Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

з теми: «Модель управління ОСББ на основі веб та мобільних технологій»

Виконав: здобувач вищої освіти групи KN1-M24

спеціальності 122 Комп’ютерні науки

Гавришко Володимир Євгенович

(прізвище, ім’я та по батькові здобувача вищої освіти)

Керівник: Іванюк Віталій Анатолійович, доктор технічних наук, доцент
(прізвище, ім’я та по батькові, науковий ступінь, вчене звання керівника)

Рецензент: Кушнір Оксана Климівна, кандидат економічних наук, доцент
(прізвище, ім’я та по батькові, науковий ступінь, вчене звання рецензента)

м. Кам’янець-Подільський – 2025 р.

ЗМІСТ

АНОТАЦІЯ

Гавришко В. Є. Модель управління ОСББ на основі веб та мобільних технологій. – Кваліфікаційна робота магістра за спеціальністю 122 «Комп'ютерні науки». – Кам'янець-Подільський національний університет імені Івана Огієнка, Кам'янець-Подільський, 2025.

Науковий керівник: Іванюк В. А., доктор технічних наук, доцент.

В роботі вдосконалено модель інформаційної підтримки діяльності ОСББ шляхом інтеграції адміністративних реєстрів та фінансових розрахунків у єдину розподілену систему на базі технології SPA (Blazor). Це забезпечує автоматизацію звітності, зниження ризику виникнення операторських помилок та оптимізацію навантаження на сервер завдяки перенесенню частини обчислень на клієнтську сторону.

Ключові слова: ОСББ, ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ ОБЛІКУ, .NET BLAZOR, ВЕБ-ТЕХНОЛОГІЇ.

Havryshko V. Ye. Management Model for Condominium Associations Based on Web and Mobile Technologies. – Master's Thesis, Specialty 122 "Computer Science". – Kamianets-Podilskyi National Ivan Ohiienko University, Kamianets-Podilskyi, 2025.

Scientific Supervisor: Ivaniuk V. A., Doctor of Technical Sciences, Associate Professor.

The thesis improves the model of information support for Condominium Associations (OSBB) activities by integrating administrative registers and financial calculations into a unified distributed system based on SPA (Blazor) technology. This ensures reporting automation, reduction of the risk of operator errors, and server load optimization by offloading part of the computations to the client side.

Keywords: CONDOMINIUM ASSOCIATION (OSBB), INFORMATION SYSTEM, ACCOUNTING AUTOMATION, .NET BLAZOR, WEB TECHNOLOGIES.

ВСТУП

Актуальність теми. Сучасний етап реформування житлово-комунального господарства України характеризується стрімким зростанням кількості Об’єднань співвласників багатоквартирних будинків (ОСББ). У цьому контексті ефективне управління спільним майном та фінансовими потоками стає ключовим фактором життєздатності таких об’єднань. Однак, існуючі підходи до адміністрування ОСББ часто базуються на застарілих методах: використанні паперового документообігу та розрізнених інструментів (електронних таблиць, месенджерів), що не інтегровані в єдину екосистему.

Така фрагментарність інформаційного простору призводить до низки проблем: низької оперативності обробки даних, ризиків втрати інформації, дублювання облікових операцій та непрозорості нарахувань для кінцевих користувачів. Це зумовлює значні часові та організаційні витрати керівництва ОСББ.

Розвиток веб-орієнтованих технологій та кросплатформних рішень відкриває нові можливості для автоматизації цих процесів. Створення єдиної цифрової платформи дозволяє уніфікувати інформаційні потоки — від реєстрів власників до білінгових операцій — та забезпечити віддалений доступ до даних у режимі реального часу [2].

Таким чином, розробка моделі управління ОСББ на основі сучасного стеку веб-технологій є актуальним науково-прикладним завданням, вирішення якого дозволить підвищити ефективність менеджменту, забезпечити прозорість фінансової звітності та покращити комунікацію між адміністрацією та мешканцями.

Метою роботи є обґрунтування та розробка моделі управління ОСББ на основі веб та мобільних технологій для автоматизації фінансового обліку та підвищення ефективності адміністративних процесів.

Для досягнення поставленої **мети** необхідно вирішити такі завдання:

1. Провести аналіз предметної області та існуючих інформаційних систем управління житловим фондом, виявити їх функціональні обмеження та недоліки.
2. Обґрунтувати вибір архітектурного підходу (SPA) та технологічного стеку для реалізації кросплатформної системи управління ОСББ.
3. Спроєктувати архітектуру системи та розробити інфологічну модель бази даних, що забезпечує цілісність обліку мешканців, житлових площ та нарахувань.
4. Реалізувати програмний засіб із використанням технології .NET Blazor, інтегрувавши модулі автоматизованого розрахунку внесків та імпорту банківських виписок.
5. Виконати тестування розробленої системи та оцінити ефективність її впровадження в діяльність ОСББ.

Об’єктом дослідження є процес інформаційного забезпечення та управління ресурсами в об'єднаннях співвласників багатоквартирних будинків.

Предметом дослідження є моделі, методи та інформаційні технології автоматизації обліку і взаємодії в ОСББ на основі веб-платформ.

Методи дослідження. Для розв’язання поставлених завдань у роботі використано комплекс методів: системного аналізу — для дослідження предметної області, виявлення недоліків існуючих підходів до управління ОСББ та формування функціональних вимог до системи; об’єктно-орієнтованого проєктування та моделювання (UML) — для розробки архітектури програмного забезпечення, побудови діаграм класів та визначення

взаємодії компонентів веб-додатка; теорії реляційних баз даних — для проєктування інфологічної схеми даних, нормалізації відношень та забезпечення цілісності інформації про мешканців і платежі; методів програмної інженерії — для реалізації бізнес-логіки системи, побудови клієнт-серверної взаємодії та організації інтерфейсу користувача; експериментального тестування — для перевірки коректності роботи алгоритмів нарахувань, валідації вхідних даних та оцінки працездатності розробленого продукту.

Наукова новизна одержаних результатів полягає у вдосконаленні моделі інформаційної взаємодії в екосистемі ОСББ шляхом створення єдиного простору інтеграції адміністративних реєстрів та фінансових потоків, що дозволяє формалізувати зв’язки між об’єктами нерухомості й суб’єктами оплати та забезпечити цілісність даних. Водночас набуло подальшого розвитку застосування архітектурного підходу SPA (Single Page Application) для систем комунального обліку, який, на відміну від традиційних рішень, передбачає перенесення обчислювального навантаження з формування звітів на клієнтську сторону, що забезпечує масштабованість системи.

Практичне значення одержаних результатів полягає у програмній реалізації веб-орієнтованої інформаційної системи на платформі .NET Blazor, яка забезпечує автоматизацію повного циклу управління ОСББ: від ведення реєстрів власників та житлового фонду до автоматичного нарахування внесків та імпорту банківських виписок, що дозволяє суттєво зменшити часові витрати на адміністрування та підвищити прозорість фінансової діяльності об’єднання.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ УПРАВЛІННЯ ОСББ

1.1 Особливості організаційно-фінансового управління ОСББ та задачі обліку

Управління об'єднанням співвласників багатоквартирного будинку (ОСББ) — це комплексний процес, який охоплює організаційні, технічні та фінансові складові. Ключовим завданням є належне утримання спільного майна, прозорий контроль за рухом коштів, налагодження ефективної комунікації між співвласниками з метою спільного ухвалення важливих рішень [4].

Однією з центральних складових фінансово-господарської діяльності в ОСББ є постійний моніторинг фінансових потоків і обґрунтоване планування бюджету. Об'єднання має чітко фіксувати всі операції — від надходжень за комунальні послуги до витрат на поточне утримання будинку. Принципово важливою є прозорість у звітності: мешканці повинні мати можливість контролювати цільове використання коштів [3, 5].

Фінансовий та операційний облік в ОСББ є фундаментальним елементом усього процесу управління будинком — від нього залежить не лише фінансова дисципліна, а й довіра мешканців. Саме він гарантує прозорість, точність і прогнозованість у розв’язанні фінансових питань.

Першочерговим завданням є підтримання актуального реєстру співвласників та квартир, у якому мають бути зазначені контактні дані мешканців, площа житла або частка у спільній власності. Саме ця база є інформаційною основою для всіх фінансових обчислень, зокрема при визначенні обсягів щомісячних внесків.

Не менш важливою складовою є облік тарифів. ОСББ має не лише актуалізувати чинні ставки за послуги, а й системно відстежувати історію їх змін: коли було впроваджено новий тариф, коли відбувалося його коригування чи скасування. Такий підхід забезпечує точність нарахувань і дозволяє уникати розбіжностей при формуванні фінансових звітів.

Нарахування щомісячних платежів здійснюються автоматизовано. Кожен рахунок формується на основі чинних тарифів та площі квартири, що забезпечує об’єктивність розрахунків для співвласників.

Важливою є реєстрація оплат. Усі надходження мають відображатися в системі, що дозволяє формувати достовірну картину фінансового стану кожного особового рахунку. Це, своєю чергою, забезпечує можливість об’єктивно обчислювати сальдо на початок і кінець звітного періоду.

І нарешті, ведення сальдо — це постійний контроль за фінансовим балансом по кожній квартирі. У результаті такого підходу ОСББ отримує можливість оперативно виявляти заборгованості, відстежувати динаміку надходжень коштів і в цілому оцінювати ефективність фінансового управління. Це не тільки сприяє своєчасному реагуванню на фінансові ризики, а й створює умови для формування стабільної стратегії розвитку будинку.

Отже, інтеграція цифрових інструментів в облікові процеси мінімізує кількість помилок, автоматизує фінансову звітність і спрощує доступ до інформації. Впровадження веб- і мобільних рішень дає змогу оптимізувати основні операції, знизити навантаження на адміністрацію та забезпечити мешканцям оперативний доступ до необхідних даних.

1.2 Проблеми існуючих підходів до обліку та управління ОСББ

Аналіз практичної діяльності ОСББ в Україні свідчить про домінування застарілих підходів до інформаційного забезпечення, що базуються на

паперовому документообігу або використанні неспеціалізованого програмного забезпечення (електронних таблиць). Відсутність централізованих цифрових інструментів призводить до децентралізації ключових бізнес-процесів — обліку, фінансових розрахунків та звітності, що суттєво знижує ефективність управління [1].

Критичним недоліком існуючих підходів є низький рівень автоматизації білінгових процесів. Ручний розрахунок нарахувань (з урахуванням змінних тарифів, площі квартир та пільгових категорій) характеризується високою трудомісткістю та значною ймовірністю виникнення операторських помилок. Несвоєчасна актуалізація вхідних даних ускладнює звірку сальдо та провокує виникнення розбіжностей у розрахунках з мешканцями.

Суттєвою проблемою є інформаційна асиметрія у комунікації між адміністрацією та співвласниками. Відсутність механізмів віддаленого доступу до особових рахунків (Personal Cabinet) позбавляє мешканців можливості оперативно контролювати нарахування, історію платежів та поточний баланс, що негативно впливає на рівень платіжної дисципліни.

Крім того, відсутність автоматизованого моніторингу дебіторської заборгованості унеможливлює оперативне реагування на накопичення боргів. Ручна обробка даних призводить до часових затримок (latency) в отриманні актуальної інформації керівництвом ОСББ.

На організаційному рівні проблемою є фрагментарність даних: облікова інформація зберігається у різних форматах та на різних носіях без єдиної політики резервного копіювання. Це порушує принципи цілісності даних та ускладнює забезпечення їхньої конфіденційності.

Також варто відзначити складність адміністрування тарифної сітки без використання спеціалізованих СУБД. Ручне відстеження періодів дії тарифів (history tracking) є ресурсоємним процесом, помилки в якому призводять до необхідності масових перерахунків за минулі періоди.

Отже, існуючі методи управління ОСББ характеризуються недостатньою гнучкістю та ресурсоємністю через відсутність єдиної інформаційної екосистеми. Вирішення цих проблем потребує впровадження інтегрованого веб-орієнтованого рішення, що об’єднує функції обліку, білінгу та комунікації.

1.3 Огляд існуючих платформ для автоматизації діяльності ОСББ

Аналіз ринку програмного забезпечення у сфері житлово-комунального господарства свідчить про наявність широкого спектра інформаційно-аналітичних систем (ІАС), орієнтованих на автоматизацію управління житловим фондом. Більшість існуючих рішень реалізовані як SaaS-платформи (Software as a Service), що пропонують уніфікований функціонал: від ведення реєстру співвласників до інтеграції з банківськими установами.

Для визначення вимог до проєктованої системи доцільно провести порівняльний аналіз найбільш поширених в Україні платформ.

«ОСББ-Онлайн» — комплексна CRM-система, орієнтована на великі управлінські компанії. Її архітектура передбачає централізоване зберігання даних та розширений функціонал для роботи персоналу (бухгалтерів, диспетчерів). Система дозволяє вести детальний облік співвласників, генерувати квитанції та формувати аналітичну звітність. Однак, орієнтація на масштабні об’єкти робить інтерфейс перевантаженим для пересічного адміністратора малого ОСББ.

«Мій Дім Online» — система класу ERP для житлових комплексів, що акцентує увагу на диспетчеризації заявок та технічному обслуговуванні. Особливістю є наявність розвиненої екосистеми мобільних додатків. Проте, висока вартість підписки та надлишкова функціональність для будинків без

штатного технічного персоналу є стримуючими факторами для її впровадження в невеликих об’єднаннях.

«Моє ОСББ» — платформа, що фокусується на соціальній складовій: комунікації, опитуваннях та електронному голосуванні. Фінансовий модуль тут реалізований як допоміжний, що обмежує можливості автоматичного імпорту банківських реєстрів та гнучкого налаштування тарифної сітки.

Для систематизації переваг та недоліків розглянутих аналогів проведено порівняльний аналіз, результати якого наведено в таблиці 1.1.

Таблиця 1.1. Порівняльна характеристика існуючих систем управління ОСББ

Критерій порівняння	**ОСББ-Онлайн**	**Мій Дім Online**	**Моє ОСББ**	**Проєктована система**
Архітектура	SaaS (Хмарна)	SaaS (Хмарна)	SaaS (Хмарна)	SPA (Клієнт-серверна)
Фінансовий облік	Повний цикл	Повний цикл	Базовий	Спеціалізований
Імпорт виписок	Є (платний модуль)	Є	Обмежений	Автоматизований
Вартість впровадження	Висока (підписка)	Висока	Середня	Низька (Open Source)
Гнучкість налаштувань	Низька (коробкове рішення)	Низька	Середня	Висока (модульна)
Мобільна адаптивність	Мобільний додаток	Мобільний додаток	Веб-версія	PWA / Adaptive Web

Проведений аналіз (табл. 1.1) дозволяє зробити висновок, що існуючі комерційні рішення часто мають надлишковий функціонал, є економічно

обтяжливими для малих ОСББ (щомісячна абонплата) та мають закритий вихідний код, що унеможливлює їх доопрацювання під специфічні потреби конкретного будинку.

Відтак, актуальним є розробка власної інформаційної системи на базі сучасних веб-технологій (.NET Blazor), яка поєднає необхідний мінімум облікових функцій із гнучкістю налаштувань та відсутністю регулярних ліцензійних платежів.

1.4 Обґрунтування актуальності розробки нової моделі управління

Сучасні ОСББ постають перед необхідністю підвищення ефективності адміністрування в умовах зростання обсягів даних та застарілості існуючих інструментальних засобів. Використання неузгоджених електронних таблиць та непрозорість механізмів нарахування платежів підтверджують необхідність переходу до якісно нових систем управління.

Розробка веб-орієнтованої інформаційної системи, що базується на сучасному технологічному стеку, є технологічно обґрунтованою відповіддю на системні виклики: низьку ефективність облікових процесів, відсутність верифікованого фінансового контролю та складність взаємодії з боржниками.

Доцільність розробки підтверджується такими чинниками:

1. *Автоматизація обчислювальних процесів***.** Виключення ручного нарахування платежів дозволяє мінімізувати вплив «людського фактора», забезпечує високу точність розрахунків згідно з площею приміщень та актуальними тарифами, що суттєво скорочує час на формування звітності.
2. *Забезпечення інформаційної прозорості***.** Надання мешканцям авторизованого онлайн-доступу до фінансових показників формує високий рівень довіри до адміністрації. Користувачі отримують

можливість у режимі реального часу контролювати нарахування та історію транзакцій без необхідності прямого звернення до правління.

3. *Централізація та цілісність даних***.** Об’єднання реєстрів мешканців, тарифних сіток та станів особових рахунків у межах єдиної бази даних усуває дублювання інформації та ризики її втрати, забезпечуючи швидкий доступ до верифікованих відомостей.
4. *Моніторинг дебіторської заборгованості***.** Система дозволяє здійснювати оперативний аналіз стану оплат, ідентифікувати заборгованості в реальному часі та автоматизувати комунікацію з боржниками, що сприяє зміцненню фінансової дисципліни в об'єднанні.
5. *Кросплатформність та доступність***.** Використання веб-технологій забезпечує незалежність системи від апаратного та програмного середовища користувача. Робота з платформою не потребує встановлення додаткового ПЗ і є доступною з будь-яких пристроїв через стандартні веб-браузери.

Отже, розробка нової моделі управління спрямована на трансформацію адміністративних процесів ОСББ у структуроване цифрове середовище. Це дозволяє замінити трудомісткі ручні операції на автоматизовані алгоритми, забезпечуючи відкритість та ефективність взаємодії всіх учасників процесу.

1.5 Аналіз вимог до функціональності для ключових користувачів

У процесі проєктування системи управління ОСББ ключовим етапом є визначення функціональних вимог відповідно до потреб двох основних категорій користувачів: адміністрації (правління) та мешканців (співвласників). Розмежування прав доступу та сценаріїв взаємодії дозволяє оптимізувати архітектуру системи.

Для *адміністрації ОСББ* пріоритетним є повний адміністративний доступ до управління даними (CRUD-операції). Функціональні вимоги включають: управління реєстрами (можливість ведення бази даних власників та об’єктів нерухомості (редагування площі, кількості зареєстрованих осіб, контактних даних)); адміністрування тарифів (гнучке налаштування тарифної сітки з підтримкою версійності та прив’язкою до часових інтервалів для забезпечення ретроспективної точності розрахунків); автоматизація білінгу (алгоритмічне нарахування внесків на основі заданих параметрів (площа, тарифи, пільги), що мінімізує ймовірність обчислювальних помилок); облік транзакцій (реєстрація надходжень коштів, можливість коригування помилкових записів та підтримка часткових оплат); аналітична звітність (інструменти для оперативного моніторингу дебіторської заборгованості та формування фінансових звітів (сальдо по квартирах) у режимі реального часу).

Для *мешканців (співвласників)* акцент зміщено на прозорість інформації та механізми самообслуговування через «Особистий кабінет». Основні вимоги включають:

- моніторинг стану рахунку - візуалізація поточного балансу, деталізація нарахованих сум та верифікація проведених платежів;
- доступ до історії - можливість перегляду хронології нарахувань та оплат за довільний період, що забезпечує верифікованість даних та знижує навантаження на адміністрацію щодо надання довідок;
- інформаційна підтримка - ознайомлення з актуальними тарифами та структурою платежів, що сприяє прозорості комунікації;
- актуалізація даних - можливість дистанційного оновлення контактної інформації для забезпечення оперативного зв’язку з правлінням.

Таким чином, при розробці системи критично важливим є дотримання балансу між функціональною повнотою для адміністратора та зручністю інтерфейсу (UI/UX) для мешканця. Реалізація цих вимог дозволяє перетворити

систему з простого інструменту обліку на інтегровану платформу ефективної взаємодії всіх учасників управління ОСББ.

1.6 Визначення основних тенденцій та технологічних можливостей

Аналіз розвитку цифрових рішень свідчить, що впровадження спеціалізованого програмного забезпечення для управління житловим фондом стає галузевим стандартом. Сучасні тенденції формуються на перетині запитів користувачів на прозорість даних та нових можливостей веб-технологій.

Ключовими напрямами розвитку систем управління ОСББ є:

- Веб-орієнтована архітектура. Використання хмарних технологій дозволяє перенести бізнес-логіку у веб-середовище. Це забезпечує доступ до системи через браузер без необхідності встановлення локального ПЗ, що гарантує незалежність від операційної системи та апаратної конфігурації пристрою (ноутбук, планшет, смартфон).
- Мобільна адаптивність та концепція Mobile First. З огляду на те, що більшість користувачів взаємодіють з інформаційними системами через мобільні пристрої, пріоритетом стає розробка адаптивних інтерфейсів. Це передбачає не лише зміну візуального відображення під розмір екрана, а й оптимізацію швидкодії в умовах мобільного інтернету.
- Обробка даних у реальному часі (Real-time processing). Актуальним трендом є мінімізація затримок при оновленні фінансової інформації. Можливість миттєвого відображення транзакцій, актуалізації сальдо та автоматичного формування звітів без ручного втручання значно підвищує оперативність управлінських рішень.
- Масштабованість та модульність. Сучасна архітектура інформаційних систем повинна підтримувати горизонтальне масштабування — стабільну роботу як з одиничними об'єктами (малі ОСББ), так і з

великими житловими комплексами. Модульний підхід дозволяє нарощувати функціональність (наприклад, додавати модулі голосування або диспетчеризації) без докорінної зміни ядра системи.

Таким чином, сучасна система управління ОСББ має відповідати високим вимогам до інформаційної цілісності, швидкодії та UX-дизайну. Пріоритетом є створення безбар'єрного цифрового середовища, яке автоматизує рутинні процеси та забезпечує високий рівень достовірності даних для всіх учасників взаємодії.

Висновки до розділу 1

У першому розділі на основі проведеного аналізу предметної області та існуючих підходів до автоматизації діяльності ОСББ отримано ряд результатів.

Виконаний аналіз теоретичних засад та огляд методів управління ОСББ засвідчив доцільність розробки комплексної інформаційної моделі для впорядкування облікових процедур. Деталізовано процеси організаційно-фінансового адміністрування, зокрема потреби щодо ведення реєстрів співвласників, обліку об'єктів нерухомості та автоматизації фінансових нарахувань.

Встановлено, що головними недоліками існуючих підходів є низький рівень автоматизації та інформаційної прозорості. Це обумовлює високу ймовірність виникнення помилок при ручній обробці даних, призводить до надлишкових часових витрат адміністрації та знижує рівень довіри співвласників до результатів розрахунків.

На основі аналізу аналогів окреслено функціональні вимоги до системи для двох категорій користувачів: адміністративного персоналу (інструменти білінгу, моніторинг балансів, формування звітності) та співвласників

(дистанційний доступ до історії нарахувань, верифікація оплат та актуалізація персональних даних).

Обґрунтовано, що оптимальним рішенням для усунення виявлених проблем є розробка веб-орієнтованої системи на базі архітектури SPA.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ МОДЕЛІ ТА АРХІТЕКТУРИ СИСТЕМИ

2.1 Функціональна та нефункціональна специфікація системи

У процесі проєктування інформаційної системи критично важливо чітко розмежувати функціональні та нефункціональні вимоги, оскільки вони визначають архітектурні особливості та логіку роботи програмного продукту.

Функціональні вимоги визначають перелік конкретних операцій, які система повинна виконувати для задоволення потреб користувачів. Для системи управління ОСББ ключовими функціональними вимогами є: управління суб'єктами та об'єктами (ведення реєстрів співвласників та житлового фонду, фіксація історії володіння приміщеннями); білінгові операції (автоматизоване нарахування платежів згідно з обраними алгоритмами, обробка вхідних транзакцій та коригування балансів); тарифне планування (ведення довідників тарифів із підтримкою архівування та версійності (зберігання історії змін)); інформаційна підтримка (формування персоніфікованої звітності (сальдо, історія нарахувань та оплат) для кожного об'єкта обліку).

Нефункціональні вимоги описують якісні характеристики системи та обмеження, в межах яких вона повинна функціонувати. До них віднесено: надійність та цілісність даних (забезпечення атомарності операцій за допомогою механізмів транзакцій, що унеможливлює втрату або пошкодження даних при редагуванні чи видаленні записів); продуктивність та масштабованість (стабільна робота системи при обробці великих масивів інформації та підтримка одночасного доступу декількох сесій користувачів); безпека (реалізація надійної системи автентифікації та авторизації для чіткого розмежування прав доступу за ролями (адміністратор, бухгалтер,

мешканець)); ергономічність (Usability) (забезпечення динамічного інтерфейсу, що підтримує фільтрацію, сортування та пошук даних без повного перезавантаження сторінок сторінок (відповідно до парадигми SPA)).

Модульна структура системи дозволяє ізолювати окремі бізнес-процеси. Зокрема, блок обліку власників фокусується на зберіганні персональних і контактних даних мешканців, а також на логуванні змін у складі співвласників. Модуль об’єктів нерухомості (квартир) інтегрує інформацію про площу приміщень, їх прив’язку до власників та актуальні тарифні плани.

Фінансовий блок реалізує повний цикл реєстрації операцій: від ініціалізації початкових залишків до фіксації поточних платежів. На основі цих даних система динамічно обчислює підсумковий баланс для кожного об’єкта нерухомості.

Варто відзначити, що реалізація клієнтської частини дозволяє здійснювати імпорт даних та маніпуляції з таблицями в режимі реального часу. Керуючі елементи інтерфейсу спроєктовані таким чином, щоб забезпечити послідовність дій користувача та мінімізувати ймовірність помилкового введення даних.

Загалом, визначені характеристики системи відповідають її головній меті — забезпеченню надійного та функціонально повного фінансово-операційного обліку в ОСББ.

2.2 Обґрунтування вибору технологічного стеку

При виборі технологічного стеку для реалізації системи пріоритет надавався рішенням, що поєднують високу продуктивність, масштабованість та ефективність розробки. Основним інструментарієм було обрано платформу .NET 8, що зумовлено її стабільністю, довготривалою підтримкою (LTS) та наявністю розвиненої екосистеми [6]. Платформа дозволяє реалізувати чіткий

поділ бізнес-логіки, забезпечує високий рівень безпеки та надає потужні засоби об’єктно-реляційного відображення через Entity Framework Core [8].

Ключову роль у виборі відіграла технологія Blazor, яка дозволяє створювати інтерактивні веб-інтерфейси за парадигмою SPA (Single Page Application) без необхідності використання сторонніх плагінів. Blazor забезпечує високу швидкість виконання коду та дозволяє використовувати мову C# як на стороні сервера, так і на стороні клієнта. Це забезпечує типізацію даних на всіх рівнях системи, скорочує кількість помилок та спрощує архітектурну підтримку продукту [7].

Для побудови користувацького інтерфейсу обрано бібліотеку компонентів MudBlazor, яка базується на принципах Material Design. Вона забезпечує адаптивність інтерфейсу для різних типів пристроїв, містить готові модулі для роботи з таблицями, діалоговими вікнами та формами, що прискорює розробку та гарантує інтуїтивність UI [9].

При проєктуванні системи було визначено вимоги щодо кросплатформності та відсутності необхідності попереднього встановлення клієнтського ПЗ. Обрана веб-орієнтована архітектура дозволяє запускати систему через стандартні браузери на будь-яких пристроях (персональні комп’ютери, планшети, смартфони). Це гарантує користувачам доступ до актуальних даних у режимі реального часу, усуваючи проблему несумісності версій.

Фундаментальним архітектурним рішенням стало використання багаторівневої архітектури (N-tier architecture). Вона передбачає чітке розмежування логіки обробки даних, рівнів доступу до бази даних та інтерфейсу користувача. Такий підхід забезпечує високу гнучкість: оновлення функціоналу одного модуля (наприклад, модуля нарахувань) не впливає на стабільність роботи інших компонентів системи.

Окрему увагу приділено централізованому зберіганню даних. Консолідація інформації про власників, об’єкти нерухомості та фінансові транзакції в єдиній базі даних гарантує цілісність, несуперечливість та швидкий доступ до верифікованих відомостей.

Кодова база побудована за модульним принципом, що спрощує технічну підтримку та масштабування системи. Реалізовано механізм розмежування доступу на основі ролей (Role-Based Access Control), що дозволяє кожній категорії користувачів (адміністратор, бухгалтер) працювати виключно в межах своїх функціональних повноважень, забезпечуючи конфіденційність та безпеку даних.

2.3 Архітектура програмної системи

Архітектура системи базується на принципі багаторівневої організації (N-tier architecture), що передбачає чіткий поділ компонентів за їхньою відповідальністю. Такий підхід забезпечує високу модульність, зручність супроводу та можливість незалежного вдосконалення окремих шарів без порушення стабільності всієї системи.

В архітектурі виділено три основні рівні:

1. Рівень збереження даних (Data Layer). Це фундамент системи, який відповідає за структурування та персистентність інформації. На цьому рівні визначено моделі (сутності) предметної області: власники (Person), об'єкти нерухомості (Residence), тарифи (Tariff), фінансові операції (Transaction) та сальдо (Balance). Використання об'єктно-реляційного відображення забезпечує цілісність даних та верифікацію зв’язків між об’єктами на рівні схеми бази даних.
2. Рівень бізнес-логіки (Business Logic Layer). Даний рівень виконує роль посередника та реалізує основні правила обробки даних. Тут

зосереджено алгоритми розрахунку щомісячних нарахувань, механізми валідації вхідних значень та логіку оновлення фінансових результатів. Наприклад, під час реєстрації платежу система автоматично ініціює перерахунок поточного балансу відповідного об'єкта нерухомості.

3. Рівень представлення (Presentation Layer). Це верхній рівень, реалізований за допомогою технології Blazor. Він забезпечує інтерактивну взаємодію користувача з системою. Рівень містить набір компонентів для візуалізації даних у вигляді динамічних таблиць, форм введення та панелей управління. Завдяки використанню асинхронних запитів, оновлення інформації в інтерфейсі відбувається реактивно, без повного перезавантаження сторінок сторінок, що критично важливо для оперативності фінансового моніторингу.

Взаємодія між компонентами системи організована через чітко визначені інтерфейси. Рівень представлення звертається до сервісів логічного шару, які, у свою чергу, здійснюють маніпуляції з даними. Така структура мінімізує надмірність коду (DRY) та дозволяє легко розширювати функціональність шляхом додавання нових модулів.

На рис. 2.1 зображено компонентну діаграму, що ілюструє взаємозв'язки між основними елементами архітектури.

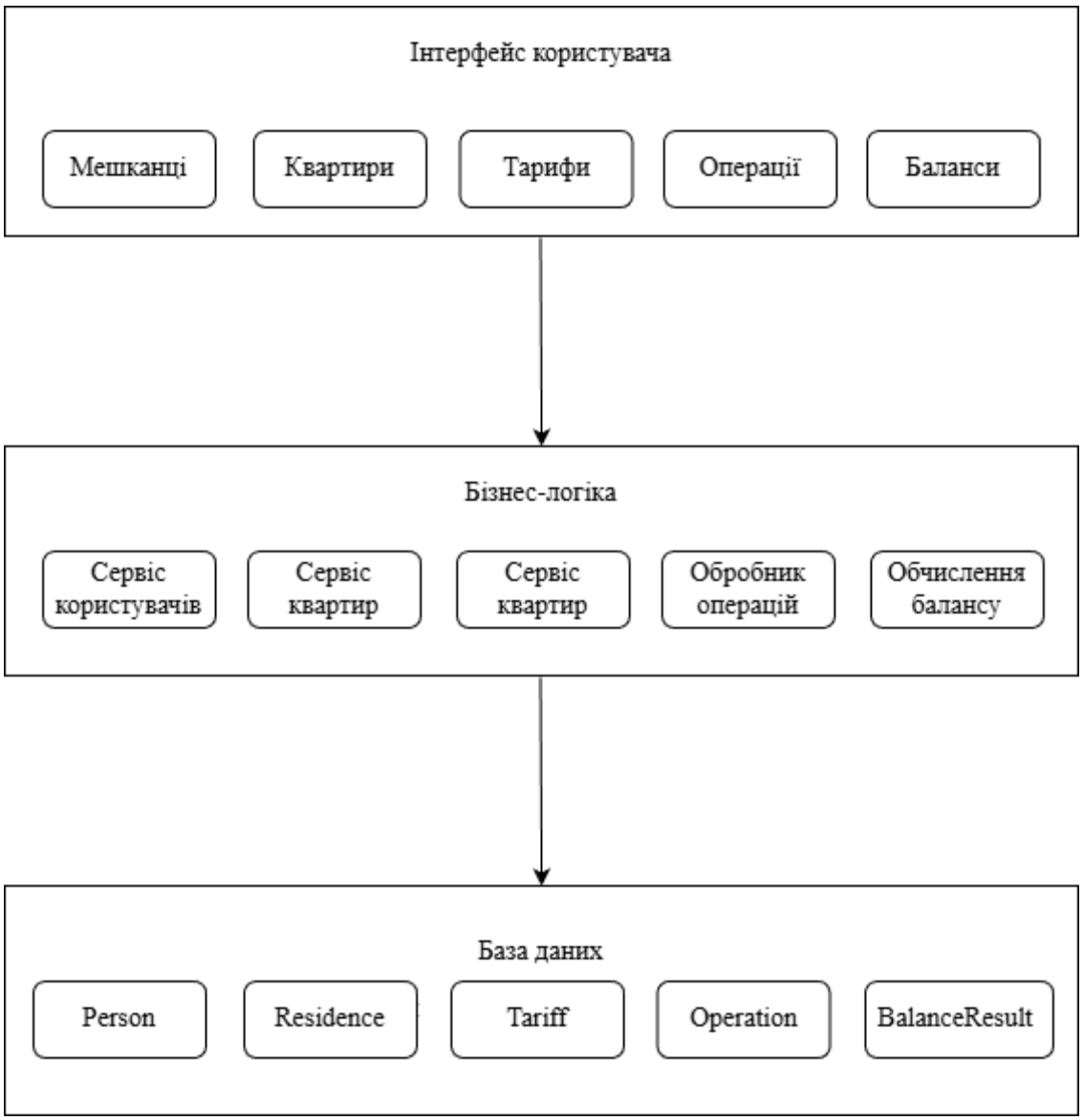

Рис. 2.1. Компонентна діаграма архітектури системи

Між структурними рівнями існують односпрямовані залежності (від інтерфейсу до логіки та від логіки до даних). Це забезпечує стабільну структуру залежностей, спрощує модульне тестування та підвищує адаптивність системи до майбутніх змін у вимогах. Завдяки такій побудові кожен елемент має чітко визначене призначення, що полегшує як розробку, так і подальшу підтримку та масштабування системи.

2.4 Проєктування бази даних

Для забезпечення надійного зберігання та ефективного опрацювання інформації в системі спроєктовано реляційну базу даних. При розробці структури особливу увагу приділено забезпеченню цілісності даних, мінімізації надмірності (шляхом нормалізації до третьої нормальної форми) та встановленню логічних зв’язків між сутностями предметної області.

На рис. 2.2 наведено інфологічну модель (ER-діаграму) бази даних, яка відображає основні сутності та типи зв’язків між ними.

Кожна таблиця в базі даних відповідає конкретній сутності, а зв’язки між ними реалізовані через механізм первинних (Primary Keys) та зовнішніх (Foreign Keys) ключів.

Центральною сутністю моделі є «Квартира» (Residence), оскільки вона виступає ключовим об’єктом обліку, з яким інтегруються інші дані. Кожен запис містить унікальний ідентифікатор, площу приміщення, а також зовнішні ключі, що вказують на поточного власника та діючий тарифний план.

Сутність «Власник» (Person) містить персональні та контактні дані співвласників. Використання унікальних ідентифікаторів дозволяє коректно пов'язувати фізичних осіб із об'єктами нерухомості, що дає змогу відстежувати історію володіння приміщеннями та забезпечувати персоналізацію фінансової звітності.

Дані про вартість послуг зосереджені в сутності «Тариф» (Tariff). Структура передбачає зберігання назви, вартості та часових інтервалів дії тарифу. Це дозволяє системі автоматично обирати актуальні розцінки залежно від дати проведення розрахунків та зберігати ретроспективну інформацію для коректного перерахунку за минулі періоди.

Ключовим елементом для фінансового моніторингу є сутність «Операція» (Operation). Вона фіксує всі типи фінансових операцій:

нарахування внесків, реєстрацію оплат, коригування залишків тощо. Між сутностями «Квартира» та «Операція» встановлено зв’язок типу «один-до-багатьох» (1:N), що дозволяє зберігати повну хронологію фінансових подій по кожному об’єкту.

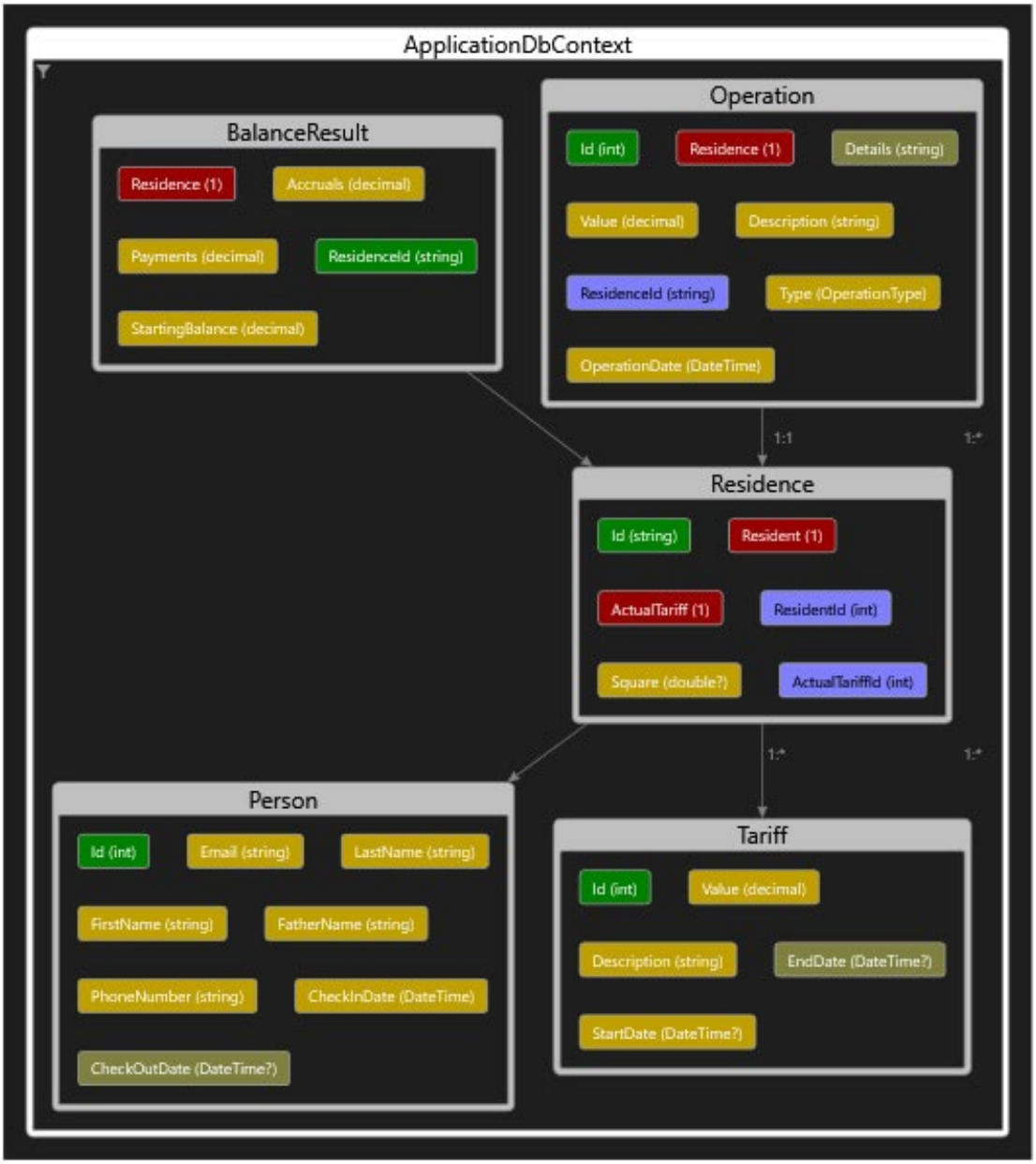

Рис. 2.2. ER-модель бази даних системи управління ОСББ

Для оптимізації швидкодії системи впроваджено сутність «Баланс» (Balance), яка зберігає агреговані результати обчислень за звітні періоди:

вхідне сальдо, загальну суму нарахувань та оплат, а також кінцевий залишок. Зв’язок із сутністю «Квартира» типу «один-до-одного» (1:1) забезпечує миттєвий доступ до фінансового стану об’єкта без необхідності виконання складних обчислювальних запитів у режимі реального часу.

Спроєктована модель забезпечує високу гнучкість та масштабованість. Структура таблиць та індексів оптимізована для виконання поширених запитів: формування виписок по особових рахунках, аналізу динаміки платежів та автоматизованої підготовки фінансової звітності ОСББ.

2.5 Моделювання бізнес-процесів

Для забезпечення коректної роботи програмного забезпечення було проведено моделювання головних бізнес-процесів, що відображають логіку нарахувань, обробки фінансових операцій і формування балансу. Це моделювання дозволяє формалізувати взаємодію користувача та системи, визначити зв’язки між компонентами та запобігти логічним помилкам під час розробки.

Одним із ключових процесів є щомісячне нарахування внесків. Алгоритм ініціюється користувачем, який обирає відповідний звітний період. Система виконує перевірку наявності нарахувань у базі даних за вказаний час. Якщо записи вже існують, система генерує попередження для уникнення дублювання даних. У разі відсутності нарахувань, система автоматично створює нові операції для кожної квартири, розраховуючи суму внеску на основі площі приміщення та діючого тарифу. Усі створені транзакції зберігаються в базі даних і стають основою для фінансових розрахунків.

Після завершення нарахувань система оновлює стан рахунків, враховуючи вхідне сальдо та попередні платежі. Результати відображаються в

інтерфейсі користувача у вигляді оновлених таблиць операцій та остаточних балансів. Логічну послідовність цих дій представлено на рис. 2.3.

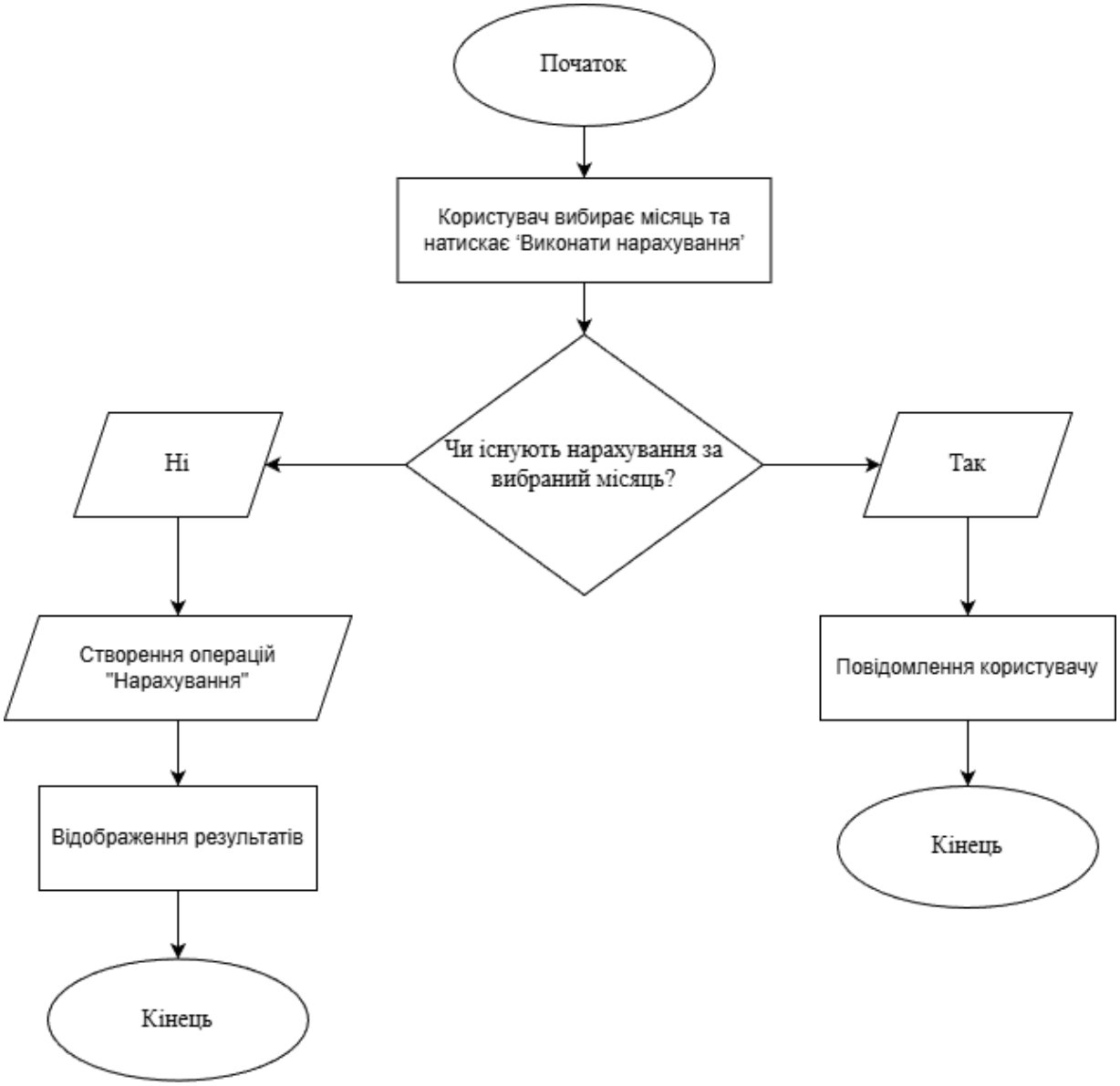

Рис. 2.3. Діаграма бізнес-процесу нарахувань і розрахунків

На даній схемі продемонстровано черговість дій: від запиту користувача до перевірки існуючих записів, створення нових фінансових операцій та фінальної візуалізації результатів.

Не менш важливим для стабільної роботи моделі є процес реєстрації та заселення мешканця. Він забезпечує формування коректних інформаційних

зв’язків, від яких залежать подальші облікові операції (нарахування, розрахунок балансів, зміна тарифів).

Процес починається з відкриття форми додавання резидента. На етапі введення даних система здійснює валідацію — перевірку правильності заповнення обов'язкових полів та відповідність форматів даних. Після успішної перевірки користувач обирає квартиру. Система виконує логічну перевірку стану об'єкта нерухомості (чи є квартира вільною для заселення). Якщо об'єкт доступний, створюється запис у таблиці мешканців та оновлюється зв'язок ResidentID у таблиці квартир, що гарантує структурну цілісність моделі даних.

На заключній стадії система оновлює стан пов'язаних таблиць в інтерфейсі та генерує сповіщення про успішне завершення операції. Візуалізацію цього процесу наведено на рис. 2.4.

На цій схемі продемонстровано послідовність операцій, які здійснює адміністратор для фіксації даних у системі. Використання таких формалізованих бізнес-процесів дозволяє забезпечити надійність розробки, виключити дублювання інформації та створити ефективний інструмент для управління ОСББ.

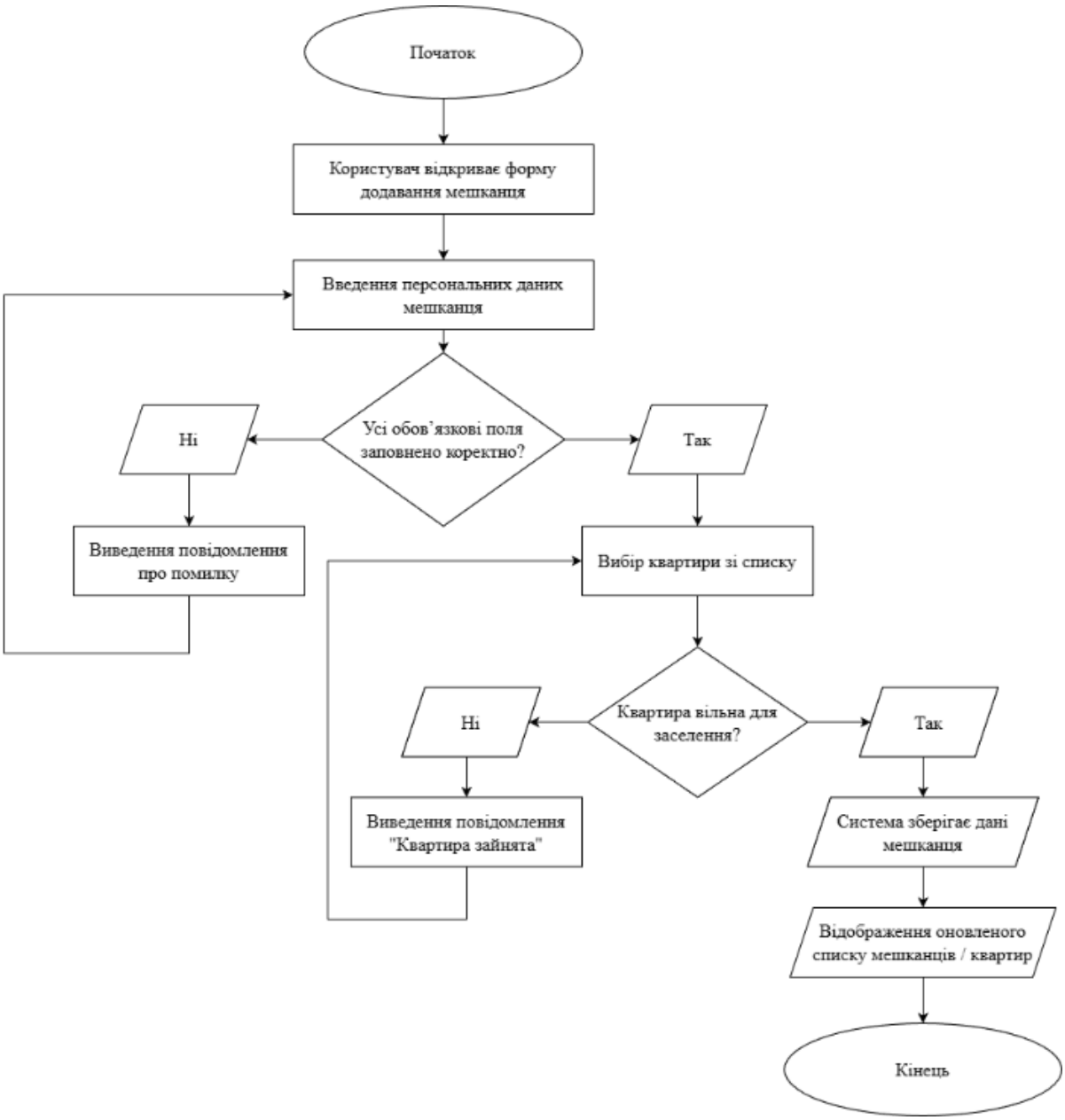

Рис. 2.4. Діаграма бізнес-процесу заселення мешканця в квартиру

2.6 Розмежування прав доступу та безпека даних

Для забезпечення надійності функціонування системи та захисту конфіденційної інформації реалізовано механізми аутентифікації, авторизації та контролю доступу.

Аутентифікація користувачів здійснюється за допомогою адреси електронної пошти та пароля. Паролі зберігаються у базі даних у вигляді криптографічних хеш-значень, що унеможливлює доступ до персональних даних у відкритому вигляді навіть у разі несанкціонованого доступу до сховища даних.

В системі впроваджено рольову модель доступу (RBAC), що визначає набір дозволених операцій для кожної категорії користувачів. Основною роллю є «Адміністратор», який володіє повним набором прав для виконання CRUD-операцій (створення, читання, оновлення, видалення) над усіма сутностями системи: тарифами, квартирами, мешканцями та фінансовими транзакціями.

Адміністратор відповідає за ведення тарифної сітки, актуалізацію реєстрів мешканців, реєстрацію оплат та проведення щомісячних нарахувань. Для ролі «Мешканець» передбачено обмежений рівень доступу (Read-only), що дозволяє переглядати результати розрахунків та стан власного особового рахунку без можливості модифікації чи видалення даних. Таке розмежування мінімізує ризики випадкового викривлення критично важливої інформації.

Захист даних реалізовано також на рівні обробки запитів. Використання контексту даних та асинхронних підходів дозволяє запобігти колізіям та несанкціонованому доступу до ресурсів під час конкурентної роботи користувачів. Усі операції, пов’язані з фінансовим обліком, здійснюються через шар сервісів бізнес-логіки, який виконує попередню перевірку коректності даних перед їх записом у базу. Це забезпечує цілісність інформації та запобігає появі помилкових записів.

Обмін даними між клієнтською та серверною частинами системи здійснюється через захищені протоколи, що унеможливлює перехоплення пакетів інформації. Крім того, на рівні інтерфейсу впроваджено механізми

валідації вхідних даних, які блокують спроби збереження некоректної або потенційно небезпечної інформації.

Висновки до розділу 2

У другому розділі спроєктовано архітектуру та логічну структуру інформаційної системи, що спрямована на автоматизацію фінансового обліку в об'єднанні співвласників багатоквартирного будинку.

Визначено функціональні та нефункціональні вимоги, на основі яких обґрунтовано вибір технологічного стеку для створення сучасного веб-рішення на базі платформи .NET 8 та технології Blazor.

Розроблено багаторівневу архітектуру системи, що включає рівень представлення (UI), шар бізнес-логіки та рівень доступу до даних. Така структура забезпечує модульність, полегшує технічну підтримку та масштабування функцій без порушення стабільності системи.

Створено ER-модель бази даних, яка описує сутності співвласників, об’єктів нерухомості, тарифів та фінансових балансів. Зв'язки між таблицями спроєктовано згідно з принципами реляційної моделі, що гарантує цілісність та несуперечливість даних [8].

Проведено моделювання ключових бізнес-процесів, зокрема механізму щомісячних нарахувань. Побудовані діаграми відображають алгоритмічну послідовність дій користувача та системи, що дозволило оптимізувати логіку розрахунку балансів.

Для забезпечення надійності системи розроблено механізми автентифікації та авторизації. Впроваджена рольова модель доступу (RBAC) гарантує захист від несанкціонованих змін та забезпечує безпечну роботу як для адміністраторів, так і для користувачів.

Отже, проведене проєктування формує цілісне підґрунтя для програмної реалізації системи, забезпечуючи логічну зв'язність її компонентів та відповідність поставленим вимогам.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ПРОТОТИПУ СИСТЕМИ

3.1 Розробка користувацького інтерфейсу

Користувацький інтерфейс системи реалізований у вигляді вебзастосунку з акцентом на ергономічність, зручність та інтуїтивність. Головною метою проектування UI було забезпечення оперативного доступу до інформації про об’єкти нерухомості, співвласників, тарифи та фінансові транзакції, а також надання можливості виконувати облікові операції без використання сторонніх інструментів.

Інтерфейс побудовано на основі сучасних вебтехнологій із застосуванням модульного підходу [9]. Кожна сторінка застосунку спроектована як окремий функціональний блок, що відповідає певній предметній області:

- Тарифи — візуалізація реєстру діючих цін із функціоналом додавання, редагування та архівації даних;
- Квартири — моніторинг характеристик об’єктів нерухомості (площа, номер), закріплених мешканців та актуальних тарифних планів;
- Власники — реєстр співвласників із контактними даними, хронологією заселення та інструментами модифікації записів;
- Нарахування — модуль для проведення регулярних та разових нарахувань, реєстрації оплат і обробки фінансових операцій;
- Баланси — консолідована панель фінансового стану кожного об’єкта обліку.

Візуальне оформлення інтерфейсу базується на узгодженій системі стилів, що покращує сприйняття великих масивів даних. Для роботи з

таблицями впроваджено механізми сортування, повнотекстового пошуку та фільтрації, що суттєво спрощує навігацію.

На рис. 3.1 представлено інтерфейс панелі нарахувань при використанні на стаціонарних комп'ютерах.

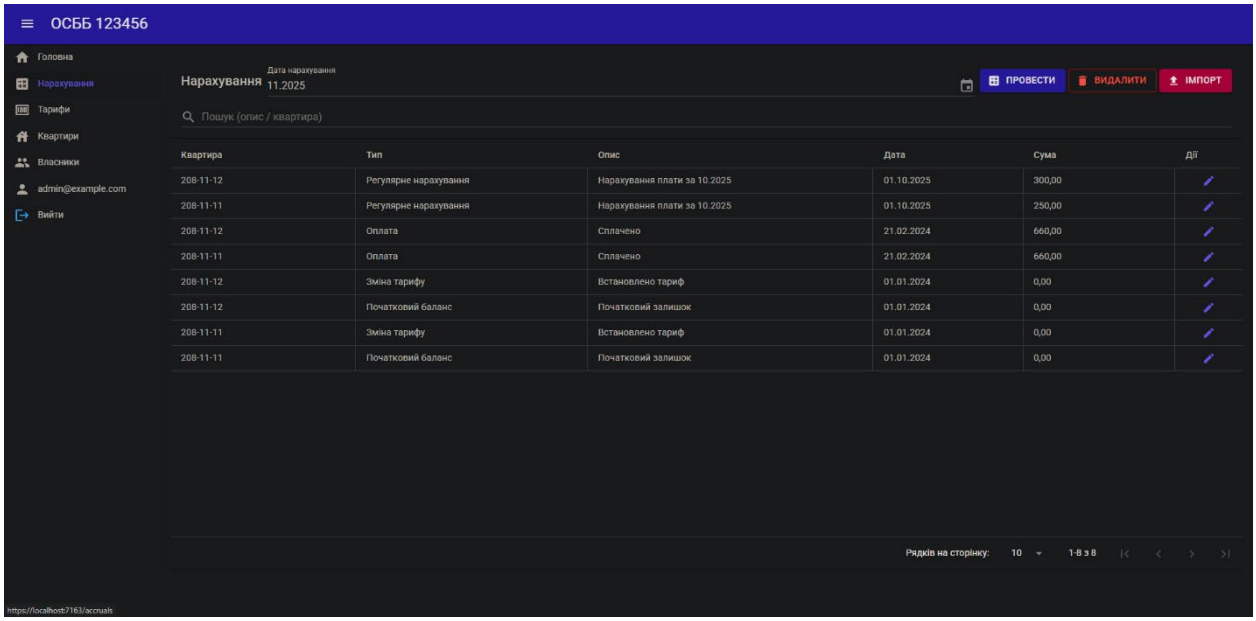

Рис. 3.1. Панель нарахувань (десктопна версія)

Особливу увагу приділено адаптивності інтерфейсу. Застосунок коректно відображається як на десктопних моніторах, так і на мобільних пристроях. Реалізована адаптивність забезпечує зручність експлуатації системи незалежно від типу апаратного забезпечення. Для взаємодії з даними використовуються модальні вікна, що дозволяють додавати, редагувати та імпортувати інформацію без повного перезавантаження сторінки. Це відповідає парадигмі SPA та робить роботу з системою більш динамічною.

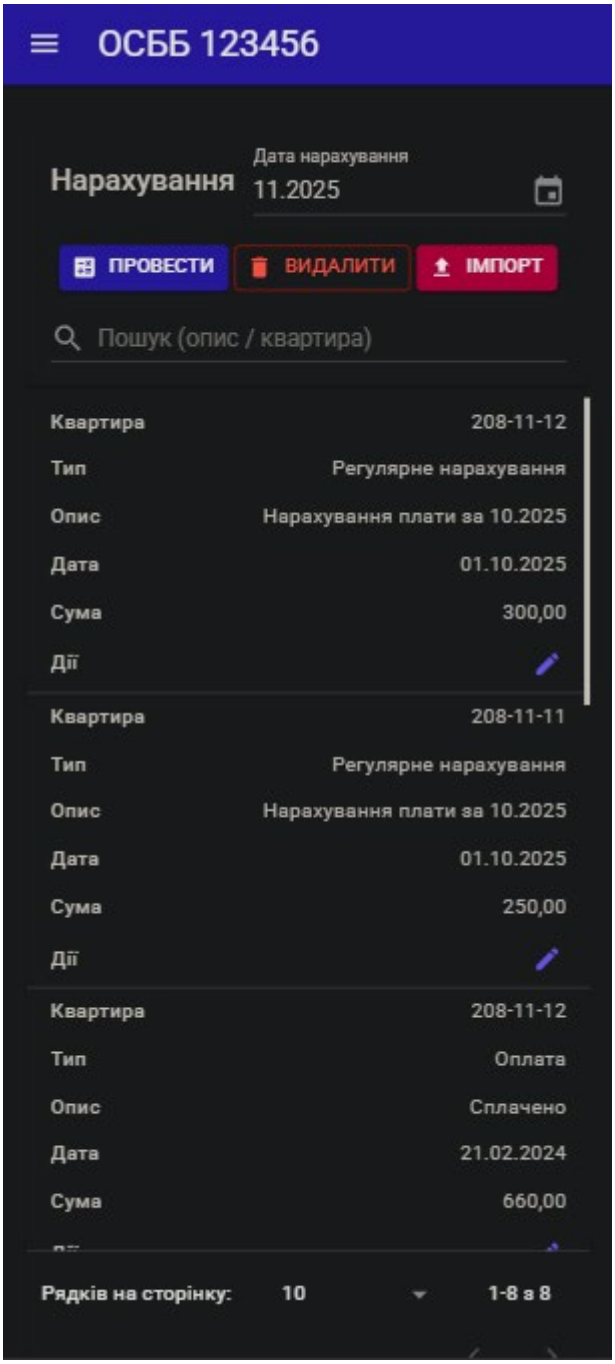

Рис. 3.2. Панель нарахувань на мобільному пристрої

Отже, розроблений інтерфейс поєднує в собі лаконічну навігацію, сучасний дизайн та високу функціональність. Це дозволяє ефективно виконувати основні облікові завдання та забезпечує високий рівень UX (User Experience) для користувачів системи.

3.2 Опис ключових модулів системи

Система має модульну архітектуру, де кожен компонент відповідає за виконання певної групи функцій. Такий підхід забезпечує високий рівень декомпозиції, спрощує технічну підтримку програмного продукту та створює умови для його подальшого масштабування без радикальної зміни існуючої структури.

Основним компонентом системи є модуль адміністрування співвласників, який забезпечує обробку персональних даних власників квартир. Для кожного мешканця ведеться електронна картка, що містить ідентифікаційні дані, контактну інформацію та хронологію періодів проживання. Це дозволяє підтримувати актуальність реєстру житлового фонду та формувати достовірну історію володіння об’єктами.

Модуль об’єктів нерухомості (квартир) оперує даними про характеристики приміщень: унікальні ідентифікатори, загальну площу, відомості про власників та прив’язані тарифні плани. Квартира виступає центральним вузлом системи, оскільки з цим об'єктом пов’язані всі подальші фінансові транзакції.

Модуль тарифікації відповідає за управління ставками внесків та їхню зміну в часі. Для кожного тарифу фіксуються розмір внеску та період дії. Це дозволяє застосовувати коректні цінові показники під час автоматичних розрахунків та забезпечує версійність тарифної сітки для ретроспективного аналізу.

Ключову роль відіграє модуль обліку фінансових транзакцій, який забезпечує моніторинг руху грошових коштів. У системі реєструються нарахування, оплати та коригування. Кожна транзакція є атомарною, має унікальний опис, суму та зв'язок із конкретним особовим рахунком. Така

деталізація дозволяє відтворити фінансовий стан об’єкта за будь-який обраний період.

Алгоритмічна частина реалізована у модулі розрахунків (білінгу), який виконує автоматичне формування щомісячних внесків. Логіка його роботи базується на обчисленні сум для кожного об'єкта згідно з діючою ставкою тарифу та площею приміщення. Система веде перевірку наявності нарахувань за обраний період для запобігання дублюванню записів. Після реєстрації операцій проводиться перерахунок балансів, що включає обчислення вхідного сальдо, агрегацію нових нарахувань, віднімання проведених оплат та визначення кінцевого залишку.

Модуль консолідованих балансів надає користувачу агреговану інформацію про стан взаєморозрахунків по всіх об’єктах нерухомості. Він дозволяє відстежувати динаміку платежів, формувати аналітичну звітність та здійснювати моніторинг дебіторської заборгованості.

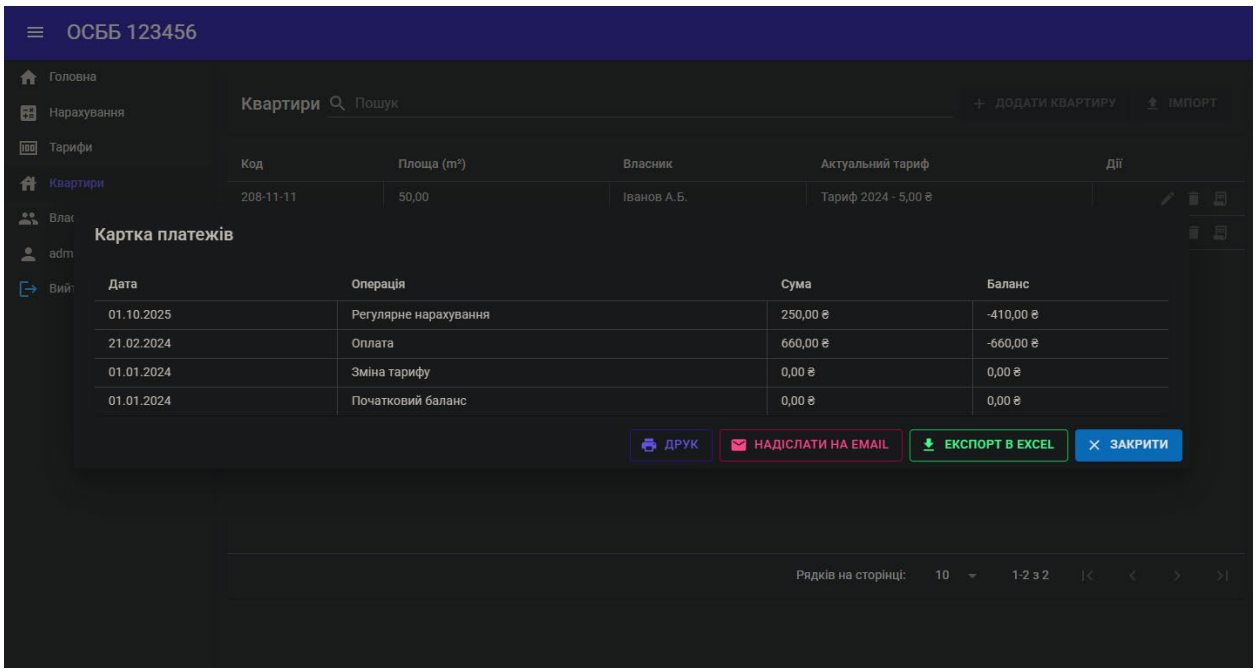

Рис. 3.3. Схема взаємодії ключових модулів системи

Взаємодія між компонентами здійснюється через внутрішні сервіси бізнес-логіки, які підтримують цілісність даних. Така структура дозволяє уникати надмірності інформації та підвищує стабільність системи. Модульний підхід гарантує прозорість обчислювальних процесів та легкість впровадження нових функціональних блоків у майбутньому.

3.3 Реалізація модулів фінансових розрахунків

Однією з ключових складових системи є підсистема обробки фінансових даних, яка автоматизує процеси білінгу, нарахування внесків та формування звітних балансів. Її реалізація базується на сервісно-орієнтованому підході, що дозволяє делегувати обчислювальну логіку окремим модулям, забезпечуючи цілісність інформації та високу точність розрахунків.

Основні операції з фінансовими даними згруповані у спеціалізованому сервісі бізнес-логіки. Через цей сервіс реалізовано життєвий цикл нарахувань: ініціалізація періоду, верифікація відсутності дублікатів, агрегація транзакцій та актуалізація балансів.

Процес формування нарахувань ініціюється користувачем шляхом вибору звітного місяця. Система виконує валідацію на наявність існуючих записів за обраний період. У разі успішної перевірки здійснюється автоматичне створення транзакцій типу «Нарахування». Для кожного об’єкта нерухомості (квартири) ідентифікується актуальна ставка тарифу та площа, на основі яких обчислюється сума внеску. Усі операції записуються в базу даних із фіксацією часових міток, описів та прив’язкою до особового рахунку. Механізм попередньої перевірки унеможливлює дублювання фінансових даних.

Після реєстрації нових транзакцій виконується процедура перерахунку підсумкових балансів. Система агрегує всі операції за обраний інтервал часу,

визначаючи вхідне сальдо, загальну суму нарахувань та фактичних оплат. Розрахунок кінцевого фінансового стану здійснюється за формулою:

$$B_{end} = B_{start} + \sum T_{accr} - \sum T_{paym},$$

де B_{end} — кінцеве сальдо (баланс); B_{start} — вхідне сальдо на початок періоду; T_{accr} — сума нарахувань за період; T_{paym} — сума проведених оплат.

Механізм анулювання (видалення) нарахувань також інтегрований у сервісний шар. Перед виконанням операції система ідентифікує останній звітний період, запитує підтвердження користувача, після чого виконує видалення відповідних записів та рекурсивне оновлення балансів. Це дозволяє підтримувати базу даних в актуальному стані та виправляти операторські помилки без порушення цілісності обліку.

Усі обчислювальні процеси реалізовані як асинхронні операції, що підвищує чуйність інтерфейсу та забезпечує стабільну роботу системи при обробці великих масивів даних. Кожен етап супроводжується системою сповіщень про статус виконання (успіх або виникнення виключних ситуацій).

Результати обчислень динамічно відображаються у відповідних модулях інтерфейсу. На сторінці транзакцій користувач може переглядати деталізацію операцій за обраний період, а в розділі консолідованих балансів — загальний фінансовий стан кожного об’єкта нерухомості.

3.4 Тестування системи та перевірка працездатності

Після завершення етапу програмної реалізації було проведено комплексне тестування розробленого програмного забезпечення з метою перевірки коректності роботи основних функцій, відповідності результатів обчислень очікуваним значенням та оцінки стабільності системи. Основна увага приділялася верифікації модулів нарахувань, взаємодії з базою даних та розмежуванню прав доступу.

Тестування проходило у кілька етапів:

1. Функціональне тестування сутностей. На першому етапі перевірялася коректність операцій при створенні об’єктів «Мешканець», «Квартира» та «Тариф». Для кожної нової сутності контролювалася цілісність записів у базі даних, правильність відображення даних у таблицях інтерфейсу та збереження зв'язків між таблицями (наприклад, зв’язок мешканця з конкретною квартирою).
2. Тестування модуля нарахувань. Проведено перевірку алгоритму білінгу: чи правильно система ідентифікує діючі тарифи та чи відповідає підрахована сума внеску встановленій формулі. Після ініціалізації нарахувань за вибраний період було підтверджено автоматичне оновлення підсумкового сальдо по кожній квартирі та коректність реєстрації відповідних фінансових транзакцій.
3. Тестування обробки платежів. У систему вводилися дані про оплати з різними сумами та датами. Верифікація підтвердила, що при реєстрації платежу дебіторська заборгованість зменшується автоматично, а кінцевий баланс перераховується без потреби в додаткових ручних операціях.
4. Тестування захисних механізмів та обмежень. Перевірено ідемпотентність операцій — відсутність можливості створення повторних нарахувань за один і той самий звітний період. Система успішно ідентифікує спроби дублювання даних, блокує їх та виводить відповідне попередження. Також підтверджено роботу механізму підтвердження критичних дій (наприклад, видалення останнього періоду нарахувань).
5. Перевірка рольової моделі доступу (RBAC). Підтверджено, що користувач із обмеженими правами має доступ лише до перегляду результатів розрахунків свого особового рахунку, тоді як адміністратор

володіє повним набором прав для управління інформацією. Це гарантує дотримання принципів безпеки, закладених на етапі проєктування.

На кінцевому етапі проведено оцінку продуктивності та стабільності. При обробці значних масивів записів система зберегла стабільний час відгуку. Використання асинхронних методів у .NET Blazor дозволило уникнути затримок інтерфейсу під час виконання фонових обчислень. Також було перевірено механізми валідації, які успішно відфільтровують некоректні або неповні вхідні дані.

Результати тестування показали, що всі ключові модулі працюють згідно з технічними вимогами, система є стабільною та працездатною. Це підтверджує готовність розробленого прототипу до впровадження у дослідну експлуатацію.

Висновки до розділу 3

У третьому розділі представлено результати програмної реалізації та апробації розробленої інформаційної системи. За результатами виконання практичної частини роботи отримано ряд висновків.

Здійснено програмну реалізацію прототипу системи на базі технологічного стеку .NET Blazor. Обрана архітектура дозволила інтегрувати інтерфейс користувача, сервіси бізнес-логіки та базу даних у межах єдиної розподіленої структури, що забезпечує високу стабільність та узгодженість роботи компонентів.

Розроблено та впроваджено функціональні модулі, що забезпечують ведення реєстрів співвласників, об’єктів нерухомості та тарифних планів. Реалізовано алгоритми автоматичного нарахування щомісячних внесків (білінгу) та формування консолідованих фінансових балансів. Усі

обчислювальні процеси реалізовані як асинхронні операції, що гарантує високу швидкість відгуку інтерфейсу при опрацюванні великих масивів даних.

Проведено комплексне тестування системи, результати якого підтвердили коректність роботи реалізованих алгоритмів. Верифіковано точність фінансових розрахунків, надійність збереження зв’язків у базі даних та ефективність механізмів розмежування прав доступу.

За результатами проведених перевірок доведено працездатність та готовність програмного забезпечення до дослідної експлуатації у сфері управління житловим фондом (ОСББ).

ВИСНОВКИ

У кваліфікаційній роботі розв’язана прикладна науково-технічна задача підвищення ефективності управління та фінансового обліку в об’єднаннях співвласників багатоквартирних будинків шляхом розроблення спеціалізованої інформаційної системи. Реалізоване програмне забезпечення забезпечує комплексну автоматизацію процесів білінгу, реєстрації транзакцій, обліку співвласників та об’єктів нерухомості.

У результаті виконання роботи отримано такі основні результати:

1. Проаналізовано сучасні підходи до організації діяльності ОСББ, досліджено недоліки існуючих методів фінансового обліку та обґрунтовано необхідність їхньої цифрової трансформації на основі сучасних вебтехнологій.
2. На основі системного аналізу предметної області проведено декомпозицію вимог до функціональності системи, визначено структуру інформаційних потоків та етапи обробки даних, що дозволило сформувати теоретичне підґрунтя для проєктування.
3. Розроблено багаторівневу архітектуру (N-tier architecture) програмної системи, що включає ізольовані модулі управління мешканцями, житловим фондом, тарифами та фінансовими операціями, забезпечуючи високу модульність та масштабованість рішення.
4. Спроектовано реляційну базу даних із цілісною моделлю сутностей. Реалізовано алгоритмічне забезпечення автоматичних нарахувань та модуль консолідації балансів, що гарантує високу точність фінансових операцій. Розроблено ергономічний кросплатформний інтерфейс за парадигмою SPA, що забезпечує динамічну взаємодію користувача з даними.

5. Проведено верифікацію працездатності програмного забезпечення, яка підтвердила точність фінансових обчислень, стабільність системи при навантаженні та коректність роботи захисних механізмів розмежування доступу.

Застосування розробленої системи дозволяє автоматизувати рутинні облікові процеси, мінімізувати вплив «людського фактора», підвищити прозорість управління та суттєво скоротити часові витрати на адміністрування ОСББ. Завдяки підтримці версійності тарифів та автоматичному формуванню звітності, система є ефективним інструментом для керуючих компаній та житлових комплексів.

Перспективами подальшого розвитку є інтеграція з банківськими API для автоматизації обробки виписок у реальному часі, розширення аналітичних інструментів прогнозування витрат та впровадження модулів електронного документообігу (ЕЦП).

Отже, поставлену мету дослідження досягнуто повністю, а всі визначені завдання — виконано. Розроблений прототип інформаційної системи відповідає сучасним вимогам програмної інженерії та рекомендований до практичного застосування.

СПИСОК ВИКОРСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Терещенко Л. О., Сніжко О. С. Інформаційні технології в управлінні // Інвестиції: практика та досвід. — 2011. — №12. — С. 28–31.
2. Тетерін О. А., Гунька Б. В., Процак К. В. Цифровізація в житлово-комунальному господарстві територіальних громад: переваги та перешкоди // Економіка та суспільство. — 2024. — Вип. 70. — С. 804–810.
3. Від першої цеглинки до сталого розвитку ОСББ. Посібник із створення та розвитку ОСББ / Проєкт ЄС/ПРООН «Місцевий розвиток, орієнтований на громаду», фаза III. — Київ, 2017. — 55 с.
4. Створення та діяльність об’єднання співвласників багатоквартирного будинку: практичний посібник / Інститут місцевого розвитку. — Київ : ДП «Видавничий дім “Козаки”», 2007. — 288 с.
5. Створення та початок діяльності ОСББ. Посібник / Проєкт ЄС/ПРООН «HOUSES», Фонд енергоефективності. — Київ, 2020. — 77 с.
6. Microsoft Corporation. .NET Documentation : офіційна технічна документація. — Режим доступу: https://learn.microsoft.com/dotnet (дата звернення: 02.07.2025).
7. Microsoft Corporation. ASP.NET Core Blazor : офіційний довідник з веб-фреймворку. — Режим доступу: https://learn.microsoft.com/aspnet/core/blazor/ (дата звернення: 02.07.2025).
8. Microsoft Corporation. Entity Framework Core Documentation : офіційна документація ORM фреймворку. — Режим доступу: https://learn.microsoft.com/ef/core/ (дата звернення: 02.07.2025).

9. MudBlazor Team. MudBlazor: Blazor Component Library : офіційна документація. — Режим доступу: https://mudblazor.com/ (дата звернення: 02.07.2025).

ДОДАТКИ

Додаток А. Лістинг коду компонента ResidencePaymentCard

```
@using Microsoft.EntityFrameworkCore
@using Microsoft.Extensions.Localization
@using MudBlazor
@using ResidentialTracker.Data
@using System.Text
@using ResidentialTracker.Services

@inject IDbContextFactory<ApplicationDbContext> DbFactory
@inject IJSRuntime JSRuntime
@inject ISnackbar Snackbar
@inject IStringLocalizer<ResidentialTracker.Resources.Pages.Residence>
Localizer
@inject IStringLocalizer<ResidentialTracker.Resources.Pages.Accruals>
AccrualsLocalizer
@inject Microsoft.AspNetCore.Identity.UI.Services.IEmailSender EmailSender
@inject IReportingService ReportingGenerator

<MudDialog>
    <DialogContent>
        @if (_loading)
        {
            <MudProgressLinear Color="Color.Primary" Indeterminate="true" />
        }
        else
        {
            <MudTable Items="_operations" Dense="true" Hover="true"
Bordered="true" Striped="true">
                <HeaderContent>
                    <MudTh>@Localizer["Apartments_Data"]</MudTh>
                    <MudTh>@Localizer["Apartments_Operation"]</MudTh>
                    <MudTh>@Localizer["Apartments_Amount"]</MudTh>
                    <MudTh>@Localizer["Apartments_Balance"]</MudTh>
                </HeaderContent>
```

```
                <RowTemplate>
                    <MudTd
DataLabel="@Localizer["Apartments_Data"]">@context.OperationDate.ToString("dd
.MM.yyyy")</MudTd>
                    <MudTd
DataLabel="@Localizer["Apartments_Operation"]">@context.Type</MudTd>
                    <MudTd
DataLabel="@Localizer["Apartments_Amount"]">@context.Value.ToString("C")</Mud
Td>
                    <MudTd
DataLabel="@Localizer["Apartments_Balance"]">@context.Balance.ToString("C")</
MudTd>
                </RowTemplate>
            </MudTable>
        }
    </DialogContent>
    <DialogActions>
        <MudButton Variant="Variant.Outlined" Color="Color.Primary"
OnClick="Print" StartIcon="@Icons.Material.Filled.Print">
            @Localizer["Apartments_Print"]
        </MudButton>
        <MudButton Variant="Variant.Outlined" Color="Color.Secondary"
OnClick="SendEmail" StartIcon="@Icons.Material.Filled.Email">
            @Localizer["Apartments_SendEmail"]
        </MudButton>
        <MudButton Variant="Variant.Outlined" Color="Color.Success"
OnClick="ExportExcel" StartIcon="@Icons.Material.Filled.Download">
            @Localizer["Apartments_ExportExcel"]
        </MudButton>
        <MudButton Variant="Variant.Filled" Color="Color.Info"
OnClick="Close" StartIcon="@Icons.Material.Filled.Close">
            @Localizer["Apartments_ButtonClose"]
        </MudButton>
    </DialogActions>
</MudDialog>

@code {
```

```
	[CascadingParameter] IMudDialogInstance MudDialog { get; set; } =
null!;
	[Parameter] public string ResidenceId { get; set; } = null!;

	private List<OperationViewModel> _operations = new();
	private bool _loading = true;

	protected override async Task OnInitializedAsync()
	{
		try
		{
			await using var db = await
DbFactory.CreateDbContextAsync();

			var ops = await db.Operations
			    .Where(o => o.ResidenceId == ResidenceId)
			    .OrderBy(o => o.OperationDate)
			    .ThenBy(o => o.Id)
			    .ToListAsync();

			decimal runningBalance = 0m;
			var operationsForView = new List<OperationViewModel>();

			foreach (var o in ops)
			{
			    switch (o.Type)
			    {
			        case OperationType.InitialBalance:
			            runningBalance = o.Value;
			            break;

			        case OperationType.RegularAccrual:
			        case OperationType.OneTimeAccrual:
			            runningBalance += o.Value;
			            break;

			        case OperationType.Payment:
```

```
                        runningBalance -= o.Value;
                        break;

                    default:
                        break;
                }

                operationsForView.Add(new OperationViewModel(o.Id,
o.OperationDate, o.Value, runningBalance)
                {
                    Type = AccrualsLocalizer[$"OperationType.{o.Type}"]
                });
            }
            _operations = operationsForView
                .OrderByDescending(x => x.OperationDate)
                .ThenByDescending(x => x.Id)
                .ToList();
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{Localizer["Apartments_ErrorData"]} {ex.Message}",
Severity.Error);
        }
        finally
        {
            _loading = false;
        }
    }

    private async Task Print()
    {
        try
        {
            var printContent =
ReportingGenerator.GenerateOperationsReportHtml(ResidenceId, _operations);
            await JSRuntime.InvokeVoidAsync("printResidenceCard",
printContent);
```

```
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{Localizer["PrintError"]}: {ex.Message}",
Severity.Error);

            }
      }

    private async Task SendEmail()
    {
        try
        {
            await using var db = await DbFactory.CreateDbContextAsync();
            var residence = await db.Residences
                .Include(r => r.Resident)
                .FirstOrDefaultAsync(r => r.Id == ResidenceId);

            if (residence?.Resident?.Email == null)
            {
                Snackbar.Add($"{Localizer["Apartments_NoEmail"]}",
Severity.Warning);
                return;
            }

            string subject = $"Звіт по квартирі {ResidenceId}";
            string body =
ReportingGenerator.GenerateOperationsReportHtml(ResidenceId, _operations);
            await EmailSender.SendEmailAsync(residence.Resident.Email,
subject, body);
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{Localizer["Apartments_EmailError"]}
{ex.Message}", Severity.Error);
        }
    }
```

```
    private async Task ExportExcel()
    {
        try
        {
            var csv = new StringBuilder();
            csv.AppendLine("Дата;Операція;Сума;Баланс");

            foreach (var op in _operations)
            {
                var valueStr = op.Value % 1 == 0 ? ((int)op.Value).ToString()
: op.Value.ToString("0.##");
                var balanceStr = op.Balance % 1 == 0 ?
((int)op.Balance).ToString() : op.Balance.ToString("0.##");

csv.AppendLine($"{op.OperationDate:dd.MM.yyyy};{op.Type};{valueStr};{balanceS
tr}");
            }

            var fileName =
$"Operations_{ResidenceId}_{DateTime.Now:yyyyMMddHHmm}.csv";

            await JSRuntime.InvokeVoidAsync("downloadCsv", fileName,
csv.ToString());

            Snackbar.Add($"{Localizer["Apratments_CsvExport"]}",
Severity.Success);
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{Localizer["Apartments_ExportError"]}
{ex.Message}", Severity.Error);
        }
    }

      private void Close() => MudDialog.Cancel();
}
```

Додаток Б. Лістинг коду компонента PersonDataGrid

```
@using Microsoft.EntityFrameworkCore
@using Microsoft.Extensions.Localization
@using MudBlazor
@using System.Collections.ObjectModel
@using System.Threading
@using ResidentialTracker.Components.Shared
@using ResidentialTracker.Data

@implements IAsyncDisposable

@inject IStringLocalizer<ResidentialTracker.Resources.Pages.Persons>
Localizer
@inject IDbContextFactory<ApplicationDbContext> DbFactory
@inject IDialogService DialogService
@inject ISnackbar Snackbar

@rendermode InteractiveServer

<MudDataGrid @ref="_dataGrid" T="Person"
             ServerData="@LoadServerData"
             SortMode="SortMode.Single"
             Filterable="true"
             Hideable="true"
             Dense="true"
             ReadOnly="false"
             FixedHeader="true"
             LoadingProgressColor="Color.Info"
             Bordered="true"
             Height="calc(100vh - 300px)"
             Class="mt-4">
    <ToolBarContent>
        <MudText Typo="Typo.h6" Class="mud-text-secondary mb-
0">@Localizer["Person_Title"]</MudText>
        <MudSpacer />
```

```
        <MudTextField @bind-Value="_searchString"
Placeholder="@Localizer["Person_Search"]" Adornment="Adornment.Start"
                      AdornmentIcon="@Icons.Material.Filled.Search"
IconSize="Size.Medium" Class="mt-0 mr-2"
                      Immediate="true"
                      DebounceInterval="500"
                      OnDebounceIntervalElapsed="@(async () => await
_dataGrid.ReloadServerData())" />
        <MudButton Color="Color.Primary"
                   Variant="Variant.Filled"
                   StartIcon="@Icons.Material.Filled.Add"
                   OnClick="OpenAddDialog"
                   Disabled="@_isProcessing">
            @Localizer["Person_AddButton"]
        </MudButton>
        <MudButton Color="Color.Secondary"
                   Variant="Variant.Filled"
                   StartIcon="@Icons.Material.Filled.FileUpload"
                   OnClick="OpenImportDialog"
                   Disabled="@_isProcessing"
                   Class="ml-2">
            @Localizer["Person_Import"]
        </MudButton>
    </ToolBarContent>

    <Columns>
        <PropertyColumn Property="x => x.Id" Title="@Localizer["Person_Id"]"
Sortable="false" Filterable="false" CellStyle="width:100px" />
        <PropertyColumn Property="x => x.LastName"
Title="@Localizer["Person_LastName"]" />
        <PropertyColumn Property="x => x.FirstName"
Title="@Localizer["Person_FirstName"]" />
        <PropertyColumn Property="x => x.FatherName"
Title="@Localizer["Person_FatherName"]" />
        <PropertyColumn Property="x => x.Email"
Title="@Localizer["Person_Email"]" />
```

```
        <PropertyColumn Property="x => x.PhoneNumber"
Title="@Localizer["Person_PhoneNumber"]" />
        <TemplateColumn CellClass="d-flex justify-center" Filterable="false"
Sortable="false" Title="@Localizer["Person_Actions"]">
            <CellTemplate Context="context">
                <MudTooltip Text="@Localizer["Person_Edit"]"
Placement="Placement.Top">
                    <MudIconButton Size="Size.Small"
Icon="@Icons.Material.Filled.Edit" Color="Color.Primary"
                                   OnClick="@(() =>
OpenEditDialog(context.Item))" Disabled="@_isProcessing" />
                </MudTooltip>
                <MudTooltip Text="@Localizer["Person_Delete"]"
Placement="Placement.Top">
                    <MudIconButton Size="Size.Small"
Icon="@Icons.Material.Filled.Delete" Color="Color.Error"
                                   OnClick="@(() =>
OpenDeleteDialog(context.Item))" Disabled="@_isProcessing" />
                </MudTooltip>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <NoRecordsContent>
        <MudText>@Localizer["Person_NoRecordsFound"]</MudText>
    </NoRecordsContent>
    <LoadingContent>
        <MudText>@Localizer["Person_Loading"]</MudText>
    </LoadingContent>
    <PagerContent>
        <MudDataGridPager T="Person"
RowsPerPageString="@Localizer["Person_RowsPerPage"]" PageSizeOptions="new
int[] { 10, 25, 50, 100 }" />
    </PagerContent>
</MudDataGrid>

@code {
    private ApplicationDbContext _context = null!;
```

```
    private MudDataGrid<Person> _dataGrid = null!;
    private string _searchString = string.Empty;
    private bool _isProcessing = false;
    private readonly CancellationTokenSource _cts = new();

    private async Task<GridData<Person>> LoadServerData(GridState<Person>
state)
    {
        try
        {
            if (_context == null)
            {
                _context = await DbFactory.CreateDbContextAsync(_cts.Token);
            }

            // Begin constructing the query with improved performance
            IQueryable<Person> query = _context.Persons
                .AsNoTracking()
                .TagWith("PersonDataGrid:LoadServerData"); // Tag for better
query diagnostics

            // Apply search filter using EF Core's optimized methods
            if (!string.IsNullOrWhiteSpace(_searchString))
            {
                var searchTerm = _searchString.Trim();

                // Only extract digits if the search term contains any digits
                var searchDigits = searchTerm.Any(char.IsDigit) ?
ExtractDigits(searchTerm) : string.Empty;

                // Use database-side filtering with EF.Functions for better
performance
                query = query.Where(p =>
                    (p.FirstName != null && EF.Functions.Like(p.FirstName,
$"%{searchTerm}%")) ||
                    (p.LastName != null && EF.Functions.Like(p.LastName,
$"%{searchTerm}%")) ||
```

```
                    (p.FatherName != null && EF.Functions.Like(p.FatherName,
$"%{searchTerm}%")) ||
                    (p.Email != null && EF.Functions.Like(p.Email,
$"%{searchTerm}%")) ||
                    (!string.IsNullOrEmpty(searchDigits) && p.PhoneNumber !=
null && p.PhoneNumber.Contains(searchDigits))
                );
            }

            // Execute count query once
            var totalItems = await query.CountAsync(_cts.Token);

            // Apply sorting with more efficient expression compilation
            if (state.SortDefinitions.Any())
            {
                query = ApplySorting(query, state);
            }
            else
            {
                // Default sorting
                query = query.OrderBy(p => p.Id);
            }

            // Apply pagination efficiently
            var pagedData = await query
                .Skip(state.Page * state.PageSize)
                .Take(state.PageSize)
                .ToListAsync(_cts.Token);

            return new GridData<Person>
            {
                Items = pagedData,
                TotalItems = totalItems
            };
        }
        catch (OperationCanceledException)
        {
```

```
            // Operation was canceled, return empty result
            return new GridData<Person> { Items = new List<Person>(),
TotalItems = 0 };
        }
        catch (Exception ex)
        {
            Snackbar.Add(Localizer["Person_ErrorLoadingData", ex.Message],
Severity.Error);
            return new GridData<Person> { Items = new List<Person>(),
TotalItems = 0 };
        }
    }

    // Helper method to apply sorting with better performance and null
handling
    private static IQueryable<Person> ApplySorting(IQueryable<Person> query,
GridState<Person> state)
    {
        var sortDefinition = state.SortDefinitions.FirstOrDefault();

        // If no sorting specified, return with default sort
        if (sortDefinition == null)
            return query.OrderBy(p => p.Id);

        // Apply sort based on property name with proper null handling
        return sortDefinition.SortBy switch
        {
            nameof(Person.LastName) => sortDefinition.Descending
                ? query.OrderByDescending(p => p.LastName ?? string.Empty)
                : query.OrderBy(p => p.LastName ?? string.Empty),

            nameof(Person.FirstName) => sortDefinition.Descending
                ? query.OrderByDescending(p => p.FirstName ?? string.Empty)
                : query.OrderBy(p => p.FirstName ?? string.Empty),

            nameof(Person.FatherName) => sortDefinition.Descending
                ? query.OrderByDescending(p => p.FatherName ?? string.Empty)
```

```
            : query.OrderBy(p => p.FatherName ?? string.Empty),

        nameof(Person.Email) => sortDefinition.Descending
            ? query.OrderByDescending(p => p.Email ?? string.Empty)
            : query.OrderBy(p => p.Email ?? string.Empty),

        nameof(Person.PhoneNumber) => sortDefinition.Descending
            ? query.OrderByDescending(p => p.PhoneNumber ?? string.Empty)
            : query.OrderBy(p => p.PhoneNumber ?? string.Empty),

        _ => sortDefinition.Descending
            ? query.OrderByDescending(p => p.Id)
            : query.OrderBy(p => p.Id)
    };
}

private static string ExtractDigits(string? input)
{
    if (string.IsNullOrWhiteSpace(input))
        return string.Empty;

    return string.Concat(input.Where(char.IsDigit));
}

private async Task OpenAddDialog()
{
    if (_isProcessing) return;
    _isProcessing = true;

    try
    {
        var parameters = new DialogParameters
        {
            { "IsEdit", false }
        };

        var options = new DialogOptions
```

```
            {
                CloseOnEscapeKey = true,
                BackdropClick = true,
                MaxWidth = MaxWidth.Medium
            };

            var dialog = await
DialogService.ShowAsync<PersonEditDialog>(Localizer["Person_AddDialogTitle"],
parameters, options);
            var result = await dialog.Result;

            if (result != null && !result.Canceled)
            {
                await _dataGrid.ReloadServerData();
                Snackbar.Add(Localizer["Person_AddedSuccessfully"],
Severity.Success);
            }
        }
        catch (Exception ex)
        {
            Snackbar.Add(Localizer["Person_Error", ex.Message],
Severity.Error);
        }
        finally
        {
            _isProcessing = false;
            StateHasChanged();
        }
    }

    private async Task OpenEditDialog(Person person)
    {
        if (_isProcessing) return;
        _isProcessing = true;

        try
        {
```

```
            var parameters = new DialogParameters
            {
                { "PersonId", person.Id },
                { "IsEdit", true }
            };

            var options = new DialogOptions
            {
                CloseOnEscapeKey = true,
                BackdropClick = true,
                MaxWidth = MaxWidth.Medium
            };

            var dialog = await
DialogService.ShowAsync<PersonEditDialog>(Localizer["Person_EditDialogTitle"]
, parameters, options);
            var result = await dialog.Result;

            if (result != null && !result.Canceled)
            {
                await _dataGrid.ReloadServerData();
                Snackbar.Add(Localizer["Person_UpdatedSuccessfully"],
Severity.Success);
            }
        }
        catch (Exception ex)
        {
            Snackbar.Add(Localizer["Person_Error", ex.Message],
Severity.Error);
        }
        finally
        {
            _isProcessing = false;
            StateHasChanged();
        }
    }
```

```
    private async Task OpenDeleteDialog(Person person)
    {
        if (_isProcessing) return;
        _isProcessing = true;

        try
        {
            var parameters = new DialogParameters<ConfirmationDialog>
            {
                { x => x.ContentText, Localizer["Person_DeleteConfirmation",
person.FullName] },
                { x => x.Color, Color.Error }
            };

            var options = new DialogOptions
            {
                CloseButton = true,
                MaxWidth = MaxWidth.ExtraSmall,
                CloseOnEscapeKey = true,
                BackdropClick = false
            };

            var dialog = await
DialogService.ShowAsync<ConfirmationDialog>(Localizer["Person_DeleteDialogTit
le"], parameters, options);
            var result = await dialog.Result;

            if (result != null && !result.Canceled)
            {
                try
                {
                    var personToDelete = await _context.Persons.FindAsync(new
object[] { person.Id }, _cts.Token);
                    if (personToDelete != null)
                    {
                        _context.Persons.Remove(personToDelete);
                        await _context.SaveChangesAsync(_cts.Token);
```

```
                    await _dataGrid.ReloadServerData();
                    Snackbar.Add(Localizer["Person_DeletedSuccessfully"],
Severity.Success);
                }
                else
                {
                    Snackbar.Add(Localizer["Person_NotFound"],
Severity.Warning);
                    await _dataGrid.ReloadServerData();
                }
            }
            catch (DbUpdateException ex)
            {
                // More specific error handling for database-related
exceptions
                if (ex.InnerException?.Message.Contains("FK_") == true)
                {

Snackbar.Add(Localizer["Person_CannotDeleteReferenced"], Severity.Error);
                }
                else
                {
                    Snackbar.Add(Localizer["Person_DatabaseError",
ex.Message], Severity.Error);
                }
            }
            catch (Exception ex)
            {
                Snackbar.Add(Localizer["Person_Error", ex.Message],
Severity.Error);
            }
        }
    }
    finally
    {
        _isProcessing = false;
        StateHasChanged();
```

```
        }
    }

    private async Task OpenImportDialog()
    {
        if (_isProcessing) return;
        _isProcessing = true;

        try
        {
            var parameters = new DialogParameters
            {
                { "EntityType", "Person" }
            };

            var options = new DialogOptions
            {
                CloseOnEscapeKey = true,
                BackdropClick = true,
                MaxWidth = MaxWidth.Medium
            };

            var dialog = await
DialogService.ShowAsync<CsvImportDialog>(Localizer["Person_ImportDialogTitle"
], parameters, options);
            var result = await dialog.Result;

            if (result != null && !result.Canceled)
            {
                await _dataGrid.ReloadServerData();
                Snackbar.Add(Localizer["Person_ImportSuccessful"],
Severity.Success);
            }
        }
        catch (Exception ex)
        {
```

```
            Snackbar.Add(Localizer["Person_ImportError", ex.Message],
Severity.Error);
        }
        finally
        {
            _isProcessing = false;
            StateHasChanged();
        }
    }

    public async ValueTask DisposeAsync()
    {
        await DisposeAsyncCore();
        GC.SuppressFinalize(this);
    }

    protected virtual async ValueTask DisposeAsyncCore()
    {
        if (_context != null)
        {
            // Dispose of the DbContext if it was created
            await _context.DisposeAsync();
        }
        // Cancel any pending operations when component is disposed
        await _cts.CancelAsync();
        _cts.Dispose();
        await ValueTask.CompletedTask;
    }
}
```