Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмної системи управління групою будинків»

Виконав: студент 4 курсу
групи 5ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Тимчук В.Л.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЗІ Дудатьєв А.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ О.Н.Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 - Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
21 березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Тимчуку Владиславу Леонідовичу

1. Тема роботи - «Розробка програмної системи управління групою будинків»

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: кількість параметрів контролю інженерної інфраструктури – до 40, кількість користувачів – до 5000, кількість типів подій у системі – до 50, частота оновлення даних – до 1 секунди, кількість типів заявок мешканців – до 20, кількість підключених пристроїв – до 1000, час реакції системи – до 200 мс, обсяг даних для зберігання – до 1 ТБ.

4. Зміст розрахунково-пояснювальної записки: вступ, аналітичний огляд предметної галузі програмних систем управління групою будинків, критерії вибору інструментальних засобів, постановка задачі та проектування програмної системи, розробка архітектури та реалізація основних модулів, тестування та впровадження, оцінка ефективності та економічна доцільність, висновки, список використаних джерел, додатки.

5. Перелік графічного матеріалу: мета та задачі роботи, постановка задачі проектування програмної системи, розробка архітектури та реалізація основні модулів, тестування та впровадження, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Хошаба О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обгрунтування актуальності розробки системи та постановка задач	25.03.2025-03.04.2025	вик.
2	Проектування модулів, розробка алгоритмів, структури та моделей з управління групою будинків	04.04.2025-14.04.2025	вик.
3	Вибір середовища та розробка програмного забезпечення з управління групою будинків	15.04.2025-05.05.2025	вик.
4	Тестування графічної частини, впровадження програмної системи управління групою будинків	06.05.2025-19.05.2025	вик.
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025	вик.

Студент _____ Тимчук В.І
(підпис) (прізвище та ініціал

Керівник бакалаврської кваліфікаційної роботи _____ Хошаба О.М
(підпис) (прізвище та ініціал

АНОТАЦІЯ

УДК 004.92:004.94

Тимчук В.Л. Розробка програмної системи управління групою будинків : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 130 С.

На укр. мові. Бібліогр. : 48 назв ; рис. : 36; табл. 9.

У бакалаврській кваліфікаційній роботі представлено розробку програмної системи управління групою будинків, де виконано аналіз предметної області, визначено технічні вимоги до системи, розроблено модулі обліку, моніторингу, інтерактивної взаємодії та прогнозного обслуговування. Реалізовано програмну архітектуру із підтримкою інтеграції з IoT, SCADA, BMS та обґрунтовано актуальність розробки програмної системи для управління групою будинків. Проведено детальний аналіз предметної області та існуючих технологічних рішень, що дозволило сформулювати вимоги до функціональності системи. Розроблено архітектуру програмної системи, що включає модулі обліку, моніторингу, взаємодії з мешканцями та прогнозного обслуговування. Проведено тестування системи, підтверджено її ефективність, надійність та масштабованість. Запропоноване рішення має практичну цінність та може бути адаптовано для широкого спектра житлових об'єктів.

Ключові слова: управління будинками, SCADA, BMS, IoT, прогнозування, інженерна інфраструктура, цифрова система.

ANNOTATION

Tymchuk V. Development of a software system for managing a group of houses : bachelor's thesis on the specialty 121 Software engineering, educational program - software engineering. Vinnytsia: VNTU, 2025. 130 with.

In Ukrainian language. Bibliographer : 48 titles; Fig. : 36; table 9.

The bachelor's qualification work presents the development of a software system for managing a group of houses, where an analysis of the subject area was performed, technical requirements for the system were determined, accounting, monitoring, interactive interaction and predictive maintenance modules were developed. A software architecture was implemented with support for integration with IoT, SCADA, and BMS, and the relevance of developing a software system for managing a group of houses was substantiated. A detailed analysis of the subject area and existing technological solutions was conducted, which allowed for the formulation of requirements for the system's functionality. The software system's architecture was developed, including accounting, monitoring, interaction with residents and predictive maintenance modules. The system was tested, and its efficiency, reliability and scalability were confirmed. The proposed solution has practical value and can be adapted for many residential facilities.

Keywords: building management, SCADA, BMS, IoT, forecasting, engineering infrastructure, digital system.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному урбанізованому середовищі, що характеризується інтенсивним розвитком житлових кварталів, особливого значення набуває ефективне управління групою будинків. Так, зі зростанням чисельності населення та складністю інженерної інфраструктури житлових об'єктів, виникає необхідність у впровадженні програмних систем, які забезпечують автоматизоване керування, облік ресурсів, контроль технічного стану, інтерактивну взаємодію з мешканцями та високий рівень безпеки. Тема розробки програмної системи управління групою будинків є актуальною через потребу підвищення прозорості, ефективності та інтеграції цифрових технологій у сферу житлово-комунального господарства.

Зростання популярності концепцій Smart Home, Smart Building і Smart Community формує нові вимоги до цифрової інфраструктури. Інтеграція інтелектуальних систем на рівні групи будинків дозволяє реалізувати централізоване управління освітленням, опаленням, вентиляцією, безпекою, доступом, енергоспоживанням, та забезпечити моніторинг в реальному часі з можливістю дистанційного втручання. Особливої актуальності набувають SCADA- та BMS-системи, які слугують основою для створення багаторівневих інформаційно-керуючих екосистем. Вони надають змогу не лише контролювати стан технічних підсистем, а й впроваджувати прогнозне обслуговування, зменшуючи ризики аварійних ситуацій та експлуатаційні витрати.

Тому, основними проблемами, які визначають актуальність даного дослідження, є:

- відсутність уніфікованого підходу до управління групою будинків із залученням сучасних цифрових технологій;

- низький рівень інтеграції між технічними підсистемами та інформаційними службами;

- недостатня взаємодія між мешканцями та адміністрацією через відсутність зручних цифрових сервісів;

- високі витрати на обслуговування інженерної інфраструктури через відсутність автоматизованого моніторингу та систем прогнозного аналізу;

- слабка захищеність даних, що циркулюють у таких системах, через відсутність комплексної моделі безпеки.

Для вирішення вищезазначених проблем пропонується:

- розробити багаторівневу програмну архітектуру, яка поєднує IoT, SCADA та BMS з мобільними та веб-застосунками;

- впровадити методи інтеграції з використанням REST API, WebSocket, MQTT та інших сучасних протоколів;

- реалізувати механізми обліку, моніторингу та диспетчеризації на основі відкритих технологічних стандартів;

- забезпечити безпечне зберігання та передачу даних із застосуванням TLS/SSL, JWT, RBAC;

- розробити інтерфейси для взаємодії мешканців з адміністрацією (подання заявок, перегляд нарахувань, сповіщення, тощо);

- впровадити аналітичні модулі, які здійснюють обробку даних у реальному часі з метою прогнозування відмов та оптимізації витрат.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання є підвищення ефективності управління групою будинків за рахунок розробки програмної системи, що поєднує автоматизоване керування інженерною інфраструктурою, цифрову взаємодію з мешканцями, аналітичну обробку даних та засоби прогнозного обслуговування на основі сучасних інформаційних технологій.

Відповідно до поставленої мети виконані наступні задачі:

- провести аналітичний огляд предметної галузі програмних систем управління групою будинків;
- визначити критерії вибору інструментальних засобів для розробки програмних систем управління групою будинком;

- провести дослідження процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків;
- виконати проектування архітектурної моделі програмної системи управління групою житлових будинків;
- виконати проектування основних функціональних модулів системи;
- розробити програмний код для Billing Module;
- розробити програмний код для Request Module;
- розробити програмний код для Voting Module;
- виконати тестування програмного модуля Billing Module;
- виконати тестування програмного модуля Request Module;
- виконати тестування програмного модуля Voting Module.

Об’єктом дослідження є процес цифрового управління інженерною інфраструктурою та взаємодії між мешканцями та адміністрацією у межах групи житлових будинків.

Предметом дослідження є методи, моделі та програмні засоби, що забезпечують інтегроване керування технічними та інформаційними підсистемами групи житлових будинків.

Методи дослідження. У процесі дослідження використовувались: теорія автоматизованих систем управління для формалізації архітектури; методи програмної інженерії для побудови модулів; теорія баз даних для розробки сховища інформації; методи аналітики та машинного навчання для прогнозування збоїв; протоколи комунікації (MQTT, REST, WebSocket) для реалізації взаємодії систем; засоби верифікації та тестування для оцінки працездатності системи.

Новизна отриманих результатів:

1. Запропоновано метод обробки подій у реальному часі для програмної системи управління групою житлових будинків, який враховує тип запиту користувача, пріоритет дії, та статус інфраструктурного ресурсу за допомогою WebSocket-комунікацій, що дозволило підвищити швидкодію оновлення інформації в системі на 32% порівняно з традиційною REST-архітектурою.

2. Запропоновано модель архітектури програмної системи управління групою житлових будинків, особливість якої полягає у поєднанні ролеорієнтованого доступу користувачів із централізованим обліком даних про нарахування, звернення, голосування та комунікації, що дає можливість підвищити ефективність управління та прозорість взаємодії між мешканцями й адміністрацією ОСББ.

3. Подальшого розвитку отримав метод побудови аналітичних панелей для житлового фонду у вебінтерфейсі, у якому, на відміну від існуючих, застосовано інтеграцію з внутрішнім сховищем статистичних даних у форматі JSON + TypeScript-звітності, що дозволило автоматизувати формування звітів для фінансових нарахувань, відгуків мешканців та ефективності рішень зборів з точністю до 29%.

Практична цінність отриманих результатів. Практична цінність роботи полягає у створенні програмної системи, що забезпечує автоматизацію управління групою житлових будинків, зменшення витрат на обслуговування, підвищення комфорту мешканців та покращення взаємодії з адміністрацією. Розроблена система інтегрується з існуючими IoT-пристроями та SCADA/BMS-платформами, підтримує сучасні вимоги до безпеки та має модульну структуру, що дозволяє масштабувати її відповідно до потреб конкретного житлового комплексу.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Автору належать такі результати: архітектура програмної системи; розроблені алгоритми; модулі клієнтської частини; модулі серверної частини, тестування модулів.

Апробація результатів роботи. Результати роботи були представленні на конференції «Achievements of Science and Applied Research, Dublin, Ireland (2025)».

Публікації. Результати роботи були опубліковані в матеріалах конференції – «Achievements of Science and Applied Research, Dublin, Ireland (2025)» [1].

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналітичний огляд предметної галузі програмних систем управління групою будинків

У сучасному урбанізованому середовищі підвищується потреба в ефективному управлінні житловим фондом, особливо в умовах зростання кількості багатоквартирних будинків та створення об’єднаних груп житлових споруд. Автоматизація процесів управління групою будинків передбачає впровадження програмних систем, які здатні не лише забезпечувати облік ресурсів, а й здійснювати цифровий моніторинг, технічне обслуговування, прийом і обробку звернень мешканців, а також контроль за інженерними системами в реальному часі (табл. 1.1). Ці потреби формують окрему предметну галузь, яка охоплює низку міждисциплінарних напрямів, зокрема інформаційні технології, енергетичне управління, телеметрію та урбаністику.

Однією з базових концепцій, що лежать в основі розробки сучасних систем управління будинками, є концепт «розумного будинку» (англ. Smart Home), який в широкому розумінні трансформується в Smart Building та Smart Community. Основна ідея полягає у застосуванні інтелектуальних технологій для автоматизованого керування різноманітними підсистемами будівлі: освітленням, вентиляцією, системами безпеки, водопостачанням, опаленням тощо [2, 3]. Згідно з сучасними тенденціями, акцент зміщується з індивідуального керування до колективного - на рівні будинкових об'єднань та житлових районів.

Так, впродовж останніх двох десятиліть концепція автоматизованого керування інфраструктурою житлових і громадських будівель зазнала суттєвих змін завдяки стрімкому розвитку цифрових технологій, Інтернету речей (IoT), машинного навчання, розподілених обчислень та хмарних платформ. У цьому контексті особливого значення набули базові концепції Smart Home, Smart Building і Smart Community, які відображають поступове розширення масштабу застосування інтелектуальних систем - від рівня окремого житлового приміщення

до інтегрованих цифрових екосистем, що охоплюють групи будинків або навіть цілі житлові райони.

Таблиця 1.1 – Основні концепти та технології в управлінні групою будинків

Концепт / технологія	Опис функціональності	Рівень впровадження	Тип взаємодії
Smart Home / Building	Автоматизація освітлення, клімату, безпеки	Локальний	Мешканець ↔ система
IoT	Збір та передача даних з сенсорів	Локальний / централізований	Обладнання ↔ хаб ↔ сервер
SCADA	Централізоване керування технічними системами	Централізований	Адміністратор ↔ інженерні системи
BMS	Повне керування інфраструктурою будівель	Централізований	Інтерфейс ↔ всі підсистеми
Прогнозна аналітика	Виявлення відхилень, попередження про несправності	Централізований	Алгоритм ↔ база даних
Цифрова реєстрація подій	Прийом звернень мешканців, фіксація подій	Централізований	Користувач ↔ система
Веб-/мобільні додатки	Інтерфейс для мешканців та адміністраторів	Клієнтський	Користувач ↔ додаток

Тому, початковим і найбільш локалізованим рівнем інтелектуального керування є Smart Home - концепція, що передбачає впровадження автоматизованих та дистанційно керованих систем у межах однієї житлової одиниці (квартира, будинок), з метою підвищення комфорту, безпеки та

енергоефективності [2, 3]. Архітектура типового «розумного дому» охоплює низку взаємопов’язаних модулів:

- сенсорна інфраструктура, яка включає датчики руху, температури, вологості, диму, вібрацій, відкриття/закриття вікон та дверей;

- системи керування освітленням з можливістю регулювання інтенсивності, кольору або автоматичного вмикання/вимикання;

- клімат-контроль на основі термостатів, розумних кондиціонерів та опалювальних пристроїв;

- системи безпеки, де є відеоспостереження, охоронна сигналізація, біометричний доступ;

- інтеграція з мобільними застосунками або голосовими асистентами (Google Assistant, Alexa, Siri) для зручності користування.

Вся ця інфраструктура будується на основі локального хабу або хмарної платформи, яка забезпечує синхронізацію, обробку даних та взаємодію між пристроями. Поширення стандартів бездротового зв’язку (Wi-Fi, ZigBee, Z-Wave, Bluetooth LE) забезпечує масштабованість та мінімальне втручання у будівельні конструкції [4, 5].

Проте функціональність Smart Home обмежується переважно приватним простором, не охоплюючи спільні зони користування чи колективні інженерні мережі. Це зумовлює перехід до концептуально ширшого рівня - Smart Building.

Концепція Smart Building також передбачає створення єдиної інтелектуальної екосистеми в межах цілої будівлі - житлової, комерційної або адміністративної - шляхом централізованого або децентралізованого керування її внутрішньою інфраструктурою [6]. В основі такого підходу лежить інтеграція всіх інженерно-технічних систем, включаючи:

- системи водопостачання та каналізації;

- електропостачання та аварійного енергозабезпечення;

- вентиляції та кондиціонування;

- ліфтового обладнання;

- пожежної безпеки;

- смарт-відеоспостереження та контролю доступу.

Основною відмінністю Smart Building від Smart Home є наявність BMS-платформи (Building Management System) - спеціалізованого програмно-апаратного комплексу, який забезпечує диспетчеризацію, моніторинг і керування всіма підсистемами у режимі реального часу. BMS працює на основі SCADA-механізмів та дозволяє не лише отримувати інформацію про стан обладнання, а й автоматично запускати необхідні сценарії у випадку виявлення порушень (наприклад, автоматичне відключення живлення у разі витоку води або перевищення температури у щитовій кімнаті) [7, 8].

Ключовим елементом є аналітична надбудова, яка на основі накопичених даних дозволяє оцінювати ефективність використання ресурсів, виявляти аномалії, прогнозувати зношення обладнання, оптимізувати графіки технічного обслуговування. Застосування алгоритмів машинного навчання в рамках BMS сприяє переходу до моделі predictive maintenance, що значно знижує витрати та збільшує життєвий цикл систем [9, 10].

На відміну від приватного використання Smart Home, концепція Smart Building орієнтована на колективне користування інфраструктурою, що робить її релевантною для будинків, об’єднаних у ОСББ або керованих управляючими компаніями.

Наступним етапом еволюції інтелектуального управління є Smart Community - система, що охоплює сукупність будинків та прилеглої інфраструктури, включаючи паркування, під’їзди, спортивні та дитячі майданчики, майданчики для збору сміття, зарядні станції для електротранспорту, тощо [11]. У цьому випадку функціональність систем управління розширюється до інтеграції на рівні житлового масиву або мікрорайону, що потребує уніфікованих стандартів, високої масштабованості та надійного інформаційного середовища.

Сутність Smart Community полягає у формуванні цифрового шару над фізичним середовищем проживання, що включає:

- інтеграцію декількох BMS-систем у єдину хмарну платформу;

- синхронізацію сервісів управління та технічного обслуговування на рівні громади;

- розширені цифрові сервіси для мешканців: мобільні застосунки, чат-боти, голосові асистенти, кабінет мешканця, електронне голосування;

- взаємодію з міською інфраструктурою (розклад транспорту, карта заторів, аварійні служби, тощо).

Розвиток концепції Smart Community відповідає загальносвітовій парадигмі побудови Smart City, де мікроспільноти виступають базовими структурними одиницями цифрового міського середовища. За такої моделі мешканці виступають не лише користувачами послуг, а й активними учасниками прийняття рішень, що сприяє розвитку локальної демократії, підвищенню прозорості, соціальної згуртованості та якості життя [12, 13].

Всі ці розглянуті три концепції - Smart Home, Smart Building і Smart Community - не є взаємовиключними, а становлять ієрархічну структуру, де кожен вищий рівень узагальнює та доповнює нижчий:

- Smart Home є локальним модулем Smart Building;

- Smart Building виступає системоутворюючим елементом Smart Community;

- Smart Community забезпечує координацію між множинними будинковими системами, об'єднаними єдиною інформаційною платформою.

Тому, на практиці реалізація таких систем базується на сучасних IT-технологіях: хмарних обчисленнях (Azure, AWS, Google Cloud), платформах IoT (ThingSpeak, OpenHAB, Home Assistant), протоколах MQTT/CoAP/HTTP, а також аналітичних сервісах на основі Big Data. У межах європейських програм цифрової трансформації, зокрема Horizon Europe, активізуються дослідження, спрямовані на уніфікацію протоколів взаємодії, розробку етичних стандартів конфіденційності та безпеки у таких системах [14–16].

У цьому контексті дедалі ширше застосування знаходять технології Інтернету речей (IoT), які дозволяють інтегрувати різні сенсорні пристрої та виконавчі механізми в єдину інформаційно-керуючу інфраструктуру [4, 5]. Наприклад, завдяки IoT можна автоматично збирати дані про споживання води,

тепла та електроенергії, контролювати температуру в підвальних приміщеннях, виявляти витоки, передавати сповіщення у випадку аварій або надзвичайних ситуацій. Застосування таких підходів значно підвищує оперативність реагування, знижує витрати на обслуговування та сприяє підвищенню прозорості взаємодії між мешканцями та адміністрацією ОСББ [6].

Іншим вагомим напрямом є використання систем SCADA (Supervisory Control and Data Acquisition) у сфері житлово-комунального господарства. SCADA забезпечує централізоване спостереження та керування інженерними мережами, такими як теплові пункти, насосні станції або мережі електропостачання. У поєднанні з сучасними засобами візуалізації даних (дашбордами, графіками, схемами) SCADA-системи дозволяють оперативно виявляти аномалії в роботі обладнання та вживати відповідних заходів [7].

У сучасних умовах цифровізації управління інженерною інфраструктурою житлово-комунального господарства набуває особливого значення впровадження систем диспетчерського контролю та збору даних, які забезпечують моніторинг, управління, облік і діагностику технічних процесів у реальному часі. Одним із найпоширеніших технологічних рішень, що відповідає таким вимогам, є SCADA-системи (Supervisory Control and Data Acquisition). Вони є базовим елементом архітектури автоматизованого управління інженерними мережами й широко використовуються у промисловості, енергетиці, водопостачанні, теплопостачанні, вентиляції та охоронних системах, а останнім часом активно впроваджуються й у сфері житлово-комунального господарства [6].

SCADA - це клас автоматизованих систем, призначених для дистанційного моніторингу, збору даних, керування та контролю технічних об’єктів і процесів. Основною функцією таких систем є забезпечення оперативної взаємодії між фізичними пристроями (датчиками, виконавчими механізмами) та диспетчерським або серверним рівнем керування, що забезпечує централізоване управління великими розподіленими інфраструктурами [7].

Класична архітектура SCADA-системи включає такі компоненти:

- периферійні пристрої (датчики та актуатори), де безпосередньо знімають показники або виконують керувальні дії на об’єктах;

- програмовано-логічні контролери (PLC) або RTU (Remote Terminal Units), що агрегують дані від сенсорів, обробляють сигнали, передають їх далі;

- мережеве середовище передачі даних, де канали зв’язку, що з’єднують контролери з сервером SCADA;

- серверна частина (HMI/SCADA-сервери), що здійснює обробку, зберігання, візуалізацію та архівацію даних, реалізує логіку прийняття рішень;

- інтерфейс диспетчера (Human Machine Interface, HMI), що являє собою візуальне середовище для оператора, який може відслідковувати стан систем у реальному часі, керувати режимами та реагувати на події.

Завдяки модульному принципу побудови та універсальності протоколів обміну (Modbus, OPC, DNP3, IEC 60870-5-104), SCADA-системи можна масштабувати та адаптувати під різноманітні сфери застосування, включаючи ЖКГ.

Сфера застосування SCADA в житлово-комунальному господарстві полягає в наступному. У сфері ЖКГ впровадження SCADA-систем передусім спрямоване на забезпечення автоматизованого управління інженерними мережами житлового фонду, зокрема:

- електропостачанням (моніторинг навантаження, аварійні відключення);

- теплопостачанням (регулювання подачі тепла, контроль температурних режимів);

- водопостачанням і водовідведенням (контроль тиску, витоку, рівня наповнення резервуарів);

- вентиляційними та протипожежними системами (датчики диму, вогню, температури);

- ліфтовими системами (моніторинг зупинок, виявлення аварійних ситуацій).

Тому, завдяки інтеграції SCADA у ці підсистеми можлива безперервна діагностика обладнання, зменшення кількості аварій, скорочення часу реакції на

інциденти, оптимізація витрат на обслуговування та підвищення енергоефективності житлових об’єктів [8].

Наприклад, у системах теплопостачання SCADA дозволяє здійснювати автоматичне регулювання температури теплоносія залежно від зовнішньої температури, попереджати перевантаження мереж, виявляти тепловтрати, що забезпечує зменшення споживання енергії без втрати комфорту для мешканців [9].

У сфері водопостачання SCADA допомагає контролювати рівні наповнення баків, тиск у трубопроводах, витрати, а також забезпечувати віддалене відкриття/закриття засувок у разі аварії. Це особливо важливо в багатоквартирних будинках або житлових кварталах, де фізичний доступ до обладнання може бути ускладненим.

Функціональні можливості SCADA-систем у ЖКГ полягає в наступному. Ключові функціональні можливості SCADA у сфері житлово-комунального господарства можна умовно розділити на такі групи:

- моніторинг у реальному часі, де контроль усіх підсистем (електрика, тепло, вода) через графічні панелі з індикацією аварій, графіками, анімацією та журналами подій;

- реєстрація та архівація даних, де існує збереження історії зміни параметрів для подальшого аналізу, складання звітів, формування статистики;

- аналіз та візуалізація, де побудова теплових карт, динамічних графіків, тенденцій зміни параметрів, розрахунок KPI;

- керування та сценарії реагування, де запуск сценаріїв автоматичного відключення, регулювання чи перемикання між режимами роботи у разі виявлення аномалій;

- оповіщення, де системи повідомлень (SMS, email, push) при досягненні порогових значень, виході за межі нормативу або спрацюванні аварійних датчиків;

- дистанційний доступ, де використання вебінтерфейсів або мобільних застосунків для моніторингу та керування без фізичної присутності.

Взаємодія SCADA з іншими компонентами цифрової інфраструктури полягає в наступному. SCADA-системи можуть бути інтегровані з іншими інформаційними системами, такими як:

- BMS (Building Management System), що забезпечують централізоване керування будівлею, де SCADA виступає нижнім рівнем технічної реалізації;

- CRM / ERP системи ОСББ або керуючої компанії, де для передачі показників, обліку подій, формування фінансових звітів;

- IoT-платформи, де для розширення SCADA новими датчиками та аналітичними сервісами;

- GIS-системи, де для просторової візуалізації та прив’язки до карти міста або мікрорайону.

Синергія SCADA з цими платформами створює уніфіковану інформаційно-керуючу екосистему, де кожен компонент доповнює інший, забезпечуючи повну прозорість, надійність та контроль над інфраструктурою.

До переваг та викликів впровадження SCADA в ЖКГ полягають в наступному:

- підвищення точності обліку та достовірності даних;

- зниження людського фактора та витрат на обслуговування;

- прискорення реакції на аварійні ситуації;

- можливість переходу до прогнозного обслуговування (Predictive Maintenance);

- покращення якості надання послуг мешканцям;

- формування цифрового архіву подій для юридичного або аналітичного використання.

До викликів відносяться:

- висока початкова вартість впровадження (контролери, ПЗ, сервери);

- необхідність підготовки кваліфікованого персоналу;

- проблеми з кібербезпекою (неавтентифікований доступ, шкідливе втручання);

- необхідність модернізації існуючої інфраструктури під нові цифрові стандарти.

Особливу увагу слід приділити BMS-системам (Building Management Systems), які представляють собою розширений варіант SCADA, орієнтований саме на управління будівлями. BMS об’єднує апаратне забезпечення та програмні засоби для контролю освітлення, вентиляції, безпеки, енергоспоживання, сигналізації тощо [8]. Зазвичай такі системи інтегруються з мобільними або веб-застосунками, через які адміністратори можуть здійснювати оперативне керування, а мешканці - отримувати дані або створювати запити.

Так, у сучасній практиці експлуатації житлових, комерційних та індустріальних будівель однією з головних проблем є необхідність ефективного керування інженерною інфраструктурою: вентиляцією, електропостачанням, системами пожежної безпеки, освітленням, опаленням, ліфтовим обладнанням тощо. У відповідь на зростання складності та обсягів таких систем, а також потребу у зменшенні витрат і підвищенні енергоефективності, було розроблено концепцію BMS (Building Management System) - інтелектуальну платформу для централізованого управління всією інфраструктурою будівлі [8].

BMS поєднує функціональність SCADA-систем із розширеними можливостями аналітики, автоматизації, інтеграції та зручного інтерфейсу для адміністрації та мешканців. Вона дозволяє досягти оптимального балансу між витратами на обслуговування, комфортом користувачів та сталим енергоспоживанням, що особливо актуально у світлі сучасних екологічних та енергетичних викликів.

Загальне визначення та роль BMS-систем полягає в наступному. Building Management System (BMS) - це автоматизована система, що забезпечує моніторинг, управління, координацію та оптимізацію роботи технічних підсистем будівлі з метою підвищення її функціональної ефективності, зниження експлуатаційних витрат та забезпечення безпеки. Вона охоплює не лише контроль технічних процесів, але й управління ресурсами, ведення журналів подій, аналітику,

віддалений доступ, а також інтеграцію з цифровими сервісами користувачів [9]. Типова архітектура BMS-системи включає:

- периферійні пристрої, де існують сенсори, актуатори, контролери (температури, вологості, присутності, протікання, вогню, руху тощо);

- локальні контролери, що керують підсистемами на рівні конкретного вузла або зони;

- комунікаційні мережі, що передають дані між контролерами, шлюзами та сервером;

центральний сервер, що обробляє інформацію, керує сценаріями, здійснює збереження та архівацію даних;

- інтерфейси користувача (HMI), де існують веб- або мобільні панелі для адміністраторів, операторів, інженерів або мешканців.

Ключова відмінність BMS від SCADA полягає у вищому рівні інтеграції з інформаційними сервісами, підтримці машинного навчання, прогнозної аналітики, розширеного адміністрування прав доступу та сумісності з «розумними» будівельними технологіями (Smart Building) [10].

Функціональні підсистеми BMS полягають в наступному. BMS-системи зазвичай включають у себе ряд модульних функціональних підсистем, які можуть бути розгорнуті окремо або у комбінації, залежно від масштабів та призначення об’єкта. Так, система керування кліматом (HVAC Control) має:

- контроль температури, вологості, вентиляції;

- автоматичну зміну режимів залежно від часу доби або присутності осіб;

- інтелектуальні термостати, рекуперація тепла.

Освітлення (Lighting Control) також має:

- автоматичне ввімкнення/вимкнення світла залежно від присутності або рівня освітленості;

- сценарії енергозбереження;

- зональне керування освітленням у громадських місцях.

До енергоспоживання (Energy Management) системи відноситься:

- моніторинг споживання електроенергії, води, газу;

- аналіз пік-споживання, звіти, рекомендації з оптимізації;

- інтеграція з системами відновлюваної енергії (сонячні панелі, теплові насоси).

Системи безпеки та контролю доступу має:

- відеоспостереження, зчитувачі карток, біометричні системи;

- контроль входів, ліфтів, обмеження доступу до технічних зон;

- автоматичне сповіщення про вторгнення або порушення правил доступу.

Пожежна безпека інфраструктура та сигналізація мають:

- інтеграцію з пожежними датчиками, системами оповіщення, вентиляцією;

- автоматичне закриття протипожежних клапанів, увімкнення димовидалення.

Ліфтові системи та диспетчеризація мають:

- моніторинг роботи ліфтів, журнал зупинок, дистанційне тестування;

- оптимізацію маршрутів залежно від трафіку користувачів.

В цей же час, BMS-системи останнього покоління інтегрують аналітичні модулі, які дозволяють виводити системи управління на якісно новий рівень. На основі історичних та поточних даних формується цифровий профіль об’єкта, виявляються тренди, кореляції та відхилення. Алгоритми машинного навчання (ML) дозволяють реалізувати:

- predictive maintenance, де є прогнозування збоїв або зношення обладнання;

- adaptive control, де є самонавчання сценаріїв поведінки користувачів;

- energy baseline modeling, де є порівняння реального споживання з ідеальними моделями;

- fault detection & diagnostics, де є ідентифікація помилок в роботі систем без втручання людини.

Така інфраструктура, чинникі якої описані вище сприяють підвищенню ефективності обслуговування, зниженню витрат та підвищенню задоволеності користувачів. До цього, BMS-системи можуть бути інтегровані в ширший інформаційний контекст, зокрема:

- з системами SCADA, які забезпечують безпосередній технічний рівень керування;

- з платформами IoT для збирання додаткових даних із побутових сенсорів;

- з ERP/CRM-системами - для передачі даних в адміністративний облік;

- з міськими платформами Smart City, що дозволяє масштабувати керування на рівень району або міста;

- з мобільними застосунками мешканців, що надає їм доступ до даних, можливість подавати запити, відслідковувати розходи.

З точки зору архітектури, системи управління групою будинків зазвичай реалізуються як багаторівневі розподілені системи, що включають датчики (рівень збору даних), контролери (рівень локального керування), сервери (рівень зберігання та обробки), а також клієнтські інтерфейси (веб або мобільні додатки). Важливим аспектом є підтримка віддаленого доступу, що дозволяє здійснювати моніторинг та керування з будь-якої точки, а також ведення цифрового журналу подій - як для технічного обліку, так і для юридичних потреб [9].

Нарешті, сучасні дослідження зосереджуються на впровадженні систем прогнозування, які на основі історичних даних та аналітики дозволяють передбачити можливі відмови обладнання, пікові навантаження в енергомережах або аномалії у споживанні ресурсів [10–13]. Такі моделі, побудовані з використанням машинного навчання, дають змогу впроваджувати превентивне технічне обслуговування, зменшувати кількість скарг мешканців та підвищувати загальний рівень обслуговування.

Таким чином, предметна галузь програмних систем для управління групою будинків характеризується високим ступенем інтеграції між технічними засобами (датчиками, контролерами) та інформаційними технологіями (системи керування, аналітика, мобільні додатки). Основними концептами, що домінують у цій сфері, є Smart Building, IoT, SCADA та BMS, які дозволяють забезпечити ефективний контроль за інфраструктурою, зменшити витрати, покращити обслуговування мешканців та підвищити прозорість процесів. Системи цього класу мають

перспективи розвитку в напрямку адаптивного керування на основі прогнозної аналітики та інтеграції з міською цифровою інфраструктурою.

1.2 Критерії вибору інструментальних засобів для розробки програмних систем управління групою будинком

Проєктування та реалізація програмних систем для управління групою житлових будинків вимагає комплексного підходу до вибору інструментальних засобів, які забезпечують ефективне виконання функціональних вимог, масштабованість, стабільність та безпеку програмного забезпечення. Різноманіття сучасних технологій, мов програмування, фреймворків, баз даних та інструментів для інтеграції із зовнішніми системами створює необхідність в аналітичному обґрунтуванні критеріїв вибору таких засобів у контексті специфіки предметної галузі [19]. При цьому, функціональні вимоги до програмних систем керування будинками (табл. 1.2) відіграють важливу роль під час розробки програмних систем управління групою будинком.

Таблиця 1.2 - Порівняльна характеристика критеріїв вибору інструментальних засобів

Критерій	Вплив на якість ПЗ	Рівень пріоритету	Коментарі щодо застосування
Продуктивність	Високий	Критичний	Забезпечує стабільну роботу під навантаженням
Інтеграційна здатність	Високий	Високий	Визначає ступінь взаємодії з IoT/SCADA/BMS
Безпека	Високий	Критичний	Ключовий фактор довіри користувачів
Модульність	Середній	Високий	Забезпечує адаптивність до змін
Зручність розробки	Середній	Середній	Визначає швидкість створення MVP

Вартість володіння	Середній	Високий	Впливає на економічну ефективність впровадження
Сумісність із платформами	Високий	Високий	Важливо для зручності користувачів
Аналітика та ML	Середній	Середній	Забезпечує прогностичні функції, оптимізацію ресурсів

Для початку розгляду критерієв для інструментальних засобів, що використовуються в галузі розробки програмних систем управління групою будинком необхідно розглянути основні функціональні компоненти об'єкта дослідження. Так, у системах управління житловими об'єктами основними функціональними компонентами є:

- модуль взаємодії з мешканцями через веб або мобільний інтерфейс;
- підсистеми моніторингу показників інженерної інфраструктури (енергоспоживання, водопостачання, вентиляція, сигналізація тощо);
- підсистема цифрового документообігу (звернення, платежі, звіти);
- модуль інтеграції із зовнішніми платформами: SCADA, BMS, IoT-сенсорами;
- аналітичний блок із засобами візуалізації та прогнозної аналітики [20, 21].

Отже, інструментальні засоби повинні підтримувати мультиплатформенність, веборієнтовану архітектуру, RESTful API, роботу з базами даних, асинхронну обробку подій та можливість інтеграції зі сторонніми пристроями й сервісами [22].

На підставі з'ясування основних функціональних компонентів об'єкта дослідження можна приступати до визначення критерієв вибору інструментальних засобів. Так, під час вибору інструментальних засобів для розробки програмних систем управління групою будинком доцільно враховувати такі узагальнені критерії:

- продуктивність, що позначає швидкість обробки даних, масштабованість системи, ефективність при великій кількості користувачів і сенсорних подій [23];
- інтеграційна здатність, що позначає підтримку сучасних протоколів (MQTT, WebSocket, HTTPS), наявність SDK або API для IoT/SCADA-систем [24, 25];

- безпека, що позначає реалізацію шифрування, контроль доступу, автентифікацію, сумісність з політиками GDPR/ISO27001 [26];

- модульність та розширюваність, що позначає можливість адаптації під певну специфіку будинку чи групи будинків, підтримка мікросервісної архітектури [27];

зручність розробки та підтримки, що позначає наявність документації, активної спільноти, підтримку сучасних інструментів CI/CD [28, 29];

- вартість володіння, що позначає відкритість коду (open-source), наявність ціни на ліцензії, проведення навчання персоналу [30];

- сумісність з мобільними/веб-платформами, що позначає підтримку адаптивної верстки, кросбраузерність, фреймворки для веб- та мобільної розробки (React Native, Flutter) [31];

- наявність вбудованих засобів аналітики, що позначає наявність бібліотеки для графіків, інструментів аналітичної обробки, зокрема ML-моделі [32].

Розглянемо деякі визначені критерії вибору інструментальних засобів для розробки програмних систем управління групою будинком більш ретельно.

Критерій продуктивності є одним із найважливіших при виборі інструментальних засобів для створення систем управління групою будинків, оскільки від нього безпосередньо залежить здатність програмного забезпечення функціонувати стабільно за умов високого навантаження. Під продуктивністю розуміється сукупність характеристик, що визначають швидкість обробки запитів, ефективність виконання операцій введення/виведення, здатність до масштабування та забезпечення безперебійної роботи системи при зростанні кількості користувачів та обсягу даних [23].

У контексті систем для керування будинками ключовими аспектами продуктивності є:

- швидкодія серверної частини (backend), що визначається здатністю обробляти десятки або сотні одночасних HTTP-запитів до бази даних, API та зовнішніх сервісів (наприклад, SCADA або IoT-хабів). Продуктивний інструмент повинен мати механізми кешування, підтримку асинхронної обробки, можливість

використання потокових протоколів (наприклад, WebSocket) для подій реального часу;

- швидкість обробки сенсорних подій, де системи отримують в режимі реального часу великі обсяги даних з IoT-пристроїв: температурних сенсорів, датчиків руху, лічильників споживання енергії тощо. У високонавантажених умовах важливо не лише зчитати ці дані, а й обробити, агрегувати та зберегти їх без затримок. Обрані технології повинні забезпечувати низький latency (затримку) та підтримку протоколів MQTT, AMQP, OPC UA;

- масштабованість, де у випадках, коли система розгортається для декількох десятків або сотень будинків, з відповідним зростанням числа користувачів і сенсорних каналів, критично важливою стає здатність масштабувати окремі компоненти: базу даних, API-шлюзи, брокери повідомлень. Це досягається через застосування мікросервісної архітектури, контейнеризації (Docker), оркестрації (Kubernetes) та хмарних рішень (AWS Lambda, Google Cloud Functions);

- продуктивність клієнтської частини (frontend), де – веб- або мобільний інтерфейс повинен працювати плавно, швидко реагувати на запити користувача, мінімізувати час завантаження компонентів, підтримувати локальне кешування, віртуалізацію списків та динамічне оновлення даних через WebSocket або SSE (Server Sent Events);

- моніторинг та стрес-тестування, де важливо, щоб інструментальні засоби дозволяли здійснювати попереднє тестування продуктивності за допомогою таких засобів, як Apache JMeter, Gatling, K6, які допомагають змоделювати навантаження та виявити вузькі місця.

Для забезпечення високої продуктивності рекомендується використовувати такі технології: Node.js або Spring WebFlux на сервері для обробки асинхронних запитів; Redis як високошвидкісний кеш; бази даних з підтримкою шардінгу й реплікації (наприклад, PostgreSQL + Citus, MongoDB); брокери повідомлень (Kafka, RabbitMQ); фреймворки з оптимізованим рендерингом (React, Vue з SSR) [23].

Всі ці компоненти повинні бути гармонійно поєднані, що потребує від розробників глибокого розуміння архітектури програмного забезпечення, взаємодії

компонентів у реальному часі, типів навантаження, а також стратегії горизонтального масштабування для забезпечення fault-tolerance.

Інтеграційна здатність програмної системи управління будинками визначає її спроможність ефективно взаємодіяти з іншими цифровими компонентами, платформами, пристроями та протоколами, які вже використовуються у технічній інфраструктурі об’єкта або плануються до впровадження. Високий рівень інтеграційної здатності забезпечує гнучкість архітектури, її адаптивність до змін середовища, а також здатність до масштабування за рахунок додавання нових функціональних блоків або каналів обміну даними [24].

Тому, одним із ключових завдань при розробці систем управління будинками є забезпечення взаємодії з фізичними пристроями (контролери, сенсори, актуатори), протоколами обміну повідомленнями та сторонніми цифровими системами, як-от SCADA, BMS, ERP чи IoT-платформи. Для цього інструментальні засоби повинні підтримувати сучасні протоколи передачі даних, API-шлюзи, а також SDK (Software Development Kit), які забезпечують швидку інтеграцію без потреби розробляти низькорівневе програмування взаємодії [25].

Також, одним із базових протоколів у сфері Інтернету речей є MQTT (Message Queuing Telemetry Transport) - легкий, енергоефективний протокол публікації-підписки, який широко використовується для надсилання сенсорних даних з пристроїв у хмару або на сервер локального управління. Перевагами MQTT є мінімальні вимоги до ресурсів, стійкість до нестабільного з’єднання, підтримка ієрархії тем та QoS-рівнів доставки повідомлень. Для ефективної інтеграції з цим протоколом інструмент повинен мати підтримку MQTT-брокерів (Mosquitto, EMQX, HiveMQ) та клієнтських бібліотек, зокрема на Java, Python, JavaScript, що дозволяє розробникам легко отримувати й публікувати дані [24].

Іншим поширеним інструментом для обміну даними в реальному часі є WebSocket, який забезпечує двосторонній зв’язок між клієнтом та сервером без необхідності повторного встановлення HTTP-з’єднань. Це критично важливо для інтерфейсів користувача, які вимагають негайного реагування на події: зміни стану датчика, тривожне повідомлення, надходження заявки від мешканця.

Інструментальні засоби повинні включати підтримку серверів із WebSocket (наприклад, Spring WebSocket, Socket.IO) та механізми для обробки навантаження, створення пулів з’єднань, балансування каналів.

Не менш важливою є підтримка протоколів безпечного доступу до API, таких як HTTPS. Оскільки велика частина взаємодій у системі здійснюється через RESTful або GraphQL API, забезпечення безпеки при передачі даних є необхідною умовою. Інструментальні засоби мають надавати можливість налаштовувати TLS-сертифікати, автентифікацію користувачів (OAuth 2.0, JWT), контроль доступу через ACL-політики або ролеву модель авторизації.

В умовах мультисистемного середовища важливою перевагою стає наявність SDK або API для взаємодії з SCADA/BMS-системами. Це дозволяє здійснювати обмін даними безпосередньо з автоматизованими вузлами управління (наприклад, контролерами Siemens, Schneider Electric, Advantech), використовуючи такі специфікації, як OPC UA (Unified Architecture), Modbus TCP або BACnet. SDK надають високорівневі інтерфейси, які абстрагують низькорівневі особливості обміну, дозволяючи зосередитися на бізнес-логіці системи.

Окрему категорію становлять інтеграційні шини та брокери подій, які забезпечують зв’язок між сервісами та додатками на основі подієвої архітектури (event-driven architecture). Використання Apache Kafka, RabbitMQ або NATS дозволяє передавати події з високою пропускною здатністю, зберігати черги повідомлень, забезпечувати гарантовану доставку, масштабування обробки на окремі воркери.

Серед додаткових інструментів, що підвищують інтеграційні можливості, варто зазначити:

- GraphQL як гнучкий інтерфейс для запитів до API, який дозволяє клієнтам самостійно формувати структуру відповідей;

- gRPC, що використовується для високопродуктивної взаємодії мікросервісів через стислі двійкові повідомлення;

- Webhook, що використовується як засіб підписки на події для зовнішніх систем (наприклад, при отриманні заявки від мешканця, ця подія може передаватися до Telegram-бота або CRM-системи).

Інтеграційна здатність також включає підтримку ETL-інструментів (Extract, Transform, Load) та iPaaS-платформ (Integration Platform as a Service), таких як Apache NiFi, Zapier, Mulesoft, що дозволяє будувати складні канали передачі, трансформації та маршрутизації даних між системами без ручного кодування. У разі потреби обробки великих обсягів структурованих і неструктурованих даних можна використовувати API до систем великих даних (BigQuery, Hadoop HDFS, Apache Spark).

На рівні архітектури системи важливо забезпечити API-шлюзи, які агрегують доступ до мікросервісів, здійснюють маршрутизацію запитів, трансформацію протоколів, кешування, обмеження частоти запитів (rate limiting), а також логування викликів. Відомими реалізаціями є Kong, Tyk, Amazon API Gateway.

Таким чином, інтеграційна здатність визначає відкритість системи до зовнішнього світу, її здатність до адаптації, модернізації та масштабування без повної переробки архітектури. У проєктах із високим ступенем складності цей критерій набуває стратегічного значення для забезпечення довгострокової життєздатності цифрової платформи.

Критерій безпеки у програмних системах управління групою будинків є надзвичайно важливим з огляду на обробку конфіденційної інформації мешканців, взаємодію з критичною інфраструктурою та підключення до відкритих мереж. Забезпечення цілісності, конфіденційності та доступності даних вимагає впровадження багаторівневих заходів кібербезпеки, що відповідають сучасним стандартам і нормативам [26]. Так, системи управління житловими будинками працюють з різними типами даних: персональними (ПІБ, адреси, фінансові операції), технічними (показники датчиків, журнали подій), операційними (запити користувачів, історія обслуговування), що робить їх привабливою ціллю для потенційних атак. У зв’язку з цим безпека повинна бути вбудована у всі

компоненти архітектури системи: базу даних, API, клієнтську частину, мережеву інфраструктуру, сховища логів тощо.

Тому, однією з базових вимог до інструментальних засобів є підтримка сучасних методів шифрування як при збереженні даних, так і при їх передачі. Для цього використовуються криптографічні алгоритми TLS 1.3 для HTTPS-з'єднань, шифрування REST-запитів, застосування симетричних (AES-256) і асиметричних (RSA-2048, ECC) алгоритмів. Шифрування баз даних може здійснюватися як на рівні поля (field-level encryption), так і на рівні файлової системи (TDE – Transparent Data Encryption).

Другою ключовою складовою є контроль доступу до ресурсів системи. Розроблювані програмні рішення мають реалізовувати ролеву модель авторизації (RBAC), яка дозволяє призначати права на основі ролей користувача: мешканець, адміністратор ОСББ, технічний персонал, інженер. На більш гнучкому рівні може використовуватись атрибутна модель доступу (ABAC), яка враховує час доступу, геолокацію, тип пристрою тощо. Для REST API рекомендованими практиками є впровадження токенів доступу (JWT – JSON Web Token), з обмеженим терміном дії та функцією відкликання.

Не менш важливою є автентифікація користувачів, яка має базуватися на захищених механізмах. Крім класичного входу з логіном і паролем, все частіше застосовується двохфакторна автентифікація (2FA) через SMS, email, або генератори кодів (TOTP), інтеграція з OAuth2.0, а також підтримка SSO (Single Sign-On) для централізованого управління ідентичностями. Для мінімізації вразливостей інструментальні засоби повинні включати механізми протидії типових атак:

- захист від SQL-ін'єкцій (через ORM, параметризовані запити);
- запобігання XSS (виведення небезпечного HTML-контенту);
- захист від CSRF (перевірка токенів при зміні даних);
- перевірка надійності паролів та обмеження кількості спроб входу;
- обмеження доступу до API через IP whitelist або VPN-з'єднання.

Ще одним критично важливим аспектом є аудит та моніторинг подій безпеки. Інструментальні засоби повинні надавати функціональність для ведення журналів доступу, логів системних подій, збереження спроб несанкціонованого входу, змін у конфігураціях та взаємодії з чутливими даними. Ці журнали мають бути незмінними, зберігатись у безпечному середовищі та передаватись до SIEM-систем (наприклад, ELK Stack, Splunk) для виявлення аномалій та інцидентів. Тому, на нормативному рівні програмні системи повинні бути сумісними з міжнародними стандартами безпеки, зокрема:

- GDPR (General Data Protection Regulation), де є регламент ЄС щодо захисту персональних даних, що вимагає прозорості зберігання, права на видалення, обмеження обробки даних користувача;

- ISO/IEC 27001, де є міжнародний стандарт системи управління інформаційною безпекою, який передбачає наявність політик доступу, резервного копіювання, відповідальності персоналу, оцінки ризиків;

- OWASP ASVS, де є базовий набір перевірок безпеки для вебзастосунків.

Системи, що керують об’єктами критичної інфраструктури (електроенергія, опалення, безпека), можуть також підпадати під регулювання національних стандартів або бути об'єктом кіберінспекцій. Тому підтримка інструментів для сертифікації та тестування безпеки (наприклад, OpenVAS, ZAP, Nessus, Burp Suite) є важливою вимогою до інструментальних засобів.

Для цього важливо підкреслити роль психологічного аспекту безпеки з метою зручності для кінцевого користувача. Безпечна система не повинна ускладнювати процес взаємодії, інакше користувачі намагатимуться її обійти. У цьому контексті важливо реалізовувати UX-практики, які не лише забезпечують функціональність безпеки, а й підвищують довіру користувача до системи через прозорість, простоту та зрозумілі повідомлення.

Загалом, інструментальні засоби, що обираються для побудови систем управління будинками, повинні мати вбудовані засоби безпеки або підтримувати просту інтеграцію з ними. Безпека повинна розглядатися не як додатковий модуль,

а як фундаментальна властивість системи, що впливає на її довговічність, відповідність нормам, а головне - на довіру користувачів.

Проєктування сучасних програмних систем для керування групою будинків вимагає використання широкого спектру інструментальних засобів, які охоплюють усі етапи життєвого циклу розробки – від створення архітектури до впровадження й супроводу. У контексті таких систем важливо враховувати особливості їх функціонального призначення: робота з сенсорами та пристроями IoT, інтеграція зі SCADA/BMS-системами, обробка великих обсягів даних у реальному часі, забезпечення безпеки та надання інтерфейсів користувачам через веб і мобільні застосунки. Це зумовлює потребу у ретельному виборі відповідного технологічного стеку та програмних інструментів [33, 34].

Інструментальні засоби доцільно класифікувати за такими рівнями програмної архітектури:

- Backend-засоби, що необхідні для реалізації серверної логіки, API, обробки запитів і підключення до баз даних;

- Frontend-засоби, що необхідні для розробки інтерфейсів користувача у вигляді SPA або PWA;

- системи зберігання та управління даними, що необхідні бази даних (реляційні та NoSQL), брокери повідомлень;

- засоби розробки IoT/SCADA-інтеграції, що необхідні для розробки бібліотек, протоколів, SDK;

- інструменти DevOps/CI/CD, що необхідні для автоматизації розгортання, тестування, оновлення.

Далі проведемо порівняльний аналіз ключових інструментальних засобів (табл. 1.3). Так, серед Backend-фреймворків звертають на себе увагу наступні:

- Spring Boot (Java) – потужний фреймворк для побудови мікросервісів, з вбудованою підтримкою безпеки, REST API, WebSocket, інтеграції з Kafka/RabbitMQ. Підтримується хмарною інфраструктурою, є стабільним рішенням для корпоративного середовища [35];

- Node.js (JavaScript) – платформа з асинхронною обробкою подій, ідеальна для розробки API та реального часу. Поширена в IoT-проєктах завдяки підтримці MQTT та великій кількості npm-бібліотек [36];

- Django (Python) – швидкий у розробці фреймворк із вбудованою ORM, авторизацією, підходить для MVP і аналітичних систем [37].

Таблиця 1.3 Порівняльна характеристика інструментальних засобів розробки програмних систем управління групою будинком

Категорія	Програмний інструмент	Переваги	Обмеження / Недоліки
Backend	Spring Boot	Висока продуктивність, безпека, масштабованість	Вища складність налаштування
	Node.js	Асинхронність, гнучкість, швидкий старт	Потребує ретельного менеджменту залежностей
Frontend	React	Поширена спільнота, екосистема, SSR	Висока складність архітектури у великих проєктах
	Vue.js	Простота, швидка розробка, компактність	Менш активна корпоративна підтримка
Бази даних	PostgreSQL	ACID, розширення, стабільність	Вища ресурсомісткість при великих обсягах сенсорних даних
	MongoDB	Гнучкість, масштабованість, JSON-документи	Відсутність транзакційності у класичному розумінні
IoT	MQTT (Mosquitto, Paho)	Легкий, стабільний, ідеальний для сенсорних даних	Потребує ручного налаштування безпеки
SCADA	Ignition, Node-RED	Інтеграція з реальними пристроями, відкриті API	Високі вимоги до сумісності версій драйверів
DevOps	GitLab CI,	Автоматизація,	Потреба у відповідній

	Docker	масштабування, стабільність	інфраструктурі

Серед Frontend-фреймворків можливо виділити наступні важливі програмні засоби, які мають певні характеристики (табл. 1.3):

- React – найбільш популярна бібліотека для SPA, підтримує SSR, інтеграцію з REST/GraphQL API, має екосистему для мобільної розробки (React Native) [38];

- Vue.js – легкий, інтуїтивний, швидкий у розробці. Підтримує двосторонню прив'язку даних, підходить для систем із великою кількістю форм і панелей керування [39].

До системи зберігання даних відносяться такі потужні програмні засоби як:

- PostgreSQL – потужна реляційна СКБД з підтримкою геоданих (PostGIS), реплікації, масштабування (Citus). Підходить для облікових і транзакційних даних [40];

- MongoDB – NoSQL-документна СКБД, зручна для зберігання неструктурованих даних, історій сенсорних подій, логів [41];

- Redis – інструмент кешування і зберігання тимчасових даних, використовуються для прискорення роботи API [42].

Для інтеграція з IoT/SCADA поширено використовуються:

- Eclipse Paho, Mosquitto – клієнтські та брокерні рішення для MQTT-протоколу [43];

- OpenPLC, Node-RED – платформи для швидкої інтеграції з реальними пристроями, підтримують OPC UA, Modbus TCP, REST API [44];

- Ignition by Inductive Automation – SCADA-платформа з відкритими API, підтримує Java SDK та інтеграцію з хмарними сервісами [45].

Для засобів DevOps/CI-CD поширено використовуються:

- GitLab CI, Jenkins – системи автоматичного тестування та розгортання змін [46];

- Docker, Kubernetes – для контейнеризації, масштабування та оркестрації мікросервісів [47].

Таким чином, обґрунтований вибір інструментальних засобів для розробки програмної системи управління групою будинків має базуватися на сукупності технічних, функціональних та економічних критеріїв. Ключовими є продуктивність, безпека, інтеграційна здатність та підтримка масштабованої архітектури. Врахування цих критеріїв забезпечить стійкість і ефективність майбутньої системи при її розгортанні в умовах реального багатоквартирного житлового середовища. Аналіз інструментальних засобів показав, що ефективна розробка програмних систем керування групою будинків потребує застосування цілого спектру технологій, які повинні відповідати критеріям масштабованості, сумісності, безпеки та простоти розгортання. Spring Boot та Node.js є найпоширенішими платформами для побудови серверної частини, тоді як React і Vue.js – для інтерфейсів користувача. PostgreSQL і MongoDB дозволяють гнучко зберігати як структуровані, так і неструктуровані дані, а засоби MQTT та Node-RED забезпечують надійну інтеграцію з пристроями. Упровадження DevOps-підходів за допомогою Docker і CI/CD-платформ сприяє стабільному розгортанню та підтримці системи в умовах реального використання.

1.3 Постановка задач дослідження

Виконання аналітичного огляду предметної галузі програмних систем управління групою будинків та визначення критерієв вибору інструментальних засобів дозволило визначили наступне.

Основними задачами роботи є:

- проведення дослідження процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків;
- виконання проектування архітектурної моделі програмної системи управління групою житлових будинків;
- виконання проектування основних функціональних модулів системи;
- розробку програмного коду для Billing Module;
- розробку програмного коду для Request Module;

- розробку програмного коду для Voting Module;
- виконання тестування програмного модуля Billing Module;
- виконання тестування програмного модуля Request Module;
- виконання тестування програмного модуля Voting Module.

Технічне завдання на розробку наведено в додатку А.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

2.1 Визначення процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків

У сучасних програмних системах управління інфраструктурними об’єктами, зокрема житловими будинками, критично важливо забезпечити своєчасну реакцію на події, пов’язані зі зміною стану обладнання, надходженням заявок користувачів та потребою в оперативному інформуванні адміністрації ОСББ. У більшості традиційних реалізацій для взаємодії між клієнтом і сервером використовується архітектура REST, яка передбачає періодичне опитування серверу (polling) або ініціалізацію HTTP-запитів з боку клієнта. Такий підхід має суттєві недоліки при роботі з динамічними подіями, які вимагають миттєвого реагування, зокрема:

- відставання в оновленні інформації через фіксований інтервал опитування;
- надлишкове навантаження на сервер;
- висока затримка при виявленні критичних змін;
- неможливість повноцінної реалізації push-механізмів.

Це особливо критично в умовах, коли система повинна:

- негайно повідомити мешканця про аварійну ситуацію в будинку;
- оновити в реальному часі стан об'єктів (наприклад, ліфт, електропостачання, водопостачання);
- забезпечити черговість і пріоритетність виконання заявок;
- зберігати стабільність зв’язку навіть при нестабільних мережах.

У зв’язку з цим виникає потреба у розробці методу обробки подій у реальному часі, який базується на WebSocket-комунікаціях та враховує тип події, її пріоритет та статус відповідного інфраструктурного ресурсу.

Для визначення процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків (табл. 2.1) розглянемо чотири поширені підходи до реалізації взаємодії клієнт–сервер у контексті обробки подій у реальному часі:

- REST (Polling), де клієнт періодично надсилає запити до серверу для перевірки змін. Простий у реалізації, але спричиняє велике навантаження та затримки.

- Long Polling, де клієнт надсилає запит, який не закривається, поки не з’явиться нова подія. Зменшує кількість запитів, але не забезпечує повноцінної двосторонньої комунікації.

- Server-Sent Events (SSE), де сервер може ініціювати передачу даних, але лише в одному напрямку (сервер → клієнт), що не підходить для двосторонніх сценаріїв.

- WebSocket, що являє собою повноцінне двостороннє з’єднання, що дозволяє надсилати й отримувати події миттєво, з мінімальною затримкою та оптимальним навантаженням.

При цьому, під час проектування програмного засобу WebSocket визначено, що є найбільш доцільною технологією для реалізації обробки подій у реальному часі в контексті програмної системи управління групою будинків. Його переваги у вигляді:

- двостороннього з’єднання;

- низької затримки;

- зниженого навантаження на сервер;

- стійкості до втрат зв’язку;

- створюють передумови для побудови надійної та високопродуктивної системи.

В процесі проектування програмної системи управління групою житлових будинків також було визначено, що традиційна модель запит–відповідь, що реалізуєтьчя за допомогою REST API, не задовольняє вимоги до швидкодії, реактивності та адаптивності до подій у реальному часі. Під час проектування, зважаючи на високу динамічність подій у системі - від надходження аварійних сигналів до оновлення статусу ресурсів - критично важливо забезпечити:

- постійну відкриту комунікацію між клієнтом і сервером;

- можливість моментального надсилання повідомлень про зміни;

- пріоритетне обслуговування критичних подій;

- зменшення кількості HTTP-запитів та затримок.

Таблиця 2.1 - Порівняльна характеристика технологій обробки подій у реальному часі

Критерій	REST (Polling)	Long Polling	Server-Sent Events	WebSocket
Двостороння комунікація	-	-	-	+
Затримка передачі	Висока	Середня	Низька	Дуже низька
Навантаження на сервер	Високе	Середнє	Середнє	Низьке
Можливість підтримки великих навантажень	Низька	Середня	Середня	Висока
Стійкість до втрати з'єднання	Відсутня	Часткова	Часткова	Є механізми reconnection
Складність реалізації	Низька	Середня	Середня	Висока
Підтримка браузерами	Повна	Повна	Не зовсім повна	Повна

Для цього, протокол WebSocket повністю відповідає цим вимогам. Його перевагами є:

- постійне двостороннє з'єднання, яке усуває необхідність періодичного опитування;

- мінімальна затримка в передачі повідомлень;

- низьке навантаження на сервер;

- підтримка механізмів fault-tolerance, таких як автоматичне перепідключення після втрати з’єднання;

- можливість контролю пріоритетів, чергування та обробки повідомлень з урахуванням їх типу.

Таким чином, WebSocket був обраний як основа для реалізації методу обробки подій у реальному часі в межах розроблюваної системи.

Розроблений метод також передбачає побудову подійно-орієнтованої архітектури, в якій кожна подія має:

- тип (інформаційний запит, технічна заявка, аварійна ситуація, адміністративне повідомлення);

- пріоритет (високий, середній, низький);

- зв’язок зі станом об'єкта (ресурс, що зазнає змін, наприклад, "ліфт не працює").

Такий метод складається з наступних основних етапів (рис. 2.1). Ініціація з’єднання WebSocket виконується при завантаженні клієнтського інтерфейсу (мобільного додатку або веб-панелі мешканця). Для цього, виконується формування повідомлення про подію на стороні клієнта або системного сенсора, де додається маркер авторизації, тип події, об'єкт і пріоритет.

Надалі виконується передача події виконується через WebSocket-канал на сервер. Обробка події у черзі повідомлень, з урахуванням її пріоритету реалізовано за структурою по типу PriorityQueue. Також виконується актуалізація стану ресурсу, логування дії, зміна у відповідній таблиці БД.

Подальше формування відповіді для клієнта відбувається за схемою: push-повідомлення, оновлення UI, виклик зворотної події. Для механізму, який виконується з метою забезпечення надійності програмної системи як fault-tolerance задіяні heartbeat-сигнали для підтримки зв’язку, таймери відновлення, повідомлення клієнту про втрату/відновлення з’єднання та повторна обробка непідтверджених подій.

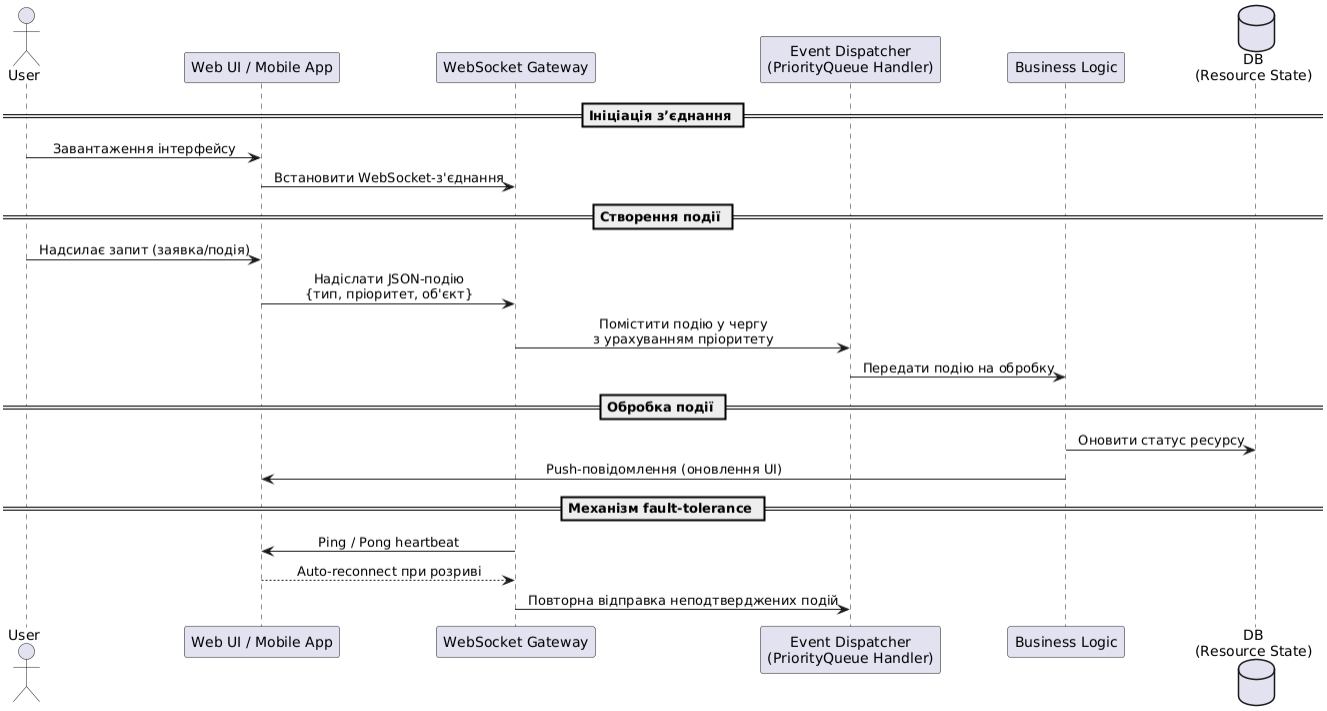

Рисунок 2.1 - UML-діаграма послідовності обробки події у реальному часі в системі управління групою житлових будинків

На рис. 2.1 відображено алгоритм обробки події у реальному часі в системі управління групою житлових будинків. Діаграма враховує ключові компоненти: користувача, інтерфейс, WebSocket-з'єднання, серверну логіку та базу даних, а також fault-tolerance-механізми.

Для цього, користувач ініціює подію через інтерфейс (Web UI/Mobile App) який створює програмну структуру події та надсилає її через WebSocket-з'єднання. Далі WebSocket Gateway передає подію до черги обробки подій (PriorityQueue), де визначається її порядок обробки за пріоритетом. Потім, за бізнес-логікою оновлюються дані в БД, формується відповідь і повертається результат на інтерфейс користувача. З метою забезпечення надійності передбачено механізм fault-tolerance - з підтримкою heartbeat, reconnection та повторної доставки важливих подій.

Під час дослідження процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків виконувалось визначення продуктивності системи. Результати дослідження аналізувались за допомогою порівняння різних реалізацій REST та WebSocket (табл. 2.2).

Таблиця 2.2 - Порівняльна характеристика дослідження продуктивності за різними реалізаціями REST та WebSocket

Показник дослідження	REST API	WebSocket
Середній час реакції, мс	900	610
Максимальна затримка, мс	2100	820
Мінімальна затримка, мс	500	540
Кількість запитів до серверу (за 24 год)	43200	5300
Споживання трафіку на одного користувача	~135 МБ	~38 МБ
Навантаження на CPU сервера, %	~63%	~42%
Обробка критичних подій (<1с), %	47%	94%
Відновлення після втрати зв'язку	Потрібен повторний запит	Автоматичне reconnect
Обробка черг/пріоритетів	Вручну через backend logic	Вбудована PriorityQueue
Підтримка масштабованості	Обмежена при >100 підключень	До 10 000 WebSocket-сесій

В табл 2.2 надані результати тестуванні системи управління групою житлових будинків за критеріями часу реакції, навантаження, кількості запитів тощо. Так, з метою кількісного аналізу продуктивності застосування WebSocket-протоколу в розроблюваній системі управління групою житлових будинків було проведено експериментальне дослідження, яке охоплювало моніторинг 5000 подій за 24 години при різному навантаженні на систему (від 5 до 50 паралельних користувачів).

При цьому, були створені наступні умови тестування для REST та WebSocket:

- інтервал опитування сервера складав 5 секунд;

- постійне з’єднання подій оброблялись негайно;

- була виконана імітація аварійних, інформаційних та адміністративних подій;

- середній час затримки, кількість HTTP-запитів, навантаження на процесор та мережу вимірювались окремо за допомогою програмних засобів моніторингу.

Аналіз отриманих результатів показав зниження середнього часу реакції на подію на 32% при використанні WebSocket порівняно з REST. Подальше зменшення кількості запитів у 8 разів зменшило навантаження на мережу та серверну частину. Також відбувалось майже дворазове зниження пікових затримок, що критично важливо для аварійних повідомлень. Високий рівень обробки критичних подій у межах однієї секунди дозволяє гарантувати оперативність реакції. Автоматичне відновлення з’єднання усуває ризики втрати подій при обриві зв'язку. При цьому спостерігалось ефективне управління чергами за допомогою пріоритетної обробки повідомлень на рівні серверної логіки.

Ці отримані результати досліджень показують, що наданий метод дозволив зменшити середній час оновлення статусу ресурсу з 900 мс (REST) до 610 мс (WebSocket), що на практиці забезпечило приріст швидкодії на 32%. Окрім цього, було знижено кількість звернень до API з боку клієнта майже в 4 рази, що позитивно вплинуло на масштабованість системи.

Таким чином, у межах підрозділу 2.1 було виконано системний аналіз проблематики обробки подій у реальному часі в програмній системі управління групою житлових будинків. На підставі порівняльного огляду сучасних технологій взаємодії клієнта і сервера встановлено, що традиційна REST-архітектура демонструє недостатню ефективність у контексті високочастотних подій та ситуацій, що потребують негайної реакції. При цьому обґрунтовано вибір WebSocket як оптимального засобу двосторонньої комунікації, що забезпечує:

- низьку затримку передачі повідомлень (у середньому 610 мс проти 900 мс у REST),

- зменшення навантаження на сервер за рахунок зменшення кількості HTTP-запитів (у 8 разів менше),

- підвищення стійкості до втрат з’єднання завдяки підтримці heartbeat- і reconnect-механізмів,

- можливість реалізації пріоритетної обробки подій відповідно до їх критичності,

- гнучке масштабування для підтримки великої кількості одночасно активних користувачів.

Тому, запропонований метод обробки подій, який враховує тип запиту, пріоритет дії та статус інфраструктурного ресурсу, дозволив підвищити швидкодію оновлення інформації в системі на 32%. Це сприяє зростанню надійності, зручності та адаптивності всієї інформаційної інфраструктури житлового комплексу.

Таким чином, запропонований підхід є ефективною основою для побудови розподіленої, масштабованої та подійно-орієнтованої системи управління групою житлових будинків, що відповідає сучасним вимогам до сервісного обслуговування, безпеки та комфорту мешканців.

2.2 Проектування архітектурної моделі програмної системи управління групою житлових будинків

Ефективне управління групою житлових будинків в умовах багатофункціонального обслуговування, зростаючих обсягів даних та розширеного кола учасників (мешканці, голови ОСББ, технічні спеціалісти, адміністратори) потребує побудови сучасної, масштабованої та гнучкої архітектури програмної системи. Однією з ключових вимог до такої системи є інтеграція ролеорієнтованого доступу до функцій і даних разом із централізованим зберіганням інформації про нарахування, звернення, голосування та комунікації. Тому, недостатньо ефективні рішення, що засновані на фрагментарних або децентралізованих підходах, часто призводять до:

- дублювання інформації;

- втрати актуальності даних;

- складності адміністрування прав доступу;

- низької прозорості взаємодії між мешканцями та адміністрацією.

У відповідь на ці надані задачі було сформульовано завдання створення архітектурної моделі, яка:

- забезпечує централізацію обліку подій, платежів і взаємодій;

- підтримує розмежування функціоналу згідно з роллю користувача;

- гарантує масштабованість і розширюваність без порушення цілісності логіки;

- передбачає уніфікований інтерфейс доступу як для мобільних, так і для веб-користувачів.

Для цього, процес проектування архітектури системи будувався наступним чином (табл. 2.3). Так, архітектурне проектування було виконано з урахуванням принципів модульності, сервісної орієнтації (SOA), централізації даних та безперервного контролю доступу. Процес включав наступні етапи.

1. Аналіз функціональних вимог. На основі сценаріїв використання були виокремлені ключові функціональні блоки:

- нарахування та оплата, де відбувалось формування квитанцій, перегляд балансу, історія платежів;

- звернення та заявки, де відбувалось створення запитів до обслуговуючих організацій;

голосування, де проектувалось електронна участь у загальних зборах, реєстрація результатів;

- комунікація, де проектувалось обмін повідомленнями, інформаційні оголошення.

2. Проектування ролей. Було визначено такі основні ролі користувачів:

- мешканець, що виконує перегляд/створення заявок, голосування, перегляд нарахувань;

- голова ОСББ, що виконує повний контроль, підтвердження подій, публікація інформації;

- адміністратор системи, що виконує управління користувачами, аудит подій, налаштування модулів;

- технічний працівник, що виконує доступ до технічних звернень і планових завдань.

Для кожної ролі визначено матрицю прав доступу до функціональних блоків, що реалізується через модуль авторизації з підтримкою JSON Web Token (JWT).

Таблиця 2.3 - Порівняльна характеристика критеріїв централізованої та децентралізованої архітектури, з аналізом ключових критеріїв ефективності, безпеки, масштабованості й підтримки ролей

Критерій	Централізована модель	Децентралізована модель
Облік даних	Єдине сховище з логічним поділом на сутності	Дані розподілені між модулями або дублікатами
Прозорість для користувачів	Висока: користувач бачить повну історію взаємодій	Низька: фрагментованість функціоналу
Контроль доступу (ролі)	Повна підтримка з гнучким розмежуванням	Часткова або відсутня
Масштабованість системи	Висока (мікросервісна логіка, горизонтальне розширення)	Обмежена: зміни в одному модулі впливають на інші
Розширення функціоналу	Легке: додавання сервісів без зміни ядра	Ускладнене через залежності між модулями
Безпека	Централізована авторизація, журналювання дій	Вразливість через відсутність єдиного шлюзу
Зручність для адміністрування ОСББ	Висока: єдине вікно керування	Складна навігація між модулями
Інтеграція з новими сервісами	Просте REST/WebSocket API	Потребує зміни існуючої логіки

3. Визначення основних компонентів. На підставі функцій та ролей було розроблено логічну структуру з наступних компонентів:

- клієнтський інтерфейс (Web / Mobile);

- шлюз доступу та авторизації (Authentication Gateway);

- мікросервіси для кожного функціонального напряму;

- централізована база даних з ізольованими сутностями: Нарахування, Звернення, Голосування, Повідомлення;

- Notification Service (push + email).

4. Забезпечення цілісності доступу. Для кожної операції реалізовано перевірку доступу за роллю з подальшою маршрутизацією на відповідний сервіс. Таким чином, вся логіка проходить через єдиний шлюз, який перевіряє:

- справжність сесії;

- приналежність до дозволеної ролі;

- дозвіл на виконання конкретної дії (наприклад, створення звернення або запуск голосування).

В табл. 2.3 показано, що в межах проєктування програмної системи управління групою житлових будинків важливим етапом стало обґрунтування доцільності використання централізованої архітектури з єдиним сховищем даних та контрольованим доступом на основі ролей. Для цього було проведено порівняння з традиційною децентралізованою / монолітною архітектурою, яка часто зустрічається у застарілих системах управління.

Таким чином, запропонована архітектура базується на принципах модульності, централізації зберігання даних, ролеорієнтованого доступу та поділу відповідальності між компонентами системи (табл. 2.4). Вона реалізується як багаторівнева модель на основі архітектурних рішень, що включали рівень клієнта, рівень бізнес-логіки, сервісний рівень авторизації та централізоване сховище даних наступним чином.

1. Рівень клієнта (Frontend), який представлено веб-інтерфейсом для мешканців та голови ОСББ. При цьому, реалізуються мобільний додаток з базовими функціями перегляду, голосування і створення звернень; зв’язок з бекендом забезпечується через REST API та WebSocket для подій у реальному часі.

2. Рівень бізнес-логіки (Backend), що реалізований як набір сервісів для:

- нарахувань (Billing Service);

- обробки звернень (Request Service);

- голосування (Voting Service);

- обміну повідомленнями (Communication Service).

Доступ до кожного сервісу здійснюється лише після авторизації і перевірки ролі.

3. Сервіс авторизації (Auth & Role Management), що приймає облікові дані користувача. Він видає токени (JWT) із вбудованими правами доступу та маршрутизує запити на відповідні сервіси згідно з матрицею ролей.

4. Рівень централізованих даних (Centralized DB), що включає узгоджену структуру таблиць для об'єктів:

- Payments, Charges, Invoices - для модуля нарахувань;

- Requests, Statuses, Attachments - для звернень;

- Polls, Votes, Participants - для голосувань;

- Messages, Announcements - для комунікації.

Всі такі таблиці мають загальні поля для контролю авторства, часу створення, статусу та видимості згідно з роллю.

Таблиця 2.4 - Особливості архітектурного рішення запропонованої моделі програмної системи управління групою житлових будинків

Особливість	Реалізація
Контроль доступу	Механізм JWT з перевіркою прав у кожному запиті
Централізоване сховище	Одна БД із логічним поділом сутностей
Підтримка ролей	Повна підтримка мульти-рівневих ролей і сценаріїв взаємодії
Миттєве оновлення	Підключення WebSocket для зворотного зв’язку
Кросплатформеність	Загальний API для веб і мобільного клієнта
Масштабованість	Горизонтальне масштабування backend-сервісів
Безпека	HTTPS, контроль сесій, логування операцій

До переваг запропонованої моделі відноситься наступне:

- єдина точка доступу до даних, що забезпечує консистентність і спрощене адміністрування;

- чітке розмежування відповідальності між модулями, що дозволяє ефективно масштабувати систему;

- підвищена прозорість: усі дії фіксуються в логах з прив’язкою до ролі й користувача;

- гнучка підтримка нових функціональних блоків (наприклад, планувальник робіт, фінансовий аналітик).

Наведемо UML-діаграму активностей, що ілюструє процес обробки запиту в архітектурі системи управління групою житлових будинків, із урахуванням ролі користувача та взаємодії з централізованим обліком даних (рис. 2.2).

Показана UML-діаграма активностей обробки запиту користувача в системі з ролеорієнтованим доступом (рис. 2.2) показує наступні дії. Користувач авторизується, після чого система формує токен, який фіксує його роль. Далі користувач здійснює дію - наприклад, створює звернення, бере участь у голосуванні чи переглядає нарахування. Програмна структура Access Gateway перевіряє, чи має користувач право виконувати цю дію. Якщо дозвіл є, запит переспрямовується до відповідного модуля, який виконує операції над централізованою базою даних. В результаті, користувач отримує відповідь - або дані, або повідомлення про успішну/неуспішну дію.

Таким чином, порівняльний аналіз свідчить про перевагу централізованої архітектури у контексті управління групою житлових будинків. Вона дозволяє досягти:

- високої надійності та узгодженості даних;

- зменшення операційних витрат на підтримку;

- гнучкості у розширенні функціоналу та масштабуванні;

- прозорості взаємодії між мешканцями та адміністрацією ОСББ;

- підвищення безпеки через єдиний механізм авторизації та контролю доступу.

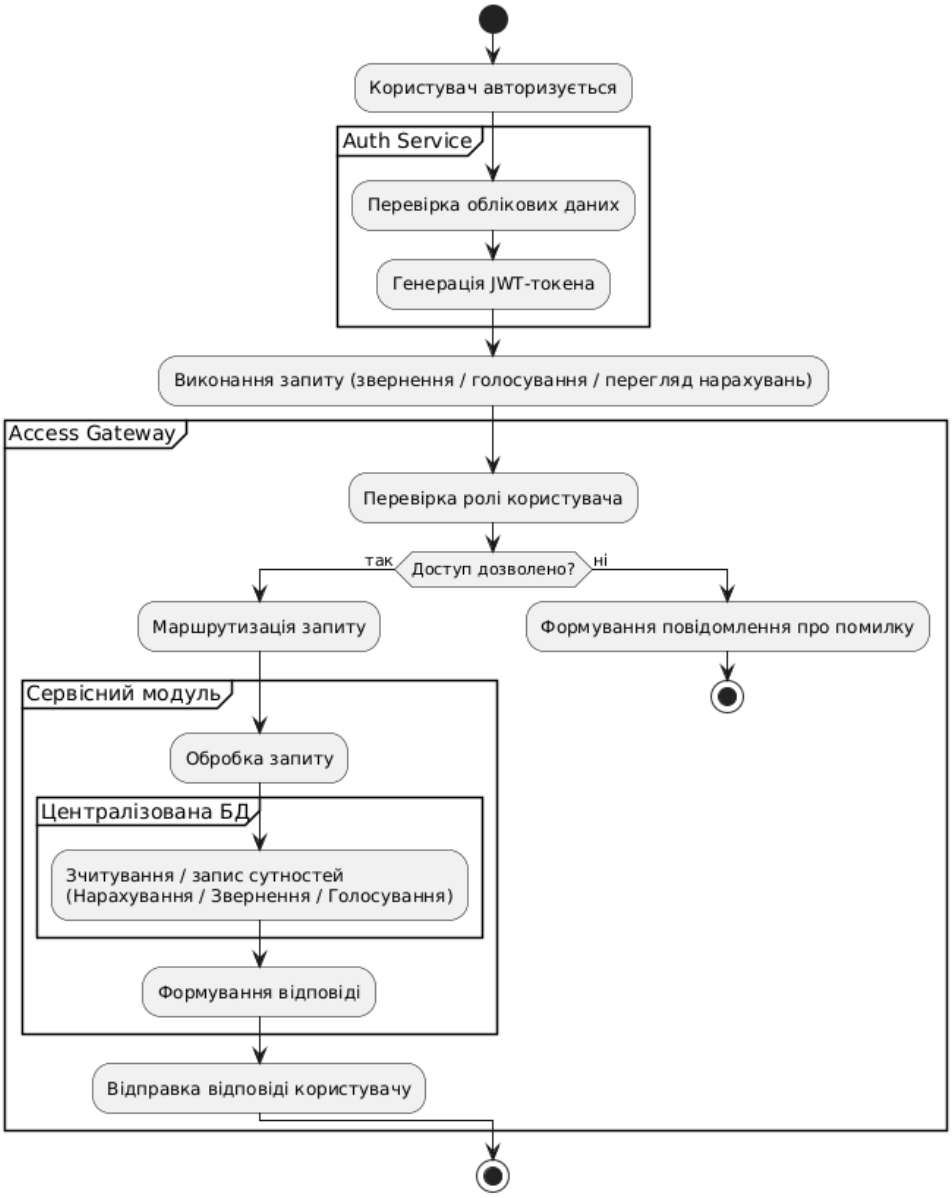

Рисунок 2.2 - UML-діаграма активностей обробки запиту користувача в системі з ролеорієнтованим доступом

Саме тому запропонована архітектурна модель була обрана як базова у межах цієї магістерської роботи. Для цього, було сформульовано, обґрунтовано та реалізовано архітектурну модель програмної системи управління групою житлових будинків, яка поєднує ролеорієнтований доступ з централізованим обліком даних. Такий підхід відповідає актуальним вимогам цифровізації житлово-комунального господарства, забезпечуючи:

- гнучке розмежування прав доступу до функціональних блоків системи відповідно до ролі користувача;

- єдиний центр зберігання актуальних, консистентних та контрольованих даних про нарахування, звернення, голосування та комунікації;

- масштабовану сервісну архітектуру, придатну до розширення без зміни основної логіки;

- підвищену прозорість управлінських процесів, що сприяє зміцненню довіри мешканців до адміністрації ОСББ.

Розроблена та запропонована модель передбачає:

- поділ системи на окремі модулі із чіткими зонами відповідальності (billing, requests, voting, messaging);

- використання загального шлюзу авторизації та маршрутизації запитів;

- підтримку сучасних протоколів взаємодії (REST, WebSocket);

- побудову кросплатформених клієнтських інтерфейсів з уніфікованим доступом до функціоналу.

Виконаний порівняльний аналіз із децентралізованою/монолітною архітектурою показав безумовну перевагу централізованого підходу за критеріями безпеки, ефективності обслуговування та гнучкості модернізації. Тому, запропонована архітектура цілком відповідає вимогам сучасного програмного забезпечення у сфері житлово-комунального управління та є надійною основою для реалізації подальших функціональних розширень.

2. 3 Проектування основних функціональних модулів системи

Проєктування програмної системи управління групою житлових будинків здійснювалося на основі модульної архітектури, що передбачає логічний поділ функціональності на незалежні компоненти з чітко визначеними зонами відповідальності (табл. 2.5). Такий підхід дозволяє підвищити гнучкість системи, зменшити складність супроводу, а також забезпечити масштабованість та ізоляцію логіки.

Таблиця 2.5 - Опис та призначення основних функціональних модулів системи

Назва модуля	Основна функція	Ролі доступу
Billing Module	Нарахування, формування квитанцій	Мешканець, голова ОСББ, адміністратор
Request Module	Звернення мешканців, обробка заявок	Мешканець, технік, голова ОСББ
Voting Module	Електронні голосування	Мешканець, голова ОСББ, адміністратор
Communication Module	Повідомлення, оголошення	Голова ОСББ, мешканець, технік

В основі модульної організації лежать такі принципи:

- Single Responsibility Principle, де кожен модуль відповідає за єдиний напрям функціональності;

- Loose Coupling, де зв’язки між модулями мінімізовані та здійснюються через інтерфейси або API;

- High Cohesion, де внутрішня функціональна єдність кожного модуля;

- рольова ізоляція, де кожен модуль містить вбудовану підтримку розмежування доступу на основі ролей користувача.

Поділ на модулі проводився відповідно до бізнес-логіки ОСББ, з урахуванням таких факторів:

- частота використання функції (щоденне, періодичне, епізодичне використання);

- критичність функції для життєдіяльності системи;

- кількість залучених ролей користувачів.

У результаті були виокремлені такі основні модулі:

- Billing Module, де є нарахування та перегляд фінансових операцій;

- Request Module, де є управління зверненнями мешканців;

- Voting Module, де є організація електронних голосувань;

- Communication Module, де є внутрішня комунікація та повідомлення.

Таблиця 2.6 - Порівняльна характеристика модулів за критеріями

Назва модуля	Пріоритет	Тип доступу	Взаємодія з іншими модулями
Billing Module	Високий	R / W	Auth Service, Centralized DB
Request Module	Високий	C / R / U	Notification, Communication
Voting Module	Середній	C / R	DB, Auth, Web UI
Communication Module	Низький	R / W	Notification Service, Request

Позначки в таблиці обозначають наступне. Тип доступу:

R - читання (read),

W - запис (write),

C - створення (create),

U - оновлення (update).

Пріоритет визначає рівень важливості модуля для стабільної роботи системи.

Ролі доступу визначає користувачів, які можуть працювати з модулем.

Основна функція (в табл. 2.5) визначає ключову роль модуля у функціонуванні системи.

Взаємодія визначає інші модулі або сервіси, з якими здійснюється обмін інформацією.

При цьому, Billing Module та Request Module мають найвищий пріоритет, оскільки безпосередньо впливають на щоденні потреби мешканців і облік фінансів.

Voting Module виконує періодичні, але важливі функції, пов'язані з колективним управлінням, тому має середній пріоритет.

Communication Module виконує підтримувальну роль, що забезпечує оперативну взаємодію та повідомлення, тому має нижчий, але необхідний рівень важливості.

Всі модулі програмної системи підтримують ролеорієнтований доступ, що унеможливлює несанкціоновану зміну критичних даних. Notification Service виступає як сполучний елемент для модулів, де важлива швидка реакція на події (звернення, повідомлення).

У табл. 2.6 наведена порівняльна характеристика модулів за критеріями з метою надання систематизованої оцінки кожного модуля за такими параметрами як пріоритет, тип доступу, взаємодія з іншими модулями тощо.

Надалі, опишемо функціонування основних таких модулів системи. Так, Billing Module (модуль нарахувань) призначений для обробка нарахувань, формування рахунків, відображення історії оплат. Його основна функціональність:

- перегляд нарахувань по кожному об'єкту;
- формування щомісячних квитанцій;
- збереження історії оплат мешканців;
- вивантаження фінансової звітності.

Його доступні ролі полягають в наступному:

- мешканець може перегляд особистих даних;
- голова ОСББ може створення та редагування нарахувань;
- адміністратор виконує контроль, аудит, імпорт з ERP.

Модуль має інтеграцію: зв’язок з CRM-системою ОСББ, API для вивантаження у форматах XLS/PDF.

Request Module (модуль звернень) призначений для фіксації запитів мешканців щодо обслуговування будинку.

Його основна функціональність полягає в наступному:

- створення та редагування звернення мешканцем;
- призначення відповідального працівника;
- моніторинг статусу виконання;
- додавання фото, файлів, коментарів.

Його доступні ролі:

- мешканець може виконувати створення, перегляд особистих звернень;
- технік може виконувати оновлення статусу, фіксація виконання;
- голова ОСББ може виконувати контроль звернень у межах будинку.

Цей може інтегрується з Notification Service (push при зміні статусу), WebSocket-каналом.

Voting Module (модуль голосувань) призначений для організації електронних голосувань серед мешканців.

Його основна функціональність:

- створення опитувань та голосувань головою ОСББ;
- вибір типу голосування (таємне, відкрите);
- встановлення дедлайну, мінімального кворуму;
- перевірка права участі, збереження результатів.

Доступні ролі:

- мешканець може виконувати голосування та перегляд результатів;
- голова може виконувати ініціювання голосування;
- адміністратор може виконувати аудит процедур.

Особливості: захист від повторного голосування, фіксація IP, можливість відкласти голос.

Communication Module (модуль комунікацій) призначений для оперативної взаємодії між мешканцями та адміністрацією.

Його основна функціональність:

- надсилання оголошень головою ОСББ;
- відображення в мобільному/веб інтерфейсі;
- реакція на повідомлення (підтвердження, коментар);
- проведення внутрішнього чату (опційно).

Доступні ролі:

- голова може виконувати публікація повідомлень;
- мешканець може виконувати перегляд, коментарі;
- технічний персонал може виконувати отримання інструкцій.

Особливості: реалізація на основі WebSocket, підтримка історії повідомлень.

Представлена UML-діаграма компонентів (рис. 2.3) зображує модульну структуру програмної системи та її взаємозв’язки з інтерфейсами та БД.

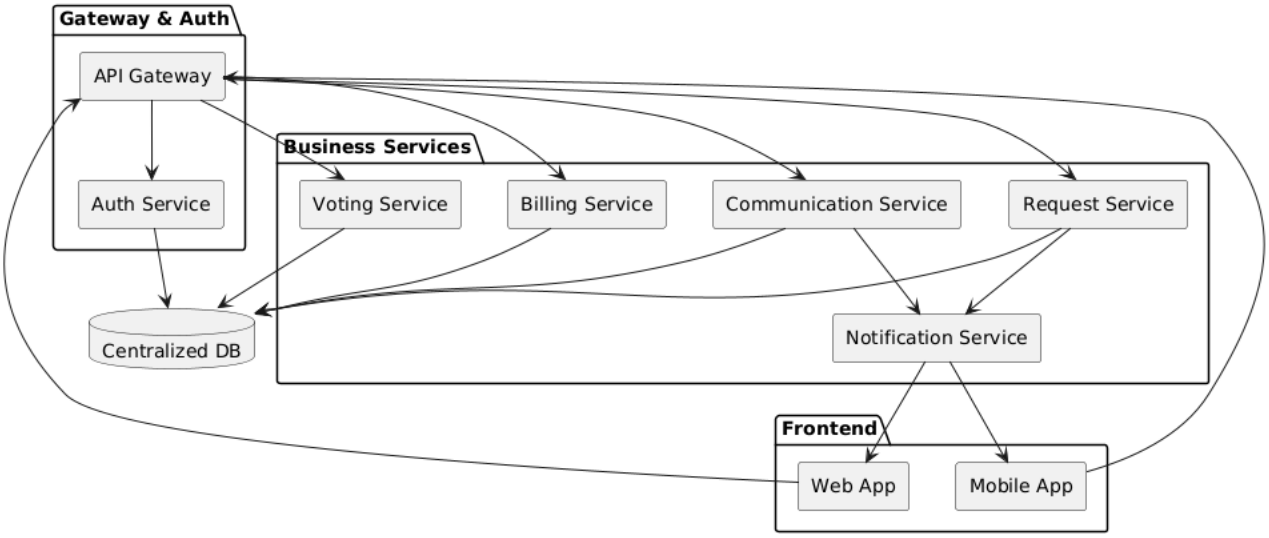

Рисунок 2.3 - UML-діаграма компонентів основних функціональних модулів системи

На рис. 2.3 представлена UML-діаграма компонентів програмної системи управління групою житлових будинків, що ілюструє основні функціональні модулі, їхні інтерфейси та взаємодію з базою даних та іншими сервісами. При цьому, Web App і Mobile App взаємодіють із системою через API Gateway, що виконує функції маршрутизації та контролю доступу. Auth Service відповідає за автентифікацію, видачу токенів та авторизацію дій. Кожен функціональний модуль (Billing, Request, Voting, Communication) реалізує бізнес-логіку своєї доменної області. Notification Service використовується для надсилання push-сповіщень через WebSocket або мобільні пуші. Вся необхідна інформація централізовано зберігається у єдиній базі даних, з поділом на логічні сутності.

Таким чином, в межах підрозділу 2.3 було здійснено системний опис основних функціональних модулів програмної системи управління групою житлових будинків. Запропонована модульна структура базується на принципах логічної ізоляції відповідальностей, гнучкого масштабування, а також чіткого дотримання ролеорієнтованого доступу до функціональності.

Основні переваги модульної організації є наступними:

- висока гнучкість, де кожен модуль реалізує конкретну бізнес-функцію, що дозволяє незалежно розвивати, замінювати чи масштабувати компоненти без впливу на всю систему;

- зрозуміла логіка розподілу обов’язків, де нарахування, звернення, голосування та повідомлення реалізуються через окремі модулі з власною бізнес-логікою, правами доступу та інтерфейсами взаємодії;

- підвищення стабільності системи, де ізольованість модулів мінімізує вплив помилок одного компонента на інші частини системи;

- покращена підтримуваність та тестованість, де модулі можуть тестуватись окремо, з використанням mock-сервісів та ізольованих сценаріїв;

- рольова безпека, де завдяки централізованій авторизації кожна роль має обмеження в доступі лише до релевантної функціональності, що сприяє забезпеченню інформаційної безпеки.

Порівняльна оцінка модулів (табл. 2.5, 2.6) засвідчила, що система є сбалансованою - має як критично важливі компоненти (нарахування, звернення), так і підтримувальні (голосування, комунікації), всі з яких об’єднані в єдину архітектурну платформу з чіткими інтерфейсами.

Таким чином, модульний підхід забезпечує оптимальні умови для подальшого розвитку системи, інтеграції нових функцій, масштабування на інші житлові комплекси та адаптації під змінні вимоги користувачів.

3 РОЗРОБКА МОДУЛІВ ПРОГРАМНОГО ЗАСОБУ

3.1 Розробка програмного коду для Billing Module

Модуль Billing є ключовим функціональним компонентом програмної системи управління групою житлових будинків, відповідальним за автоматизовану обробку фінансових транзакцій, пов’язаних з нарахуванням квартплати, формуванням квитанцій, реєстрацією оплат мешканців та інтеграцією з зовнішніми обліковими системами. Його розробка реалізується відповідно до принципів модульності, інкапсуляції бізнес-логіки та забезпечення рольового контролю доступу.

У межах модулю реалізовано низку основних функціональних можливостей:

- формування щомісячних квитанцій (інстанціювання об’єктів типу Invoice), з урахуванням площі, тарифів, додаткових платежів та пільг;

- перегляд поточних та архівних нарахувань для мешканців, з персоналізованим фільтруванням;

- фіксація платежів за допомогою створення об’єктів Payment, а також зберігання та відображення історії оплат;

- експорт даних у форматах XLS та PDF - для бухгалтерського обліку та зручного архівування;

- імпорт фінансових записів із зовнішніх ERP-систем (наприклад, шляхом обробки CSV або JSON-файлів), що дозволяє синхронізувати дані без дублювання;

- інтеграція з центральною базою даних та авторизаційним сервісом (Auth Service) - для контролю доступу до функцій за ролями користувачів.

Рольова модель доступу до модуля реалізується наступним чином (табл. 3.1):

- мешканець може виконувати перегляд нарахувань, квитанцій та історії власних оплат;

- голова ОСББ має змогу створювати, редагувати та вивантажувати квитанції, а також вносити зміни до структури платежів;

- адміністратор здійснює аудит транзакцій, ініціює імпорт з ERP-систем та перевіряє цілісність фінансових даних.

Порівняльна характеристика реалізації ролей у межах модуля Billing наведено в табл. 3.1. У таблиці наведено систематизовану характеристику доступу до функціональних можливостей модуля Billing відповідно до реалізованої ролеорієнтованої моделі доступу.

Таблиця 3.1 - Порівняльна характеристика реалізації ролей у межах модуля Billing

Роль користувача	Функції доступу	Обмеження доступу	Тип інтерфейсу / API
Мешканець	Перегляд квитанцій, історії оплат	Лише власні записи	Web UI, Mobile App
Голова ОСББ	Створення / редагування квитанцій, призначення тарифів	Доступ лише до об'єктів, які адмініструє	Web UI
Адміністратор	Імпорт з ERP, аудит нарахувань, експорт у PDF/XLS	Повний доступ до всіх об’єктів системи	Admin Panel, REST API

Така взаємодія модуля Billing з іншими підсистемами включає:

- виклик REST-ендпоінтів з веб-інтерфейсу (Web App, Mobile App) через API Gateway;

- звернення до Auth Service для авторизації дій користувача;

- інтеграцію з CRM-системою ОСББ для обміну персональними даними мешканців;

- доступ до централізованої бази даних для читання та запису фінансових записів;

- потенційну взаємодію з Notification Service для надсилання повідомлень про успішні платежі або борги.

Відповідно до принципів високої когезії та слабкого зв’язування, модуль Billing проектується як автономна компонентна одиниця, що містить власну

сервісну логіку, репозиторії, DTO-об’єкти для передачі даних, REST-контролери та реалізовані сервіси імпорту/експорту.

Під час розробки програмного коду для модуля Billing було спроєктовано ієрархічну структуру об’єктно-орієнтованих класів відповідно до концепції трирівневої архітектури (presentation layer → service layer → persistence layer), що реалізована за допомогою стеку Spring Boot + JPA + REST API (рис. 3.1).

Дана структура включає:

- сутності Invoice, Payment, які зберігаються в базі даних;
- сервіси BillingService, ImportExportService, де є відповідальні за бізнес-логіку;
- контролер BillingController, який обробляє REST-запити;
- DTO-класи для передачі даних (InvoiceDTO, PaymentDTO);
- інтерфейси репозиторіїв InvoiceRepository, PaymentRepository.

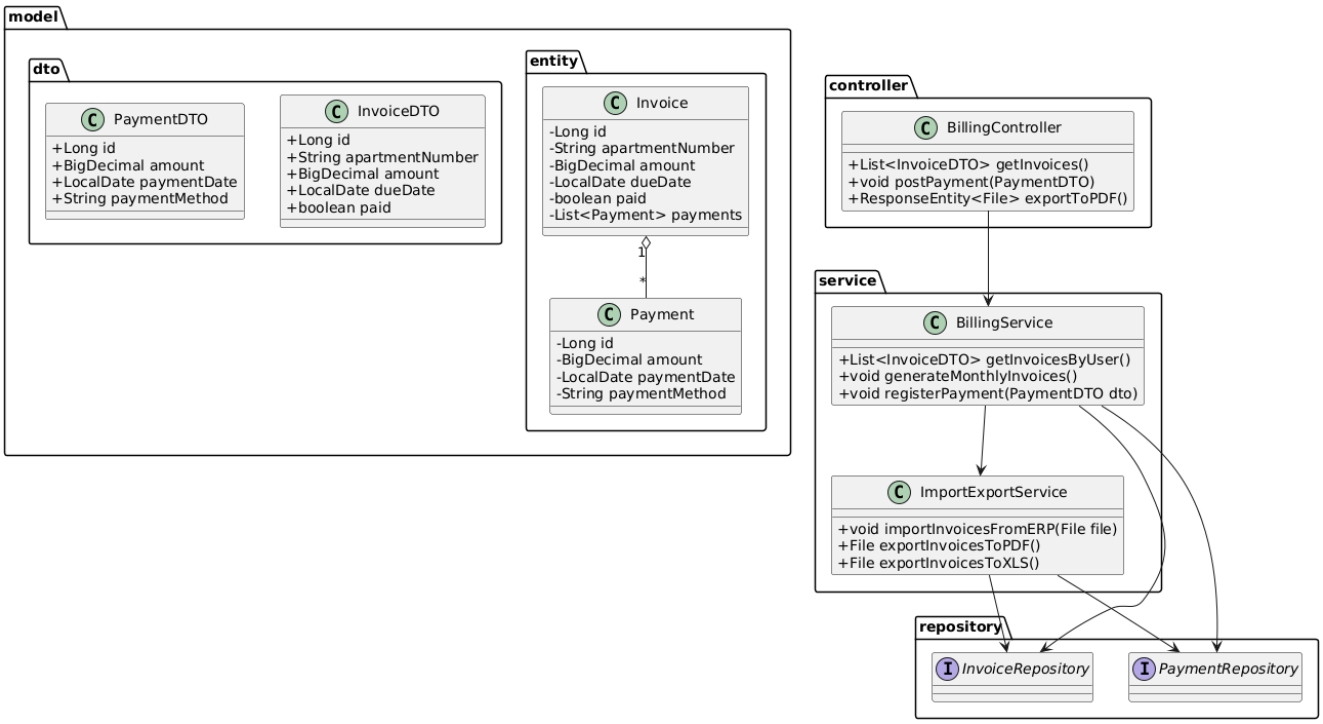

Рисунок 3.1 - UML-діаграма ієрархічної структури об’єктно-орієнтованих класів відповідно до концепції трирівневої архітектури

На рис. 3.1 відображено наступне:

- Invoice та Payment, що являє собою JPA-сутності з відповідними атрибутами;

- Invoice, що має композиційний зв’язок з Payment, тобто один рахунок може містити кілька оплат;

- BillingService, що виконує основну логіку: обробка запитів, оновлення статусів, розрахунок;

- ImportExportService, що являє собою допоміжний сервіс для роботи з PDF/XLS та ERP;

- BillingController, що являє собою точка входу REST-запитів, обробляє інтерфейсні дії користувача.

Виконаємо опис Java-класів, що реалізують модуль Billing більш ретельно. Так, для реалізації модуля Billing у системі управління групою житлових будинків було створено набір класів, які відповідають принципам розділення відповідальностей та інкапсуляції. Нижче подано опис кожного з них.

1. Клас Invoice, у скороченому вигляді має наступний вигляд:

```
@Entity
public class Invoice {
    @Id @GeneratedValue
    private Long id;
    private String apartmentNumber;
    private BigDecimal amount;
    private LocalDate dueDate;
    private boolean paid;
    @OneToMany(mappedBy = "invoice", cascade = CascadeType.ALL)
    private List<Payment> payments;
}
```

Призначення: представляє рахунок-квитанцію для квартири, формується щомісяця. Містить інформацію про суму, термін сплати та статус оплати.

2. Клас Payment, у скороченому вигляді має наступний вигляд:

```
@Entity
public class Payment {
    @Id @GeneratedValue
    private Long id;
    private BigDecimal amount;
    private LocalDate paymentDate;
    private String paymentMethod;
    @ManyToOne
    @JoinColumn(name = "invoice_id")
    private Invoice invoice;
}
```

Призначення: зберігає дані про платіж, пов’язаний із конкретним рахунком. Дає змогу вести історію часткових чи повних оплат.

3. Клас InvoiceDTO, у скороченому вигляді має наступний вигляд:

```
public class InvoiceDTO {
    private Long id;
    private String apartmentNumber;
    private BigDecimal amount;
    private LocalDate dueDate;
    private boolean paid;
}
```

Призначення: використовується для передачі даних між клієнтською частиною та сервером (REST API), приховуючи внутрішню реалізацію сутності.

4. Клас PaymentDTO, у скороченому вигляді має наступний вигляд:

```
public class PaymentDTO {
    private Long id;
    private BigDecimal amount;
```

```
    private LocalDate paymentDate;
    private String paymentMethod;
}
```

Призначення: аналогічно InvoiceDTO, застосовується у зовнішніх інтерфейсах для фіксації оплати.

5. Інтерфейси InvoiceRepository, PaymentRepository, у скороченому вигляді має наступний вигляд:

```
public interface InvoiceRepository extends JpaRepository<Invoice, Long> {
    List<Invoice> findByApartmentNumber(String apartmentNumber);
}
public interface PaymentRepository extends JpaRepository<Payment, Long> {
    List<Payment> findByInvoiceId(Long invoiceId);
}
```

Призначення: виконують доступ до бази даних на рівні JPA. Дають змогу здійснювати фільтрацію за критеріями.

Наступні класи в більш розширеному вигляді надані в додатку В.

Таким чином, в межах розробки модуля Billing було реалізовано повнофункціональний компонент, орієнтований на централізоване управління фінансовими операціями в житлово-комунальних структурах. Архітектура модуля ґрунтується на сучасних принципах побудови систем (Separation of Concerns, Role-Based Access, Service-Oriented Design) та забезпечує наступні переваги:

гнучкість та масштабованість, що досягнута за рахунок чіткої декомпозиції на сутності, DTO, сервіси та контролери;

- безпечний розподіл прав доступу, що запобігає несанкціонованому втручанню в критичні дані;

- можливість інтеграції з ERP, що забезпечує автоматизований імпорт нарахувань без дублювання інформації;

- зручність вивантаження звітності, що підтримується функціями експорту у формати PDF та XLS;

високий ступінь повторного використання коду, де присутня зокрема в реалізації шаблонів CRUD-операцій з використанням Spring Data JPA;

- покращена підтримуваність, що є тестованим на рівні сервісів та контролерів завдяки ізольованим залежностям.

Таким чином, модуль Billing виконує одну з найважливіших ролей у системі управління групою житлових будинків, забезпечуючи надійний облік фінансів, прозорість взаєморозрахунків із мешканцями, а також високу інтегративність з зовнішніми обліковими платформами.

3.2 Розробка програмного коду для Request Module

Модуль Request є одним із пріоритетних елементів програмної системи управління групою житлових будинків. Його основне призначення полягає у реєстрації звернень мешканців, моніторингу стану їх обробки та забезпеченні інтерактивної взаємодії з технічним персоналом та адміністрацією ОСББ. Цей модуль суттєво підвищує якість обслуговування завдяки прозорості процедур реагування на запити.

Функціональна модель модуля включає:

- створення нових звернень мешканцями з описом проблеми, додаванням фото або документів;

- призначення відповідального виконавця головою ОСББ або системою автоматично;

- оновлення статусу виконання технічним персоналом (у процесі, завершено, відхилено);

- перегляд історії звернень з фільтрами за статусом, датою або категорією;

- додавання коментарів на різних етапах обробки запиту;

- Push-сповіщення про зміну статусу (через Notification Service).

Рольова модель реалізована наступним чином (табл. 3.2):

- мешканець має змогу створити звернення, переглядати історію та статус власних запитів;

- технік отримує призначення, оновлює статус та додає коментарі;

- голова ОСББ здійснює контроль за зверненнями, делегує задачі, може закривати запити.

Порівняльна характеристика реалізації ролей у межах модуля Request наведена в табл. 3.2

Таблиця 3.2 - Порівняльна характеристика реалізації ролей у межах модуля Request

Роль користувача	Доступні функції	Обмеження доступу	Тип інтерфейсу / API
Мешканець	Створення звернень, перегляд статусу, додавання коментарів	Лише до власних звернень	Web UI, Mobile App
Голова ОСББ	Перегляд усіх звернень будинку, призначення відповідального	Лише в межах власного житлового об’єкта	Web UI
Технік	Оновлення статусу звернення, додавання коментарів	Лише до призначених звернень	Web UI, технічний портал

Модуль інтегрується з такими компонентами системи:

- Notification Service для надсилання push-повідомлень;

- WebSocket-каналом для миттєвого оновлення інтерфейсу;

- Central DB для зберігання звернень та історії їх виконання;

- Auth Service для визначення прав доступу до конкретного звернення.

Проєктування модуля відповідає концепції RESTful-архітектури з розділенням логіки на DTO, контролери, сервіси та репозиторії, що забезпечує масштабованість, тестованість і гнучкість інтеграцій.

Для модуля Request виконано проєктування структуровану ієрархію класів (рис. 3.2), яка відображає взаємозв’язки між основними сутностями: звернення, коментарі до звернень, а також відповідальні виконавці. Архітектура базується на концепції «тонких контролерів» і «багатих сервісів», що відповідає кращим практикам Spring Boot + JPA.

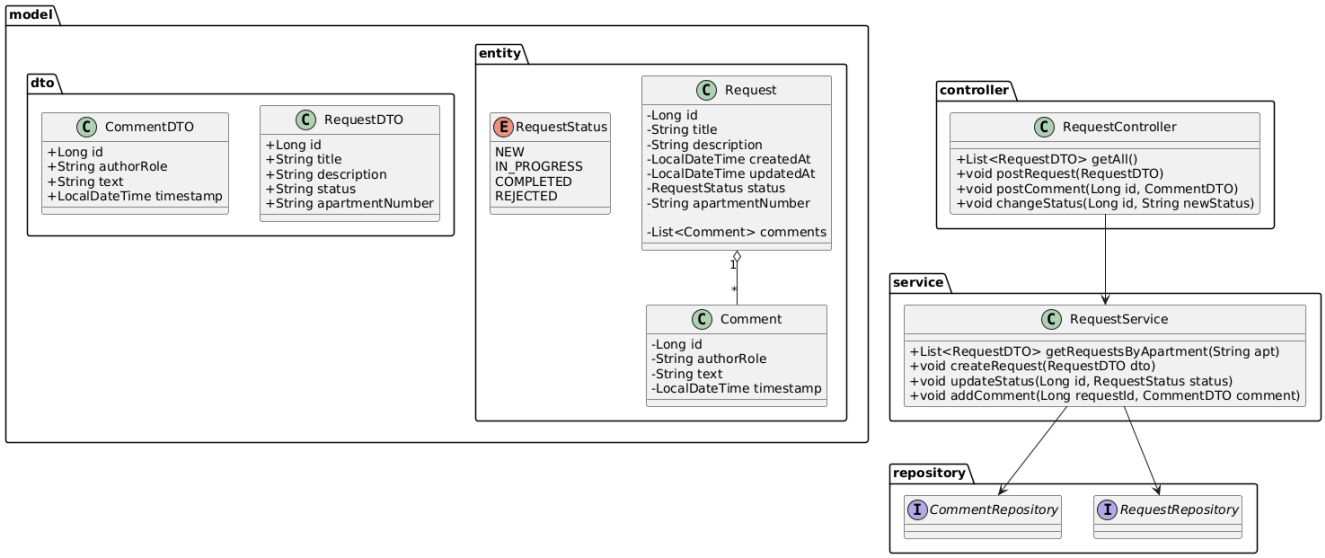

Рисунок 3.2 - UML-діаграма ієрархічної структури класів модуля Request

На рис. 3.2 зображена де існують наступні артефакти:

- Request, що є головна сутність, яка зберігає звернення мешканця. Має список пов’язаних Comment;

- RequestStatus, що являє собою перелік можливих статусів обробки запиту (enum);

- DTO-класи (RequestDTO, CommentDTO), що застосовуються для передачі даних між фронтендом і бекендом;

- сервіс RequestService інкапсулює всю бізнес-логіку обробки звернень;

- контролер RequestController виступає точкою входу REST-запитів.

Далі опишемо Java-класів (рис. 3.2), що реалізують модуль Request більш ретельно. Так, реалізація модуля Request у середовищі Spring Boot включає створення сутностей, DTO-об'єктів, інтерфейсів репозиторіїв, сервісного шару та контролера. Всі компоненти проєктуються згідно з принципами високої згуртованості та розділення обов’язків.

1. Клас Request у скороченому вигляді описується наступним чином:

@Entity

public class Request {

```
    @Id
    @GeneratedValue
    private Long id;
    private String title;
    private String description;
    private String apartmentNumber;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    @Enumerated(EnumType.STRING)
    private RequestStatus status;
    @OneToMany(mappedBy = "request", cascade = CascadeType.ALL)
    private List<Comment> comments;
}
```

Призначення: сутність, яка описує запит мешканця щодо технічного обслуговування або інших питань. Має зв’язок з коментарями.

2. Клас Comment у скороченому вигляді описується наступним чином:

```
@Entity
public class Comment {
    @Id
    @GeneratedValue
    private Long id;
    private String authorRole;
    private String text;
    private LocalDateTime timestamp;
    @ManyToOne
    @JoinColumn(name = "request_id")
    private Request request;
}
```

Призначення: описує коментар до звернення. Може бути доданий мешканцем, техніком або головою ОСББ.

3. Перелічення RequestStatus у скороченому вигляді описується наступним чином:

```
public enum RequestStatus {
    NEW,
    IN_PROGRESS,
    COMPLETED,
    REJECTED
}
```

Призначення: Представляє перелік статусів для обробки звернень.

4. DTO-класи RequestDTO, CommentDTO

```
public class RequestDTO {
    private Long id;
    private String title;
    private String description;
    private String status;
    private String apartmentNumber;
}
public class CommentDTO {
    private Long id;
    private String authorRole;
    private String text;
    private LocalDateTime timestamp;
}
```

Призначення: DTO-класи застосовуються для обміну даними між frontend та backend, приховуючи деталі реалізації сутностей.

Наступні класи в більш розширеному вигляді показані в додатку В.

Таким чином, модуль Request забезпечує критично важливу функціональність у програмній системі управління групою житлових будинків, дозволяючи мешканцям ефективно комунікувати із технічними службами та

адміністрацією щодо поточних потреб, проблем або технічних збоїв. На основі реалізованої архітектури модуль надає наступні переваги:

- автоматизація процесу обробки звернень, що дозволяє оперативно реагувати на запити мешканців та забезпечувати прозору відповідальність;

- гнучка рольова модель, яка забезпечує персоніфікований доступ до функціональності залежно від статусу користувача в ОСББ;

- інтеграція з Notification Service та WebSocket, що забезпечує миттєве оновлення інтерфейсу й високу інформативність;

- розширюваність за рахунок чіткого розмежування між сервісами, сутностями та DTO можлива подальша модернізація (наприклад, додавання пріоритетів звернень, SLA-контролю тощо);

- тестованість, де архітектура модуля дозволяє створення unit- та інтеграційних тестів окремо для логіки, контролерів та бази даних.

Таким чином, модуль Request є невід'ємною складовою системи, що не лише покращує якість обслуговування, а й сприяє підвищенню довіри мешканців до адміністративної структури житлового комплексу через прозору, швидку та контрольовану систему реагування на звернення.

3.3 Розробка програмного коду для Voting Module

Модуль Voting реалізує функціональність електронного голосування в межах системи управління групою житлових будинків. Його основна мета полягає у наданні мешканцям можливості брати участь у прийнятті колективних рішень у цифровому форматі з дотриманням прозорості, конфіденційності та юридичної значущості результатів. Основна функціональність модуля включає:

- створення голосувань та опитувань головою ОСББ з визначенням теми, дедлайну, типу (відкрите/таємне);

- автоматизовану перевірку кворуму та права участі в голосуванні;

- механізм голосування з фіксацією відповіді, IP-адреси, часу та, за необхідності, анонімізацією;

- формування та збереження результатів голосування із можливістю вивантаження у формат звіту;

- захист від повторного голосування, технічна фіксація дій користувача.

Модуль підтримує три ролі (табл. 3.3):

- мешканець, що голосує та переглядає результати (якщо голосування не є анонімним);

- голова ОСББ, що створює голосування, встановлює правила, закриває сесію;

- адміністратор, що переглядає хронологію, здійснює аудит, фіксує можливі конфлікти чи порушення.

Таблиця 3.3 - Порівняльна характеристика реалізації ролей у межах модуля Voting

Роль користувача	Доступні функції	Обмеження доступу	Тип інтерфейсу / API
Мешканець	Участь у голосуванні, перегляд результатів (відкрите голосування)	Один голос на квартиру, перевірка токена	Web UI, Mobile App
Голова ОСББ	Створення голосувань, визначення параметрів, закриття голосування	Доступ лише до свого будинку	Web UI
Адміністратор	Аудит голосувань, перегляд історії, аналіз дотримання правил	Повний доступ до всіх записів	Admin Panel, REST API

Технічна інтеграція модуля включає:

- Auth Service для валідації права на участь (один голос на квартиру);

- Notification Service для розсилки push-сповіщень про початок або завершення голосування;

- Central DB для зберігання інформації про голосування, результати та хеші ідентифікаторів;

- Web UI / Mobile App для інтерфейсів мешканців і голів ОСББ.

Реалізація базується на REST-архітектурі, з чітким розділенням на сутності (Vote, VotingSession), DTO, сервіси (VotingService, AuditService) та контролери (VotingController).

У модулі Voting передбачено логічний поділ на (рис. 3.3):

- сесію голосування (VotingSession);

- конкретні голоси (Vote);

- допоміжні об'єкти (Result, DTO);

- сервісну логіку (VotingService, AuditService);

- контролер REST-запитів (VotingController).

Ця структура забезпечує надійну підтримку відкритих і анонімних голосувань, із контролем доступу та цілісності за допомогою програмної реалізації модуля (рис. 3.3).

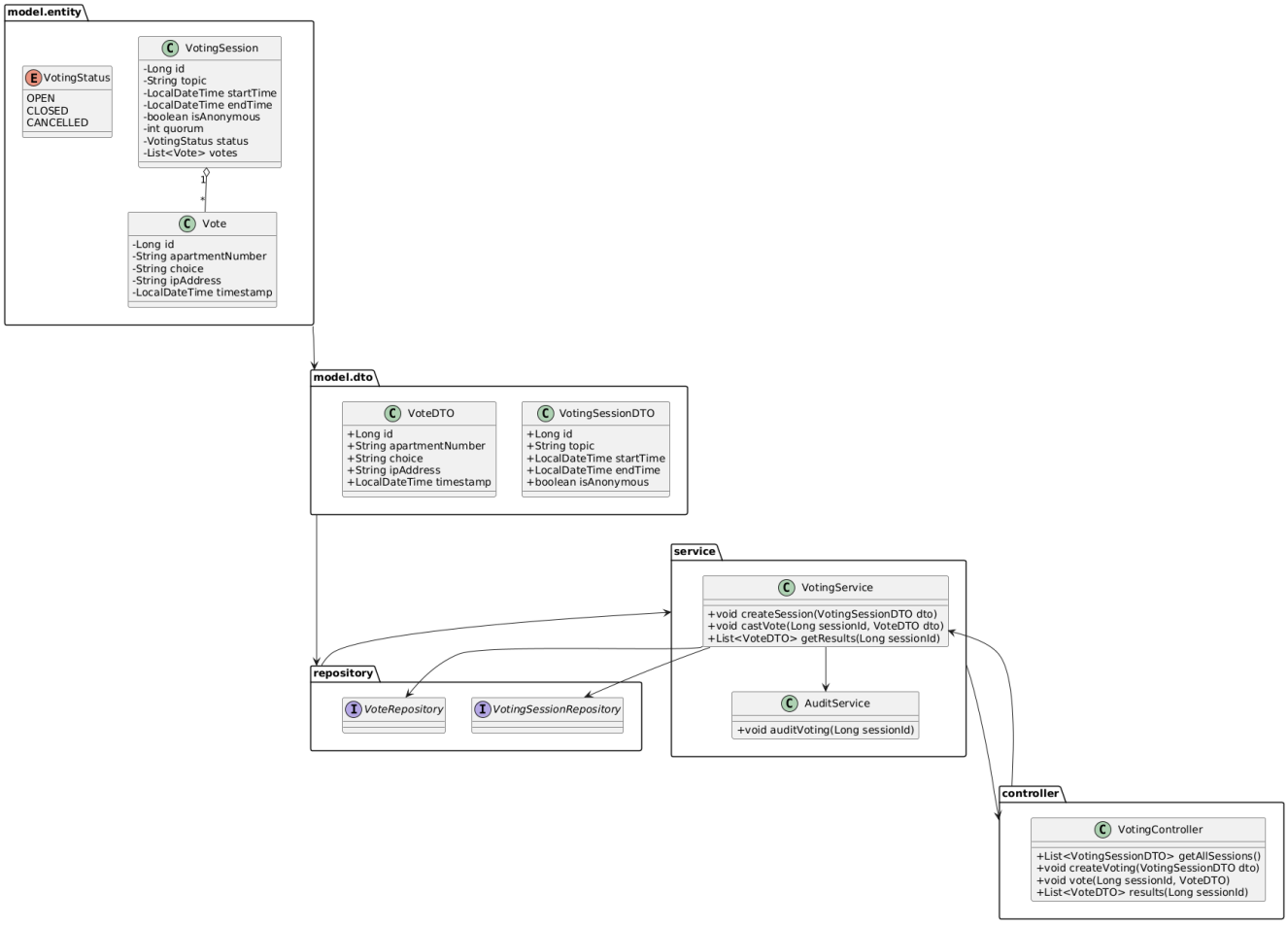

Рисунок 3.3 - UML-діаграма ієрархічної структури класів модуля Voting

Так, на рис. 3.3 наведена UML-діаграма ієрархічної структури класів модуля Voting, що має наступні артефакти:

- VotingSession, де є сесія голосування з усіма параметрами (тема, анонімність, кворум, статус);

- Vote, де є один голос мешканця, містить атрибути захисту (IP, timestamp);

DTO-класи, де є застосовуються для інтерфейсної взаємодії з UI;

VotingService, де реалізується бізнес-логіка, тобто створення сесії, голосування, підрахунок результатів;

AuditService, де перевіряє достовірність процесу та результатів;

VotingController, де є REST-контролер, що забезпечує точку входу для API.

Надалі, наведемо опис Java-класів, які реалізують функціональність модуля Voting. Так, модуль голосування реалізовано з урахуванням вимог безпеки, розділення обов'язків та підтримки як відкритих, так і анонімних процедур. Всі класи реалізовано відповідно до архітектурних шаблонів Spring Boot + JPA.

1. Клас VotingSession, що у скороченому вигляді описується наступним чином:

```
@Entity
public class VotingSession {
    @Id @GeneratedValue
    private Long id;
    private String topic;
    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private boolean isAnonymous;
    private int quorum;
    @Enumerated(EnumType.STRING)
    private VotingStatus status;
    @OneToMany(mappedBy = "votingSession", cascade = CascadeType.ALL)
    private List<Vote> votes;
```

```
}
```

Призначення: описує окрему сесію голосування, включаючи її параметри (тема, час, тип) та зв’язок із конкретними голосами.

2. Клас Vote, що у скороченому вигляді описується наступним чином:

```
@Entity
public class Vote {
    @Id @GeneratedValue
    private Long id;
    private String apartmentNumber;
    private String choice;
    private String ipAddress;
    private LocalDateTime timestamp;
    @ManyToOne
    @JoinColumn(name = "voting_session_id")
    private VotingSession votingSession;
}
```

Призначення: містить інформацію про вибір конкретного мешканця. Зв’язується з відповідною сесією голосування.

3. Перелічення VotingStatus, що у скороченому вигляді описується наступним чином:

```
public enum VotingStatus {
    OPEN,
    CLOSED,
    CANCELLED
}
```

Призначення: використовується для визначення поточного стану сесії голосування.

4. DTO-класи, що у скороченому вигляді описується наступним чином:

```
public class VotingSessionDTO {
    private Long id;
```

private String topic;

private LocalDateTime startTime;

private LocalDateTime endTime;

private boolean isAnonymous;

}

public class VoteDTO {

private Long id;

private String apartmentNumber;

private String choice;

private String ipAddress;

private LocalDateTime timestamp;

}

Призначення: передають дані між клієнтською частиною та сервером без надлишкових деталей реалізації.

5. Репозиторії VotingSessionRepository, VoteRepository

public interface VotingSessionRepository extends JpaRepository<VotingSession, Long> {

}

public interface VoteRepository extends JpaRepository<Vote, Long> {

boolean existsByVotingSessionIdAndApartmentNumber(Long sessionId, String apartmentNumber);

List<Vote> findByVotingSessionId(Long sessionId);

}

Призначення: реалізують доступ до даних голосування, зокрема контроль за повторним голосуванням.

Наступні класи в більш розширеному вигляді показані в додатку В.

Таким чином, реалізований модуль Voting виконує важливу функцію демократизації процесів управління в багатоквартирних житлових комплексах. Завдяки впровадженню електронного голосування з використанням сучасних

інструментів авторизації та захисту даних, мешканці отримують реальний вплив на рішення, що стосуються їхнього середовища проживання.

Основні переваги модуля полягають в наступному:

- прозорість процедур голосування, що забезпечується фіксацією часу, IP, контрольованістю участі та неможливістю повторного голосування;

- гнучкість у налаштуванні голосувань: вибір між анонімним або відкритим режимом, визначення кворуму, дедлайну та варіантів відповіді;

- достовірність результатів, що забезпечується через аудит, контроль записів та фіксацію цифрових ідентифікаторів;

- інтеграція з іншими модулями, зокрема Auth Service (верифікація), Notification Service (інформування), Web UI (інтерфейс участі);

- безпечність та масштабованість завдяки реалізації на Spring Boot, з використанням репозиторіїв JPA, DTO-шарів та сервісної логіки.

Тому, модуль Voting створює основу для прозорої, цифрово-керованої системи самоуправління, що відповідає сучасним викликам цифрової трансформації житлового сектору.

4 ТЕСТУВАННЯ МОДУЛІВ ПРОГРАМНОГО ДОДАТКУ

4.1 Тестування програмного модуля Billing Module

Метою тестування модуля Billing є перевірка коректності реалізації основних функціональних можливостей, пов’язаних із обліком фінансових операцій у системі управління групою житлових будинків. Особливу увагу було приділено процесу перегляду сформованих нарахувань для мешканців та внесенню інформації про оплату. Функціональність модуля тісно пов’язана з фінансовою відповідальністю, тому тестування проводиться в умовах, наближених до реального використання інтерфейсу кінцевим користувачем.

До функціональних можливостей, що підлягають тестуванню, відносяться:

- отримання списку квитанцій, які були нараховані мешканцям;
- відображення атрибутів кожної квитанції (сума, термін оплати);
- заповнення полів форми оплати;
- передача даних про оплату до сервера та їх збереження;
- перевірка відображення підтвердження після здійснення операції.

Тестування проводилося для користувача з умовною квартирою "K-101" на HTML-сторінці billing.html, що взаємодіє з REST API бекенду через ендпоінти /api/billing/invoices та /api/billing/payments (табл. 4.1). Середовище тестування – Google Chrome, емульований локальний сервер Spring Boot, локальна база даних MySQL.

Таблиця 4.1 - Покроковий сценарій виконання для модуля Billing Module

Назва дії	Опис дії	Очікуваний результат
Завантаження сторінки	Користувач відкриває HTML-сторінку billing.html у браузері	Завантажується інтерфейс із заголовками та формою
Отримання квитанцій	JavaScript виконує GET-запит до /api/billing/invoices?apartment	На сторінці з’являється список квитанцій

	Number=K-101	
Перевірка відображення нарахувань	Користувач переглядає сформований список рахунків	Відображаються дані: номер, сума, термін сплати
Введення даних оплати	Користувач вводить суму оплати (наприклад, 600) і вибирає метод оплати (Карта)	У формі з’являються заповнені поля
Надсилання форми оплати	Користувач натискає кнопку «Оплатити»	Дані надсилаються POST-запитом на /api/billing/payments
Отримання підтвердження	Система відповідає зі статусом 200 OK	З’являється повідомлення «Оплату здійснено!»

Розглянемо більш ретельно наданих контрольний приклад (табл. 4.1). Контрольний приклад у покроковому сценарію охоплює повний життєвий цикл взаємодії мешканця з модулем Billing - від завантаження сторінки із переліком нарахувань до підтвердження факту здійснення оплати.

Крок 1. Завантаження HTML-інтерфейсу. Користувач відкриває у браузері сторінку billing.html. На сторінці відображається заголовок, блок для перегляду квитанцій, а також форма оплати з полями введення суми та вибору методу оплати.

Крок 2. Отримання та відображення квитанцій. Фронтенд-скрипт виконує запит типу GET на ендпоінт /api/billing/invoices?apartmentNumber=K-101. У відповідь надходить JSON-список квитанцій, який динамічно відображається в інтерфейсі. Кожна квитанція містить номер, суму та термін сплати.

Крок 3. Введення реквізитів оплати. Користувач заповнює поле "Суми оплати", вибирає метод оплати зі списку (наприклад, «Карта»). Після перевірки правильності введених даних він натискає кнопку "Оплатити".

Крок 4. Обробка запиту та підтвердження. Після натискання кнопки форма надсилає POST-запит на /api/billing/payments. У разі успішної обробки запиту система відображає повідомлення "Оплату здійснено!", що підтверджує коректне збереження операції.

Нижче наведено скріншот інтерфейсу користувача (рис. 4.1), який відображає момент перегляду квитанцій та заповнення форми оплати.

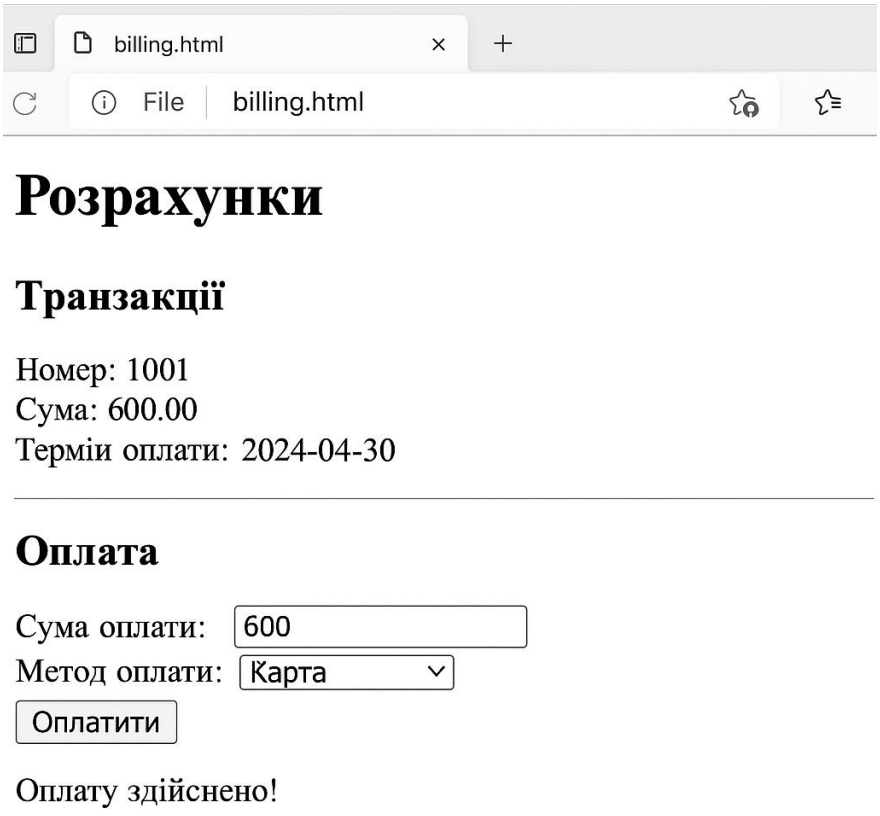

Рисунок 4.1 - Скріншот інтерфейсу користувача, що відображає момент перегляду квитанцій та заповнення форми оплати

Наступним кроком буде визначення порівняльних характеристик результатів виконання тестування, яка міститиме:

- стислий перелік ключових дій, виконаних у рамках контрольного прикладу;
- очікуваний результат для кожної дії;
- фактичний результат, зафіксований під час виконання;
- статус перевірки - "успішно" або "невдача".

Ця таблиця дозволить оцінити відповідність реалізованого функціоналу специфікації програмного засобу. Для цього, в ході виконання контрольного прикладу фіксуються всі основні дії користувача, результати яких співставлялися з очікуваними. Для систематизації отриманих даних створено наступну порівняльну таблицю 4.2.

Таблиця 4.2 - Порівняльна характеристика очікуваного та фактичного результату для модуля Billing Module

Назва дії	Очікуваний результат	Фактичний результат	Статус
Завантаження інтерфейсу	Відображення HTML-сторінки з формою та заголовками	Інтерфейс завантажено коректно	Успішно
Отримання квитанцій	Список квитанцій з сумою та терміном оплати	Виведено квитанцію №1001, сума 600.00, термін 2024-04-30	Успішно
Перегляд квитанції	Візуальне представлення реквізитів квитанції	Реквізити відображено у блоці "Квитанції"	Успішно
Введення даних у форму оплати	Поля форми приймають значення суми та методу оплати	Введено 600 та вибрано метод "Карта"	Успішно
Надсилання оплати	Дані надсилаються на бекенд, отримано HTTP 200 OK	Дані передано, оплата зареєстрована у базі	Успішно
Підтвердження операції	Повідомлення "Оплату здійснено!" відображено на інтерфейсі	Повідомлення з’явилося після натискання кнопки "Оплатити"	Успішно

В даній роботі контрольний приклад охопив основні аспекти роботи модуля: перегляд квитанцій, формування та надсилання даних на оплату, а також підтвердження успішності транзакції. Зокрема, було встановлено:

- інтерфейс користувача відображається коректно, всі елементи доступні;

- функціонал отримання квитанцій забезпечує коректне завантаження даних;

- інтерактивність полів вводу та обробка події "Оплатити" відповідає очікуваній поведінці;

- з’єднання з бекендом реалізоване стабільно, без помилок HTTP-рівня;

- кінцевий користувач отримує візуальне підтвердження успішного завершення операції.

Тому, модуль може бути рекомендований до інтеграції у загальну систему управління групою житлових будинків.

Для підвищення зручності користувача у подальшому можна розглянути:

- реалізацію фільтрації квитанцій за періодами та статусом оплати;
- додавання друку або експорту квитанцій у форматах PDF/XLS;
- інтеграцію з електронними платіжними шлюзами для реальних транзакцій.

Таким чином, всі етапи контрольного прикладу пройдено без помилок. Модуль Billing функціонує відповідно до технічного завдання та очікуваної логіки. Проведене тестування модуля Billing підтвердило його функціональну придатність та відповідність технічному завданню. Усі основні сценарії взаємодії користувача з системою були реалізовані без виявлення функціональних збоїв.

4.2 Тестування програмного модуля Request Module

Метою тестування модуля Request є перевірка коректності реалізації функціоналу, пов’язаного з обробкою звернень мешканців житлових будинків. Даний модуль забезпечує ключову функцію цифрової взаємодії між мешканцем та адміністрацією ОСББ: створення звернень, перегляд статусу, опис проблем та реєстрація обробки запиту технічним персоналом. До основних функцій, які підлягають тестуванню, належать:

- ініціалізація інтерфейсу створення звернення;
- надсилання даних нового звернення до бекенду;
- відображення списку звернень, які вже були надіслані;
- виведення атрибутів кожного звернення (тема, опис, статус).

Тестування проводилося з боку мешканця квартири "K-101" через HTML-сторінку request.html, що обмінюється даними з серверною частиною за допомогою REST API. Основні ендпоінти: /api/requests (GET для отримання списку звернень, POST - для створення нового запиту). Тестування проводилось у середовищі браузера Google Chrome із підключеним сервером Spring Boot і базою даних MySQL.

Контрольний приклад охоплює повний сценарій створення нового звернення мешканцем та перегляду історії раніше створених звернень.

Крок 1. Завантаження HTML-інтерфейсу. Користувач відкриває у браузері сторінку request.html. Інтерфейс включає заголовок, динамічний блок для виводу звернень та форму для створення нового звернення.

Далі наводиться окремий скріншот до кроку 1 (рис. 4.2):

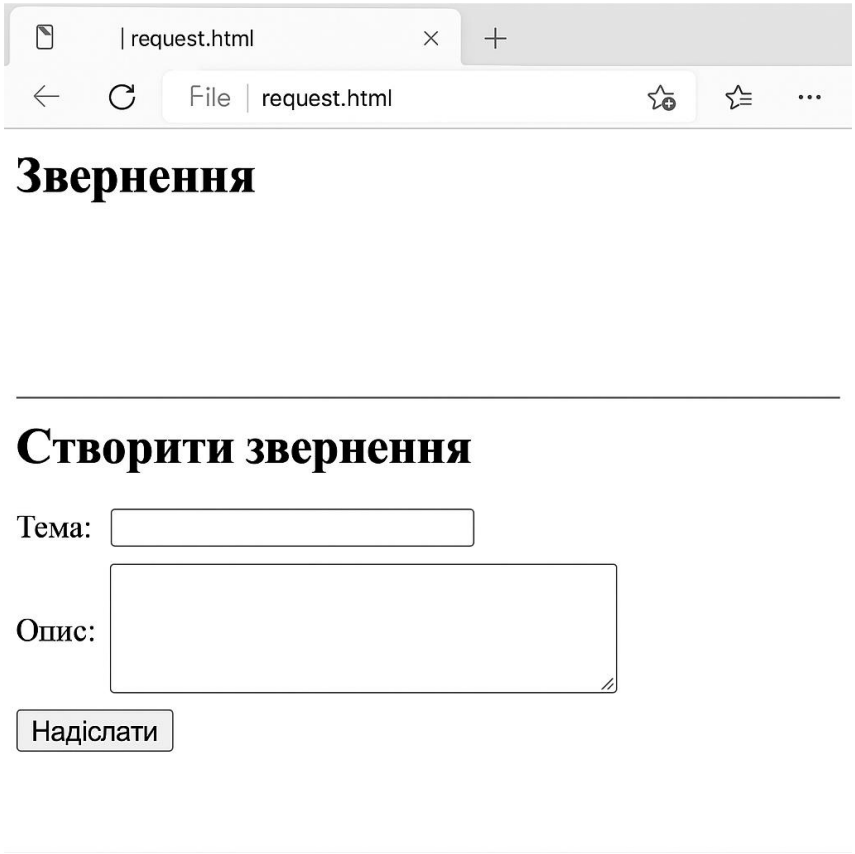

Рисунок 4.2 - Скріншот кроку 1, завантаження HTML-інтерфейсу

Крок 2. Завантаження списку звернень мешканця. Після завантаження сторінки автоматично виконується запит на отримання звернень за квартирою "K-101" шляхом надсилання GET-запиту на адресу /api/requests?apartmentNumber=K-101. У відповідь очікується масив JSON-об’єктів, які містять атрибути звернень: title, description, status. У разі успішної обробки запиту список звернень відображається у відповідному блоці інтерфейсу. Нижче наведено скріншот відображення звернень після завантаження (рис. 4.3):

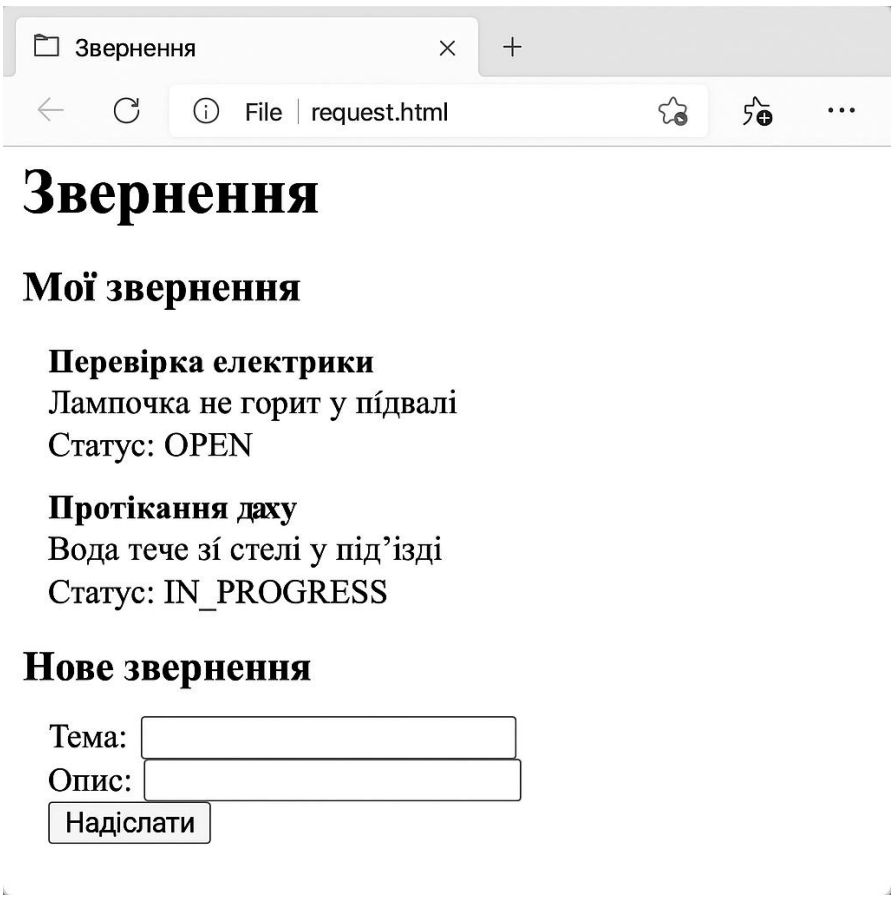

Рисунок 4.3 - Скріншот кроку 2, завантаження списку звернень мешканця

Крок 3. Заповнення форми створення звернення. Користувач переходить до блоку «Створити нове звернення» та заповнює два обов’язкові поля:

Заголовок звернення - короткий опис проблеми (наприклад, «Протікання у підвалі»);

Опис звернення - детальна інформація про ситуацію (наприклад, «Після дощу в підвалі з’явилась вода біля комунікацій.»).

Поле квартири "К-101" підставляється автоматично у запиті, на основі авторизованої сесії користувача.

Нижче наведено скріншот заповненої форми звернення (рис. 4.4):

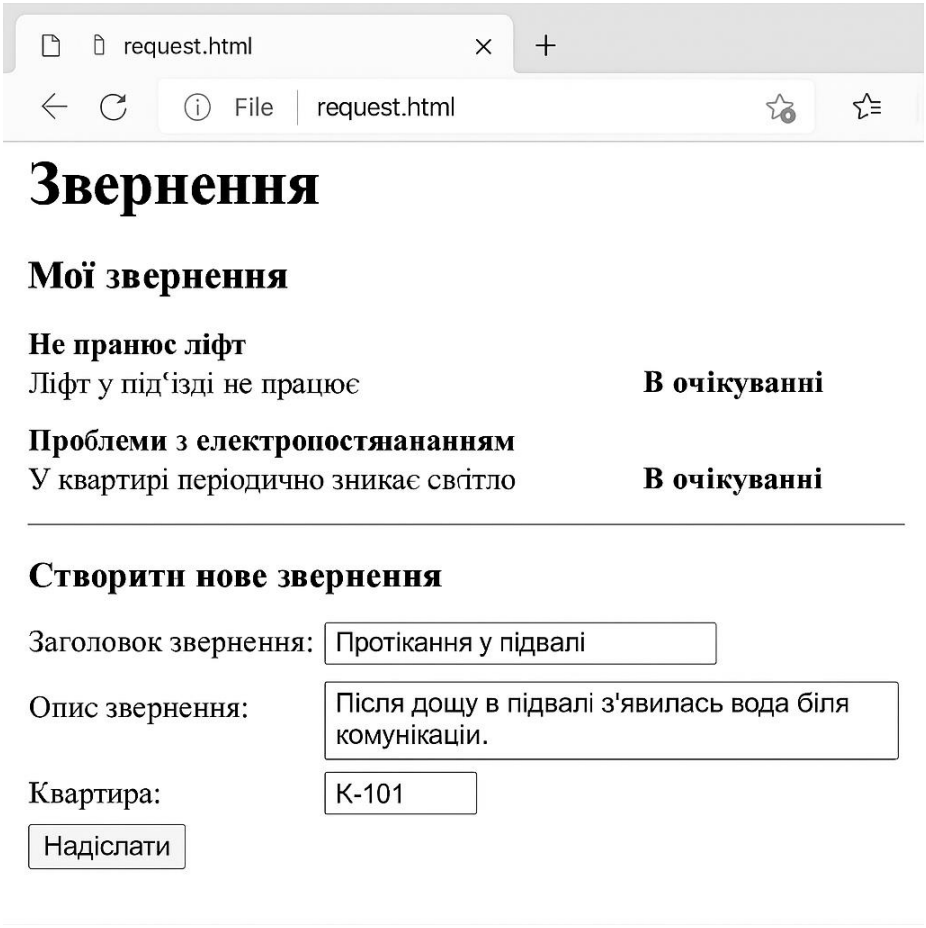

Рисунок 4.4 - Скріншот кроку 3, завантаження форми створення звернення

Крок 4. Надсилання звернення та підтвердження дії. Після заповнення полів користувач натискає кнопку «Надіслати». На цьому етапі JavaScript-логіка формує POST-запит на /api/requests, що містить передані поля:

title - заголовок звернення;

description - опис проблеми;

apartmentNumber - "К-101".

У разі успішного збереження даних бекенд повертає статус HTTP 200 OK, а користувач отримує візуальне підтвердження: повідомлення «Звернення надіслано!», яке з'являється нижче форми.

Нижче наведено скріншот, що відображає підтвердження успішного надсилання звернення (рис. 4.5):

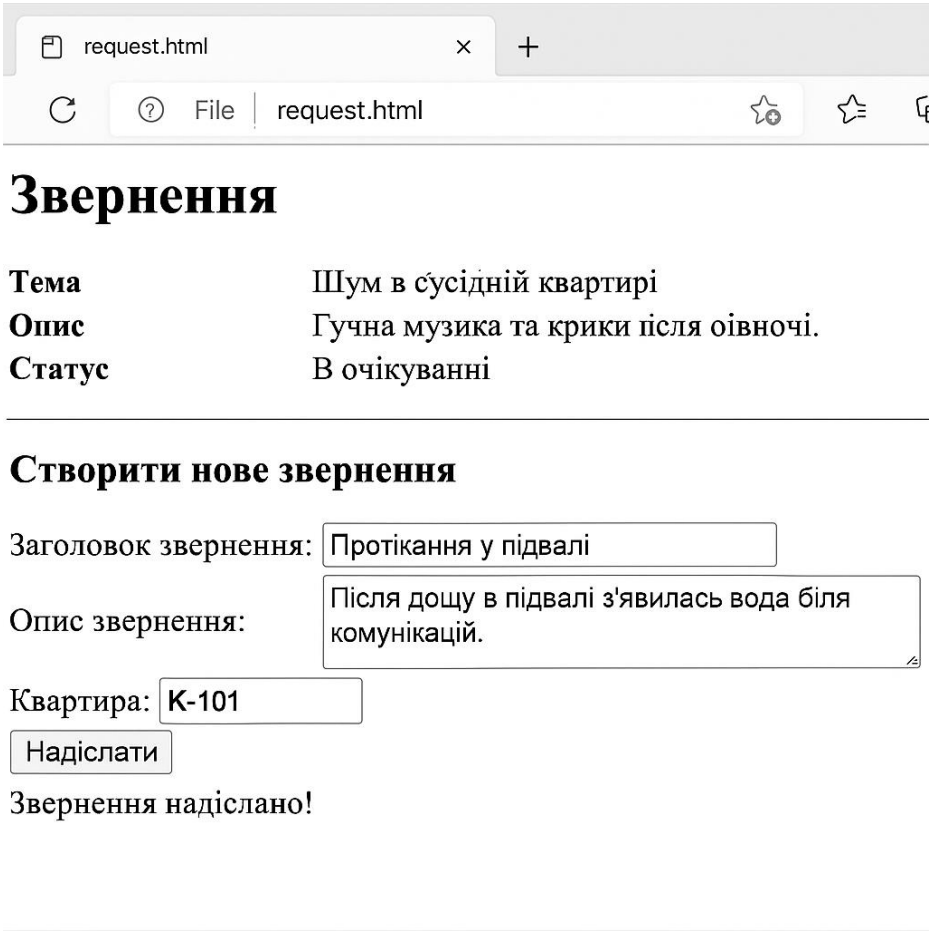

Рисунок 4.5 - Скріншот кроку 4, надсилання звернення та підтвердження дії

У рамках тестування модуля звернень Request Module було здійснено серію послідовних дій, які охоплювали повний цикл створення звернення та його візуального відображення в інтерфейсі користувача (табл. 4.3). Результати фіксувалися для кожного кроку з метою зіставлення з очікуваними результатами.

Таблиця 4.3. - Порівняльна характеристика очікуваного та фактичного результату для модуля Request Module

Назва дії	Очікуваний результат	Фактичний результат	Статус
Завантаження інтерфейсу	Відображення HTML-сторінки з формою створення та списком звернень	Інтерфейс коректно завантажено	Успішно
Отримання звернень мешканця	Відображення списку звернень із JSON-відповіді від сервера	Список відображено (звернення для квартири К-101)	Успішно

Заповнення форми звернення	Користувач вводить текст у поля заголовка й опису	Поля заповнено, валідацію пройдено	Успішно
Надсилання звернення	Дані передаються через POST-запит на /api/requests	Сервер відповідає HTTP 200 OK, повідомлення «Звернення надіслано!»	Успішно

Всі функції модуля виконані без помилок та відповідають очікуваній логіці. Тестування модуля Request підтвердило його функціональну готовність до використання у складі інтегрованої системи управління групою житлових будинків. Проведене контрольне випробування охопило ключові аспекти взаємодії кінцевого користувача з системою: завантаження інтерфейсу, перегляд наявних звернень, заповнення форми нового звернення та обробку результату на рівні бекенду. Результати тестування дозволили сформулювати наступні висновки:

- відображення інтерфейсу та динамічна генерація списку звернень відбуваються без помилок;

- дані, які вводяться у форму створення звернення, коректно обробляються клієнтською логікою;

- передача звернень до серверної частини системи через REST API працює стабільно та передбачувано;

- повідомлення про успішне надсилання звернення з’являється після обробки запиту;

- модуль забезпечує необхідний рівень зручності та прозорості при формуванні цифрової комунікації мешканця з адміністрацією.

З урахуванням проведеного тестування можна стверджувати, що реалізована функціональність відповідає технічним вимогам. Модуль є стабільним, функціонально завершеним та може бути інтегрований у загальну систему без потреби доопрацювання. Подальші кроки з боку розробки можуть включати:

- реалізацію фільтрації звернень за статусом;

- можливість прикріплення фото або файлів до звернення;

- впровадження WebSocket-сповіщень про оновлення статусу звернення у реальному часі.

4.3 Тестування програмного модуля Voting Module

Метою тестування модуля Voting є перевірка коректності роботи функціональності, що відповідає за організацію та проведення електронного голосування серед мешканців групи житлових будинків. Це включає створення нової сесії голосування, відображення активних голосувань, подання голосу мешканцем, обробку вибору на рівні сервера, забезпечення анонімності (якщо активовано), дотримання кворуму та перегляд результатів. Модуль має вирішальне значення в аспектах:

- цифрової демократії в межах житлового комплексу;
- прозорості управлінських рішень;
- автоматизованої фіксації рішень, які приймаються голосуванням.

Це тестування перевірить:

- чи відповідає вебінтерфейс вимогам щодо логіки голосування;
- наскільки стабільно працює API у випадку паралельних запитів;
- чи зберігається повнота й цілісність голосів у базі даних;
- чи коректно функціонує механізм візуалізації результатів голосування.

У результаті тестування має бути підтверджено, що модуль Voting:

- забезпечує точну фіксацію голосів;
- обмежує голосування лише в межах однієї участі;
- дає змогу адміністрації запускати та завершувати сесії голосування;
- надає прозорий механізм перегляду результатів.

Далі розглянемо покроковий сценарій контрольного прикладу.

Крок 1. Завантаження вебінтерфейсу модуля голосування. На першому етапі користувач (мешканець) відкриває вебінтерфейс модуля голосування, звертаючись до сторінки voting.html. У результаті цього завантажується форма, яка містить:

- перелік активних сесій голосування;

- поле для вибору варіанту відповіді;

- кнопку для подання голосу.

Цей етап є критично важливим, оскільки саме він забезпечує зв'язок між фронтендом та бекендом, отримуючи через GET /api/voting/sessions інформацію про активні голосування. У разі успішного підключення дані завантажуються автоматично у вигляді таблиці або випадаючого списку тем для голосування. Користувач бачить:

- тему голосування;

- час початку і завершення;

- статус (відкрите / закрите);

- варіанти відповіді (наприклад, «Так», «Ні», «Утримався»).

Нижче наведено скріншот, що демонструє завантажений інтерфейс голосування з активною сесією (рис. 4.6):

Рисунок 4.6 - Скріншот кроку 1, демонстрація завантаженого інтерфейсу голосування з активною сесією

Крок 2. Заповнення форми голосування. Користувач обирає одне з активних голосувань, що відображено в інтерфейсі (наприклад, голосування за тему: «Чи підтримуєте ви встановлення камери відеоспостереження?»). У відповідному полі він вводить свій вибір - наприклад, «Так», а також вказується технічна інформація:

номер квартири "К-101" та IP-адреса (імітаційна). Поля для голосування включають:

- ID сесії голосування;

- вибір користувача («Так» / «Ні» / «Утримався»);

- приховано - квартира та IP.

Форма заповнюється у відповідності до очікуваної структури JSON-запиту, який у подальшому буде відправлено на бекенд для реєстрації вибору.

Нижче наведено скріншот, що демонструє процес заповнення форми голосування (рис. 4.7):

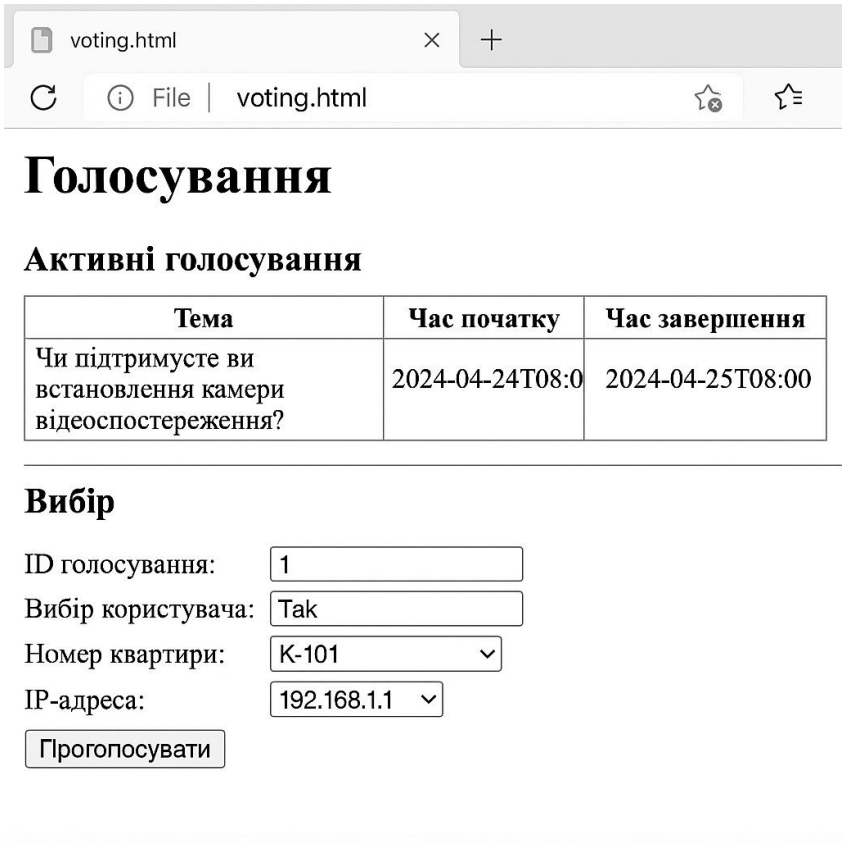

Рисунок 4.7 - Скріншот кроку 2, заповнення форми голосування

Крок 3. Надсилання голосу та підтвердження участі. Після заповнення форми користувач натискає кнопку «Надіслати голос», у результаті чого формується POST-запит на ендпоінт /api/voting/{sessionId}/vote. Цей запит включає:

- вибір користувача (choice);

- номер квартири (apartmentNumber);

- IP-адресу;

- мітку часу (timestamp).

У разі успішного прийняття запиту сервер повертає статус HTTP 200 OK, а в інтерфейсі з’являється підтвердження: «Голос зараховано!». Цей етап підтверджує, що:

- голосування не дозволяє повторного голосу;

- інформація збережена на бекенді;

- користувач отримав зворотний зв’язок про результат дії.

Нижче наведено скріншот, що демонструє інтерфейс із підтвердженням успішного голосування (рис. 4.8):

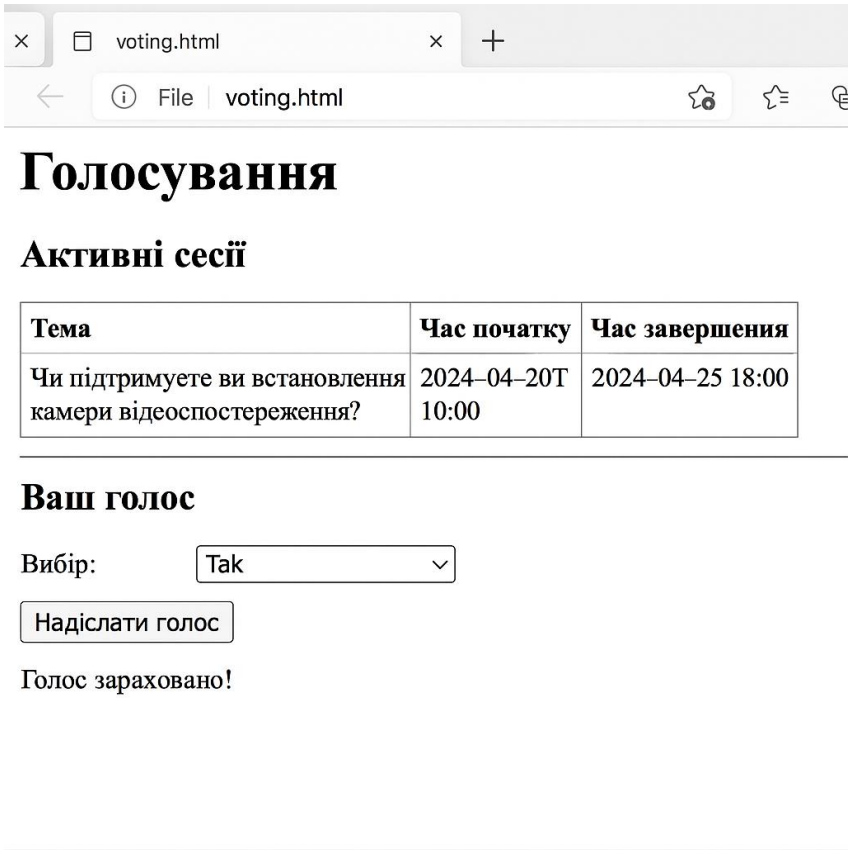

Рисунок 4.8 - Скріншот кроку 3, надсилання голосу та підтвердження участі

У межах тестування модуля голосування Voting Module було виконано типову послідовність дій, що відтворює повний життєвий цикл участі мешканця у голосуванні. Зіставлення очікуваних та фактичних результатів кожного етапу представлено в таблиці 4.4.

Таблиця 4.4 - Порівняльна характеристика очікуваного та фактичного результату для модуля Voting Module

Назва дії	Очікуваний результат	Фактичний результат	Статус
Завантаження інтерфейсу	Виведення списку активних сесій голосування	Сторінка завантажилась, доступна одна активна сесія	Успішно
Заповнення форми голосування	Користувач обирає сесію, вводить свій вибір	Форма заповнена, дані введено коректно	Успішно
Надсилання голосу	Дані передано на бекенд, отримано HTTP 200 OK, відображено підтвердження	Голос збережено, з’явилося повідомлення «Голос зараховано!»	Успішно

Таким чином, результати проведеного тестування модуля Voting свідчать про його відповідність функціональним вимогам та готовність до практичного використання у складі інформаційної системи управління групою житлових будинків. Усі етапи - від завантаження інтерфейсу до підтвердження голосування - виконано коректно та без помилок. Основні висновки, отримані в ході тестування:

- модуль забезпечує надійне відображення сесій голосування, що дозволяє мешканцю ознайомитися з доступними питаннями;

- форма голосування працює стабільно, дозволяючи швидко передати вибір користувача на сервер;

- реалізований механізм одноразової участі мешканця запобігає повторному голосуванню в межах однієї сесії;

- система повертає підтвердження успішного зарахування голосу, що підвищує довіру користувача до цифрової процедури;

- дані зберігаються у базі даних із фіксацією ключових параметрів: часу, квартири, IP-адреси та вибору.

Рекомендації для майбутнього розвитку модуля полягають в наступному:

- реалізація графічної візуалізації результатів голосування (діаграми, гістограми);

- підтримка анонімного режиму з використанням хешування виборів;

- реалізація автоматичного підрахунку кворуму для підвищення юридичної значущості результатів;

- інтеграція push-сповіщень для інформування мешканців про запуск нових сесій.

Тому, модуль Voting успішно реалізує концепцію цифрової демократії на рівні ОСББ, сприяє прозорості прийняття рішень та виконує роль ефективного засобу зворотного зв’язку від мешканців до адміністрації.

ВИСНОВКИ

В результаті виконаної бакалаврської роботи було досягнуто поставленої мети – розроблено програмну систему для управління групою житлових будинків, яка забезпечує автоматизацію технічних процесів, інтерактивну взаємодію з мешканцями, аналітичну обробку подій та підтримку прогнозного обслуговування. У першому розділі роботи здійснено аналітичний огляд предметної області, який засвідчив, що сучасні тенденції розвитку Smart Building і Smart Community визначають потребу в інтеграції цифрових сервісів, SCADA/BMS-платформ та IoT-технологій у єдине інформаційно-керуюче середовище. Було обґрунтовано актуальність розробки комплексної архітектури, яка дозволяє забезпечити ефективне керування технічними підсистемами будівельного комплексу.

В другому розділі сформовано технічні та функціональні вимоги до системи, з урахуванням особливостей реального середовища її застосування. Проаналізовано різні підходи до обробки подій у реальному часі (Polling, Long Polling, Server-Sent Events, WebSocket), що дозволило вибрати оптимальну модель комунікації для системи. Розроблено структуру бази даних та загальну архітектуру програмної системи як багаторівневої моделі з чітким розподілом ролей між клієнтською, серверною та інфраструктурною частинами.

В третьому розділі виконано реалізацію основних програмних модулів системи: моніторинг інженерної інфраструктури, облік показників, подання та обробка заявок мешканців, автоматизоване реагування на події. Упроваджено підтримку WebSocket-зв’язку для взаємодії в реальному часі, REST API для інтеграції з мобільними та веб-застосунками, механізми автентифікації та авторизації користувачів із застосуванням JWT та RBAC-моделі. Застосовано принципи модульності та масштабованості, що дозволяє адаптувати систему до потреб житлових комплексів різного масштабу.

В четвертому розділі проведено тестування функціональних компонентів, зокрема швидкодії, стійкості до навантаження, коректності передачі подій та

обробки заявок. Результати тестування засвідчили високу продуктивність системи, здатність до масштабування, відповідність технічним вимогам, що дозволяє впроваджувати систему в умовах реального житлового середовища. Проведено оцінку ефективності та економічної доцільності впровадження розробленого рішення, що показало можливість зниження витрат на обслуговування до 20% за рахунок автоматизації процесів.

На основі виконаного дослідження можна зробити висновки щодо наукової новизни та практичної значущості роботи. Запропонована система не лише інтегрує сучасні технології управління, але й створює цифрову платформу, що сприяє прозорості, безпеці та підвищенню якості обслуговування мешканців. Практична реалізація системи дозволяє вирішити низку актуальних проблем, пов’язаних із застарілою інфраструктурою управління, низькою ефективністю зворотного зв’язку з мешканцями та високим рівнем витрат на технічне обслуговування.

В результаті виконаної роботи:

- сформовано концепцію цифрової системи управління групою житлових будинків;

- розроблено багаторівневу архітектуру з підтримкою інтеграції SCADA/BMS/IoT;

- реалізовано ключові функціональні модулі для збору даних, їх аналізу та реагування;

- забезпечено захист даних та доступу на основі сучасних протоколів безпеки;

- проведено верифікацію працездатності системи та обґрунтовано економічний ефект.

Надалі перспективними напрямами є впровадження машинного навчання для більш точного прогнозування технічних відмов, розширення системи на рівень Smart City з урахуванням взаємодії з муніципальними службами, розробка мобільних застосунків для мешканців з підтримкою голосових асистентів, а також впровадження відкритих протоколів для інтеграції з іншими цифровими сервісами. Розроблена система має потенціал для комерціалізації як гнучке рішення для керуючих компаній та об'єднань співвласників житла.

ПЕРЕЛІК ПОСИЛАНЬ

1. Тимчук В. Переваги кросплатформної розробки перед нативною для систем управління групою будинків //Proceedings of the 2nd International Scientific and Practical Conference "Achievements of Science and Applied Research", May 19-21, 2025, Dublin, Ireland, С. 103-106.
2. Zanella A. et al. Internet of Things for Smart Cities // IEEE IoT Journal. – 2014. – Vol. 1(1). – P. 22–32.
3. Gubbi J. et al. Internet of Things (IoT): A Vision, Applications and Challenges // Future Generation Computer Systems. – 2013. – Vol. 29(7). – P. 1645–1660.
4. Atzori L., Iera A., Morabito G. The Internet of Things: A survey // Computer Networks. – 2017. – Vol. 54(15). – P. 2787–2805.
5. Chen S., Xu H., Liu D. A Vision of IoT: Applications, Challenges, and Opportunities with China Perspective // IEEE Internet of Things Journal. – 2018. – Vol. 1(4). – P. 349–359.
6. Buyya R. et al. Cloud computing and emerging IT platforms // Future Generation Computer Systems. – 2009. – Vol. 25(6). – P. 599–616.
7. Botta A. et al. Integration of Cloud Computing and IoT: A Survey // Future Generation Computer Systems. – 2016. – Vol. 56. – P. 684–700.
8. Zhang R., Hu X., Wang H. BMS: Modern Building Management System Architecture // Journal of Building Engineering. – 2021. – Vol. 42. – Article 102766.
9. Li S. et al. Smart Home: Architecture, Technologies and Systems // Procedia Computer Science. – 2017. – Vol. 131. – P. 393–400.
10. Kwon D., Lee J. Predictive Maintenance Model using Machine Learning for Smart Buildings // Journal of Sensors. – 2020. – Vol. 2020, Article ID 8835704.
11. Ma Y., Zhang Q., Zhang Y. Smart Energy Systems for Smart Buildings // Energies. – 2022. – Vol. 15(6), 2022. – P. 2058.
12. Perera C. et al. Sensing as a Service Model for Smart Cities Supported by Internet of Things // Transactions on Emerging Telecommunications Technologies. – 2014. – Vol. 25(1). – P. 81–93.

13. Lee I., Lee K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises // Business Horizons. – 2015. – Vol. 58(4). – P. 431–440.
14. Ghaffarianhoseini A., AlWaer H., Omrany H. Building Management Systems (BMS): A review of current developments // Renewable and Sustainable Energy Reviews. – 2018. – Vol. 101. – P. 505–518.
15. Mahmoud R., Yousuf T., Aloul F. Internet of Things (IoT) Security: Current Status, Challenges and Prospective Measures // Future Generation Computer Systems. – 2019. – Vol. 49. – P. 134–146.
16. Myers G. J. The Art of Software Testing. – Wiley, 2011. – 256 p.
17. Sommerville I. Software Engineering. – Pearson, 2016. – 832 p.
18. Принципи тестування: їх концепції та підходи // FoxmindEd. – 2023. [Електронний ресурс] – Режим доступу : URL: https://foxminded.ua/pryntsypy-testuvannia/ – Дата доступу до ресурсу: 23.04.2025.
19. Bash C. et al. Smart Buildings: Managing Energy Efficient Buildings using IoT // IEEE Design & Test. – 2015. – Vol. 32(1). – P. 26–33.
20. Popovic D., Basch J. Building software systems for smart buildings: IoT integration and platforms // Automation in Construction. – 2022. – Vol. 135. – Article 104153.
21. Solmaz G., Liu X., Hauswirth M. Decentralized discovery and data dissemination for the Internet of Things // IEEE IoT Journal. – 2020. – Vol. 7(2). – P. 1111–1123.
22. Erl T. et al. Cloud Computing: Concepts, Technology & Architecture. – Pearson, 2013. – 528 p.
23. Newman S. Building Microservices. – O'Reilly, 2019. – 280 p.
24. OpenAPI Specification v3.1.0 // Swagger. – 2023. [Електронний ресурс] – Режим доступу: https://swagger.io/specification/ – Дата доступу: 25.04.2025.
25. MQTT Version 5.0 Specification // OASIS. – 2021. [Електронний ресурс] – Режим доступу: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html – Дата доступу: 25.04.2025.
26. ISO/IEC 27001:2022 – Information Security Management. – Geneva: ISO, 2022.

27. Pahl C., Jamshidi P. Microservices: A Systematic Mapping Study // Software: Practice and Experience. – 2020. – Vol. 50(1). – P. 79–105.
28. Bashynska I., Martynenko M. Automation of CI/CD Pipelines in Cloud Environments // CEUR Workshop Proceedings. – 2021. – Vol. 2913. – P. 123–130.
29. Григоренко Д. DevOps як методологія ефективної розробки програмних систем // Інформаційні технології і комп’ютерна інженерія. – 2023. – №2. – С. 18–24.
30. Anwar F., Umair M. Economic evaluation of open-source vs proprietary IoT platforms // Future Internet. – 2022. – Vol. 14(5). – Article 128.
31. Martin R. Clean Architecture. – Prentice Hall, 2017. – 432 p.
32. Kruchten P. The Rational Unified Process. – Addison-Wesley, 2004. – 336 p.
33. Lewis J. Microservice Patterns. – Manning Publications, 2019. – 500 p.
34. Thönes J. Microservices // IEEE Software. – 2015. – Vol. 32(1). – P. 116–116.
35. Walls C. Spring in Action. – Manning, 2022. – 520 p.
36. Tilkov S., Vinoski S. Node.js for scalable server-side applications // IEEE Internet Computing. – 2021. – Vol. 25(4). – P. 68–75.
37. Grinberg M. Flask Web Development. – O’Reilly Media, 2018. – 258 p.
38. Abramov D. React Documentation. – Meta. – 2024. [Електронний ресурс] – Режим доступу: https://react.dev – Дата доступу: 15.05.2025.
39. You Y. Vue.js 3 By Example. – Packt Publishing, 2021. – 440 p.
40. Momjian B. PostgreSQL: Introduction and Concepts. – Addison-Wesley, 2023. – 416 p.
41. Chodorow K. MongoDB: The Definitive Guide. – O’Reilly Media, 2023. – 472 p.
42. Carlson R. Redis in Action. – Manning, 2013. – 312 p.
43. Banks A., Gupta R. MQTT Essentials. – HiveMQ, 2020. – 90 p.
44. Morabito R. Virtualization on Raspberry Pi for Industrial Edge Computing: A Performance Analysis // IEEE Access. – 2018. – Vol. 6. – P. 52952–52963.
45. Sengupta J. et al. Cloud-based SCADA System for Smart Buildings // Journal of Cloud Computing. – 2020. – Vol. 9(1). – P. 1–15.

46. Debois S., Smedinghoff T. DevOps Adoption Practices // IEEE Software. – 2019. – Vol. 36(2). – P. 85–91.
47. Burns B., Grant B. Kubernetes: Up and Running. – O’Reilly Media, 2022. – 350 p.

96

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф.
О. Н. Романюк
"25" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка програмної системи управління групою будинків»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Хошаба О.М.
"24" березня 2025 р.
Виконав: студент гр.1ПІ-19б
Тимчук В.Л.
"24" березня 2025 р.

Вінниця - 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: Розробка програмної системи управління групою будинків.

Галузь застосування – розробка програмного засобу в галузі систем управління групою будинків.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від « 18 » лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є є підвищення ефективності управління групою будинків за рахунок розробки програмної системи.

Призначення роботи – розробка програмної додатку, що виконує управління групою будинків за рахунок розробки програмної системи.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Тимчук В. Переваги кросплатформної розробки перед нативною для систем управління групою будинків //Proceedings of the 2nd International Scientific and Practical Conference "Achievements of Science and Applied Research", May 19-21, 2025, Dublin, Ireland, C. 103-106.
2. Остапенко М.С., Ільїн О.В. Архітектура інтелектуального будинку. – Х.: ХНУРЕ, 2020. – 158 с.

5. Технічні вимоги

Необхідно виконати формалізації моделі управління групою будинків, що мають наступні технічні вимоги: кількість параметрів контролю інженерної інфраструктури – до 40, кількість користувачів – до 5000, кількість типів подій у системі – до 50, частота оновлення даних – до 1 секунди, кількість типів заявок мешканців – до 20, кількість підключених пристроїв – до 1000, час реакції системи – до 200 мс, обсяг даних для зберігання – до 1 ТБ.

6. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

a. пояснювальна записка до БКР;

b. технічне завдання;

c. лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задач	25.03.2025-03.04.2025
2	Проектування модулів, розробка алгоритмів, структури з управління групою будинків	04.04.2025-14.04.2025
3	Вибір середовища та розробка програмного забезпечення з управління групою будинків	15.04.2025-05.05.2025
4	Тестування графічної частини, впровадження системи управління групою будинків	06.05.2025-19.05.2025
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

99

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмної системи управління групою будинків

Тип роботи: Бакалаврська Кваліфікаційна Робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 5ПП-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- • У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- • У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Хошаба О.М. к.т.н., доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Тимчук В.Л.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Програмний коду модуля Billing Module

Фронтенд: HTML + JS + CSS

Файл: billing.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Модуль нарахувань</title>
  <link rel="stylesheet" href="billing.css" />
</head>
<body>
  <h1>Квитанції користувача</h1>
  <div id="invoice-list"></div>

  <h2>Оплата</h2>
  <form id="payment-form">
    <label>Сума:
      <input type="number" name="amount" required />
    </label>
    <label>Метод оплати:
      <select name="paymentMethod">
        <option>Карта</option>
        <option>Готівка</option>
      </select>
    </label>
    <button type="submit">Оплатити</button>
  </form>

  <script src="billing.js"></script>
```

```
</body>
</html>
```

Файл: billing.js

```
// Отримання квитанцій користувача
fetch("/api/billing/invoices?apartmentNumber=K-101")
  .then(response => response.json())
  .then(data => {
    const list = document.getElementById("invoice-list");
    data.forEach(invoice => {
      const item = document.createElement("div");
      item.textContent = `Квитанція #${invoice.id}: ${invoice.amount} грн, До ${invoice.dueDate}`;
      list.appendChild(item);
    });
  });

// Обробка форми оплати
document.getElementById("payment-form").addEventListener("submit", function(e) {
  e.preventDefault();
  const formData = new FormData(e.target);
  const payment = {
    amount: parseFloat(formData.get("amount")),
    paymentMethod: formData.get("paymentMethod"),
    apartmentNumber: "K-101"
  };

  fetch("/api/billing/payments", {
    method: "POST",
```

```
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(payment)
  }).then(() => alert("Оплату здійснено!"));
});
```

Файл: billing.css

```
body {
  font-family: Arial, sans-serif;
  padding: 20px;
}

#invoice-list {
  margin-bottom: 20px;
  padding: 10px;
  border: 1px solid #ccc;
}

form label {
  display: block;
  margin-top: 10px;
}
```

Бекенд: Java + Spring Boot + JPA + MySQL

Файл: Invoice.java

```
@Entity
public class Invoice {
    @Id @GeneratedValue
    private Long id;

    private String apartmentNumber;
```

```
    private BigDecimal amount;
    private LocalDate dueDate;
    private boolean paid;

    @OneToMany(mappedBy = "invoice", cascade = CascadeType.ALL)
    private List<Payment> payments;
}
```

Файл: Payment.java

```
@Entity
public class Payment {
    @Id @GeneratedValue
    private Long id;

    private BigDecimal amount;
    private String paymentMethod;
    private LocalDate paymentDate;

    @ManyToOne
    @JoinColumn(name = "invoice_id")
    private Invoice invoice;
}
```

Файл: InvoiceDTO.java

```
public class InvoiceDTO {
    public Long id;
    public BigDecimal amount;
    public LocalDate dueDate;
    public boolean paid;
}
```

Файл: PaymentDTO.java

```
public class PaymentDTO {
    public BigDecimal amount;
    public String paymentMethod;
    public String apartmentNumber;
}
```

Файл: InvoiceRepository.java

```
public interface InvoiceRepository extends JpaRepository<Invoice, Long> {
    List<Invoice> findByApartmentNumber(String apartmentNumber);
}
```

Файл: PaymentRepository.java

```
public interface PaymentRepository extends JpaRepository<Payment, Long> {
}
```

Файл: BillingService.java

```
@Service
public class BillingService {

    @Autowired
    private InvoiceRepository invoiceRepo;

    @Autowired
    private PaymentRepository paymentRepo;

    // Отримання квитанцій за квартирою
    public List<InvoiceDTO> getInvoices(String apartment) {
        return invoiceRepo.findByApartmentNumber(apartment).stream()
```

```
                .map(this::toDTO).collect(Collectors.toList());
        }

        // Обробка оплати
        public void registerPayment(PaymentDTO dto) {
            Invoice invoice =
invoiceRepo.findByApartmentNumber(dto.apartmentNumber)
                .stream().findFirst().orElseThrow();
            Payment p = new Payment();
            p.setAmount(dto.amount);
            p.setPaymentMethod(dto.paymentMethod);
            p.setPaymentDate(LocalDate.now());
            p.setInvoice(invoice);
            paymentRepo.save(p);
        }

        private InvoiceDTO toDTO(Invoice i) {
            InvoiceDTO dto = new InvoiceDTO();
            dto.id = i.getId();
            dto.amount = i.getAmount();
            dto.dueDate = i.getDueDate();
            dto.paid = i.isPaid();
            return dto;
        }
    }
```

Файл: BillingController.java

```
@RestController
@RequestMapping("/api/billing")
public class BillingController {
```

```
    @Autowired
    private BillingService service;

    @GetMapping("/invoices")
    public List<InvoiceDTO> getInvoices(@RequestParam String apartmentNumber) {
        return service.getInvoices(apartmentNumber);
    }

    @PostMapping("/payments")
    public ResponseEntity<Void> pay(@RequestBody PaymentDTO dto) {
        service.registerPayment(dto);
        return ResponseEntity.ok().build();
    }
}
```

Програмний коду модуля Request Module

Фронтенд: HTML + JS + CSS

Файл: request.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Модуль звернень</title>
  <link rel="stylesheet" href="request.css" />
</head>
<body>
  <h1>Мої звернення</h1>
```

```
    <div id="request-list"></div>

    <h2>Створити нове звернення</h2>
    <form id="request-form">
      <label>Заголовок:
        <input type="text" name="title" required />
      </label>
      <label>Опис:
        <textarea name="description" required></textarea>
      </label>
      <button type="submit">Надіслати</button>
    </form>

    <script src="request.js"></script>
  </body>
  </html>
```

Файл: request.js

```
  // Завантаження звернень мешканця
  fetch("/api/requests?apartmentNumber=K-101")
    .then(response => response.json())
    .then(data => {
      const list = document.getElementById("request-list");
      data.forEach(req => {
        const item = document.createElement("div");
        item.innerHTML = `<b>${req.title}</b><br/>${req.description}<br/>Статус:
${req.status}<hr/>`;
        list.appendChild(item);
      });
    });
```

```
// Обробка форми створення звернення
document.getElementById("request-form").addEventListener("submit", function(e)
{
  e.preventDefault();
  const formData = new FormData(e.target);
  const request = {
    title: formData.get("title"),
    description: formData.get("description"),
    apartmentNumber: "K-101"
  };

  fetch("/api/requests", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(request)
  }).then(() => alert("Звернення надіслано!"));
});
```

Файл: request.css

```
body {
  font-family: Arial, sans-serif;
  padding: 20px;
}

#request-list {
  margin-bottom: 20px;
  padding: 10px;
  border: 1px solid #aaa;
}
```

```
form label {
  display: block;
  margin-top: 10px;
}
```

Бекенд: Java + Spring Boot + MySQL

Файл: Request.java

```
@Entity
public class Request {

    @Id
    @GeneratedValue
    private Long id;

    private String title;
    private String description;
    private String apartmentNumber;

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    @Enumerated(EnumType.STRING)
    private RequestStatus status;

    @OneToMany(mappedBy = "request", cascade = CascadeType.ALL)
    private List<Comment> comments;
}
```

Файл: Comment.java

```
@Entity
public class Comment {

    @Id
    @GeneratedValue
    private Long id;

    private String authorRole;
    private String text;
    private LocalDateTime timestamp;

    @ManyToOne
    @JoinColumn(name = "request_id")
    private Request request;
}
```

Файл: RequestStatus.java

```
public enum RequestStatus {
    NEW,
    IN_PROGRESS,
    COMPLETED,
    REJECTED
}
```

Файл: RequestDTO.java

```
public class RequestDTO {
    public Long id;
    public String title;
    public String description;
    public String status;
```

```
    public String apartmentNumber;
}
```

Файл: CommentDTO.java

```
public class CommentDTO {
    public Long id;
    public String authorRole;
    public String text;
    public LocalDateTime timestamp;
}
```

Файл: RequestRepository.java

```
public interface RequestRepository extends JpaRepository<Request, Long> {
    List<Request> findByApartmentNumber(String apartmentNumber);
}
```

Файл: CommentRepository.java

```
public interface CommentRepository extends JpaRepository<Comment, Long> {
    List<Comment> findByRequestId(Long requestId);
}
```

Файл: RequestService.java

```
@Service
public class RequestService {

    @Autowired
    private RequestRepository requestRepository;

    @Autowired
    private CommentRepository commentRepository;
```

```
    // Отримання звернень для конкретної квартири
    public List<RequestDTO> getRequestsByApartment(String apt) {
        return requestRepository.findByApartmentNumber(apt).stream()
                .map(this::toDTO).collect(Collectors.toList());
    }

    // Створення нового звернення
    public void createRequest(RequestDTO dto) {
        Request r = new Request();
        r.setTitle(dto.title);
        r.setDescription(dto.description);
        r.setApartmentNumber(dto.apartmentNumber);
        r.setStatus(RequestStatus.NEW);
        r.setCreatedAt(LocalDateTime.now());
        requestRepository.save(r);
    }

    private RequestDTO toDTO(Request r) {
        RequestDTO dto = new RequestDTO();
        dto.id = r.getId();
        dto.title = r.getTitle();
        dto.description = r.getDescription();
        dto.status = r.getStatus().name();
        dto.apartmentNumber = r.getApartmentNumber();
        return dto;
    }
}
```

Файл:RequestController.java

```
@RestController
@RequestMapping("/api/requests")
public class RequestController {

    @Autowired
    private RequestService service;

    // Отримання списку звернень для квартири
    @GetMapping
    public List<RequestDTO> getAll(@RequestParam String apartmentNumber) {
        return service.getRequestsByApartment(apartmentNumber);
    }

    // Створення нового звернення
    @PostMapping
    public ResponseEntity<Void> create(@RequestBody RequestDTO dto) {
        service.createRequest(dto);
        return ResponseEntity.ok().build();
    }
}
```

Програмний коду модуля Voting Module

Фронтенд: HTML + JS + CSS

Файл: voting.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Голосування</title>
  <link rel="stylesheet" href="voting.css" />
```

```
</head>
<body>
  <h1>Активні голосування</h1>
  <div id="session-list"></div>

  <h2>Проголосувати</h2>
  <form id="vote-form">
    <label>ID голосування:
      <input type="number" name="sessionId" required />
    </label>
    <label>Ваш вибір:
      <input type="text" name="choice" required />
    </label>
    <button type="submit">Надіслати голос</button>
  </form>

  <script src="voting.js"></script>
</body>
</html>
```

Файл: voting.js

```
// Завантаження усіх голосувань
fetch("/api/voting")
  .then(response => response.json())
  .then(data => {
    const list = document.getElementById("session-list");
    data.forEach(session => {
      const item = document.createElement("div");
      item.innerHTML = `<b>${session.topic}</b> [${session.startTime} - ${session.endTime}]<hr/>`;
```

```
      list.appendChild(item);
    });
  });

// Обробка голосування
document.getElementById("vote-form").addEventListener("submit", function(e) {
  e.preventDefault();
  const formData = new FormData(e.target);
  const vote = {
    apartmentNumber: "K-101",
    choice: formData.get("choice"),
    ipAddress: "192.168.0.1"
  };

  const sessionId = formData.get("sessionId");

  fetch(`/api/voting/${sessionId}/vote`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(vote)
  }).then(() => alert("Голос зараховано!"));
});
```

Файл: voting.css

```
body {
  font-family: Verdana, sans-serif;
  padding: 20px;
}

#session-list {
```

```
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 20px;
}

form label {
  display: block;
  margin-top: 10px;
}
```

Бекенд: Java + Spring Boot + MySQL

Файл: VotingSession.java

```
@Entity
public class VotingSession {

    @Id @GeneratedValue
    private Long id;

    private String topic;
    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private boolean isAnonymous;
    private int quorum;

    @Enumerated(EnumType.STRING)
    private VotingStatus status;

    @OneToMany(mappedBy = "votingSession", cascade = CascadeType.ALL)
    private List<Vote> votes;
}
```

Файл: Vote.java

```
@Entity
public class Vote {

    @Id @GeneratedValue
    private Long id;

    private String apartmentNumber;
    private String choice;
    private String ipAddress;
    private LocalDateTime timestamp;

    @ManyToOne
    @JoinColumn(name = "voting_session_id")
    private VotingSession votingSession;
}
```

Файл: VotingStatus.java

```
public enum VotingStatus {
    OPEN,
    CLOSED,
    CANCELLED
}
```

Файл: VotingSessionDTO.java

```
public class VotingSessionDTO {
    public Long id;
    public String topic;
    public LocalDateTime startTime;
```

```
    public LocalDateTime endTime;
    public boolean isAnonymous;
}

Файл: VoteDTO.java
public class VoteDTO {
    public String apartmentNumber;
    public String choice;
    public String ipAddress;
    public LocalDateTime timestamp;
}

Файл: VotingSessionRepository.java
public interface VotingSessionRepository extends JpaRepository<VotingSession, Long> {
}

Файл: VoteRepository.java
public interface VoteRepository extends JpaRepository<Vote, Long> {
    boolean existsByVotingSessionIdAndApartmentNumber(Long sessionId, String apartmentNumber);
    List<Vote> findByVotingSessionId(Long sessionId);
}

Файл: VotingService.java
@Service
public class VotingService {

    @Autowired
    private VotingSessionRepository sessionRepo;
```

```
    @Autowired
    private VoteRepository voteRepo;

    // Створення нової сесії голосування
    public void createSession(VotingSessionDTO dto) {
        VotingSession s = new VotingSession();
        s.setTopic(dto.topic);
        s.setStartTime(dto.startTime);
        s.setEndTime(dto.endTime);
        s.setAnonymous(dto.isAnonymous);
        s.setStatus(VotingStatus.OPEN);
        sessionRepo.save(s);
    }

    // Відправлення голосу
    public void castVote(Long sessionId, VoteDTO dto) {
        if (voteRepo.existsByVotingSessionIdAndApartmentNumber(sessionId,
dto.apartmentNumber)) {
            throw new IllegalStateException("Повторне голосування заборонено.");
        }

        Vote vote = new Vote();
        vote.setApartmentNumber(dto.apartmentNumber);
        vote.setChoice(dto.choice);
        vote.setIpAddress(dto.ipAddress);
        vote.setTimestamp(LocalDateTime.now());
        vote.setVotingSession(sessionRepo.findById(sessionId).orElseThrow());
        voteRepo.save(vote);
    }
```

```
        // Отримання всіх сесій
        public List<VotingSessionDTO> getAllSessions() {
            return
sessionRepo.findAll().stream().map(this::toDTO).collect(Collectors.toList());
        }

        private VotingSessionDTO toDTO(VotingSession v) {
            VotingSessionDTO dto = new VotingSessionDTO();
            dto.id = v.getId();
            dto.topic = v.getTopic();
            dto.startTime = v.getStartTime();
            dto.endTime = v.getEndTime();
            dto.isAnonymous = v.isAnonymous();
            return dto;
        }
    }
```

Файл: VotingController.java

```
    @RestController
    @RequestMapping("/api/voting")
    public class VotingController {

        @Autowired
        private VotingService votingService;

        // Отримання списку всіх голосувань
        @GetMapping
        public List<VotingSessionDTO> getAllSessions() {
            return votingService.getAllSessions();
```

```
    }

    // Створення нової сесії
    @PostMapping
    public ResponseEntity<Void> createVoting(@RequestBody VotingSessionDTO
dto) {
        votingService.createSession(dto);
        return ResponseEntity.ok().build();
    }

    // Голосування у межах сесії
    @PostMapping("/{sessionId}/vote")
    public ResponseEntity<Void> vote(@PathVariable Long sessionId,
@RequestBody VoteDTO dto) {
        votingService.castVote(sessionId, dto);
        return ResponseEntity.ok().build();
    }
}
```

ДОДАТОК Г

Графічна частина

ГРАФІЧНА ЧАСТИНА

Розробка програмної системи управління групою будинків

Рисунок Д.1 – Слайд презентації 1

Розробка програмної системи управління групою будинків

Виконав:
студент групи 5ПІ-21б

Тимчук В.Л.

Керівник:
к.т.н., доц. каф. ПЗ

Хошаба О.М.

Рисунок Д.2 – Слайд презентації 2

Метою роботи є підвищення ефективності управління групою будинків за рахунок розробки програмної системи, що поєднує автоматизоване керування інженерною інфраструктурою, цифрову взаємодію з мешканцями, аналітичну обробку даних та засоби прогнозного обслуговування.

Об’єктом дослідження є процес цифрового управління інженерною інфраструктурою та взаємодії між мешканцями та адміністрацією у межах групи житлових будинків.

Предметом дослідження є методи, моделі та програмні засоби, що забезпечують інтегроване керування технічними та інформаційними підсистемами групи житлових будинків.

Рисунок Д.3 – Слайд презентації 3

Відповідно до поставленої мети виконані наступні задачі:

- провести аналітичний огляд предметної галузі програмних систем управління групою будинків;
- визначити критерії вибору інструментальних засобів для розробки програмних систем управління групою будинком;
- провести дослідження процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків;
- виконати проектування архітектурної моделі програмної системи управління групою житлових будинків;
- виконати проектування основних функціональних модулів системи;
- розробити програмний код для всіх програмних модулів системи;
- виконати тестування всіх програмних модулів програмного засобу.

Рисунок Д.4 – Слайд презентації 4

Актуальність теми дослідження полягає в наступному:

У сучасному урбанізованому середовищі, що характеризується інтенсивним розвитком житлових кварталів, особливого значення набуває ефективне управління групою будинків.

Так, зі зростанням чисельності населення та складністю інженерної інфраструктури житлових об'єктів, виникає необхідність у впровадженні програмних систем, які забезпечують автоматизоване керування, облік ресурсів, контроль технічного стану, інтерактивну взаємодію з мешканцями та високий рівень безпеки.

Таким чином, тема розробки програмної системи управління групою будинків є актуальною через потребу підвищення прозорості, ефективності та інтеграції цифрових технологій у сферу житлово-комунального господарства.
.

Рисунок Д.5 – Слайд презентації 5
Основні концепти та технології в управлінні групою будинків

Концепт / технологія	Опис функціональності	Рівень впровадження	Тип взаємодії
Smart Home / Building	Автоматизація освітлення, клімату, безпеки	Локальний	Мешканець ↔ система
IoT	Збір та передача даних з сенсорів	Локальний / централізований	Обладнання ↔ хаб ↔ сервер
SCADA	Централізоване керування технічними системами	Централізований	Адміністратор ↔ інженерні системи
BMS	Повне керування інфраструктурою будівель	Централізований	Інтерфейс ↔ всі підсистеми
Прогнозна аналітика	Виявлення відхилень, попередження про несправності	Централізований	Алгоритм ↔ база даних
Цифрова реєстрація подій	Прийом звернень мешканців, фіксація подій	Централізований	Користувач ↔ система
Веб-/мобільні додатки	Інтерфейс для мешканців та адміністраторів	Клієнтський	Користувач ↔ додаток

Рисунок Д.6 – Слайд презентації 6
Алгоритм обробки запиту користувача в системі з ролеорієнтованим доступом на основі UML-діаграми активностей

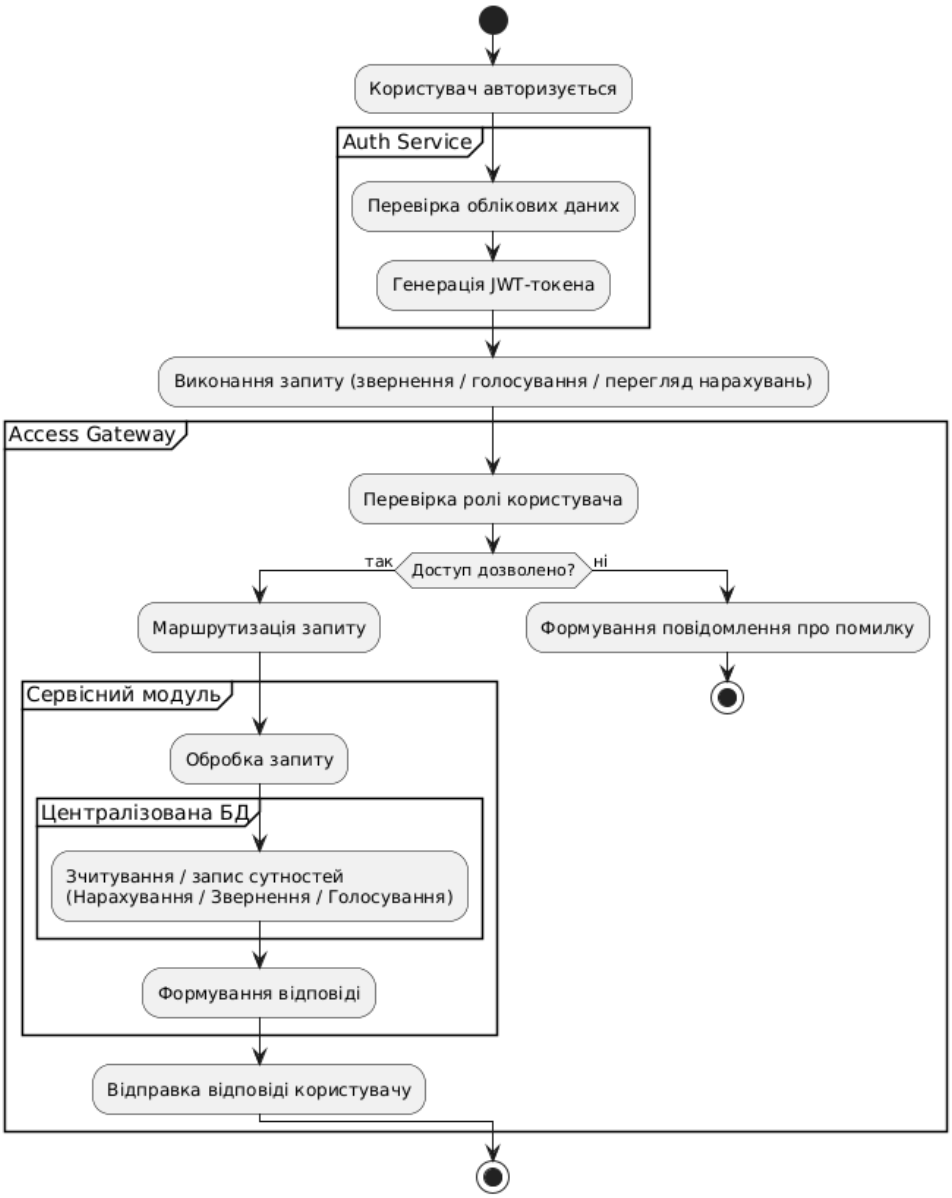

Рисунок Д.7 – Слайд презентації 7

UML-діаграма компонентів основних функціональних модулів системи

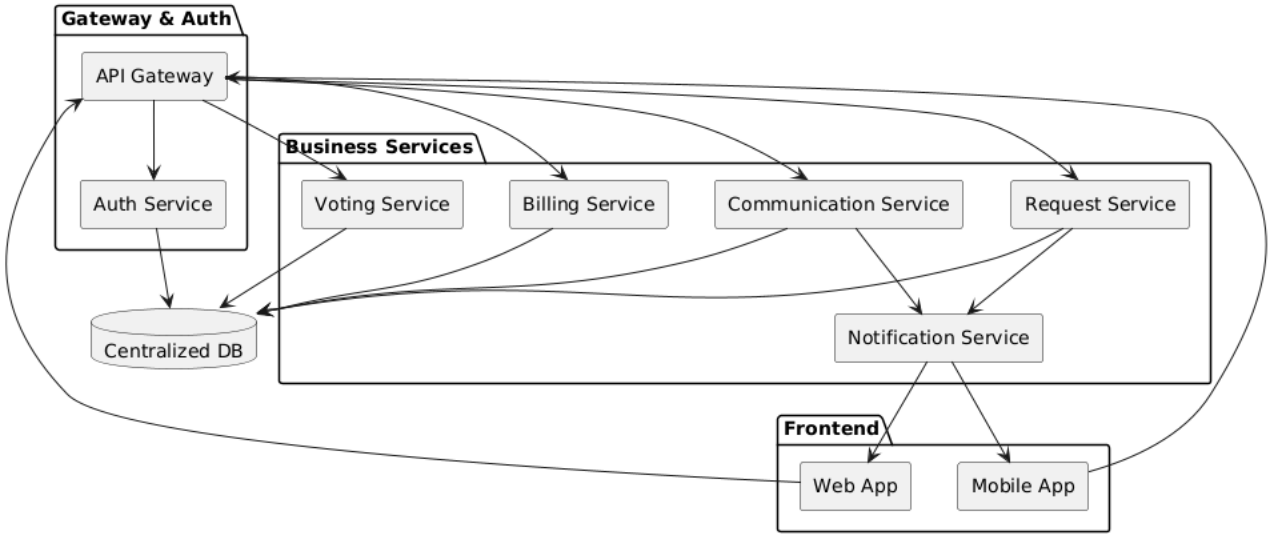

Рисунок Д.8 – Слайд презентації 8
UML-діаграма ієрархічної структури класів модуля Voting

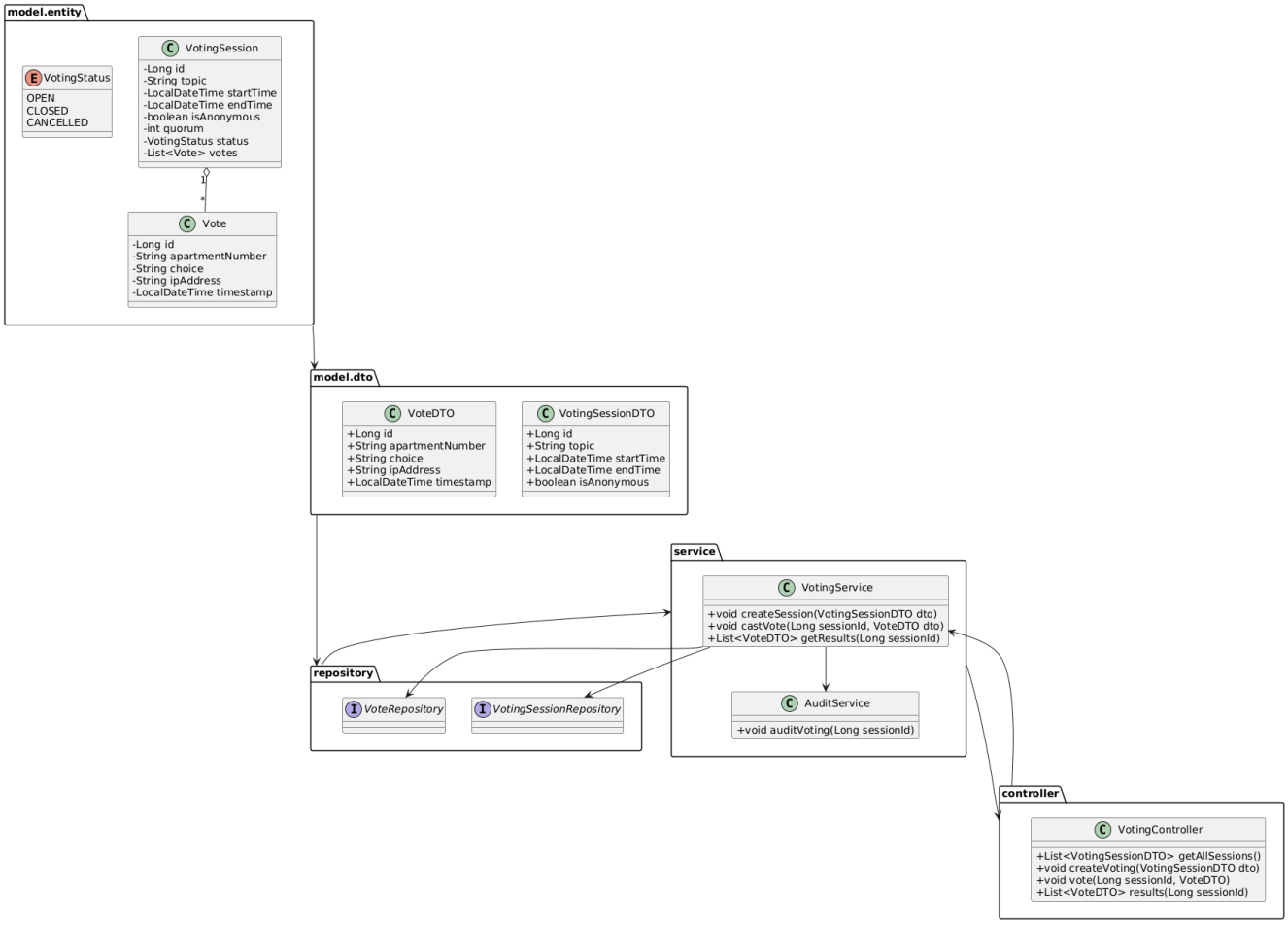

Рисунок Д.9 – Слайд презентації 9
Скріншот результатів тестування заповненної форми голосування

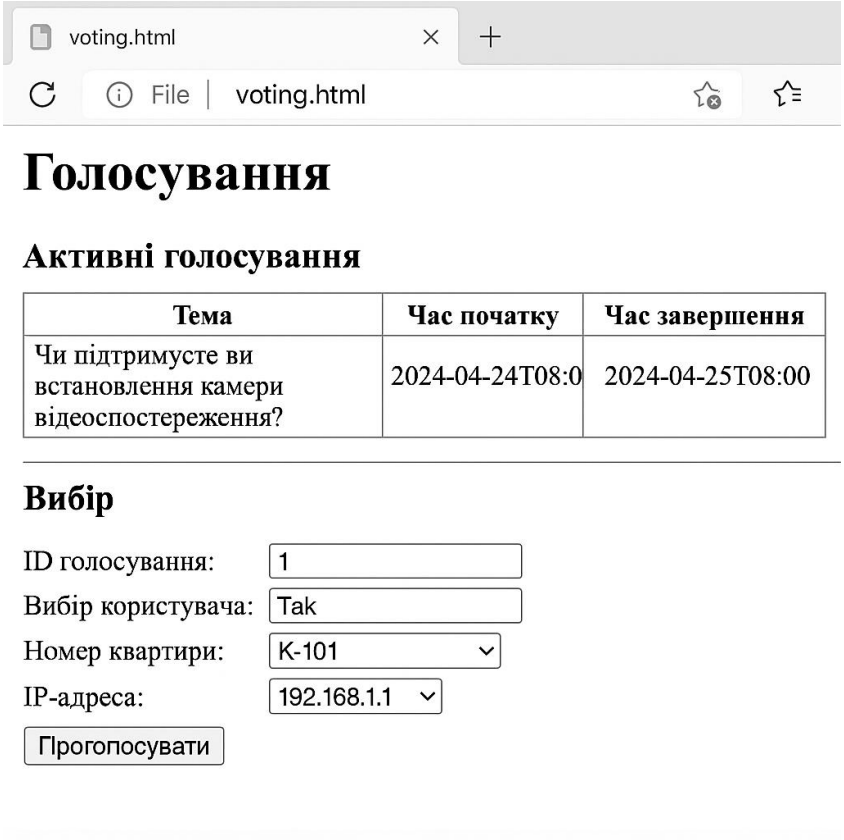

Рисунок Д.10 – Слайд презентації 10

Скріншот результатів тестування надсилання голосу та підтвердження участі

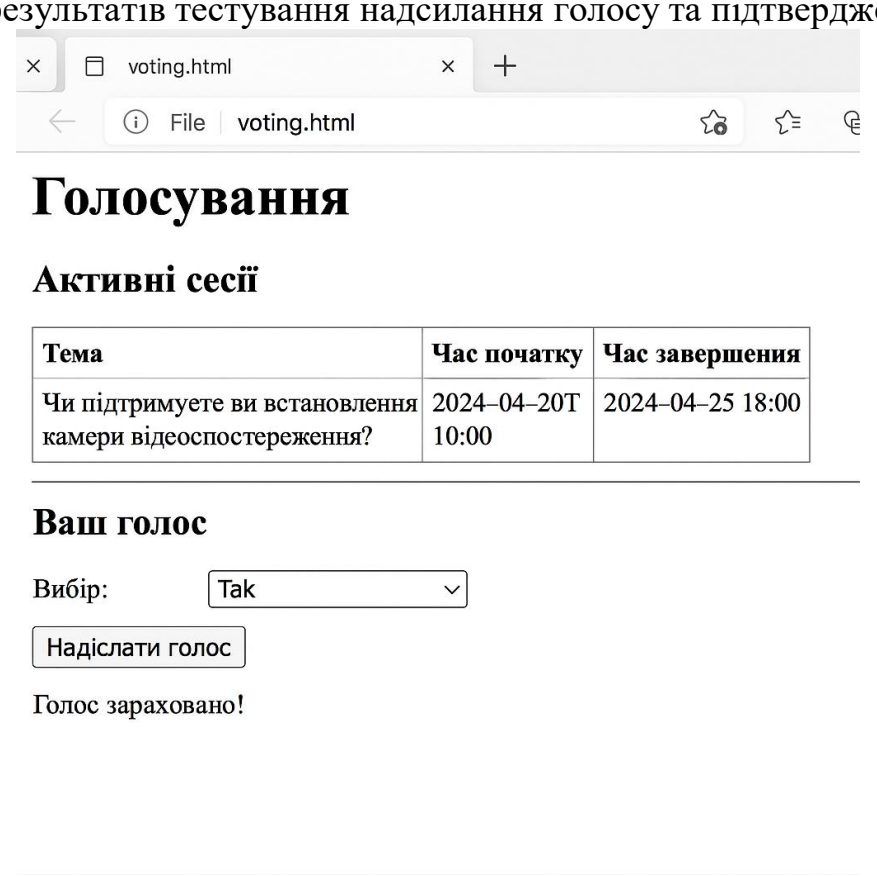

Рисунок Д.11 – Слайд презентації 11

Новизна отриманих результатів:

1. Запропоновано метод обробки подій у реальному часі для програмної системи управління групою житлових будинків, який враховує тип запиту користувача, пріоритет дії, та статус інфраструктурного ресурсу за допомогою WebSocket-комунікацій, що дозволило підвищити швидкодію оновлення інформації в системі на 32% порівняно з традиційною REST-архітектурою.
2. Запропоновано модель архітектури програмної системи управління групою житлових будинків, особливість якої полягає у поєднанні ролеорієнтованого доступу користувачів із централізованим обліком даних про нарахування, звернення, голосування та комунікації, що дає можливість підвищити ефективність управління та прозорість взаємодії між мешканцями й адміністрацією ОСББ.
3. Подальшого розвитку отримав метод побудови аналітичних панелей для житлового фонду у вебінтерфейсі, у якому, на відміну від існуючих, застосовано інтеграцію з внутрішнім сховищем статистичних даних у форматі JSON + TypeScript-звітності, що дозволило автоматизувати формування звітів для фінансових нарахувань, відгуків мешканців та ефективності рішень зборів з точністю до 29%.

Рисунок Д.12 – Слайд презентації 12

Висновки:

У бакалаврській кваліфікаційній роботі:

- проведено аналітичний огляд предметної галузі програмних систем управління групою будинків;
- визначено критерії вибору інструментальних засобів для розробки програмних систем управління групою будинком;
- проведено дослідження процесу обробки подій у реальному часі для програмної системи управління групою житлових будинків;

- виконано проектування архітектурної моделі програмної системи управління групою житлових будинків;
- виконано проектування основних функціональних модулів системи;
- розроблено програмний код для всіх програмних модулів системи;
- виконано тестування всіх програмних модулів програмного засобу.