МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ З АВТОМАТИЗАЦІЇ ДІЯЛЬНОСТІ ОБ'ЄДНАННЯ СПІВВЛАСНИКІВ БАГАТОКВАРТИРНОГО БУДИНКУ»**

Виконала: студентка 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.О. Ткаліч
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Ткаліч Маргариті Олександрівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи з автоматизації діяльності об’єднання співвласників багатоквартирного будинку

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз та специфікація вимог.
2. Проєктування системи.
3. Реалізація та тестування системи управління ОСББ.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	28.01.2026	
3.	Обробка методичних та теоретичних	20.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	08.04.2026	
5.	Розробка третього розділу.	15.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ М.О. Ткаліч
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи з автоматизації діяльності об’єднання співвласників багатоквартирного будинку»: 71 с., 25 рис., 7 табл., 9 джерел, 2 додатки.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ІНФОРМАЦІЙНА СИСТЕМА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ОСББ, ASP.NET CORE, BLAZOR WEBASSEMBLY, ENTITY FRAMEWORK CORE, JWT, MYSQL, REST API, UML.

Об’єкт дослідження – процеси розроблення веборієнтованих інформаційних систем автоматизації діяльності ОСББ.

Предмет дослідження – методи та програмні засоби реалізації клієнт-серверної інформаційної системи ОСББ.

Мета роботи – зменшення часу обробки інформації та спрощення управління діяльністю ОСББ шляхом розробки інформаційної системи автоматизації діяльності об’єднання співвласників багатоквартирного будинку.

Методи дослідження – аналіз предметної області, аналіз вимог до програмного забезпечення, методи моделювання та проєктування інформаційних систем, методи проєктування баз даних, методи розробки та методи тестування програмного забезпечення.

Під час виконання роботи проведено аналіз предметної області та існуючих програмних рішень для управління діяльністю ОСББ. Спроєктовано архітектуру системи, базу даних та UML-діаграми, що відображають структуру і функціонування програмного забезпечення. Реалізовано клієнтську частину засобами Blazor WebAssembly та серверну частину на платформі ASP.NET Core із використанням REST API та СУБД MySQL. У результаті роботи розроблено вебзастосунок для автоматизації діяльності ОСББ.

SUMMARY

Bachelor's qualifying paper "Development of an Information System for Automating the Activities of Condominium Associations": 71 pages, 25 figures, 7 tables, 9 references, 2 supplements.

ASP.NET CORE, BLAZOR WEBASSEMBLY, CLIENT-SERVER ARCHITECTURE, DATABASE, ENTITY FRAMEWORK CORE, HOMEOWNERS ASSOCIATION, INFORMATION SYSTEM, JWT, MYSQL, REST API, UML, WEB APPLICATION.

The object of the study is the process of developing web-oriented information systems for automating the activities of homeowners associations.

The subject of the study is methods and software tools for implementing a client-server information system for automating the activities of a homeowners association.

The aim of the study is to reduce information processing time and simplify the management of homeowners association activities through the development of an information system for automating the activities of a homeowners association.

The methods of research include analysis of the subject area, software requirements analysis, information system modeling and design methods, database design methods, software development methods, and software testing methods.

During the research, the subject area and existing software solutions for homeowners association management were analyzed. The architecture of the information system, database structure, and UML diagrams were designed to describe the structure and functionality of the software. The client-side application was implemented using Blazor WebAssembly, while the server-side application was developed using ASP.NET Core, REST API, Entity Framework Core, and MySQL.

As a result of the study, a web application for automating homeowners association activities was developed.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз та специфікація вимог 10
1.1 Аналіз діяльності ОСББ 10
1.2 Аналіз існуючих програмних рішень..... 11
1.2.1 Аналіз існуючого програмного рішення «ОСББ 365» 11
1.2.2 Аналіз існуючого програмного рішення «Мій дім online»..... 13
1.3 Вимоги до програмного забезпечення 15
1.4 Вибір технологій 17
2 Проєктування системи..... 19
2.1 Архітектура системи..... 19
2.1.1 Архітектура клієнтської частини 21
2.1.2 Архітектура серверної частини 22
2.2 Проєктування бази даних..... 24
2.2.1 Сутності бази даних..... 25
2.2.2 Зв’язки та кратність 26
2.3 UML діаграми..... 27
2.3.1 UML діаграма використання 27
2.3.2 Діаграма активності процесу оплати нарахувань..... 29
2.3.3 Діаграма активності послідовності процесу створення оголошення 31
2.3.4 Діаграма класів..... 33
3 Реалізація та тестування системи управління ОСББ..... 36
3.1 Загальна структура серверної частини 36
3.1.1 Взаємодія клієнта та сервера 37

3.1.2 Реалізація бази даних 39
3.1.3 Реалізація розрахунку платежів (нарахування) 40
3.1.4 Реалізація оплати та формування квитанцій 43
3.2 Реалізація клієнтської частини та основні функції вебзастоснку 46
3.2.1 Головна сторінка 47
3.2.2 Призначення нарахувань (голова ОСББ) 48
3.2.3 Публікація оголошень 50
3.2.4 Сторінка подання заявки на заселення 51
3.2.5 Сторінка обробки заявок мешканців 52
3.3 Тестування вебзастосунку 54
3.3.1 Ручне тестування 54
3.3.2 Автоматизоване тестування 55
Висновки 58
Перелік посилань 60
Додаток А Лістинги SQL-скриптів бази даних 61
Додаток Б Автоматизоване тестування 70

ВСТУП

У сучасних умовах автоматизація процесів управління житловими комплексами та об’єднаннями співвласників багатоквартирних будинків (ОСББ) є важливим напрямом розвитку інформаційних технологій. Використання вебзастосунків дозволяє значно спростити облік мешканців, контроль оплат, формування нарахувань, обробку заявок та інформування співвласників про діяльність ОСББ.

Більшість ОСББ продовжують використовувати паперовий документообіг або прості електронні таблиці, що ускладнює обробку інформації, призводить до дублювання даних та збільшує час виконання адміністративних операцій. Крім того, відсутність єдиної централізованої системи ускладнює взаємодію між мешканцями та керівництвом ОСББ.

Актуальність теми полягає у необхідності створення сучасної інформаційної системи, яка забезпечить автоматизацію основних бізнес-процесів ОСББ, підвищить ефективність управління та покращить взаємодію між користувачами системи.

Метою кваліфікаційної роботи є розробка інформаційної системи автоматизації діяльності ОСББ із можливістю управління квартирами, мешканцями, заявками, нарахуваннями, оплатами та оголошеннями.

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

– проаналізувати предметну область діяльності ОСББ;
– обґрунтувати вибір технологій розробки;
– архітектуру системи та структуру бази даних;
– реалізувати клієнтську та серверну частини вебзастосунку;
– реалізувати REST API для взаємодії між компонентами системи;
– реалізувати механізми обліку оплат, заявок та оголошень;
– провести тестування функціональних можливостей системи.

Під час розробки системи використано технології Blazor WebAssembly, ASP.NET Core API, Entity Framework Core та MySQL. Для взаємодії між

клієнтською та серверною частинами використано REST API та формат обміну даними JSON.

Отриманий результат та його значення полягає у можливості використання розробленої системи для автоматизації діяльності ОСББ, спрощення обліку платежів, управління квартирами та покращення комунікації між мешканцями й адміністрацією ОСББ.

Перший розділ включає аналіз діяльності ОСББ, огляд існуючих програмних рішень для автоматизації управління об’єднаннями співвласників багатоквартирних будинків, визначення функціональних та нефункціональних вимог до програмного забезпечення, а також обґрунтування вибору засобів і технологій розробки.

Другий розділ присвячений проєктуванню системи. У ньому наведено ER-модель бази даних, UML-діаграму варіантів використання (Use Case Diagram), діаграми активності основних бізнес-процесів, а також описано архітектуру програмної системи.

Третій розділ містить опис реалізації інформаційної системи управління діяльністю ОСББ. У розділі розглянуто структуру клієнтської та серверної частин застосунку, реалізацію бази даних, REST API, механізми автентифікації та авторизації користувачів, особливості реалізації основного функціоналу системи, а також результати тестування розробленого програмного забезпечення.

1 АНАЛІЗ ТА СПЕЦИФІКАЦІЯ ВИМОГ

1.1 Аналіз діяльності ОСББ

ОСББ – це об’єднання співвласників багатоквартирного будинку, яке створюється для управління будинком та прибудинковою територією. Основним завданням ОСББ є забезпечення комфортних умов проживання мешканців, контроль платежів та організація роботи будинку [1].

У процесі роботи ОСББ виконується багато різних задач. Голова ОСББ веде облік квартир і мешканців, нараховує платежі за комунальні послуги, перевіряє заборгованість, публікує оголошення та приймає заявки від мешканців на заселення в ОСББ [1].

Мешканці будинку, у свою чергу, повинні мати можливість переглядати інформацію про свою квартиру, перевіряти нарахування, оплачувати послуги та подавати заявки на заселення.

У багатьох ОСББ частина цих процесів виконується вручну: інформація зберігається у таблицях або записується на папері, а спілкування відбувається через телефон чи месенджери. Це займає багато часу та може призводити до помилок або втрати інформації.

Однією з головних проблем є контроль платежів та заборгованостей. При великій кількості квартир складно швидко перевіряти оплату та вести точний облік. Також незручно повідомляти всіх мешканців про важливі події або зміни.

Для вирішення цих проблем доцільно створити вебзастосунок для автоматизації роботи ОСББ. Така система дозволить зберігати всю інформацію в одному місці, спростить роботу голові ОСББ та зробить взаємодію мешканців з ОСББ більш зручною.

Розроблюваний вебзастосунок повинен забезпечувати:

– управління інформацією про ОСББ;
– роботу з квартирами та користувачами;

– обробку заявок на заселення або переїзд;
– створення оголошень;
– формування нарахувань;
– оплат та квитанцій.

Автоматизація діяльності ОСББ допоможе зменшити кількість ручної роботи, покращити облік інформації та підвищити зручність користування системою для мешканців і адміністрації будинку.

1.2 Аналіз існуючих програмних рішень

Під час проєктування інформаційної системи для автоматизації діяльності ОСББ було проведено аналіз існуючих програмних рішень: «ОСББ 365» та «Мій дім online», які використовуються для управління житловими об’єднаннями. Аналіз аналогів дозволив визначити основні функціональні можливості сучасних систем, їх переваги та недоліки, а також сформувати вимоги до власного програмного продукту.

1.2.1 Аналіз існуючого програмного рішення «ОСББ 365»

Для аналізу існуючих програмних рішень було розглянуто систему «ОСББ 365», яка призначена для автоматизації роботи ОСББ та управління багатоквартирними будинками. Система надає можливість вести облік платежів, контролювати заборгованість, організовувати взаємодію між мешканцями та адміністрацією будинку.

На рисунку 1.1 представлено інформацію про тарифні плани системи та умови використання сервісу.

Після завершення безкоштовного пробного періоду користувачам необхідно оформити платну підписку для подальшого використання

функціоналу системи. Для невеликих ОСББ це може створювати додаткові фінансові витрати.

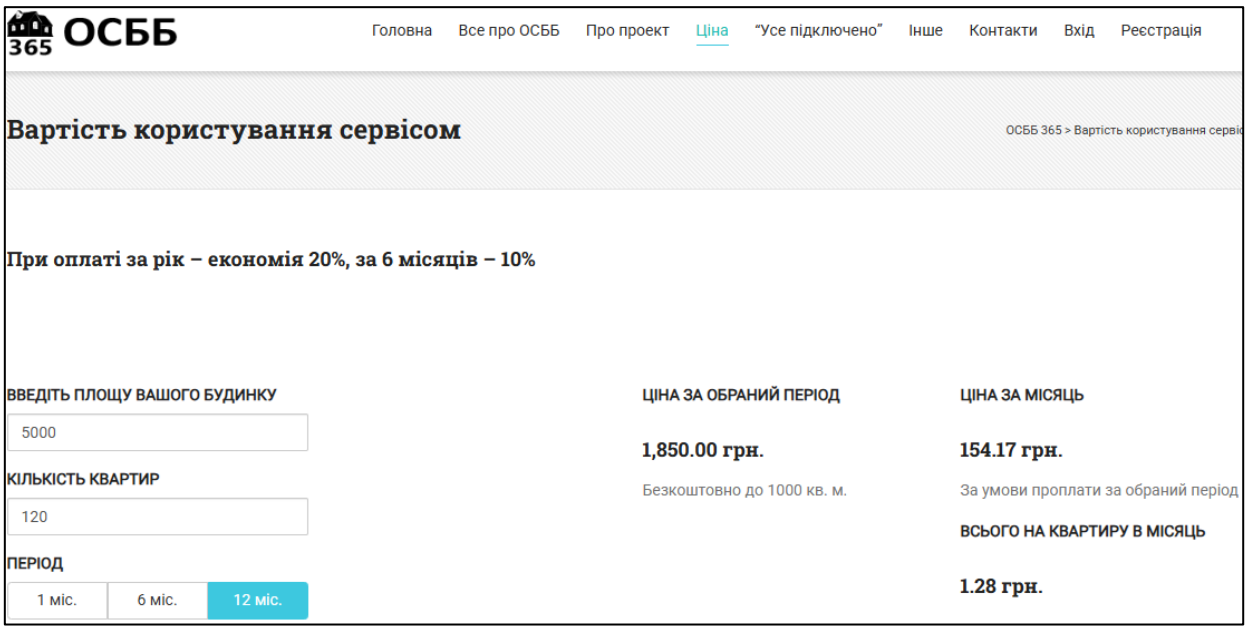


Рисунок 1.1 – Тарифні плани системи «ОСББ 365»

Система «ОСББ 365» також надає широкий набір функцій для адміністрування та контролю роботи ОСББ. На рисунку 1.2 представлено основні можливості системи.

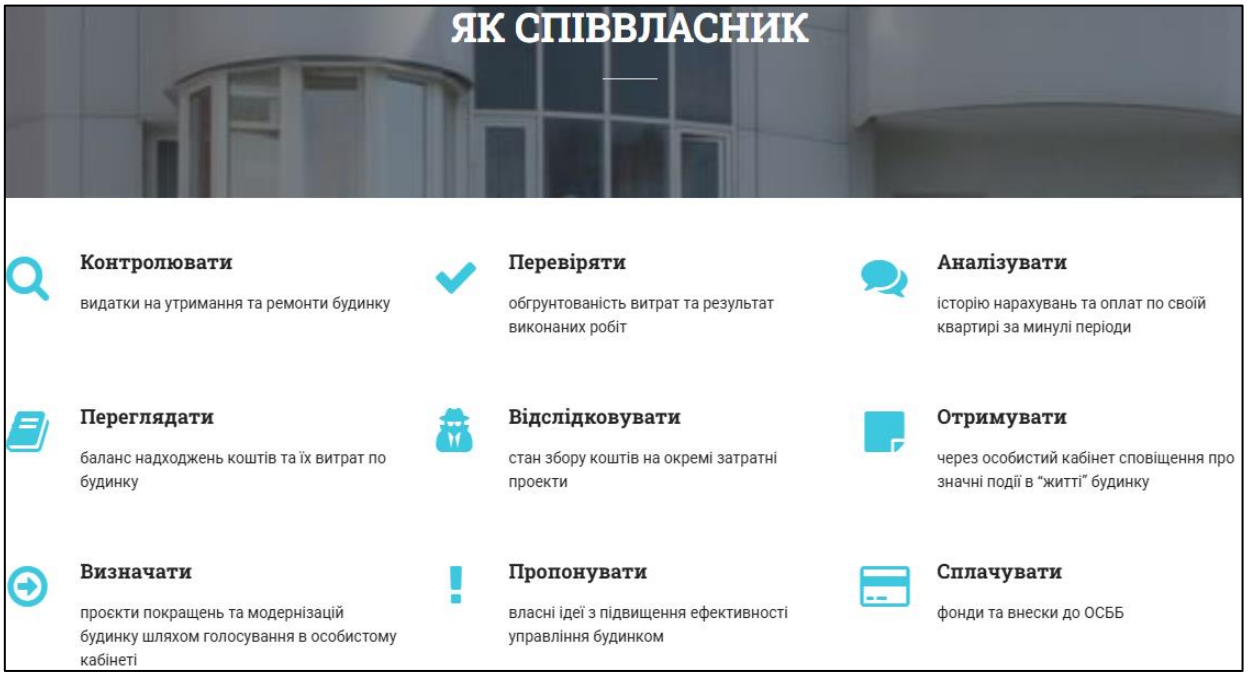


Рисунок 1.2 – Основні функції системи «ОСББ 365»

Система дозволяє виконувати облік платежів, формувати звітність, аналізувати інформацію, контролювати нарахування та інформувати мешканців будинку. Крім того, користувачі можуть переглядати дані про оплату послуг та здійснювати контроль заборгованостей.

На основі проведеного аналізу системи «ОСББ 365» можна виділити її переваги та недоліки, які представлені у таблиці 1.1.

Таблиця 1.1 – Аналіз системи «ОСББ 365»

Переваги	– широкий функціонал; – контроль платежів та боргів; – можливість інформування мешканців; – введення обліку мешканців.
Недоліки	– платна підписка після пробного періоду; – застарілий інтерфейс; – частина функцій може бути складною для користувачів.

Отже, незважаючи на широкий функціонал існуючих систем, вони мають певні обмеження, що обґрунтовує доцільність розробки власного вебзастосунку для управління ОСББ.

1.2.2 Аналіз існуючого програмного рішення «Мій дім online»

Одним із найбільш відомих програмних рішень для автоматизації діяльності ОСББ в Україні є система «Мій дім online», інтерфейс представлено на рисунку 1.3. Дане рішення призначене для ведення бухгалтерського та абонентського обліку, взаємодії між мешканцями та головою ОСББ, а також автоматизації основних процесів управління.

Система реалізована у вигляді вебплатформи для мешканців. Інтерфейс містить модулі управління нерухомістю, особовими рахунками, заявками,

лічильниками, фінансами та документами. Також система підтримує електронні опитування, розсилки повідомлень, інтеграцію з платіжними системами та ведення звітності.

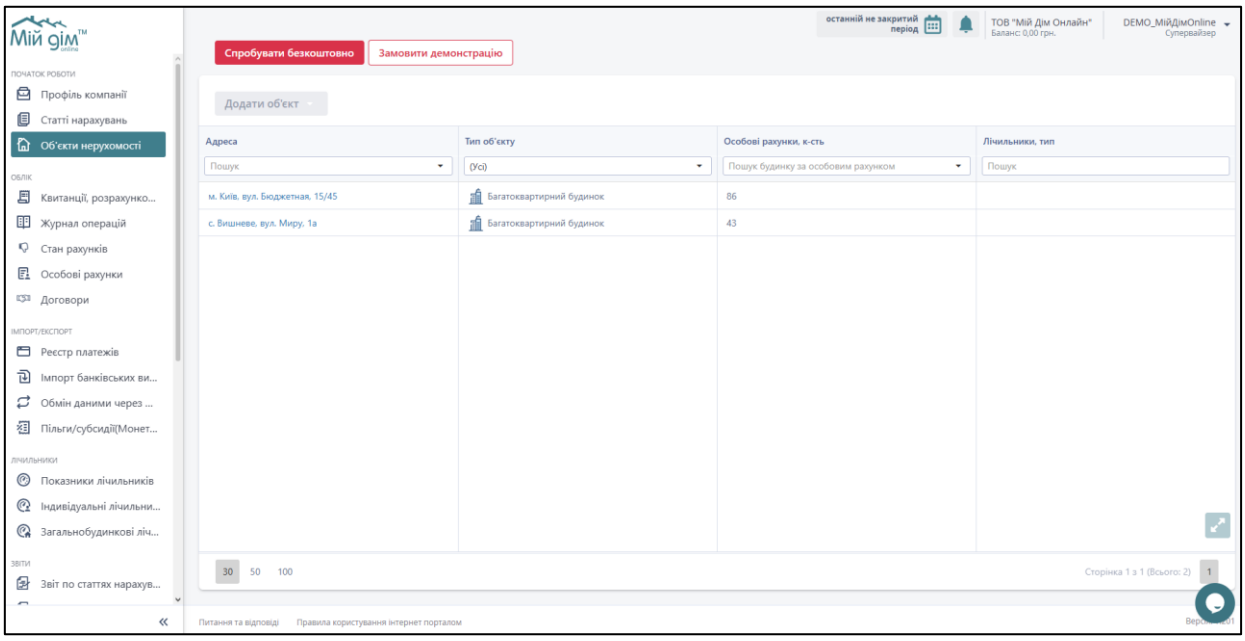


Рисунок 1.3 – Особистий кабінет мешканця в системі ОСББ «Мій дім online»

На рисунку 1.4 представлено приклад тарифних планів системи. Аналіз показує, що функціональні можливості сервісу залежать від обраного тарифу. Це дозволяє масштабувати систему відповідно до потреб ОСББ, однак частина функцій є недоступною у базовому тарифі.

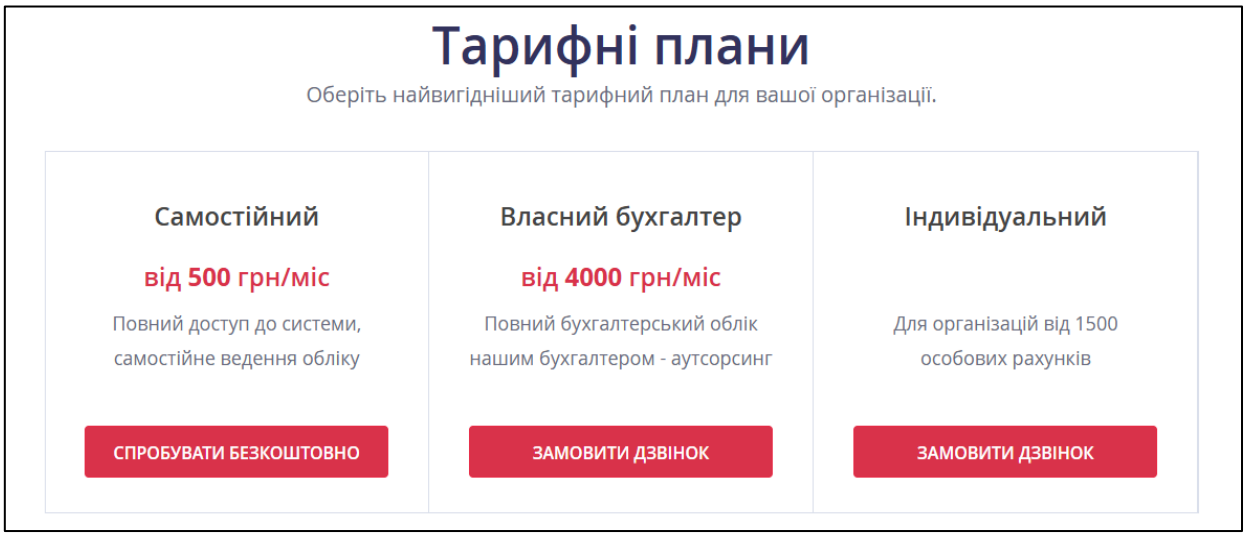


Рисунок 1.4 – Тарифи системи ОСББ «Мій дім online»

На основі проведеного аналізу системи ОСББ «Мій дім online» можна виділити її переваги та недоліки, які представлені у таблиці 1.2.

Таблиця 1.2 – Аналіз системи ОСББ «Мій дім online»

Переваги	– наявність вебверсії та мобільного за стосунку; – автоматизація бухгалтерського обліку; – підтримка онлайн-оплати послуг; – підтримка роботи з лічильниками та нарахуваннями.
Недоліки	– велика кількість функцій ускладнює інтерфейс для нових користувачів; – частина можливостей доступна лише у платних тарифах; – перевантаження інтерфейсу великою кількістю елементів меню; – частина модулів може бути надлишковою для користувачів.

Проведений аналіз показав, що сучасні системи для ОСББ орієнтовані на автоматизацію управління будинком та взаємодії з мешканцями. Водночас надмірна кількість функцій та складність інтерфейсу можуть негативно впливати на зручність використання системи. Тому під час розробки власного програмного рішення доцільно приділити особливу увагу простоті інтерфейсу, адаптивності та зручності взаємодії користувача із системою.

1.3 Вимоги до програмного забезпечення

Розроблюваний вебзастосунок призначений для автоматизації роботи ОСББ та спрощення взаємодії між мешканцями будинку і адміністрацією будинку (голова ОСББ). Система повинна забезпечувати зручний доступ до інформації, контроль платежів та можливість швидкого обміну повідомленнями й заявками.

Розроблювана інформаційна система автоматизації діяльності ОСББ повинна забезпечувати виконання таких функціональних вимог:

– реєстрація та авторизація користувачів у системі;
– розмежування прав доступу між мешканцями та головою ОСББ;
– перегляд інформації про квартиру та особовий рахунок мешканця;
– перегляд нарахувань, заборгованостей та статусів оплат;
– можливість здійснення оплат та формування квитанцій;
– перегляд та створення оголошень;
– подання та обробка заявок мешканців;
– редагування особистих даних користувача;
– керування квартирами та мешканцями з боку голови ОСББ;
– створення та редагування нарахувань;
– перегляд списку боржників;
– збереження даних у реляційній базі даних;
– взаємодія клієнтської та серверної частин через REST API;
– автоматичне оновлення інформації після виконання операцій у системі.

До розроблюваної системи належать такі нефункціональні вимоги:

– система повинна коректно працювати у сучасних веббраузерах;
– інтерфейс користувача повинен бути простим, зрозумілим та зручним для користувачів різного віку;
– вебзастосунок повинен підтримувати адаптивний інтерфейс для мобільних пристроїв;
– система повинна підтримувати світлу та темну теми оформлення;
– доступ до адміністративних функцій повинен бути обмежений відповідно до ролі користувача;
– система повинна забезпечувати захист персональних даних користувачів;
– час обробки основних операцій не повинен створювати затримок для користувача;

– база даних повинна забезпечувати цілісність та надійність збереження інформації;
– архітектура системи повинна забезпечувати можливість подальшого розширення функціональних можливостей.

Для розробки програмного забезпечення використовуються технології Blazor, ASP.NET Core, мова програмування C# та СУБД MySQL. Використання вебтехнологій дозволяє працювати із системою через браузер без необхідності встановлення додаткових програм на пристрій користувача.

1.4 Вибір технологій

Для розробки вебзастосунку управління діяльністю ОСББ було обрано сучасний стек технологій, який забезпечує зручність використання системи, стабільну роботу та можливість подальшого розвитку проєкту. При виборі технологій враховувалася необхідність створення адаптивного вебзастосунку, який буде працювати як на комп’ютерах, так і на мобільних пристроях.

Основною мовою програмування було обрано C# та платформу .NET 9. Даний стек є популярним для створення сучасних вебзастосунків та має велику кількість готових інструментів для розробки. Платформа .NET підтримується компанією Microsoft та регулярно оновлюється, що забезпечує стабільність і безпеку системи.

Для створення клієнтської частини застосунку використовується Blazor WebAssembly [7]. Дана технологія дозволяє створювати вебінтерфейс за допомогою мови C# [6], без необхідності використання великої кількості JavaScript. Це спрощує розробку та підтримку проєкту, оскільки одна мова програмування використовується як для серверної, так і для клієнтської частини системи. Також Blazor підтримує компонентний підхід, що дозволяє повторно використовувати елементи інтерфейсу.

Для оформлення інтерфейсу використовуються HTML та CSS. Адаптивність системи реалізована за допомогою окремих стилів для

мобільних пристроїв, завдяки чому вебзастосунок коректно відображається на смартфонах, планшетах та персональних комп’ютерах.

Серверна частина системи реалізована за допомогою ASP.NET Core [5]. Обмін даними між клієнтською та серверною частинами відбувається через REST API. Для реалізації API використовується Minimal API, який дозволяє зробити серверний код простішим та компактнішим. Також у режимі розробки використовується OpenAPI для автоматичного створення документації та тестування API.

Для зберігання даних використовується система керування базами даних MySQL 8 [8]. Вона забезпечує надійне зберігання інформації та підтримує роботу з SQL-запитами. Для взаємодії з базою даних використовується Entity Framework Core. Це дозволяє працювати з базою даних через об’єкти мови C#, що спрощує реалізацію основних операцій із даними. Також у проєкті використовуються SQL-скрипти та збережені процедури.

Перевагою обраного стеку технологій є використання однієї мови програмування для всієї системи, сучасний адаптивний інтерфейс та зручна робота з базою даних. Крім того, застосунок можна запускати як на комп’ютері, так і на мобільних пристроях.

Разом із перевагами існують і певні недоліки. Наприклад, Blazor WebAssembly може мати довший час першого завантаження у браузері через завантаження компонентів .NET. Проте, незважаючи на це, обрані технології добре підходять для реалізації вебзастосунку управління діяльністю ОСББ.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Архітектура системи

Розроблений вебзастосунок управління діяльністю ОСББ побудований за клієнт-серверною архітектурою. Такий підхід дозволяє розділити систему на окремі компоненти, які відповідають за відображення інтерфейсу, обробку даних та їх зберігання [9]. Це спрощує підтримку програмного коду, забезпечує зручну організацію проєкту та дозволяє незалежно розвивати окремі частини системи.

Архітектура системи складається з трьох основних компонентів:

– клієнтської частини;

– серверної частини;

– бази даних.

Користувач взаємодіє із системою через вебінтерфейс у браузері на персональному комп’ютері або мобільному пристрої. Клієнтська частина, реалізована за допомогою Blazor WebAssembly, відповідає за відображення сторінок, роботу інтерфейсу та надсилання HTTP запитів до серверної частини.

Серверна частина реалізована за допомогою ASP.NET Core API. Вона виконує обробку запитів, реалізує бізнес-логіку системи та забезпечує взаємодію з базою даних.

Обмін даними між клієнтом та сервером здійснюється через REST API з використанням формату JSON.

Для зберігання інформації використовується база даних MySQL. У ній зберігаються дані про ОСББ, користувачів, квартири, заявки, оголошення, нарахування та квитанції. Доступ до бази даних реалізовано за допомогою Entity Framework Core, що дозволяє працювати з даними через об’єкти мови C#.

Програмна структура системи поділена на декілька логічних модулів:

– OSBBApp.Client – клієнтський вебзастосунок;

– OSBBApp.Server – серверна частина та API;

– OSBBApp.Shared – спільні моделі даних та DTO-об’єкти.

Загальну архітектуру системи представлено на рисунку 2.1.

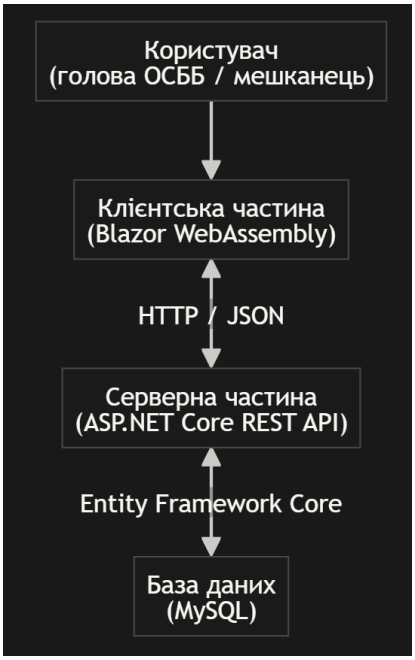


Рисунок 2.1 – Архітектура вебзастосунку ОСББ

Як показано на рисунку 2.1, користувач взаємодіє із системою через клієнтську частину, яка надсилає запити до серверної частини через API. Сервер обробляє отримані дані, виконує необхідні операції та взаємодіє з базою даних MySQL. Після обробки результат повертається клієнтській частині та відображається у вебінтерфейсі.

Перевагою такої архітектури є чіткий поділ системи на окремі компоненти, що спрощує підтримку та розвиток застосунку. Крім того, використання REST API дозволяє забезпечити стабільний обмін даними між клієнтом та сервером.

Разом із перевагами існують і недоліки. Наприклад, клієнт-серверна архітектура потребує додаткової реалізації взаємодії між frontend та backend частинами системи. Також обробка HTTP запитів та робота з API можуть

збільшувати складність розробки у порівнянні з простими монолітними застосунками.

2.1.1 Архітектура клієнтської частини

Клієнтська частина вебзастосунку реалізована за допомогою технології Blazor WebAssembly на платформі .NET 9. Дана технологія дозволяє виконувати логіку інтерфейсу безпосередньо у браузері користувача з використанням мови програмування C#. Завдяки цьому забезпечується швидка взаємодія користувача із системою та зменшується кількість перезавантажень сторінок під час роботи із застосунком.

Основним призначенням клієнтської частини є забезпечення взаємодії користувача із системою, відображення інформації та надсилання запитів до серверної частини через API. Залежно від ролі користувача система надає доступ до різних функціональних можливостей. Голова ОСББ має доступ до управління квартирами, заявками, нарахуваннями та оголошеннями, а мешканець до перегляду платежів, оголошень та власного профілю.

Інтерфейс застосунку побудований із використанням razor-компонентів, HTML та CSS. Компонентний підхід дозволяє розділити інтерфейс на окремі логічні елементи, які можуть повторно використовуватись у різних частинах системи. Це спрощує підтримку програмного коду та робить структуру проєкту більш зрозумілою.

Структура клієнтської частини включає декілька основних зон:

– публічну частину для входу у систему та подачі заявок;
– адміністративну частину для голови ОСББ;
– кабінет мешканця для роботи із власними даними.

Для навігації між сторінками використовується вбудований механізм маршрутизації Blazor. Перехід між сторінками здійснюється без повного перезавантаження вебзастосунку, що позитивно впливає на швидкість роботи інтерфейсу.

Окрему увагу приділено адаптивності інтерфейсу. Для цього використовуються CSS-стилі для мобільних пристроїв, що дозволяє коректно відображати систему на смартфонах, планшетах та персональних комп’ютерах. На мобільних пристроях змінюється розташування елементів навігації та меню для більш зручного використання системи.

Для реалізації окремих допоміжних функцій використовується JavaScript, зокрема для зміни теми оформлення інтерфейсу. Також у проєкті використовуються іконки Boxicons та шрифти Google Fonts для покращення зовнішнього вигляду вебзастосунку.

Обмін даними із серверною частиною здійснюється через HTTP запити у форматі JSON. Клієнтська частина не взаємодіє з базою даних напряму, а отримує необхідну інформацію лише через API серверної частини.

Перевагою використання Blazor WebAssembly є можливість використовувати одну мову програмування для клієнтської та серверної частин системи [7]. Це спрощує процес розробки та підтримки застосунку. Крім того, компонентний підхід дозволяє створювати більш структурований та зрозумілий інтерфейс.

До недоліків можна віднести те, що Blazor WebAssembly може використовувати більше ресурсів браузера у порівнянні зі звичайними вебсторінками.

2.1.2 Архітектура серверної частини

Серверна частина вебзастосунку реалізована за допомогою ASP.NET Core API на платформі .NET 9. Вона виконує роль центрального компонента системи та відповідає за обробку запитів від клієнтської частини, виконання бізнес-логіки та взаємодію з базою даних.

Основною функцією серверної частини є прийом HTTP запитів від клієнтського застосунку, обробка отриманих даних та повернення результату

у форматі JSON. Обмін даними між клієнтом та сервером здійснюється через REST API, що дозволяє забезпечити стабільну взаємодію між різними компонентами системи. Серверна частина забезпечує роботу основного функціоналу системи.

Для взаємодії з базою даних використовується Entity Framework Core та провайдер Pomelo для MySQL. ORM-підхід дозволяє працювати з даними через об’єкти мови C#, що спрощує реалізацію CRUD-операцій та зменшує кількість SQL-коду у проєкті.

У якості системи керування базами даних використовується MySQL [8]. У базі даних зберігається інформація про користувачів, квартири, заявки, оголошення, платежі та інші сутності системи. Для забезпечення цілісності даних використовуються зв’язки між таблицями та зовнішні ключі.

Архітектура серверної частини побудована таким чином, щоб розділити логіку обробки запитів та роботу з даними. API відповідає за прийом і повернення HTTP запитів, а взаємодія з базою даних виконується через окремий шар доступу до даних. Такий підхід спрощує підтримку проєкту та робить структуру серверної частини більш зрозумілою.

Під час розробки використовується OpenAPI, який дозволяє автоматично створювати документацію для API та тестувати серверні маршрути. Для забезпечення взаємодії між клієнтською та серверною частинами також налаштована підтримка CORS.

Перевагою ASP.NET Core є висока продуктивність, підтримка сучасної клієнт-серверної архітектури та наявність великої кількості готових інструментів для роботи з API та базами даних. Використання Entity Framework Core значно спрощує роботу з базою даних та прискорює процес розробки.

До недоліків можна віднести необхідність додаткового налаштування взаємодії між клієнтською та серверною частинами системи. Крім того, використання ORM-технологій може впливати на продуктивність при виконанні складних SQL-запитів у великих обсягах даних.

2.2 Проєктування бази даних

На рисунку 2.2 наведено ER-модель – структури бази даних osbb_app, реалізованої в СУБД MySQL [8]. Модель відображає сутності предметної області обліку ОСББ, їхні атрибути та зв’язки між ними. Імена сутностей і атрибутів на діаграмі повністю відповідають іменам таблиць і полів у проєкті. Первинні ключі позначені як PK, зовнішні – як FK.

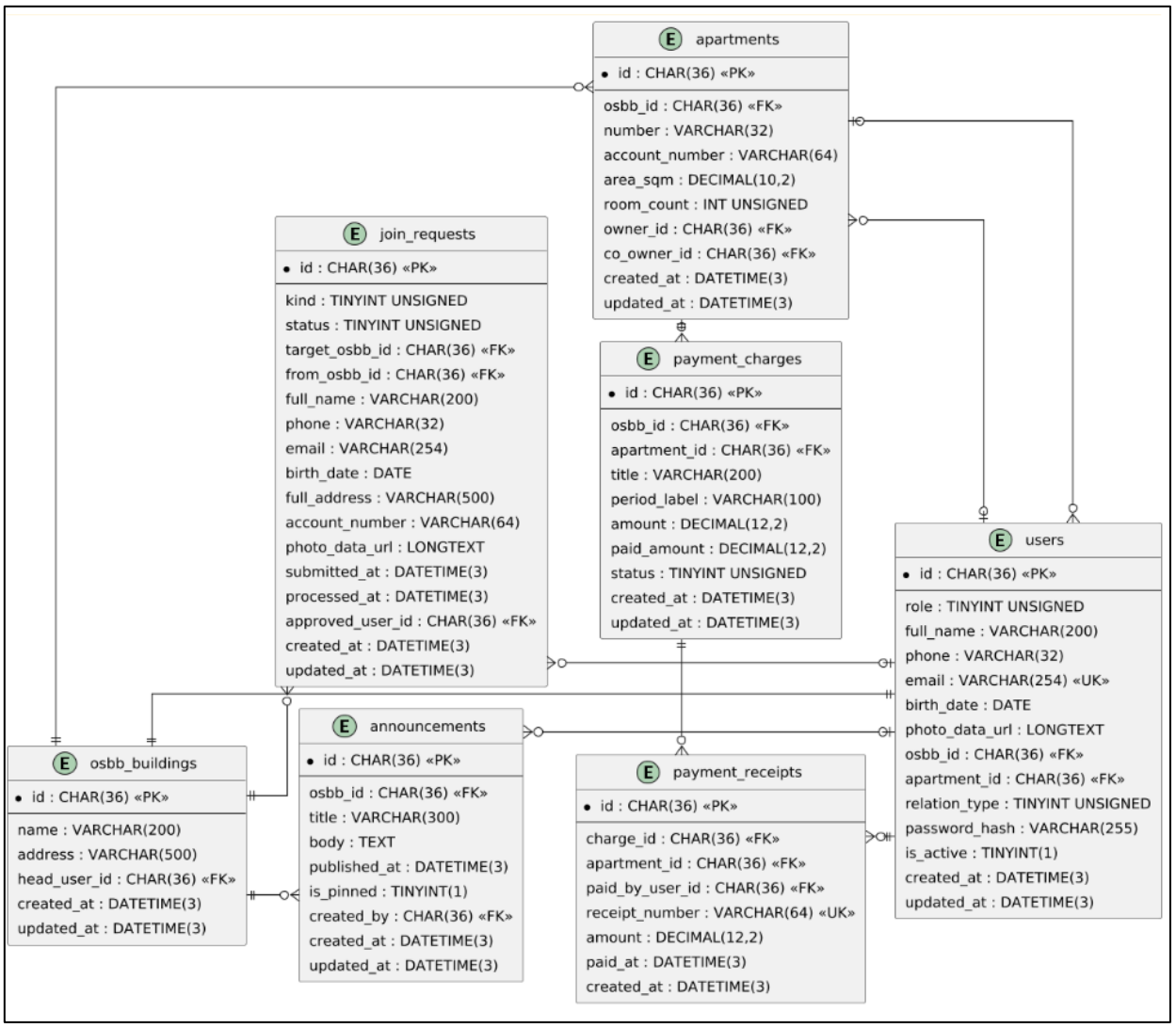


Рисунок 2.2 – ER-модель бази даних

Наведена ER-модель [2] повністю описує структуру бази даних розробленої системи та відображає основні бізнес-процеси: облік ОСББ і квартир, реєстрацію мешканців, обробку заявок, публікацію оголошень,

нарахування та фіксацію оплат. Модель є основою для фізичної реалізації в MySQL та для доступу до даних із серверної частини застосунку через Entity Framework Core.

2.2.1 Сутності бази даних

У базі даних інформаційної системи управління діяльністю ОСББ виділено сім основних взаємопов’язаних сутностей.

Сутність **osbb_buildings** призначена для зберігання інформації про об’єднання співвласників багатоквартирного будинку. Для кожного ОСББ зберігаються назва, адреса та посилання на голову правління. Дана сутність є центральною в моделі даних, оскільки з нею пов’язані квартири, користувачі, оголошення, заявки на вступ та нарахування.

Сутність **users** містить інформацію про користувачів системи, зокрема голову ОСББ та мешканців будинку. Для кожного користувача зберігаються персональні дані, контактна інформація, роль у системі, тип відношення до квартири, а також посилання на відповідне ОСББ та квартиру.

Сутність **apartments** використовується для обліку квартир у межах конкретного ОСББ. Вона містить номер квартири, особовий рахунок, площу, кількість кімнат та посилання на власника і співвласника квартири.

Сутність **join_requests** призначена для зберігання заявок на заселення або переїзд між ОСББ. У таблиці фіксуються тип заявки, її поточний статус, персональні дані заявника, цільове ОСББ, а також інформація про попереднє ОСББ у разі переїзду. Після схвалення заявки вона пов’язується зі створеним користувачем системи.

Сутність **announcements** містить оголошення, які публікуються головою ОСББ для мешканців будинку. Для кожного оголошення зберігаються заголовок, текст повідомлення, дата публікації, ознака закріплення та інформація про автора.

Сутність **payment_charges** використовується для обліку нарахувань за житлово-комунальні послуги. Вона містить назву нарахування, розрахунковий період, суму до сплати, суму фактично сплачених коштів та поточний статус оплати.

Сутність **payment_receipts** призначена для зберігання інформації про виконані платежі. Для кожної квитанції фіксуються номер квитанції, сума оплати, дата платежу, користувач, який здійснив оплату, а також нарахування, до якого належить платіж.

Для забезпечення аудиту змін більшість таблиць містять службові поля **created_at** та **updated_at**, які дозволяють відстежувати дату створення та останнього оновлення записів. У таблиці **payment_receipts** використовується лише поле **created_at**, оскільки записи про здійснені платежі після створення не змінюються.

2.2.2 Зв’язки та кратність

Між сутностями встановлено такі основні зв’язки, вони зазначені в таблиці 2.1.

Таблиця 2.1 – Зв’язки та кратності бази даних

Зв’язок	**Кратність**	**Зміст**
osbb_buildings – apartments	1 to many	Одне ОСББ об’єднує багато квартир
osbb_buildings – users (голова)	1 to 1	У кожного ОСББ один голова (head_user_id)
users – osbb_buildings	many to 1	Багато користувачів можуть належати одному ОСББ
users – apartments	many to 1	Мешканець прив’язаний до однієї квартири
apartments – users (власник)	many to 1	Квартира має одного власника (owner_id)
osbb_buildings – announcements	1 to many	Одне ОСББ може містити багато оголошень

Продовження таблиці 2.1

Зв’язок	**Кратність**	**Зміст**
users – announcements	1 to many	Один користувач може створити багато оголошень
osbb_buildings – payment_charges	1 to many	По одній квартирі може бути багато квитанцій
apartments – payment_charges	1 to many	По ОСББ формується багато нарахувань
apartments – payment_receipts	1 to many	По одній квартирі багато нарахувань
users – payment_receipts	1 to many	Один користувач може здійснити кілька оплат
payment_charges – payment_receipts	1 to many	По одному нарахуванню може бути кілька квитанцій
osbb_buildings – join_requests	1 to many	До одного ОСББ надходить багато заявок
users – join_requests	1 to many	Після схвалення заявки створюється або прив’язується один користувач

Зв’язки між таблицями закріплено зовнішніми ключами, що забезпечує цілісність даних на рівні СУБД: неможливо створити квартиру без існуючого ОСББ, нарахування без квартири, квитанцію без нарахування тощо. Первинні ключі сутностей мають тип CHAR(36), що узгоджено з генерацією ідентифікаторів у програмній реалізації системи.

2.3 UML діаграми

2.3.1 UML діаграма використання

На рисунку 2.3 представлено діаграму варіантів використання (Use Case Diagram) [3] інформаційної системи обліку діяльності ОСББ. Дана діаграма демонструє основних користувачів системи та їх можливості взаємодії з функціоналом вебзастосунку.

Система передбачає три основні ролі користувачів: голова ОСББ, мешканець та адміністратор.

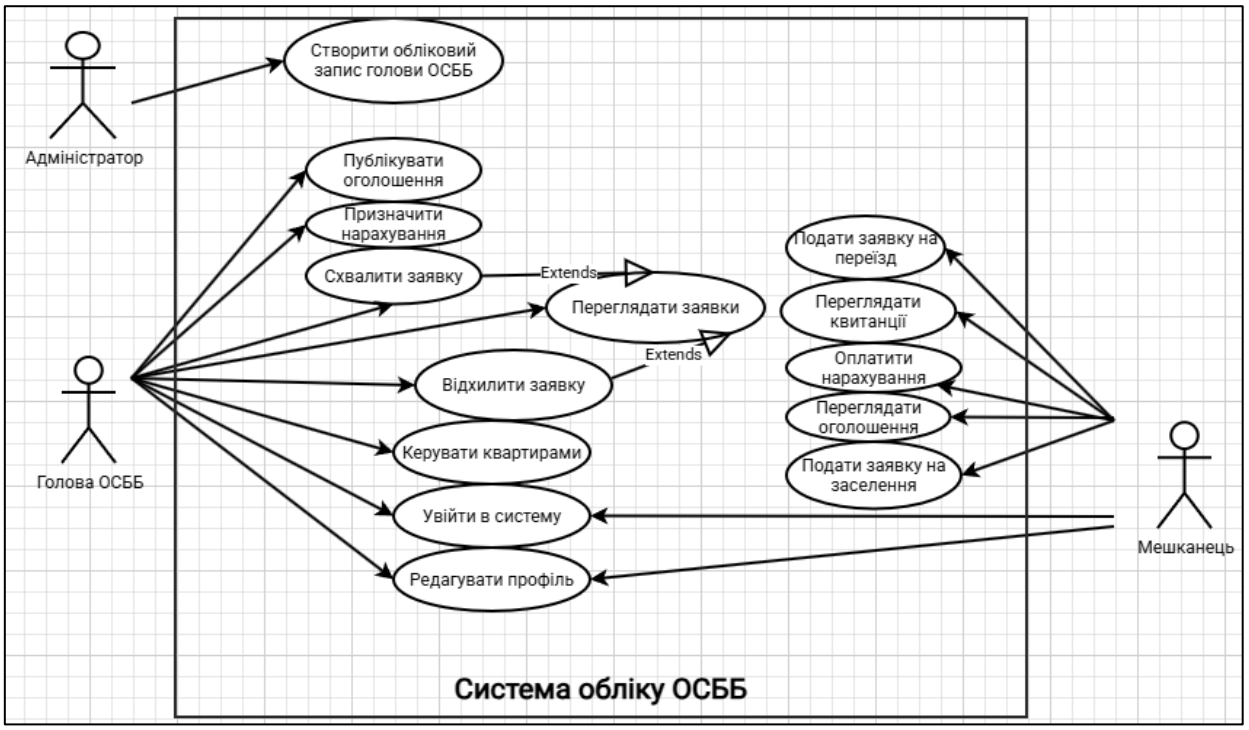


Рисунок 2.3 – Діаграма використання (Use Case Diagram)

Для кожної ролі визначено окремий набір функцій відповідно до прав доступу.

Користувач із роллю голови ОСББ має розширені можливості керування системою. До основних функцій належать:

– вхід у систему;
– керування квартирами;
– перегляд заявок;
– схвалення або відхилення заявок;
– призначення нарахувань;
– перегляд боржників;
– публікація оголошень.

Мешканець має доступ до користувацького функціоналу системи. Йому доступні такі можливості:

– реєстрація та вхід у систему;
– подання заявок на заселення;
– перегляд оголошень;
– оплата нарахувань;

– перегляд квитанцій;
– подання заявки на переїзд;
– редагування власного профілю.

Ці функції дозволяють мешканцю взаємодіяти з ОСББ у дистанційному форматі через вебзастосунок.

Адміністратор системи виконує функції початкового налаштування та керування обліковими записами. На діаграмі показано можливість створення облікового запису голови ОСББ.

На діаграмі також показано зв’язок розширення між прецедентами «Переглядати заявки», «Схвалити заявку» та «Відхилити заявку». Це означає, що операції схвалення або відхилення можуть виконуватися лише в контексті перегляду заявки та є додатковими сценаріями обробки відповідного запиту.

Таким чином, діаграма варіантів використання демонструє структуру взаємодії користувачів із системою та відображає основний функціонал, реалізований у вебзастосунку обліку діяльності ОСББ.

2.3.2 Діаграма активності процесу оплати нарахувань

На рисунку 2.4 представлено діаграму активності [3] процесу оплати нарахувань у системі обліку діяльності ОСББ. Діаграма демонструє послідовність дій мешканця та реакцію системи під час виконання.

Процес починається з входу мешканця до особистого кабінету. Після авторизації користувач відкриває розділ «платежі», обирає необхідне нарахування, вводить суму оплати та натискає кнопку «оплатити».

Після отримання запиту система виконує перевірку прав доступу користувача та завантажує інформацію про вибране нарахування з бази даних. Далі система перевіряє, чи не було нарахування вже повністю оплачене. Якщо заборгованість відсутня, користувачу відображається повідомлення про помилку, після чого процес завершується.

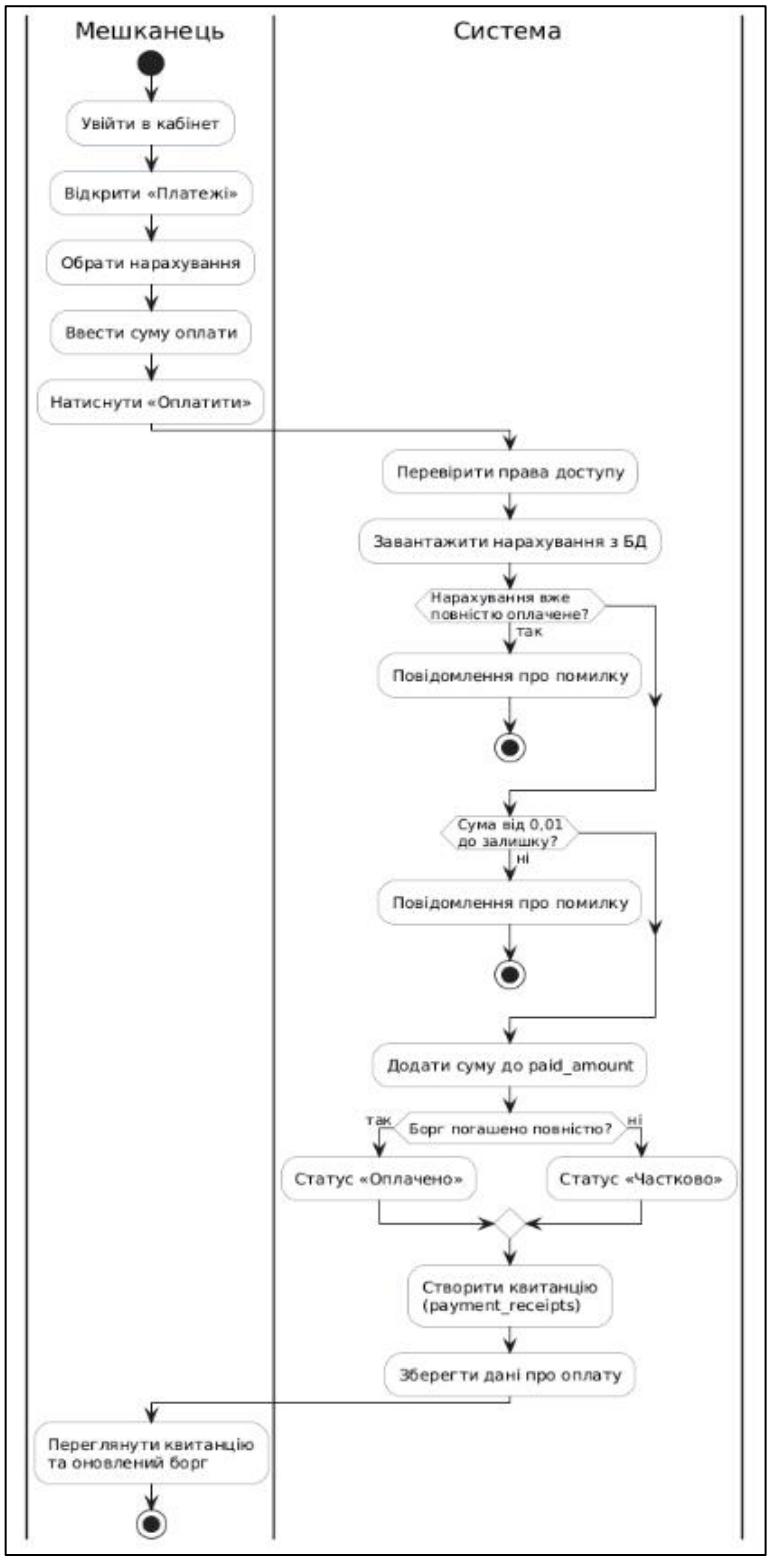


Рисунок 2.4 – Діаграма активності процесу оплати нарахувань (activity diagram)

У випадку, якщо введена сума є меншою за мінімально допустиме значення, система також формує повідомлення про помилку та припиняє виконання операції.

Після успішного проходження перевірок система додає введену суму до вже сплачених коштів (paid_amount) та визначає статус оплати. Якщо заборгованість погашено повністю, встановлюється статус «оплачено». В іншому випадку система встановлює статус «частково».

Після оновлення статусу система створює квитанцію про оплату (payment_receipts) та зберігає інформацію про проведений платіж у базі даних.

На завершальному етапі мешканець може переглянути оновлену інформацію про борг та сформовану квитанцію.

Таким чином, дана діаграма відображає логіку обробки платежів у системі, включаючи перевірку коректності даних, оновлення статусів нарахувань та формування квитанцій про оплату.

2.3.3 Діаграма активності послідовності процесу створення оголошення

На рисунку 2.5 представлено діаграму послідовності процесу створення оголошення в інформаційній системі обліку діяльності ОСББ. Діаграма демонструє взаємодію між головою ОСББ, клієнтською частиною системи на базі Blazor та серверною частиною під час публікації нового оголошення.

Процес починається з входу голови ОСББ до адміністративної панелі системи. Після авторизації користувач відкриває розділ «оголошення», де заповнює заголовок та текст нового оголошення. За потреби користувач може встановити позначку «закріпити», що дозволяє відображати оголошення у верхній частині списку. Після введення необхідної інформації голова ОСББ натискає кнопку «опублікувати».

Після цього клієнтська частина системи формує POST-запит до серверного API за адресою /api/announcements. Отримавши запит, сервер виконує перевірку прав доступу користувача для підтвердження можливості створення оголошення.

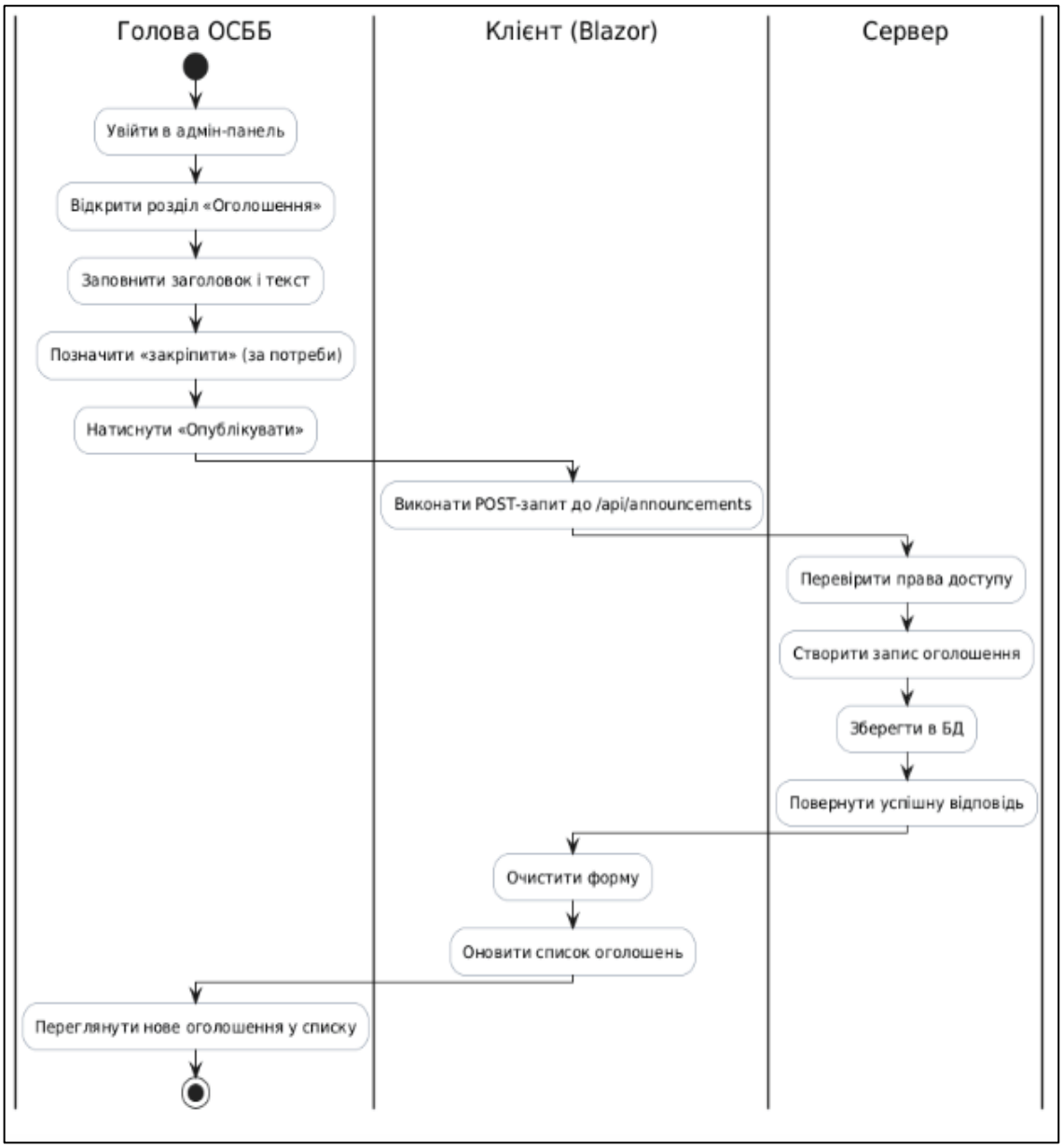


Рисунок 2.5 – Діаграма активності послідовності процесу створення оголошення (activity diagram)

У разі успішного проходження перевірки сервер створює новий запис оголошення та зберігає його у базі даних. Після завершення операції сервер повертає клієнтській частині повідомлення про успішне створення оголошення.

Отримавши відповідь від сервера, клієнтська частина очищує форму введення та оновлює список оголошень. У результаті нове оголошення відображається у загальному списку та стає доступним для перегляду мешканцями.

Таким чином, дана діаграма відображає повний процес створення та публікації оголошення в системі, включаючи взаємодію клієнтської та серверної частин, перевірку прав доступу та збереження даних у базі даних.

2.3.4 Діаграма класів

Діаграма класів (див. рис. 2.6) відображає логічну структуру інформаційної системи автоматизації діяльності ОСББ та демонструє основні сутності предметної області, їх атрибути і взаємозв’язки.

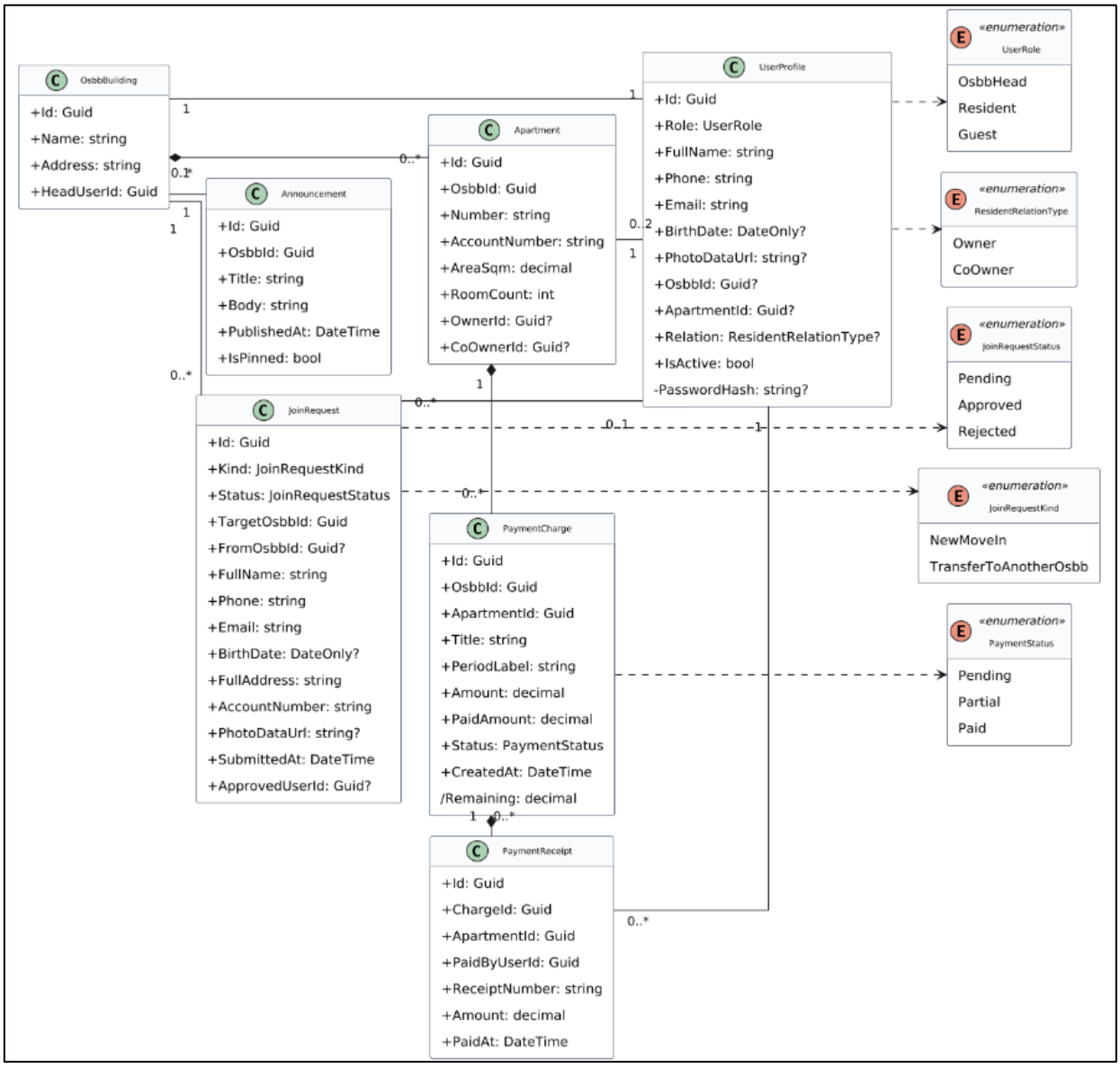


Рисунок 2.6 – Діаграма класів

Діаграма побудована відповідно до нотації UML [3] та використовується для опису внутрішньої структури програмної системи на етапі проєктування.

Логічна модель відповідає структурі класів, що використовуються у клієнтській та серверній частинах застосунку.

Центральною сутністю моделі є клас **OsbbBuilding**, який представляє об’єднання співвласників багатоквартирного будинку. Даний клас містить основну інформацію про ОСББ та пов’язаний із квартирами, оголошеннями і заявками мешканців. Також із конкретним ОСББ пов’язується профіль голови правління, який здійснює управління будинком та має розширені права доступу в системі.

Для зберігання інформації про користувачів використовується клас **UserProfile**. Він містить персональні дані користувача, його роль у системі, а також інформацію про належність до ОСББ та квартири. Ролі користувачів визначаються за допомогою переліку **UserRole**, який містить значення **OsbbHead**, **Resident** та **Guest**. Після реєстрації користувач отримує роль Guest, а після схвалення заявки головою ОСББ його профіль оновлюється до ролі Resident із прив’язкою до відповідного ОСББ та квартири.

Клас **Apartment** описує квартиру, що входить до складу ОСББ. Для кожної квартири зберігаються номер, особовий рахунок, площа, кількість кімнат та інформація про власника або співвласника. Одне ОСББ може містити багато квартир, тоді як кожна квартира належить лише одному ОСББ. До квартири можуть бути прив’язані власник та співвласник, які представлені користувачами системи.

Процес приєднання мешканців до ОСББ реалізовано за допомогою класу **JoinRequest**. У ньому зберігається інформація про подану заявку, її тип та поточний статус. Для визначення виду заявки використовується перелік **JoinRequestKind**, який містить значення заселення або переїзду між ОСББ. Стан обробки заявки визначається за допомогою переліку **JoinRequestStatus**, що містить статуси очікування, схвалення або відхилення. Після схвалення заявки вона пов’язується з відповідним профілем користувача.

Інформування мешканців реалізовано за допомогою класу **Announcement**, який містить заголовок, текст повідомлення, дату публікації та ознаку закріплення оголошення. Одне ОСББ може створювати необмежену кількість оголошень для своїх мешканців.

Функції фінансового обліку реалізуються класами **PaymentCharge** та **PaymentReceipt**. Клас **PaymentCharge** містить інформацію про нарахування платежів за визначений розрахунковий період, суму нарахування, суму сплачених коштів та поточний статус оплати. Статус платежу визначається переліком **PaymentStatus**, який дозволяє відстежувати стан заборгованості. Також клас містить обчислюване поле залишку боргу, що визначається як різниця між нарахованою та сплаченою сумою. Клас **PaymentReceipt** використовується для зберігання інформації про виконані платежі та сформовані квитанції. Одному нарахуванню може відповідати декілька квитанцій, що забезпечує підтримку часткової оплати.

Між класами встановлено асоціативні зв’язки з відповідними кратностями. Одне ОСББ може містити багато квартир, оголошень та заявок мешканців. Квартира може мати одного власника та одного співвласника. Для кожної квартири може бути сформовано багато нарахувань, а кожне нарахування може містити декілька записів про оплату. Квитанція про оплату також пов’язується з користувачем, який здійснив платіж. Такі зв’язки відображають бізнес-логіку предметної області та забезпечують цілісність даних у системі.

Таким чином, діаграма класів описує структуру інформаційної системи управління ОСББ, визначає основні сутності предметної області, їх атрибути та взаємозв’язки. Побудована модель є основою для реалізації програмного забезпечення, формування структури бази даних та забезпечення взаємодії між клієнтською і серверною частинами системи.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ОСББ

3.1 Загальна структура серверної частини

Серверна частина інформаційної системи управління діяльністю ОСББ реалізована на платформі ASP.NET Core 9 у вигляді REST API з використанням підходу minimal api. Основним призначенням серверної частини є обробка HTTP-запитів від клієнтського застосунку, виконання бізнес-логіки системи та взаємодія з базою даних MySQL.

Архітектура серверної частини побудована за багаторівневим принципом, що дозволяє розділити логіку маршрутизації, обробки даних та доступу до бази даних. Такий підхід спрощує підтримку програмного коду, забезпечує зрозумілу структуру проєкту та дозволяє розширювати функціональні можливості системи без суттєвих змін у вже реалізованих компонентах.

Серверна частина складається з таких основних компонентів:

– Program.cs – точка входу застосунку, у якій виконується налаштування сервісів, маршрутизації, автентифікації, авторизації та HTTP-конвеєра;
– REST API маршрути – набір ендпоінтів для взаємодії клієнтської частини із сервером через HTTP-запити;
– Authservice – сервіс автентифікації та авторизації користувачів, який відповідає за реєстрацію, перевірку паролів і формування JWT-токенів;
– Osbbrepository – сервіс доступу до даних та реалізації основної бізнес-логіки системи;
– Osbbdbcontext – контекст Entity Framework Core для взаємодії з базою даних MySQL;
– база даних MySQL – централізоване сховище інформації про ОСББ, користувачів, квартири, нарахування, платежі та оголошення.

Клієнтська частина надсилає HTTP-запити до REST API у форматі JSON. Після отримання запиту сервер виконує перевірку прав доступу користувача, обробляє бізнес-логіку та звертається до бази даних через Entity Framework Core. Після завершення операції результат повертається клієнтській частині у форматі JSON.

Для реалізації серверної частини використовується підхід minimal api, який дозволяє описувати маршрути без використання класичних MVC-контролерів. Це зменшує кількість шаблонного коду та робить структуру api більш компактною і зрозумілою.

У серверній частині також реалізовано механізм JWT-автентифікації. Після успішного входу користувач отримує токен доступу, який надалі використовується для виконання захищених операцій системи. Розмежування прав доступу здійснюється на основі ролей користувачів, зокрема ролей голови осбб та мешканця.

Окремий вебсервер одночасно обслуговує серверну частину API та статичні файли клієнтського застосунку Blazor WebAssembly. Завдяки цьому вебзастосунок може працювати через один мережевий порт без необхідності окремого розгортання frontend та backend частин системи.

3.1.1 Взаємодія клієнта та сервера

Клієнтська та серверна частини працюють окремо, але разом: у браузері виконується вебінтерфейс (Blazor WebAssembly), на сервері обробка запитів і робота з базою даних. Між ними немає прямого доступу до MySQL з браузера – усі дані проходять через мережевий обмін за єдиними правилами.

Зв’язок побудовано на протоколі HyperText Transfer Protocol. Клієнт надсилає запит, сервер повертає відповідь. Для отримання даних використовують методи GET, для створення або зміни – POST і PUT. У режимі розробки та тестування в локальній мережі застосовується HTTP; для реальної експлуатації в Інтернеті доцільно увімкнути HTTPS шифрований канал.

Тіло запитів і відповідей передається у форматі JSON – текстовий формат, зручний для вебзастосунків. Наприклад, список квартир або дані про нарахування серіалізуються з об’єктів C# у JSON і навпаки. Це забезпечує однакове розуміння структури даних клієнтом і сервером.

Для захисту API передбачено механізм JSON Web Token [4]. Після успішного входу (логін і пароль) сервер видає токен – закодований рядок, у якому міститься інформація про користувача (наприклад, ідентифікатор і роль). Клієнт зберігає токен і надсилає його в заголовку кожного наступного запиту:

Authorization: Bearer <токен>.

Сервер перевіряє підпис і термін дії токена; лише після цього виконує операцію. У таблиці users є поле password_hash для зберігання хеша пароля, що відповідає підходу до безпечної автентифікації.

Архітектура системи дозволяє підключити повноцінну авторизацію через JWT у ASP.NET Core без зміни структури REST API.

Основні маршрути серверної частини наведено в таблиці 3.1.

Таблиця 3.1 – Маршрути REST API системи обліку ОСББ

Метод	**Маршрут**	**Призначення**
GET	/api/health	перевірка доступності сервера та БД
GET	/api/osbbs	список ОСББ
GET	/api/osbbs/{id}	ОСББ за ідентифікатором
GET	/api/users/head	профіль голови ОСББ
GET	/api/users/{id}	профіль користувача
PUT	/api/users/{id}	оновлення профілю
GET	/api/apartments?osbbId=...	квартири ОСББ
GET	/api/apartments/{id}	квартира за id
POST	/api/apartments	додати / оновити квартиру
GET	/api/join-requests?osbbId=...	заявки на вступ

Продовження таблиці 3.1

Метод	**Маршрут**	**Призначення**
GET	/api/join-requests/pending-count?osbbId=...	кількість очікуючих заявок
POST	/api/join-requests	подати заявку
POST	/api/join-requests/{id}/approve	схвалити заявку
POST	/api/join-requests/{id}/reject	відхилити заявку
GET	/api/announcements?osbbId=...	оголошення
POST	/api/announcements	додати оголошення
GET	/api/charges?apartmentId=...	нарахування по квартирі
POST	/api/charges/assign-all	призначити нарахування всім квартирам
POST	/api/charges/{id}/pay	оплата нарахування
GET	/api/receipts?apartmentId=...	квитанції по квартирі
GET	/api/receipts/{id}	квитанція за id
GET	/api/debtors?osbbId=...	список боржників

3.1.2 Реалізація бази даних

Для зберігання даних обрано реляційну MySQL 8.0: зовнішні ключі не допускають «висячих» записів, а представлення (views) дозволяють формувати звіти без дублювання складної логіки в інтерфейсі. База osbb_app працює в кодуванні utf8mb4, що необхідно для коректного зберігання українських текстів у ПІБ, адресах та оголошеннях.

Фізична схема задається SQLскриптами каталогу database/mysql/ (див. рис. 3.1).

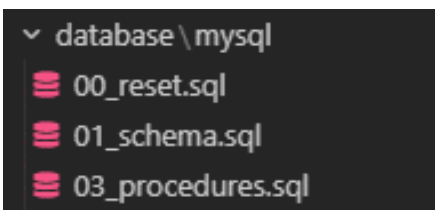


Рисунок 3.1 – Структура SQL-скриптів бази даних

Такий підхід обрано тому, що структуру БД можна відтворити на будь-якому комп’ютері незалежно від запуску застосунку. Первинне розгортання виконується послідовно:

– 03_reset.sql створює базу (див. додаток А.1);
– 01_schema.sql сім таблиць (імена збігаються з сутностями на ER-діаграмі), зовнішні ключі, індекси та представлення v_debtors і v_osbb_summary (див. додаток А.2);
– за потреби підключається 02_procedures.sql із збереженими процедурами (див. додаток А.3).

У таблицях реалізовано ключові обмеження проєкту: первинні ключі типу CHAR(36) узгоджені з генерацією ідентифікаторів у застосунку; унікальна пара (osbb_id, account_number) у apartments запобігає дублюванню особових рахунків; у payment_charges зберігаються сума нарахування (amount), накопичена сплачена сума (paid_amount) та статус; у payment_receipts на одне нарахування може припадати кілька записів – індекс по charge_id не є унікальним. Це узгоджено зі зв’язком 1 to many між нарахуванням і квитанціями на ER-моделі.

3.1.3 Реалізація розрахунку платежів (нарахування)

Однією з основних функцій інформаційної системи управління ОСББ є автоматизоване масове призначення нарахувань для всіх квартир будинку. Реалізація даного механізму виконується методом AssignChargeToAllAsync у класі OsbbRepository (див. рис. 3.2).

Метод забезпечує автоматичне створення записів нарахувань для кожної квартири ОСББ із урахуванням площі квартири, тарифу утримання та додаткових фіксованих платежів. Такий підхід дозволяє уникнути ручного створення платежів та автоматизувати процес щомісячного нарахування комунальних внесків.

```
  public async Task AssignChargeToAllAsync(AssignChargeRequest request,
CancellationToken ct = default)
  {
    var apartments = await db.Apartments.Where(a => a.OsbbId ==
request.OsbbId).ToListAsync(ct);
    foreach (var apt in apartments)
    {
      var amount = decimal.Round(
        (apt.AreaSqm * request.RatePerSqm) + request.CapRepairAmount +
request.GarbageAmount,
        2,
        MidpointRounding.AwayFromZero);

      var title =
        $"Утримання: {request.RatePerSqm:N2} ₴/м²"
        + (request.CapRepairAmount > 0 ? $" · Капремонт: {request.CapRepairAmount:N2}
₴" : "")
        + (request.GarbageAmount > 0 ? $" · ТПВ: {request.GarbageAmount:N2} ₴" : "");

      db.PaymentCharges.Add(new PaymentCharge
      {
        Id = Guid.NewGuid(),
        OsbbId = request.OsbbId,
        ApartmentId = apt.Id,
        Title = title,
        PeriodLabel = request.PeriodLabel,
        Amount = amount,
        PaidAmount = 0,
        Status = PaymentStatus.Pending,
        CreatedAt = DateTime.Now
      });
    }
    await db.SaveChangesAsync(ct);
  }
```

Рисунок 3.2 – Реалізація розрахунку платежів (нарахування)

Для формування нарахувань використовується об’єкт AssignChargeRequest, який містить такі параметри:

– OsbbId – ідентифікатор ОСББ;

– PeriodLabel – мітка розрахункового періоду;

– RatePerSqm – тариф утримання у гривнях за квадратний метр;

– CapRepairAmount – фіксований внесок на капітальний ремонт;

– GarbageAmount – фіксований платіж за вивіз твердих побутових відходів.

Алгоритм роботи полягає у проходженні по всіх квартирах, які належать відповідному ОСББ, та обчисленні індивідуальної суми нарахування для кожної квартири.

Формула розрахунку суми платежу:

$$S=round(area_sqm \times RatePerSqm+CapRepairAmount+GarbageAmount,2),$$

де:

– *area_sqm* – площа квартири у квадратних метрах;
– *RatePerSqm* – тариф за 1 м²;
– *CapRepairAmount* – внесок на капітальний ремонт;
– *GarbageAmount* – оплата ТПВ;
– *S* – підсумкова сума нарахування.

Округлення результату виконується методом MidpointRounding.AwayFromZero, що відповідає математичному округленню до двох знаків після коми.

Наприклад, для квартири площею 62,50 м² при тарифі 8,00 грн/м², внеску на капітальний ремонт 150 грн та оплаті ТПВ 80 грн підсумкова сума нарахування становитиме:

$$S=62,50 \times 8+150+80=500,00+150+80=730 \text{ грн}.$$

Для квартири площею 45 м² за аналогічних умов сума платежу буде:

$$S = 45 \times 8+230=590 \text{ грн}.$$

Таким чином, змінною частиною платежу є тариф утримання залежно від площі квартири, тоді як внески на капітальний ремонт та ТПВ залишаються однаковими для всіх мешканців у межах одного нарахування.

Після обчислення суми система створює запис у таблиці charges (payment_charges), де автоматично заповнюються:

– amount – сума нарахування;
– paid_amount – початково 0;
– status – статус Pending;
– title – автоматично сформований опис платежу.

Для забезпечення узгодженості бізнес-логіки аналогічний механізм реалізовано також на рівні СКБД у вигляді збереженої процедури. Проте у вебзастосунку використовується розширена формула з урахуванням площі квартири, що відповідає реальній практиці нарахувань у ОСББ.

Реалізований механізм дозволяє автоматизувати процес формування платежів, мінімізувати кількість помилок при ручному введенні даних та забезпечити коректний облік заборгованостей мешканців.

3.1.4 Реалізація оплати та формування квитанцій

У системі реалізовано механізм оплати нарахувань, який підтримує як повну, так і часткову оплату заборгованостей мешканців ОСББ. Основна бізнес-логіка виконання платежів реалізована методом PayChargeAsync у класі OsbbRepository.

Метод забезпечує перевірку коректності платежу, оновлення інформації про стан нарахування та автоматичне формування квитанції про оплату.

На першому етапі система завантажує запис нарахування з таблиці payment_charges за ідентифікатором chargeId. Якщо нарахування не знайдено, система повертає повідомлення про помилку. Далі виконується перевірка, чи не було нарахування вже повністю сплачене.

Після цього система обчислює залишок заборгованості:

$$remaining = amount - paid_amount.$$

Далі перевіряється коректність введеної суми оплати. Значення платежу повинно бути більшим за нуль та не перевищувати поточний залишок боргу.

Після успішного проходження перевірок система виконує округлення суми до двох знаків після коми за допомогою MidpointRounding.AwayFromZero та оновлює поле paid_amount.

Статус нарахування визначається автоматично:

– Partial – якщо сплачено лише частину суми;

– Paid – якщо заборгованість погашена повністю.

Формально оновлення статусу можна подати таким чином:

$$paid_amount \geq amount,\ status = Paid.$$

У випадку часткової оплати:

$$paid_amount < amount,\ status = Partial.$$

Після оновлення інформації про нарахування система створює запис у таблиці payment_receipts. Для кожної оплати автоматично генерується унікальний номер квитанції у форматі:

KV-YYYYMMDD-XXXX.

де:

– YYYYMMDD – поточна дата;

– XXXX – випадкове чотиризначне число.

Приклад роботи механізму оплати: нехай сума нарахування становить 730,00 грн. Мешканець виконує першу оплату у розмірі 300,00 грн. Після проведення операції система встановлює:

– paid_amount = 300,00;

– status = Partial.

Також створюється квитанція, наприклад:

KV-20260529-4521.

Після другої оплати у розмірі 430,00 грн загальна сума оплати становитиме 730,00 грн, а статус нарахування автоматично зміниться на Paid.

Таким чином, одне нарахування може містити кілька квитанцій про оплату, що відповідає реальній практиці роботи ОСББ та дозволяє реалізувати підтримку часткових платежів.

```
  public async Task<PaymentReceipt> PayChargeAsync(Guid chargeId, Guid userId, decimal
amount, CancellationToken ct = default)
  {
    var charge = await db.PaymentCharges.FirstOrDefaultAsync(c => c.Id == chargeId, ct)
      ?? throw new InvalidOperationException("Нарахування не знайдено");

    if (charge.PaidAmount >= charge.Amount)
      throw new InvalidOperationException("Нарахування вже повністю оплачене");

    var remaining = charge.Amount - charge.PaidAmount;
    if (amount <= 0 || amount > remaining)
      throw new InvalidOperationException($"Сума оплати має бути від 0,01 до
{remaining:N2} грн");

    amount = decimal.Round(amount, 2, MidpointRounding.AwayFromZero);
    charge.PaidAmount += amount;
    charge.Status = charge.PaidAmount >= charge.Amount
      ? PaymentStatus.Paid
      : PaymentStatus.Partial;

    var receipt = new PaymentReceipt
    {
      Id = Guid.NewGuid(),
      ChargeId = charge.Id,
      ApartmentId = charge.ApartmentId,
      PaidByUserId = userId,
      ReceiptNumber = $"KV-{DateTime.Now:yyyyMMdd}-{Random.Shared.Next(1000,
9999)}",
      Amount = amount,
      PaidAt = DateTime.Now
    };

    db.PaymentReceipts.Add(receipt);
    await db.SaveChangesAsync(ct);
    return receipt;
  }
```

Рисунок 3.3 – Реалізація оплати та формування квитанцій

Реалізований механізм дозволяє вести облік усіх платежів мешканців, контролювати залишок заборгованості та автоматично формувати квитанції про оплату.

3.2 Реалізація клієнтської частини та основні функції вебзастоснку

Клієнтська частина інформаційної системи управління ОСББ реалізована на платформі Blazor WebAssembly у вигляді односторінкового вебзастосунку (SPA).

На відміну від JavaScript-фреймворків React або Vue, Blazor WebAssembly дозволяє реалізовувати клієнтську логіку мовою C# із використанням компонентів Razor (.razor), які поєднують HTML-розмітку та програмний код в одному файлі.

Після компіляції застосунок виконується безпосередньо у браузері користувача за допомогою технології WebAssembly. Це дозволяє реалізувати динамічний інтерфейс без постійного перезавантаження сторінок та забезпечує швидку взаємодію з REST API серверної частини.

Візуальне оформлення базується на власній дизайн-системі: семантичні кольори, типографіка (шрифт Plus Jakarta Sans), 8-піксельна сітка відступів, уніфіковані компоненти (картки, кнопки, поля форм, списки). Додаткові стилі для темної теми та публічних сторінок; правила мобільної адаптації.

Інтерфейс розділено на три зони за ролями користувача, які представлені у таблиці 3.2.

Таблиця 3.2 – Зони інтерфейсу

Зона	**Макет (Layout)**	**Призначення**
Публічна	MainLayout	Вхід, реєстрація, подача заявки
Кабінет мешканця	ResidentLayout	Огляд, оголошення, платежі, профіль
Кабінет голови ОСББ	AdminLayout	Квартири, заявки, нарахування, боржники

Навігація між сторінками здійснюється клієнтським маршрутизатором Blazor без повного перезавантаження сторінки.

Кожен layout виконує перевірку ролі користувача, формує структуру інтерфейсу та відображає навігаційні елементи.

Сторінки застосунку поділяються на публічні та захищені. До публічних належать сторінки входу, реєстрації та подання заявки на приєднання. Захищені сторінки доступні лише авторизованим користувачам відповідної ролі.

3.2.1 Головна сторінка

Головна сторінка є стартовою точкою роботи з системою та виконує функцію автентифікації користувача. На комп’ютері сторінка входу виконана у вигляді landing: зліва – інформаційний блок із назвою системи та коротким описом можливостей; справа – форма з полями «Email» і «Пароль», кнопка «Увійти» та посилання на реєстрацію мешканця. У шапці розміщено логотип і перемикач світлої/темної теми (див. рис. 3.4).

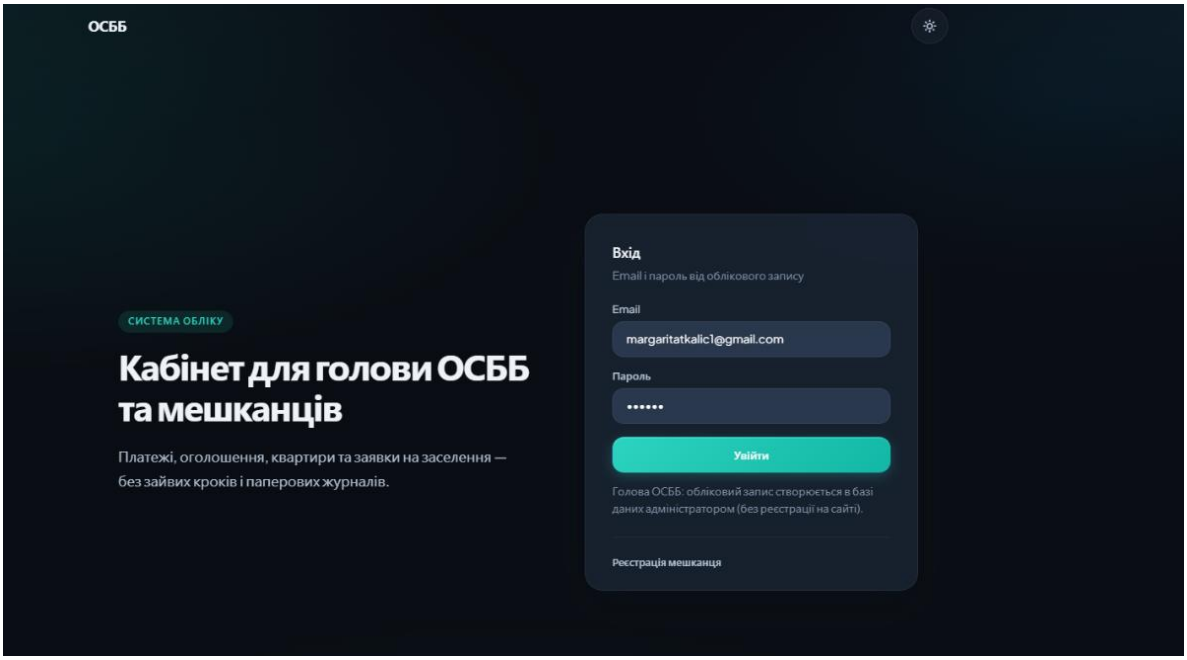


Рисунок 3.4 – Головна сторінка входу до системи (темна тема / ПК версія)

У системі реалізовано світлу тему оформлення інтерфейсу для мобільної версії (див. рис. 3.5). Вона побудована на світлій колірній палітрі з високою контрастністю тексту та елементів керування, що покращує читабельність. Користувач може змінювати оформлення через перемикач теми, при цьому обрана тема застосовується до всіх сторінок інтерфейсу.

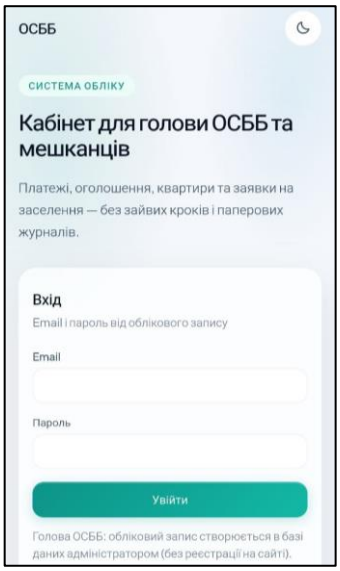


Рисунок 3.5 – Головна сторінка входу до системи мобільної версії

На смартфоні блоки розташовані в одну колонку: спочатку заголовок і опис, нижче – форма входу на всю ширину екрана. Кнопки та поля мають збільшений розмір (не менше 44 px) для зручного натискання.

3.2.2 Призначення нарахувань (голова ОСББ)

Для голови ОСББ у системі реалізовано окрему сторінку призначення нарахувань, яка дозволяє автоматизовано формувати платежі для всіх квартир будинку (див. рис. 3.6). На сторінці користувач може вказати: розрахунковий період; тариф за 1 м²; суму внеску на капітальний ремонт; оплату за ТПВ.

Після введення даних система автоматично формує підсумок нарахування та дозволяє виконати масове створення платежів для всіх квартир ОСББ за допомогою кнопки «Призначити нарахування».

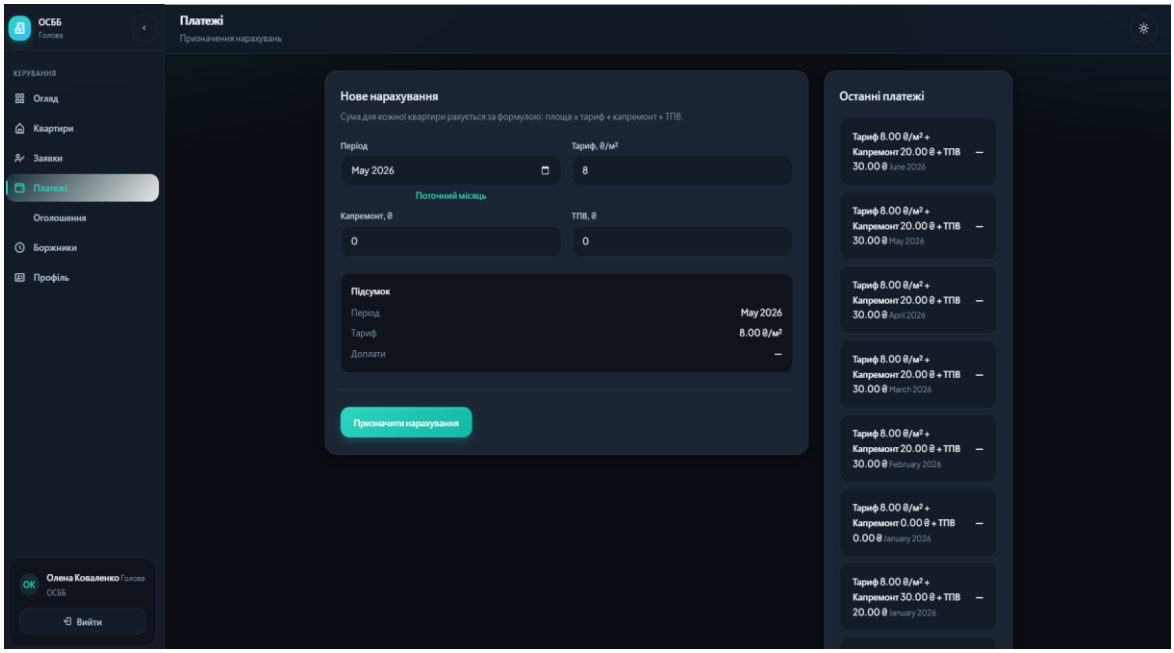


Рисунок 3.6 – Форма призначення нарахувань (ПК версія)

У правій частині сторінки відображається список останніх нарахувань із короткою інформацією про тариф, додаткові внески та період. Це дозволяє голові ОСББ контролювати історію створених платежів.

На рисунку 3.7 представлено мобільну версію сторінки призначення нарахувань у вебзастосунку управління діяльністю ОСББ.

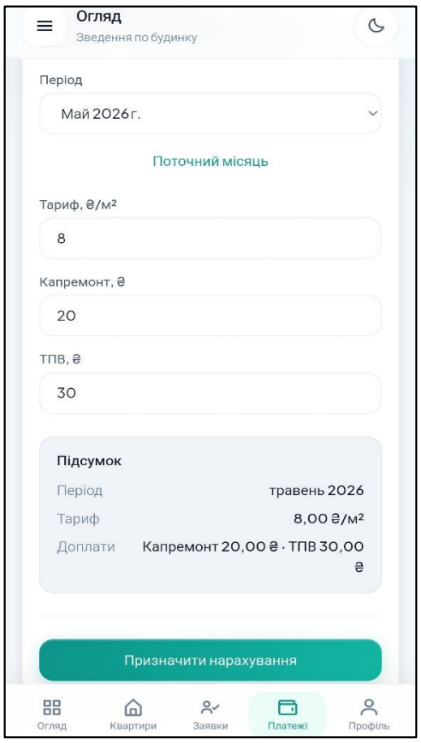


Рисунок 3.7 – Форма призначення нарахувань (мобільна версія)

Інтерфейс адаптований для роботи на смартфонах та забезпечує зручне введення параметрів нарахувань.

3.2.3 Публікація оголошень

Сторінка призначена для створення та керування оголошеннями для мешканців будинку (див. рис. 3.8).

Голова ОСББ може ввести заголовок і текст оголошення, за потреби позначити його як актуальне та опублікувати.

У правій частині сторінки відображається список раніше опублікованих оголошень із датою створення.

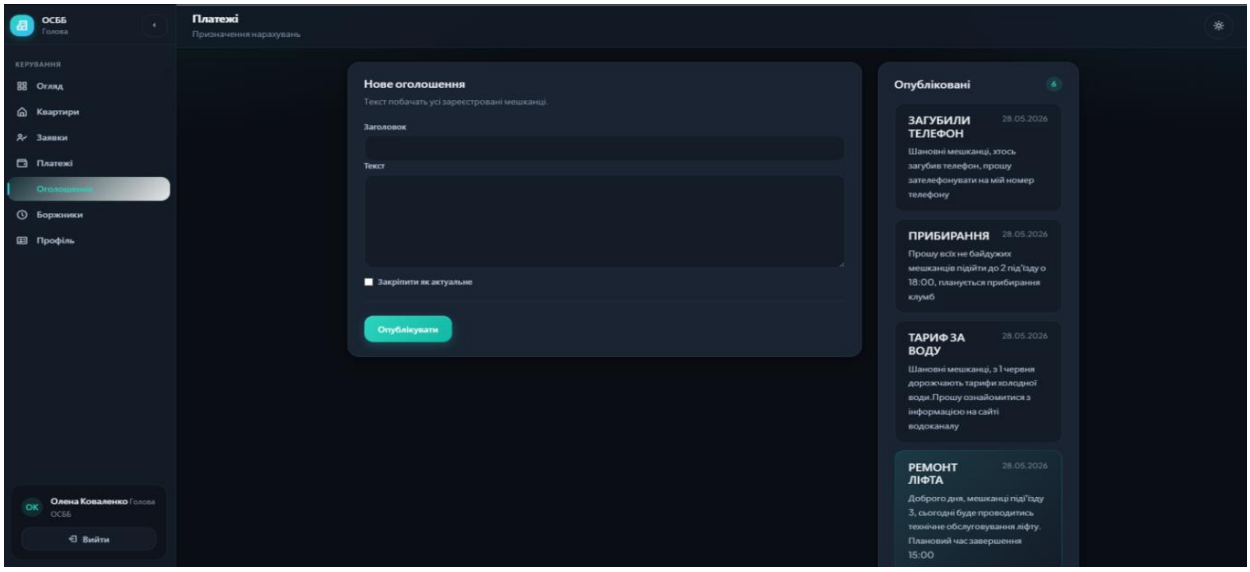


Рисунок 3.8 – Сторінка публікації оголошень (ПК версія)

У мобільній версії форма створення оголошення та список опублікованих оголошень розташовуються вертикально для зручного перегляду на невеликих екранах (див. рис. 3.9).

Користувач може швидко створювати нові оголошення, переглядати опубліковані повідомлення та виконувати всі основні дії без втрати функціональності.

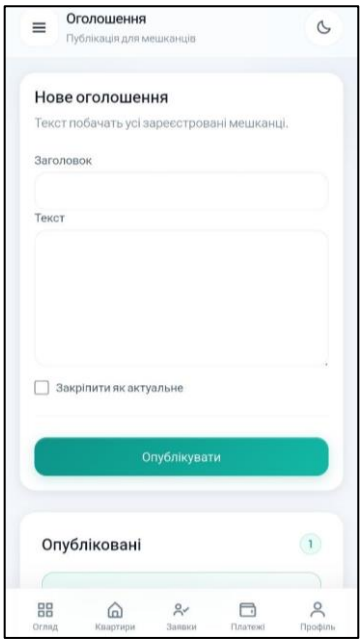


Рисунок 3.9 – Сторінка публікації оголошень (мобільна версія)

3.2.4 Сторінка подання заявки на заселення

Сторінка подання заявки на заселення призначена для введення персональних даних мешканця та надсилання заявки на розгляд голові ОСББ (див. рис. 3.10). Основну частину займає форма, що містить поля для вибору ОСББ, введення адреси, прізвища, імені та по батькові, номера телефону.

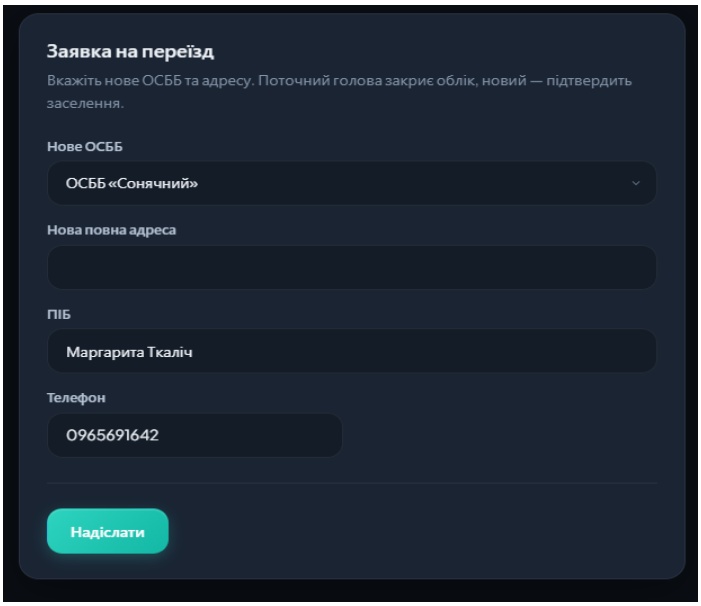


Рисунок 3.10 – Форма подання заявки на заселення (версія для ПК)

На комп’ютері елементи форми розташовані у декілька колонок, що забезпечує раціональне використання екранного простору та підвищує зручність введення даних.

У мобільній версії сторінки всі елементи форми розташовуються в один стовпець для забезпечення зручної роботи на пристроях з невеликою шириною екрана (див. рис. 3.11). Поля введення автоматично адаптуються до розмірів дисплея, що дозволяє комфортно взаємодіяти із системою за допомогою сенсорного керування.

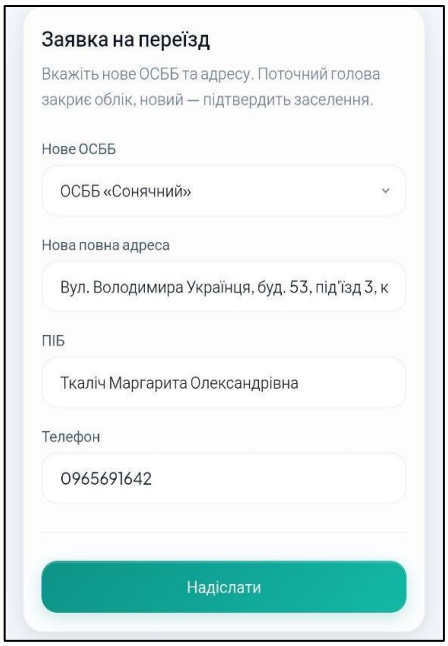


Рисунок 3.11 – Форма подання заявки на заселення (мобільна версія)

3.2.5 Сторінка обробки заявок мешканців

Сторінка обробки заявок призначена для перегляду та розгляду заявок на заселення і переїзд, поданих мешканцями. Голова ОСББ має можливість переглядати інформацію про заявника, включаючи його персональні дані, дату подання заявки та її поточний статус.

У версії для персонального комп’ютера сторінка містить бічне меню навігації та робочу область зі списком заявок. Інформація про кожну заявку

представлена у вигляді окремої картки, що містить дані заявника, тип заявки, дату подання та статус її розгляду (див. рис. 3.12).

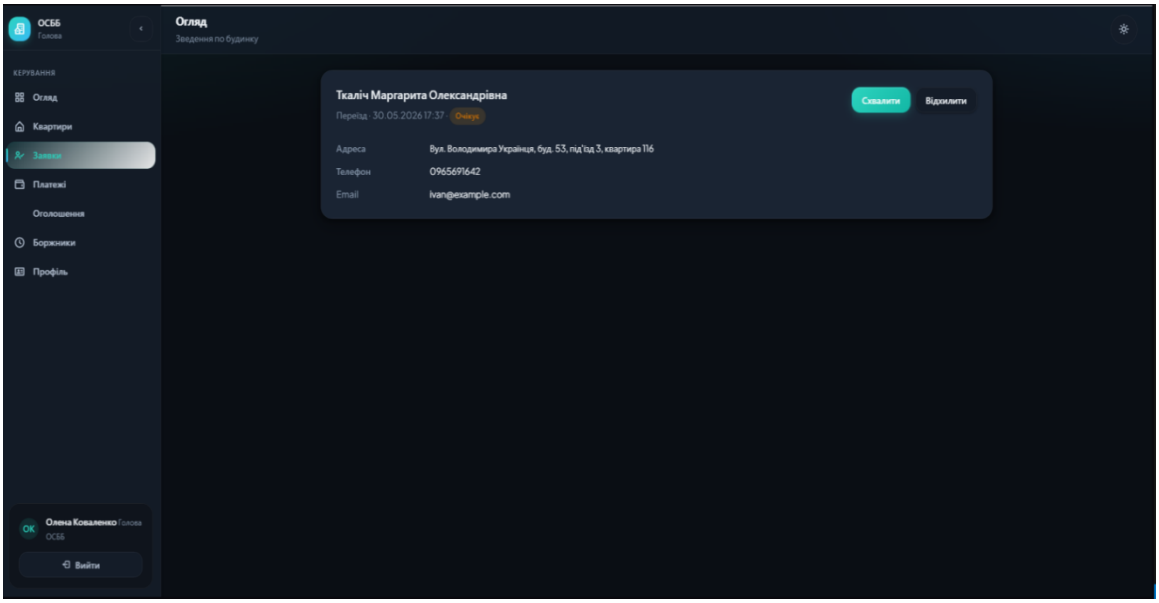


Рисунок 3.12 – Сторінка обробки заявок мешканців (ПК версія)

У мобільній версії інтерфейс адаптований для роботи на пристроях із невеликим розміром екрана. Список заявок відображається у вертикальному вигляді, а елементи керування автоматично підлаштовуються під ширину екрана (див. рис. 3.13).

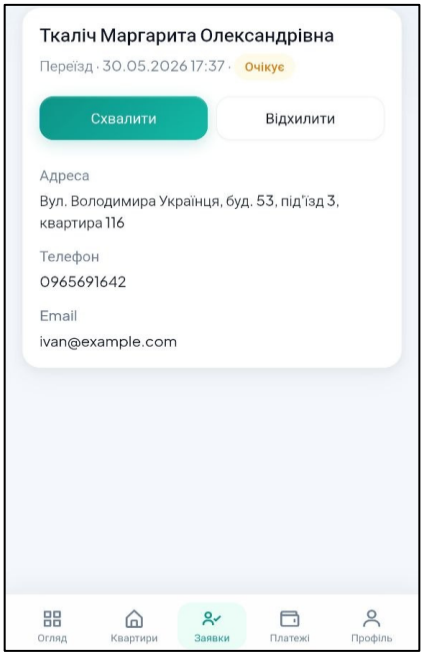


Рисунок 3.13 – Сторінка обробки заявок мешканців (мобільна версія)

Користувач може переглядати інформацію про заявки та виконувати їх схвалення або відхилення так само, як і в версії для персонального комп’ютера. Усі функціональні можливості сторінки повністю збережені.

Отже, розроблений інтерфейс інформаційної системи забезпечує зручну та зрозумілу взаємодію користувачів із функціональними можливостями системи. Використання адаптивного дизайну дозволяє комфортно працювати як на персональних комп’ютерах, так і на мобільних пристроях. Усі сторінки мають єдину структуру та стиль оформлення, що спрощує навігацію та підвищує ефективність роботи користувачів.

3.3 Тестування вебзастосунку

3.3.1 Ручне тестування

Для забезпечення якості розробленого програмного забезпечення та перевірки відповідності функціональним вимогам, визначеним у підрозділі 1.3, було проведено ручне функціональне тестування. Тест-кейси охоплюють основні бізнес-процеси системи: автентифікацію користувачів, реєстрацію мешканців через заявки, управління нарахуваннями (як з боку голови ОСББ, так і з боку мешканця), оплату послуг (у тому числі часткову), публікацію та перегляд оголошень, а також обробку помилкових дій користувача (негативне тестування). У таблиці 3.3 наведено перелік виконаних тест-кейсів із зазначенням очікуваних та фактичних результатів.

Таблиця 3.3 – Тест-кейси ручного тестування

№	Сценарій	Очікуваний результат	Фактичний результат
1.	Вхід із правильними обліковими даними	Перехід до кабінету користувача	Збігається
2.	Вхід із неправильним паролем	Повідомлення про помилку, доступ не надано	Збігається

Продовження таблиці 3.3

№	Сценарій	Очікуваний результат	Фактичний результат
3.	Реєстрація нового мешканця (подача заявки)	Заявка з’являється у списку голови ОСББ	Збігається
4.	Схвалення заявки головою ОСББ	Користувач отримує роль Resident, прив’язка до ОСББ	Збігається
5.	Призначення нарахувань	Для всіх квартир створюються записи	Збігається
6.	Оплата нарахування	Зменшення заборгованості, створення квитанції	Збігається
7.	Публікація оголошення головою ОСББ	Оголошення з’являється у списку	Збігається
8.	Перегляд оголошень мешканцем	Відображаються актуальні оголошення	Збігається
9.	Часткова оплата нарахування	Статус змінюється на Partial, створюється квитанція	Збігається

Критеріями успішного проходження тестування є відповідність фактичного результату очікуваному, коректне збереження даних у базі даних та відсутність критичних помилок, пов’язаних із втратою інформації або неправильним розрахунком платежів.

3.3.2 Автоматизоване тестування

Для перевірки коректності роботи серверної частини інформаційної системи було реалізовано автоматизоване тестування з використанням фреймворку xUnit. Для ізоляції тестового середовища застосовано Entity Framework Core InMemory, що дозволяє виконувати перевірку бізнес-логіки без підключення до реальної бази даних MySQL.

Автоматизованими тестами охоплено чотири групи компонентів: сервіс автентифікації (AuthService), репозиторій роботи із заявками (JoinRequestRepository), репозиторій роботи з платежами (PaymentRepository), а також інтеграційні API-тести.

У таблиці Б.1 (додаток Б) наведено перелік реалізованих автоматизованих тестів, їх призначення та результати виконання.

На рисунку 3.14 наведено тест, який перевіряє процес схвалення заявки на заселення. Після виконання операції користувач отримує роль мешканця, прив’язується до ОСББ, а статус заявки змінюється на «Схвалено».

```
public async Task ApproveJoinRequest_WithoutAccount_UpgradesExistingGuest()
{
    var dbName = Guid.NewGuid().ToString();
    var (db, repo) = TestDatabase.CreateRepository(dbName);
    TestDatabase.SeedOsbbWithHead(db);
    var guest = TestDatabase.SeedGuest(db, "approve@test.local");

    var request = await repo.SubmitJoinRequestAsync(new JoinRequest
    {
        Kind = JoinRequestKind.NewMoveIn,
        TargetOsbbId = TestDatabase.OsbbId,
        FullName = "Оновлене ПІБ",
        FullAddress = "м. Запоріжжя, вул. Володимира Українця , буд. 53, кв. 12",
        Phone = "+380509999999",
        Email = "test@gmail.com"
    });

    await repo.ApproveJoinRequestAsync(request.Id);

    var user = db.Users.Single(u => u.Id == guest.Id);
    Assert.Equal(UserRole.Resident, user.Role);
    Assert.Equal(TestDatabase.OsbbId, user.OsbbId);
    Assert.Null(user.ApartmentId);

    var stored = db.JoinRequests.Single(r => r.Id == request.Id);
    Assert.Equal(JoinRequestStatus.Approved, stored.Status);
    Assert.Equal(guest.Id, stored.ApprovedUserId);
}
```

Рисунок 3.14 – Тест перевірки схвалення заявки на заселення

Результати виконання підтверджують стабільну роботу основних компонентів серверної частини інформаційної системи (див. рис. 3.15).

```
Test summary: total: 16; failed: 0; succeeded: 16; skipped: 0; duration: 2,6s
Build succeeded in 6,0s
```

Рисунок 3.15 – Результати виконання автоматичного тестування

Проведене ручне та автоматизоване тестування підтвердило коректність функціонування розробленої інформаційної системи автоматизації діяльності ОСББ.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було розроблено інформаційну систему управління діяльністю ОСББ у форматі вебзастосунку. Основною метою роботи було створення програмного рішення для автоматизації обліку мешканців, квартир, заявок, нарахувань та оплат у межах ОСББ.

Під час виконання роботи було проведено аналіз предметної області та існуючих підходів до організації систем управління ОСББ. Розглянуто програмні рішення «ОСББ 365» та «Мій дім online», визначено їх переваги та недоліки. На основі проведеного аналізу сформульовано основні функціональні та нефункціональні вимоги до системи та обрано сучасний стек технологій для реалізації вебзастосунку.

У процесі проєктування було розроблено UML-діаграми (діаграму класів, діаграму варіантів використання, діаграми активності) та ER-модель бази даних, які відображають структуру предметної області, взаємозв’язки між сутностями та логіку взаємодії користувачів із системою. Для зберігання даних використано СУБД MySQL, а взаємодія із базою даних реалізована за допомогою Entity Framework Core.

Серверну частину системи реалізовано на платформі ASP.NET Core 9 із використанням підходу Minimal API. Клієнтська частина реалізована за допомогою Blazor WebAssembly. Обмін даними між клієнтом та сервером здійснюється через REST API у форматі JSON. Для забезпечення авторизації користувачів реалізовано механізм JWT-аутентифікації.

У системі реалізовано функціонал реєстрації та авторизації користувачів, керування квартирами, подання та обробки заявок на заселення (схвалення або відхилення), створення та перегляду оголошень, автоматизованого нарахування платежів з урахуванням площі квартири, формування квитанцій та перегляду заборгованостей. Також реалізовано

розмежування прав доступу відповідно до ролей користувачів (голова ОСББ, мешканець) та підтримку адаптивного інтерфейсу для мобільних пристроїв.

Під час розробки виконано тестування основних функціональних можливостей системи. Ручне тестування охопило основні сценарії взаємодії користувачів, зокрема автентифікацію, подання заявок, оплату нарахувань та публікацію оголошень. Автоматизоване тестування серверної частини реалізовано з використанням xUnit та InMemory Database. Результати тестування підтвердили коректність роботи вебзастосунку, правильність взаємодії клієнтської та серверної частин, а також коректність операцій із базою даних.

Розроблена інформаційна система може бути використана як основа для подальшого розвитку та впровадження в діяльність ОСББ. У подальшому систему можна розширити шляхом інтеграції платіжних сервісів, реалізації push-сповіщень, електронного голосування мешканців та мобільного застосунку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Закон України «Про об’єднання співвласників багатоквартирного будинку». *База даних «Законодавство України».* URL: https://zakon.rada.gov.ua/laws/show/2866-14 (дата звернення: 15.01.2026).

2. Elmasri R., Navathe S. Fundamentals of Database Systems : 7th ed. Boston : Pearson, 2017. 1272 p.

3. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language : 3rd ed. Boston : Addison-Wesley, 2004. 208 p.

4. Introduction to JSON Web Tokens. *JWT.IO*. URL: https://jwt.io/introduction (дата звернення: 10.03.2026).

5. Microsoft. Blazor documentation. *Microsoft Learn.* URL: https://learn.microsoft.com/aspnet/core/blazor (дата звернення: 20.02.2026).

6. Microsoft. C# documentation. *Microsoft Learn.* URL: https://learn.microsoft.com/dotnet/csharp (дата звернення: 16.02.2026).

7. Microsoft. Blazor WebAssembly. *Microsoft Learn.* URL: https://learn.microsoft.com/aspnet/core/blazor/webassembly (дата звернення: 25.02.2026).

8. Oracle. MySQL Documentation. *Oracle Documentation*. URL: https://dev.mysql.com/doc (дата звернення: 07.04.2026).

9. Sommerville I. Software Engineering : 10th ed. Boston : Pearson, 2016. 816 p.

ДОДАТОК А

Лістинги SQL-скриптів бази даних

А.1 Файл 03_reset.SQL

```
DROP DATABASE IF EXISTS osbb_app;
CREATE DATABASE osbb_app
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;
```

А.2 Файл 01_schema.sql

```
CREATE DATABASE IF NOT EXISTS osbb_app
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

USE osbb_app;

CREATE TABLE osbb_buildings (
   id            CHAR(36)     NOT NULL,
   name          VARCHAR(200) NOT NULL,
   address       VARCHAR(500) NOT NULL,
   head_user_id  CHAR(36)     NOT NULL COMMENT 'FK users – голова ОСББ',
   created_at    DATETIME(3)  NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
   updated_at    DATETIME(3)  NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON
UPDATE CURRENT_TIMESTAMP(3),
   PRIMARY KEY (id),
   KEY ix_osbb_head_user (head_user_id)
) ENGINE=InnoDB;

CREATE TABLE users (
   id            CHAR(36)     NOT NULL,
   role          TINYINT UNSIGNED NOT NULL COMMENT '0=OsbbHead, 1=Resident,
2=Guest',
   full_name     VARCHAR(200)  NOT NULL,
   phone         VARCHAR(32)   NOT NULL DEFAULT '',
   email         VARCHAR(254)  NOT NULL DEFAULT '',
   birth_date    DATE          NULL,
```

```
  photo_data_url  LONGTEXT      NULL COMMENT 'Base64 data URL; у проді краще URL у object storage',
  osbb_id         CHAR(36)      NULL COMMENT 'ОСББ, до якого прив’язаний користувач',
  apartment_id    CHAR(36)      NULL COMMENT 'Квартира мешканця',
  relation_type   TINYINT UNSIGNED NULL COMMENT '0=Owner, 1=CoOwner; NULL для голови/гостя',
  password_hash   VARCHAR(255)  NULL COMMENT 'bcrypt/Argon2 – для API',
  is_active       TINYINT(1)    NOT NULL DEFAULT 1,
  created_at      DATETIME(3)   NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  updated_at      DATETIME(3)   NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  KEY ix_users_osbb (osbb_id),
  KEY ix_users_apartment (apartment_id),
  KEY ix_users_role (role),
  UNIQUE KEY uq_users_email (email)
) ENGINE=InnoDB;

CREATE TABLE apartments (
  id              CHAR(36)      NOT NULL,
  osbb_id         CHAR(36)      NOT NULL,
  number          VARCHAR(32)   NOT NULL COMMENT 'Номер квартири',
  account_number  VARCHAR(64)   NOT NULL COMMENT 'Лицевий рахунок',
  area_sqm        DECIMAL(10, 2)  NOT NULL DEFAULT 0,
  room_count      INT UNSIGNED  NOT NULL DEFAULT 0,
  owner_id        CHAR(36)      NULL,
  co_owner_id     CHAR(36)      NULL,
  created_at      DATETIME(3)   NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  updated_at      DATETIME(3)   NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  UNIQUE KEY uq_apartments_osbb_account (osbb_id, account_number),
  KEY ix_apartments_osbb (osbb_id),
  KEY ix_apartments_owner (owner_id),
  KEY ix_apartments_co_owner (co_owner_id)
) ENGINE=InnoDB;

CREATE TABLE join_requests (
  id              CHAR(36)      NOT NULL,
  kind            TINYINT UNSIGNED NOT NULL COMMENT '0=NewMoveIn, 1=TransferToAnotherOsbb',
  status          TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '0=Pending, 1=Approved, 2=Rejected',
  target_osbb_id  CHAR(36)      NOT NULL COMMENT 'Куди вступає / куди переїжджає',
  from_osbb_id    CHAR(36)      NULL COMMENT 'Звідки переїжджає (Transfer)',
  full_name       VARCHAR(200)  NOT NULL,
```

```
  phone            VARCHAR(32)    NOT NULL DEFAULT '',
  email            VARCHAR(254)   NOT NULL DEFAULT '',
  birth_date       DATE           NULL,
  full_address     VARCHAR(500)   NOT NULL DEFAULT '',
  account_number   VARCHAR(64)    NOT NULL DEFAULT '',
  photo_data_url   LONGTEXT       NULL,
  submitted_at     DATETIME(3)    NOT NULL,
  approved_user_id CHAR(36)       NULL COMMENT 'Створений мешканець після Approve',
  processed_at     DATETIME(3)    NULL COMMENT 'Час схвалення/відхилення',
  created_at       DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  updated_at       DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  KEY ix_join_target_osbb_status (target_osbb_id, status),
  KEY ix_join_from_osbb (from_osbb_id),
  KEY ix_join_submitted (submitted_at DESC),
  KEY ix_join_approved_user (approved_user_id)
) ENGINE=InnoDB;

CREATE TABLE announcements (
  id           CHAR(36)       NOT NULL,
  osbb_id      CHAR(36)       NOT NULL,
  title        VARCHAR(300)   NOT NULL,
  body         TEXT           NOT NULL,
  published_at DATETIME(3)    NOT NULL,
  is_pinned    TINYINT(1)     NOT NULL DEFAULT 0,
  created_by   CHAR(36)       NULL COMMENT 'Голова ОСББ',
  created_at   DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  updated_at   DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  KEY ix_announcements_osbb_published (osbb_id, published_at DESC),
  KEY ix_announcements_pinned (osbb_id, is_pinned, published_at DESC)
) ENGINE=InnoDB;

CREATE TABLE payment_charges (
  id           CHAR(36)       NOT NULL,
  osbb_id      CHAR(36)       NOT NULL,
  apartment_id CHAR(36)       NOT NULL,
  title        VARCHAR(200)   NOT NULL,
  period_label VARCHAR(100)   NOT NULL COMMENT 'Напр. «Березень 2026»',
  amount       DECIMAL(12, 2) NOT NULL,
  paid_amount  DECIMAL(12, 2) NOT NULL DEFAULT 0 COMMENT 'Накопичена сплачена сума',
  status       TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '0=Pending,
```

```
1=Partial, 2=Paid',
  created_at     DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  updated_at     DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON
UPDATE CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  KEY ix_charges_apartment_status (apartment_id, status, created_at DESC),
  KEY ix_charges_osbb_status (osbb_id, status),
  KEY ix_charges_osbb_period (osbb_id, period_label)
) ENGINE=InnoDB;

CREATE TABLE payment_receipts (
  id             CHAR(36)       NOT NULL,
  charge_id      CHAR(36)       NOT NULL,
  apartment_id   CHAR(36)       NOT NULL,
  paid_by_user_id CHAR(36)      NOT NULL,
  receipt_number  VARCHAR(64)   NOT NULL,
  amount         DECIMAL(12, 2)  NOT NULL,
  paid_at        DATETIME(3)    NOT NULL,
  created_at     DATETIME(3)    NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  PRIMARY KEY (id),
  KEY ix_receipts_charge (charge_id),
  UNIQUE KEY uq_receipts_number (receipt_number),
  KEY ix_receipts_apartment_paid (apartment_id, paid_at DESC),
  KEY ix_receipts_user (paid_by_user_id)
) ENGINE=InnoDB;

ALTER TABLE osbb_buildings
  ADD CONSTRAINT fk_osbb_head
    FOREIGN KEY (head_user_id) REFERENCES users (id)
    ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE users
  ADD CONSTRAINT fk_users_osbb
    FOREIGN KEY (osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT fk_users_apartment
    FOREIGN KEY (apartment_id) REFERENCES apartments (id)
    ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE apartments
  ADD CONSTRAINT fk_apartments_osbb
    FOREIGN KEY (osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  ADD CONSTRAINT fk_apartments_owner
    FOREIGN KEY (owner_id) REFERENCES users (id)
    ON DELETE SET NULL ON UPDATE CASCADE,
```

```
  ADD CONSTRAINT fk_apartments_co_owner
    FOREIGN KEY (co_owner_id) REFERENCES users (id)
    ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE join_requests
  ADD CONSTRAINT fk_join_target_osbb
    FOREIGN KEY (target_osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  ADD CONSTRAINT fk_join_from_osbb
    FOREIGN KEY (from_osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT fk_join_approved_user
    FOREIGN KEY (approved_user_id) REFERENCES users (id)
    ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE announcements
  ADD CONSTRAINT fk_announcements_osbb
    FOREIGN KEY (osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT fk_announcements_created_by
    FOREIGN KEY (created_by) REFERENCES users (id)
    ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE payment_charges
  ADD CONSTRAINT fk_charges_osbb
    FOREIGN KEY (osbb_id) REFERENCES osbb_buildings (id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  ADD CONSTRAINT fk_charges_apartment
    FOREIGN KEY (apartment_id) REFERENCES apartments (id)
    ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE payment_receipts
  ADD CONSTRAINT fk_receipts_charge
    FOREIGN KEY (charge_id) REFERENCES payment_charges (id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  ADD CONSTRAINT fk_receipts_apartment
    FOREIGN KEY (apartment_id) REFERENCES apartments (id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  ADD CONSTRAINT fk_receipts_user
    FOREIGN KEY (paid_by_user_id) REFERENCES users (id)
    ON DELETE RESTRICT ON UPDATE CASCADE;


CREATE OR REPLACE VIEW v_debtors AS
SELECT
  a.id          AS apartment_id,
```

```
    a.osbb_id,
    a.number         AS flat_number,
    a.account_number,
    u.id             AS resident_id,
    u.full_name      AS resident_name,
    u.phone,
    COUNT(c.id)      AS overdue_charges_count,
    SUM(c.amount - c.paid_amount) AS total_debt,
    MIN(c.created_at) AS oldest_charge_at
FROM apartments a
INNER JOIN payment_charges c
    ON c.apartment_id = a.id AND c.paid_amount < c.amount
LEFT JOIN users u
    ON u.id = a.owner_id
GROUP BY
    a.id, a.osbb_id, a.number, a.account_number,
    u.id, u.full_name, u.phone;

CREATE OR REPLACE VIEW v_osbb_summary AS
SELECT
    o.id AS osbb_id,
    o.name,
    (SELECT COUNT(*) FROM apartments ap WHERE ap.osbb_id = o.id) AS apartments_count,
    (SELECT COUNT(*) FROM join_requests jr
     WHERE jr.target_osbb_id = o.id AND jr.status = 0) AS pending_requests_count,
    (SELECT COUNT(*) FROM announcements an WHERE an.osbb_id = o.id) AS
announcements_count,
    (SELECT COUNT(*) FROM payment_charges pc
     WHERE pc.osbb_id = o.id AND pc.paid_amount < pc.amount) AS pending_charges_count
FROM osbb_buildings o;
```

A.3 Файл 02_procedures.sql

```
USE osbb_app;

DELIMITER $$

CREATE PROCEDURE sp_approve_join_request(IN p_request_id CHAR(36))
BEGIN
    DECLARE v_target_osbb CHAR(36);
    DECLARE v_full_name VARCHAR(200);
    DECLARE v_phone VARCHAR(32);
```

```
    DECLARE v_email VARCHAR(254);
    DECLARE v_birth DATE;
    DECLARE v_photo LONGTEXT;
    DECLARE v_account VARCHAR(64);
    DECLARE v_address VARCHAR(500);
    DECLARE v_new_user_id CHAR(36);
    DECLARE v_apt_id CHAR(36);
    DECLARE v_flat_number VARCHAR(32);

    SELECT target_osbb_id, full_name, phone, email, birth_date,
           photo_data_url, account_number, full_address
    INTO v_target_osbb, v_full_name, v_phone, v_email, v_birth,
         v_photo, v_account, v_address
    FROM join_requests
    WHERE id = p_request_id AND status = 0
    FOR UPDATE;

    IF v_target_osbb IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Заявку не знайдено або вона вже
оброблена';
    END IF;

    SET v_new_user_id = UUID();

    SELECT id INTO v_apt_id
    FROM apartments
    WHERE osbb_id = v_target_osbb AND account_number = v_account
    LIMIT 1;

    IF v_apt_id IS NULL THEN
        SET v_apt_id = UUID();
        SET v_flat_number = TRIM(SUBSTRING_INDEX(v_address, ',', -1));

        INSERT INTO apartments (id, osbb_id, number, account_number, area_sqm, room_count,
owner_id)
        VALUES (v_apt_id, v_target_osbb, IFNULL(NULLIF(v_flat_number, ''), '–'), v_account, 0,
0, v_new_user_id);
    ELSE
        UPDATE apartments SET owner_id = COALESCE(owner_id, v_new_user_id) WHERE id
= v_apt_id;
    END IF;

    INSERT INTO users (id, role, full_name, phone, email, birth_date, photo_data_url,
                       osbb_id, apartment_id, relation_type)
    VALUES (v_new_user_id, 1, v_full_name, v_phone, v_email, v_birth, v_photo,
            v_target_osbb, v_apt_id, 0);
```

```
    UPDATE join_requests
    SET status = 1,
        approved_user_id = v_new_user_id,
        processed_at = NOW(3)
    WHERE id = p_request_id;
END$$

CREATE PROCEDURE sp_reject_join_request(IN p_request_id CHAR(36))
BEGIN
    UPDATE join_requests
    SET status = 2, processed_at = NOW(3)
    WHERE id = p_request_id AND status = 0;

    IF ROW_COUNT() = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Заявку не знайдено';
    END IF;
END$$
CREATE PROCEDURE sp_assign_charge_to_all(
    IN p_osbb_id CHAR(36),
    IN p_title VARCHAR(200),
    IN p_period_label VARCHAR(100),
    IN p_amount DECIMAL(12, 2)
)
BEGIN
    INSERT INTO payment_charges (id, osbb_id, apartment_id, title, period_label, amount, status)
    SELECT UUID(), p_osbb_id, a.id, p_title, p_period_label, p_amount, 0
    FROM apartments a
    WHERE a.osbb_id = p_osbb_id;
END$$

-- Оплата нарахування (повна або часткова)
CREATE PROCEDURE sp_pay_charge(
    IN p_charge_id CHAR(36),
    IN p_user_id CHAR(36),
    IN p_amount DECIMAL(12, 2),
    OUT p_receipt_id CHAR(36)
)
BEGIN
    DECLARE v_apartment_id CHAR(36);
    DECLARE v_total DECIMAL(12, 2);
    DECLARE v_paid DECIMAL(12, 2);
    DECLARE v_remaining DECIMAL(12, 2);
    DECLARE v_new_paid DECIMAL(12, 2);
    DECLARE v_receipt_no VARCHAR(64);

    SELECT apartment_id, amount, paid_amount
```

```
    INTO v_apartment_id, v_total, v_paid
    FROM payment_charges
    WHERE id = p_charge_id AND paid_amount < amount
    FOR UPDATE;

    IF v_apartment_id IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Нарахування не знайдено або вже повністю оплачене';
    END IF;

    SET v_remaining = v_total - v_paid;

    IF p_amount IS NULL OR p_amount <= 0 OR p_amount > v_remaining THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Некоректна сума оплати';
    END IF;

    SET v_new_paid = v_paid + p_amount;
    SET p_receipt_id = UUID();
    SET v_receipt_no = CONCAT('KV-', DATE_FORMAT(NOW(), '%Y%m%d'), '-', LPAD(FLOOR(RAND() * 9000) + 1000, 4, '0'));

    UPDATE payment_charges
    SET paid_amount = v_new_paid,
        status = IF(v_new_paid >= v_total, 2, 1)
    WHERE id = p_charge_id;

    INSERT INTO payment_receipts (id, charge_id, apartment_id, paid_by_user_id, receipt_number, amount, paid_at)
    VALUES (p_receipt_id, p_charge_id, v_apartment_id, p_user_id, v_receipt_no, p_amount, NOW(3));
END$$

DELIMITER ;
```

ДОДАТОК Б

Автоматизоване тестування

Перевірку працездатності та коректності функціонування серверної частини вебзастосунку було виконано за допомогою автоматизованого тестування, результати якого наведено в таблиці Б.1.

Таблиця Б.1 – Автоматизовані тести серверної частини

№	Тестовий метод	Компонент	Призначення тесту	Результат
1.	RegisterAsync_CreatesGuestWithHashedPassword	AuthService	Перевірка реєстрації нового користувача	Успішно
2.	RegisterAsync_DuplicateEmail_Throws	AuthService	Перевірка обробки помилки при спробі реєстрації з email, що вже існує	Успішно
3.	RegisterAsync_ShortPassword_Throws	AuthService	Перевірка валідації довжини пароля	Успішно
4.	LoginAsync_ValidCredentials_ReturnsJwt	AuthService	Перевірка успішної автентифікації з правильними обліковими даними	Успішно
5.	LoginAsync_WrongPassword_Throws	AuthService	Перевірка відмови в автентифікації при неправильному паролі	Успішно
6.	SubmitJoinRequest_SetsPendingStatusAndId	JoinRequestRepository	Перевірка створення заявки на заселення	Успішно
7.	ApproveJoinRequest_WithoutAccount_UpgradesExistingGuest	JoinRequestRepository	Перевірка схвалення заявки	Успішно
8.	ApproveJoinRequest_LinksVacantApartmentByFlatNumber	JoinRequestRepository	Перевірка автоматичної прив'язки мешканця до квартири за її номером	Успішно
9.	RejectJoinRequest_SetsRejectedStatus	JoinRequestRepository	Перевірка відхилення заявки: статус змінюється на Rejected	Успішно
10.	GetJoinRequestsAsync_FiltersByOsbbAndStatus	JoinRequestRepository	Перевірка фільтрації заявок за ОСББ та статусом	Успішно

Продовження таблиці Б.1

№	Тестовий метод	Компонент	Призначення тесту	Результат
11.	AssignChargeToAll_CalculatesAmountFromAreaAndExtras	PaymentRepository	Перевірка автоматичного розрахунку нарахувань з урахуванням площі квартири, тарифу утримання та додаткових фіксованих платежів	Успішно
12.	PayChargeAsync_PartialThenFull_UpdatesStatus	PaymentRepository	Перевірка часткової оплати нарахування та повного погашення	Успішно
13.	PayChargeAsync_AmountExceedsRemaining_Throws	PaymentRepository	Перевірка обробки помилки при спробі оплати суми, що перевищує залишок заборгованості	Успішно
14.	PayChargeAsync_AlreadyPaid_Throws	PaymentRepository	Перевірка обробки помилки при спробі оплати вже повністю сплаченого нарахування	Успішно
15.	PostJoinRequests_AllowsAnonymous	API (/api/join-requests)	Перевірка, що подача заявки недоступна неавторизованим користувачам	Успішно
16.	GetJoinRequests_RequiresHeadRole	API (/api/join-requests)	Перевірка, що перегляд заявок доступний лише авторизованому голові ОСББ	Успішно