НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики Кафедра системного програмування

**і спеціалізованих комп‘ютерних систем**

«На правах рукопису» «До захисту допущено»

УДК 004.67

Завідувач кафедри

Віталій РОМАНКЕВИЧ “ ” 2024 р.

Магістерська дисертація

**на здобуття ступеня магістра**

**за освітньо-професійною програмою «Системне програмування та спеціалізовані комп‘ютерні системи»**

**зі спеціальності 123 «Комп‘ютерна інженерія»**

**на тему: «Інформаційна система управління та комунікації для ОСББ»**

Виконав:

студент ІІ курсу, групи КВ-32мп Панченко Денис Андрійович

Науковий керівник:

Доцент кафедри СПіСКС, к.т.н., доцент, Щербина Олександр Андрійович

Рецензент:

Доцент кафедри ОТ ФІОТ, к.т.н., доцент,

Верба Олександр Андрійович

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Київ – 2024 року

Національний технічний університет України

**«Київський політехнічний інститут імені Ігоря Сікорського» Факультет прикладної математики**

Кафедра системного програмування

**і спеціалізованих комп’ютерних систем**

Рівень вищої освіти – другий (магістерський) Спеціальність – 123 «Комп‘ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп‘ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Віталій РОМАНКЕВИЧ “ ” 2024 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Панченко Денису Андрійовичу

Тема дисертації «Інформаційна система управління та комунікації для ОСББ», науковий керівник дисертації Щербина Олександр Андрійович, к.т.н., доцент, затверджені наказом по університету від «08» листопада 2024 р. № 5007-с

Термін подання студентом дисертації 9 грудня 2024

Об’єкт дослідження: управління процесами в об'єднаннях співвласників багатоквартирних будинків.

Предмет дослідження: методи та засоби автоматизації управління та комунікації в ОСББ із використанням інформаційних систем.

Перелік завдань, які потрібно розробити:

* + провести аналіз існуючих інформаційних систем управління та комунікації в ОСББ та їхніх можливостей для автоматизації фінансових, технічних і комунікаційних процесів;
  + покращити методологію створення інтегрованої інформаційної системи для ОСББ, яка включає:

1. інтеграцію з платіжними системами для автоматизації фінансових операцій;
2. платформу для електронного голосування та комунікації співвласників;
3. моніторинг електропостачання та інфраструктури;
4. покращення автентифікації користувачів за допомогою JWT-токенів для забезпечення безпеки.
   * реалізувати запропоновану інформаційну систему, провести її тестування та виконати порівняльний аналіз ефективності .

Перелік ілюстративного матеріалу: презентація (кількість аркушів: 21)

Перелік публікацій:

* Панченко Д.А., Щербина О.А. Інформаційна система управління та комунікації для ОСББ. VII Міжнародна студентська наукова конференція «Модернізація та сучасні українські і світові наукові дослідження» – 2024. – с. 258-261. – Режим доступу до ресурсу: https://archive.liga.science/index.php/conference-proceedings/issue/view/inter-06.12.2024/108
* Панченко Д.А., Щербина О.А. Інформаційна система управління та комунікації ОСББ, Прикладна математика та комп’ютинг. ПМК, 2024: сімнадцята наук. конф. магістрантів та аспірантів, 20–22 листопада 2024 р.: зб. тез доп. / [редкол.: Дичка І.А. та ін.]. – К. : Просвіта, 2024. – с. 519-522.

Дата видачі завдання: 25.10.2023

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів магістерської дисертації | Примітка |
| 1 | Вивчення літератури за тематикою проекту | 09.12.2023 |  |
| 2 | Розроблення та узгодження завдання магістерської дисертації | 30.12.2023 |  |
| 3 | Аналіз існуючих аналіз існуючих інформаційних систем управління та комунікації в ОСББ | 01.03.2024 |  |
| 4 | Аналіз існуючих способів автоматизації фінансових, технічних і комунікаційних процесів | 01.05.2024 |  |
| 5 | Підготовка матеріалів першого розділу магістреської дисертації | 01.07.2024 |  |
| 6 | Підготовка матеріалів другого розділу магістреської дисертації | 01.08.2024 |  |
| 7 | Реалізація мови опису відповідно до запропонованого способу | 01.09.2024 |  |
| 8 | Підготовка матеріалів третього розділу магістерської дисертації | 01.10.2024 |  |
| 9 | Підготовка матеріалів четвертого розділу магістерської дисертації | 01.11.2024 |  |
| 10 | Попередній розгляд магістерської дисертації на кафедрі | 2.12.2024 |  |

Студент Денис ПАНЧЕНКО

Науковий керівник Олександр ЩЕРБИНА

РЕФЕРАТ

**Актуальність теми.** Ефективне управління багатоквартирними будинками є невід'ємною складовою розвитку житлово-комунального господарства. В Україні, де багато багатоквартирних будинків експлуатуються ОСББ (об'єднання співвласників багатоквартирного будинку), залишається проблема недостатньої автоматизації управлінських процесів. Це створює ризики затримок у вирішенні важливих питань, непрозорості у фінансовій звітності, а також труднощі в моніторингу стану будинкової інфраструктури.   
В умовах сучасного суспільства цифровізація є ключем до підвищення ефективності, прозорості та задоволеності мешканців. Інформаційні системи для ОСББ мають забезпечувати швидке прийняття рішень, оперативний обмін інформацією та прозорість у фінансових питаннях, що є актуальним як для співвласників, так і для органів управління.

**Мета і задачі дослідження.** Метою дослідження є створення комплексної інформаційної системи для автоматизації управлінських процесів у ОСББ, що забезпечують прозорість фінансових операцій, якісний моніторинг стану інфраструктури та налагодження ефективної комунікації між співвласниками. Система спрямована на вирішення таких задач:

* мінімізація адміністративних витрат;
* створення механізму електронного голосування;
* автоматизація фінансового обліку;
* своєчасний контроль технічного стану будівельних елементів;
* забезпечення інтерактивного доступу до звітності для мешканців.

**Об’єктом дослідження** виступають процеси управління багатоквартирними будинками, зокрема в контексті фінансової звітності, комунікації, моніторингу інфраструктури та технічного обслуговування, що здійснюються ОСББ.

**Предметом дослідження** є інструменти автоматизації управління, які включають програмні рішення для обліку фінансів, моніторингу інфраструктури (наприклад, ліфти, системи водопостачання та опалення) та комунікаційні платформи, що дозволяють організовувати електронне голосування, обмін повідомленнями та сповіщення мешканців.

**Методи дослідження*.*** Для досягнення поставлених цілей було застосовано такі методи:

* **Спостереження та порівняння** – для збору даних про існуючі процеси ОСББ.
* **Системний аналіз** – для аналізу структури ОСББ та визначення їхніх потреб.
* **Імітаційне моделювання** – для моделювання фінансових потоків та технічного обслуговування.
* **Аналіз потреб** – для вивчення потреб мешканців через опитування та анкетування.
* **Прототипування** – для розробки інтерактивних модулів (комунікація, голосування, облік).
* **Програмування і тестування** – для створення і перевірки функціональних компонентів.
* **Методи обробки даних** – для автоматизації фінансового обліку та прогнозування витрат.

**Наукова новизна.** У дисертації набули подальшого розвитку методи управління процесами в об'єднаннях співвласників багатоквартирних будинків, які були реалізовані під час розробки інтегрованої інформаційної системи, що поєднує фінансовий, технічний та комунікаційний аспекти управління в єдиному цифровому середовищі. Це дало змогу автоматизувати ключові процеси, підвищити ефективність прийняття рішень та покращити облік і контроль технічного стану об'єктів у відповідності до сучасних технологічних викликів. Запропонований підхід до інтеграції сучасних технологій програмування, автоматизації процесів та архітектурних рішень, забезпечує не лише реалізацію кожного із зазначених аспектів, а й їхню взаємодію, що сприяє прозорості, ефективності та покращенню комунікації між співвласниками і адміністрацією ОСББ. Зокрема вдосконалено методологію побудови інформаційних систем для ОСББ, що включає:

* Інтеграцію з платіжними системами, що забезпечує зручність і прозорість фінансових операцій.
* Розробку систем електронного голосування та комунікації, які сприяють колективному прийняттю рішень і підвищують рівень залучення співвласників.
* Інтеграцію з системами моніторингу електропостачання, що дозволяє оперативно відстежувати стан електромереж і планувати заходи для запобігання відключенням.
* Покращення методів автентифікації користувачів за допомогою JWT-токенів, що підвищує безпеку доступу до системи та захищає персональні дані користувачів.

**Практичне значення одержаних результатів** полягає у створенні інформаційної системи, яка дає змогу:

* зменшити адміністративні витрати за рахунок автоматизації рутинних процесів;
* забезпечити прозорість фінансової звітності, що сприяє довірі мешканців;
* організовувати голосування та опитування без фізичної присутності співвласників;
* здійснювати оперативний контроль стану інфраструктури, запобігаючи аварійним ситуаціям;
* покращити якість обслуговування мешканців завдяки своєчасним сповіщенням та доступу до актуальної інформації.

Особистий внесок магістранта. Автор самостійно провів аналіз потреб ОСББ через опитування мешканців, що стало основою для формулювання вимог до інформаційної системи. Запропонував структуру мультифункціональної інформаційної системи для ОСББ, яка інтегрує фінансовий, технічний і комунікаційний модулі. Розробив алгоритм автоматизованого обліку платежів і адаптації технологій електронного голосування до специфіки багатоквартирних будинків. Усі результати були апробовані особисто автором шляхом створення та тестування прототипів системи в умовах, наближених до реальних.

**Апробацiя результатiв.** Основні положення і результати дисертації представлені та обговорювались на:

● XVII-й науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2024 (Київ, 20-22 листопада 2024 р.);

● VII-й Міжнародній студентській науковій конференції «Модернізація та сучасні українські і світові наукові дослідження» (Чернігів, 6 грудня 2024 р.).

Публікаї роботи

1. Панченко Д.А., Щербина О.А. Інформаційна система управління та комунікації для ОСББ. VII Міжнародна студентська наукова конференція «Модернізація та сучасні українські і світові наукові дослідження» – 2024. – с.258-261. – URL: https://archive.liga.science/index.php/conference-proceedings/issue/view/inter-06.12.2024/108
2. Панченко Д.А., Щербина О.А. Інформаційна система управління та комунікації ОСББ, Прикладна математика та комп’ютинг. ПМК, 2024: сімнадцята наук. конф. магістрантів та аспірантів, 20–22 листопада 2024 р.: зб.тез доп./ [редкол.: Дичка І.А. та ін.]. – К. : Просвіта, 2024. – с. 519-522.

**Структура та обсяг роботи.** Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 110 аркушів основного тексту, 63 ілюстрацій, 4 таблиць, список літератури з 22 різних джерел.

**Ключові слова:** ОСББ, автоматизація, інформаційна система, фінансовий облік, моніторинг інфраструктури, електронне голосування, комунікація. інформаційне забезпечення, підсистеми інформаційного забезпечення.

ABSTRACT

**Relevance of the subject.** Efficient management of multi-apartment buildings is an integral part of the development of the housing and utilities sector. In Ukraine, where many multi-apartment buildings are operated by condominiums (HOAs – Homeowners' Associations), there is a persistent issue of insufficient automation of management processes. This leads to risks of delays in addressing important issues, a lack of transparency in financial reporting, and challenges in monitoring the state of building infrastructure. In modern society, digitalization is key to improving efficiency, transparency, and resident satisfaction. Information systems for HOAs enable quick decision-making, prompt information exchange, and transparency in financial matters, which is relevant for both residents and management bodies.

**Goal and objectives of the study.** The goal of the study is to develop a comprehensive information system for automating management processes in HOAs, ensuring transparency of financial operations, effective monitoring of infrastructure conditions, and establishing efficient communication between residents. The system aims to address the following objectives:

* Minimize administrative costs;
* Develop a mechanism for electronic voting;
* Automate financial accounting;
* Ensure timely monitoring of the technical condition of building components;
* Provide interactive access to reports for residents.

**Object of the study.** The study focuses on the management processes of multi-apartment buildings, particularly in the context of financial reporting, communication, infrastructure monitoring, and technical maintenance carried out by HOAs.

**Subject of the study.** The subject of the study includes automation tools for management, encompassing software solutions for financial accounting, infrastructure monitoring (e.g., elevators, water supply, and heating systems), and communication platforms that enable electronic voting, messaging, and resident notifications.

**Research methods.** To achieve the set goals, the following methods were applied:

* **Observation and comparison** – to collect data on existing HOA processes;
* **System analysis** – to analyze the structure of HOAs and identify their needs;
* **Simulation modeling** – to model financial flows and technical maintenance;
* **Needs analysis** – to study resident needs through surveys and questionnaires;
* **Prototyping** – to develop interactive modules (communication, voting, accounting);
* **Programming and testing** – to create and verify the functional components;
* **Data processing methods** – to automate financial accounting and forecast expenses.

**Scientific novelty.** The dissertation advances the methods of managing processes in homeowners' associations through the development of an integrated information system combining financial, technical, and communication aspects in a unified digital environment. This approach automates key processes, enhances decision-making efficiency, and improves infrastructure monitoring and control in response to modern technological challenges. The proposed approach integrates modern programming technologies, process automation, and architectural solutions to ensure interaction between these components, contributing to transparency, efficiency, and improved communication between HOA management and residents.

In particular, the methodology for developing information systems for HOAs was improved, including:

* **Integration with payment systems** to ensure convenience and transparency of financial operations;
* **Development of electronic voting and communication systems** to facilitate collective decision-making and increase resident engagement;
* **Integration with power supply monitoring systems** to track the condition of electrical networks and plan measures to prevent outages;
* **Enhancement of user authentication methods** using JWT tokens, improving system access security and protecting user data.

**Practical significance of the results.** The created information system allows:

* Reducing administrative costs by automating routine processes;
* Ensuring transparency in financial reporting, fostering resident trust;
* Organizing voting and surveys without the physical presence of residents;
* Enabling prompt monitoring of infrastructure conditions to prevent emergencies;
* Improving the quality of resident services through timely notifications and access to relevant information.

**Personal contribution of the author.** The author independently analyzed HOA needs through resident surveys, forming the basis for defining system requirements. The author proposed the structure of a multifunctional HOA information system integrating financial, technical, and communication modules. Algorithms for automated payment accounting and electronic voting technology adaptation for multi-apartment buildings were developed. All results were personally tested by the author through the creation and testing of system prototypes under conditions close to real ones.

**Presentation of results.** The main findings and results of the dissertation were presented and discussed at:

* The XVIIth Scientific Conference of Master's and Ph.D. Students "Applied Mathematics and Computing" (Kyiv, November 20-22, 2024);
* The VIIth International Student Scientific Conference "Modernization and Current Ukrainian and Global Scientific Research" (Chernihiv, December 6, 2024).

**Publications:**

1. Panchenko D.A., Shcherbyna O.A. *Information System for HOA Management and Communication.* VII International Student Scientific Conference *"Modernization and Current Ukrainian and Global Scientific Research,"* 2024. – pp. 258-261. – Available at:<https://archive.liga.science/index.php/conference-proceedings/issue/view/inter-06.12.2024/108>
2. Panchenko D.A., Shcherbyna O.A. *Information System for HOA Management and Communication.* Applied Mathematics and Computing. PMC, 2024: XVII Scientific Conference of Master's and Ph.D. Students, November 20–22, 2024: Abstracts Collection / [Editorial Board: Dyshka I.A. et al.]. – Kyiv: Prosvita, 2024. – pp. 519-522.

**Structure and scope of the work.** The work consists of an introduction and four chapters. Total volume: 111 pages of main text, 63 illustrations, 4 tables. The preparation used literature from 22 different sources.

**Keywords:** HOA, automation, information system, financial accounting, infrastructure monitoring, electronic voting, communication, information support, information support subsystems.

ЗМІСТ

[ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ 3](#_Toc184561359)

[ВСТУП 4](#_Toc184561360)

[1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ 6](#_Toc184561361)

[1.1 ОСББ: Визначення, важливість та актуальність 6](#_Toc184561362)

[1.2 Загальні положення 7](#_Toc184561363)

[1.2.1 Змістовний опис і аналіз предметної області 8](#_Toc184561364)

[1.2.2 Вимоги та виклики 9](#_Toc184561365)

[1.3 Підґрунтя для розробки основних модулів системи 10](#_Toc184561366)

[1.3.1 Система управління фінансами 11](#_Toc184561367)

[1.3.2 Система комунікації між мешканцями ОСББ 13](#_Toc184561368)

[1.3.3 Система моніторингу інфраструктури 15](#_Toc184561369)

[1.3.4 Система колективного прийняття рішень 17](#_Toc184561370)

[1.3.5 Автентифікація та управління користувачами 19](#_Toc184561371)

[1.4.1 Аналіз сервісу «ОСББ – online» 22](#_Toc184561372)

[1.4.2 Аналіз сервісу МОЄОСББ 23](#_Toc184561373)

[1.4.3 Порівняльна характеристика 25](#_Toc184561374)

[ВИСНОВКИ ДО РОЗДІЛУ 1 27](#_Toc184561375)

[2. ПОСТАНОВКА ЗАДАЧІ ТА РОЗРОБКА МЕТОДУ 29](#_Toc184561376)

[2.1 Опис постановки задачі 29](#_Toc184561377)

[2.2 Структура бізнес-процесів 31](#_Toc184561378)

[2.3 Вибір архітектури. 33](#_Toc184561379)

[2.4 Серверна частина 36](#_Toc184561380)

[2.4.1 Середовище програмування 36](#_Toc184561381)

[2.4.2 Express.js 38](#_Toc184561382)

[2.4.3 База данних MongoDB 40](#_Toc184561383)

[2.5 Клієнтська частина 44](#_Toc184561384)

[ВИСНОВКИ ДО РОЗДІЛУ 2 49](#_Toc184561385)

[**3.** **РЕАЛІЗАЦІЯ МЕТОДІВ** 51](#_Toc184561386)

[3.1 Автентифікація 52](#_Toc184561387)

[3.2 Модуль голосувань 58](#_Toc184561388)

[3.3 Модуль комунікації. 63](#_Toc184561389)

[3.4 Фінансовий модуль 66](#_Toc184561390)

[3.4.1 osbbStatisticsController.js 66](#_Toc184561391)

[3.4.2 financeController.js 67](#_Toc184561392)

[3.4.3 paymentController.js 69](#_Toc184561393)

[3.5 Модуль інфраструктури 73](#_Toc184561394)

[3.5.1 Контролер infrastructureController.js 73](#_Toc184561395)

[3.5.2 Контролер для роботи з розкладом відключень електроенергії getSchedule 74](#_Toc184561396)

[3.5.3 Утиліта fetchTableFromWebsite для парсингу розкладу відключень 75](#_Toc184561397)

[ВИСНОВКИ ДО РОЗДІЛУ 3 78](#_Toc184561398)

[4. ТЕСТУВАННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ 80](#_Toc184561399)

[4.1 Тестування інформаційної системи 81](#_Toc184561400)

[4.2 Порівняльний аналіз та швидкодія системи 99](#_Toc184561401)

[ВИСНОВКИ ДО РОЗДІЛУ 4 103](#_Toc184561402)

[ВИСНОВКИ 105](#_Toc184561403)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 107](#_Toc184561404)

# 

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

**ОСББ** – Об’єднання співвласників багатоквартирного будинку. Організація, створена власниками квартир і нежитлових приміщень для управління спільним майном будинку.

**JWT** – JSON Web Token. Стандарт для безпечної передачі даних між сторонами у вигляді JSON-об'єкта, що підписаний цифровим підписом.

**CSRF** – Cross-Site Request Forgery. Тип атаки на вебдодатки, який полягає в тому, що зловмисник змушує користувача виконувати небажані дії в додатку, де той вже авторизований.

**XSS** – Cross-Site Scripting. Уразливість вебдодатків, за якої зловмисник може вставити шкідливий код (зазвичай JavaScript) у сторінку, яку бачать інші користувачі.

**API** – Application Programming Interface. Інтерфейс програмування додатків, який дозволяє одному програмному забезпеченню взаємодіяти з іншим.

**UI** – User Interface. Інтерфейс користувача, що забезпечує взаємодію людини із системою.

**UX** – User Experience. Досвід користувача при взаємодії з інформаційною системою чи програмним забезпеченням.

**REST** – Representational State Transfer. Архітектурний стиль для створення вебсервісів, який базується на використанні HTTP-запитів для доступу до ресурсів.

**JSON** – JavaScript Object Notation. Формат передачі даних, який використовується для обміну інформацією між клієнтом і сервером.

**HTTPS** – HyperText Transfer Protocol Secure. Протокол передачі гіпертексту з використанням захищеного з'єднання через SSL/TLS.

# ВСТУП

Ефективне управління багатоквартирними будинками завжди було викликом для ОСББ (об’єднань співвласників багатоквартирних будинків) в Україні. З огляду на зростаючу кількість нових житлових комплексів і модернізацію існуючих будівель, вимоги до прозорості, оперативності та якості адміністративного міського обслуговування, стають дедалі актуальнішими. Сучасне життя диктує потребу в цифрових рішеннях, які б дозволяли вирішувати складні завдання управління будинком максимально швидко та зручно.

Виклики стають ще більш серйозними в умовах нестабільності енергопостачання, що є актуальною проблемою для України з 2022 року. Через аварійні або планові відключення світла мешканці залишаються без інформації про стан будинку та терміни відновлення електропостачання. Це створює значний дискомфорт і призводить до втрати довіри до органів управління. З огляду на це, виникає потреба у системі, яка могла б оперативно інформувати співвласників про зміни в режимах енергопостачання.

Одним із ключових аспектів актуальності розробки є недостатня кількість ефективних і персоналізованих інформаційних систем для ОСББ. Існуючі рішення зазвичай охоплюють лише окремі аспекти управління, як-от ведення фінансової звітності чи організація голосувань. Однак комплексних систем, що включають модулі для комунікації, фінансового обліку, моніторингу стану будинкової інфраструктури та функції сповіщення про відключення світла, практично немає.

Розроблена система також відповідає потребам автора, оскільки ОСББ, до якого він належить, стикається з типовими для багатьох об’єднань проблемами. Це, зокрема, складнощі з обміном інформацією, відсутність доступу до оперативних даних про стан будинку, непрозорість у фінансових питаннях. Запропонована інформаційна система не лише допоможе вирішити ці проблеми, а й стане базою для створення аналогічних рішень у майбутньому.

Очікувана користь від впровадження системи значна:

* **Підвищення прозорості**. Співвласники матимуть доступ до повної фінансової звітності ОСББ у режимі реального часу.
* **Оперативність комунікації**. Мешканці зможуть швидко отримувати важливу інформацію через сповіщення, зокрема про відключення електроенергії чи аварійні ситуації.
* **Покращення управління інфраструктурою**. Система дозволить моніторити стан ліфтів, систем опалення та водопостачання.
* **Економія часу та ресурсів**. Завдяки автоматизації рутинних процесів знизяться адміністративні витрати ОСББ.

Актуальність цієї роботи також визначається тим, що розробка такої системи сприятиме вирішенню не лише поточних, а й довгострокових завдань ОСББ, забезпечуючи комфорт мешканців і прозорість у всіх аспектах управління будинком.

* + 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ

## 1.1 ОСББ: визначення, важливість та актуальність

**ОСББ** є формою організації, що створюється власниками квартир для ефективного управління спільною власністю багатоквартирного будинку та забезпечення належного утримання і благоустрою. Мешканці таких будинків часто не усвідомлюють, що, окрім своєї квартири, вони є співвласниками загальних частин будівлі, таких як дах, сходи, ліфти та інженерні мережі. Це є основою для необхідності об’єднання зусиль у створенні та управлінні ОСББ.

В Україні законодавство чітко визначає, що багатоквартирні будинки є спільною сумісною власністю всіх власників квартир. Власники зобов’язані утримувати ці будівлі в належному стані, що включає в себе як загальний благоустрій, так і регулярний технічний контроль. Відповідно до статті 382 Цивільного кодексу України, власник на правах спільної власності володіє всіма елементами, що необхідні для функціонування житла.

Актуальність ОСББ пов’язана з тим, що традиційні системи управління будинками, такі як ЖЕКи, показали свою неефективність через відсутність прозорості та контролю. Вони самостійно визначають перелік і вартість послуг, що призводить до відсутності впливу з боку мешканців на обсяг та якість послуг. Істотним є і те, що, не маючи системи контролю, ЖЕКи часто не забезпечують належного обслуговування будинків.

За даними Держстату у 2022, в Україні діє 28377 ОСББ, і їхня кількість продовжує зростати. Такі об’єдання відрізняється від ЖЕКів тим, що дає змогу мешканцям самостійно вибирати, хто і коли буде ремонтувати їм під’їзд та внутрішні комунікації, прибирати та вивозити сміття, латати діряві дахи, а головне – за скільки, адже співвласники можуть укладати договори з компаніями-виконавцями таких послуг [1]. Це дає змогу мешканцям будинку заощаджувати до 20-30% накладних витрат, які додатково закладає ЖЕК у структуру тарифу за квартплату (див. рис 1.1).



Рис 1.1 – Фінансові переваги ОСББ

**Інформаційні системи управління ОСББ** стають важливими інструментами, що дозволяють власникам краще взаємодіяти, контролювати витрати та ефективно планувати ремонти і обслуговування. Вони забезпечують прозорість у фінансових потоках, можливість голосування та прийняття рішень, знижують адміністративні витрати і полегшують комунікацію між власниками.

## 1.2 Загальні положення

Інформаційні системи (ІС) є основою сучасного управління і комунікацій у різних секторах, включаючи і житлово-комунальне господарство. Вони сприяють автоматизації процесів, підвищенню ефективності управління та забезпеченню прозорості у веденні справ. Інформаційна система управління та комунікації для ОСББ дає змогу здійснювати контроль за станом майна, координувати дії мешканців та керівництва, а також управляти фінансами та ресурсами.

Такі системи повинні бути адаптовані до потреб власників, сприяти їхній участі в управлінні спільним майном, а також забезпечувати інтеграцію з іншими системами житлово-комунального сектора.

### 1.2.1 Змістовний опис і аналіз предметної області

Інформаційні системи для ОСББ мають на меті автоматизацію процесів, що включають фінансове управління, облік витрат, планування ремонту, комунікацію між власниками, контроль за наданням послуг та інші аспекти житлового обслуговування. У рамках таких систем особливу увагу приділяють:

1. **Фінансовому управлінню**: автоматизація ведення бухгалтерії, облік витрат і доходів, генерація квитанцій та звітності.
2. **Управлінню обслуговуванням та ремонтом**: планування робіт, контроль за виконанням, збереження історії обслуговування.
3. **Комунікації та голосуванню**: онлайн-платформи для обговорення питань, голосування, обмін інформацією та ухвалення рішень.
4. **Прозорості та звітності**: можливість власників перевіряти фінансові звіти, отримувати інформацію про витрати та плани.

Отже, ОСББ є колективами співвласників, які не лише мають права, але й обов'язки щодо утримання майна та забезпечення його належного стану. Інформаційні системи повинні відповідати сучасним вимогам до безпеки, зручності у використанні та адаптивності до змін.

### 1.2.2 Вимоги та виклики

Вимоги до інформаційних систем управління та комунікації для ОСББ включають високий рівень захисту даних та конфіденційності. Це вимагає впровадження сучасних методів шифрування та захисту інформації для забезпечення безпеки даних. Також важливим є створення зручного інтерфейсу, який має бути інтуїтивно зрозумілим і зручним для користувачів із різним рівнем технічної підготовки. Система повинна бути адаптивною та масштабованою, щоб мати змогу змінюватися відповідно до зростання кількості співвласників та інтегруватися з іншими програмними рішеннями, що використовуються в сфері управління [2].

У сучасних умовах цифровізація діяльності ОСББ стає важливою складовою управління спільним майном. Це стало особливо актуальним в умовах пандемії COVID-19 та війни, коли фізичні зустрічі та взаємодія у звичайному режимі стають небезпечними або ускладненими.

Під час пандемії онлайн-інструменти стали необхідністю для забезпечення безперервності роботи ОСББ. Віртуальні зустрічі та обговорення допомогли власникам квартир залишатися на зв'язку та активно брати участь у прийнятті рішень, навіть перебуваючи вдома. Це дозволило уникнути необхідності фізичних зборів, що ускладнює процес у випадках карантинних обмежень. Крім того, електронні системи дали змогу оперативно обмінюватися інформацією, що зробило процес управління більш прозорим і доступним.

Цифрові технології особливо важливі у періоди економічної нестабільності та соціальних змін. В умовах війни, коли частина мешканців змушена виїжджати за межі країни, онлайн-платформи та системи дають можливість брати участь у голосуваннях і ухваленні рішень, не залишаючи місця проживання. Це дає змогу забезпечити більшу залученість усіх співвласників і зберігати ефективність управлінських процесів.

Додатково, використання цифрових рішень сприяє зменшенню витрат, на паперові документи, послуги пошти, а також фізичні зустрічі. Це дає змогу економити час і ресурси, сприяючи більшій економічній ефективності управління ОСББ.

Застосування електронних підписів та онлайн-опитувань дає змогу оперативно збирати голоси та ухвалювати рішення у коротші строки, що особливо важливо в умовах обмеженого часу та швидких змін у зовнішньому середовищі. Усе це підвищує рівень довіри серед мешканців і покращує управлінські процеси [3].

Серед основних викликів, з якими стикаються розробники та впроваджувачі таких систем, є проблеми інтеграції. Необхідно забезпечити сумісність і можливість обміну даними з іншими платформами, такими як міські системи. Витрати на модернізацію та підтримку системи також є значним викликом, оскільки необхідно регулярно оновлювати програмне забезпечення та підтримувати інфраструктуру.

Впровадження інформаційних систем управління та комунікації для ОСББ значно підвищує ефективність управління. Автоматизація процесів дає змогу зменшити трудозатрати, мінімізувати помилки, що виникають через людський фактор, і підвищити загальну ефективність роботи. Окрім того, така система покращує прозорість, адже власники отримують доступ до актуальної інформації, що запобігає можливим зловживанням і непотрібним витратам. Завдяки цим системам підвищується залученість власників, оскільки вони мають можливість активно брати участь у процесах прийняття рішень і комунікації між мешканцями та управлінням.

## 1.3 Підґрунтя для розробки основних модулів системи

Розробка інформаційної системи для управління та комунікації в ОСББ вимагає глибокого аналізу як теоретичних основ, так і практичних потреб користувачів. Підхід до створення основних модулів системи базується на інтеграції наукових знань, практичних вимог і технічних можливостей, що забезпечує високу ефективність і релевантність рішення. В цьому параграфі розглянемо вимоги для розробки модулів.

### 1.3.1 Система управління фінансами

Система управління фінансами є ключовим елементом у роботі об'єднання співвласників багатоквартирних будинків, оскільки саме від неї залежить прозорість фінансових процесів, ефективність управління спільними ресурсами та довіра мешканців до діяльності керівництва. Ефективна фінансова система дає змогу не лише вести облік доходів і витрат, але й забезпечує автоматизацію рутинних операцій, надаючи мешканцям доступ до актуальної інформації про стан рахунку, історію платежів і фінансові звіти.

Актуальність системи управління фінансами пояснюється кількома важливими чинниками. По-перше, прозорість фінансових операцій зменшує ризики корупції та усуває непорозуміння між мешканцями щодо розподілу коштів. Це є важливим аспектом у сучасних умовах, коли довіра до управління колективним бюджетом є однією з головних умов ефективного функціонування ОСББ. По-друге, автоматизація процесів, як-от розрахунок боргів чи формування платіжних документів, значно знижує навантаження на бухгалтерів та адміністраторів ОСББ, дозволяючи їм зосередитися на стратегічних питаннях. Впровадження цифрових платіжних систем, таких як онлайн-банкінг чи інтеграція з API платіжних платформ, робить процес оплати більш зручним для мешканців і мінімізує затримки у виконанні транзакцій.

Однак на шляху до створення дієвої фінансової системи ОСББ стоять певні проблеми. Традиційний ручний облік, який часто ведеться у таблицях Excel або навіть у паперових журналах, є трудомістким, схильним до помилок і малоефективним. Брак прозорості у фінансових операціях може викликати недовіру мешканців, особливо якщо витрати не документуються належним чином або мешканці не мають доступу до детальної інформації. Ще однією важливою проблемою є складність інтеграції з сучасними платіжними системами, через що мешканці стикаються з незручностями при оплаті послуг, а адміністрація — з труднощами у відстеженні боргів. Неефективна комунікація між ОСББ і мешканцями також є значним викликом: наприклад, нерегулярні або несвоєчасні нагадування про оплату часто призводять до накопичення боргів.

Розв'язання цих проблем можливе через впровадження сучасних автоматизованих систем обліку. Такі системи здатні автоматично обробляти дані про доходи, витрати та залишки, надаючи доступ до інформації адміністраторам і мешканцям через зручний інтерфейс. Інтеграція з платіжними платформами, такими як LiqPay, Stripe чи PayPal, дає змогу мешканцям здійснювати платежі онлайн, а системі — автоматично фіксувати надходження та оновлювати статуси рахунків у реальному часі. Генерація звітів є ще одним важливим компонентом таких систем: за допомогою інструментів для створення PDF-файлів або інтеграції з Google Sheets можна легко формувати фінансові звіти, які доступні для перегляду мешканцями.

Не менш важливим є модуль сповіщень, який автоматично надсилає нагадування про борги, зміни тарифів чи нові платіжні реквізити через email або push-сповіщення. Завдяки цьому мешканці завжди в курсі актуальної фінансової ситуації. Доповнення системи інструментами аналітики дає змогу аналізувати витрати, прогнозувати бюджет та візуалізувати фінансові дані за допомогою інтерактивних графіків.

З технічної точки зору, система управління фінансами для ОСББ може бути реалізована як вебзастосунок з використанням сучасних технологій. Інтерфейс користувача можна створити на базі React чи Vue.js, а серверну частину — за допомогою Node.js (Express) або Python (Django/Flask). База даних, як-от PostgreSQL або MongoDB, зберігатиме інформацію про транзакції, а Redis може використовуватися для кешування частих запитів, наприклад, для перегляду залишків на рахунку. Інтеграція з платіжними системами через API забезпечить автоматичну обробку транзакцій, а безпека системи гарантується через використання SSL/TLS і токенів авторизації (JWT).

Система управління фінансами в ОСББ дає змогу підвищити ефективність роботи об'єднання, зробити процеси прозорішими й сприяти зростанню довіри між мешканцями та керівництвом. У сучасних умовах впровадження таких рішень є не лише доцільним, а й необхідним для переходу ОСББ до більш сучасних і ефективних моделей управління.

### 1.3.2 Система комунікації між мешканцями ОСББ

Система комунікації між мешканцями є однією з ключових складових інформаційних систем для ОСББ, оскільки ефективне спілкування є основою для вирішення спільних питань, обміну інформацією та побудови дружнього мікроклімату в будинку. Така система повинна забезпечувати швидкий та зручний обмін повідомленнями між співвласниками, керівництвом ОСББ та іншими зацікавленими сторонами, наприклад, сервісними компаніями чи постачальниками послуг.

Актуальність системи комунікації зростає в умовах сучасного ритму життя, коли мешканці часто не мають можливості відвідувати загальні збори чи обговорювати важливі питання особисто. У багатоквартирних будинках з великою кількістю мешканців комунікація без спеціальних інструментів може бути хаотичною, що ускладнює організацію спільних дій. Цифрові рішення дають змогу централізувати інформацію, підвищити оперативність обміну даними та мінімізувати непорозуміння.

До системи комунікації висуваються такі основні вимоги, як доступність, простота використання та багатофункціональність. Система повинна забезпечувати зручний обмін текстовими повідомленнями, можливість прикріплення файлів (наприклад, платіжних квитанцій, фото чи документів), а також функціонал для створення групових чатів або форумів. Важливим є забезпечення безпеки даних: повідомлення мають бути захищеними, а доступ до системи – обмеженим для сторонніх осіб.

Однак реалізація такої системи може супроводжуватися проблемами. Наприклад, не всі мешканці активно користуються сучасними цифровими технологіями, особливо люди похилого віку. Крім того, можуть виникати конфлікти між користувачами через анонімність у системі або незрозумілий інтерфейс. Також важливо забезпечити регулярне оновлення контактів мешканців та управління доступом, щоб уникнути використання застарілих даних.

Можливі рішення включають створення централізованого вебдодатку чи мобільного додатку для ОСББ. Така система може мати декілька ключових функцій: спільний чат для обговорення загальних питань, можливість надсилання приватних повідомлень між мешканцями, групові сповіщення від керівництва ОСББ та функціонал для створення оголошень. Для зручності мешканців система може бути інтегрована з популярними месенджерами, такими як Telegram або Viber, через боти, які надсилатимуть сповіщення про новини, збори чи важливі події.

З технічної точки зору, для реалізації можна використати такі фреймворки, як React Native для створення мобільного додатку або Vue.js для вебдодатку. Серверна частина може бути розроблена на базі Node.js чи Python (Django/Flask), з використанням бази даних MongoDB чи PostgreSQL для збереження історії повідомлень та контактів користувачів. Для забезпечення безпеки даних можна використовувати токени доступу (JWT) та шифрування повідомлень за допомогою протоколів SSL/TLS.

Запровадження системи комунікації дозволить ОСББ підвищити рівень взаємодії між мешканцями, забезпечить прозорість обміну інформацією та сприятиме формуванню активної спільноти. Завдяки доступу до оперативних повідомлень та можливості швидкого вирішення питань, мешканці почуватимуться залученими до управління спільним майном, що позитивно вплине на загальну атмосферу у будинку.

### 1.3.3 Система моніторингу інфраструктури

Система моніторингу інфраструктури є важливим компонентом інформаційної системи для ОСББ, яка дає змогу стежити за станом ключових об’єктів спільного користування та забезпечувати їхню належну роботу. До таких об’єктів належать пасажирські та вантажні ліфти, системи водопостачання, теплопостачання, освітлення місць загального користування, а також елементи безпеки, такі як камери спостереження та системи пожежної сигналізації. Окрім цього, особливо актуальним є включення до системи можливості відображення актуальних графіків відключення електроенергії, що є важливим в умовах нестабільного енергопостачання.

Актуальність такої системи зумовлена необхідністю своєчасного реагування на технічні несправності, запобігання аварійним ситуаціям та забезпечення безпеки мешканців. Наприклад, оперативна інформація про зупинку ліфта чи несправності в системах водопостачання дає змогу значно швидше усунути проблему, ніж це можливо без автоматизованого моніторингу. Відображення графіків відключення електроенергії, з урахуванням змін або позапланових аварій, забезпечує мешканцям комфорт та можливість краще адаптувати свій розпорядок дня.

Основними вимогами до системи моніторингу є реальність часу, доступність інформації для відповідальних осіб і мешканців, а також можливість інтеграції з іншими компонентами системи управління ОСББ. Зокрема, система повинна підтримувати автоматичне сповіщення про критичні ситуації (наприклад, аварійні вимкнення), збереження історії даних для аналізу ефективності інфраструктури та доступ до графіків енергопостачання з урахуванням регіональних особливостей.

Проблеми, які можуть виникати під час реалізації такої системи, включають складність інтеграції з існуючим обладнанням, особливо якщо воно застаріле або не підтримує сучасних протоколів передачі даних. Також важливим є забезпечення безпеки інформації, адже дані про функціонування інфраструктури повинні бути захищені від несанкціонованого доступу. Не менш складним викликом є адаптація системи для різних ОСББ, враховуючи різноманітність технічного обладнання та його стан.

Можливі рішення включають використання інтернету речей (IoT) для моніторингу інфраструктури. Наприклад, датчики, підключені до мережі, можуть відстежувати стан ліфтів, освітлення чи систем водопостачання та передавати дані до централізованої бази. У якості платформи для реалізації можна використовувати Node.js або Python з фреймворком Flask чи Django для серверної частини, а MongoDB або PostgreSQL для збереження даних. Для відображення графіків відключення світла можна інтегрувати дані з офіційних сайтів енергокомпаній або використовувати API, якщо такі доступні.

Для інтерфейсу системи може бути використано React або Angular, що дозволить створити інтуїтивно зрозумілий вебдодаток або мобільний додаток. На інтерфейсі користувачі зможуть переглядати актуальну інформацію про стан інфраструктури, графіки відключення світла, а також отримувати сповіщення про надзвичайні ситуації. Інтеграція з мапами дозволить наочно демонструвати зони відключення електроенергії, якщо такі функції підтримуються.

Реалізація такої системи забезпечить не лише своєчасний моніторинг та управління інфраструктурою ОСББ, але й підвищить рівень довіри мешканців до керівництва ОСББ, оскільки вони будуть завжди поінформовані про стан спільного майна та перебіг відновлювальних робіт. Це також сприятиме більш ефективному використанню ресурсів та плануванню заходів з модернізації будинкових систем.

### 1.3.4 Система колективного прийняття рішень

Система колективного прийняття рішень є фундаментальною складовою ефективного управління об'єднанням співвласників багатоквартирних будинків. Вона забезпечує організацію процесів обговорення, голосування та впровадження рішень, які стосуються спільного майна, фінансів чи організаційних питань. Така система сприяє підвищенню залученості мешканців, прозорості ухвалення рішень та уникненню конфліктів між співвласниками.

Актуальність системи колективного прийняття рішень зумовлена низкою чинників. Сучасне законодавство вимагає, щоб важливі питання, пов’язані з управлінням багатоквартирними будинками, вирішувалися шляхом колективного голосування. Це включає питання затвердження кошторисів, вибору підрядників, змін тарифів на обслуговування чи розв'язання конфліктів. У традиційному форматі зібрань ОСББ процес голосування часто супроводжується труднощами: не всі мешканці можуть бути присутніми через зайнятість, а фізичні обговорення можуть затягуватися, викликаючи розбіжності. Застосування цифрових рішень дає змогу вирішити ці проблеми, надаючи можливість дистанційної участі та автоматизації багатьох аспектів.

Основними вимогами до системи є зручність, доступність і відповідність законодавчим нормам. Інтерфейс повинен бути зрозумілим навіть для користувачів з базовими цифровими навичками. Крім того, система має враховувати специфіку голосування в ОСББ, де рішення можуть прийматися як більшістю голосів, так і на основі площі квартир, які належать співвласникам. Важливою є й безпека, адже результати голосування повинні бути захищені від фальсифікацій, а персональні дані мешканців — конфіденційними.

Попри очевидні переваги цифровізації, процеси колективного прийняття рішень у ОСББ стикаються з низкою проблем. Однією з основних є низький рівень залученості мешканців, оскільки не всі готові активно брати участь у справах будинку. Також проблема може виникнути через брак технічних знань або доступу до цифрових інструментів у певної частини співвласників. Крім того, складність організації прозорого й достовірного голосування вимагає впровадження сучасних рішень, які мають враховувати всі можливі сценарії, включаючи помилки користувачів або спроби маніпуляції.

Одним із можливих рішень є впровадження онлайн-платформи для голосування та обговорення питань. Така система може включати декілька ключових модулів: форум для обговорення, голосування з різними типами вибору (одне або декілька рішень), архів результатів та функціонал для автоматичної генерації протоколів засідань. Система може передбачати багаторівневий доступ: наприклад, адміністратори зможуть створювати нові теми для голосування, а мешканці — брати участь у голосуванні та залишати коментарі.

Для забезпечення доступності система повинна бути розроблена як вебдодаток з адаптивним дизайном, щоб користувачі могли працювати з нею як з комп’ютера, так і з мобільних пристроїв. Можлива інтеграція з месенджерами, такими як Telegram чи Viber, дозволить надсилати сповіщення про нові голосування або обговорення. Ще одним важливим елементом є система аутентифікації, наприклад через логін та пароль, SMS-код або електронний підпис, що забезпечить безпеку доступу.

З технічної точки зору, для реалізації системи можна використати фреймворки, такі як React або Angular для фронтенду та Node.js або Django для бекенду. База даних, наприклад PostgreSQL, дозволить зберігати результати голосувань, обговорення та дані користувачів. Для верифікації особистості мешканців та забезпечення захисту даних можуть використовуватися шифрування (SSL/TLS) і токени доступу (JWT).

Загалом, система колективного прийняття рішень значно полегшує процес управління ОСББ, забезпечує прозорість і довіру мешканців до роботи адміністрації. Її впровадження дозволить усім співвласникам брати участь у важливих рішеннях, підвищуючи ефективність управління будинком і покращуючи якість життя мешканців.

### 1.3.5 Автентифікація та управління користувачами

Автентифікація та управління користувачами є основою будь-якої сучасної інформаційної системи, що забезпечує доступ до функцій та ресурсів відповідно до ролей і прав користувачів. У контексті ОСББ така система дає змогу розмежувати доступ між мешканцями, членами правління та адміністративним персоналом, забезпечуючи як безпеку даних, так і зручність використання.

Актуальність цієї системи зумовлена необхідністю захисту конфіденційної інформації, включаючи фінансові дані, документи та особисту інформацію мешканців. Наприклад, мешканці повинні мати доступ до своїх рахунків чи сповіщень, але не до внутрішніх управлінських документів правління. Члени правління, у свою чергу, потребують доступу до аналітичних даних і засобів прийняття рішень, але їхні дії мають бути прозорими для інших членів ОСББ.

Основними вимогами до системи є забезпечення безпечної автентифікації, можливість управління ролями користувачів, ведення історії дій (аудит) і підтримка різних типів доступу. Наприклад, звичайні мешканці можуть мати роль "користувач", тоді як голова правління отримає розширені привілеї для адміністрування. Система також має підтримувати можливість тимчасового доступу для зовнішніх підрядників (наприклад, технічного персоналу).

Проблематика полягає в тому, що навіть базова система автентифікації повинна відповідати сучасним стандартам безпеки, зокрема захищати паролі через хешування (наприклад, за допомогою алгоритмів bcrypt або Argon2) та забезпечувати захист від атак на основі перебору. Іншою складністю є забезпечення гнучкості управління ролями користувачів у системі, яка може змінюватися відповідно до потреб ОСББ.

Можливі рішення включають використання технологій JSON Web Token (JWT) для реалізації токенів доступу та оновлення. Це дає змогу створити безпечну систему автентифікації для веб- і мобільних додатків. Для управління базою даних користувачів можна використовувати MongoDB або PostgreSQL, залежно від потреб у масштабованості та складності запитів. Для безпеки та зручності система може підтримувати двофакторну автентифікацію (2FA), наприклад, через SMS або додатки на зразок Google Authenticator.

Для фронтенда можна застосовувати React або Angular для створення інтуїтивного інтерфейсу для входу та управління профілями. Адміністраторська панель, яка забезпечує управління ролями, може бути реалізована через інтегровані бібліотеки управління доступом, такі як RBAC (Role-Based Access Control).

Управління користувачами також передбачає можливість самостійної реєстрації та відновлення доступу. Наприклад, мешканці можуть реєструватися через форму, підтверджуючи свою адресу електронною поштою або номером телефону. Для зручності додаток може підтримувати інтеграцію з системами "єдиного входу" (Single Sign-On), що дає змогу входити через Google, Facebook чи інші популярні сервіси.

Реалізація системи автентифікації та управління користувачами дозволить створити безпечне середовище для спільної роботи мешканців та адміністрації ОСББ. Це сприятиме прозорості, підвищить довіру до системи управління та зробить користування платформою більш зручним і ефективним для всіх учасників.

### 1.4 Аналіз існуючих рішень і реалізацій

В умовах сучасних викликів, пов'язаних із управлінням багатоквартирними будинками та зростанням потреб у прозорості, ефективності та зручності для мешканців, розвиток інформаційних систем для управління ОСББ набирає великої популярності. Науковий підхід до створення та впровадження таких систем ґрунтується на аналізі існуючих рішень і практичному використанні сучасних технологій, що дозволяють поліпшити управління ресурсами, оптимізувати фінансові процеси та підвищити рівень комунікацій.

Існуючі системи обліку, комунікації та управління дозволяють зрозуміти потреби ОСББ, їхні сильні сторони та обмеження, а також визначити напрямки для вдосконалення. З точки зору практичної значущості, інтеграція цифрових рішень дає змогу знизити витрати на управління, збільшити прозорість фінансових потоків та полегшити доступ до інформації для всіх зацікавлених сторін. З наукової точки зору, дослідження та розробка таких систем є прикладом застосування інноваційних методів у галузі інформаційних технологій і економіки для покращення управління і прийняття рішень у сфері житлово-комунального господарства.

Головною метою аналізу існуючих рішень є виявлення оптимальних практик, що можуть бути застосовані у вашій системі, а також визначення недоліків і можливостей для інновацій. На основі проведеного аналізу можна розробити більш ефективні рішення, які зможуть покращити поточну ситуацію в управлінні ОСББ, включаючи інтеграцію нових функцій та можливість адаптації під специфічні потреби українських ОСББ.

### 1.4.1 Аналіз сервісу «ОСББ – online»

«OSBB Online» — це платформа, що автоматизує управління ОСББ, зосереджуючи увагу на зручності та функціональності. Вона надає мешканцям особисті кабінети, які дозволяють переглядати стан платежів, отримувати новини та доступ до важливих документів. Завдяки фінансовому модулю платформа спрощує облік платежів, забезпечує генерацію звітів і контроль заборгованостей, сприяючи прозорості в управлінні [4].

Комунікаційні можливості сервісу дуже обмежені та майже не включають засоби для розсилки оголошень, сповіщень та доступу до загальної документації, що ускладнює обмін інформацією між адміністрацією ОСББ та мешканцями. Крім того, платформа не дає змоги організовувати онлайн-голосування, через що мешканці не можуть брати участь у прийнятті важливих рішень дистанційно, що особливо актуально у сучасних умовах.

Однією з найбільших переваг OSBB Online є його спрямованість на фінансову систему. Вона пропонує готові рішення для ведення фінансових операцій та обліку платежів. Платформа також акцентує увагу на прозорості, адже всі учасники ОСББ можуть перевірити витрати та надходження через загальні звіти.

Платформа працює у форматі SaaS (Software as a Service), що дає змогу ОСББ не перейматися питаннями технічного обслуговування або встановлення програмного забезпечення. Дані зберігаються у хмарному середовищі, що забезпечує доступ до них у будь-який час та з будь-якого пристрою.

OSBB Online пропонує гнучкі тарифні плани, що дозволяють ОСББ вибрати відповідний рівень послуг залежно від кількості квартир у будинку та функцій, які необхідні для роботи (див. рис.1.2).

Попри розвинутий функціонал в сфері фінансів і документації, сервіс має багато обмежень. Він не орієнтований на моніторинг стану інфраструктури та має обмежений модуль комунікації між користувачами у вигляді форуму, на якому об'яви можуть робити лише адміністратори ОСББ, інтеграції з українськими реаліями, наприклад, урахування графіків відключення електроенергії, використання електронних підписів наразі є відсутнє.



Рис. 1.2 – ОСББ-online

### 1.4.2 Аналіз сервісу МОЄОСББ

"МОЄОСББ" є однією з платформ, що орієнтована на спрощення управлінських процесів у багатоквартирних будинках через об'єднання співвласників багатоквартирних будинків (ОСББ). Цей сервіс пропонує базові функції для ведення фінансового обліку, комунікацій і організації управління житловими фондами. Основна мета платформи – створити комфортні умови для взаємодії мешканців із правлінням ОСББ, підвищити рівень автоматизації та знизити навантаження на адміністрацію [5].

Сервіс забезпечує створення особистих кабінетів для мешканців, що дає змогу зручно відслідковувати оплату комунальних послуг, отримувати важливі сповіщення і доступ до документації. Автоматизація фінансових операцій допомагає спрощувати процеси обліку, зменшуючи ризик помилок та скорочуючи час на адміністративні завдання. Додаткові функції, такі як організація онлайн-голосувань, сприяють залученню мешканців до ухвалення рішень, що покращує комунікацію та прозорість.

Незважаючи на переваги, "МОЄОСББ" має ряд обмежень, які знижують ефективність сервісу у повсякденному використанні. Одним із суттєвих недоліків є відсутність розширеного моніторингу стану інфраструктури будинку, такого, як облік технічного стану ліфтів або систем опалення, що обмежує можливості для комплексного управління і обслуговування. Також відзначаються обмеження в аналітичних функціях – платформа не забезпечує глибокого аналізу витрат і доходів, що ускладнює прийняття обґрунтованих управлінських рішень.

Крім того, сервіс має недостатній рівень персоналізації функцій, що може бути проблематичним для ОСББ з унікальними потребами. Недостатньо зручний інтерфейс і не завжди зрозуміле налаштування функціоналу можуть створювати труднощі для менш досвідчених користувачів, що підвищує ризик помилок та знижує загальну ефективність системи. Не всі функції автоматизації, як-от сповіщення про планові ремонти або аварії, реалізовані на належному рівні, що впливає на безперебійність управлінських процесів (див. рис.1.3).

Вартість користування платформою, хоч і є доступною, не компенсує деякі обмеження у функціоналі. У разі потреби в додаткових модулях або розширених опціях сервіс може стати значно дорожчим, що ставить під сумнів ефективність витрат на нього в умовах обмежених фінансових ресурсів ОСББ.



Рис. 1.3 – МОЄОСББ

### 1.4.3 Порівняльна характеристика

У цьому підрозділі представлено порівняльний аналіз сучасних інформаційних систем для управління та комунікації в об’єднаннях співвласників багатоквартирних будинків (ОСББ). Розглянуто три основні рішення: **«**Інформаційна система управління та комунікації для ОСББ» (яка була розроблена в межах цього дослідження), «ОСББ-online» та «МОЄОСББ» (див Таблиця 1.1).

Таблиця 1.1 - Порівняльна характеристика





# ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі магістерської дисертації проведено аналіз сучасного стану проблематики, пов'язаної з управлінням і комунікацією в об'єднаннях співвласників багатоквартирних будинків . Дослідження предметної області та оцінка існуючих рішень створили основу для визначення вимог і викликів, що стоять перед розробкою інтегрованої інформаційної системи.

У підрозділі 1.1 розглянуто сутність ОСББ, їх важливість у сучасному управлінні житловими комплексами, а також обґрунтовано актуальність теми дослідження. Проаналізовано основні проблеми, які постають перед ОСББ, зокрема недостатній рівень прозорості фінансових операцій, складнощі в комунікації між співвласниками та адміністрацією, а також обмежені можливості для колективного прийняття рішень.

У підрозділі 1.2 деталізовано особливості предметної області, зокрема ключові вимоги до функціоналу інформаційної системи та виклики, пов’язані з її реалізацією. Визначено, що для ефективного функціонування ОСББ потрібна автоматизація управлінських процесів, яка включає фінансові, комунікаційні, інфраструктурні та організаційні аспекти.

У підрозділі 1.3 сформульовано підґрунтя для розробки основних модулів системи. Описано ключові компоненти, зокрема:

* **Система управління фінансами**, яка забезпечує автоматизацію платежів і ведення фінансової звітності;
* **Система комунікації**, що створює платформу для ефективного обміну інформацією між мешканцями та адміністрацією;
* **Система моніторингу інфраструктури**, яка дозволяє оперативно оцінювати технічний стан об'єктів;
* **Система колективного прийняття рішень**, що полегшує процес голосування серед співвласників;
* **Система автентифікації та управління користувачами**, яка забезпечує безпеку доступу та захист даних.

У підрозділі 1.4 проведено аналіз існуючих рішень, таких як сервіси «ОСББ-online» і «МОЄОСББ». Виявлено їхні сильні сторони, обмеження та функціональні прогалини. Зокрема, розглянуто недостатню інтеграцію фінансових модулів, обмежені можливості моніторингу інфраструктури та слабкі механізми комунікації, що стало підґрунтям для формування вимог до нової інформаційної системи.

Порівняльна характеристика показала, що жоден із наявних сервісів повністю не відповідає сучасним потребам ОСББ, що підтвердило доцільність і необхідність створення нової системи.

Таким чином, у першому розділі обґрунтовано актуальність дослідження, сформульовано вимоги до системи та проаналізовано існуючі рішення, що дало змогу визначити оптимальні напрями розробки інформаційної системи управління та комунікації для ОСББ.

# 2. ПОСТАНОВКА ЗАДАЧІ ТА РОЗРОБКА МЕТОДУ

## 2.1 Опис постановки задачі

Постановка задачі є важливим етапом створення інформаційної системи для управління та комунікації в ОСББ. Мета системи — створення інтегрованої платформи, яка забезпечить ефективну взаємодію між мешканцями багатоквартирних будинків, допоможе організовувати фінансові процеси, слідкувати за станом інфраструктури, а також сприятиме прийняттю колективних рішень через голосування і обговорення. Основним завданням є автоматизація рутинних процесів, що звільнить час адміністрації та мешканців, підвищуючи загальну ефективність управління.

Однією з ключових цілей є забезпечення прозорості фінансових операцій. Це передбачає автоматизацію обліку доходів і витрат, створення детальних фінансових звітів, які дозволять мешканцям краще розуміти стан фінансів ОСББ та зміцнювати довіру до управлінських процесів. Система повинна забезпечити швидкий обмін інформацією між мешканцями та адміністрацією, створюючи умови для ефективної комунікації. Це включає можливість отримання сповіщень, обміну повідомленнями та доступу до актуальної інформації у реальному часі.

Важливим аспектом є надання можливості для організації зборів і голосувань, що дасть змогу мешканцям брати участь у прийнятті рішень і обговореннях важливих питань, що стосуються управління будинком. Автоматизація збору даних про споживання ресурсів і їх аналіз дозволить оптимізувати витрати та планувати майбутні дії на основі даних.

Також система повинна мати можливість зберігати та управляти документами, такими як протоколи зборів, акти виконаних робіт, договори та інші важливі матеріали, забезпечуючи доступ до них на запит. Це допоможе зберігати необхідну документацію в організованому вигляді і спрощувати процеси, що раніше потребували значних витрат часу та зусиль.

Актуальність розробки цієї системи зумовлена численними проблемами, з якими стикаються ОСББ. Зокрема, це недостатня прозорість фінансових звітів, неефективна комунікація між мешканцями та адміністрацією, відсутність зручних інструментів для управління інфраструктурою та залучення мешканців до прийняття рішень. Інформаційна система має вирішити ці проблеми, забезпечуючи зручний і ефективний інструмент для управління будинком і підтримки комунікацій.

Система повинна бути зрозумілою та доступною для різних категорій користувачів, з надійною системою захисту даних та можливістю інтеграції з іншими платформами та сервісами, щоб забезпечити відповідність місцевим вимогам і стандартам. Впровадження такої системи дозволить значно покращити управлінські процеси в ОСББ, підвищити рівень комунікації та залучення мешканців, а також зменшити витрати на обслуговування і підвищити довіру до управління. У результаті створена система стане потужним інструментом для сучасного управління, який відповідатиме потребам українських ОСББ і сприятиме розвитку прозорого та ефективного управління багатоквартирними будинками.

Розробка системи, яка відповідатиме зазначеним вимогам, має забезпечити:

* Зниження адміністративних витрат завдяки автоматизації процесів.
* Підвищення прозорості та надійності фінансових операцій.
* Поліпшення комунікації та зменшення часу на ухвалення рішень через інтерактивні інструменти.
* Сприяння розвитку організованого колективного управління.

Реалізація цієї системи дозволить підвищити ефективність управління, забезпечить прозорість процесів і зміцнить довіру мешканців до адміністрації ОСББ, що є важливим кроком у розвитку сучасного управління житловими комплексами.

## 2.2 Структура бізнес-процесів

Структура бізнес-процесів в інформаційній системі управління та комунікації для ОСББ визначає взаємодію між різними учасниками системи та послідовність виконання завдань, що забезпечують ефективне управління, комунікацію і підтримку життєдіяльності багатоквартирного будинку [7] (див. рис. 2.1). Основні бізнес-процеси можна умовно розділити на кілька ключових категорій:

1. **Фінансові процеси**. Фінансові бізнес-процеси включають управління бюджетом ОСББ, облік доходів і витрат, розрахунок комунальних послуг, сплату внесків, а також автоматизоване формування фінансових звітів. Ці процеси дозволяють забезпечити прозорість фінансових операцій, знижуючи ризик помилок і зловживань, і дають змогу мешканцям відслідковувати стан фінансів в режимі реального часу.
2. **Процеси комунікації та взаємодії з мешканцями**. Сюди входять процеси обміну повідомленнями між мешканцями та адміністрацією ОСББ, повідомлення про важливі події та система збору і обробки запитів від мешканців. Використання WebSocket у цьому випадку дає змогу миттєво доставляти сповіщення та інформацію, що підвищує ефективність комунікації.
3. **Процеси управління інфраструктурою**. Ці процеси охоплюють моніторинг технічного стану інфраструктури будинку, управління обслуговуванням і ремонтами. Також ця система має змогу збирати актуальні дані і попереджати щодо вимкнень світла у будинку.
4. **Процеси збору та аналізу даних**. Збір даних про споживання ресурсів (електроенергії, води, газу), використання приміщень і споживання послуг є важливою складовою управлінської діяльності. Аналіз цих даних дає змогу знижувати витрати, прогнозувати потреби та створювати умови для ефективного планування та оптимізації ресурсів.
5. **Процеси голосування та колективного прийняття рішень**. Оскільки ОСББ є організацією, де важливо враховувати думки всіх членів, система повинна включати процеси, які дозволяють організовувати збори, проводити голосування за важливими питаннями (наприклад, зміну умов обслуговування, затвердження бюджету) і приймати колективні рішення. Завдяки інструментам електронного голосування, процеси ухвалення рішень стають швидшими та зручнішими для мешканців.
6. **Процеси управління документами та звітністю**. Обробка та збереження документів, таких як договори, протоколи зборів, акти виконаних робіт, є необхідною частиною управлінської діяльності ОСББ. Система має забезпечувати можливість збереження та доступу до важливих документів у цифровому форматі, а також автоматичне формування звітів і підготовку документів для подання в контролюючі органи.

Кожен з цих бізнес-процесів має важливу роль у загальній структурі діяльності ОСББ і забезпечує інтеграцію функцій, що дозволяють підвищити рівень управління, прозорість та ефективність діяльності. Гнучкість системи та її здатність адаптуватися під змінні вимоги мешканців та нормативні вимоги створюють умови для сталого розвитку та покращення умов проживання в багатоквартирних будинках.

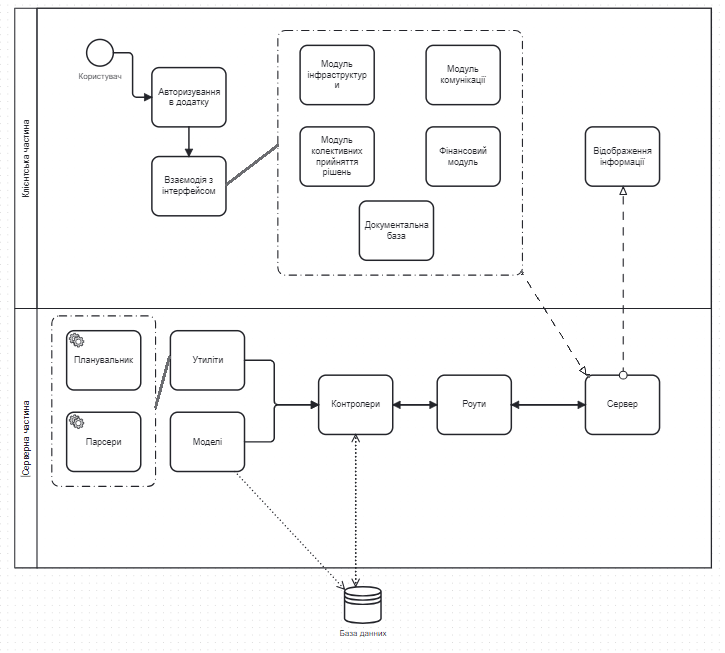


Рис. 2.1 – Бізнес модель

## 2.3 Вибір архітектури.

Проектування архітектури програмного забезпечення (ПЗ) є важливим етапом розробки, що слідує за аналізом і формуванням вимог до системи. Основною задачею цього процесу є перетворення вимог до системи на вимоги до програмного забезпечення та створення архітектурної структури, яка відповідає цим вимогам. Архітектура системи визначається як структурна схема, що включає компоненти, які взаємодіють через інтерфейси. Ці компоненти можуть складатися з послідовностей більш дрібних складових і взаємодіяти між собою за допомогою визначених інтерфейсів [6].

Проектування архітектури базується на використанні загальних довідників, класифікаторів і методологій. Також в цьому процесі ідентифікуються ключові частини системи, включаючи як уже існуючі програмні продукти, так і нові компоненти, а також ті, що часто використовуються в інших проєктах.

Побудова архітектури системи включає в себе визначення цілей і завдань системи, її вхідних і вихідних даних, а також декомпозицію на підсистеми та модулі. Основні рішення щодо структури системи зазвичай приймаються групою архітекторів і аналітиків, після чого окремі частини передаються для реалізації невеликим групам розробників.

Архітектурне проектування може здійснюватися за різними методами, такими як стандартизоване, об’єктно-орієнтоване, компонентне проектування та інші. Кожен метод пропонує свої підходи до створення концептуальних і об'єктних моделей за допомогою конструктивних елементів, таких як блок-схеми, графи, структурні діаграми тощо.

Один із традиційних підходів, що часто використовується, — загальносистемний. Він включає чотири основні рівні:

1. **Системні компоненти**: ці компоненти забезпечують взаємодію з периферійними пристроями комп’ютерів, такими як принтери, клавіатури, сканери. Вони є основою для створення операційних систем і базових інтерфейсів.
2. **Загальносистемні компоненти**: компоненти, що взаємодіють із сервісами середовища роботи, такими як операційні системи, СУБД, системи управління мережами тощо.
3. **Специфічні компоненти прикладної області**: компоненти, що реалізують функції, пов’язані з певними бізнес-завданнями або специфічними цілями системи.
4. **Прикладні програмні системи**: програми, які виконують завдання обробки інформації для користувачів, такі як офісні системи або системи бухгалтерського обліку.

При створенні архітектури програмного забезпечення система розглядається як об'єднання компонентів третього рівня з доступом до компонентів першого і другого рівнів. Архітектурне проектування зосереджене на розробці елементів середнього рівня, встановленні їхніх взаємозв'язків і визначенні вхідних та вихідних даних.

Результатом цього процесу є створення архітектури і інфраструктури, що включають у себе набір об'єктів, які можна використовувати для створення конкретної архітектурної схеми відповідно до середовища, в якому система буде виконуватись.

В рамках цього проєкту було обрано **модульну архітектуру**, яка є важливою складовою системи, що використовує **RESTful API** для взаємодії між клієнтською та серверною частинами [8]. Цей підхід дає змогу чітко розділити систему на логічно незалежні компоненти, які можна розробляти та тестувати окремо. Модульність забезпечує простоту підтримки і можливість масштабування системи, що є важливими факторами для ефективної роботи програмного забезпечення.

Використання **RESTful API** дає змогу реалізувати універсальний і структурований доступ до ресурсів, підтримуючи основні **CRUD**-операції. Це значно спрощує інтеграцію з іншими системами і фронтендом, забезпечуючи зручність у роботі з даними [9].

Усі компоненти системи — від контролерів і моделей до проміжного програмного забезпечення (middleware) і утиліт — взаємодіють через чітко визначені інтерфейси, що дає змогу зменшити складність коду та полегшити його тестування. Вибір модульного підходу відображає високий рівень організації коду та здатність підтримувати та адаптувати систему в умовах постійних змін і розвитку.

Використання модульної архітектури та застосування RESTful API не тільки підвищує ефективність розробки та тестування, але й сприяє створенню більш гнучкої і масштабованої системи. Цей підхід демонструє потенціал застосування сучасних принципів проектування ПЗ у розробці комплексних систем, що можуть швидко адаптуватися до змінюваних вимог і забезпечити зручність для кінцевого користувача.

## 2.4 Серверна частина

Розробка серверної частини проєкту є ключовим елементом, що забезпечує логіку роботи системи, зберігання та обробку даних, а також взаємодію з клієнтською частиною. Для реалізації серверної частини було використано набір сучасних інструментів, модулів і технологій, які вже змогли продемонструвати свою ефективність та надійність, де кожен із яких обрано з урахуванням специфіки та вимог проекту, і в результаті об’єднались в одну унікальну та уніфіковану систему.

## 22.4.1 Середовище програмування

Для розробки серверної частини було обрано інтегроване середовище розробки Visual Studio Code (VS Code), яке є одним із найкращих рішень для розробників. Це безкоштовне та багатофункціональне середовище розробки, що відзначається своєю гнучкістю та широкими можливостями налаштування [10].

Visual Studio Code пропонує широкий вибір розширень для підтримки JavaScript та Node.js, які є основними технологіями для серверної частини. Завдяки цьому середовище автоматично пропонує автозавершення коду, аналіз помилок і оптимізацію. Ці можливості значно пришвидшують процес розробки та підвищують ефективність роботи над проєктом.

Також не менш важливим фактором є наявність вбудованих інструментів для відлагодження коду, що дає змогу швидко знаходити й усувати помилки. Відлагодження прямо в середовищі розробки спрощує тестування серверної частини.

Ще однією вагомою перевагою є легка інтеграція Visual Studio Code із системами контролю версій, такими як Git. Завдяки вбудованим інструментам є можливість зручно відслідковувати зміни, створювати коміти, працювати з гілками та віддаленими репозиторіями. Це забезпечує кращий контроль над версіями коду та відладку, та можливість завжди відновити код у разі кризисної ситуації (див. рис. 2.2).

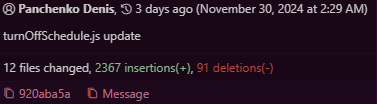


Рис. 2.2 - Приклад використання Git в проекті.

Для середовища виконання коду JavaScript було обрано Node.js. Node.js є популярним середовищем виконання JavaScript на стороні сервера, яке дає змогу створювати високопродуктивні вебдодатки. Node.js має численні переваги, які роблять його привабливим вибором для розробки вебдодатків. Однією з головних переваг є висока продуктивність, яка досягається завдяки неблокуючій моделі введення/виведення. Завдяки цьому Node.js здатний обробляти велику кількість одночасних запитів з мінімальними витратами ресурсів, що є критично важливим для високонавантажених додатків, таких як чат-додатки або системи реального часу [11].

Ще однією важливою перевагою є використання єдиної мови програмування — JavaScript — як на клієнтській, так і на серверній частині. Це значно спрощує розробку та інтеграцію, знижуючи складність комунікації між фронтендом і бекендом. Node.js також підтримує масштабування, що дає змогу ефективно розширювати додатки відповідно до зростаючих потреб користувачів. Завдяки кластеризації, яка дає змогу розподіляти навантаження між кількома ядрами процесора, і неблокуючій обробці запитів, Node.js відмінно підходить для створення масштабованих додатків, які можуть обробляти великий обсяг даних і користувачів.

Велика екосистема модулів і бібліотек через npm (Node Package Manager) є ще однією суттєвою перевагою. Це дає змогу розробникам швидко інтегрувати сторонні бібліотеки для різних завдань, що значно спрощує розробку та скорочує час на реалізацію проєкту (див. рис. 2.3).

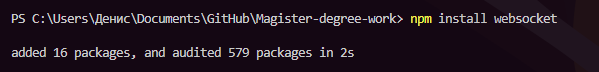


Рис.2.3 - Приклад використання публічних пакетів npm

Кожен користувач Node.js може самостійно розробити і запушити власний пакет для публічного доступу, за допомогою команди npm publish.

Також node packet manager дає змогу користуватись вже попередньо створеними, але приватними модулями, які зберігаються у вигляді посилання на архіви з вихідним кодом, наприклад у GitHub (див. рис. 2.4). У package.json така залежність виглядає наступним чином:

"dependencies": {

**"jwt-middleware":** "https://github.com/meloncholic-san/**jwt-middleware**/tarball/main"

}

Рис.2.4 - Приклад додавання своїх модулів, інтегруючи їх в систему

## 2.4.2 Express.js

**Express.js** є мінімалістичним і водночас потужним фреймворком для Node.js, який чудово підходить для створення вебдодатків і RESTful API. У рамках цього проєкту Express.js був обраний як основа серверної частини завдяки його простоті, гнучкості та широким можливостям налаштування [12].

Однією з ключових причин використання Express.js є його мінімалістичний підхід, що дає змогу зосередитися на основних аспектах розробки, таких як налаштування маршрутизації та обробка HTTP-запитів. У поєднанні з Node.js це забезпечує високу продуктивність та ефективність додатку [13].

Express.js забезпечує гнучкість у налаштуванні. Це дає змогу легко інтегрувати сторонні модулі для додавання необхідних функцій, таких як jsonwebtoken для аутентифікації, або підключати middleware для обробки запитів і відповідей. Такий підхід значно спрощує роботу з API та забезпечує масштабованість додатка [14].

У межах проєкту Express.js забезпечує створення чіткого і гнучкого серверного середовища, що відповідає вимогам сучасних вебдодатків.

Приклад підключення і функціонування Express.js в системі:

Для початку ми імпортуємо Express.js у наш Node.js-додаток:

const express = require**(**'express'**)**;

Далі ініціалізуємо додаток:

const app = express**()**;

На прикладі реалізації функціоналу платіжної системи ми підключаємо маршрутизатор, визначений у файлі paymentRoutes.js:

const paymentRoutes = require**(**'./routes/paymentRoutes'**)**; *// Імпорт роутів*

app.use**(**'/api/payments', paymentRoutes**)**; *// Регіструємо маршрути оплати*

Маршрутизатор paymentRoutes включає два основних маршрути для роботи з платежами: створення платежу та перевірка його статусу.

Далі, для того, щоб сервер почав приймати запити, реалізовано наступний підхід. Основний файл сервера server.js є центральною точкою входу для додатка. Після налаштування необхідних залежностей і підключення бази даних, сервер починає слухати запити на вказаному порту, отриманому з файлу .env. Такий підхід забезпечує гнучкість, портативність, безпеку і надійність у різних середовищах і дає змогу уникнути конфліктів із зайнятими портами. Також це дає змогу швидко розгорнути та інтегрувати сервер у майже будь-якому хмарному середовищі. У разі відсутності налаштувань у середовищі, використовується порт за замовчуванням 5000.

*// Запуск сервера*

app.listen**(**PORT, () => {

    console.log**(**`Сервер запущено на порту ${PORT}`**)**;

}**)**;

Тепер сервер готовий до обробки запитів.

## 2.4.3 База данних MongoDB

MongoDB був вибраний за базу данних завдяки своїй природі нереляційної бази даних, що дає змогу зручно зберігати дані у форматі JSON-подібних документів. Цей формат забезпечує гнучкість при роботі з даними, оскільки кожен документ може мати унікальну структуру, що дає змогу адаптувати базу даних до змінюваних потреб додатка. Завдяки такому підходу, розробка нових функцій і адаптація до нових вимог стають значно простішими і швидшими, порівняно з традиційними реляційними системами, де потрібно змінювати структуру таблиць і здійснювати складні операції для інтеграції нових даних [15].

Однією з ключових переваг MongoDB є її здатність працювати з великими обсягами інформації швидко та ефективно. База даних використовує індексацію та можливість горизонтального масштабування, що дає змогу підтримувати високу продуктивність навіть за значних навантажень. Це особливо важливо для проектів, де потрібно забезпечити безперебійну роботу з великою кількістю користувачів та даних, таких як запити на отримання та збереження інформації про фінансові операції, голосування, чат-повідомлення та інші важливі функції.

MongoDB забезпечує значну гнучкість завдяки можливості зберігати складні структури даних, такі як вкладені об'єкти та масиви. Це дає змогу ефективно працювати з даними, які не потребують суворої схемної структури, що робить розробку більш швидкою та менш схильною до помилок.

Іншою важливою перевагою є інтеграція MongoDB з Node.js через бібліотеку Mongoose. Mongoose є потужним інструментом, що дає змогу спростити роботу з базою даних завдяки можливості створення моделей, валідації даних і застосування проміжних обробників (middleware). Це забезпечує не тільки легкість у написанні коду, але й підвищену безпеку завдяки вбудованим механізмам перевірки даних перед їх збереженням у базі.

Однією з ключових особливостей Mongoose є можливість створення моделей, які представляють документи в колекціях. Модель є конструкцією, яка на основі заданої схеми дає змогу працювати з даними: створювати нові документи, шукати, оновлювати або видаляти їх. Наприклад, модель Ad, що використовується у цьому проекті, дає змогу створити документ оголошення, зберігати його в MongoDB та здійснювати запити для отримання, оновлення або видалення даних (див. рис. 2.5).

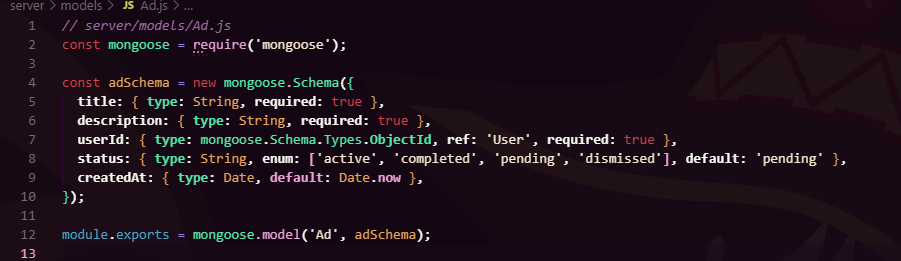


Рис.2.5 - Приклад створення моделі “Ad” Mongoose

Схеми в Mongoose визначають структуру документа, типи даних для кожного поля та додаткові параметри, такі як обов’язковість (required), значення за замовчуванням (default), унікальність (unique) і перерахування можливих значень (enum). Це дає змогу створювати надійні моделі, де кожне поле відповідає певним критеріям. Наприклад, у схемі для оголошення можна встановити, що поле title має бути рядком і обов’язковим, а поле status може мати одне з визначених значень, таких як active, completed, pending, або dismissed.

Mongoose також підтримує потужну систему валідації, яка допомагає уникнути збереження некоректних даних. Наприклад, можна додати валідацію, щоб перевірити, чи відповідає формат електронної пошти, чи є значення у відповідному діапазоні тощо. Це значно спрощує та покращує роботу, оскільки валідація даних здійснюється автоматично перед тим, як документ буде збережено в базі.

Ще однією перевагою використання Mongoose є можливість створення так званих "популяцій" (populate). Це дає змогу заповнювати посилання на інші документи з інших колекцій, що корисно для роботи з реляційними структурами даних. Наприклад, у моєму проекті userId у документі оголошення є посиланням на користувача, що дає змогу отримати всі оголошення певного користувача разом із його даними.

Mongoose також підтримує проміси та асинхронні функції, що дає змогу легко інтегрувати асинхронні запити до коду на основі async/await. Це покращує читабельність коду і спрощує обробку асинхронних операцій.

Іншою важливою особливістю є підтримка плагінів, які дозволяють розширювати функціонал моделей, додаючи нові можливості, наприклад, автозбільшення лічильників, перевірки на унікальність, налаштування схем тощо.

Mongoose є потужним інструментом, якщо працювати з Node.js і MongoDB, забезпечуючи зручний і гнучкий спосіб роботи з даними. Вона допомагає зменшити обсяг коду, спрощує інтеграцію з іншими бібліотеками та забезпечує зручний інтерфейс для роботи з базою даних, що підвищує ефективність розробки.

В цілому, використання MongoDB значно спрощує роботу з даними, дозволяючи зосередитись на розробці функціоналу та інтеграції, не турбуючись про складності зі зміною схеми бази даних або масштабуванням системи при росту обсягу даних. Це робить таке рішення гнучким, надійним і готовим до роботи у сучасних умовах, де швидкість і масштабування є основними критеріями ефективної роботи додатків.

**WebSockets,** були використані у магістерській дисертації**,** які забезпечують двосторонній зв’язок між клієнтом і сервером у реальному часі. З технічної точки зору, WebSocket — це протокол зв’язку, який працює поверх TCP і дає змогу встановити постійне підключення між клієнтом і сервером. На відміну від HTTP, де кожен запит вимагає нового підключення, WebSocket дає змогу один раз встановити з'єднання і використовувати його для постійного обміну даними [16].

У традиційній вебархітектурі клієнт надсилає запити на сервер і чекає на відповідь. Якщо потрібне оновлення даних у реальному часі (наприклад, нові повідомлення в чаті або оновлення статусу), клієнт мусить постійно опитувати сервер, що створює значні витрати на трафік і серверні ресурси. WebSockets вирішують цю проблему, надаючи можливість сервера самостійно надсилати оновлення клієнту, коли це необхідно.

Розглянемо практичну реалізацію встановлення WebSocket-з'єднання:

**1. Ініціалізація WebSocket-з'єднання**

* Коли користувач відкриває сторінку, клієнтський додаток (React) встановлює з'єднання з сервером за допомогою WebSocket.

const ws = new WebSocket**(**'ws://localhost:5000'**)**;//або фактична адреса сервера у хмарі

* Сервер приймає з'єднання та зберігає інформацію про підключеного клієнта.

ws.onopen = () => {

    console.log**(**'Підключено до вебсокета'**)**;

};

**2. Обмін даними між клієнтом і сервером**

* Сервер може надсилати повідомлення клієнтам у будь-який момент. Це дає змогу отримувати оновлення в реальному часі (наприклад, нові оголошення).

Сервер надсилає дані у форматі JSON, наприклад:

{

"type": "newAd",

"ad": {

"\_id": "123",

"title": "Нове оголошення",

"description": "Опис",

"createdAt": "2028-12-01"

}

}

* Клієнт отримує повідомлення через метод onmessage, розпізнає їх тип і відповідно оновлює інтерфейс.

Клієнт обробляє це повідомлення:

ws.onmessage = (event) => {

    const message = JSON.parse**(**event.**data)**;

    if **(**message.**type** === 'newAd'**)** {

        setAds**(**(prevAds) => **[**...prevAds, message.**ad])**;

    }

};

**3. Динамічне оновлення інтерфейсу**

* Дані, отримані через WebSocket, додаються до стану (state) React-компонента.
* React автоматично оновлює UI відповідно до нового стану.

setAds**(**(prevAds) => **[**...prevAds, message.**ad])**;

* Нове оголошення додається до списку:
* React оновлює список оголошень на сторінці в реальному часі.

**4. Закриття з'єднання**

* Якщо користувач закриває сторінку, з'єднання WebSocket потрібно закрити, щоб уникнути витоків ресурсів.
* При розмонтуванні компонента викликається:

return () => {

    ws.close**()**;

};

## 2.5 Клієнтська частина

Клієнтська частина системи? була реалізована за допомогою сучасних вебтехнологій, що забезпечують зручність, високу продуктивність і розширюваність інтерфейсу користувача. Для розробки були використані такі фреймворки, бібліотеки та інструменти, кожен із яких відіграє ключову роль у функціональності та зручності системи.

React — це JavaScript-бібліотека, яка дає змогу створювати компонентний інтерфейс користувача. Причиною вибору React є його ефективність, висока продуктивність і широка підтримка. React дає змогу створювати багаторазові компоненти, що спрощує розробку складних додатків [17].

React допомогає позбутися проблеми, де дані, які компонент отримує від свого батьківського елемента, передаються у вигляді атрибутів, схожих на ті, що використовуються у HTML-тегах. Ці дані є лише для читання всередині компонента, тобто їх не можна змінювати безпосередньо. Для оновлення або передачі змін компонент має викликати функцію зворотного виклику (callback), яку передає батьківський елемент. Такий підхід називається "згори вниз — дані, знизу вгору — події".

Також React працює з концепцією віртуального DOM, що є оптимізованою копією звичайного DOM. Коли відбуваються зміни у компонентах, React порівнює нову версію віртуального DOM із попередньою (процес "diff") і вносить мінімальні зміни у реальний DOM браузера. Це дає змогу швидко оновлювати сторінку без зайвих витрат ресурсів. Наприклад, замість повного оновлення всього інтерфейсу змінюється лише та частина, яка зазнала змін [18].

React не обмежується лише рендерингом HTML у веббраузері. Наприклад, у Facebook за допомогою цієї бібліотеки створюються інтерактивні графіки, які відображаються через елементи <canvas>. Крім того, такі компанії, як Netflix і PayPal, застосовують ізоморфний рендеринг, що дає змогу створювати однаковий HTML-код як на стороні сервера, так і клієнта. Це забезпечує швидке завантаження сторінок і покращений досвід стилкористувача, особливо для складних або масштабних додатків.

У 2012 році Марк Цукерберг зазначив: «Найбільша помилка, яку ми зробили як компанія — це надто велика ставка на HTML5, на відміну від нативних додатків». І вже у 2015 році, Facebook випустила React, який дає змогу генерувати елементи інтерфейсу напряму з фонового Javascript потоку. Тобто, це дає змогу створити користувацький інтерфейс без необхідності використовувати HTML [19].

Такий підхід дає змогу уникати роботи з "монолітними" HTML-файлами. Завдяки JavaScript потоку, можна створювати складну логіку для управління станом компонентів та взаємодії з даними. Замість написання довгого коду HTML, можно будувати динамічні частини інтерфейсу з невеликих частин, що спрощує підтримку та масштабування проєкту. HTML не призначений для виконання подібних завдань — він лише забезпечує статичну структуру. React дає змогу уникнути використання сторонніх технологій для реалізації динамічної поведінки, що скорочує кількість залежностей.

Для реалізації клієнтської частини додатку було використано зручний і актуальний CSS3, що дає змогу стилізувати інтерфейс користувача.

Одним із ключових аспектів стилізації є адаптивність. Завдяки модульному підходу та використанню сучасних можливостей CSS, таких як flexbox, grid, а також медіа-запитів, інтерфейс виглядає привабливо на всіх пристроях — від настільних комп’ютерів до мобільних телефонів. Для інтерактивних елементів, таких як форми або календарі, використовуються бібліотеки, які підтримують вбудовану стилізацію. Наприклад, **React-datepicker** дає змогу швидко налаштовувати компоненти вибору дат без необхідності додаткових стилів, що значно прискорює розробку.

Також у проєкті застосовуються динамічні стилі, які змінюються залежно від стану компонентів. Це реалізовано за допомогою JavaScript, що дає змогу інтегрувати стилі прямо в логіку додатка. Крім того, бібліотека **React-toastify**, яка відповідає за повідомлення користувача, забезпечує як функціональність, так і естетику без необхідності самостійного написання стилів для цих елементів.

Загалом використання модульного підходу, готових бібліотек і динамічної стилізації допомогло створити сучасний, зручний і адаптивний інтерфейс, який відповідає всім можливим вимогам проєкту ОСББ. Це забезпечує зручність для користувачів і значно полегшує підтримку та розвиток системи в майбутньому.

**Axios** — це потужна бібліотека JavaScript, що дає змогу здійснювати HTTP-запити з клієнтської частини вебдодатка. Вона стала популярною завдяки своїй простоті використання, гнучкості та підтримці сучасних функцій, які значно полегшують процес інтеграції клієнта з сервером. Axios дає змогу не тільки надсилати запити до серверного API, але й обробляти відповіді, що надходять від сервера, у зручному форматі [20].

Однією з основних переваг Axios є автоматичне перетворення відповіді у формат JSON. Це забезпечує швидке й безпомилкове оброблення даних, що надходять від сервера, без необхідності ручного парсингу відповіді. Бібліотека також підтримує відправлення даних у форматі application/json, що часто використовується в REST API.

Axios має зручний синтаксис для виконання асинхронних запитів, що дає змогу з легкістю інтегрувати його з асинхронними функціями та промісами JavaScript. Це значно полегшує написання чистого та зрозумілого коду, що є важливим аспектом при розробці складних вебдодатків. Наприклад, з Axios можна легко додати обробку помилок за допомогою catch, що дає змогу ефективно реагувати на різні типи помилок, від помилок мережі до непередбачених помилок сервера.

Завдяки можливостям конфігурації, Axios спрощує реалізацію механізмів автентифікації, зокрема через токени. Це дає змогу додати заголовки для запитів, включаючи токени авторизації, що забезпечують безпеку при взаємодії з захищеними ресурсами серверної частини. Наприклад, у випадках використання JWT (JSON Web Token), можна додати токен у заголовок Authorization, що спрощує управління сесіями користувачів.

Ще однією корисною особливістю Axios є підтримка відмінної роботи з промісами та асинхронними запитами, що особливо корисно для реалізації функцій, що потребують завантаження даних з сервера перед відображенням на сторінці. Це знижує складність коду та допомагає уникнути глибоких вкладених колбеків (callback hell), забезпечуючи зрозумілу структуру програмного коду.

Завдяки широкому спектру можливостей конфігурації, таких як тайм-аути запитів, перехоплювачі запитів (interceptors) для обробки запитів і відповідей, Axios є універсальним інструментом для розробників, що працюють з вебдодатками, які потребують взаємодії з зовнішніми API. Ця бібліотека здатна покрити потреби багатьох сценаріїв, від простих GET-запитів до складних запитів з авторизацією і обробкою помилок.

У рамках цієї системи, Axios був використаний для інтеграції клієнтської частини з сервером, забезпечуючи зручну взаємодію і обробку даних. Це сприяло підвищенню ефективності розробки та поліпшенню загальної якості коду завдяки його простоті і потужним можливостям.

На прикладі хука для завантаження очікуючих оголошень з серверу при зміні користувача, розберемо, як Axios взаємодії з додатком (див. рис.2.6):

*// Завантаження очікуючих оголошень тільки для адміністраторів*

    useEffect**(**() => {

        if **(**user && user.**role** === 'admin'**)** {

            const fetchPendingAds = async () => {

                try {

                    const response = await axios.get**(**'http://localhost:5000/api/ads/pending', {

**headers:** {

**Authorization:** `Bearer ${token}`,

                        },

                    }**)**;

                    setPendingAds**(**response.**data)**;

                } catch **(**err**)** {

                    console.error**(**'Помилка при завантаженні очікуючих оголошень:', err**)**;

                    setError**(**'Не вдалося завантажити очікуючі оголошення'**)**;

                }

            };

            fetchPendingAds**()**;

        }

    }, **[**user**])**; *// Залежність від user*

Рис. 2.6 – Приклад використання Axios

1. **Запит до сервера**: За допомогою axios.get() надсилається GET-запит на сервер за вказаною URL-адресою, в даному випадку — 'http://localhost:5000/api/ads/active'. Цей запит намагається отримати дані про активні оголошення. Або ж фактична адреса сервера, якщо він розташований у хмарному середовищі.
2. **Передача заголовків**: У заголовках запиту передається токен авторизації Authorization: Bearer ${token}. Це потрібно для того, щоб сервер знав, хто робить запит, і чи є у нього необхідні права для доступу до цих даних. Токен є частиною механізму автентифікації на сервері, що дає змогу зберігати безпеку.
3. **Асинхронність запиту**: Запит виконується асинхронно, що означає, що програма не зупиняється під час отримання відповіді від сервера. Використовується async/await, що дає змогу очікувати відповідь без блокування основного потоку виконання.
4. **Обробка відповіді**: Якщо запит успішний, в response.data буде міститися відповідь сервера, яка потім передається в стан компоненту через setAds(response.data), що оновлює інтерфейс.
5. **Обробка помилок**: Якщо запит не вдається, наприклад, через відсутність інтернет-з'єднання або помилку на сервері, в блоці catch користувач сповіщається про помилку через setError('Не вдалося завантажити активні оголошення').
6. **Завершення запиту**: Після того, як запит завершено, незалежно від того, чи був він успішним, виконується блок finally, стан loading становиться false, що вказує на завершення процесу завантаження.

# ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі магістерської дисертації проведено детальний аналіз та розробку методів і засобів, необхідних для створення інформаційної системи управління та комунікації для об’єднань співвласників багатоквартирних будинків. Результати виконаного дослідження дали змогу розробити оптимальні технічні та архітектурні рішення, що відповідають сучасним вимогам до ефективного управління багатофункціональними інформаційними системами.

У підрозділі 2.1 визначено основну мету та завдання, які вирішуються в межах розробки системи. Зокрема, було обґрунтовано необхідність автоматизації процесів управління ОСББ, що включають фінансову діяльність, моніторинг технічного стану інфраструктури, комунікацію між співвласниками та адміністрацією, а також прийняття колективних рішень через голосування. У підрозділі 2.2 побудовано структуру бізнес-процесів, які лягли в основу функціоналу інформаційної системи.

У підрозділі 2.3 обґрунтовано вибір архітектури для розробки інформаційної системи. Вибір модульного підходу відображає високий рівень організації коду та здатність підтримувати та адаптувати систему в умовах постійних змін і розвитку, а використання RESTful API дає змогу реалізувати універсальний і структурований доступ до ресурсів, підтримуючи основні CRUD-операції. У підрозділі 2.4 детально описано серверну частину системи. Для розробки обрано середовище програмування Node.js, яке забезпечує високу продуктивність, асинхронність виконання операцій та зручність роботи з великою кількістю користувачів. Використання Express.js як основного серверного фреймворку дозволяє оптимізувати розробку за рахунок його легкості та багатофункціональності. У підрозділі 2.5 розглянуто клієнтську частину системи, яка є інтерфейсом взаємодії кінцевого користувача з усіма модулями системи. Таким чином, у другому розділі дисертації обґрунтовано вибір архітектурних та технологічних рішень, а також описано методи їх реалізації. Виконані роботи створюють наукову та технічну основу для вдосконалення ключових процесів створення інформаційних систем ОСББ та подальшого впровадження системи в реальну практику.

1. РЕАЛІЗАЦІЯ МЕТОДІВ

Як вже було обговорено в попередніх розділах, систему можна умовно розділити на чотири повноцінні модулі: фінансовий модуль, модуль комунікації, модуль голосувань, та модуль моніторингу інфраструктури. У цьому розділі буде продемонстровано, як ці модулі інтегровані в систему, з використанням сучасних методів і технологій. Для кожного модуля буде представлено код, який відображає ключові функції та процеси, реалізовані у проекті.

## 3.1 Автентифікація

Перш ніж отримати доступ до захищених ресурсів проходить автентифікація користувачів за допомогою JWT токенів.

JWT (JSON Web Token) — це популярний стандарт для реалізації аутентифікації і передачі інформації між клієнтом і сервером у форматі JSON. Він забезпечує безпечний обмін даними завдяки використанню цифрових підписів, що гарантують цілісність інформації та можливість перевірки джерела [21].

Після того як користувач входить у систему, сервер генерує токен, який потім відправляється клієнту та зберігається на сервері або базі даних. Цей токен додається до заголовків HTTP-запитів під час взаємодії з API, що дає змогу серверу перевіряти права доступу користувача.

JWT складається з трьох основних частин:

1. **Header (Заголовок)**: Визначає тип токена (зазвичай "JWT") і алгоритм шифрування (наприклад, HMAC SHA256).
2. **Payload (Дані)**: Містить інформацію про користувача (наприклад, його ідентифікатор, роль, дату закінчення дії тощо).
3. **Signature (Підпис)**: Створюється шляхом використання секретного ключа та зазначеного алгоритму для підписування header і payload. Це забезпечує цілісність токена і гарантує, що його не було змінено.

Але, як показує практика, захист завдяки JWT не дає 100% гарантії безпеки даних. Для того, щоб покращити JWT авторизацію, в проекті було використано:

1. **Шифрування Payload**: Хоча підписування забезпечує цілісність даних, сам payload не шифрується. Для захисту конфіденційної інформації можна використовувати додаткове шифрування.
2. **Терміни дії та оновлення**: Потрібно використовувати короткий термін дії токена і оновлювати його за допомогою механізму оновлення токенів (refresh tokens).
3. **Безпечне зберігання**: Уникнення зберігання токенів у місцях, які можуть бути доступні зловмисникам, наприклад, у localStorage. Замість цього використовується HttpOnly cookies, які не доступні через JavaScript.

Для того, щоб позбутися цих недоліків запропоновано такий метод шифрування JWT токенів:

В функції encryptedPayload конфедеційні дані спочатку перетворюються у JSON-строку за допомогою JSON.stringify (див. рис. 3.1).

*// Шифруємо конфіденційну інформацію*

        const encryptedPayload = encryptData**(**{

**firstName:** user.**firstName**,

**lastName:** user.**lastName**,

**apartmentNumber:** user.**apartmentNumber**,

**password:** user.**password**,

        }**)**;

Рис. 3.1 – Отримання конфіденційних даних

Далі процес шифрування можна розбити на три етапи:

1. **Створення шифра:**

В функції encryptData алгоритм crypto.createCipher створює шифрувальну функцію на основі алгоритму aes-256-cbc і секретного ключа (AES\_SECRET\_KEY), де **aes-256-cbc** використовує 256-бітовий ключ (32 байти) та працює в режимі CBC, де кожен блок шифрується, враховуючи попередній (див. рис. 3.2).

1. **Процес шифрування:**

Вхідні дані шифруються блоками:

* cipher.update(data, 'utf8', 'hex'): Шифрує дані з тексту у форматі UTF-8 у шістнадцятковий (hex).
* cipher.final('hex'): Завершує шифрування і повертає останню частину зашифрованих даних.

1. **Об’єднання результату:**

* Об’єднуються результати cipher.update і cipher.final, формуючи кінцевий зашифрований текст у форматі hex.

*// Шифрування*

const encryptData = (data) => {

    const cipher = crypto.createCipher**(**'aes-256-cbc', AES\_SECRET\_KEY**)**;

    let encrypted = cipher.update**(**JSON.stringify**(**data**)**, 'utf8', 'hex'**)**;

    encrypted += cipher.final**(**'hex'**)**;

    return encrypted;

};

Рис. 3.2 – Процесс шифрування

Для того, щоб забезпечети довгостроковий термін дії JWT токену, але при цьому зберегти гарантованість безпеки, запропоновано підхід Access\Refresh токена. Умовно, цей процес теж можна поділити на три етапи:

1. **Генерація Access токена:**

Access токен генерується за допомогою jwt.sign (див. рис. 3.3), який:

* Використовує секретний ключ process.env.JWT\_SECRET.
* Включає дані accessTokenPayload.
* Має короткий термін дії (15 хвилин).

const accessToken = jwt.sign**(**accessTokenPayload, process.**env**.JWT\_SECRET, {

**expiresIn:** '15m',

}**)**;

Рис. 3.3 – Генерація Access токена

Він використовується для доступу до захищених ресурсів API.

1. **Генерація Refresh токена:**

Refresh токен генерується для довгострокової автентифікації користувача (наприклад, 7 днів):

* Використовує інший секретний ключ process.env.JWT\_REFRESH\_SECRET.
* Зберігається на сервері у змінній refreshTokens.
* Використовується для створення нового Access токена після його закінчення (див. рис. 3.4).

const refreshToken = jwt.sign**(**payload, process.**env**.JWT\_REFRESH\_SECRET, {

**expiresIn:** '7d',

}**)**;

refreshTokens.push**(**refreshToken**)**;

Рис. 3.4 - Генерація Refresh токена

1. **Безпечне зберігання**:

* **Access токен** і **Refresh токен** відправляються клієнту у вигляді cookies (див. рис. 3.5).
* Cookies налаштовуються як httpOnly, тобто вони недоступні через JavaScript (захист від XSS атак).
* Встановлюються флаги:
* secure: передача лише через HTTPS.
* sameSite: 'strict': обмежує доступ до cookies лише для запитів із того ж домену (захист від CSRF атак).
* maxAge: встановлює час життя cookies (15 хвилин для Access токена, 7 днів для Refresh токена).

res.cookie**(**'accessToken', accessToken, {

**httpOnly:** true,

**secure:** true,

**sameSite:** 'strict',

**maxAge:** 15 \* 60 \* 1000,

}**)**;

res.cookie**(**'refreshToken', refreshToken, {

**httpOnly:** true,

**secure:** true,

**sameSite:** 'strict',

**maxAge:** 7 \* 24 \* 60 \* 60 \* 1000,

}**)**;

Рис.3.5 – Приклад HTTP-only Cookies

Отже, загальний робочий процес можна представити у вигляді показаному на рис. 3.6:

1. **Авторизація:**

* Після успішного входу користувача сервер:
  + 1. Генерує Access токен (дійсний 15 хвилин).
    2. Генерує Refresh токен (дійсний 7 днів).
    3. Надсилає обидва токени клієнту через cookies.

1. **Використання Access токена:**

* Усі подальші запити клієнта до захищених API надсилаються з Access токеном (у cookies або як заголовок Authorization).

1. **Оновлення токена:**

* Коли Access токен закінчується:
  + 1. Клієнт автоматично звертається до endpoint, https:localhost:5000/api/refresh), передаючи Refresh токен.
    2. Сервер перевіряє Refresh токен, генерує новий Access токен і відправляє його клієнту.

1. **Вихід (Logout):**

* Коли користувач виходить із системи, Refresh токен видаляється зі сховища (наприклад, з бази даних).



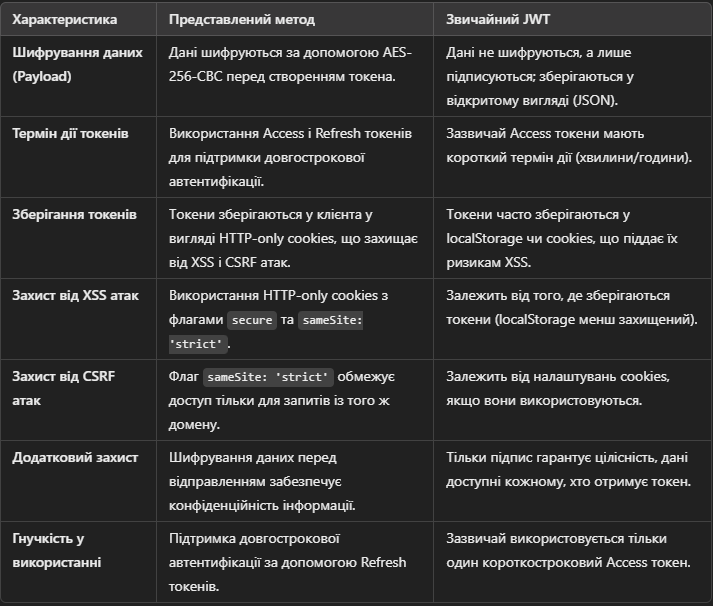
Рис. 3.6 - Загальна бізнес логіка авторизування

Переваги такої архітектури полягають у забезпеченні високого рівня безпеки. Access токен має короткий термін дії, тому навіть у разі його викрадення можливість його використання буде обмежена. Refresh токен надійно зберігається на сервері та недоступний для JavaScript, що знижує ризик його компрометації (див. таблиця 3.1).

Крім того, ця архітектура забезпечує ефективний захист від атак. Використання httpOnly cookies і короткочасних Access токенів мінімізує ризик XSS і CSRF атак, роблячи систему більш стійкою до зловмисних дій.

Окремо варто відзначити зручність для користувачів. Завдяки механізму оновлення токенів клієнт автоматично отримує новий Access токен після закінчення терміну його дії, що виключає необхідність повторного входу в систему.

Таблиця 3.1 - Порівняння запропонованого метода JWT   
та звичайного підходу



## 3.2 Модуль голосувань

Модуль голосувань створено для забезпечення можливості організації та управління голосуванням в межах системи співвласників багатоквартирних будинків. Ця функціональність дає змогу адміністраторам створювати голосування, а користувачам брати участь у ньому шляхом вибору однієї з доступних опцій. На цьому етапі детально розглянемо серверну реалізацію модуля голосувань.

Модуль включає наступні основні функціональні можливості:

1. **Створення голосування**

Адміністратори мають змогу створювати нові голосування, вказуючи питання, доступні опції для вибору, а також тривалість голосування. Перед створенням перевіряється, чи немає активного голосування. Нове голосування має статус active і може автоматично завершуватись після закінчення зазначеного часу (див. рис. 3.7).

*// Створення нового голосування*

exports.createVote = async (req, res) => {

  const { question, options, durationHours  } = req.**body**;

*// Перевірка, на активні голосування*

  const activeVote = await Vote.findOne**(**{ **status:** 'active' }**)**;

  if **(**activeVote**)** {

    return res.status**(**400**)**.json**(**{ **message:** 'Есть активное голосование. Необходимо завершить его, прежде чем начать новое.' }**)**;

  }

*// Якщо немає, створюємо нове*

  try {

*// Перевірка, чи є хоча б одна опція*

    if **(**!options || options.length < 2**)** {

      return res.status**(**400**)**.json**(**{ **message:** 'Для голосування потрібно мінімум дві опції.' }**)**;

    }

*// Розрахувати час закінчення, якщо задано*

    const expiresAt = durationHours ? new Date**(**Date.now**()** + durationHours \* 60 \* 60 \* 1000**)** : null;

*// Створюємо нове голосування*

    const newVote = new Vote**(**{

**question**,

**options**,

**status:** 'active',

**createdBy:** req.user.**id**,

**expiresAt**, *// Додаємо час закінчення, якщо задано*

    }**)**;

    await newVote.save**()**;

    res.status**(**201**)**.json**(**newVote**)**;

  } catch **(**error**)** {

    res.status**(**500**)**.json**(**{ **message:** 'Ошибка при создании голосования' }**)**;

  }

};

Рис. 3.7 – Приклад реалізації створення голосування

1. **Участь у голосуванні**

Користувачі можуть проголосувати за одну з доступних опцій у поточному активному голосуванні. Перед записом голосу перевіряється, чи користувач уже голосував, щоб запобігти повторному голосуванню. Один користувач може проголосувати лише один раз.

1. **Отримання поточного голосування**

Користувачі можуть отримати дані про активне голосування, включаючи текст питання, доступні опції, кількість голосів за кожну опцію, загальну кількість голосів та статус голосування.

1. **Перегляд історії голосувань**

Система дає змогу переглядати завершені голосування, включаючи їхні результати та деталі. Усі голосування зі статусом completed доступні в історії.

1. **Перегляд списку учасників голосування**

Функція призначена для отримання інформації про користувачів, які проголосували у певному голосуванні. Вона повертає дані про кожного, хто проголосував, включаючи його ім’я, прізвище, номер квартири та обраний варіант відповіді. Далі на клієнтській частині можна відобразити отримані дані та створити детальну інформацію про тих, хто проголосував.  
Використовується метод findById моделі Vote, щоб знайти голосування з відповідним ідентифікатором.

Для того щоб отримати повну інформацію про користувачів, які голосували, використовується метод populate, який замінює userId у масиві голосів на реальні дані користувача із пов’язаної колекції (див. рис. 3.8).

exports.getVoterDetails = async (req, res) => {

  try {

    const { voteId } = req.**params**;

    const vote = await Vote.findById**(**voteId**)**.populate**(**'votes.userId', 'firstName lastName apartmentNumber'**)**;

    if **(**!vote**)** {

      return res.status**(**404**)**.json**(**{ **message:** 'Голосування не знайдено' }**)**;

    }

*// формування списку голосувавших*

    const voters = vote.**votes**.map**(**vote => **(**{

**firstName:** vote.**userId**.**firstName**,

**lastName:** vote.**userId**.**lastName**,

**apartmentNumber:** vote.**userId**.**apartmentNumber**,

**option:** vote.**option**,

    }**))**;

    res.status**(**200**)**.json**(**voters**)**;

  } catch **(**error**)** {

    console.error**(**'Помилка при отримані данных о голосовавших:', error**)**;

    res.status**(**500**)**.json**(**{ **message:** 'Server error' }**)**;

  }

};

Рис. 3.8 – Приклад серверної реалізації перегляду списку учасників голосування

1. **Завершення голосування**

Адміністратор може вручну завершити поточне голосування, змінюючи його статус на completed.

1. **Автоматичне завершення голосування**

Автоматичне завершення голосування досягається завдяки додатковому планувальнику.

Цей модуль відповідає за автоматичне завершення голосувань у визначений час. Він використовує бібліотеку node-schedule для створення планувальника завдань, який регулярно перевіряє статус активних голосувань і завершує їх, якщо час завершення (expiresAt) минув (див. рис. 3.9).

*// autoscheduler.js*

const schedule = require**(**'node-schedule'**)**;

const Vote = require**(**'../models/Vote'**)**;

*// Функція для планування завершення поточного голосування*

const scheduleVoteCompletion = () => {

  schedule.scheduleJob**(**'\*/1 \* \* \* \*', async () => {

    const now = new Date**()**;

    const votesToComplete = await Vote.find**(**{ **status:** 'active', **expiresAt:** { **$lte:** now } }**)**;

    for **(**const vote of votesToComplete**)** {

      vote.**status** = 'completed';

      await vote.save**()**;

      console.log**(**`Голосовання "${vote.**question**}" завершено автоматично.`**)**;

    }

  }**)**;

};

module.exports = { scheduleVoteCompletion };

Рис 3.9 – Реалізація компонента автозавершення голосовування

Функція scheduleVoteCompletion використовує метод schedule.scheduleJob, щоб створити завдання, яке виконується щохвилини. У рамках цього завдання отримується поточний час за допомогою new Date(). Далі з бази даних вибираються всі активні голосування, у яких час завершення вже минув. Для цього використовується умова пошуку { status: 'active', expiresAt: { $lte: now } }.

Для кожного голосування, яке потребує завершення, оновлюється статус на completed, і ці зміни зберігаються в базу даних за допомогою await vote.save(). Після цього в консоль виводиться повідомлення, яке інформує про автоматичне завершення голосування, наприклад: "Голосування 'Питання Х' завершено автоматично".

Server.js запускає планувальник scheduleVoteCompletion(), забезпечуючи регулярну перевірку статусу голосувань. Це дає змогу автоматизувати процес завершення голосувань, гарантувати точність виконання завдання та зменшити потребу в ручному втручанні (див. рис. 3.10).



Рис.3.10 – Приклад автозавершення голосування.

У проекті всі об'єкти бази даних були сформовані моделями, що базуються на схемах Mongoose. Схема визначає структуру документа, обов’язкові поля, їхні типи даних, можливі значення та поведінку при обробці запитів. Модель Vote описує, як зберігається голосування: питання, варіанти відповідей, голоси користувачів, підрахунок результатів, статус голосування, ідентифікатор автора та час завершення. Завдяки такому підходу вдається забезпечити чітку структуру даних у базі, навіть для неструктурованої MongoDB (див. рис. 3.11-3.12).

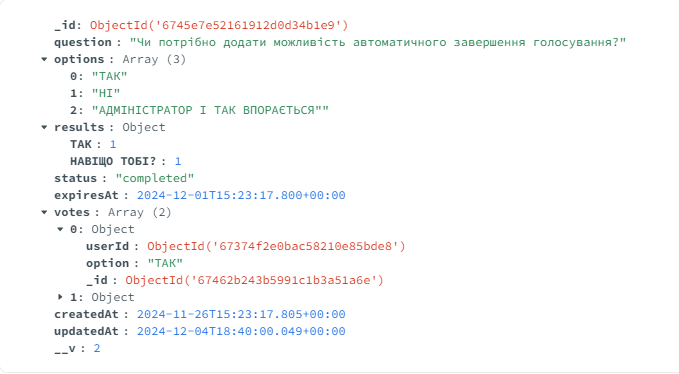


Рис. 3.11 – Об’єкт «Голосування» бази даних votes.

*//server/models/Vote.js*

const mongoose = require**(**'mongoose'**)**;

const voteSchema = new mongoose.Schema**(**{

**question:** { **type:** String, **required:** true },

**options:** **[**{ **type:** String, **required:** true }**]**,

**votes:** **[**

      {

**userId:** { **type:** mongoose.Schema.Types.**ObjectId**, **ref:** 'User' },

**option:** { **type:** String },

      },

**]**,

**results:** {

**type:** Map,

**of:** Number,

**default:** {}

  },

**status:** {

**type:** String,

**enum:** **[**'active', 'completed'**]**,

**default:** 'active',

  },

**createdBy:** { **type:** mongoose.Schema.Types.**ObjectId**, **ref:** 'User' },

**expiresAt:** { **type:** Date },

}, { **timestamps:** true }**)**;

const Vote = mongoose.model**(**'Vote', voteSchema**)**;

module.exports = Vote;

Рис. 3.12 – Модель Vote

Цей код визначає схему моделі Vote для роботи з колекцією голосувань у базі даних MongoDB. Схема включає кілька ключових полів. Поле question зберігає текст питання, винесеного на голосування, а options містить масив варіантів відповідей, які пропонуються користувачам. Поле votes зберігає масив об'єктів, де кожен об'єкт представляє голос конкретного користувача: userId посилається на ідентифікатор користувача, а option зберігає вибраний варіант відповіді. Поле results — це карта (тип Map), що використовується для збереження підрахунку голосів за кожен варіант, де ключ — це текст варіанту, а значення — кількість голосів. Статус голосування зберігається у полі status, яке може бути або active (активне голосування), або completed (завершене голосування). Поле createdBy зберігає ідентифікатор користувача, який створив голосування. expiresAt вказує час, коли голосування має завершитися автоматично. У схемі також активовано таймстемпи (timestamps), які автоматично додають поля createdAt і updatedAt для відстеження часу створення та оновлення запису. Усі ці властивості разом формують структуру, необхідну для збереження, відстеження та управління голосуваннями.

Усі маршрути модуля захищені за допомогою middleware verifyToken, яке перевіряє валідність JWT токену користувача.  
 Файл маршрутизації votes обробляє запити та передає їх до відповідних методів контролера.

## 3.3 Модуль комунікації.

Контроллер для модуля чату відповідає за обробку запитів, пов'язаних із повідомленнями в системі. Він забезпечує функціонал для отримання всіх повідомлень, додавання нових, а також видалення повідомлен. Контролер використовує Express.js для створення маршрутів і взаємодіє з моделлю Message, яка представляє повідомлення в базі даних.

Основні функції контролера:

**1. Отримання всіх повідомлень:**

Цей API маршрут виконує запит до бази даних за допомогою Message.find(), для отримання всіх повідомлень, відсортованих від найновішого до найстарішого за датою створення. Запит підтримує пагінацію, яка дає змогу користувачам отримувати лише частину повідомлень на кожній сторінці.

Параметр page визначає номер сторінки, а limit – кількість повідомлень на сторінці. За замовчуванням значення page дорівнює 1, а limit – 10. Пагінація реалізується за допомогою методів skip() і limit(), де skip() пропускає повідомлення на попередніх сторінках, а limit() обмежує кількість повідомлень на поточній сторінці.

Всі повідомлення, що відповідають запиту, повертаються у форматі JSON. У разі помилки сервер відповідає статусом 500 і повідомленням "Помилка сервера".

**2. Додавання нового повідомлення:**

Middleware verifyToken використовується для перевірки токена користувача, щоб забезпечити авторизацію. Після успішної авторизації інформація про користувача розшифровується з токена і доступна через req.user. Дані нового повідомлення (поле text і інформація про користувача) передаються через тіло запиту і додаються до нового екземпляра моделі Message. Нове повідомлення зберігається в базі даних через newMessage.save(). Якщо збереження успішне, сервер повертає статус 201 Created і об'єкт нового повідомлення. У разі виникнення помилки повертається статус 500.

**3. Видалення повідомлення:**

Middleware verifyToken також перевіряє авторизацію користувача. Запит обробляє параметр id з URL, який використовується для пошуку повідомлення в базі даних через Message.findById(messageId). Якщо повідомлення не знайдене, повертається статус 404 Not Found. Для видалення повідомлення виконується перевірка прав. Користувач має бути або автором повідомлення, або адміністратором (перевіряється поле role з токена). Якщо користувач не має відповідних прав, повертається статус 403 Forbidden. Якщо перевірка успішна, повідомлення видаляється за допомогою Message.deleteOne({ \_id: messageId }), і сервер повертає статус 200 OK із підтвердженням видалення. У разі помилки повертається статус 500.

Контролер працює за кількома ключовими аспектами, щоб забезпечити ефективну обробку запитів і безпеку застосунку.

Маршрутизація в контролері реалізується за допомогою різних HTTP-методів (GET, POST, DELETE), які обробляють різні типи запитів. Використання middleware, такого як verifyToken, дає змогу захищати маршрути, надаючи доступ тільки авторизованим користувачам. Це забезпечує додатковий рівень безпеки, обмежуючи функціональність для неавторизованих користувачів.

Робота з базою даних реалізується через модель Message, яка інкапсулює логіку взаємодії з MongoDB. Усі запити до бази даних виконуються через цю модель, що забезпечує централізоване управління операціями над даними та спрощує їх обробку.

Верифікація прав доступу є важливою частиною контролера. Для додавання нових повідомлень необхідна авторизація через токен, що гарантує, що тільки авторизовані користувачі можуть створювати нові повідомлення. Для видалення повідомлень доступ має лише автор самого повідомлення або адміністратор, що забезпечує контроль доступу до чутливих операцій.

Модель Message, створена за допомогою бібліотеки Mongoose, є схемою для колекції повідомлень у MongoDB. Вона визначає структуру документів у цій колекції та забезпечує валідацію та взаємодію з даними.

Схема messageSchema описує такі поля:

1. **text**: Поле типу String, яке є обов'язковим (required: true). Воно зберігає текст повідомлення.
2. **createdAt**: Поле типу Date, яке автоматично отримує поточну дату та час при створенні документа завдяки default: Date.now. Це поле відображає дату та час створення повідомлення.
3. **user**: Об'єкт, що містить інформацію про користувача, який створив повідомлення. Він включає:

* **firstName**: Поле типу String, обов'язкове, яке зберігає ім'я користувача.
* **lastName**: Поле типу String, обов'язкове, яке зберігає прізвище користувача.
* **apartment**: Поле типу String, обов'язкове, яке вказує номер квартири користувача.

Ця модель експортується через module.exports, щоб її можна було використовувати в інших частинах сервера для взаємодії з колекцією Message у базі даних MongoDB (див. рис. 3.14). Завдяки Mongoose, ця модель дає змогу легко створювати, читати, оновлювати та видаляти документи, виконуючи ці операції у вигляді об'єктів JavaScript.

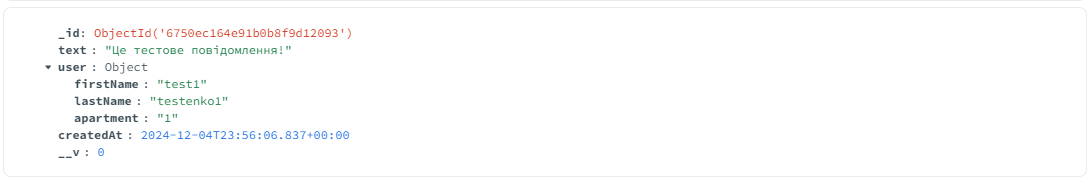


Рис. 3.14 - Приклад об’єкта Message

## 3.4 Фінансовий модуль

За логіку фінансового модуля відповідають три взаємозв’язні контролери, які оперують двома моделями.

### 3.4.1 osbbStatisticsController.js

Цей контролер відповідає за створення та управління статистикою зборів ОСББ.

* **createStatistic**:
  + Використовується для створення нового збору з вказаною метою та загальною сумою.
  + Перевіряються обов'язкові поля (мета збору та загальна сума).
  + Створюється новий об'єкт OsbbStatistics та зберігається в базі даних.
  + Повертається повідомлення про успішне створення збору разом із даними створеного об'єкта.

### 3.4.2 financeController.js

Цей контролер відповідає за управління фінансовою інформацією та наданням даних про активні збори для користувачів та адміністраторів.

* **getActiveCollections**:

Метод **getActiveCollections** дає змогу отримати детальну інформацію про активні збори в ОСББ, враховуючи роль поточного користувача. Активні збори, тобто збори з непогашеним боргом, вибираються з бази даних **OsbbStatistics**, причому дані про платежі користувачів у цих зборах зв’язуються через **populate** для отримання імені користувача та номера квартири.

Далі логіка обробки залежить від ролі користувача. Якщо це **адміністратор**, метод формує повну інформацію про всі збори. Для кожного збору розраховується внесок кожного мешканця на основі площі його квартири (використовується функція **getApartmentArea**) у пропорції до загальної площі будинку (5776 м²). Згідно з даними про платежі в базі, визначається, скільки кожен користувач уже заплатив, і обчислюється залишок. Усі ці дані включаються в масив **contributions**, який додається до інформації про збір.

Наприклад, якщо у зборі мета — ремонт даху, а загальна сума збору становить 50,000 грн, адміністратор побачить повний список користувачів із їхнім внеском. Наприклад, для мешканця з квартирою площею 60 м² (загальна площа будинку 5776 м²), внесок буде:  
 **userContribution = (50000 / 5776) \* 60 ≈ 519.38 грн**. Якщо цей мешканець уже сплатив 300 грн, то залишок становитиме 219.38 грн.

Якщо ж користувач **не є адміністратором**, то він отримує лише інформацію про свій власний внесок у кожен активний збір. Логіка аналогічна, але дані про платежі та залишок обмежуються лише поточним користувачем. Наприклад, мешканець з тією ж квартирою 60 м² побачить:

* Загальна сума збору: 50,000 грн.
* Його обов’язковий внесок: 519.38 грн.
* Уже сплачено: 300 грн.
* Залишок: 219.38 грн.

У підсумку, цей API є інструментом для прозорого управління фінансами в ОСББ, надаючи адміністраторам деталізовану інформацію про всі збори, а звичайним мешканцям — лише їхній внесок (див. рис. 3.15).

exports.getActiveCollections = async (req, res) => {

  try {

    const userId = req.user.**userId**; *// ID поточного користувача*

    const isAdmin = req.user.**role** === 'admin';

*// Отримуємо активні збори*

    const collections = await OsbbStatistics.find**(**{ **debt:** { **$gt:** 0 } }**)**.populate**(**'payments.userId', 'username apartmentNumber'**)**;

    const users = await User.find**(**{}, 'username apartmentNumber'**)**;

    const detailedCollections = collections.map**(**(collection) => {

      if **(**isAdmin**)** {

*// Логіка для адміністратора (всі внески)*

        const contributions = users.map**(**(user) => {

          const area = getApartmentArea**(**user.**apartmentNumber)**;

          const userContribution = **(**collection.**totalAmount** / 5776**)** \* area; *// Розрахунок внеску*

          const userPaid = collection.**payments**.find**(**

            (p) => p.**userId** && p.**userId**.toString**()** === user.**\_id**.toString**()**

**)**?.**amount** || 0;

          const remaining = userContribution - userPaid;

          return {

**name:** user.**username**,

**apartmentNumber:** user.**apartmentNumber**,

**area**,

**requiredContribution:** userContribution.toFixed**(**2**)**,

**paid:** userPaid.toFixed**(**2**)**,

**remaining:** Math.abs**(**remaining**)** < 0.01 ? '0.00' : remaining.toFixed**(**2**)**,

          };

        }**)**;

        return {

**purpose:** collection.**purpose**,

**totalAmount:** collection.**totalAmount**.toFixed**(**2**)**,

**collectedAmount:** collection.**collectedAmount**.toFixed**(**2**)**,

**debt:** collection.**debt**,

**contributions**,

        };

      } else {

*// Логіка для користувача (тільки його внесок)*

        const userArea = getApartmentArea**(**req.user.**apartmentNumber)**;

        const userContribution = **(**collection.**totalAmount** / 5776**)** \* userArea;

        const userPaid = collection.**payments**.find**(**

          (p) => p.**userId** && p.**userId**.toString**()** === userId

**)**?.**amount** || 0;

        const remaining = userContribution - userPaid;

        return {

**purpose:** collection.**purpose**,

**totalAmount:** collection.**totalAmount**.toFixed**(**2**)**,

**collectedAmount:** collection.**collectedAmount**.toFixed**(**2**)**,

**debt:** collection.**debt**.toFixed**(**2**)**,

**userPayment:** {

**requiredContribution:** userContribution.toFixed**(**2**)**,

**paid:** userPaid.toFixed**(**2**)**,

**remaining:** Math.abs**(**remaining**)** < 0.01 ? '0.00' : remaining.toFixed**(**2**)**,

          },

        };

      }

    }**)**;

    res.json**(**detailedCollections**)**;

  } catch **(**error**)** {

    console.error**(**'Error fetching active collections:', error.**message)**;

    res.status**(**500**)**.json**(**{ **message:** 'Error fetching active collections' }**)**;

  }

* };

Рис. 3.15 – Реалізація метода розрахунку та відображення зборів

### 3.4.3 paymentController.js

Цей контролер відповідає за створення та перевірку статусу платежів через платіжну систему LiqPay [22]. Дані API щільно зв'язано з API **getActiveCollections,** який вже був описаний.Дані, які отримує користувач про його необхідний внесок, отримує **createPayment.**

* **createPayment**:

Одна із головних особливостей взаємодії з **API LiqPay** – це криптографічний підпис, який повинен мати кожен запит до API LiqPay (див. рис. 3.16).

*// Публічні та приватні ключі LiqPay*

const PUBLIC\_KEY = process.**env**.LIQPAY\_PUBLIC\_KEY;

const PRIVATE\_KEY = process.**env**.LIQPAY\_PRIVATE\_KEY;

*// Генерація підпису*

const generateSignature = (data) => {

  const str = PRIVATE\_KEY + data + PRIVATE\_KEY;

  const signature = crypto.createHash**(**'sha1'**)**.update**(**str**)**.digest**(**'base64'**)**;

  console.log**(**'Generated signature:', signature**)**;

  return signature;

};

Рис.3.16 – Генерація підпису

Функція **generateSignature** використовується для створення криптографічного підпису, необхідного для взаємодії з API платіжної системи LiqPay. Підпис гарантує, що передані дані є автентичними та не були змінені під час передачі. Для цього використовується приватний ключ, який має бути відомий лише серверу.

Спочатку функція формує рядок для хешування, об'єднуючи приватний ключ, дані в Base64-форматі та знову приватний ключ. Цей підхід забезпечує цілісність переданих даних.

Далі, для обчислення підпису, використовується алгоритм хешування SHA-1. Хеш обчислюється для сформованого рядка, після чого результат хешу кодується у формат Base64.

Результат цієї операції, тобто згенерований підпис, повертається функцією та використовується для підпису запиту до API LiqPay. Перед відправкою підпис додається до даних, які LiqPay перевіряє на відповідність. Якщо підпис збігається з очікуваним, API вважає запит справжнім.

Ця функція є ключовим компонентом у забезпеченні безпеки, оскільки виключає можливість підробки даних, а також підтверджує їх цілісність. Завдяки цьому LiqPay може довіряти отриманим даним, знаючи, що вони згенеровані та підписані сервером з правильним приватним ключем.

Коли користувач ініціює платіж, сервер отримує дані (сума, опис, мета збору) - генерується унікальний ідентифікатор платежу (**orderId**).

Потім формується об'єкт із обов’язковою платіжною інформацією, що включає:

1. Суму платежу (**amount**),
2. Валюту (UAH),
3. Мету (опис збору),
4. Ідентифікатор замовлення,

Ця дата шифрується, і об'єднується з підписом (див. рис. 3.17).

*// Формування даних для платіжної форми*

const createPaymentData = (paymentData) => {

  console.log**(**'Preparing payment data:', paymentData**)**;

  const data = Buffer.from**(**JSON.stringify**(**paymentData**))**.toString**(**'base64'**)**;

  const signature = generateSignature**(**data**)**;

  console.log**(**'Encoded payment data (base64):', data**)**;

  console.log**(**'Generated signature for payment:', signature**)**;

  return { **data**, **signature** };

};

Рис. 3.17 – Функція формування даних.

Після кодування та підписа даних створюється запис у базі **UserPayment** зі статусом "pending". Перед збереженням сервер перевіряє, чи цей користувач уже робив внесок у цей збір. Якщо внесок уже існує, повертається статус **403** із відповідним повідомленням.

Як результат, користувач отримує платіжні дані та посилання на форму LiqPay, де він може виконати платіж (див. рис. 3.18).

*// Запит до API LiqPay*

const apiRequest = async (path, data) => {

  try {

    console.log**(**'Making API request to LiqPay:', `https://www.liqpay.ua/api/request`**)**;

    const signature = generateSignature**(**data**)**;

    const formData = new URLSearchParams**()**;

    formData.append**(**'data', data**)**;

    formData.append**(**'signature', signature**)**;

    console.log**(**'Request data:', data**)**;

    console.log**(**'Generated signature:', signature**)**;

    const response = await axios.post**(**`https://www.liqpay.ua/api/request`, formData, {

**headers:** { **'Content-Type':** 'application/x-www-form-urlencoded' }

    }**)**;

    console.log**(**'API response:', response.**data)**;

    return response.**data**;

  } catch **(**error**)** {

    console.error**(**`Error during API request to ${path}:`, error.**message)**;

    throw new Error**(**'Error while communicating with LiqPay API'**)**;

  }

};

Рис. 3.18 – Формування запиту до LiqPay

* **getPaymentStatus**:

Функція **getPaymentStatus** використовується для перевірки статусу платежу на основі переданого **orderId**. Вона дає змогу дізнатися, чи було платіж успішним, у процесі, чи невдалим, а також оновлює інформацію про платіж у базі даних. Ця функція є важливою для забезпечення актуальності даних про платежі як у системі ОСББ, так і в користувацькому інтерфейсі.

Спочатку функція отримує параметр orderId з URL запиту. Далі створюється об'єкт із запитом, який містить дані для перевірки статусу платежу. Ці дані включають публічний ключ, версію API, тип дії (status) і унікальний ідентифікатор платежу (orderId). Цей об'єкт перетворюється на формат JSON і кодується у Base64 для передачі в API LiqPay. Потім за допомогою функції apiRequest виконується HTTP-запит до LiqPay, і система отримує відповідь із серверу LiqPay.

Якщо API повертає статус **success** або **sandbox** (тобто платіж був успішним або виконаний у тестовому режимі), функція оновлює запис у колекції **UserPayment**, встановлюючи статус платежу, збережений **paymentId**, і час останнього оновлення. Також вона оновлює статистику ОСББ. Для цього шукається відповідний запис у **OsbbStatistics**, і якщо такий запис існує, додається сума платежу до зібраних коштів (collectedAmount), а борг (debt) перераховується на основі залишку між загальною сумою збору і вже зібраною сумою. Інформація про новий платіж додається до масиву платежів для конкретного збору.

Якщо ж статус платежу в API LiqPay повертається як failure (невдалий платіж), запис у **UserPayment** оновлюється зі статусом failure, щоб відобразити невдачу.

Ці контролери разом утворюють важливу частину фінансового модуля системи, який забезпечує управління збором коштів, моніторинг фінансових внесків користувачів та інтеграцію з платіжною системою LiqPay.

## 3.5 Модуль інфраструктури

### 3.5.1 Контролер infrastructureController.js

Цей контролер управляє статусом інфраструктури будівлі (стан ліфтів та комунікацій) і статистикою споживання ресурсів (вода, газ, електроенергія).

**Функції:**

* **getInfrastructureStatus**  
  Отримує поточний статус інфраструктури. Шукає запис у базі даних моделі Infrastructure і повертає його.
* **updateInfrastructureStatus**  
  Цей метод відповідає за оновлення статистики споживання ресурсів у базі даних. У функції спочатку отримуються необхідні дані з тіла запиту: місяць (month), рік (year), а також значення споживання води, газу та електроенергії. Це дає змогу працювати з конкретним періодом, для якого буде здійснено оновлення або створення запису.

Основна операція виконується за допомогою методу findOneAndUpdate моделі ConsumptionStatistics. Цей метод шукає запис у базі даних за переданими значеннями month і year. Якщо запис знайдено, його значення оновлюються відповідно до даних з тіла запиту. Якщо запис відсутній, створюється новий завдяки опції upsert: true. Опція new: true забезпечує, що у відповідь повертається вже оновлений або створений запис.

Після успішного оновлення клієнту повертається JSON-об'єкт із оновленою статистикою (див. рис. 3.19).

*// Оновлення статистики споживання*

const updateConsumptionStatistics = async (req, res) => {

  try {

    const { month, year, waterConsumption, gasConsumption, electricityConsumption } = req.**body**;

    console.log**(**month,year**)**;

    const updatedStatistics = await ConsumptionStatistics.findOneAndUpdate**(**

      { **month**, **year** },

      { **waterConsumption**, **gasConsumption**, **electricityConsumption** },

      { **new:** true, **upsert:** true }

**)**;

    res.json**(**updatedStatistics**)**;

  } catch **(**error**)** {

    console.error**(**'Помилка при оновленні статистики споживання:', error**)**;

    res.status**(**500**)**.json**(**{ **message:** 'Внутрішня помилка сервера' }**)**;

  }

};

Рис. 3.19 – Приклад оновлення статистики споживання ресурсів

* **getConsumptionStatistics**  
  Отримує статистику споживання за конкретний місяць і рік (вказується у query).
* **updateConsumptionStatistics**  
  Оновлює статистику споживання для конкретного місяця і року. Якщо запису ще немає, створює новий.
* **createConsumptionStatistics**  
  Створює новий запис статистики споживання для певного місяця і року.

### 3.5.2 Контролер для роботи з розкладом відключень електроенергії getSchedule

Цей контролер читає файл JSON (schedule.json), що містить розклад, і повертає його вміст.

* **getSchedule**  
  Перевіряє наявність файлу schedule.json і, якщо файл існує, читає та повертає його вміст як JSON. У разі відсутності файлу повертає помилку 404.

### 3.5.3 Утиліта fetchTableFromWebsite для парсингу розкладу відключень

Цей скрипт автоматизує процес парсингу даних з офіційного вебсайту поставника послуг про графіки відключення електроенергії та записує інформацію у JSON-файл для подальшого використання. Він використовує бібліотеки puppeteer для автоматизації браузера, fs для роботи з файлами, і jsdom для обробки HTML.

Оскільки постачальних послуг, яким буде користуватись ОСББ не надає публічних API для робот з графіками відключення світла, був розроблен свій скрипт, який дає змогу автоматично відображати графіки світла. Попередньо, було переглянуто ліцензії та умови використання сайту. Також було переглянуто robots.txt, який налаштовує поведінку вебсканерів (див. рис.3.20).

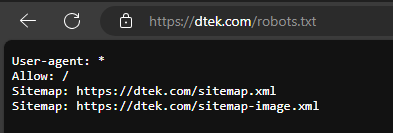


Рис.3.20 – приклад robots.txt

1. User-agent: \*

* Це означає, що правила, які йдуть далі, застосовуються до всіх вебсканерів (тобто, "всі користувацькі агенти").

1. Allow: /

* Цей рядок вказує, що сканери мають доступ до всіх розділів сайту. Відсутні заборони на індексацію.

Для цього реалізовано метод автоматизованого аналізу вебресурсів (парсинг). Загалом процес парсингу складається з таких етапів:

1. Пошук даних – програма-парсер завантажує вихідний HTML-код сторінки сайту. Скрипт аналізує код, розбиваючи текст на лексеми та виділяючи необхідну інформацію.
2. Вилучення інформації – здійснюється пошук даних за визначеними шаблонами для їх подальшого використання.
3. Збереження даних – отримана інформація структурується та зберігається у вигляді таблиць або в базі даних.

Таким чином, основна мета процесу збору даних полягає в автоматизованому пошуку, вилученні та збереженні необхідної інформації.

Всі запити виконуються виключно з серверної сторони, що не дає змогу робити запуски з багатьох одночасних сесій. Запити налаштовані на чотири рази на день, саме в ті періоди, коли найбільш імовірні критичні оновлення. Додатково оновлення статистики відбувається під час запланованих відключень електроенергії. Наприклад, для четверга додаткові точкові запити будуть виконуватися о 01:30, 10:30 і 19:30. Це дає змогу отримувати не лише загальну інформацію на день, але й актуальний графік відключень**.**

У середньому генерується 14-16 запитів на день, що не перевантажує сервер постачальника послуг, але водночас забезпечує інформування користувачів у реальному часі (див. рис. 3.21).

{

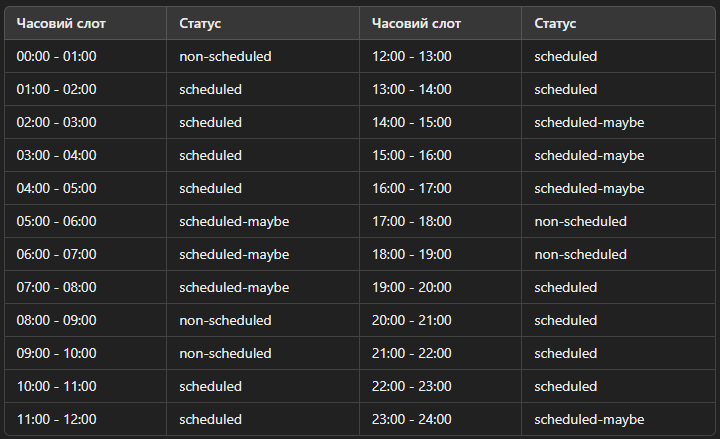
**"outageInfo"**: {

**"status"**: "Світло за нашою адресою є",

**"updateTime"**: "– 15:03 05.12.2024"

  },

**"Четвер"**: [

****

],

}

Рис. 3.21 – приклад сформованного json.файлу

**Функції:**

* **shouldRunToday**  
  Перевіряє, чи потрібно виконувати парсинг сьогодні (перевіряє файл lastRun.txt). Якщо з часу останнього запуску пройшло понад 8 годин або файл відсутній, повертає true.
* **updateLastRunTime**  
  Оновлює файл lastRun.txt із часом останнього запуску.
* **fetchTableFromWebsite**  
   Парсить розклад відключень електроенергії з вебсайту за допомогою Puppeteer. Результат записується у файл schedule.json.  
   Основні дії:
  + Відкриває вебсайт і вводить адресу.
  + Читає таблицю відключень електроенергії.
  + Обробляє інформацію та зберігає її в JSON-файл.
  + Планує додаткові запити, якщо є розклад запланованих відключень.
* **processTable**  
  Обробляє отриману таблицю з HTML у формат JSON. Також визначає час для додаткових запитів на основі запланованих відключень.
* **scheduleDailyTask**  
  Запускає щоденну перевірку парсингу з 01:00. Якщо парсинг ще не виконувався сьогодні, запускає fetchTableFromWebsite.

# ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі реалізовано ключові функціональні модулі інформаційної системи управління та комунікації для ОСББ. Кожен із розроблених компонентів відповідає сучасним вимогам до багатофункціональних інформаційних систем, забезпечуючи надійність, ефективність і зручність використання для кінцевих користувачів.

У підрозділі 3.1 детально розглянуто реалізацію автентифікації з використанням JWT токенів, доповнену сучасними методами безпеки. Це забезпечує високу надійність та захищеність персональних даних користувачів, що є критично важливим для таких систем. Використання кращих практик у роботі з JWT дозволяє мінімізувати ризики компрометації сесій та зловживання доступом.

Модуль голосувань, описаний у підрозділі 3.2, дає можливість реалізувати демократичний процес прийняття рішень для ОСББ. Цей модуль автоматизує створення, проведення та підрахунок голосів, спрощуючи участь співвласників у колективних процесах і забезпечуючи прозорість рішень.

У підрозділі 3.3 розроблено модуль комунікації, який є центральною частиною системи. Він об’єднує учасників ОСББ, забезпечуючи швидкий та ефективний обмін інформацією між адміністрацією та співвласниками. Інтеграція функцій для створення, відправки та збереження повідомлень значно полегшує комунікацію в ОСББ.

Фінансовий модуль, описаний у підрозділі 3.4, забезпечує облік фінансових операцій, автоматизацію внесків і інтеграцію з платіжною системою LiqPay. Використання трьох основних контролерів дозволяє здійснювати деталізований облік фінансової діяльності, зокрема статистику витрат, обробку платежів та автоматичне оновлення фінансових даних.

У підрозділі 3.5 описано модуль інфраструктури, який дозволяє моніторити стан об’єктів ОСББ і працювати з розкладом відключень електроенергії.

Таким чином, у третьому розділі реалізовано інтегровану та високопродуктивну систему, яка автоматизує ключові процеси управління ОСББ. Розроблені модулі забезпечують безперебійну роботу системи, підвищують її функціональність і значно полегшують повсякденну діяльність адміністрації та співвласників багатоквартирних будинків.

# 4. ТЕСТУВАННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ

У будь-якій науково-дослідній роботі важливе місце займає практичне підтвердження досягнутих результатів. Саме результати є відображенням проведеної роботи, її ефективності та можливості реалізації запропонованих рішень у реальних умовах. У цьому розділі розглядаються підсумки реалізації інформаційної системи, описуються її ключові функціональні можливості, демонструється впровадження інноваційних рішень, які покращують якість управління та взаємодії у межах ОСББ.

Представлені результати не лише ілюструють технічні досягнення, але й підкреслюють новизну підходів до вирішення типових проблем у комунальному господарстві. Основна увага приділяється тому, як система оптимізує процеси, підвищує прозорість у взаєминах між співвласниками та адміністрацією, а також сприяє покращенню управління інфраструктурою.

Крім того, в межах аналізу результатів виконано тестування системи на предмет продуктивності, стабільності та відповідності поставленим вимогам. Здобуті дані слугують основою для порівняльного аналізу, що дозволяє оцінити переваги запропонованого підходу у порівнянні з традиційними методами або аналогічними системами.

Далі у підрозділі 4.1 "Тестування" докладно розглядаються досягнення інформаційної системи, її функціонал, переваги та ілюстрації роботи.

## 4.1 Тестування інформаційної системи

У цьому підрозділі наведено приклад роботи інформаційної системи, продемонстровано її основні можливості та етапи реалізації. Для ілюстрації результатів використовуються скріншоти інтерфейсу системи, структурна схема, а також деталізований опис основних функціональних модулів.

Даний інтерфейс забезпечує авторизацію користувачів у системі. Після введення коректних даних доступ до функціоналу ОСББ стає можливим. Інтерфейс включає поле для введення електронної пошти та пароля, а також кнопки для входу чи переходу до реєстрації (див. рис. 4.1).

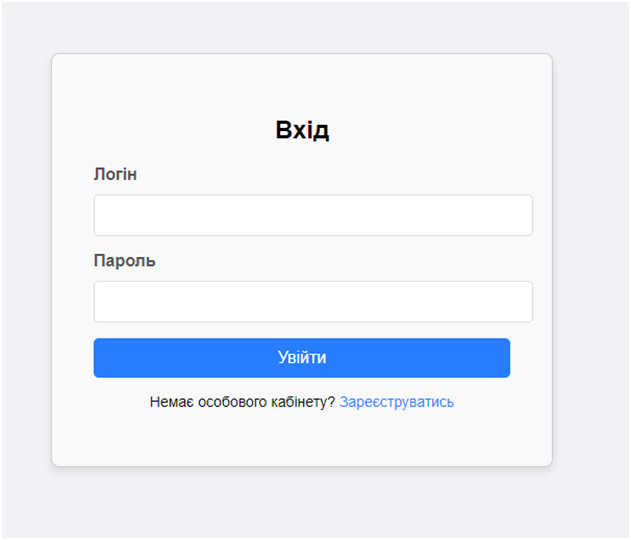


Рис. 4.1 - Вхід користувача

Цей екран демонструє процес створення нового облікового запису. Користувачеві необхідно вказати особисті дані, такі як ім’я, електронну пошту, пароль. Реєстрація гарантує індивідуальний доступ до функцій системи ОСББ (див. рис. 4.2).

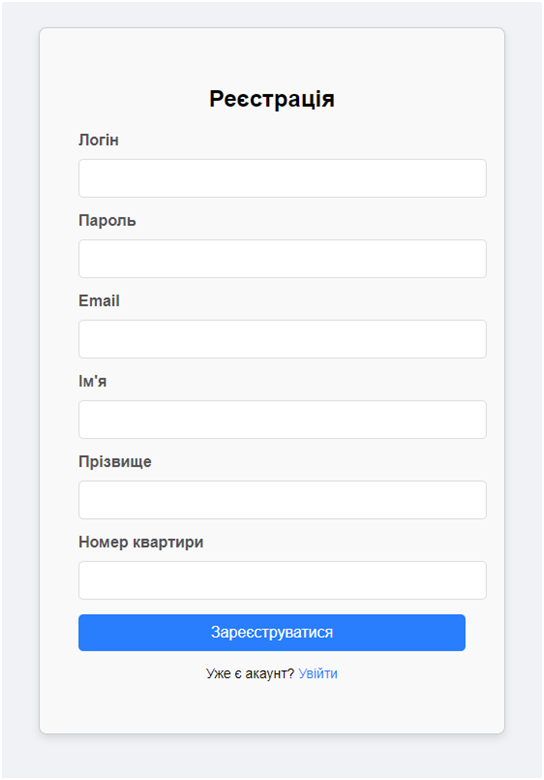


Рис. 4.2 - Реєстрація користувача

При успішному логіні, дані до клієнта завантажуються за 15 мс. Це показує швидкість та стабільність роботи сервера при передачі даних до клієнта. Наведений приклад демонструє, як швидко інформація зберігається та передається користувачеві у реальному часі (див. рис. 4.3 та 4.4).

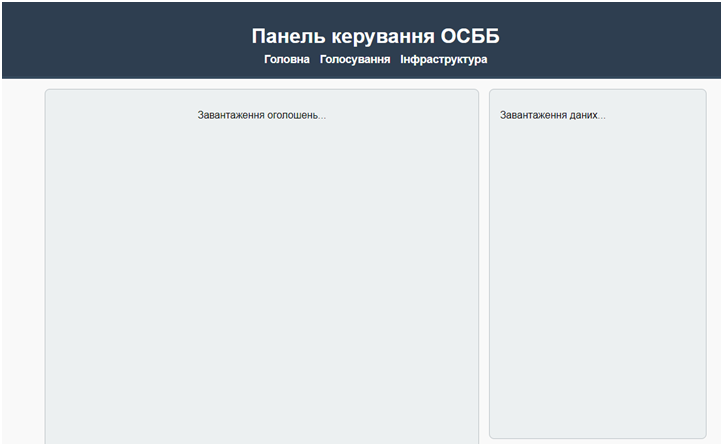


Рис. 4.3 – Приклад загрузки даних з сервера

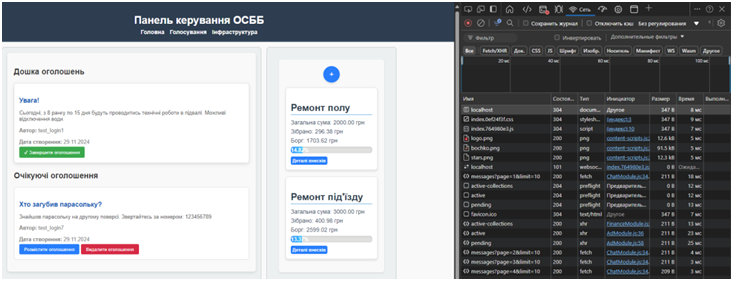


Рис. 4.4 – Завантаження цих даних з сервера.

Також користувачі можуть створювати оголошення, які потім адміністратори можуть одобрити та запостити, або відхилити (див. рис. 4.5).

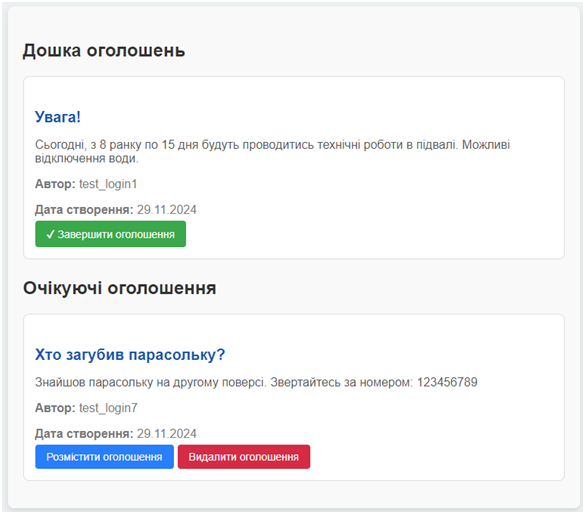


Рис. 4.5 – Інтерфейс оголошень адміністратора

Процес завершення створення оголошення, яке одразу стає доступним для перегляду користувачами. Система надає підтвердження публікації (див. рис. 4.6).

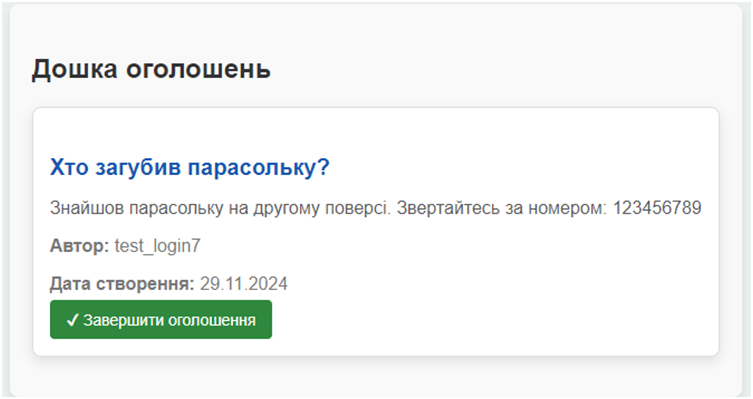


Рис. 4.6 – Завершення оголошення та розміщення

У даному прикладі зображено інтерфейс дошки оголошень для мешканців (див. рис. 4.7).

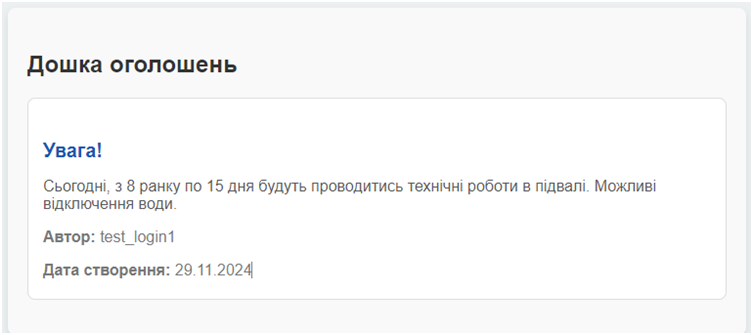


Рис. 4.7 – Приклад дошки оголошень для користувача

Інтерфейс фінансового модуля дозволяє користувачам переглядати деталі своїх платежів, автоматично розраховані внески та оплачувати рахунки через інтегровану платіжну систему (див. рис. 4.8).

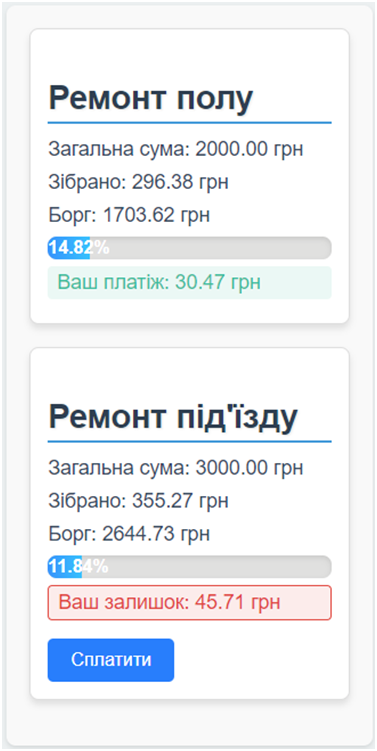


Рис. 4.8 – Фінансовий модуль

Демонструє процес оплати рахунків через систему LiqPay. Користувач отримує підтвердження успішної транзакції або повідомлення про її статус (див. рис. 4.9).

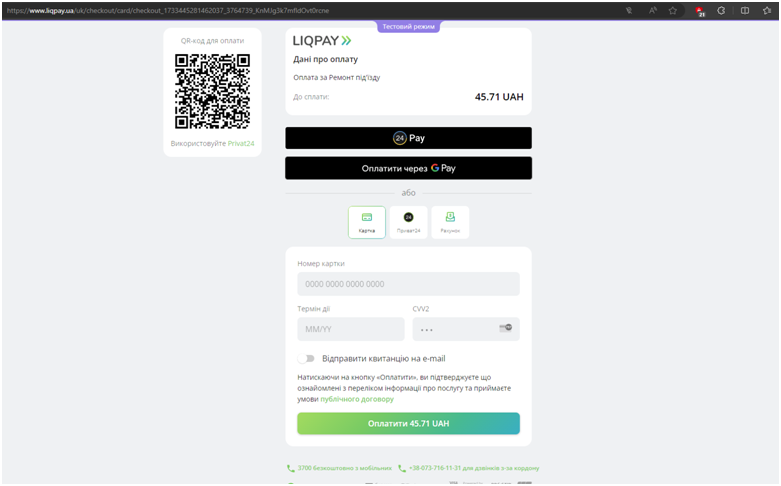


Рис. 4.9 – Інтерфейс сплати

Оновлення бази даних не буде, доки з сервера не прийде відповідь про позитивний результат сплати (див. рис. 4.10).

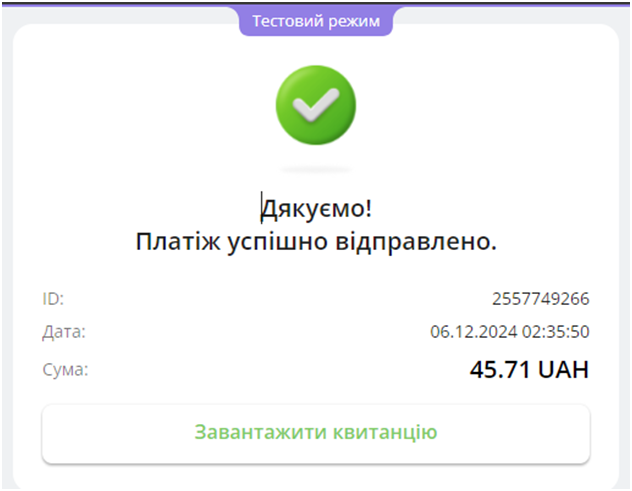


Рис. 4.10 – Успішний платіж

Користувач також може завантажити квитанцію, що підтверджує проведену оплату (див. рис. 4.11).

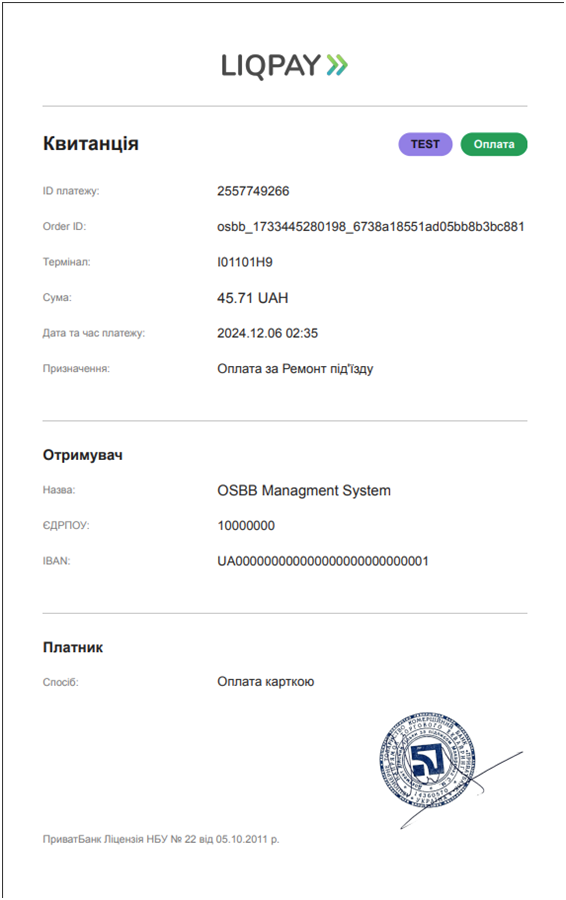


Рис. 4.11 – Квитанція про сплату

Візуалізація відображає статус транзакції: успішно завершено, в процесі або відхилено (див. рис. 4.12).

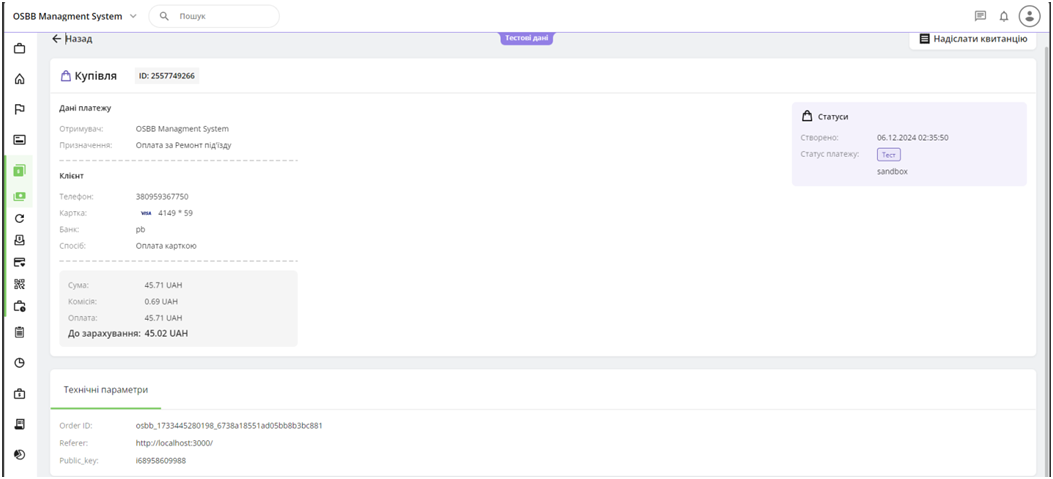


Рис. 4.12 – Статус платежа LiqPay

Після того, як транзакція має статус «успішна», дані потрапляють до бази даних, та оновлюють статистику (див. рис. 4.13).

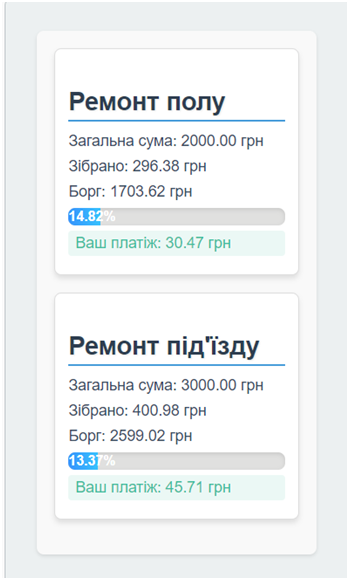


Рис. 4.13 – Успішний платіж

Адміністратори можуть створювати нові збори, які будуть йти як до загальної статистики ОСББ, так і як окремий збір (див. рис. 4.14).

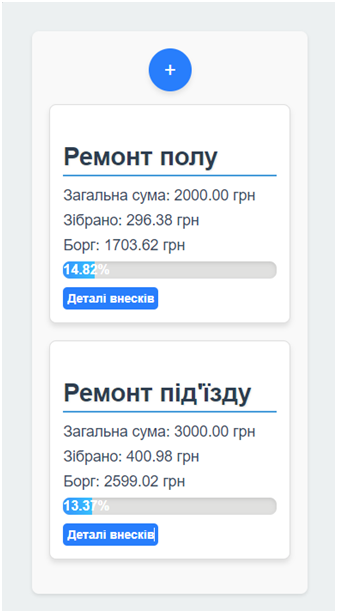


Рис. 4.14 – Інтерфейс внесків (для адміністраторів)

Також адміністратор, може дивитись деталі внесків (див. рис. 4.15).

Рис. 4.15 – Деталі внесків(тільки для адміністраторів)

Інструмент для внутрішнього зв’язку між мешканцями. Адміністратор має змогу видаляти будь-які повідомлення, якщо вони неактуальні або порушують правила (див. рис 4.16).

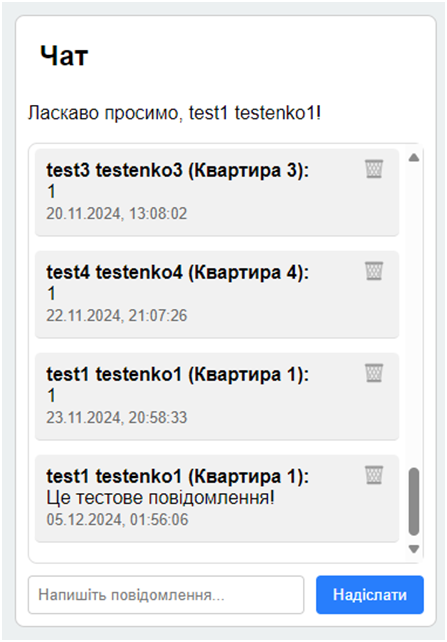


Рис. 4.16 – Модуль комунікації (для адміністраторів)

Користувачі можуть видаляти тільки власні повідомлення (див. рис. 4.17).



Рис. 4.17 – Модуль комунікації (для користувача)

Інтерфейс створення та керування голосуваннями. Адміністратор може додати нове голосування, встановити термін дії та завершити його достроково (див. рис. 4.18, 4.19, 4.20).



Рис. 4.18 – Інтерфейс голосувань

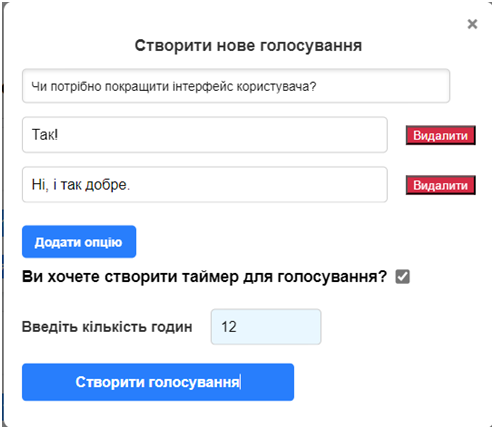


Рис. 4.19 – Інтерфейс створення голосування

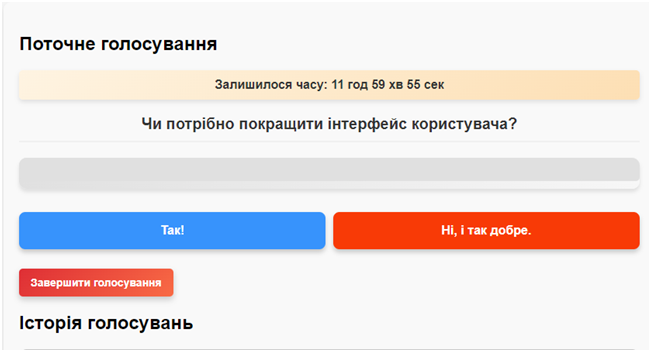


Рис. 4.20 – Інтерфейс керування голосуваннями

Графічна демонстрація, як система автоматично завершує голосування після завершення терміну дії, а результати переносяться до історії (див. рис. 4.21 та 4.22).



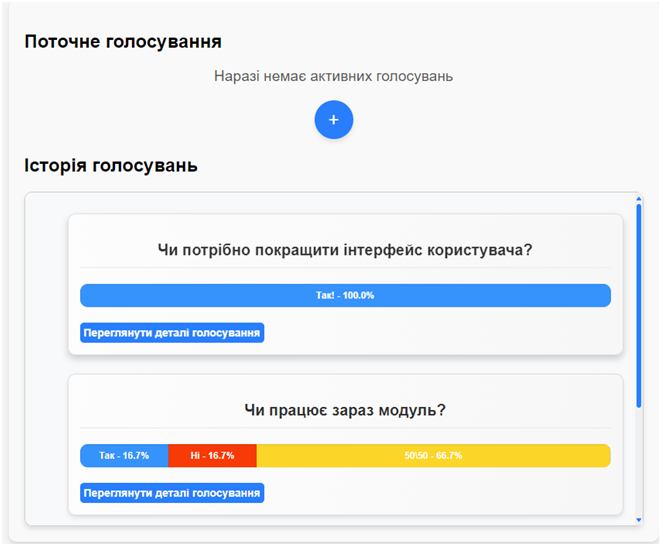
Рис. 4.21 – приклад автоматичного завершення голосування на сервері

Рис. 4.22 – Приклад автоматичного завершення голосування на клієнті

Також користувачі можуть побачити деталі історії голосувань. Ці деталі бачно тільки для завершених, для того, щоб не було додатково соціального тиску при активному голосуванні (див. рис. 4.23).



Рис. 4.23 – Приклад історії голосувань

Якщо користувач повторно голосує, виникає сповіщення (див. рис. 4.24 та 4.25).

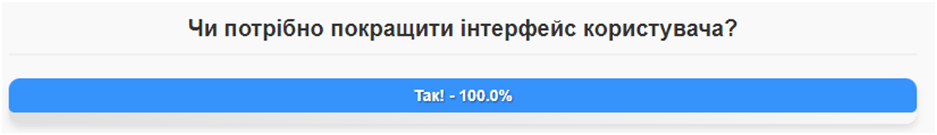


Рис. 4.24 – користувач проголосував і відсоткове відображення змінилось.

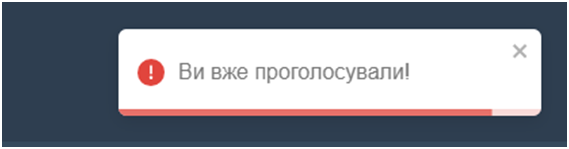


Рис. 4.25 – Приклад сповіщення, що користувач вже проголосував

Однією з особливостей модуля є сповіщення про актуальний розклад відключень. Приклад серверної реалізації вже був обговорений в розділі 3.5 (див. рис. 4.26 і 4.27, а також 4.28 та 4.29).

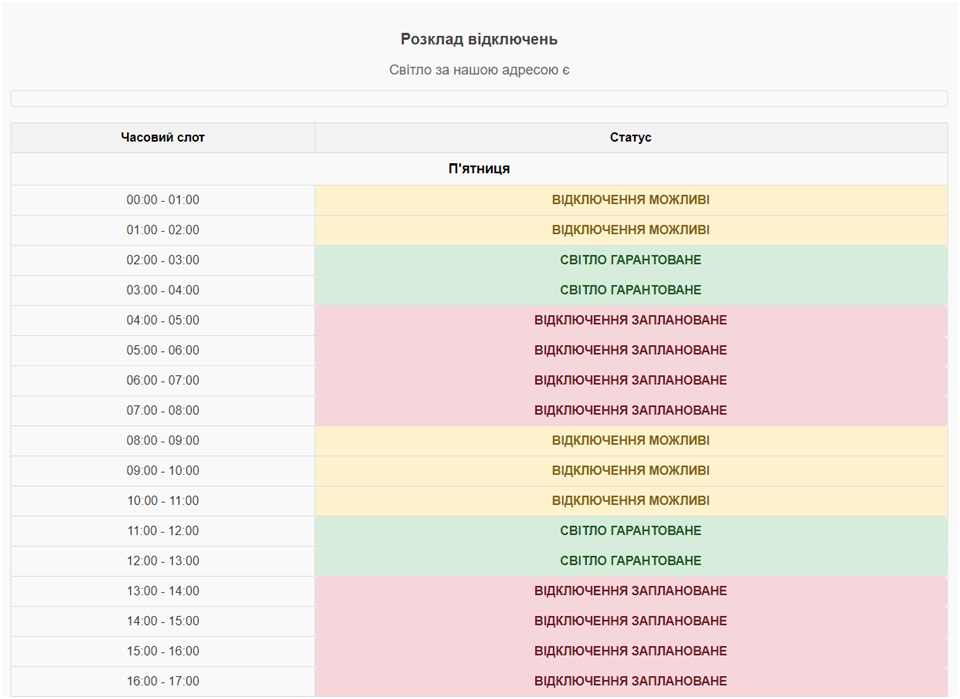


Рис. 4.26 – Розклад відключень

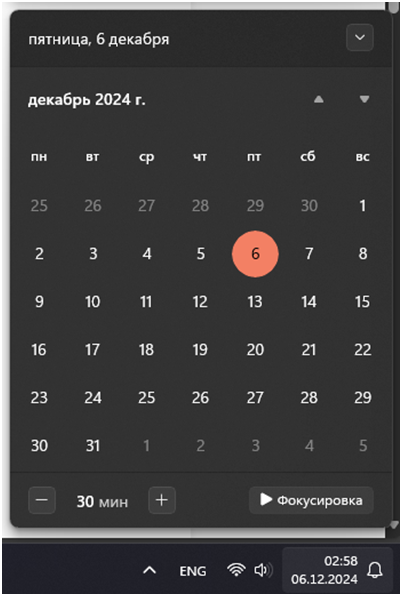


Рис. 4.27 – Поточна дата

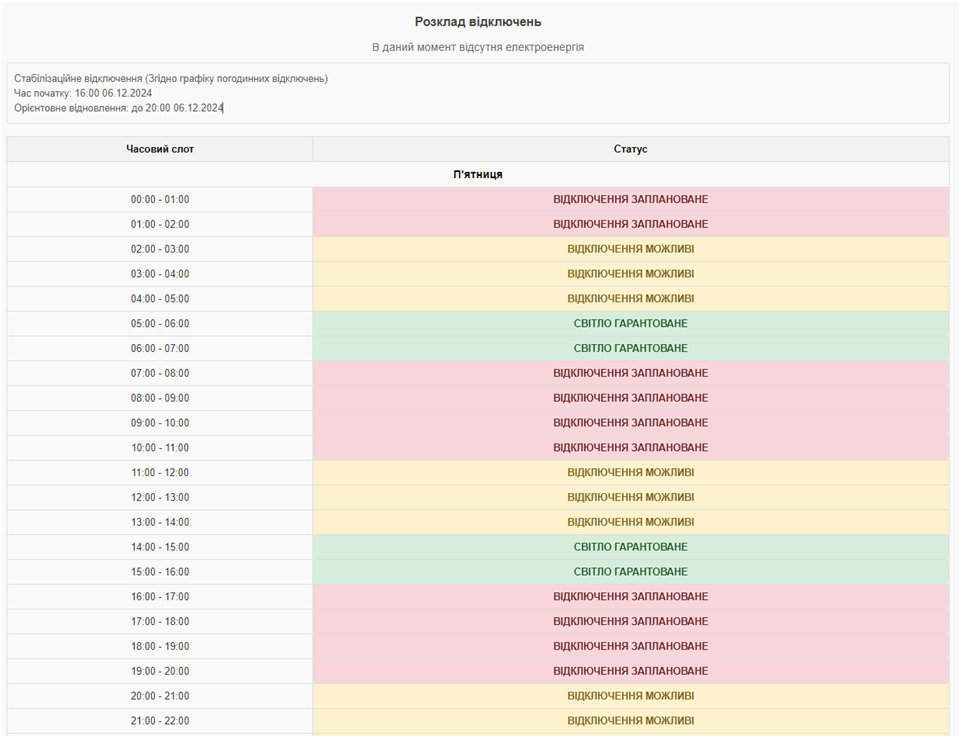


Рис. 4.28 – Приклад, що світла немає

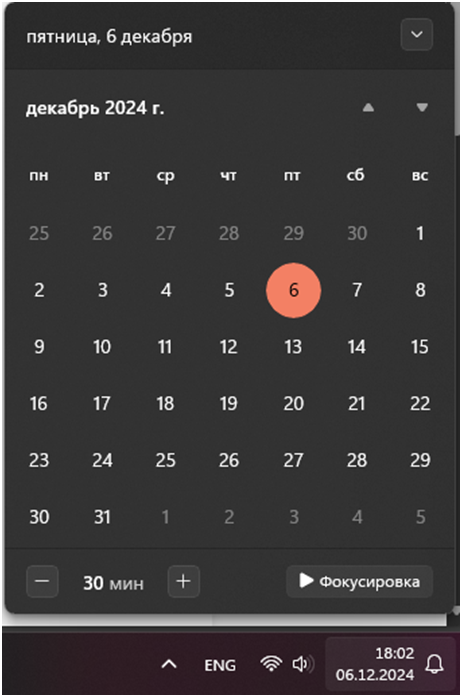


Рис. 4.29 – Поточний час і дата

Графік та таблиця, що демонструють витрати води, електроенергії та газу за обраний місяць (див. рис. 4.30 та 4.31).



Рис. 4.30 – Споживання ресурсів за листопад



Рис. 4.31 – Споживання ресурсів за грудень

Також користувач може дізнатись активний статус ліфтів (див. рис. 4.32 та 4.33).

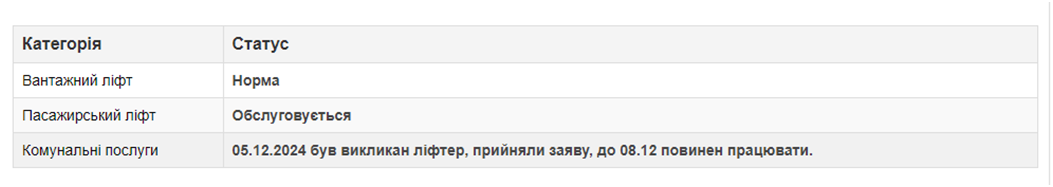


Рис. 4.32 – Статус ліфтів

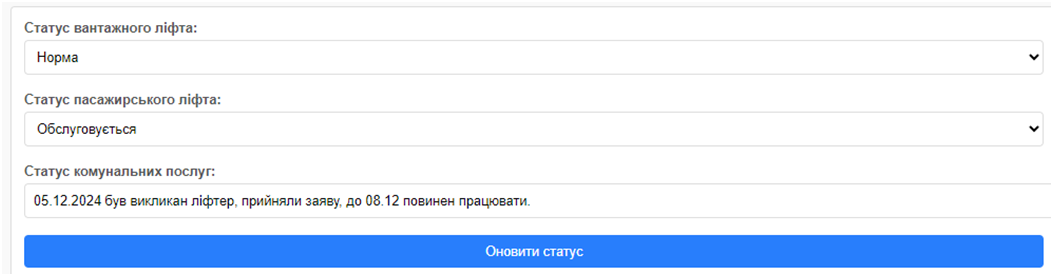
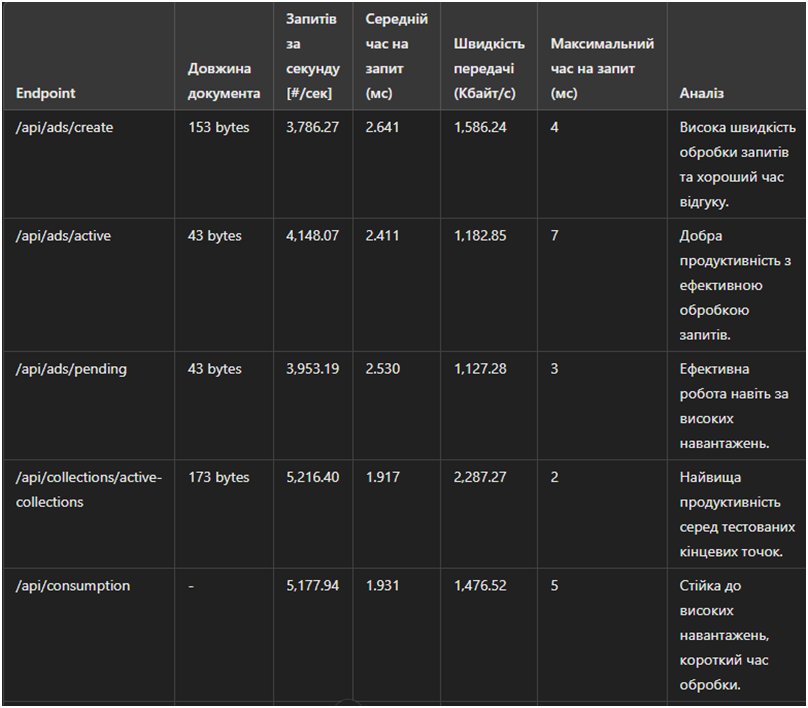
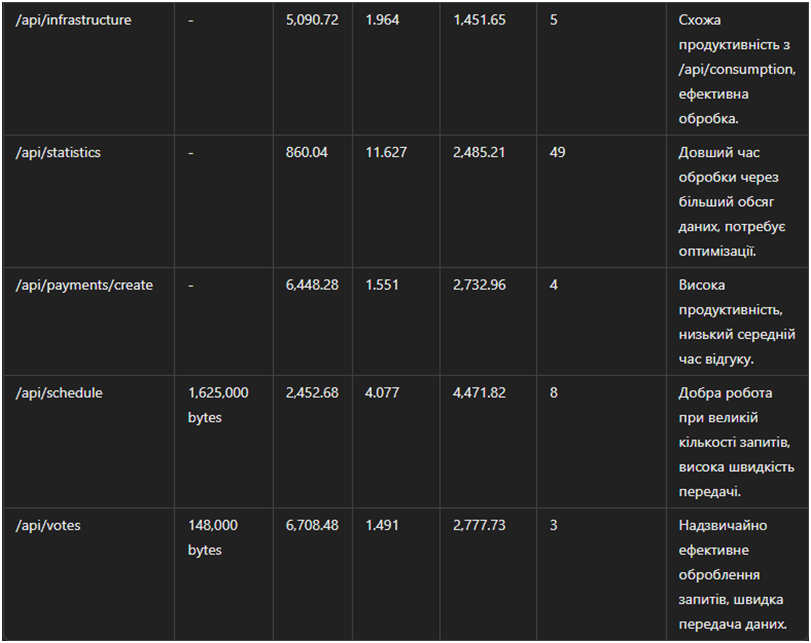


Рис. 4.33 – Оновлення статуса адміністратором

## 4.2 Порівняльний аналіз та швидкодія системи

Таблиця 4.1 - результат тестувань



Аналізуючи таблицю 4.1, можна зробити висновок, що продуктивність API-ендпоїнтів у системі в цілому висока, але деякі аспекти потребують додаткового вдосконалення.

Наприклад, ендпоїнти, такі як /api/votes і /api/payments/create, демонструють надзвичайно високу швидкість обробки запитів та короткий час відгуку, що свідчить про їхню оптимізовану роботу. Аналогічно, /api/collections/active-collections має хорошу продуктивність завдяки низькому середньому часу на запит і ефективній швидкості передачі даних.

Разом із цим, ендпоїнт /api/statistics вимагає уваги. Його середній час на запит (11.627 мс) значно перевищує показники інших ендпоїнтів, що свідчить про необхідність оптимізації. Це пов'язано з складністю логіки та обробки запитів, тому варто розглянути можливість впровадження кешування чи оптимізації запитів до бази даних.

Окремо варто виділити /api/schedule, який працює з великими обсягами даних (1,625,000 байт), але демонструє високу швидкість передачі (4,471.82 Кбайт/с) та помірний час обробки (4.077 мс). Це свідчить про ефективну роботу ендпоїнта навіть при значних навантаженнях, але є потенціал для подальшого зниження обсягу переданих даних шляхом стиснення.

Система в цілому демонструє стійкість до високих навантажень і забезпечує швидку обробку більшості запитів. Однак необхідно продовжувати моніторинг продуктивності та проводити точкові оптимізації для покращення найповільніших ендпоїнтів, таких як /api/statistics. Це дозволить зберігати загальну продуктивність системи на високому рівні навіть за умов підвищеного навантаження.

Таблиця 4.2 - результат тестувань 

Для загального аналізу зручності додатку порівняно з виконанням тих самих функцій у месенджері (див Таблиця 4.2). Для цього було максимально адаптовано всі початкові дані для месенджера “Telegram”. Створено чат, отримані необхідні документи та дані. Загальний аналіз з урахуванням отриманої статистики:

1. **Розрахунок внеску по площі**: У додатку автоматизація дозволяє знизити час на розрахунок з 30 секунд до 1-2 секунд на учасника. Це скорочення становить близько **95%**, що значно підвищує ефективність та зручність виконання завдання.

2. **Перевірка правильності даних**: Перехід від 2-3 хв на учасника у месенджері до 1-2 секунд у додатку забезпечує скорочення часу на **98-99%**, що покращує оперативність перевірки даних.

3. **Оплата за розрахунками**: У додатку час виконання становить 3-5 хв на платіж порівняно з 5-7 хв у месенджері, що складає скорочення близько **30-50%**. Хоча це не така велика економія часу, але зручність підтвердження платежів значно підвищується.

4. **Оновлення та корекція даних**: У додатку це займає 10 секунд, що є значне скорочення від 5-10 хв у месенджері, що забезпечує зменшення часу на **90-95%**.

5. **Написання повідомлень**: У додатку цей процес займає 5 секунд, як і у месенджері, тому відмінність нульова — **0%**.

6. **Голосування**: Час на голосування скорочується з 40-50 секунд у месенджері до 20-25 секунд у додатку, що складає економію часу на **50-60%**.

7. **Перегляд графіків відключень світла та інфраструктури**: У додатку час на збір даних — 20 секунд замість 1-2 хв у месенджері, що забезпечує скорочення часу на **80-90%**.

8. **Перегляд статистики витрат**: У додатку перегляд статистики займає 20-30 секунд, тоді як у месенджері — 3-5 хв, що є скороченням часу на **75-80%**.

# ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі було проведене тестування інформаційної системи управління та комунікацій для ОСББ, а також здійснений порівняльний аналіз продуктивності та зручності її використання.

Тестування системи включло перевірку основних функціональних можливостей системи, щоб забезпечити її правильну роботу та відповідність вимогам. Для документування результатів тестування були використані скріншоти роботи програми, які підтверджують коректність виконання всіх основних функцій, таких як автоматизація фінансового обліку, організація голосувань, а також моніторинг інфраструктури та комунікація між користувачами.

Результати тестування показують, що система працює стабільно і надійно, з високою швидкістю відгуку на запити користувачів. Це підкреслює ефективність її архітектури та реалізації, що забезпечує зручний досвід користувача.

Аналіз зручності додатку порівняно з виконанням подібних функцій у месенджері показав значні переваги розробленої системи. У той час як месенджер забезпечує базові можливості для комунікації та передачі інформації, система управління для ОСББ пропонує значно ширший спектр функцій, які інтегровані у єдину платформу. Це включає організацію голосувань, автоматизацію фінансових процесів, моніторинг інфраструктури, що значно покращує ефективність управління та взаємодії між співвласниками.

Система має сучасний інтерфейс, що полегшує її використання та забезпечує простоту доступу до потрібних функцій. Порівняльний аналіз показав, що застосунок більш інтуїтивно зрозумілий і адаптований до потреб користувачів ОСББ, ніж стандартні месенджери, які не спеціалізуються на управлінських процесах.

Впровадження інтеграції з платіжними системами, відстеження графіків світла, застосування кращих практик генерації JWT токенів значно розширює можливості системи та підвищує її практичну цінність. Ці нововведення удосконалюють методологію побудови інформаційних систем для ОСББ, яка базується на взаємодії різнорідних функціональних модулів у єдиній уніфікованій платформі. Такий підхід дозволяє забезпечити комплексне управління, підтримку колективного прийняття рішень, а також високий рівень автоматизації, що є значним кроком вперед у порівнянні з існуючими рішеннями на ринку.

# ВИСНОВКИ

В магістерській дисертації виконано комплексний аналіз сучасного стану управління та комунікацій в об'єднаннях співвласників багатоквартирних будинків, визначено основні проблеми, що стоять перед ОСББ, такі як недостатня прозорість фінансових операцій, труднощі комунікації та обмежені можливості для колективного прийняття рішень та застарілість технічних рішень. Це створило підґрунтя для розробки нової інтегрованої інформаційної системи, яка б задовольняла сучасні вимоги ефективного управління та підтримувала оптимізацію ключових процесів.

У процесі дослідження було розроблено концептуальне і технічне обґрунтування архітектури та основних модулів інформаційної системи. Важливими елементами системи є модуль автентифікації із застосуванням сучасних практик методів безпеки JWT токенів, модуль автоматизації голосувань, модуль системи комунікацій, фінансовий модуль, який включає інтеграцію з платіжною системою LiqPay та модуль моніторингу інфраструктури з інтеграцією розкладу відключень світла. Кожен модуль забезпечує не лише окрему реалізацію зазначених аспектів, а й їхню взаємодію, що забезпечує прозорість, ефективність та покращує комунікацію між адміністрацією та співвласниками.

Під час тестування системи було підтверджено її стабільну та надійну роботу, високу швидкість відгуку та зручний інтерфейс, що покращує користувацький досвід. Порівняльний аналіз з іншими системами та інструментами, показав значні переваги розробленої системи завдяки інтеграції численних функціональних модулів у єдину платформу.

Створена інформаційна система є сучасним і комплексним інструментом для управління ОСББ, який відповідає потребам сучасних користувачів і дозволяє значно підвищити ефективність управлінських процесів. Інтеграція різних функціональних модулів у єдину платформу забезпечує ширший спектр можливостей і створює основу для подальшого розвитку та вдосконалення системи в майбутньому.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Від мешканця до власника. Навіщо створювати ОСББ? [Електронний ресурс] / Центр громадського моніторингу та контролю. – 2018. – Режим доступу до ресурсу:<https://naglyad.org/uk/2018/05/07/vid-meshkantsya-do-vlasnika-navishho-stvoryuvati-osbb/>.
2. Статистика і перспективи ОСББ [Електронний ресурс] / Асоціація ОСББ. – Режим доступу до ресурсу:<https://aosbb.kiev.ua/statystyka-perspektyvy-osbb/>.
3. Цифровізація роботи ОСББ в нових реаліях [Електронний ресурс] / Блог ПРООСББ. – Режим доступу до ресурсу:<https://proosbb.info/node/795>.
4. Сайт ОСББ-online в Україні. ОСББ Бухгалтерія, програма для обліку ОСББ [Електронний ресурс]. – Режим доступу до ресурсу:<https://osbb-online.com/>.
5. Моє ОСББ. Керуємо будинком, дружимо з мешканцями [Електронний ресурс]. – Режим доступу до ресурсу:<https://osbb.online/>.
6. Лавріщева К.М. Програмна інженерія. – К.: Видавництво КНУ, 2008. – 319 с. – ISBN 978-966-02-5052-9. – Режим доступу до ресурсу:<https://csc.knu.ua/en/library/books/lavrishcheva-6.pdf>.
7. Крижановський Є.М., Ящолт А.Р., Жуков С.О., Козачко О.М. Моделювання бізнес-процесів та управління ІТ-проектами. – Вінниця: ВНТУ, 2018. – Режим доступу до ресурсу:<https://duikt.edu.ua/uploads/l_2150_84887870.pdf>.
8. About the REST API [Електронний ресурс] / GitHub Docs. – 2022. – Режим доступу до ресурсу:<https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api?apiVersion=2022-11-2>.
9. What is REST? REST API Tutorial [Електронний ресурс]. – Режим доступу до ресурсу:<https://restfulapi.net/>.
10. Documentation for Visual Studio Code [Електронний ресурс]. – Режим доступу до ресурсу:<https://code.visualstudio.com/docs>.
11. Index | Node.js v23.3.0 Documentation [Електронний ресурс]. – Режим доступу до ресурсу:<https://nodejs.org/docs/latest/api/>.
12. Грег Лим. Beginning Node.js, Express & MongoDB Development.
13. Casciaro M. Node.js Design Patterns. Design and Implement Production-Grade Node.js Applications Using Proven Patterns and Techniques.
14. Express - Node.js web application framework [Електронний ресурс]. – Режим доступу до ресурсу:<https://expressjs.com/>.
15. MongoDB Documentation [Електронний ресурс]. – Режим доступу до ресурсу:<https://www.mongodb.com/docs/>.
16. WebSocket [Електронний ресурс]. – Режим доступу до ресурсу:<https://uk.javascript.info/websocket>.
17. React [Електронний ресурс]. – Режим доступу до ресурсу:<https://react.dev/>.
18. Minnick C. Beginning ReactJS Foundations: Building User Interfaces with ReactJS [Електронний ресурс]. – Режим доступу до ресурсу:<https://dl.ebooksworld.ir/books/Beginning.ReactJS.Foundations.Chris.Minnick.Wiley.9781119685548.EBooksWorld.ir.pdf>.
19. Facebook's Zuckerberg: 'The biggest mistake we've made as a company is betting on HTML5 over native.' [Електронний ресурс] / VentureBeat. – Режим доступу до ресурсу:<https://venturebeat.com/mobile/facebooks-zuckerberg-the-biggest-mistake-weve-made-as-a-company-is-betting-on-html5-over-native/>.
20. Починаючи | Axios Docs [Електронний ресурс]. – Режим доступу до ресурсу:<https://axios-http.com/uk/docs/intro>.
21. JSON Web Token Introduction [Електронний ресурс] / jwt.io. – Режим доступу до ресурсу:<https://jwt.io/introduction>.
22. LiqPay - Ласкаво просимо в LiqPay! [Електронний ресурс]. – Режим доступу до ресурсу:<https://www.liqpay.ua/doc>.