree("RoomArea", data.Group.Area.ToString());
        p.WriteXmlTree("RoomAreaText", NumToText.NumberToText(data.Group.Area));
        p.WriteXmlTree("RoomAreaAddress", data.Group.Address);
        p.WriteXmlTree("subleaseDopContractNumber", rentInfo?.RentNumber ?? "____");
        p.WriteXmlTree("subleaseDopStartDate", rentInfo?.StartDate.ToString("dd.MM.yyyy") ?? "____");
        p.WriteXmlTree("subleaseDopName", rentInfo?.Person ?? "____");
        p.WriteXmlTree("rnokppSublessorDop", rentInfo != null ? DocHelper.GetSublessorId(rentInfo) : "____");
        p.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
        Save(p, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–продовження");
    }
}

public class Rent1ContractWithActAndAnnexTov : LeaseDocument
{
    public Rent1ContractWithActAndAnnexTov(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");
        var rentInfo = data.RentInfo as RentType1Info ?? throw new InvalidOperationException("Expected
RentType1Info.");
        var naPid = DocHelper.BuildNaPidType1(rentInfo);

        var dog = CreateProcessor("rent–tov:rent–dog–tov");
        dog.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
        dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("PIB", llc.Fullname);
        dog.WriteXmlTree("rnokpp", llc.Rnokpp);
        dog.WriteXmlTree("area", data.Group.Area.ToString());
        dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
        dog.WriteXmlTree("address_p", data.Group.Address);
        dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("address", llc.Address);
        dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
        dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
        dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
```

```
            dog.WriteXmlTree("Director", llc.Director);
            dog.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            dog.WriteXmlTree("NaPid", naPid);
            Save(dog, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–договір");

            var akt = CreateProcessor("rent–tov:rent–act–tov");
            akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
            akt.WriteXmlTree("PIB", llc.Fullname);
            akt.WriteXmlTree("rnokpp", llc.Rnokpp);
            akt.WriteXmlTree("address", llc.Address);
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("Director", llc.Director);
            akt.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
            akt.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(akt, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–акт");

            var dod = CreateProcessor("sublease–tov:supplement–tov");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", llc.Fullname);
            dod.WriteXmlTree("address", llc.Address);
            dod.WriteXmlTree("rnokpp", llc.Rnokpp);
            dod.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            dod.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(dod, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–додаток");
        }
    }

    public class Rent1SupplementaryAgreementTov : LeaseDocument
    {
        public Rent1SupplementaryAgreementTov(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");

            var p = CreateProcessor("rent–tov:rent–prup–tov");
            p.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.AddDays(–1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("PIB", llc.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", llc.Rnokpp);
            p.WriteXmlTree("address", llc.Address);
            p.WriteXmlTree("Director", llc.Director);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            p.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(p, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–припинення–ТОВ");
        }
```

```
}

public class Rent1ReturnActTov : LeaseDocument
{
    public Rent1ReturnActTov(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");

        var p = CreateProcessor("rent–tov:rent–return–act–tov");
        p.WriteXmlTree("fullDate", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
        p.WriteXmlTree("PIB", llc.Fullname);
        p.WriteXmlTree("rnokpp", llc.Rnokpp);
        p.WriteXmlTree("address", llc.Address);
        p.WriteXmlTree("Director", llc.Director);
        p.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
        p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
        p.WriteXmlTree("area", data.Group.Area.ToString());
        p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
        p.WriteXmlTree("address_p", data.Group.Address);
        p.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
        p.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
        Save(p, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–повернення");
    }
}

public class Rent1ContractWithActAndAnnexFop : LeaseDocument
{
    public Rent1ContractWithActAndAnnexFop(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");
        var rentInfo = data.RentInfo as RentType1Info ?? throw new InvalidOperationException("Expected
RentType1Info.");
        var naPid = DocHelper.BuildNaPidType1(rentInfo);

        var dog = CreateProcessor("rent–fop:rent–dog–fop");
        dog.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
        dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("PIB", fop.Fullname);
        dog.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
        dog.WriteXmlTree("address", fop.Address);
        dog.WriteXmlTree("edruofop_Data", fop.Edryofop);
        dog.WriteXmlTree("area", data.Group.Area.ToString());
        dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
        dog.WriteXmlTree("address_p", data.Group.Address);
        dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
        dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
        dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
        dog.WriteXmlTree("PIBS", fop.ResPerson);
        dog.WriteXmlTree("NaPid", naPid);

        dog.WriteXmlTree("subleaseDopNum", "____");
        dog.WriteXmlTree("subleaseDopDate", "____");
        dog.WriteXmlTree("subleaseDopName", "____");
```

```
            dog.WriteXmlTree("subleaseDopRnokpp", "____");
            Save(dog, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–договір");

            var akt = CreateProcessor("rent–fop:rent–act–fop");
            akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
            akt.WriteXmlTree("PIB", fop.Fullname);
            akt.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            akt.WriteXmlTree("address", fop.Address);
            akt.WriteXmlTree("edruofop_Data", fop.Edryofop);
            akt.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
            akt.WriteXmlTree("PIBS", fop.ResPerson);
            Save(akt, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–акт");

            var dod = CreateProcessor("sublease:supplement–fop");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", fop.Fullname);
            dod.WriteXmlTree("address", fop.Address);
            dod.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            dod.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            dod.WriteXmlTree("PIBS", fop.ResPerson);
            Save(dod, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–додаток");
        }
    }

    public class Rent1SupplementaryAgreementFop : LeaseDocument
    {
        public Rent1SupplementaryAgreementFop(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");

            var p = CreateProcessor("rent–fop:rent–prup–fop");
            p.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
            p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            p.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.AddDays(–1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("OriginalStrokDii",
data.Sublease.ContractEndDate.AddDays(1).ToString("dd/MM/yyyy"));
            p.WriteXmlTree("PIB", fop.Fullname);
            p.WriteXmlTree("address_p", data.Group.Address);
            p.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            p.WriteXmlTree("address", fop.Address);
            p.WriteXmlTree("edruofop_Data", fop.Edryofop);
            p.WriteXmlTree("area", data.Group.Area.ToString());
            p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            p.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            p.WriteXmlTree("PIBS", fop.ResPerson);
            Save(p, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–припинення–ФОП");
        }
    }
```

```
public class Rent1ReturnActFop : LeaseDocument
{
    public Rent1ReturnActFop(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");

        var p = CreateProcessor("rent–fop:rent–return–act–fop");
        p.WriteXmlTree("fullDate", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
        p.WriteXmlTree("PIB", fop.Fullname);
        p.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
        p.WriteXmlTree("address", fop.Address);
        p.WriteXmlTree("edruofop_Data", fop.Edryofop);
        p.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
        p.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
        p.WriteXmlTree("area", data.Group.Area.ToString());
        p.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
        p.WriteXmlTree("address_p", data.Group.Address);
        p.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
        p.WriteXmlTree("PIBS", fop.ResPerson);
        Save(p, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–повернення");
    }
}

public class Rent2ContractWithActAndAnnexTov : LeaseDocument
{
    public Rent2ContractWithActAndAnnexTov(DatabaseModel context) : base(context) { }

    public override async Task CreateAsync(Guid subleaseId)
    {
        var data = await GetDataAsync(subleaseId);
        var llc = data.CounterpartyLLC ?? throw new InvalidOperationException("Expected LLC counterparty.");
        var rentInfo = data.RentInfo as RentType2Info ?? throw new InvalidOperationException("Expected
RentType2Info.");
        var naPid = DocHelper.BuildNaPidType2(rentInfo);

        var dog = CreateProcessor("rent–tov:rent–dog–tov");
        dog.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
        dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("PIB", llc.Fullname);
        dog.WriteXmlTree("rnokpp", llc.Rnokpp);
        dog.WriteXmlTree("area", data.Group.Area.ToString());
        dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
        dog.WriteXmlTree("address_p", data.Group.Address);
        dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
        dog.WriteXmlTree("address", llc.Address);
        dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
        dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
        dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
        dog.WriteXmlTree("Director", llc.Director);
        dog.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
        dog.WriteXmlTree("NaPid", naPid);
        Save(dog, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–договір");

        var akt = CreateProcessor("rent–tov:rent–act–tov");
        akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
        akt.WriteXmlTree("PIB", llc.Fullname);
```

```
            akt.WriteXmlTree("rnokpp", llc.Rnokpp);
            akt.WriteXmlTree("address", llc.Address);
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("Director", llc.Director);
            akt.WriteXmlTree("DogovirRent", data.Sublease.ContractNumber);
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(llc.BankAccount ?? ""));
            akt.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(akt, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–акт");

            var dod = CreateProcessor("sublease–tov:supplement–tov");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", llc.Fullname);
            dod.WriteXmlTree("address", llc.Address);
            dod.WriteXmlTree("rnokpp", llc.Rnokpp);
            dod.WriteXmlTree("BanckAccount", llc.BankAccount ?? "");
            dod.WriteXmlTree("PIBSDirector", llc.ShortNameDirector);
            Save(dod, data, $"{data.Sublease.ContractNumber}–{llc.GroupName}–додаток");
        }
    }

    public class Rent2ContractWithActAndAnnexFop : LeaseDocument
    {
        public Rent2ContractWithActAndAnnexFop(DatabaseModel context) : base(context) { }

        public override async Task CreateAsync(Guid subleaseId)
        {
            var data = await GetDataAsync(subleaseId);
            var fop = data.CounterpartyFop ?? throw new InvalidOperationException("Expected FOP counterparty.");
            var rentInfo = data.RentInfo as RentType2Info ?? throw new InvalidOperationException("Expected
RentType2Info.");
            var naPid = DocHelper.BuildNaPidType2(rentInfo);

            var dog = CreateProcessor("rent–fop:rent–dog–fop");
            dog.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dog.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dog.WriteXmlTree("PIB", fop.Fullname);
            dog.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            dog.WriteXmlTree("address", fop.Address);
            dog.WriteXmlTree("edruofop_Data", fop.Edryofop);
            dog.WriteXmlTree("area", data.Group.Area.ToString());
            dog.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            dog.WriteXmlTree("address_p", data.Group.Address);
            dog.WriteXmlTree("StrokDii", data.Sublease.ContractEndDate.ToString("dd/MM/yyyy"));
            dog.WriteXmlTree("suma", DocHelper.Doublezero(data.Sublease.RentalFee));
            dog.WriteXmlTree("sum_text", NumToText.SumToText(data.Sublease.RentalFee));
            dog.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
            dog.WriteXmlTree("PIBS", fop.ResPerson);
            dog.WriteXmlTree("NaPid", naPid);
            dog.WriteXmlTree("subleaseDopNum", "____");
            dog.WriteXmlTree("subleaseDopDate", "____");
            dog.WriteXmlTree("subleaseDopName", "____");
            dog.WriteXmlTree("subleaseDopRnokpp", "____");
            Save(dog, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–договір");

            var akt = CreateProcessor("rent–fop:rent–act–fop");
```

```
            akt.WriteXmlTree("fullDate",
NumToText.NumberMonthToText(data.Sublease.AktDate.ToString("dd/MM/yyyy")));
            akt.WriteXmlTree("PIB", fop.Fullname);
            akt.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            akt.WriteXmlTree("address", fop.Address);
            akt.WriteXmlTree("edruofop_Data", fop.Edryofop);
            akt.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            akt.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            akt.WriteXmlTree("area", data.Group.Area.ToString());
            akt.WriteXmlTree("area_text", NumToText.NumberToText(data.Group.Area));
            akt.WriteXmlTree("address_p", data.Group.Address);
            akt.WriteXmlTree("BanckAccount", DeleteSpace.Deletespace(fop.BankAccount ?? ""));
            akt.WriteXmlTree("PIBS", fop.ResPerson);
            Save(akt, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–акт");

            var dod = CreateProcessor("sublease:supplement–fop");
            dod.WriteXmlTree("DogovirSuborendu", data.Sublease.ContractNumber);
            dod.WriteXmlTree("DateTime", data.Sublease.ContractSigningDate.ToString("dd/MM/yyyy"));
            dod.WriteXmlTree("address_p", data.Group.Address);
            dod.WriteXmlTree("PIB", fop.Fullname);
            dod.WriteXmlTree("address", fop.Address);
            dod.WriteXmlTree("rnokpp", fop.Rnokpp ?? "");
            dod.WriteXmlTree("BanckAccount", fop.BankAccount ?? "");
            dod.WriteXmlTree("PIBS", fop.ResPerson);
            Save(dod, data, $"{data.Sublease.ContractNumber}–{fop.GroupName}–додаток");
        }
    }

    public class Rent2SupplementaryAgreementTov : Rent1SupplementaryAgreementTov
    {
        public Rent2SupplementaryAgreementTov(DatabaseModel context) : base(context) { }
    }

    public class Rent2ReturnActTov : Rent1ReturnActTov
    {
        public Rent2ReturnActTov(DatabaseModel context) : base(context) { }
    }

    public class Rent2SupplementaryAgreementFop : Rent1SupplementaryAgreementFop
    {
        public Rent2SupplementaryAgreementFop(DatabaseModel context) : base(context) { }
    }

    public class Rent2ReturnActFop : Rent1ReturnActFop
    {
        public Rent2ReturnActFop(DatabaseModel context) : base(context) { }
    }

    public static class ContractDocumentFactory
    {

        public static ContractDocument Create(
            LeaseType leaseType,
            ContractorType contractorType,
            ContractDocumentType docType,
            DatabaseModel context)
            => (leaseType, contractorType, docType) switch
            {
                (LeaseType.Sublease, ContractorType.Tov, ContractDocumentType.ContractWithActAndAnnex)
                    => new SubleaseContractWithActAndAnnexTov(context),
```

```
        (LeaseType.Sublease, ContractorType.Tov, ContractDocumentType.SupplementaryAgreement)
            => new SubleaseSupplementaryAgreementTov(context),
        (LeaseType.Sublease, ContractorType.Tov, ContractDocumentType.ReturnAct)
            => new SubleaseReturnActTov(context),
        (LeaseType.Sublease, ContractorType.Tov, ContractDocumentType.ExtensionAgreement)
            => new SubleaseExtensionAgreementTov(context),

        (LeaseType.Sublease, ContractorType.Fop, ContractDocumentType.ContractWithActAndAnnex)
            => new SubleaseContractWithActAndAnnexFop(context),
        (LeaseType.Sublease, ContractorType.Fop, ContractDocumentType.SupplementaryAgreement)
            => new SubleaseSupplementaryAgreementFop(context),
        (LeaseType.Sublease, ContractorType.Fop, ContractDocumentType.ReturnAct)
            => new SubleaseReturnActFop(context),
        (LeaseType.Sublease, ContractorType.Fop, ContractDocumentType.ExtensionAgreement)
            => new SubleaseExtensionAgreementFop(context),

        (LeaseType.Rent1, ContractorType.Tov, ContractDocumentType.ContractWithActAndAnnex)
            => new Rent1ContractWithActAndAnnexTov(context),
        (LeaseType.Rent1, ContractorType.Tov, ContractDocumentType.SupplementaryAgreement)
            => new Rent1SupplementaryAgreementTov(context),
        (LeaseType.Rent1, ContractorType.Tov, ContractDocumentType.ReturnAct)
            => new Rent1ReturnActTov(context),

        (LeaseType.Rent1, ContractorType.Fop, ContractDocumentType.ContractWithActAndAnnex)
            => new Rent1ContractWithActAndAnnexFop(context),
        (LeaseType.Rent1, ContractorType.Fop, ContractDocumentType.SupplementaryAgreement)
            => new Rent1SupplementaryAgreementFop(context),
        (LeaseType.Rent1, ContractorType.Fop, ContractDocumentType.ReturnAct)
            => new Rent1ReturnActFop(context),

        (LeaseType.Rent2, ContractorType.Tov, ContractDocumentType.ContractWithActAndAnnex)
            => new Rent2ContractWithActAndAnnexTov(context),
        (LeaseType.Rent2, ContractorType.Tov, ContractDocumentType.SupplementaryAgreement)
            => new Rent2SupplementaryAgreementTov(context),
        (LeaseType.Rent2, ContractorType.Tov, ContractDocumentType.ReturnAct)
            => new Rent2ReturnActTov(context),

        (LeaseType.Rent2, ContractorType.Fop, ContractDocumentType.ContractWithActAndAnnex)
            => new Rent2ContractWithActAndAnnexFop(context),
        (LeaseType.Rent2, ContractorType.Fop, ContractDocumentType.SupplementaryAgreement)
            => new Rent2SupplementaryAgreementFop(context),
        (LeaseType.Rent2, ContractorType.Fop, ContractDocumentType.ReturnAct)
            => new Rent2ReturnActFop(context),

        _ => throw new ArgumentOutOfRangeException(
            $"Unknown combination: {leaseType}, {contractorType}, {docType}")
    };

public static ContractDocument Create(
    ContractData data,
    ContractDocumentType docType,
    DatabaseModel context)
{
    var leaseType = data.RentInfo?.RentType switch
    {
        GroupRentType.Sublease => LeaseType.Sublease,
        GroupRentType.Type1 => LeaseType.Rent1,
        GroupRentType.Type2 => LeaseType.Rent2,
        _ => throw new InvalidOperationException(
            $"Cannot map GroupRentType '{data.RentInfo?.RentType}' to LeaseType.")
```

```
            };

            var contractorType = data.IsFop ? ContractorType.Fop : ContractorType.Tov;

            return Create(leaseType, contractorType, docType, context);
        }
    }
}
```

Лістинг Б.2 — ContractBuilder.cs

```
using Microsoft.EntityFrameworkCore;
using SharedLibraries.Database;
using SharedLibraries.model;

namespace Sublease.Builder
{
    public record CounterpartySubleaseLLC(
        string Fullname,
        string Rnokpp,
        string BankAccount,
        string Address,
        string Director,
        string ShortNameDirector,
        string GroupName,
        CounterpartyStatus Status
    );

    public record CounterpartySubleaseFop(
        string Fullname,
        string Edryofop,
        string BankAccount,
        string Address,
        string ResPerson,
        string ShortName,
        CounterpartyStatus Status
    );

    public record GroupData(
        int NumberGroup,
        string Address,
        double Area,
        string Rnokpp
    );

    public record SubleaseData(
        string ContractNumber,
        DateOnly ContractSigningDate,
        DateOnly AktDate,
        DateOnly ContractEndDate,
        double RentalFee
    );

    public class ContractData
    {
        public CounterpartySubleaseLLC? CounterpartyLLC { get; init; }
        public CounterpartySubleaseFop? CounterpartyFop { get; init; }
        public GroupData Group { get; init; } = null!;
        public SubleaseData Sublease { get; init; } = null!;
```

```
    public bool IsFop => CounterpartyFop is not null;
}

public class ContractDataBuilder
{
    private readonly DatabaseModel _context;

    public ContractDataBuilder(DatabaseModel context)
    {
        _context = context;
    }

    public async Task<ContractData> BuildAsync(Guid subleaseId)
    {
        var sublease = await _context.Subleases
            .Include(s => s.Group)
                .ThenInclude(g => g.RentInfo)
            .Include(s => s.Group)
                .ThenInclude(g => g.Counterparty)
            .Where(s => s.Id == subleaseId)
            .FirstOrDefaultAsync()
            ?? throw new InvalidOperationException($"Sublease {subleaseId} not found.");

        var group = sublease.Group
            ?? throw new InvalidOperationException("Sublease has no Group.");

        var counterparty = group.Counterparty
            ?? throw new InvalidOperationException("Group has no Counterparty.");

        var rentInfo = (group.RentInfo as SubleaseRentInfo)
            ?? throw new InvalidOperationException("Group does not have SubleaseRentInfo.");

        CounterpartySubleaseLLC? llcData = null;
        CounterpartySubleaseFop? fopData = null;

        switch (counterparty)
        {
            case CounterpartyLLC llc:
                llcData = new CounterpartySubleaseLLC(
                    llc.Fullname,
                    llc.Rnokpp,
                    llc.BankAccount,
                    llc.Address,
                    llc.Director,
                    llc.ShortNameDirector,
                    llc.GroupName,
                    llc.Status
                );
                break;

            case CounterpartyFop fop:
                fopData = new CounterpartySubleaseFop(
                    fop.Fullname,
                    fop.Edryofop,
                    fop.BankAccount,
                    fop.Address,
                    fop.ResPerson,
                    fop.ShortName,
                    fop.Status
                );
```

```
                break;

            default:
                throw new InvalidOperationException(
                    $"Unknown counterparty type: {counterparty.GetType().Name}");
        }

        var groupData = new GroupData(
            group.NumberGroup,
            group.Address,
            group.Area,
            rentInfo.Rnokpp
        );

        var subleaseData = new SubleaseData(
            sublease.ContractNumber,
            sublease.ContractSigningDate,
            sublease.AktDate,
            sublease.ContractEndDate,
            sublease.RentalFee
        );

        return new ContractData
        {
            CounterpartyLLC = llcData,
            CounterpartyFop = fopData,
            Group = groupData,
            Sublease = subleaseData
        };
    }
  }
}
```

Лістинг Б.3 — DocumentGenerationController.cs

```
using Lease.Factory;
using Microsoft.AspNetCore.Mvc;
using SharedLibraries.model;
using SharedLibraries.Database;
using Microsoft.EntityFrameworkCore;

[Route("api/[controller]")]
[ApiController]
public class DocumentController : ControllerBase
{
    private readonly DatabaseModel _context;

    public DocumentController(DatabaseModel context)
    {
        _context = context;
    }

    [HttpPost("sublease/{subleaseId}/contract")]
    public async Task<IActionResult> SubleaseContract(Guid subleaseId)
        => await GenerateAsync(subleaseId, ContractDocumentType.ContractWithActAndAnnex);

    [HttpPost("sublease/{subleaseId}/supplementary–agreement")]
    public async Task<IActionResult> SubleaseSupplementaryAgreement(Guid subleaseId)
        => await GenerateAsync(subleaseId, ContractDocumentType.SupplementaryAgreement);
```

```
    [HttpPost("sublease/{subleaseId}/return–act")]
    public async Task<IActionResult> SubleaseReturnAct(Guid subleaseId)
        => await GenerateAsync(subleaseId, ContractDocumentType.ReturnAct);

    [HttpPost("sublease/{subleaseId}/extension")]
    public async Task<IActionResult> SubleaseExtension(Guid subleaseId)
        => await GenerateAsync(subleaseId, ContractDocumentType.ExtensionAgreement);

    private async Task<IActionResult> GenerateAsync(Guid subleaseId, ContractDocumentType docType)
    {
        try
        {
            var builder = new ContractDataBuilder(_context);
            var data = await builder.BuildAsync(subleaseId);

            var contractorType = data.IsFop ? ContractorType.Fop : ContractorType.Tov;
            var leaseType = data.RentInfo?.RentType switch
            {
                GroupRentType.Sublease => LeaseType.Sublease,
                GroupRentType.Type1 => LeaseType.Rent1,
                GroupRentType.Type2 => LeaseType.Rent2,
                _ => throw new InvalidOperationException("Unknown rent type.")
            };

            var document = ContractDocumentFactory.Create(leaseType, contractorType, docType, _context);
            await document.CreateAsync(subleaseId);

            return Ok(new { message = "Document created." });
        }
        catch (Exception ex)
        {
            return StatusCode(500, new { error = ex.Message });
        }
    }
}
```