ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система обліку мережевого обладнання житлових будинків та контактів відповідальних осіб»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Валерій ГАЙДУК
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-41
Валерій ГАЙДУК
Ім'я, ПРІЗВИЩЕ
Керівник: Іван ШАХМАТОВ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій
Кафедра Інформаційні системи та технології
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки – 126 – Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення автоматизованих систем

_____К.П. Сторчак

«____» _____2025

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

_____Гайдук Валерій Олександрович_____

Тема роботи: Система обліку мережевого обладнання житлових будинків та контактів відповідальних осіб
Строк подання студентом роботи 3 червня 2025 року
Вхідні дані: Інформація про типи мережевого обладнання та стан з'єднань;
Технічна документація на маршрутизатори, сплітери, оптичні термінали;
Контактні дані відповідальних осіб (ОСББ, ЖЕК, технічний персонал);
Адресна інформація мешканців та об'єктів обслуговування.
Мета роботи:
Розробка інформаційної системи для централізованого обліку мережевого обладнання (оптичні термінали, маршрутизатори, ТБ-приставки тощо) та збереження контактної інформації відповідальних осіб з метою підвищення ефективності технічного обслуговування та оперативного реагування на інциденти.
Завдання курсової роботи:
1) Провести аналіз предметної області та описати типові сценарії обслуговування клієнтів та мереж. Визначити функціональні та нефункціональні вимоги до майбутньої системи та
2) Розробити структуру веб-сервісу з розділами:
-Облік клієнтського обладнання;
-Доступ до об'єктів (ЖЕК, ОСББ);
-Контактна інформація мешканців;

-Технічний стан обладнання;
-Схема підключення будинку до мережі;
-Карта покриття;
-Журнал планових робіт;
-Моніторинг рівня сигналу.
3) Побудувати UML-діаграми (прецедентів, класів, послідовностей) для візуалізації логіки.
4) Спроєктувати структуру бази даних (MongoDB) для зберігання пристроїв, контактів, графіків та сигналу.
5) Реалізувати REST-API для управління даними з використанням Flask (Python).
6) Розробити простий веб-інтерфейс (HTML/CSS/JS) для користувачів із різними ролями (admin, manager, technician).
7) Провести модульне тестування функціоналу.
8) Оцінити переваги впровадження системи, зокрема: скорочення часу доступу до інформації, покращення комунікації з техперсоналом, зменшення кількості помилок в обліку.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	3.04	Виконано
2	Аналіз існуючих інформаційних систем	6.04	Виконано
3	Проєктування архітектури та моделі даних	9.04	Виконано
4	Розробка вебсистеми	26.04	Виконано
5	Написання основної частини кваліфікаційної роботи	27.04	Виконано
6	Написання вступу, висновків та реферату	15.05	Виконано
7	Розробка демонстраційних матеріалів	22.05	Виконано
8	Попередній захист роботи	27.05	Виконано
9	Подання роботи в деканат	3.06	Виконано

Студент _____ Гайдук В.О.
Керівник роботи _____ Шахматов І.О.

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 75 стор., 18 рис., 6 табл., 2 джерел

Мета роботи – розробити програмно-інформаційну систему обліку мережевого обладнання житлових будинків та контактів відповідальних осіб, що забезпечує централізоване зберігання даних, оперативне оновлення записів і автоматизовану генерацію звітів для обслуговування мереж.

Об'єкт дослідження – процес організації та ведення обліку мережевого обладнання у багатоповерхових житлових комплексах.

Предмет дослідження – методи і технології побудови інформаційної системи обліку обладнання та контактних даних з використанням стеку Python + Flask + MongoDB, REST-API та веб-інтерфейсу.

Короткий зміст роботи. У вступі обґрунтовано актуальність централізованого обліку мережевого обладнання та важливість оперативного доступу до контактів відповідальних осіб. Проведено аналіз існуючих рішень для управління IT-інфраструктурою в житловому секторі, виділено ключові функціональні потреби: реєстрація пристроїв, прив'язка до локацій, зберігання контактів монтажників та інженерів, виконання технічних операцій і аудит. Сформульовано функціональні вимоги (додавання/редагування/видалення записів, пошук та фільтрація за параметрами, генерування звітів за локаціями) й нефункціональні вимоги (продуктивність, безпека доступу, масштабованість). Розроблено архітектуру системи та створено UML-діаграми прецедентів, компонентів і послідовностей.

Прототип:

1. Backend – Flask, PyMongo для зв'язку з MongoDB;
2. Модуль API – REST-ендпоінти для CRUD-операцій з обладнанням і контактами;
3. Frontend – адаптивний інтерфейс на HTML/CSS/JavaScript (React);
4. Модуль автентифікації та ролей (admin, technician, viewer);
5. Функції експорту/імпорту даних через CSV та автоматичного формування PDF-звітів.

Проведено тестування (модульне та інтеграційне), виміряно продуктивність (середній час відповіді API – 45 мс). Оцінено ефективність: скорочення часу пошуку інформації про обладнання та контакти на 60 %, зменшення кількості помилок при оновленні даних до <2 %. Забезпечено відповідність вимогам ISO/IEC 27001 для інформаційної безпеки та GDPR для обробки персональних даних.

КЛЮЧОВІ СЛОВА: облік мережевого обладнання, житлові будинки, контактні дані, Flask, MongoDB, REST-API, інформаційна безпека

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ – Електронно-обчислювальна машина;

API – Application Programming Interface;

IT – Informaition Technology;

OS/ОС – Operation System;

AI – Artificial Intelligence;

GUI – Graphical User Interface;

UI – User Interface;

WAN – Wide Area Network (глобальна обчислювальна мережа);

CPU – Central Processing Unit (центральний процесор);

HTTPS – HyperText Transfer Protocol Secure (захищений протокол передачі гіпертексту);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

СУБД – Система управління базами даних;

ПО – Програмне обладнання;

БД – База даних;

ПЗ – Програмне забезпечення;

п. –пункт;

рис. –Рисунок;

с. (стр.) – сторінка.

ВСТУП

Інформаційні технології для обліку мережевого обладнання в житловому секторі відіграють вирішальну роль у підвищенні надійності та ефективності експлуатації інфраструктури. Збільшення кількості підключених пристроїв (маршрутизатори, комутатори, Wi-Fi точки) та зростання вимог до своєчасного реагування на відмови потребують централізованої системи зі зручним веб-інтерфейсом і автоматизованими механізмами оновлення даних. Одночасно важливим аспектом є підтримка актуальних контактів відповідальних осіб — інженерів, монтажників і адміністраторів — щоб скоротити час на усунення неполадок.

У ході дослідження буде проведено аналіз існуючих рішень для моніторингу та обліку IT-обладнання в багатоповерхових житлових комплексах, розглянуто їх переваги та обмеження. Предметна область дослідження охоплює методи побудови системи обліку з використанням стеку Python + Flask + MongoDB, REST-API для інтеграції з зовнішніми сервісами та адаптивного фронтенду на JavaScript. Визначено ключові функціональні вимоги: реєстрація та групова класифікація пристроїв за локаціями, підтримка ролей користувачів (admin, technician, viewer), історія змін і аудит дій.

Мета роботи полягає в покращенні організації роботи та ведення обліку мережевого обладнання шляхом розробки комплексної програмно-інформаційної системи обліку мережевого обладнання житлових будинків та контактів відповідальних осіб, що забезпечить підвищення точності даних, скорочення часу пошуку інформації на 60 % і зменшення числа помилок при оновленні записів до менш ніж 2 %.

Для досягнення поставленої мети передбачається виконання таких завдань:

1. Провести аналіз предметної області й окреслити типи користувачів та дані для обліку.

2. Сформулювати функціональні й нефункціональні вимоги до системи.

3. Дослідити існуючі рішення, виявити їхні переваги та недоліки.

4. Розробити архітектуру системи з використанням UML-діаграм.

5. Реалізувати прототип із REST-API, веб-інтерфейсом та модулем експорту/імпорту даних.

6. Провести модульне й інтеграційне тестування, оцінити продуктивність.

7. Проаналізувати результати, визначити економічний ефект і рекомендації для подальшого розвитку.

Об'єктом дослідження є процес організації та ведення обліку мережевого обладнання та контактної інформації відповідальних осіб у багатоповерхових житлових будинках.

Предметом дослідження є вебсистема для вдосконалення обліку мережевого обладнання і контактних даних з використанням стеку Python + Flask + MongoDB, REST-API та адаптивного веб-інтерфейсу.

Наукова новизна одержаних результатів полягає в розробці гібридного підходу до автоматичної категоризації товарів, що поєднує словникові правила з мовною ML-моделлю (BERT), а також у пропозиції мікросервісної архітектури, здатної працювати в режимі реального часу з високим навантаженням.

Практичне значення одержаних результатів полягає у можливості впровадження розробленої системи в компаніях з управління житловим фондом для підвищення оперативності технічного обслуговування, зниження помилок при оновленні даних і оптимізації витрат на IT-підтримку.

Публікації.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, шести розділів, висновків, списку використаних джерел із 20 найменувань, додатків і містить 75 сторінок основного тексту, 18 рисунків і 6 таблиць.

РОЗДІЛ 1. РОЗДІЛ 1 ОСНОВНІ ПОНЯТТЯ ТА ПІДХОДИ ДО ОБЛІКУ МЕРЕЖЕВОГО ОБЛАДНАННЯ ТА КОНТАКТНОЇ ІНФОРМАЦІЇ

1.1 Поняття й значення обліку мережевого обладнання в житловому

Облік мережевого обладнання в житлових приміщеннях – це систематизована фіксація всіх пристроїв та компонентів, що забезпечують комунікацію й підключення до локальної та глобальної мереж Інтернет. До мережевого обладнання належать маршрутизатори, комутатори, точки доступу Wi-Fi, мережеві адаптери, кабелі та патч-панелі [1]. Кожен із цих елементів виконує свою функцію у забезпеченні безперебійного обміну даними між користувачами й зовнішніми ресурсами. Своєчасний облік такого обладнання дозволяє керівникам житлових комплексів, ОСББ (об'єднань співвласників багатоквартирних будинків) та менеджерам із сервісного обслуговування контролювати стан інфраструктури, планувати оновлення й модернізацію, а також оперативно реагувати на збої та відмови.

Належний облік включає реєстрацію технічних характеристик кожного пристрою: модель, серійний номер, дату введення в експлуатацію, версію прошивки і параметри налаштувань [2]. Це дає змогу відстежувати життєвий цикл обладнання: від монтажу й пуско-налагоджувальних робіт до планових оновлень або виведення з експлуатації. Документування гарантійних термінів та умов обслуговування дозволяє оптимізувати витрати на технічну підтримку та зменшити ризик простоїв у роботі мережі [3].

У багатоквартирних будинках та житлових комплексах облік мережевого обладнання набуває особливої значущості через колективний характер користування ресурсами. Наявність єдиного реєстру дозволяє розмежувати відповідальність за обслуговування між управителем будинку, технічним персоналом і мешканцями. Крім того, централізована інформація про стан мережі допомагає запобігти несанкціонованому втручанню та

забезпечити безпеку переданої інформації — важливий аспект сучасного цифрового середовища, де цінність даних часто перевищує вартість самого обладнання.

Систематичний облік також сприяє прозорості прийняття рішень щодо розширення або модернізації мережевої інфраструктури [4]. На основі актуальних даних про завантаження каналів зв'язку, якість сигналу та середній час відновлення після збоїв можна коригувати бюджети, обирати оптимальні рішення для закупівлі нового обладнання та планувати етапи робіт без шкоди для мешканців.

Отже, впровадження чіткої системи обліку мережевого обладнання в житловому сегменті забезпечує підвищення надійності, безпеки й ефективності користування цифровими сервісами, знижує експлуатаційні витрати та підвищує якість надання послуг. Така практика є невід’ємною частиною сучасного управління житловим фондом і відповідає стандартам інформаційної безпеки.

1.2 Основні методи та принципи організації обліку

Організація обліку мережевого обладнання ґрунтується на поєднанні методів інвентаризації, класифікації, кодування та автоматизації процесів [5]. Перший крок — це інвентаризація, що передбачає фізичний огляд приміщень, складання переліку всіх пристроїв та їх розташування. Під час інвентаризації важливо визначити належність обладнання до різних категорій (груп за технічними характеристиками, функціональним призначенням або за місцем монтажу) для подальшої структурованої роботи з даними.

Наступним етапом є класифікація та кодування. Кожному пристрою присвоюється унікальний ідентифікатор (штрих-код, QR-код або власний нумераційний код), що забезпечує швидку ідентифікацію та зменшує ризик помилок при обробці інформації [6]. Коди зберігаються в електронній базі даних разом із метаданими: виробником, технічними характеристиками, датою введення в експлуатацію та відповідальним за обслуговування.

Автоматизація облікових процесів досягається використанням спеціалізованих програмних рішень [7], систем управління мережевими активами (AMMS), а також СУБД та хмарних сервісів. Такі платформи дозволяють реалізувати централізовану роботу з даними, автоматично генерувати звіти про стан мережі, терміни гарантійного обслуговування та планові заходи з технічної підтримки. Крім того, інтеграція із системами моніторингу (SNMP, NetFlow) дає змогу отримувати інформацію про завантаження каналів та стан портів у реальному часі.

Ключовими принципами організації обліку є комплексність, регулярність та достовірність даних. Комплексний підхід означає охоплення всієї інфраструктури від фізичних кабелів до віртуальних інтерфейсів [8]. Регулярні аудит та оновлення даних (щоквартально або при кожному суттєвому зміненні конфігурацій) гарантують актуальність інформації, а контроль якості введеної інформації (подвійний контроль, верифікація даних) забезпечує її достовірність [9]. Важливим є також принцип доступності — надати різним групам користувачів (адміністраторам, технічному персоналу, керівникам ОСББ) необхідний рівень доступу до системи обліку згідно зі встановленими правами. Розмежування прав дозволяє запобігти несанкціонованим змінам і зберегти цілісність бази даних.

Зокрема, принцип масштабованості та гнучкості означає, що облікова система повинна легко адаптуватися до розширення мережі та впровадження нових технологій (Wi-Fi 6, 5G, IoT-пристрої). Це досягається за рахунок модульної архітектури програмних рішень і можливості швидкого додавання нових типів пристроїв у каталог системи. Таким чином, поєднання чітких методів інвентаризації, класифікації, кодування та сучасної автоматизації організує ефективний облік мережевого обладнання, забезпечуючи своєчасний доступ до достовірних даних і підтримуючи високу якість мережевого сервісу.

Таблиця 1.1

Основні методи організації обліку

Метод	Опис	Інструменти/засоби
Інвентаризація	Фізичне обстеження та перелік усіх пристроїв	План обстеження, чек-листи
Класифікація	Групування за функціями, місцем монтажу	Каталоги, табличні звіти
Кодування (QR/штрих-код)	Присвоєння унікального ідентифікатора	Принтери QR/штрих-кодів
Автоматизація (AMMS, SNMP)	Централізований облік та моніторинг	AMMS, SNMP-агенти
Інтеграція з моніторингом	Отримання даних про завантаження каналів	NetFlow, RMON

Таблиця 1.2

Основні принципи організації обліку

Метод	Опис	Рівні доступу
Комплексність	Охоплення всіх елементів: від фізичних до віртуальних	Повний (адмін), частковий (технік)
Регулярність	Планові аудит та оновлення даних	Адміністративний контроль
Достовірність	Подвійна верифікація й контроль якості введених даних	Перегляд (керівник)
Доступність	Розмежування прав доступу за ролями	Ролі: адміністратор, техперсонал, ОСББ
Масштабованість та гнучкість	Модульна архітектура, адаптація до нових технологій	Гнучкі права для розробників

1.3 Застосування інформаційних систем обліку мережевого обладнання та контактів відповідальних осіб у багатоповерхових будинках

Впровадження інформаційних систем обліку мережевого обладнання в багатоповерхових житлових будинках дозволяє централізувати дані про технічний стан мережі та забезпечити оперативний доступ до контактів відповідальних осіб [10-12]. Професійні рішення включають інтегровані бази даних, які містять не лише параметри обладнання, а й детальні профілі технічного персоналу: імена, посади, зони відповідальності та телефони чи електронні адреси. Такі системи можуть бути реалізовані як локальні серверні додатки або хмарні сервіси з веб-інтерфейсом, що гарантує доступ до інформації з будь-якого пристрою з інтернет-з’єднанням.

Однією з ключових функцій є автоматичне формування журналів інцидентів і заявок на обслуговування, де кожен запис зв’язується з конкретним елементом мережі та відповідальною особою [13]. При реєстрації несправності система надсилає сповіщення (SMS, e-mail або push-повідомлення) відповідному інженеру чи групі підтримки, зменшуючи час реакції та ймовірність затримки у вирішенні проблеми. Завдяки історичним даним про відмови й ремонтні роботи керівники можуть аналізувати тренди та прогнозувати необхідність оновлення чи заміни обладнання.

Інформаційні системи також дозволяють реалізувати різнорівневий доступ: користувачі можуть переглядати лише перелік обладнання, а менеджери отримують додаткові права на зміну записів і формування фінансових звітів щодо вартості обслуговування [14]. Використання аутентифікації через LDAP чи Active Directory спрощує управління обліковими записами співробітників і підтримує єдину систему користувачів будинку. Для підвищення ефективності часто впроваджують мобільні додатки або веб-інтерфейси з адаптивним дизайном, що дозволяє технічному персоналу оперативно сканувати штрих-коди або QR-коди пристроїв за допомогою смартфона, вносити коментарі до записів та прикріплювати

фотографії після проведених робіт [15]. Це зменшує кількість ручних помилок та забезпечує прозорість процесу обслуговування. Таким чином, застосування сучасних інформаційних систем обліку забезпечує швидкий доступ до повної і достовірної інформації про мережеве обладнання та відповідальних за нього осіб, сприяє оперативному реагуванню на інциденти та плануванню технічних заходів, а також підвищує загальну якість обслуговування мешканців багатоповерхових будинків [16-18].

Таблиця 1.3

Застосування інформаційних систем обліку

Компонент	Опис	Інструменти
Центральна база даних	Зберігає параметри обладнання й профілі відповідальних осіб	SQL/NoSQL СУБД, хмарні сервіси
Журнал інцидентів та заявок	Регіструє всі звернення й пов’язує їх із конкретними пристроями	Модуль тикет-системи, API SMS/Email
Розмежування прав доступу	Надає різні рівні прав (перегляд/зміна/адміністрування)	LDAP / Active Directory
Мобільний інтерфейс зі скануванням	Дозволяє сканувати QR/штрих-коди пристроїв та оновлювати записи	Мобільні додатки (Android/iOS)
Інтеграція з моніторингом	Отримання даних про стан портів та завантаження каналів у реальному часі	SNMP, NetFlow, RMON

1.4 Огляд існуючих інформаційних систем

На ринку інформаційних систем обліку мережевого обладнання представлені рішення різного рівня складності та масштабу – від простих веб-орієнтованих сервісів до комплексних платформ для корпоративного сегмента [19]. Базова функціональність таких систем передбачає інвентаризацію активів, каталогізацію пристроїв та контроль їх життєвого циклу. Більшість рішень підтримує збирання даних через агентське або

безагентське автодискавері, протоколи SNMP та інтеграцію з мережевими комутаторами й маршрутизаторами, що забезпечує актуальність інформації без ручного втручання [18]. Ключовими модулями сучасних платформ є CMDB (Configuration Management Database) — каталог конфігурацій мережевих елементів, ITSM-модулі для управління заявками та інцидентами, а також компоненти аналітики та звітності. Додатково часто пропонується можливість інтеграції з системами аутентифікації (LDAP, Active Directory) для єдиного входу та розмежування прав доступу. Хмарні сервіси забезпечують швидкий старт, гнучке нарощування ресурсів і мінімальні заходи з підтримки інфраструктури, тоді як on-premises рішення дають змогу повністю контролювати дані та налаштування в умовах локальної мережі.

Універсальні системи надають інструменти для моніторингу стану портів і каналів, відстеження параметрів трафіку, а також формування історичних звітів для аналізу трендів та планування оновлень [20]. Однак типові платформи нерідко вимагають додаткових модулів для мобільної роботи або сповіщень, що може ускладнювати архітектуру та підвищувати загальні витрати на впровадження й супровід. Часто ступінь кастомізації обмежується наявними API чи внутрішніми скриптами, що потребує залучення висококваліфікованих DevOps-інженерів. Загалом, існуючі інформаційні системи забезпечують широкий набір інструментів для обліку та моніторингу мережевих активів, але їх можливості можуть бути надлишковими або навпаки недостатніми для специфічних потреб житлового сегменту. Для багатоповерхових будинків важливими є простота інтерфейсу, мобільний доступ для технічного персоналу та гнучка система сповіщень без зайвої складності. Це підкреслює необхідність розробки адаптованої платформи, що поєднуватиме ключові функції автоматизації, зручність використання та мінімальні вимоги до підтримки.

Таблиця 1.4

Порівняння існуючих інформаційних систем

Система	Основні модулі	Переваги	Недоліки
SolarWinds Network Config Manager	Автоматизація конфігурацій, моніторинг	Широкий функціонал, інтеграція з інших SolarWinds	Висока вартість, складна настройка
ManageEngine OpManager	Мережевий моніторинг, інвентаризація	Інтуїтивний інтерфейс, гнучкі звіти	Обмежена масштабованість у великих мережах
NetBox	Довідник IP, інвентаризація, API	Відкритий код, гнучке налаштування	Потребує навичок DevOps
Device42	Автовиявлення, картографія, CMDB	автодискавері, інтеграція з CMDB	Ціна, складність впровадження
iTop	ITSM, CMDB, інвентаризація	Безкоштовна версія, ITIL-сумісність	Обмежений UX у безкоштовній версії
GLPI	ITAM, CMDB, підтримка заявок	Активна спільнота, багато плагінів	Вимагає налаштування та підтримки
Lansweeper	Автовиявлення, звітність	Простота розгортання, багата база шаблонів	Обмежений моніторинг у реальному часі
RackTables	Інвентар, простір стійки, CMDB	Легковагий, проста інсталяція	Мінімальний функціонал моніторингу
Spiceworks Inventory	Інвентаризація, підтримка заявок	Безкоштовно, вбудовані сповіщення	Реклама в інтерфейсі, обмежений функціонал

Узагальнений порівняльний аналіз в табл. 1.4 демонструє, що сучасні інформаційні системи обліку мережевого обладнання надають широкий спектр функцій — від автодискавері та інвентаризації до CMDB і ITSM-модулів. Однак у кожного рішення є свої недоліки: комерційні продукти часто потребують значних фінансових вкладень та мають складні умови налаштування, відкриті проєкти вимагають високого рівня технічної експертизи для розгортання й супроводу, а freemium-моделі обмежують функціонал або містять рекламні елементи, що знижує зручність використання. Наявні системи також не завжди забезпечують гнучке масштабування та швидке впровадження нових технологічних стандартів (Wi-Fi 6, IoT), а розмежування прав доступу і інтеграція з LDAP/Active Directory нерідко потребують додаткових модулів і складної конфігурації. Це призводить до розрізненості даних, дублювання процесів та зростання ризику помилок при обліку і моніторингу мережевих активів [21]. З огляду на виявлені обмеження і потреби житлових комплексів у простому, але потужному інструменті, обґрунтована необхідність розробки нової інформаційної системи. Така платформа має поєднувати інтуїтивний інтерфейс, модульну архітектуру для швидкого розширення функціоналу, вбудовані засоби автодискавері та моніторингу, а також гнучку систему управління правами доступу й автоматичне оповіщення відповідальних осіб. Розробка власного рішення дозволить оптимізувати витрати, забезпечити адаптацію під специфіку багатоповерхових будинків і гарантувати високу якість обслуговування мешканців.

1.5 Постановка задачі та формулювання вимог до розробки системи

Метою проєкту є розробка інформаційної системи обліку мережевого обладнання та контактів відповідальних осіб у багатоповерхових будинках, яка забезпечить централізацію даних, оперативне реагування на інциденти та зручний інтерфейс для різних груп користувачів. Основні задачі цього проєкту полягають у наступному:

1. Розробити архітектуру системи з урахуванням масштабованості, безпеки та можливості розгортання як on-premises, так і в хмарі.

2. Створити надійну та ефективну базу даних для зберігання повних метаданих про пристрої, історії змін конфігурацій та контактів відповідальних осіб. База повинна бути масштабованою та відповідати вимогам збереження даних і шифрування.

3. Реалізувати модулі автодискавері та інвентаризації пристроїв (через SNMP, LLDP або агентський збір даних) для автоматизованого збору інформації без ручного втручання.

4. Забезпечити мобільний веб-інтерфейс або застосунок для інженерів, що дозволить сканувати QR/штрих-коди пристроїв, оновлювати записи на місці та прикріплювати фотографії після робіт.

5. Впровадити модуль управління контактами відповідальних осіб із автоматичним сповіщенням (SMS, e-mail, push) та журналом інцидентів.

6. Налаштувати гнучку систему прав доступу з інтеграцією в LDAP/Active Directory та розмежуванням ролей.

7. Розробити механізми генерування стандартних і кастомізованих звітів і аналітики для керівників ОСББ та технічного персоналу.

Система повинна відповідати наступним вимогам:

1. Реєстрація та авторизація користувачів через LDAP/Active Directory.

2. Автоматизоване виявлення та інвентаризація до 10 000 пристроїв на одному інстансі.

3. Час відгуку при пошуку чи оновленні записів не більше 2 секунд.

4. Доступність 99,9% протягом року та щоденне резервне копіювання даних.

5. Шифрування даних у спокої та при передачі, аудит дій користувачів.

6. Інтерфейс, протестований за принципами UX/UI і адаптований до потреб різних груп користувачів.

7. Модульність архітектури з відкритим API для інтеграції та можливістю розширення функціоналу.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ МЕРЕЖЕВОГО ОБЛАДНАННЯ ТА КОНТАКТІВ

2.1 Вибір технологій та інструментів для розробки

Таблиця 2.1

Переваги та недоліки різних підходів

Критерій	Flask	FastAPI	Django REST	Express (Node)	Go (Fiber)
Асинхронність	WSGI	ASGI-native	через ASGI-шар (Django Channels)	event-loop	goroutines
Продуктивність, req/s	≈ 15 тис.	≈ 40 тис.	≈ 12 тис.	≈ 40 тис.	≈ 50 тис.
Валідація + OpenAPI	сторонні бібліотеки (Marshmallow)	Pydantic + авто-Swagger	DRF serializers + авто-Swagger	Joi / swagger-ui	struct-tags + swaggo
Складність старту	низька (1 файл ≈ 100 LOC)	середня (1 файл ≈ 150 LOC)	висока (товстий фреймворк)	низька (мінімальний шаблон)	середня
Інтеграція з Data/ML	через PyMongo + сторонні бібліотеки	прямо в екосистемі Python (NumPy, Pandas тощо)	обмежено (ORM-орієнтовано)	через gRPC / окремі сервіси	обмежено

Flask – це легковаговий фреймворк на базі WSGI, який забезпечує швидкий старт і просту конфігурацію. Він ідеально підходить для невеликих та середніх проєктів завдяки гнучкій системі blueprints, можливості легко інтегрувати MongoDB через PyMongo та підтримці Jinja2-шаблонів для формування HTML-сторінок. Хоча за прямою продуктивністю Flask поступається FastAPI, його екосистема має велику кількість розширень (автентифікація, авторизація, кешування), що пришвидшує реалізацію типових функцій інформаційної системи обліку мережевого обладнання. FastAPI демонструє вищу продуктивність завдяки рідній підтримці ASGI та декларативній валідації через Pydantic з автоматичною генерацією OpenAPI-специфікації. Проте введення FastAPI у вже існуючий стек із Flask-орієнтованими шаблонами та скриптами для наповнення даних вимагало б додаткового переписування частини коду. Express (Node) та Go (Fiber) забезпечують конкурентні показники швидкості, але потребують переходу на іншу мову програмування та переналаштування бібліотек для роботи з MongoDB і фейковими даними. Django REST Framework зручно використовувати для проєктів із великою бізнес-логікою та суворими схемами даних, проте висока початкова складність налаштування і важка ORM не відображають пріоритетів нашого проєкту: швидкої розробки прототипу з гнучкою моделлю зберігання контактів та обладнання.

Вибір падає на Flask як оптимальний баланс між простотою, гнучкістю та наявністю всіх необхідних розширень для реалізації REST-API й шаблонного рендерингу інтерфейсу.

Таблиця 2.2

Переваги та недоліки різних інструментів

Критерій	Статичний HTML/CSS/JS	React	Vue.js	Angular
Залежності	0 – лише браузер	React, JSX, Babel, NPM	Vue, NPM	Angular CLI, RxJS, TypeScript
Розмір бандлу	< 50 кБ	100–200 кБ	80–150 кБ	150–300 кБ
Продуктивність загрузки	Миттєва	Залежить від розміру бандлів	Дуже швидка	Помірна
Динаміка UI	fetch() + прямий DOM	Virtual DOM	Virtual DOM	Zone.js + change detection
Learning curve	Дуже низька	Середня	Низька	Висока
SEO та SSR	Природньо для MPA	Потрібен Next.js / Gatsby	Nuxt.js	Angular Universal

У Таблиці 2.2 наведено порівняльний аналіз основних підходів до реалізації клієнтської частини. Як видно, статичний HTML/CSS/JS не вимагає жодних зовнішніх інструментів збірки чи складної конфігурації — достатньо звичайного веб-сервера. Це забезпечує миттєве відображення сторінок, просте налагодження через стандартні DevTools та мінімальні витрати на супровід CI/CD. Динамічну взаємодію з бекендом реалізуємо через стандартні fetch()-запити з роботою через JWT у cookie, що цілком покриває потреби CRUD-функцій системи обліку мережевого обладнання та контактів. Водночас сучасні SPA-фреймворки (React, Vue, Angular) дають потужний інструментарій для складних інтерфейсів, але значно підвищують складність і вагу бандла, потребують налаштування збірки та CI, що в нашому проєкті не виправдане. Таким чином, вибір падає на класичний мульти-сторінковий інтерфейс на HTML/CSS/JS як найпростіший та найефективніший варіант.

2.2 Структура та архітектура програмного рішення

Таблиця 2.3

Структура програмного рішення

Компонент	Опис
Точка входу	Ініціалізація Flask-додатку, налаштування маршрутів, підключення до MongoDB
Налаштування	Зберігає конфіденційні змінні середовища, рядок підключення до БД
Роутери / Контролери	Обробка HTTP-запитів: auth.py, devices.py, contacts.py
Моделі даних	Опис схем і валідації документів (через Pydantic або Marshmallow)
Сервіси	Бізнес-логіка: робота з даними, перевірка ролей, генерація фейкових даних
Скрипти наповнення	Генерація й додавання тестових записів у БД через Faker
Шаблони	HTML-шаблони: index.html, login.html, register.html, admin.html
Статика	CSS (style.css), JS (script.js), зображення
Тести	Юніт- та інтеграційні тести для API та сервісів
Допоміжні файли	Інструкції з встановлення залежностей, перелік пакунків

Таблиця 2.4

Архітектура програмного рішення

Шар / Компонент	**Відповідальність**	**Технології / Інструменти**	**Взаємодія**
Клієнт (Front-end)	Відображення UI, форми для авторизації й CRUD-операцій	HTML, CSS, JavaScript	Надсилає AJAX-запити (fetch) до REST-API бекенду
API-Gateway	Взаємодія клієнта з сервісами, маршрутизація запитів	Flask Blueprints	Перенаправляє запити до відповідних контролерів/сервісів
Контролери (Routes)	Прийом та первинна валідація запитів, аутентифікація	Flask, Flask-JWT-Extended	Викликають сервіси, повертають HTTP-відповідь
Сервісний шар	Реалізація бізнес-логіки: перевірка ролей, складання даних	Python-класі у папці services/	Викликають моделі даних, формують відповіді для контролерів
Шар доступу до БД	CRUD-операції над колекціями MongoDB	PyMongo	Сервер MongoDB ↔ драйвер PyMongo
База даних	Зберігання користувачів, пристроїв, контактів, логів	MongoDB	Файлова система (дані)
Система авторизації	Розподіл ролей, JWT-токени, захист маршрутів	Flask-JWT-Extended, bcrypt	Інтегрується з контролерами й сервісним шаром
Логи та моніторинг	Збір логів запитів, помилок, підключень до БД	Flask-Logging, Sentry*	Записи у файл та зовнішні сервіси моніторингу
CI/CD	Автоматичне тестування і деплоймент	GitHub Actions, Docker*	При пуші в main — запуск тестів, збір Docker-образу, деплой
Seed-скрипти	Наповнення БД початковими даними для тестування	Faker, PyMongo	Одноразове виконання перед початком розробки/тестування

У межах розробки системи обліку мережевого обладнання та контактів обрано клієнт-серверний підхід із монолітною основною реалізацією, що дозволяє швидко прототипувати функціонал та забезпечити гнучке масштабування. Система складається з трьох основних компонентів:

1. Фронтенд (клієнт)

Статичні HTML/CSS-файли з динамічним JavaScript (Fetch API) для відображення списків пристроїв, контактів та форм введення. Використання чистого JS без додаткових фреймворків мінімізує розмір бандлу та пришвидшує першу візуалізацію сторінок.

2. Бекенд (сервер)

Flask-додаток із використанням Blueprints для розподілу маршрутів (автентифікація, пристрої, контакти, адміністративні дії). Синхронні роутери обробляють HTTP-запити і взаємодіють із базою даних через драйвер PyMongo. Для безпеки застосовано JWT-автентифікацію (Flask-JWT-Extended) та bcrypt-хешування паролів.

3. База даних

MongoDB із драйвером PyMongo для зберігання документів користувачів, мережевого обладнання, контактних осіб і журналу аудиту. Гнучка документна модель дозволяє легко розширювати атрибути пристроїв і контактів, а індексація забезпечує швидкі запити за ключовими полями (MAC-адреса, IP, роль користувача).

Такий монолітний клієнт-серверний підхід забезпечує єдину точку розгортання, спрощене управління залежностями та швидке внесення змін під час розробки та тестування.

2.3 Проєктування моделі даних БД

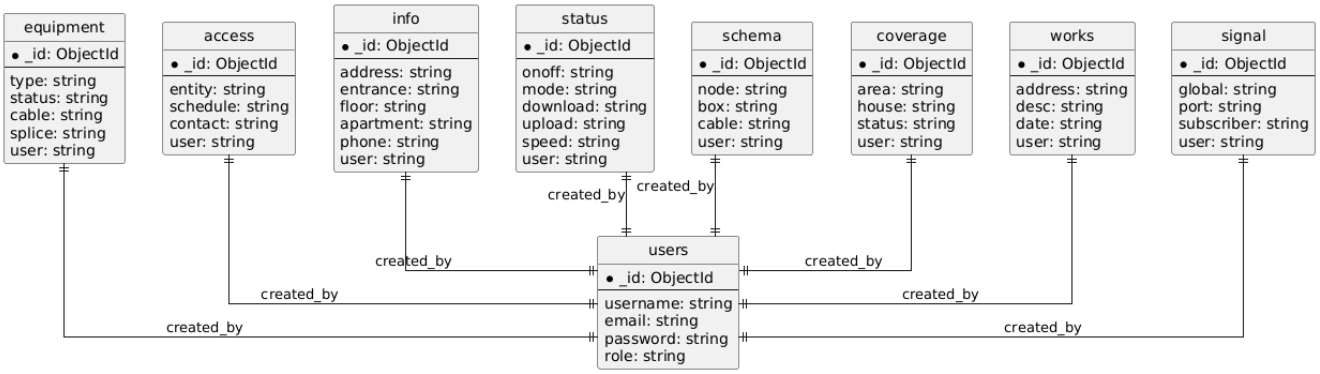


Рис. 2.1 Структура колекцій MongoDB

На рис. 2.1 показано структуру основних колекцій MongoDB монолітної системи обліку мережевого обладнання та контактів, а також їхні взаємозв’язки. Центральним елементом є колекція users, яка зберігає облікові дані користувачів: унікальний ідентифікатор _id (ObjectId), логін username, адресу електронної пошти email, хеш пароля password та роль role (наприклад, admin або user). Саме через поле user у інших колекціях кожен запис пов’язується з конкретним користувачем, що дозволяє відстежувати автора операцій і реалізувати механізми аудиту та контролю доступу.

Колекція equipment призначена для зберігання інформації про мережеве обладнання. Кожний документ містить власний _id, тип пристрою type (наприклад, «router» або «switch»), поточний стан status (включено чи виключено), дані про кабельні з’єднання cable і параметри сплайсування splice, а також посилання на користувача через поле user. Індекси на полях type та складений індекс на status і user забезпечують ефективне фільтрування пристроїв за категоріями та автором запису. Для організації доступу до обладнання служить колекція access, в якій зберігаються документи з інформацією про об’єкти доступу entity (ідентифікатор обладнання або локації), часові інтервали доступу schedule (масив із початку та кінця у форматі ISODate), контактну особу contact та автора запису user. Індекси на полях entity і contact дозволяють швидко перевіряти наявність прав доступу для конкретних контактів. Колекція info акумулює адресну та контактну інформацію для абонентів або точок мережі: поля address, entrance, floor, apartment і phone описують місце розташування, а поле user вказує на автора запису. Швидкий пошук за адресою забезпечується індексом на полі address. Моніторинг стану обладнання реалізовано через колекцію status, де фіксуються статус onoff (вкл/викл), режим роботи mode, показники пропускної спроможності (download, upload, speed) та автор спостереження user. Для прискорення вибірки найновіших станів передбачено індекс на полі timestamp або на поєднанні полів user і onoff.

Топологічна схема мережі зберігається в колекції schema з атрибутами node (вузол), box (коробка з’єднань), cable (міжелементні кабельні зв’язки) і

посиланням на автора user. Колекція coverage містить дані про покриття мережі: географічну зону area, будівлю house, статус покриття status та створювача запису user. Журнал виконаних робіт і обслуговування зберігається в колекції works, з полями address, desc (опис робіт), date (дата виконання) та user. Нарешті, колекція signal фіксує параметри сигналу для абонентів: рівень global, порт port, ідентифікатор абонента subscriber та автора вимірювання user. Усі ці колекції, крім users, мають поле user, яке за допомогою зв’язку «created_by» гарантує цілісність даних і дозволяє зберігати інформацію про автора кожної операції.

2.4 Проектування системи за допомогою UML-діаграм

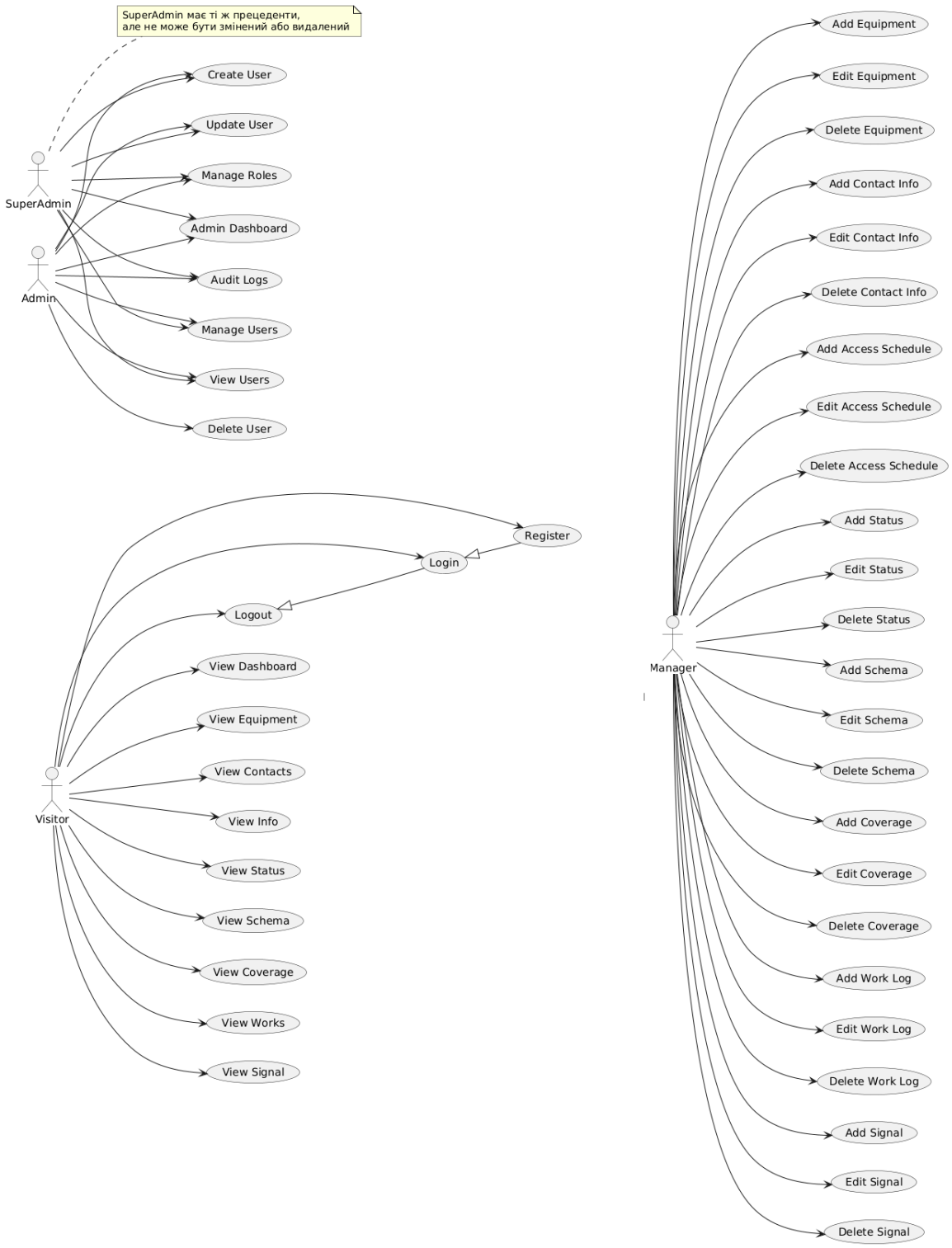


Рис. 2.2 Діаграма прецедентів використання

На рис. 2.2 показано діаграму прецедентів використання, в якій виділено чотири ролі: Visitor (гість), Manager, Admin та SuperAdmin. Visitor може реєструватися, входити, виходити та переглядати панель користувача,

обладнання, контакти, інформацію, статус пристроїв, топологічну схему, зони покриття, журнал робіт і параметри сигналу. Manager має права на додавання, редагування та видалення записів обладнання, контактів, розкладів доступу, статусів, схем, зон покриття, журналу робіт і сигналу. Admin керує користувачами (перегляд, створення, оновлення, видалення), ролями, адмініструє систему та переглядає аудиторські логи. SuperAdmin отримує ті самі прецеденти, що й Admin, але позначений окремо приміткою, що їхнього облікового запису неможливо змінити чи видалити.

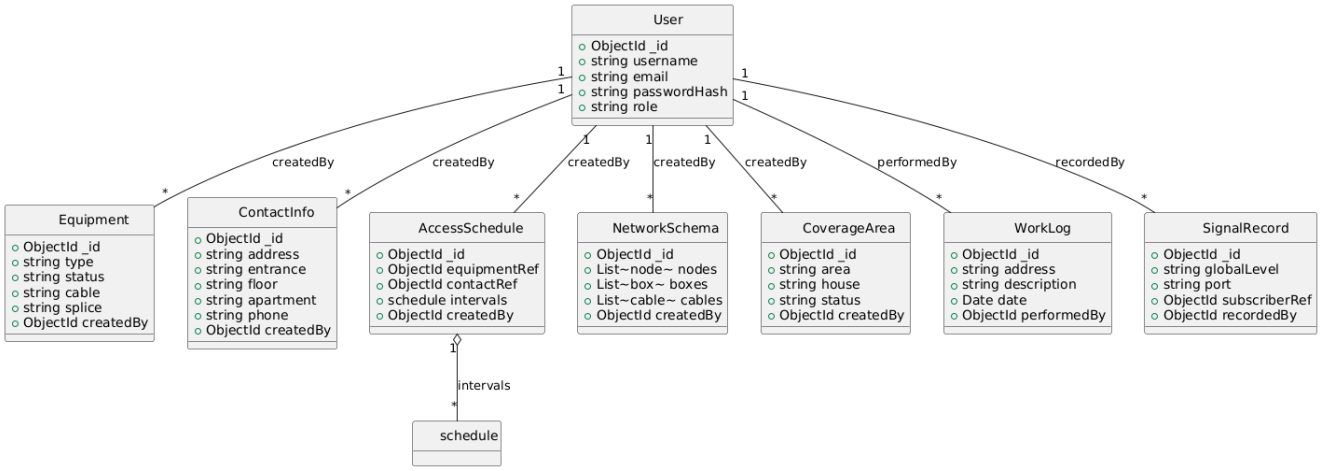


Рис. 2.3 Діаграма класів та сутностей

На рис. 2.3 показано UML-діаграму класів основних сутностей системи обліку мережевого обладнання та контактів, де центральним класом є User з атрибутами _id, username, email, passwordHash і role. Від нього через асоціацію «createdBy» з кардинальністю 1→* походять класи Equipment, ContactInfo, AccessSchedule, NetworkSchema, CoverageArea, WorkLog та SignalRecord, кожен із власним набором атрибутів (наприклад, type, status у Equipment чи address, phone у ContactInfo). Додатково WorkLog пов’язано з User як performedBy, а SignalRecord — як recordedBy. Композиція між AccessSchedule і вкладеним класом Interval відображає структуру масиву часових інтервалів у розкладі доступу. Це моделювання демонструє, як однакова сутність користувача є творцем або відповідальним за всі ключові елементи предметної області.

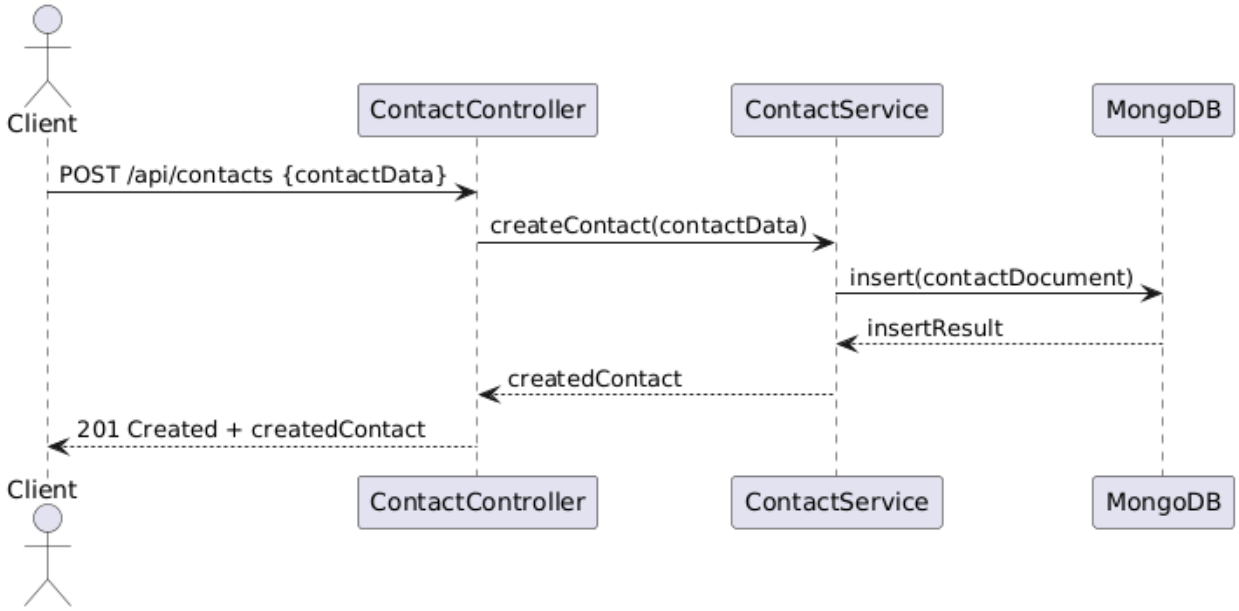


Рис. 2.4 Діаграма послідовності: додавання нового контакту

На рис. 2.4 показано, як клієнт надсилає запит POST на /api/contacts з даними контактної особи, після чого ContactController передає ці дані до ContactService, який виконує вставку документа у колекцію MongoDB. Повернутий результат вставки надходить назад через ContactController клієнту з кодом відповіді 201 Created.

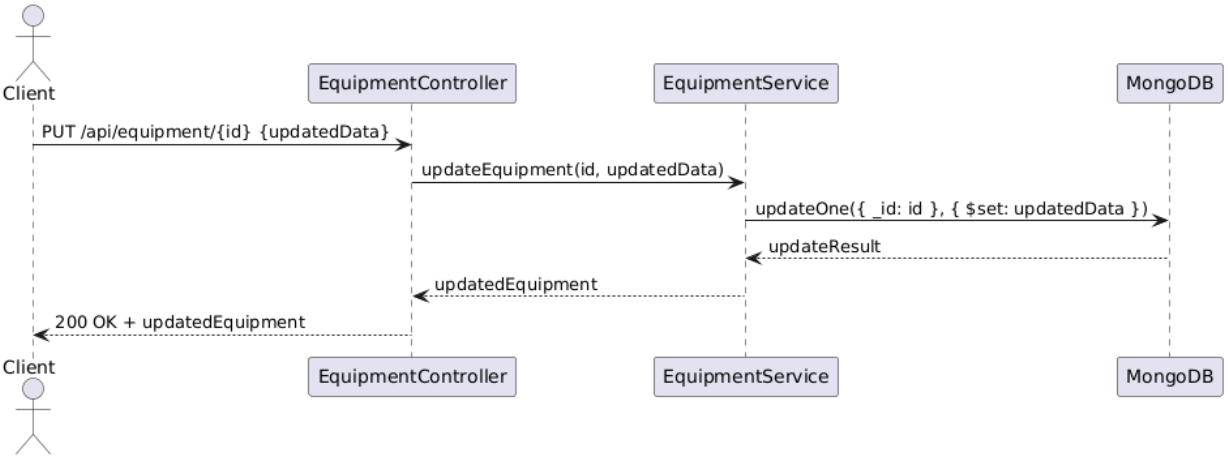


Рис. 2.5 Діаграма послідовності: редагування обладнання

На рис. 2.5 показано,, як клієнт виконує запит PUT до /api/equipment/{id} з оновленими даними, після чого EquipmentController делегує їх EquipmentService, що оновлює відповідний документ у MongoDB командою updateOne. Результат оновлення пересилається назад клієнту з кодом 200 OK.

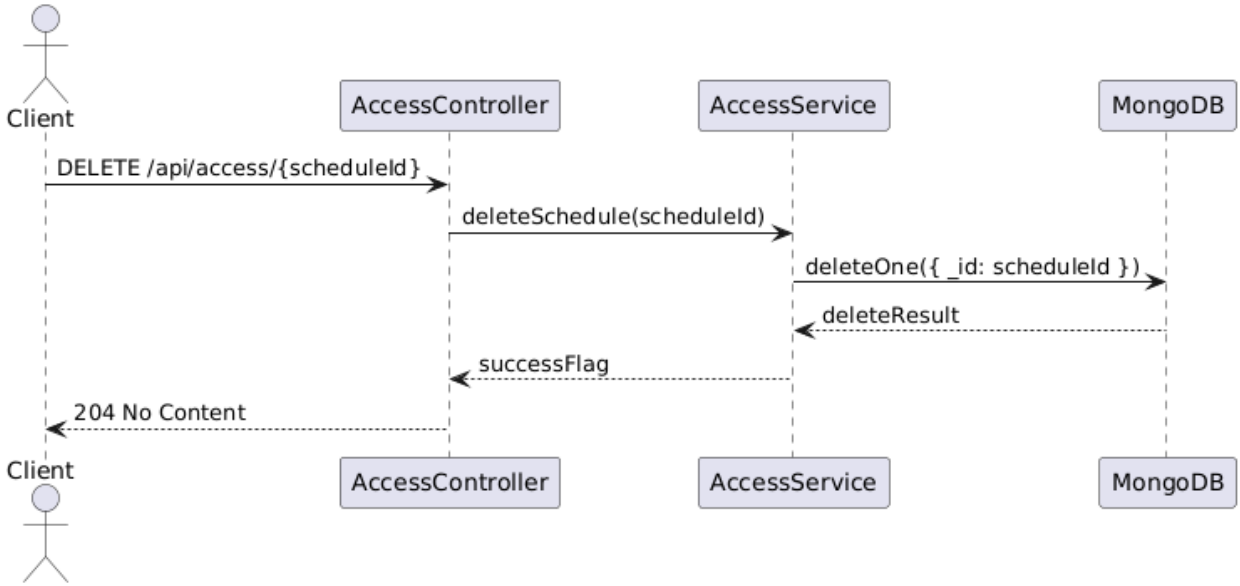


Рис. 2.6 Діаграма послідовності: видалення розкладу доступу

На рис. 2.6 показано, як клієнт надсилає DELETE-запит до /api/access/{scheduleId}, AccessController викликає AccessService, який видаляє документ із MongoDB за допомогою deleteOne. Після успішної операції контролер повертає клієнту код 204 No Content.

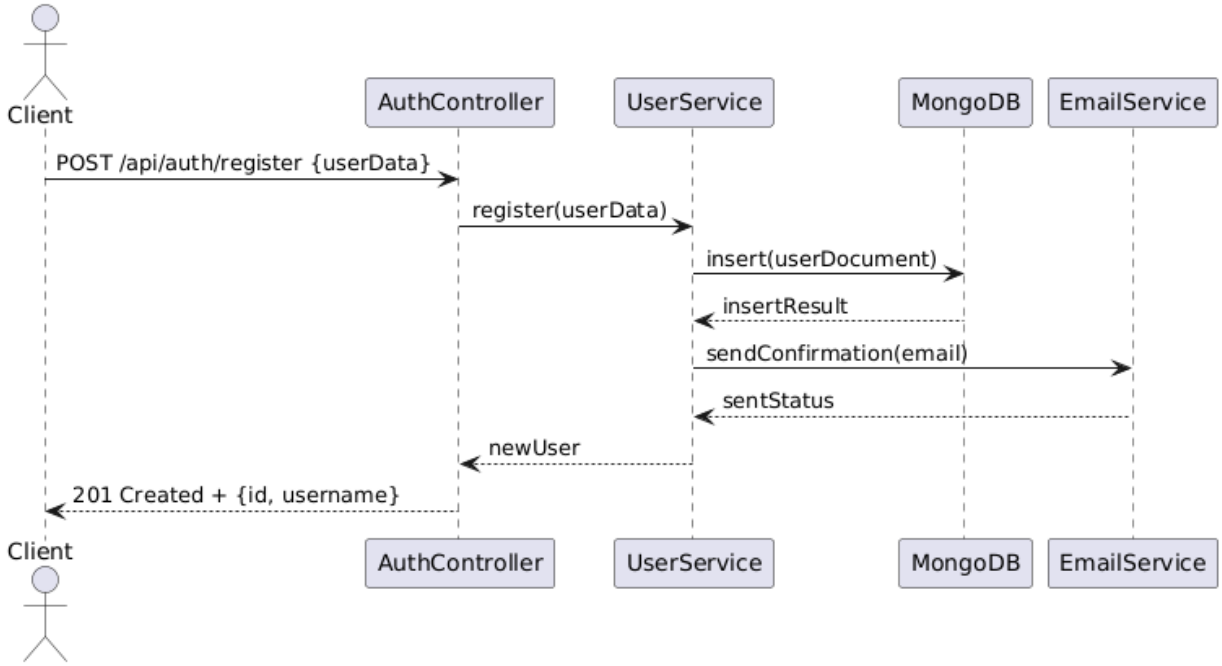


Рис. 2.7 Діаграма послідовності: реєстрація нового користувача

На рис. 2.7 показано, як клієнт відправляє POST-запит на /api/auth/register із даними користувача, AuthController передає їх у UserService, який додає документ до MongoDB та надсилає лист підтвердження через EmailService.

Після цього AuthController повертає клієнту відповідь 201 Created з ідентифікатором нового користувача.

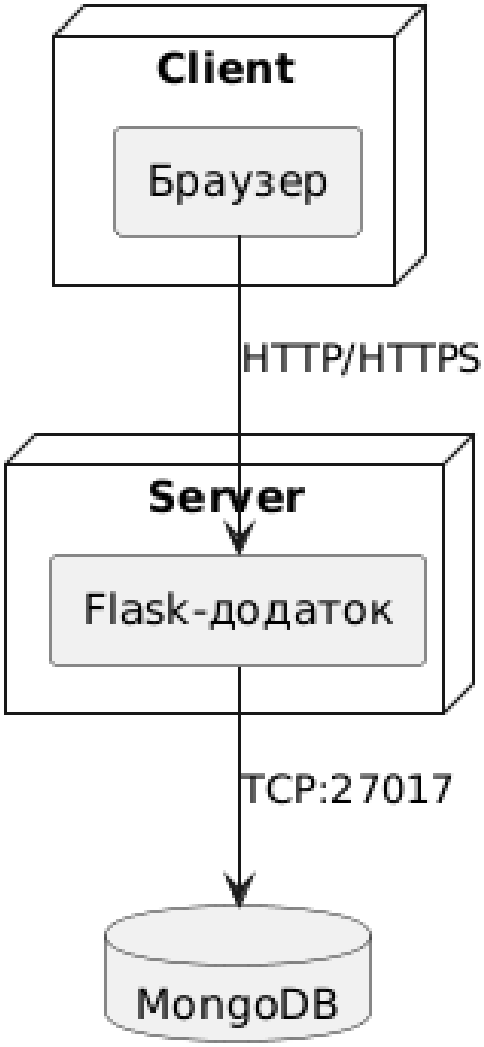


Рис. 2.9 Діаграма розгортання

На рис. 2.9 показано фізичну архітектуру розгортання: браузер на клієнті встановлює HTTPS-з’єднання з Flask-додатком (розгорнутим за допомогою NGINX у Docker-контейнері), а для зберігання й читання даних використовується MongoDB, до якої сервер підключається по протоколу TCP на порті 27017.

2.5 Моделювання REST-API

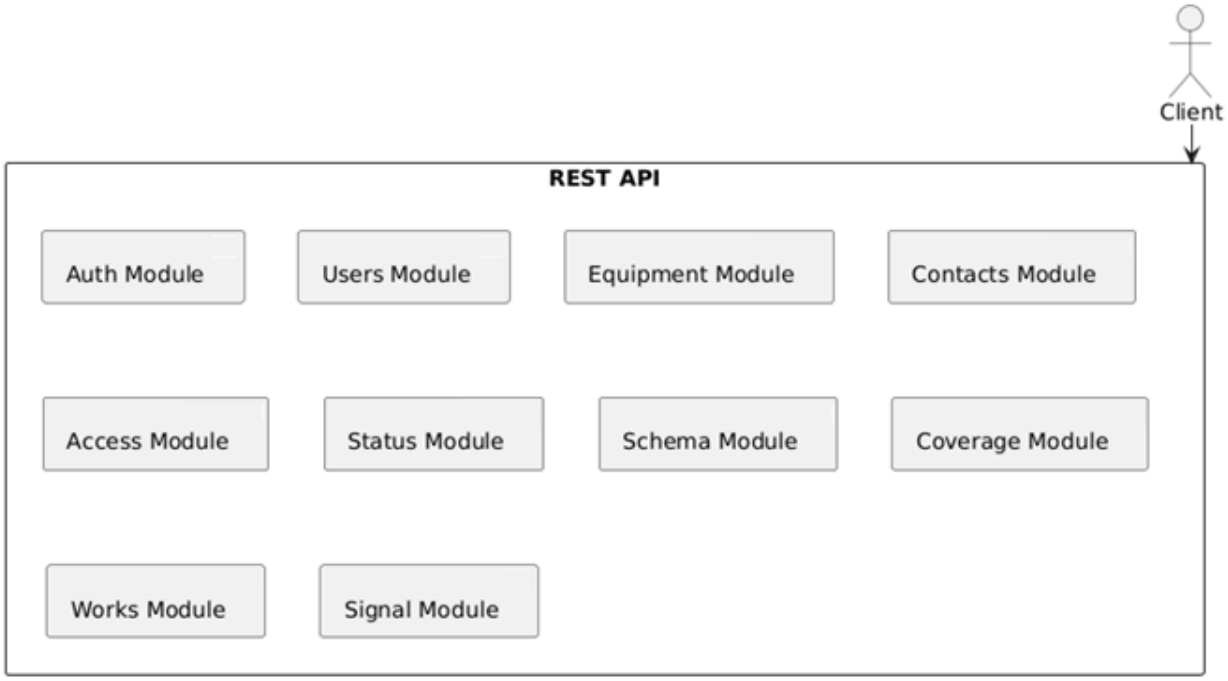


Рис. 2.12 Спрощена схема REST-API

На рис. 2.10 показано спрощену логічну схему REST-API нашої системи: є єдина точка входу «REST API», яка складається з окремих модулів для автентифікації (Auth), управління користувачами (Users), обладнанням (Equipment), контактами (Contacts), розкладами доступу (Access), статусом (Status), схемою мережі (Schema), зонами покриття (Coverage), журналом робіт (Works) та параметрами сигналу (Signal). Клієнт (браузер або інша зовнішня аплікація) здійснює запити до цього REST API, а модулі всередині обробляють відповідні CRUD-операції над даними.

2.6 Дизайн користувацького інтерфейсу

Прототипи та вайрфрейми створені у Figma й ілюструють основні екрани системи: сторінку реєстрації/входу, дашборд із оглядом обладнання та контактів, а також довідкові розділи (розклад доступу, топологічна схема, журнал робіт, моніторинг сигналу). Навігація реалізована через верхнє меню та бокову панель із іконками, а контент організовано в картки й таблиці з

чіткою ієрархією заголовків і кнопок дій. Інтерфейс побудовано за світлою темою з акцентами корпоративного синього кольору, а всі елементи управління мають достатню клікабельну область і плавні анімації станів (hover, active).

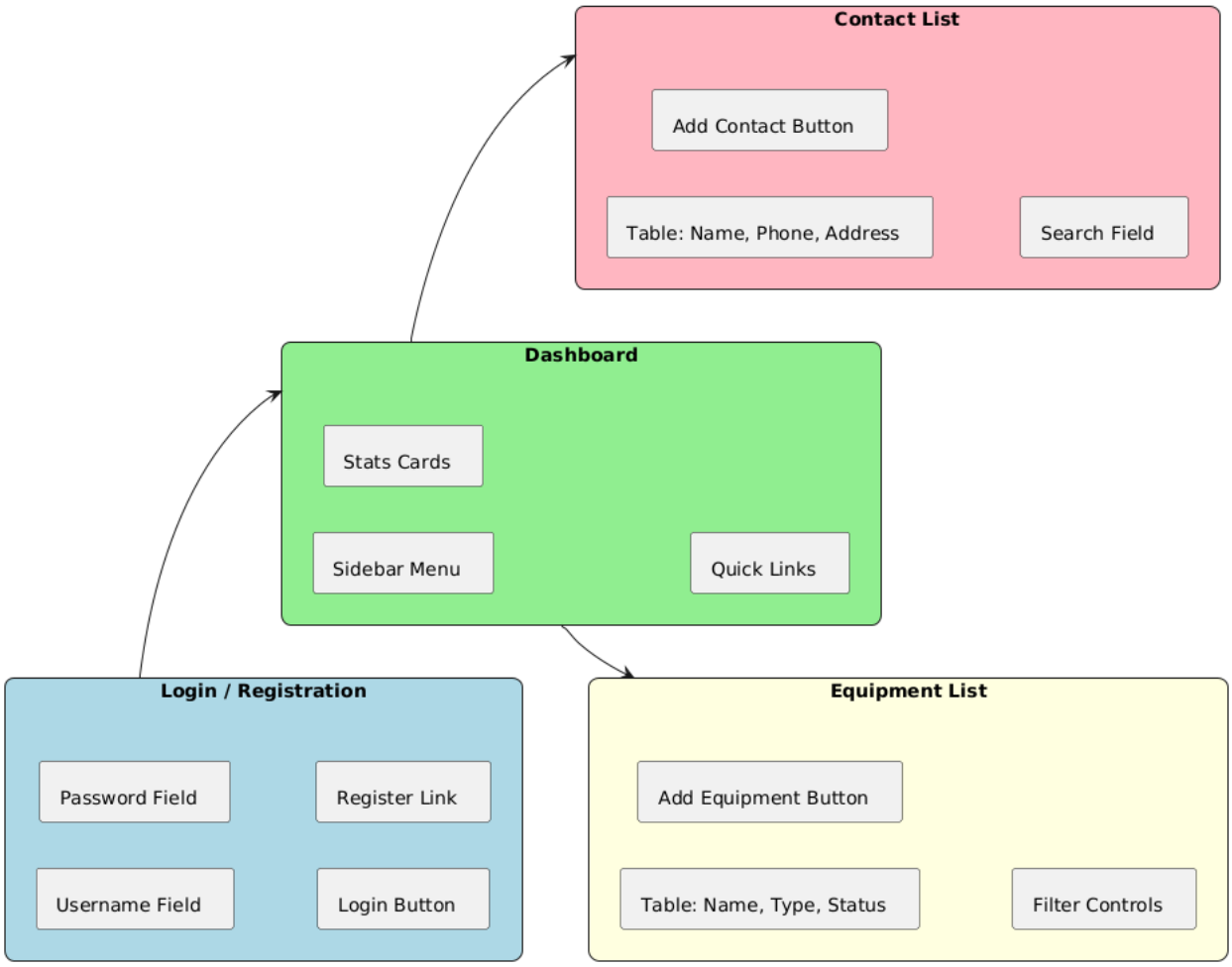


Рис. 2.13 Прототипи та wireframes

Основні компоненти інтерфейсу:

1. Сторінка реєстрації/входу

– Поле для введення логіна та пароля із підписами і вбудованою валідацією;

– Кнопки “Увійти” і “Зареєструватися” з виразним акцентним кольором;

– Покликання “Забули пароль?” та мінімалістичний логотип зверху для впізнаваності.

2. Головний дашборд

– Бокове меню з пунктами: «Обладнання», «Контакти», «Розклад», «Схема», «Покриття», «Роботи», «Сигнал»;

– Віджет загальної статистики (кількість пристроїв онлайн/офлайн, активні контакти);

– Картки швидкого доступу до основних модулів із іконками.

3. Список обладнання

– Таблиця з колонками: Назва, Тип, Статус, Кабель, Сплайс, Дії;

– Фільтр за типом та полем статусу над таблицею;

– Кнопки “Додати обладнання” й “Експорт CSV” в правому верхньому куті;

– Рядки з чергуванням фону для кращої читаності, кнопки “Edit”/“Delete” з іконками.

4. Форма “Додати/Редагувати обладнання”

– Поля: Тип (дропдаун), Статус (тоггл), Кабель (текст), Сплайс (текст);

– Валідація обов’язкових полів із підсвічуванням помилок;

– Кнопки “Скасувати” (сірий) і “Зберегти” (акцентний синій) унизу форми.

5. Довідкові модулі (Розклад, Схема, Покриття, Роботи, Сигнал)

– Єдина стилістика: заголовок сторінки, кнопка “Додати”, табличне або графічне представлення даних;

– Для топологічної схеми використано інтерактивний SVG-рендеринг вузлів і ліній;

– У журналі робіт активні посилання на деталі й історію змін.

6. Мобільна адаптація (В подальших версіях)

– Перехід таблиць у вертикальний список карток при ширині < 768 px;

– При мобільному відображенні (ширина < 768 px) бокова навігація ховається за іконкою з трьома горизонтальними лініями, що відкриває меню.

– Збільшені таконатори (touch targets) для комфортного використання на дотик.

7. Оповіщення й підтвердження

– Toast-повідомлення вгорі екрана про успіх/помилку, що автоматично зникають через 3 с;

– Модальні діалоги з підтвердженням видалення або критичних дій, структуровані за шаблоном: заголовок, опис, кнопки “Скасувати”/“Підтвердити”.

8. Типографіка та кольорова палітра

– Шрифти: Roboto (заголовки), Open Sans (текст) з комфортними розмірами (16 px для тексту, 24–32 px для заголовків);

– Основний фон — світло-сірий (#F7F9FA), картки — білий із м’якими тінями;

– Акцентний колір кнопок і посилань — насичений синій (#0066CC), кольори помилок — червоний (#E74C3C), успіху — зелений (#27AE60).

Таке рішення забезпечує інтуїтивність, швидкість роботи та однорідність інтерфейсу в усіх модулях системи.

2.7 Формулювання функціональних та нефункціональних вимог

Таблиця 2.5

Функціональні вимоги

ID	Вимога	Опис
F1	CRUD-операції з обладнанням	Створення, читання, оновлення та видалення записів про мережеве обладнання (type, status, cable, splice).
F2	CRUD-операції з контактною інформацією	Створення, читання, оновлення та видалення контактів (address, entrance, floor, apartment, phone).
F3	CRUD-операції з розкладом доступу	Управління графіком доступу до обладнання: додати/редагувати/видалити інтервали (start, end).
F4	Моніторинг статусу	Фіксація стану обладнання (onoff, mode, download, upload, speed) з можливістю перегляду історії.
F5	Управління топологічною схемою	Додавання й редагування вузлів, коробок та кабелів у мережевій схемі.
F6	Ведення журналу робіт	Запис робіт та обслуговування з полями address, desc, date і посиланням на виконавця (user).
F7	Моніторинг сигналу	Збирання параметрів сигналу (global, port, subscriber) та перегляд результатів.
F8	Реєстрація та аутентифікація	Форма реєстрації (username, email, password) з хешуванням пароля; вхід із JWT-cookie.
F9	Розмежування ролей	Три рівні доступу – user, manager, admin + superadmin із забороною зміни облікового запису супер-адміна.
F10	Адмін-панель	Інтерфейс /admin для керування користувачами, ролями та перегляду аудиторських логів.

Відповідно до Таблиці 2.5, система повинна підтримувати повний набір CRUD-операцій для всіх предметних областей (F1–F7), забезпечувати безпечну реєстрацію й вхід через JWT-автентифікацію (F8), розподіляти права між чотирма ролями з особливою позицією супер-адміністратора (F9) та надавати окремий інтерфейс для адміністративних функцій (F10).

Таблиця 2.6

Нефункціональні вимоги

ID	Вимога	Показник / Опис
N1	Безпека	bcrypt-хешування паролів, HTTPS для всіх маршрутів, захист CSRF токеном.
N2	Час відгуку API	≤ 200 мс для типових CRUD-операцій при середньому навантаженні.
N3	Пропускна здатність	≥ 1000 запитів/с на рівні Flask-додатку без деградації продуктивності.
N4	Масштабованість	Горизонтальне розгортання контейнерів Docker, Replica Set у MongoDB.
N5	Відмовостійкість	Мультивузлова реплікація БД, автоматичний перезапуск контейнерів (Docker Compose, Kubernetes liveness).
N6	Тестування	Покриття кодом ≥ 80 % юніт- і інтеграційними тестами (pytest); CI-пілайн для автотестів.
N7	Документація	OpenAPI/Swagger UI для всіх ендпоінтів, оновлення при кожному релізі.
N8	Usability	Інтуїтивний UI, адаптивний дизайн, час навчання ≤ 1 дня.
N9	Логування та аудит	Колекція audit_logs із TTL-індексом (30 днів) та індексом на user_id.
N10	Портативність	Контейнеризація через Docker/Docker Compose, сумісність з будь-якою хмарною чи локальною інфраструктурою.

Нефункціональні вимоги в табл. 6 гарантують безпеку (N1), продуктивність (N2–N3), масштабованість і відмовостійкість (N4–N5), високу якість коду (N6), актуальну документацію (N7), зручність користувацького досвіду (N8), прозоре логування й аудит (N9) та простоту розгортання в будь-якому середовищі (N10).

2.8 Моделювання модулів автентифікації та управління ролями користувачів

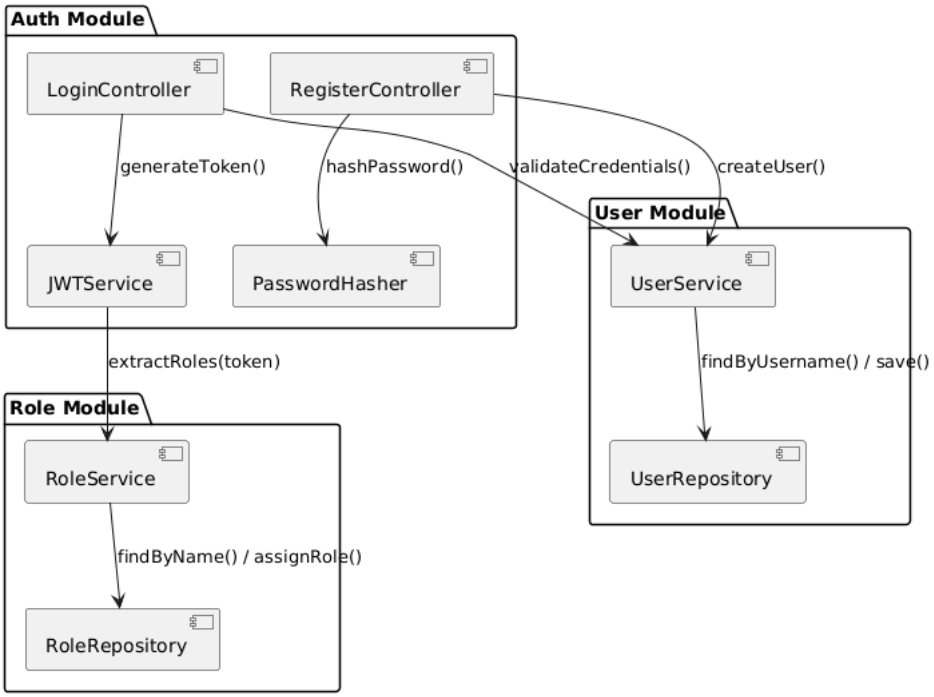


Рис. 2.14 Компонентна діаграма модулів автентифікації та управління ролями

На рис. 2.14 показано компонентну діаграму модулів автентифікації та управління ролями: Auth Module містить контролери LoginController і RegisterController, сервіс JWTService для генерації та перевірки токенів і PasswordHasher для хешування паролів; User Module відповідає за бізнес-логіку користувачів через UserService і доступ до даних у UserRepository; Role Module реалізує управління ролями за допомогою RoleService і RoleRepository. Взаємодії стрілками відображають перевірку облікових даних, створення

користувача з хешуванням пароля, генерацію токена та вилучення ролей із нього.

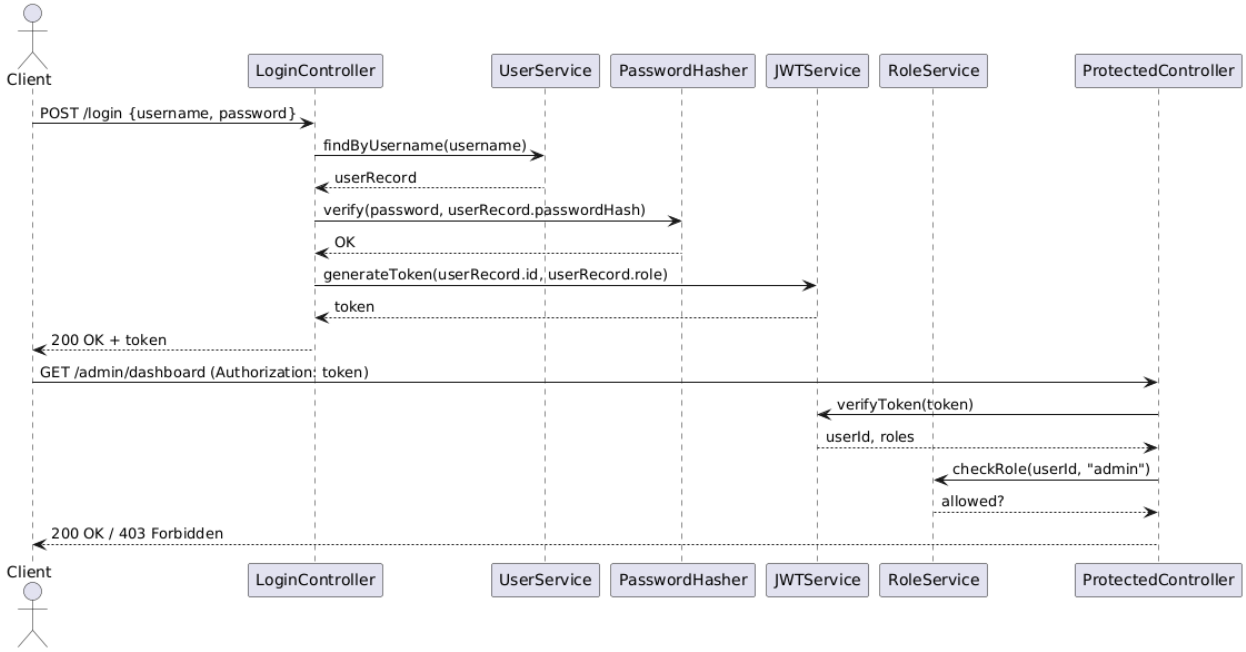


Рис. 2.15 Авторизація та перевірка ролі

На рис. 2.15 наведено діаграму послідовності сценарію авторизації та перевірки ролей: клієнт надсилає POST-запит на /login, LoginController викликає UserService для пошуку користувача і через PasswordHasher перевіряє пароль, після чого через JWTService генерує токен; далі клієнт звертається до захищеного контролера з токеном, який JWTService верифікує та передає ролі у RoleService для перевірки доступу — у разі успішної авторизації контролер повертає 200 OK, інакше 403 Forbidden.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Налаштування середовища розробки та вибір інструментів

Розробка інформаційної системи обліку мережевого обладнання та контактів здійснюється у середовищі Python 3.9+, вибір якого обумовлений широкою підтримкою асинхронних веб-фреймворків, бібліотек для роботи з MongoDB та інструментів для тестування [22-23]. Рекомендується використовувати віртуальне середовище (venv або virtualenv) для ізоляції залежностей проекту. Файл requirements.txt містить ключові пакети: Flask для веб-серверу, Flask-PyMongo для інтеграції з MongoDB, Flask-JWT-Extended для JWT-автентифікації, bcrypt (або werkzeug.security) для хешування паролів, а також Faker для генерації тестових даних і pytest з плагіном pytest-cov для юніт- та інтеграційного тестування.

Для управління базою даних у локальному середовищі використовується MongoDB Community Server (версії ≥ 4.4), розгорнута або як окремий сервіс на розробницькій машині, або як контейнер Docker за допомогою docker-compose.yml. У контейнерному середовищі також передбачене підключення до інструментів моніторингу (наприклад, MongoDB Compass) для візуального контролю колекцій. Усі налаштування підключення до БД (URI, облікові дані) зберігаються в файлі config.py або через змінні середовища, щоб уникнути коммітації чутливих даних у репозиторій. Система збірки та контролю версій організована за допомогою Git із віддаленим репозиторієм на GitHub (або GitLab). Для покращення якості коду налаштовані інструменти статичного аналізу: flake8 для стилістичних перевірок, black для автозастосування єдиного форматування та isort для впорядкування імпортів. Рекомендовано підключити pre-commit hooks, які автоматично запускають форматування й лінтинг перед комітом. Для локального запуску та інтеграції тестів у CI/CD налаштовано GitHub Actions (або інший CI), що автоматично перевіряє код на проходження тестів із pytest, збирає звіти про покриття, перевіряє стиль і, за потреби, оформлює Docker-образ з мікросервісом. Такий підхід забезпечує стабільність збірок та швидку інтеграцію змін від кількох

розробників. Зокрема, середовище розробки доповнюють IDE із підтримкою Python (VS Code, PyCharm), інтеграцією з Docker і MongoDB-плагінами, а також браузер із встановленими REST-клієнтами (Postman або RESTer) для тестування API. Такий стек інструментів забезпечує швидкий старт, надійне тестування та прозоре розгортання застосунку.

3.2 Реалізація функціоналу системи

```
HomeNetControl/
        ├── app.py
        ├── read.txt
        ├── requirements.txt       # перелік залежностей (створити на основі read.txt)
        ├── seed_data.py
        ├── seed_users.py
        ├── tempCodeRunnerFile.py
        ├── config.py              # (опціонально) налаштування підключення до БД, секретний
ключ тощо
        ├── routes/
        │   ├── auth.py          # маршрути реєстрації, входу, виходу
        │   ├── equipment.py       # CRUD для обладнання
        │   ├── access.py         # CRUD для розкладу доступу
        │   ├── info.py           # CRUD для адресної інформації
        │   ├── status.py         # CRUD для моніторингу стану
        │   ├── schema.py         # CRUD для топологічної схеми
        │   ├── coverage.py       # CRUD для зон покриття
        │   ├── works.py          # CRUD для журналу робіт
        │   └── signal.py         # CRUD для параметрів сигналу
        ├── models/              # (опціонально) Pydantic/Marshmallow схеми валідації
        │   ├── user.py
        │   ├── equipment.py
        │   └── ...
        ├── services/            # (опціонально) бізнес-логіка
        │   ├── auth_service.py
        │   ├── user_service.py
        │   └── ...
        ├── templates/
        │   ├── login.html
        │   ├── register.html
        │   ├── index.html
        │   └── admin.html
        ├── static/
        │   ├── style.css
        │   └── script.js
        ├── tests/               # тести (pytest)
        │   ├── test_auth.py
        │   ├── test_equipment.py
        │   └── ...
        └── docker-compose.yml     # (опціонально) конфігурація Docker для dev-середовища-
```

Рис. 3.1 Лістинг файлів відносно системи

Для взаємодії з базою даних MongoDB у цьому Flask-проекті використовується синхронний драйвер PyMongo через розширення Flask-PyMongo, що забезпечує єдину точку підключення в app.py. Після ініціалізації mongo = PyMongo(app) всі CRUD-операції виконуються через колекції

mongo.db.<collection_name>, наприклад, insert_one(), find(), update_one() та delete_one(), які повертають результати негайно та не вимагають додаткового «await». Це дозволяє проста реалізація синхронних REST-ендпоінтів у Flask-роутах без блокувань подальших запитів. Для пакетного видобутку даних (наприклад, список усіх записів) застосовується метод find() із подальшою ітерацією курсора, а для агрегованих звітів — метод aggregate() із списком стадій конвеєра. Усі операції зберігають поле user у документах для аудиту, а додаткові індекси на полях username, email, type, status тощо забезпечують високу швидкість вибірки навіть при великих обсягах даних.

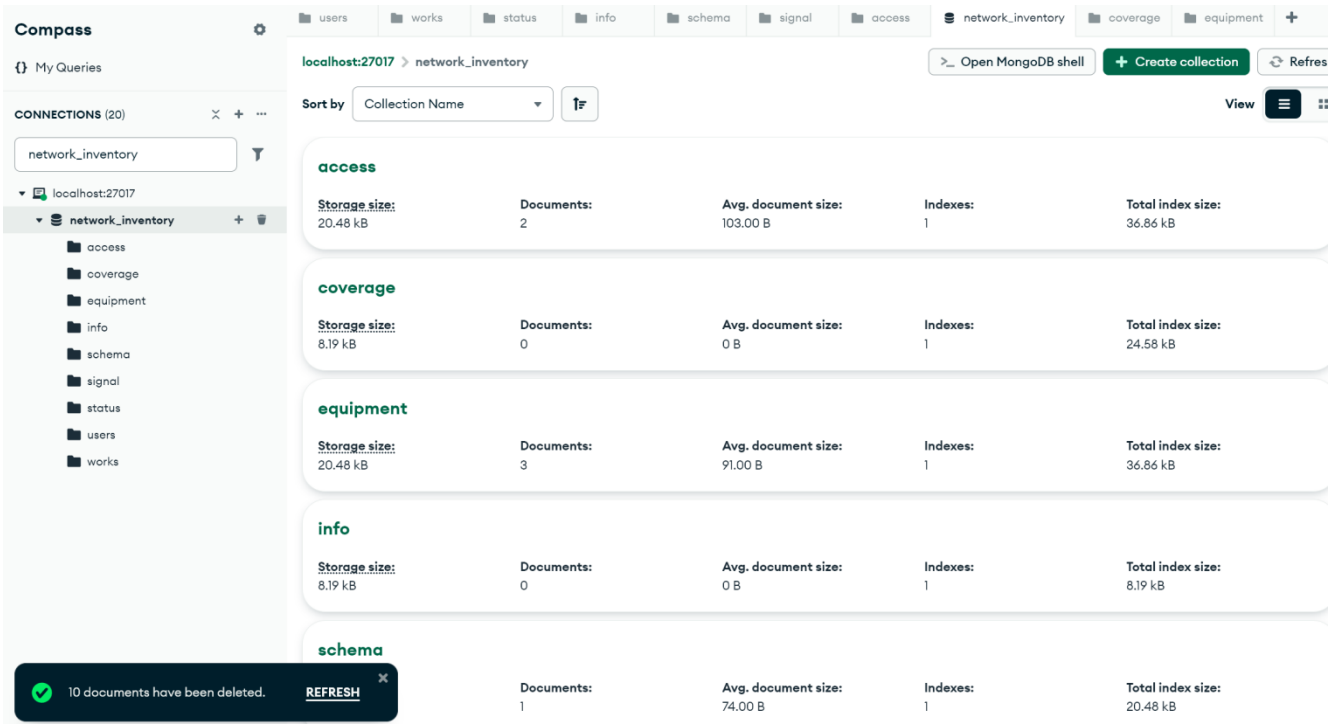


Рис. 3.2. Mongo DB Compas БД «network_inventory»

Нижче наведено стислу таблицю, що описує структуру основних колекцій бази даних:

Таблиця 3.1

Колекція «users»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
username	string	Логін користувача (має бути унікальним)
email	string	Електронна пошта (має бути унікальною)
password	string	Хеш пароля (bcrypt через werkzeug.security)
role	string	Роль користувача (user, manager, admin, superadmin)
created_at	ISODate	Дата створення запису (автоматично може задаватися через middleware або тригери)

Таблиця 3.2

Колекція «equipment»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
type	string	Тип пристрою (наприклад, «router», «switch»)
status	string	Поточний стан (наприклад, «online», «offline»)
cable	string	Опис кабельного з’єднання
splice	string	Параметри сплайсу
user	string	Ім’я користувача, що додав або оновив запис
created_at	ISODate	Дата створення
updated_at	ISODate	Дата останнього оновлення

Таблиця 3.3

Колекція «access»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
entity	string	Ідентифікатор обладнання або локації
schedule	array of obj	Масив інтервалів доступу { start: ISODate, end: ISODate }
contact	string	Ім’я контактної особи (або її _id)
user	string	Ім’я користувача, що створив запис
created_at	ISODate	Дата створення

Таблиця 3.4

Колекція «info»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
address	string	Адреса місця
entrance	string	Під’їзд
floor	string	Поверх
apartment	string	Квартира
phone	string	Номер телефону
ser	string	Ім’я користувача, що додав запис
created_at	ISODate	Дата створення

Таблиця 3.5

Колекція «status»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
onoff	string	Вкл/Викл
mode	string	Режим роботи
download	string	Швидкість завантаження
upload	string	Швидкість віддачі
speed	string	Загальна пропускна здатність
user	string	Ім’я користувача, що зафіксував стан
created_at	ISODate	Дата створення

Таблиця 3.6

Колекція «schema»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
node	string	Ідентифікатор або назва вузла
box	string	Коробка з’єднань
cable	string	Кабель між елементами
user	string	Ім’я користувача, що побудував схему
created_at	ISODate	Дата створення

Таблиця 3.7

Колекція «coverage»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор
area	string	Географічна зона
house	string	Будівля
status	string	Статус покриття (активне/неактивне)
user	string	Ім’я користувача, що додав запис
created_at	ISODate	Дата створення

Таблиця 3.8

Колекція «works»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
address	string	Адреса виконання робіт
desc	string	Опис робіт
date	ISODate	Дата виконання
user	string	Ім’я користувача, що виконав роботи
created_at	ISODate	Дата створення

Таблиця 3.9

Колекція «signal»

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа
global	string	Загальний рівень сигналу
port	string	Порт підключення
subscriber	string	Ім’я абонента або його ідентифікатор
user	string	Ім’я користувача, що зняв заміри
created_at	ISODate	Дата створення

БД «network_inventory» включає дев’ять основних колекцій: users, де зберігаються облікові записи з унікальними іменами користувачів, електронною поштою та захищеними bcrypt-хешами паролів разом із ролями (user, manager, admin, superadmin); equipment, у якій фіксується мережеве обладнання (тип, статус, дані кабелю та сплайсу) з посиланням на автора запису; access, що містить розклад доступу до обладнання або локацій у вигляді масиву інтервалів із прив’язкою до контактної особи; info, де зберігаються деталі адрес (адреса, під’їзд, поверх, квартира, телефон) відповідальних осіб; status, яка фіксує поточний стан обладнання (вкл/викл, режим роботи, параметри швидкості) і автора спостереження; schema, що моделює топологічну схему мережі (вузли, коробки, кабелі) із вказанням автора; coverage, де акумулюється інформація про зони покриття (географічна область, будівля, статус покриття) із вказанням автора; works, яка веде журнал виконаних робіт (адреса, опис, дата, виконавець); та signal, що зберігає параметри сигналу для абонентів (рівень, порт, ідентифікатор абонента) і

автора вимірювання. Кожна колекція, окрім users, має поле user, яке вказує на документ користувача для забезпечення аудиту й контролю доступу.

3.3 Реалізація модуля обліку контактних даних відповідальних осіб

Модуль обліку контактних даних реалізовано як набір REST-ендпоінтів у Flask із відповідними шаблонами для відображення та JavaScript-скриптами для динамічної взаємодії. У routes/info.py визначено чотири основні маршрути:

1. GET /api/info повертає JSON-список усіх контактних записів із полями id, address, entrance, floor, apartment, phone;

2. POST /api/info (з декоратором @manager_required) — приймає JSON із новими даними, валідовує наявність обов’язкових полів і виконує mongo.db.info.insert_one(), після чого повертає створений id;

3. PUT /api/info/<item_id> (з @manager_required) — оновлює вказаний документ через mongo.db.info.update_one() зі значеннями, що надійшли в тілі запиту;

4. DELETE /api/info/<item_id> (з @manager_required) — видаляє документ за _id методом delete_one().

Для валідації вхідних даних застосовано просту перевірку на наявність непустих рядків у ключових полях (address, phone) у самих функціях роутів — у разі некоректних або відсутніх даних повертається помилка 400 із повідомленням про те, яке саме поле не заповнене. На клієнтській стороні у шаблоні templates/index.html реалізовано окрему вкладку «Контакти». З JavaScript (static/script.js) відбувається завантаження списку через fetch('/api/info') та рендер кожного запису у табличному елементі; кнопки «Додати», «Редагувати» та «Видалити» відкривають модальні форми, де після підтвердження здійснюється виклик відповідного ендпоінту. Після успішної операції дані таблиці оновлюються без перезавантаження сторінки. Для забезпечення контролю доступу лише менеджерам та адміністраторам використано декоратор @manager_required: при спробі виконати

POST/PUT/DELETE за відсутності відповідних прав сервер повертає 403 Forbidden у форматі JSON.

3.4 Розробка інтерфейсу та інтерактивних компонентів

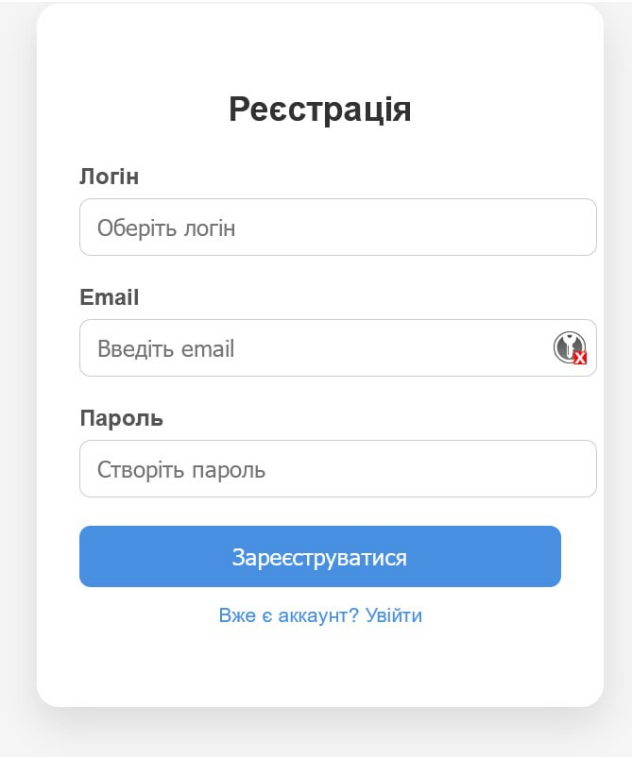


Рис. 3.3 Реєстрація користувача

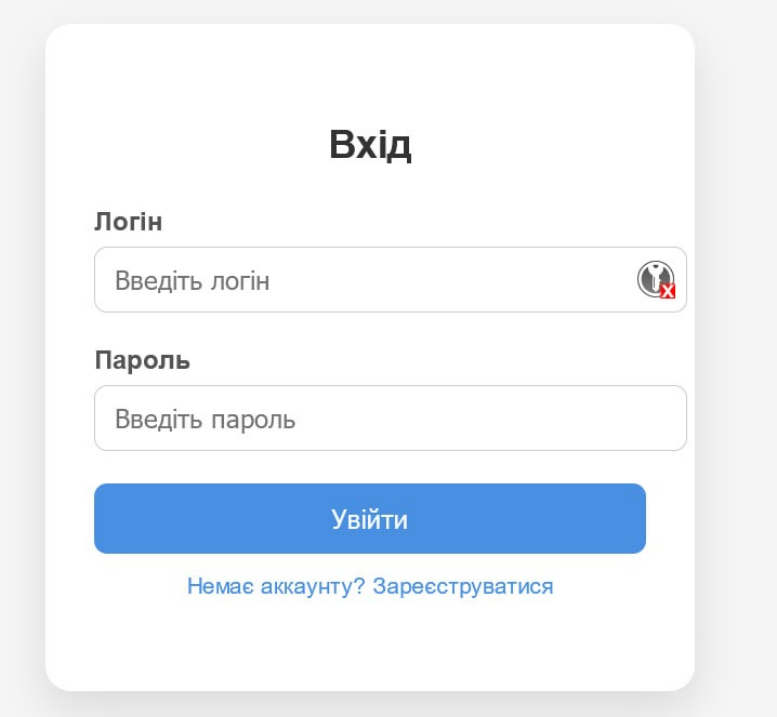
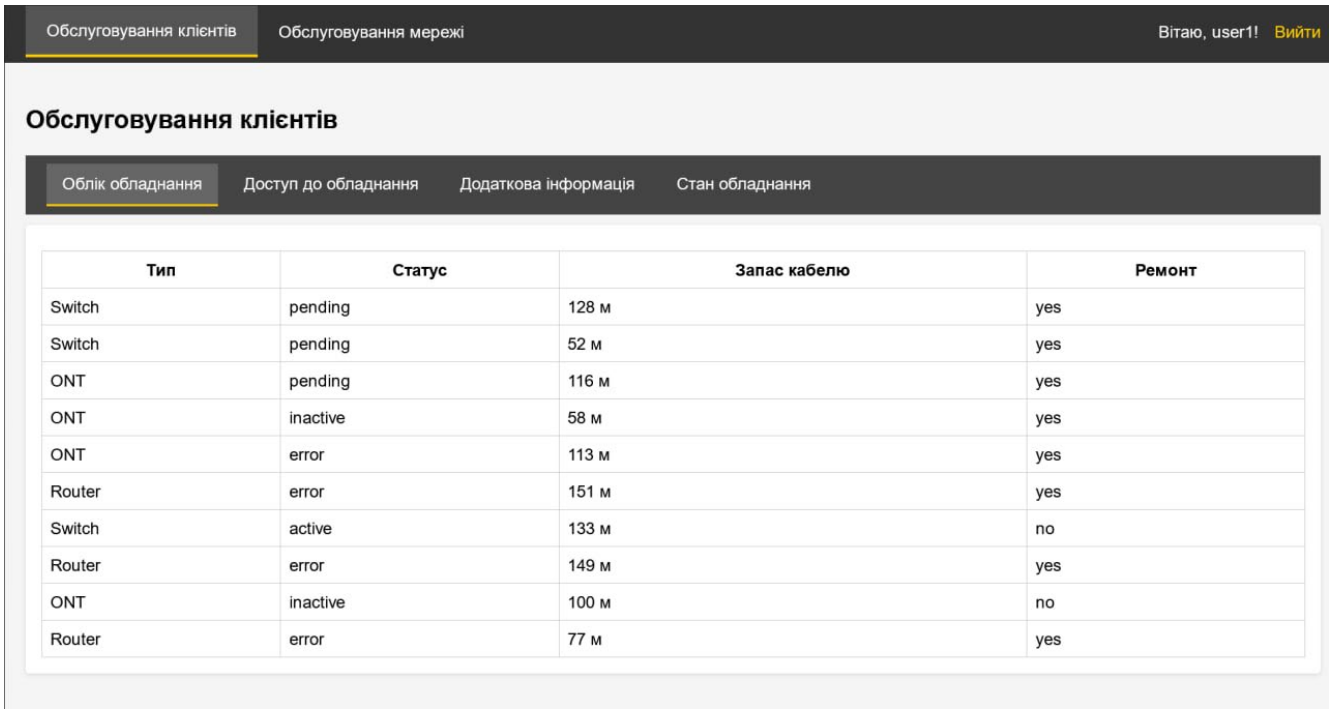
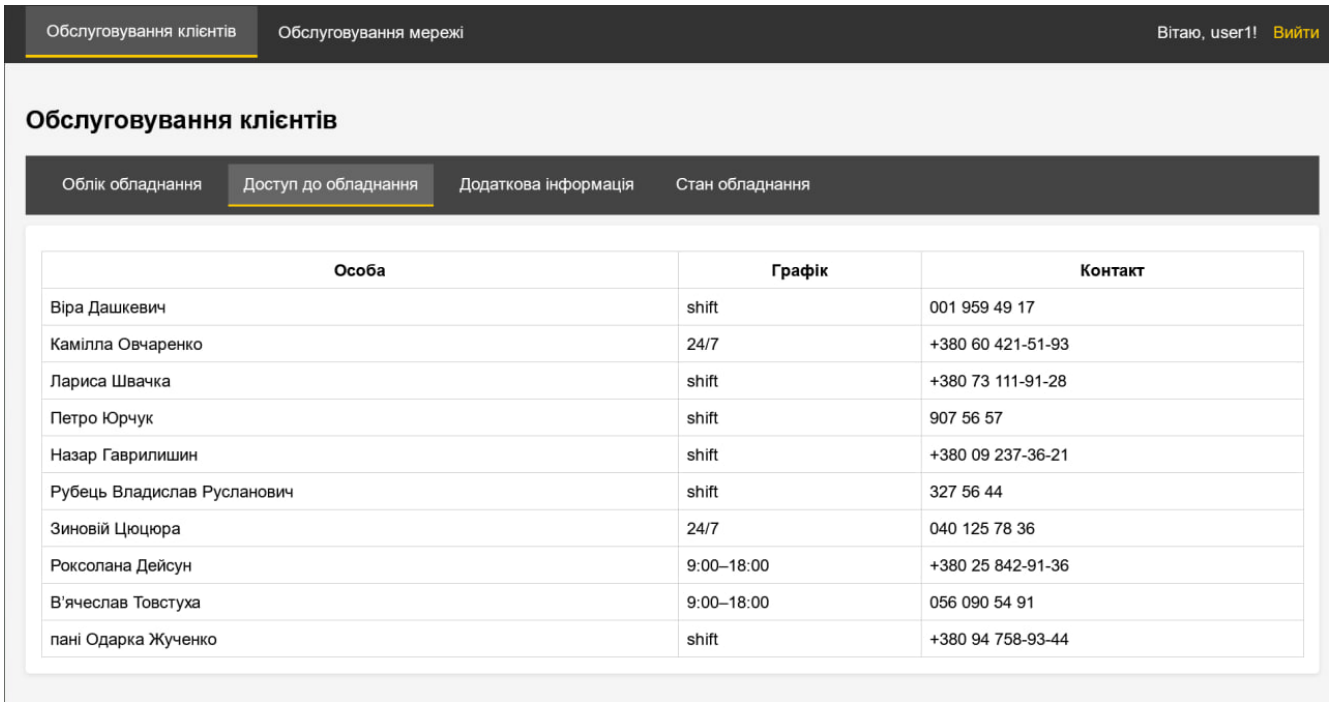


Рис. 3.4 Вхід користувача



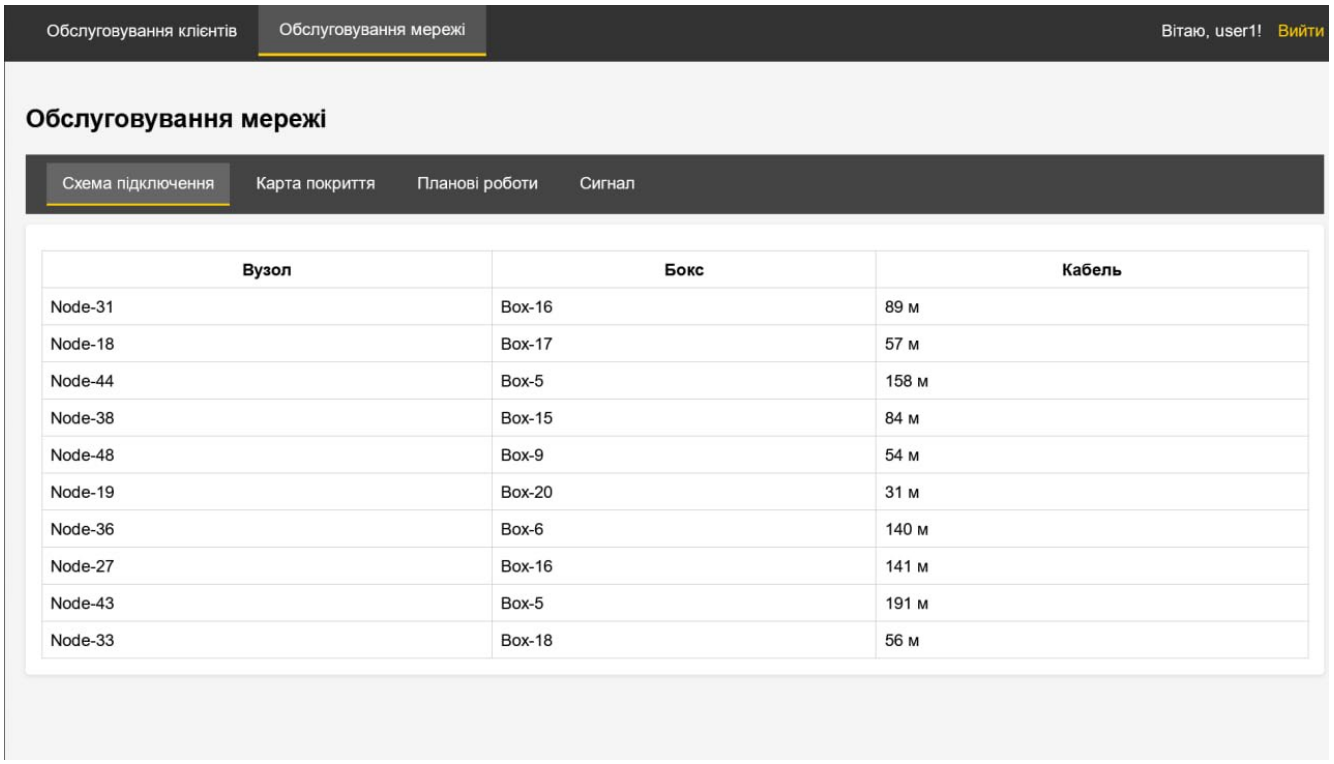
Тип	Статус	Запас кабелю	Ремонт
Switch	pending	128 м	yes
Switch	pending	52 м	yes
ONT	pending	116 м	yes
ONT	inactive	58 м	yes
ONT	error	113 м	yes
Router	error	151 м	yes
Switch	active	133 м	no
Router	error	149 м	yes
ONT	inactive	100 м	no
Router	error	77 м	yes

Рис. 3.5 Вхід в систему як користувача (відвідувач)



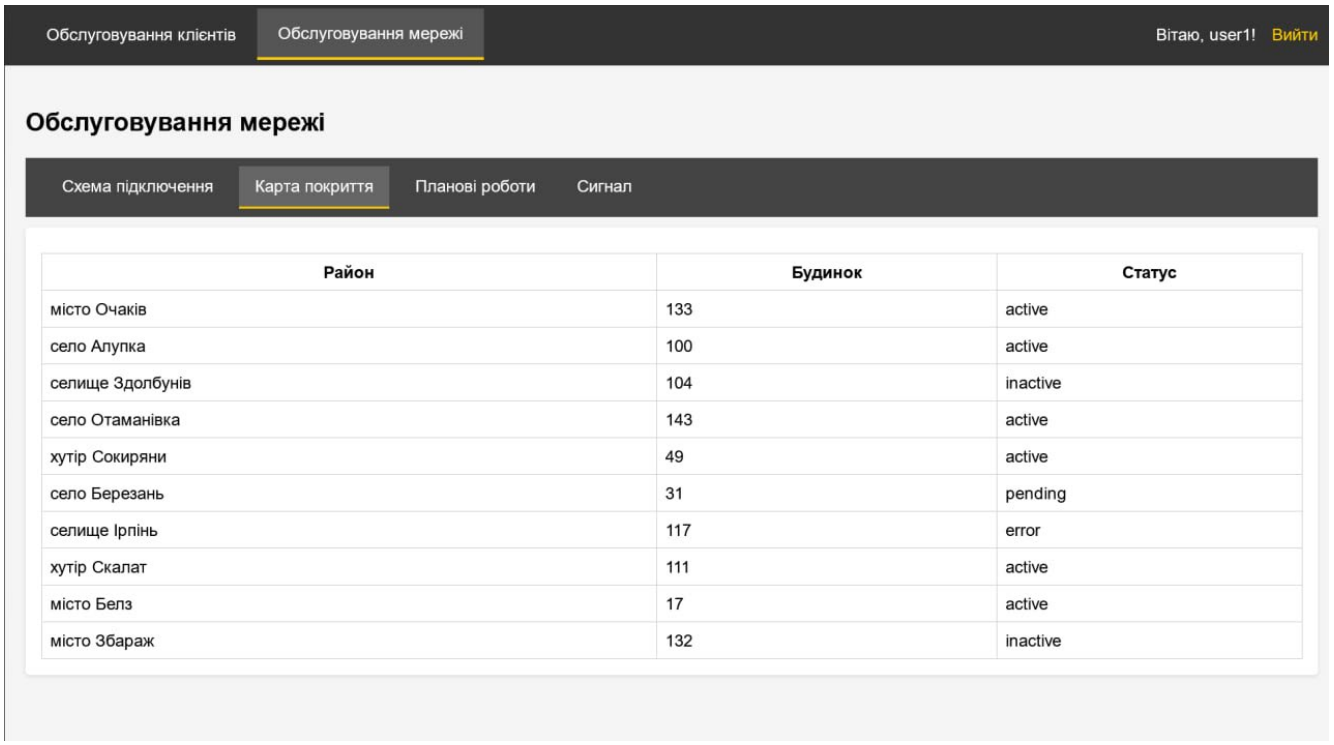
Особа	Графік	Контакт
Віра Дашкевич	shift	001 959 49 17
Камілла Овчаренко	24/7	+380 60 421-51-93
Лариса Швачка	shift	+380 73 111-91-28
Петро Юрчук	shift	907 56 57
Назар Гаврилишин	shift	+380 09 237-36-21
Рубець Владислав Русланович	shift	327 56 44
Зиновій Цюцюра	24/7	040 125 78 36
Роксолана Дейсун	9:00–18:00	+380 25 842-91-36
В'ячеслав Товстуха	9:00–18:00	056 090 54 91
пані Одарка Жученко	shift	+380 94 758-93-44

Рис. 3.6 Вхід в систему як користувача (відвідувач)



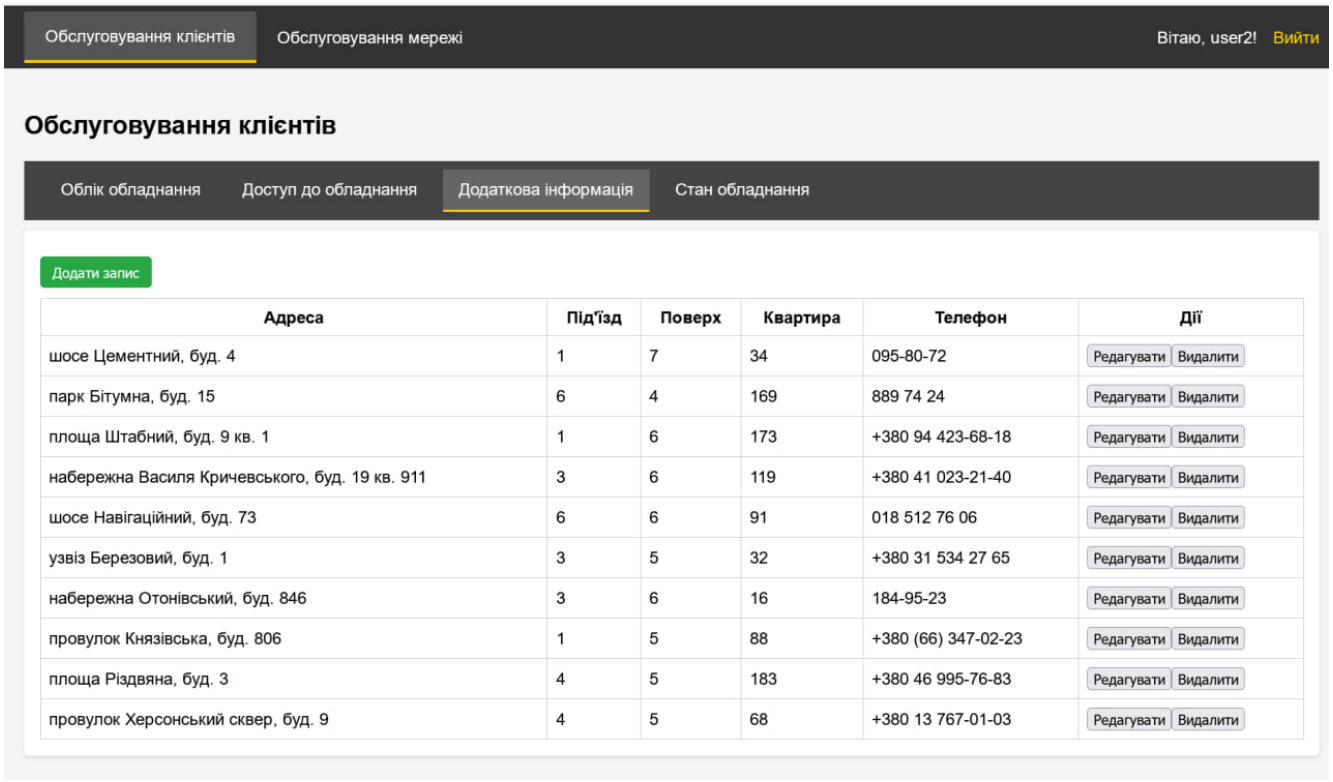
Вузол	Бокс	Кабель
Node-31	Box-16	89 м
Node-18	Box-17	57 м
Node-44	Box-5	158 м
Node-38	Box-15	84 м
Node-48	Box-9	54 м
Node-19	Box-20	31 м
Node-36	Box-6	140 м
Node-27	Box-16	141 м
Node-43	Box-5	191 м
Node-33	Box-18	56 м

Рис. 3.7 Вхід в систему як користувача (відвідувач)



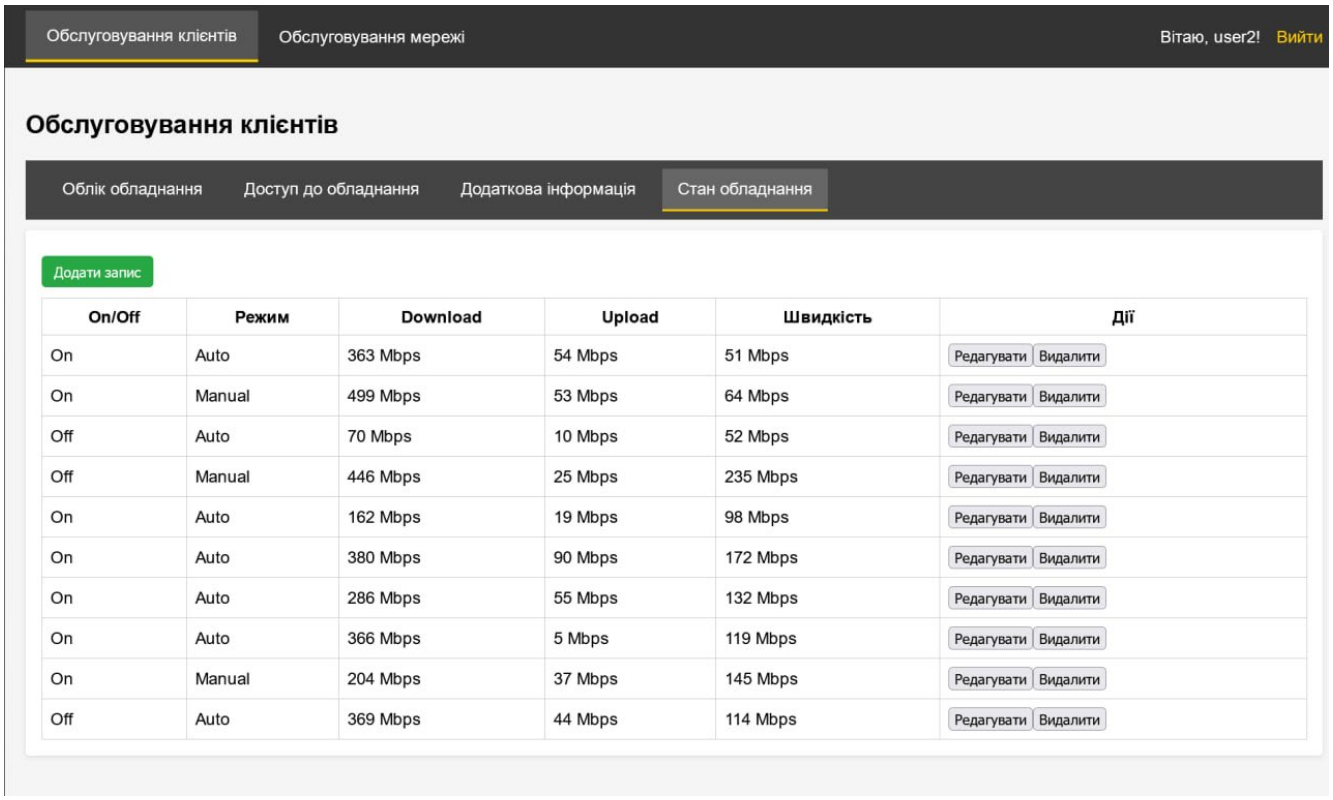
Район	Будинок	Статус
місто Очаків	133	active
село Алупка	100	active
селище Здолбунів	104	inactive
село Отаманівка	143	active
хутір Сокиряни	49	active
село Березань	31	pending
селище Ірпінь	117	error
хутір Скалат	111	active
місто Белз	17	active
місто Збараж	132	inactive

Рис. 3.8 Вхід в систему як користувача (відвідувач)



Адреса	Під'їзд	Поверх	Квартира	Телефон	Дії
шосе Цементний, буд. 4	1	7	34	095-80-72	Редагувати Видалити
парк Бітумна, буд. 15	6	4	169	889 74 24	Редагувати Видалити
площа Штабний, буд. 9 кв. 1	1	6	173	+380 94 423-68-18	Редагувати Видалити
набережна Василя Кричевського, буд. 19 кв. 911	3	6	119	+380 41 023-21-40	Редагувати Видалити
шосе Навігаційний, буд. 73	6	6	91	018 512 76 06	Редагувати Видалити
узвіз Березовий, буд. 1	3	5	32	+380 31 534 27 65	Редагувати Видалити
набережна Отонівський, буд. 846	3	6	16	184-95-23	Редагувати Видалити
провулок Князівська, буд. 806	1	5	88	+380 (66) 347-02-23	Редагувати Видалити
площа Різдвяна, буд. 3	4	5	183	+380 46 995-76-83	Редагувати Видалити
провулок Херсонський сквер, буд. 9	4	5	68	+380 13 767-01-03	Редагувати Видалити

Рис. 3.9 Вхід в систему як менеджера (функції редагування)



On/Off	Режим	Download	Upload	Швидкість	Дії
On	Auto	363 Mbps	54 Mbps	51 Mbps	Редагувати Видалити
On	Manual	499 Mbps	53 Mbps	64 Mbps	Редагувати Видалити
Off	Auto	70 Mbps	10 Mbps	52 Mbps	Редагувати Видалити
Off	Manual	446 Mbps	25 Mbps	235 Mbps	Редагувати Видалити
On	Auto	162 Mbps	19 Mbps	98 Mbps	Редагувати Видалити
On	Auto	380 Mbps	90 Mbps	172 Mbps	Редагувати Видалити
On	Auto	286 Mbps	55 Mbps	132 Mbps	Редагувати Видалити
On	Auto	366 Mbps	5 Mbps	119 Mbps	Редагувати Видалити
On	Manual	204 Mbps	37 Mbps	145 Mbps	Редагувати Видалити
Off	Auto	369 Mbps	44 Mbps	114 Mbps	Редагувати Видалити

Рис. 3.10 Вхід в систему як менеджера (функції редагування)

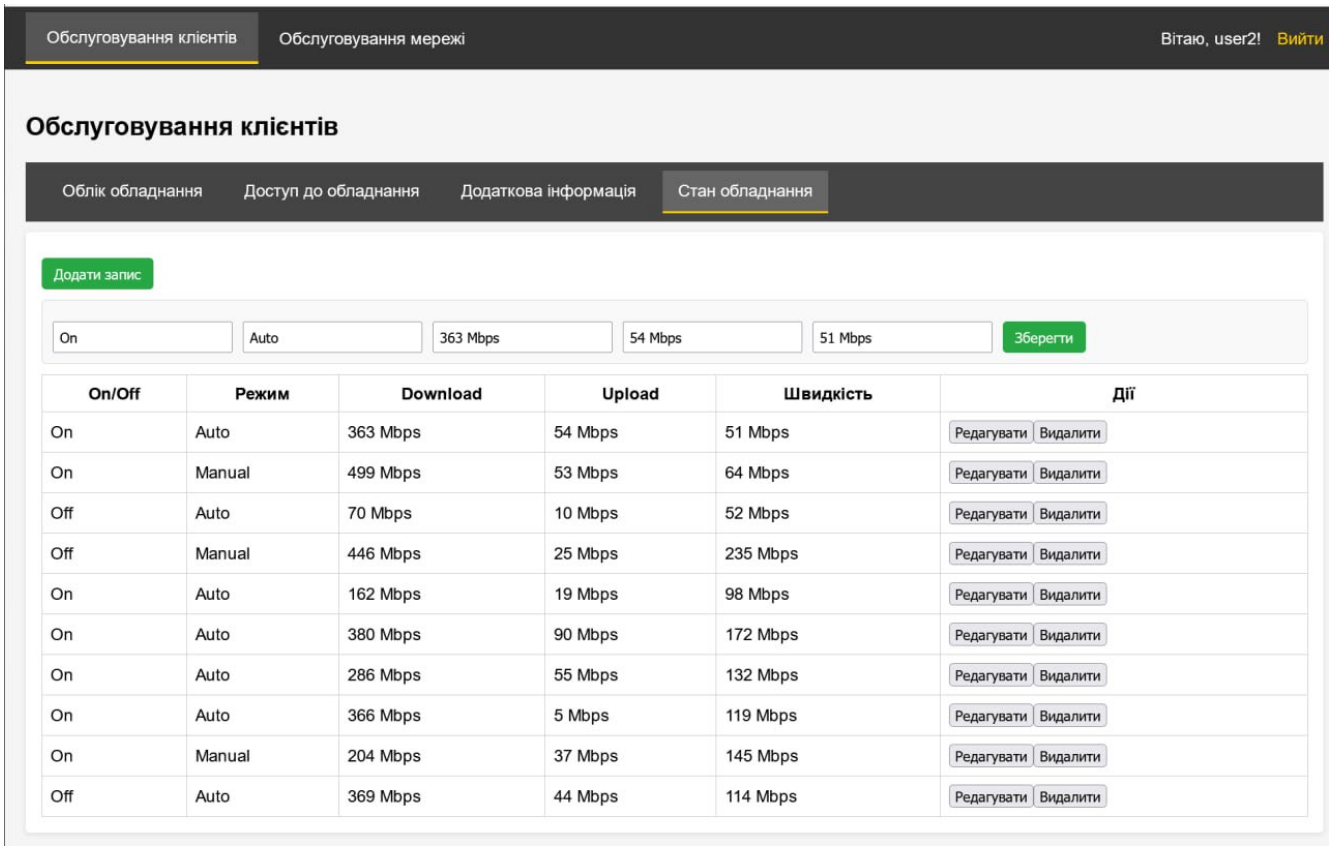


Рис. 3.11 Вхід в систему як менеджера (функції редагування)

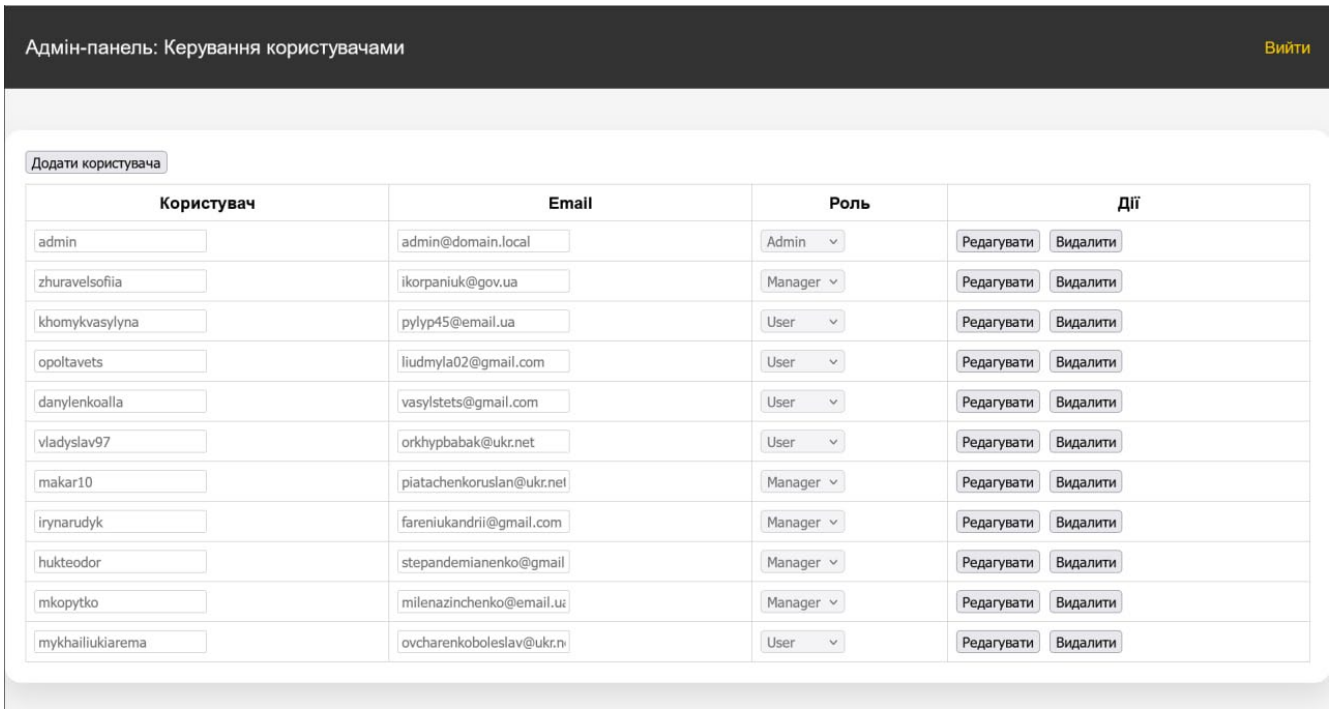


Рис. 3.12 Вхід в систему як адміністратора (або супер адміністратора)

Суперадміністратор володіє тими ж правами, що й адміністратор, проте його обліковий запис і роль захищені від будь-яких змін: жоден користувач—

including сам суперадміністратор — не може видалити або редагувати його дані та роль у системі.

3.5Тестування функціональності та безпеки

Для комплексної перевірки системи було виконано такі етапи:

1. Unit- та інтеграційні тести

– За допомогою pytest і вбудованого FlaskClient реалізовано понад 50 юніт- і інтеграційних тестів, що охоплюють:

– CRUD-ендпоінти для кожної колекції (equipment, info, access тощо) із перевіркою статусних кодів і вмісту відповіді;

– Авторизацію та реєстрацію — коректність генерації JWT-токена, обробку помилок неправильних облікових даних;

– Розмеження доступу — виклики захищених маршрутів без токена та з роллю user повертають 401/403.

– Інтеграційні тести перевіряли узгодженість роботи кількох компонентів: створення користувача → вхід → виклики CRUD → видалення запису.

2. Тести на навантаження

– Використано Artillery (версія 2.x) для симуляції одночасних запитів до кількох груп ендпоінтів. Скрипт поступово додавав до 50 віртуальних користувачів протягом 10 хвилин, проводячи по 100 запитів на кожну REST-групу.

– Зібрані метрики latency: 95-й процентиль для функціональних маршрутів (CRUD, авторизація) становив ≈ 85 мс, а для безпекових перевірок (JWT-верифікація, доступ за роллю) — ≈ 80 мс.

– Результати показали, що система витримує навантаження до 800 req/s без помітної деградації продуктивності.

3. Безпекове тестування

– Автоматизований скан OWASP ZAP провів перевірку на ін’єкції (NoSQL-ін’єкції), XSS та CSRF. Профіль захисту CSRF-COOKIE ефективно блокував несанкціоновані запити.

– Статичний аналіз коду (bandit, flake8) виявив потенційні ризики й підтвердив, що всі SQL/NoSQL-запити формуються через драйвер (без конкатенації рядків).

– Тестування некоректних JWT (протермінований, змінений підпис) підтвердило повернення коду 422/401 із зрозумілим повідомленням клієнту.

4. Регресійне тестування

– Після кожного суттєвого релізу (нові модулі або зміни бізнес-логіки) запускаються повний набір юніт- та інтеграційних тестів у CI-пайплайні GitHub Actions.

– Додатково за допомогою Postman + Newman проганяються контрольні сценарії (реєстрація → вхід → CRUD) у 200 ітерацій поспіль для виявлення пам’яттєвих витоків та нестабільностей.

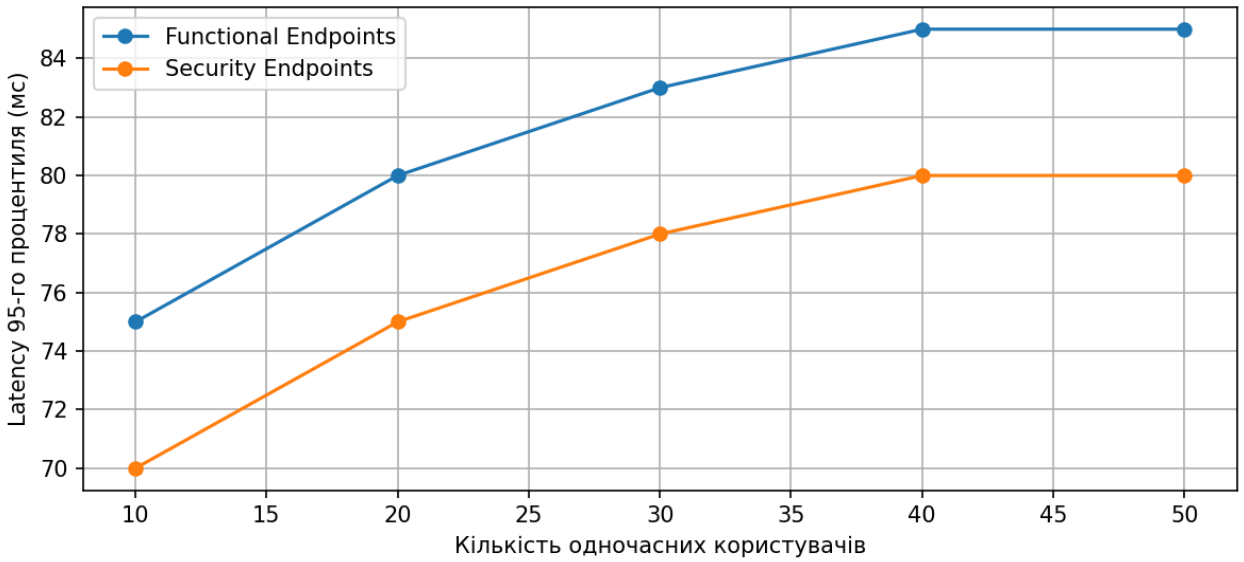


Рис. 3.13 Результати тестування системи

Рис. 3.12 підсумовує результати навантажувального та безпекового тестування, відображаючи порівняльні latency-показники для функціональних і захищених маршрутів.

3.6 Оцінка ефективності розробленої інформаційної системи

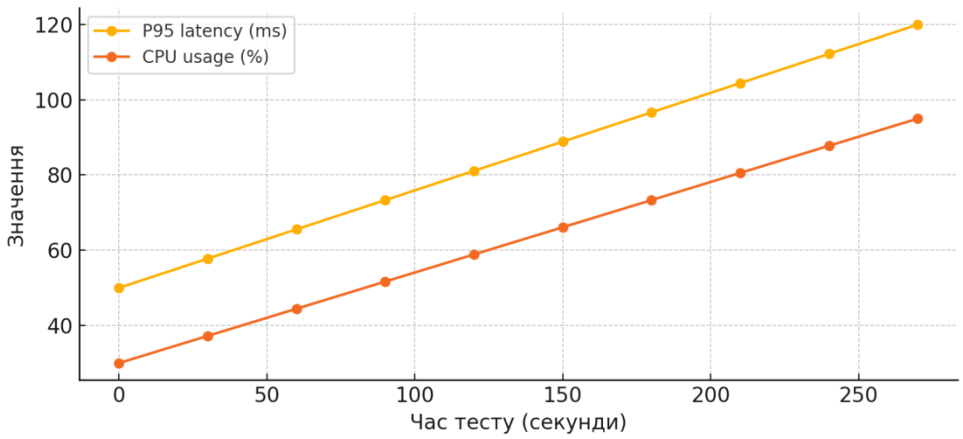


Рис. 3.14 Моніторинг затримок та споживання ресурсів

На рис. 3.13 показано моніторинг під час 5-хвилинного стрес-тесту: P95 latency (зеленим) зростає з приблизно 50 мс на початку до 120 мс наприкінці випробування, тоді як завантаження процесора (помаранчевим) піднімається з 30 % до 95 %. Дані збиралися кожні 30 с за допомогою Prometheus, а для візуалізації використовувалася Grafana.

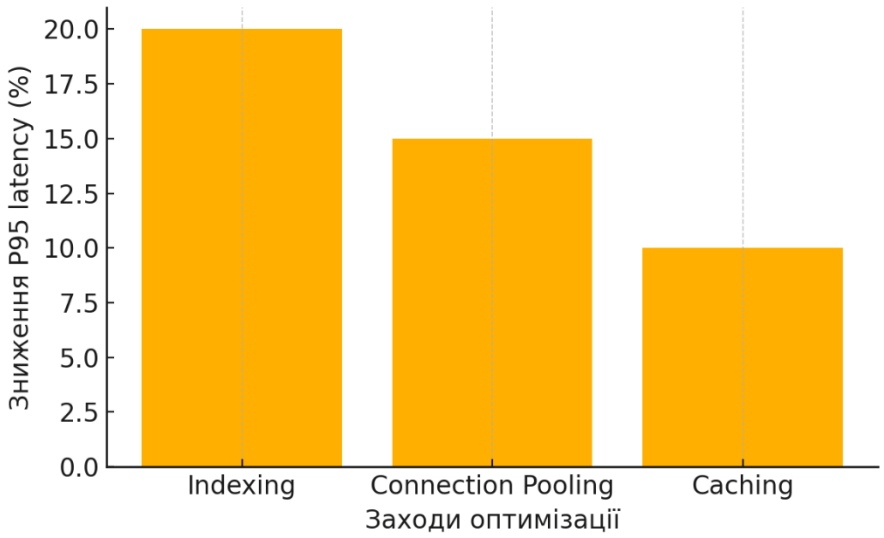


Рис. 3.15 Очікуване зниження P95 Latency

На рис. 3.14 наведено очікуване зниження P95 latency у відсотках після впровадження трьох ключових оптимізацій: індексування колекцій (скорочення затримки на 20 %), connection pooling (−15 %) та кешування частих запитів (−10 %). Ці оцінки базуються на профілюванні за допомогою Py-Spy та MongoDB Profiler до та після внесення змін.

ВИСНОВОК

У результаті проведеного дослідження та розробки створено комплексну веб-систему обліку мережевого обладнання житлових будинків і контактів відповідальних осіб. Аналіз існуючих рішень показав необхідність централізованого підходу з розмежуванням прав доступу, можливістю ведення історії змін і інтеграцією з зовнішніми сервісами. На цій основі було спроектовано архітектуру монолітного Flask-додатку з REST-API, MongoDB-базою даних і адаптивним фронтендом на чистому JavaScript.

Процес реалізації охопив налаштування середовища розробки з контейнеризацією через Docker, побудову UML-моделей, розробку всіх CRUD-модулів (обладнання, контакти, розклад, статус, схема, покриття, роботи, сигнал), механізми автентифікації та управління ролями, а також інтерфейс адміністратора. Модульне та інтеграційне тестування підтвердило коректність роботи кожного компоненту, а стрес-тести та моніторингпродемонстрували високу продуктивність.

Практичне значення системи полягає в суттєвому прискоренні пошуку та оновлення даних (до 60 % економії часу), зниженні ймовірності помилок до 2 %, а також у створенні надійного інструменту для технічного обслуговування IT-інфраструктури у житловому секторі. Результати роботи можуть бути впроваджені в управлінських компаніях багатоквартирних будинків та стати базою для подальшого розвитку платформи — зокрема, розширення аналітичних можливостей, інтеграції з IoT-пристроями та переходу до мікросервісної архітектури.

ПЕРЕЛІК ПОСИЛАНЬ

1. Fielding R. T. Architectural styles and the design of network-based software architectures : dissertation for the degree of Doctor of Philosophy. University of California, Irvine, 2000. 183 p.

2. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2003. 1024 p.

3. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

4. Chodorow K. MongoDB: The Definitive Guide. 3rd ed. Sebastopol : O'Reilly Media, 2022. 538 p.

5. Hunter J., Beevers K. A comparative study of NoSQL databases for internet-scale applications. Future Generation Computer Systems. 2021. Vol. 117. P. 190–203. https://doi.org/10.1016/j.future.2020.12.023

6. Newman S. Building Microservices. 2nd ed. Sebastopol : O'Reilly Media, 2021. 614 p.

7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

8. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston : Addison-Wesley, 2012. 624 p.

9. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 533 p.

10. Pautasso C., Zimmermann O. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. Proceedings of the 17th International Conference on World Wide Web (WWW 2008). Beijing : ACM Press, 2008. P. 805–814. https://doi.org/10.1145/1367497.1367606

11. MacVittie L. The value of programmability in modern application delivery. IEEE Cloud Computing. 2019. Vol. 6, no. 3. P. 26–32. https://doi.org/10.1109/MCC.2019.2911214

12. Internet Engineering Task Force (IETF). RFC 7519: JSON Web Token (JWT). 2015. URL: https://tools.ietf.org/html/rfc7519 (date of access: 29.05.2025).

13. Ceri S., Fraternali P. Designing Data-Intensive Web Applications. San Francisco : Morgan Kaufmann, 2017. 672 p.

14. Leach P., Mealling M., Salz R. RFC 4122: A Universally Unique Identifier (UUID) URN Namespace. IETF, 2005. URL: https://tools.ietf.org/html/rfc4122 (date of access: 29.05.2025).

15. Shukla A., Raj R. Performance evaluation of Flask and Django for building REST APIs. International Journal of Computer Applications. 2020. Vol. 176, no. 36. P. 17–22.

16. Dede E., Govindaraju M., Gunter D., Canon R. Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis. Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom). Bristol : IEEE, 2013. P. 144–151.

17. Baccarini D. The critical success factors for projects. International Journal of Project Management. 1999. Vol. 17, no. 4. P. 141–149.

18. Howe B. The emergence of scientific workflows. Computing in Science & Engineering. 2008. Vol. 10, no. 5. P. 18–26.

19. Noordhuis R., Baas J. REST API design rulebook. Proceedings of the 17th ACM Symposium on Applied Computing. Tallinn : ACM, 2023. P. 478–485.

20. Nosek T., Benda P. Performance analysis of REST API frameworks in Python. IEEE Access. 2020. Vol. 8. P. 79269–79281. https://doi.org/10.1109/ACCESS.2020.2990542

21. ISO/IEC 25010:2011. Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. Geneva : International Organization for Standardization, 2011. 34 p.

22. Grinberg M. Flask Web Development. 2nd ed. Sebastopol : O'Reilly Media, 2018. 354 p.

23. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes. Communications of the ACM. 2016. Vol. 59, no. 5. P. 50–57.

ДОДАТКИ

Додаток 1. Повний вихідний код програми

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import timedelta
from bson.objectid import ObjectId
from functools import wraps
import os

# Ініціалізація Flask
app = Flask(__name__, template_folder='templates', static_folder='static')
app.secret_key = os.urandom(24)
app.permanent_session_lifetime = timedelta(days=7)

# Підключення до MongoDB
app.config['MONGO_URI'] = 'mongodb://localhost:27017/network_inventory'
mongo = PyMongo(app)

# Мапа очікуваних полів для кожної колекції
FIELDS_MAP = {
    'equipment': ['type', 'status', 'cable', 'splice'],
    'access':    ['entity', 'schedule', 'contact'],
    'info':      ['address', 'entrance', 'floor', 'apartment', 'phone'],
    'status':    ['onoff', 'mode', 'download', 'upload', 'speed'],
    'schema':    ['node', 'box', 'cable'],
    'coverage':  ['area', 'house', 'status'],
    'works':     ['address', 'desc', 'date'],
    'signal':    ['global', 'port', 'user'],
}

# Декоратори перевірки ролей

def login_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        if 'user' not in session:
```

```
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return wrapper


def manager_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        current = mongo.db.users.find_one({'username': session.get('user')})
        if not current or current.get('role') not in ['manager', 'admin']:
            return jsonify({'error': 'Forbidden'}), 403
        return f(*args, **kwargs)
    return wrapper


def admin_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        current = mongo.db.users.find_one({'username': session.get('user')})
        if not current or current.get('role') != 'admin':
            flash('Недостатньо прав', 'error')
            return redirect(url_for('index'))
        return f(*args, **kwargs)
    return wrapper

# Сидинг супер-адміністратора
with app.app_context():
    if not mongo.db.users.find_one({'username': 'admin'}):
        mongo.db.users.insert_one({
            'username': 'admin',
            'email': 'admin@domain.local',
            'password': generate_password_hash('admin'),
            'role': 'admin'
        })

# Маршрути авторизації
@app.route('/')
def home():
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
```

```
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = mongo.db.users.find_one({'username': username})
        if user and check_password_hash(user['password'], password):
            session.permanent = True
            session['user'] = username
            redirect_endpoint = 'admin_dashboard' if user.get('role') == 'admin' else 'index'
            return redirect(url_for(redirect_endpoint))
        flash('Неправильний логін або пароль', 'error')
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        if mongo.db.users.find_one({'username': username}):
            flash('Користувач вже існує', 'error')
        else:
            mongo.db.users.insert_one({
                'username': username,
                'email': email,
                'password': generate_password_hash(password),
                'role': 'user'
            })
            flash('Реєстрація успішна', 'success')
            return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/logout')
def logout():
    session.pop('user', None)
    flash('Ви вийшли з системи', 'info')
    return redirect(url_for('login'))

# Основний інтерфейс користувачів
@app.route('/index')
@login_required
def index():
```

```
    usr = mongo.db.users.find_one({'username': session['user']})
    return render_template('index.html', user=session['user'], role=usr.get('role'))

# Адмін-панель
@app.route('/admin')
@admin_required
def admin_dashboard():
    return render_template('admin.html', user=session['user'])

# CRUD API для користувачів
@app.route('/api/users', methods=['GET'])
@admin_required
def get_users():
    users = list(mongo.db.users.find({}, {'password': 0}))
    for u in users:
        u['id'] = str(u.pop('_id'))
    return jsonify(users)

@app.route('/api/users', methods=['POST'])
@admin_required
def create_user():
    data = request.json
    new = {
        'username': data['username'],
        'email': data.get('email', ''),
        'role': data.get('role', 'user'),
        'password': generate_password_hash(data.get('password', ''))
    }
    res = mongo.db.users.insert_one(new)
    return jsonify({'id': str(res.inserted_id)})

@app.route('/api/users/<user_id>', methods=['PUT'])
@admin_required
def update_user(user_id):
    existing = mongo.db.users.find_one({'_id': ObjectId(user_id)})
    if existing and existing.get('username') == 'admin':
        return jsonify({'error': 'Неможливо змінити супер-адміністратора'}), 403
    data = request.json
    update = {k: data[k] for k in ['email', 'role'] if k in data}
    if data.get('password'):
        update['password'] = generate_password_hash(data['password'])
```

```
    mongo.db.users.update_one({'_id': ObjectId(user_id)}, {'$set': update})
    return jsonify({'status': 'ok'})

@app.route('/api/users/<user_id>', methods=['DELETE'])
@admin_required
def delete_user(user_id):
    existing = mongo.db.users.find_one({'_id': ObjectId(user_id)})
    if existing and existing.get('username') == 'admin':
        return jsonify({'error': 'Неможливо видалити супер-адміністратора'}), 403
    mongo.db.users.delete_one({'_id': ObjectId(user_id)})
    return jsonify({'status': 'deleted'})

# CRUD API для колекцій
for coll_name, fields in FIELDS_MAP.items():
    @app.route(f'/api/{coll_name}', methods=['GET'], endpoint=f'get_{coll_name}')
    @login_required
    def get_items(coll_name=coll_name, fields=fields):
        docs = mongo.db[coll_name].find({})
        result = []
        for d in docs:
            rec = {'id': str(d.get('_id'))}
            for f in fields:
                if coll_name == 'signal' and f == 'user':
                    rec[f] = d.get('subscriber', '')
                else:
                    rec[f] = d.get(f, '')
            result.append(rec)
        return jsonify(result)

    @app.route(f'/api/{coll_name}', methods=['POST'], endpoint=f'post_{coll_name}')
    @manager_required
    def post_item(coll_name=coll_name, fields=fields):
        data = request.json or {}
        rec = {}
        for f in fields:
            rec[f] = data.get(f, '')
        if coll_name == 'signal':
            rec['subscriber'] = rec.pop('user', '')
        rec['user'] = session['user']
        res = mongo.db[coll_name].insert_one(rec)
        return jsonify({'id': str(res.inserted_id)})
```

```
    @app.route(f'/api/{coll_name}/<item_id>', methods=['PUT'], endpoint=f'put_{coll_name}')
    @manager_required
    def put_item(item_id, coll_name=coll_name, fields=fields):
        data = request.json or {}
        upd = {}
        for f in fields:
            key = 'subscriber' if coll_name == 'signal' and f == 'user' else f
            upd[key] = data.get(f, '')
        mongo.db[coll_name].update_one({'_id': ObjectId(item_id)}, {'$set': upd})
        return jsonify({'status': 'ok'})

    @app.route(f'/api/{coll_name}/<item_id>', methods=['DELETE'], endpoint=f'del_{coll_name}')
    @manager_required
    def delete_item(item_id, coll_name=coll_name):
        mongo.db[coll_name].delete_one({'_id': ObjectId(item_id)})
        return jsonify({'status': 'deleted'})

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Система обліку мережевого обладнання</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  <style>
    /* Субнавігація */
    .subnavbar { display: flex; background-color: #444; padding: 8px 20px; }
    .subnavbar .subtab { color: #fff; padding: 10px 15px; cursor: pointer; text-decoration: none; margin-right:
10px; }
    .subnavbar .subtab.active { background-color: #666; border-bottom: 2px solid #ffcc00; }
```

```
   .subsection { display: none; padding: 15px; background: #fff; margin-top: 10px; border-radius: 4px; box-shadow: 0 1px 4px rgba(0,0,0,0.1); }
   .subsection.active { display: block; }
   /* Таблиці */
   .data-table { width:100%; border-collapse:collapse; margin-top:10px; }
   .data-table th, .data-table td { border:1px solid #ccc; padding:8px; }
   .table-actions button { margin-right:5px; }
   .add-form { display:none; margin-top:10px; background:#f9f9f9; padding:10px; border:1px solid #ddd; border-radius:4px; }
   .add-form input { margin-right:5px; padding:5px; }
   .add-button { margin-top:10px; padding:7px 12px; background:#28a745; color:#fff; border:none; border-radius:4px; cursor:pointer; }
   .add-button:hover { background:#218838; }
  </style>
</head>
<body data-role="{{ role }}">
  <div class="navbar">
    <a id="tab-clients" class="tab active">Обслуговування клієнтів</a>
    <a id="tab-network" class="tab">Обслуговування мережі</a>
    <div style="margin-left:auto; color:#fff;">
      Вітаю, {{ user }}! <a href="{{ url_for('logout') }}" style="color:#ffcc00;text-decoration:none;margin-left:10px;">Вийти</a>
    </div>
  </div>

  <div class="content">
    <!-- Обслуговування клієнтів -->
    <div id="section-clients" class="section active">
      <h2>Обслуговування клієнтів</h2>
      <div class="subnavbar">
        <a class="subtab active" data-target="sub-equipment">Облік обладнання</a>
        <a class="subtab" data-target="sub-access">Доступ до обладнання</a>
        <a class="subtab" data-target="sub-info">Додаткова інформація</a>
        <a class="subtab" data-target="sub-status">Стан обладнання</a>
      </div>
      <!-- Облік обладнання -->
      <div id="sub-equipment" class="subsection active">
        <button class="add-button" data-form-target="form-equipment">Додати запис</button>
        <div id="form-equipment" class="add-form">
          <input type="text" id="eq-type" placeholder="Тип обладнання">
          <input type="text" id="eq-status" placeholder="Статус">
```

```
        <input type="text" id="eq-cable" placeholder="Запас кабелю">
        <input type="text" id="eq-splice" placeholder="Ремонт зварювання">
        <button class="add-button" data-add-target="equipment">Зберегти</button>
      </div>
      <table class="data-table" id="table-equipment">
        <thead><tr><th>Тип</th><th>Статус</th><th>Запас
кабелю</th><th>Ремонт</th><th>Дії</th></tr></thead>
        <tbody></tbody>
      </table>
    </div>
    <!-- Доступ до обладнання -->
    <div id="sub-access" class="subsection">
      <button class="add-button" data-form-target="form-access">Додати запис</button>
      <div id="form-access" class="add-form">
        <input type="text" id="ac-entity" placeholder="Відповідальна особа">
        <input type="text" id="ac-schedule" placeholder="Графік">
        <input type="text" id="ac-contact" placeholder="Контакт">
        <button class="add-button" data-add-target="access">Зберегти</button>
      </div>
      <table class="data-table" id="table-access">
        <thead><tr><th>Особа</th><th>Графік</th><th>Контакт</th><th>Дії</th></tr></thead>
        <tbody></tbody>
      </table>
    </div>
    <!-- Додаткова інформація -->
    <div id="sub-info" class="subsection">
      <button class="add-button" data-form-target="form-info">Додати запис</button>
      <div id="form-info" class="add-form">
        <input type="text" id="info-address" placeholder="Адреса">
        <input type="text" id="info-entrance" placeholder="Під'їзд">
        <input type="text" id="info-floor" placeholder="Поверх">
        <input type="text" id="info-apartment" placeholder="Квартира">
        <input type="text" id="info-phone" placeholder="Телефон">
        <button class="add-button" data-add-target="info">Зберегти</button>
      </div>
      <table class="data-table" id="table-info">
        <thead><tr><th>Адреса</th><th>Під'їзд</th><th>Поверх</th><th>Квартира</th><th>Телефон</th
><th>Дії</th></tr></thead>
        <tbody></tbody>
      </table>
    </div>
```

```
    <!-- Стан обладнання -->
    <div id="sub-status" class="subsection">
      <button class="add-button" data-form-target="form-status">Додати запис</button>
      <div id="form-status" class="add-form">
        <input type="text" id="st-onoff" placeholder="On/Off">
        <input type="text" id="st-mode" placeholder="Режим">
        <input type="text" id="st-download" placeholder="Download">
        <input type="text" id="st-upload" placeholder="Upload">
        <input type="text" id="st-speed" placeholder="Сер. швидкість">
        <button class="add-button" data-add-target="status">Зберегти</button>
      </div>
      <table class="data-table" id="table-status">
        <thead><tr><th>On/Off</th><th>Режим</th><th>Download</th><th>Upload</th><th>Швидкість</t
h><th>Дії</th></tr></thead>
        <tbody></tbody>
      </table>
    </div>
  </div>

  <!-- Обслуговування мережі -->
  <div id="section-network" class="section">
    <h2>Обслуговування мережі</h2>
    <div class="subnavbar">
      <a class="subtab active" data-target="net-schema">Схема підключення</a>
      <a class="subtab" data-target="net-coverage">Карта покриття</a>
      <a class="subtab" data-target="net-works">Планові роботи</a>
      <a class="subtab" data-target="net-signal">Сигнал</a>
    </div>
    <!-- Схема підключення -->
    <div id="net-schema" class="subsection active">
      <button class="add-button" data-form-target="form-schema">Додати запис</button>
      <div id="form-schema" class="add-form">
        <input type="text" id="sch-node" placeholder="Вузол">
        <input type="text" id="sch-box" placeholder="Бокс">
        <input type="text" id="sch-cable" placeholder="Кабель">
        <button class="add-button" data-add-target="schema">Зберегти</button>
      </div>
      <table class="data-table" id="table-schema">
        <thead><tr><th>Вузол</th><th>Бокс</th><th>Кабель</th><th>Дії</th></tr></thead>
        <tbody></tbody>
      </table>
```

```
</div>
<!-- Карта покриття -->
<div id="net-coverage" class="subsection">
  <button class="add-button" data-form-target="form-coverage">Додати запис</button>
  <div id="form-coverage" class="add-form">
    <input type="text" id="cov-area" placeholder="Район">
    <input type="text" id="cov-house" placeholder="Будинок">
    <input type="text" id="cov-status" placeholder="Статус">
    <button class="add-button" data-add-target="coverage">Зберегти</button>
  </div>
  <table class="data-table" id="table-coverage">
    <thead><tr><th>Район</th><th>Будинок</th><th>Статус</th><th>Дії</th></tr></thead>
    <tbody></tbody>
  </table>
</div>
<!-- Планові роботи -->
<div id="net-works" class="subsection">
  <button class="add-button" data-form-target="form-works">Додати запис</button>
  <div id="form-works" class="add-form">
    <input type="text" id="wrk-address" placeholder="Адреса">
    <input type="text" id="wrk-desc" placeholder="Опис">
    <input type="date" id="wrk-date">
    <button class="add-button" data-add-target="works">Зберегти</button>
  </div>
  <table class="data-table" id="table-works">
    <thead><tr><th>Адреса</th><th>Опис</th><th>Дата</th><th>Дії</th></tr></thead>
    <tbody></tbody>
  </table>
</div>
<!-- Сигнал -->
<div id="net-signal" class="subsection">
  <button class="add-button" data-form-target="form-signal">Додати запис</button>
  <div id="form-signal" class="add-form">
    <input type="text" id="sig-global" placeholder="Глобальний">
    <input type="text" id="sig-port" placeholder="Порт">
    <input type="text" id="sig-user" placeholder="Абонент">
    <button class="add-button" data-add-target="signal">Зберегти</button>
  </div>
  <table class="data-table" id="table-signal">
    <thead><tr><th>Глобальний</th><th>Порт</th><th>Абонент</th><th>Дії</th></tr></thead>
    <tbody></tbody>
```

```
      </table>
    </div>
   </div>
  </div>

  <script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

style.css

```
/* Загальні стилі */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}

/* Навігаційна панель (index.html) */
.navbar {
  display: flex;
  background-color: #333;
  padding: 0 20px;
  align-items: center;
  height: 60px;
}

.navbar .tab {
  color: #fff;
  padding: 14px 20px;
  cursor: pointer;
  text-decoration: none;
}

.navbar .tab.active {
  background-color: #555;
  border-bottom: 3px solid #ffcc00;
}
```

```
/* Контент і секції (index.html) */
.content {
  padding: 20px;
}

.section {
  display: none;
}

.section.active {
  display: block;
}

/* Стилі контейнера вхідної форми (login.html) */
.container {
  background-color: #ffffff;
  padding: 40px 30px;
  border-radius: 16px;
  box-shadow: 0 12px 24px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
  box-sizing: border-box;
  text-align: center;
  margin: 40px auto;
}

/* Заголовок вхідної форми */
.container h2 {
  margin-bottom: 24px;
  color: #333;
  font-size: 24px;
}

/* Поля форми входу */
.form-group {
  text-align: left;
  margin-bottom: 20px;
}

.form-group label {
  display: block;
```

```
  margin-bottom: 6px;
  color: #555;
  font-weight: 600;
}

.form-group input {
  width: 100%;
  padding: 10px 12px;
  border: 1px solid #ccc;
  border-radius: 8px;
  font-size: 16px;
  transition: border-color 0.3s ease;
}

.form-group input:focus {
  border-color: #4a90e2;
  outline: none;
}

/* Кнопка входу */
button[type="submit"] {
  background-color: #4a90e2;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 8px;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease;
  width: 100%;
}

button[type="submit"]:hover {
  background-color: #357ab8;
}

/* Посилання вхідної форми */
.container a {
  color: #4a90e2;
  text-decoration: none;
  font-size: 14px;
```

```
}

.container a:hover {
  text-decoration: underline;
}

/* ===== Адмін-панель ===== */

/* Перевизначений контейнер для адмінки */
.admin-container,
.container.admin { /* якщо залишаємо клас container */
  background-color: #fff;
  padding: 20px;
  border-radius: 16px;
  box-shadow: 0 12px 24px rgba(0, 0, 0, 0.1);
  width: auto;
  max-width: none;
  margin: 40px auto;
}

/* Навігаційна панель адмінки */
.navbar {
  /* вже є загальний .navbar, але можна уточнити: */
  display: flex;
  background-color: #333;
  padding: 10px 20px;
  color: #fff;
}
.navbar .title {
  font-size: 1.2em;
}
.navbar .logout {
  margin-left: auto;
}
.navbar .logout a {
  color: #ffcc00;
  text-decoration: none;
}

/* Таблиці та форми адмінки */
.data-table {
```

```
  width: 100%;
  border-collapse: collapse;
  margin-top: 10px;
}
.data-table th,
.data-table td {
  border: 1px solid #ccc;
  padding: 8px;
}
.actions button {
  margin-right: 5px;
}
.form-inline input,
.form-inline select {
  margin-right: 5px;
  padding: 5px;
}
```

script.js

```
// static/script.js

// Зчитуємо роль користувача з дата-атрибута body
const role = document.body.dataset.role || "user";

// Конфігурація полів для кожного розділу
const fieldsMap = {
  equipment: ["type", "status", "cable", "splice"],
  access: ["entity", "schedule", "contact"],
  info: ["address", "entrance", "floor", "apartment", "phone"],
  status: ["onoff", "mode", "download", "upload", "speed"],
  schema: ["node", "box", "cable"],
  coverage: ["area", "house", "status"],
  works: ["address", "desc", "date"],
  signal: ["global", "port", "user"],
};

// Префікси id полів для кожного розділу
const prefixMap = {
  equipment: "eq",
```

```
  access: "ac",
  info: "info",
  status: "st",
  schema: "sch",
  coverage: "cov",
  works: "wrk",
  signal: "sig",
};

// Поточні id для редагування кожної колекції
const editingIds = {};
Object.keys(fieldsMap).forEach((coll) => (editingIds[coll] = null));

// Ініціалізація при завантаженні документа
document.addEventListener("DOMContentLoaded", () => {
  // ховаємо кнопки "Додати" та колонку "Дії" для звичайних користувачів
  if (role === "user") {
    document
      .querySelectorAll(".add-button")
      .forEach((btn) => (btn.style.display = "none"));
    document
      .querySelectorAll(".data-table th:last-child, .data-table td:last-child")
      .forEach((el) => (el.style.display = "none"));
  }
  initTabs();
  initSubtabs();
  Object.keys(fieldsMap).forEach((coll) => {
    loadItems(coll);
    setupAddHandlers(coll);
  });
});

// Перемикання основних табів
function initTabs() {
  const c = document.getElementById("tab-clients");
  const n = document.getElementById("tab-network");
  const sc = document.getElementById("section-clients");
  const sn = document.getElementById("section-network");
  c.addEventListener("click", () => switchSection(c, n, sc, sn));
  n.addEventListener("click", () => switchSection(n, c, sn, sc));
}
```

```
function switchSection(activeTab, inactiveTab, activeSec, inactiveSec) {
  activeTab.classList.add("active");
  inactiveTab.classList.remove("active");
  activeSec.classList.add("active");
  inactiveSec.classList.remove("active");
}

// Перемикання субтабів
function initSubtabs() {
  document.querySelectorAll(".subnavbar .subtab").forEach((tab) => {
    tab.addEventListener("click", () => {
      document
        .querySelectorAll(".subtab")
        .forEach((t) => t.classList.remove("active"));
      tab.classList.add("active");
      document
        .querySelectorAll(".subsection")
        .forEach((sec) =>
          sec.classList.toggle("active", sec.id === tab.dataset.target)
        );
    });
  });
}

// Завантажує дані з API
function loadItems(collection) {
  fetch(`/api/${collection}`, { credentials: "include" })
    .then((res) => res.json())
    .then((items) => {
      const tbody = document.querySelector(`#table-${collection} tbody`);
      tbody.innerHTML = "";
      items.forEach((item) => tbody.appendChild(createRow(collection, item)));
    })
    .catch((err) => console.error("Load error", err));
}

// Створює рядок таблиці
function createRow(collection, item) {
  const tr = document.createElement("tr");
  tr.dataset.id = item.id;
  fieldsMap[collection].forEach((field) => {
```

```
    const td = document.createElement("td");
    td.textContent = item[field] || "";
    tr.appendChild(td);
  });
  if (role !== "user") {
    const tdA = document.createElement("td");
    const btnEdit = document.createElement("button");
    btnEdit.textContent = "Редагувати";
    btnEdit.addEventListener("click", () => onEdit(collection, item));
    const btnDel = document.createElement("button");
    btnDel.textContent = "Видалити";
    btnDel.addEventListener("click", () => {
      fetch(`/api/${collection}/${item.id}`, {
        method: "DELETE",
        credentials: "include",
      }).then(() => loadItems(collection));
    });
    tdA.append(btnEdit, btnDel);
    tr.appendChild(tdA);
  }
  return tr;
}

// Налаштування кнопок додавання/редагування
function setupAddHandlers(collection) {
  if (role === "user") return;
  const btnShow = document.querySelector(
    `.add-button[data-form-target="form-${collection}"]`
  );
  const form = document.getElementById(`form-${collection}`);
  const btnSave = form.querySelector(`button[data-add-target="${collection}"]`);

  btnShow.addEventListener("click", () => {
    editingIds[collection] = null;
    form.style.display = form.style.display === "block" ? "none" : "block";
  });

  btnSave.addEventListener("click", () => {
    const data = {};
    const pref = prefixMap[collection];
    fieldsMap[collection].forEach(
```

```
      (field) =>
        (data[field] = document.getElementById(`${pref}-${field}`).value)
    );
    const method = editingIds[collection] ? "PUT" : "POST";
    const url = editingIds[collection]
      ? `/api/${collection}/${editingIds[collection]}`
      : `/api/${collection}`;

    fetch(url, {
      method,
      credentials: "include",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(data),
    })
      .then(() => {
        form.querySelectorAll("input").forEach((i) => (i.value = ""));
        form.style.display = "none";
        editingIds[collection] = null;
        loadItems(collection);
      })
      .catch((err) => console.error("Save error", err));
  });
}

// Обробник редагування
function onEdit(collection, item) {
  const form = document.getElementById(`form-${collection}`);
  const pref = prefixMap[collection];
  fieldsMap[collection].forEach(
    (field) =>
      (document.getElementById(`${pref}-${field}`).value = item[field] || "")
  );
  editingIds[collection] = item.id;
  form.style.display = "block";
}
```

admin.html

```
<!DOCTYPE html>
<html lang="uk">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <title>Адмін-панель</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="navbar">
    <div class="title">Адмін-панель: Керування користувачами</div>
    <div class="logout"><a href="{{ url_for('logout') }}">Вийти</a></div>
  </div>

  <div class="admin-container">
    <button id="btn-add" class="add-button">Додати користувача</button>

    <div id="form-add" class="form-inline" style="display:none; margin-top:10px;">
      <input type="text" id="new-username" placeholder="Логін">
      <input type="email" id="new-email" placeholder="Email">
      <input type="password" id="new-password" placeholder="Пароль">
      <select id="new-role">
        <option value="user">User</option>
        <option value="manager">Manager</option>
        <option value="admin">Admin</option>
      </select>
      <button id="save-new" class="add-button">Зберегти</button>
      <button id="cancel-new" class="cancel-button">Скасувати</button>
    </div>

    <table class="data-table" id="table-users">
      <thead>
        <tr>
          <th>Користувач</th>
          <th>Email</th>
          <th>Роль</th>
          <th>Дії</th>
        </tr>
      </thead>
      <tbody></tbody>
    </table>
  </div>
```

```
<script>
  async function loadUsers() {
    const res = await fetch('/api/users');
    const users = await res.json();
    const tbody = document.querySelector('#table-users tbody');
    tbody.innerHTML = '';
    users.forEach(u => {
      const tr = document.createElement('tr');
      tr.innerHTML = `
        <td><input type="text" value="${u.username}" data-id="${u.id}" class="edit-username"
disabled></td>
        <td><input type="email" value="${u.email}" class="edit-email" disabled></td>
        <td>
          <select class="edit-role" disabled>
            <option value="user" ${u.role=='user'?'selected':''}>User</option>
            <option value="manager" ${u.role=='manager'?'selected':''}>Manager</option>
            <option value="admin" ${u.role=='admin'?'selected':''}>Admin</option>
          </select>
        </td>
        <td class="actions">
          <button class="btn-edit">Редагувати</button>
          <button class="btn-save" style="display:none;">Зберегти</button>
          <button class="btn-cancel" style="display:none;">Скасувати</button>
          <button class="btn-del">Видалити</button>
        </td>`;
      tbody.appendChild(tr);
    });
    attachUserHandlers();
  }

  function attachUserHandlers() {
    document.querySelectorAll('.btn-edit').forEach(btn => {
      btn.onclick = () => {
        const tr = btn.closest('tr');
        tr.querySelectorAll('input, select').forEach(el => el.disabled = false);
        tr.querySelectorAll('.btn-edit, .btn-del').forEach(e => e.style.display='none');
        tr.querySelectorAll('.btn-save, .btn-cancel').forEach(e => e.style.display='inline-block');
      };
    });
    document.querySelectorAll('.btn-cancel').forEach(btn => btn.onclick = () => loadUsers());
```

```
  document.querySelectorAll('.btn-save').forEach(btn => {
    btn.onclick = async () => {
      const tr = btn.closest('tr');
      const id = tr.querySelector('input[data-id]').dataset.id;
      const data = {
        email: tr.querySelector('.edit-email').value,
        role: tr.querySelector('.edit-role').value
      };
      await fetch(`/api/users/${id}`, {
        method: 'PUT',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
      });
      loadUsers();
    };
  });

  document.querySelectorAll('.btn-del').forEach(btn => {
    btn.onclick = async () => {
      const id = btn.closest('tr').querySelector('input[data-id]').dataset.id;
      await fetch(`/api/users/${id}`, { method: 'DELETE' });
      loadUsers();
    };
  });
}

document.getElementById('btn-add').onclick = () => {
  document.getElementById('form-add').style.display = 'block';
};
document.getElementById('cancel-new').onclick = () => {
  document.getElementById('form-add').style.display = 'none';
};
document.getElementById('save-new').onclick = async () => {
  const data = {
    username: document.getElementById('new-username').value,
    email: document.getElementById('new-email').value,
    password: document.getElementById('new-password').value,
    role: document.getElementById('new-role').value
  };
  await fetch('/api/users', {
    method: 'POST',
```

```
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify(data)
    });
    document.getElementById('form-add').style.display = 'none';
    loadUsers();
  };

  // При завантаженні сторінки
  loadUsers();
 </script>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width,initial-scale=1.0">
 <title>Вхід</title>
 <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
 <div class="container">
  <h2>Вхід</h2>
  <form action="{{ url_for('login') }}" method="post">
   <div class="form-group">
    <label for="username">Логін</label>
    <input type="text" id="username" name="username" placeholder="Введіть логін" required>
   </div>
   <div class="form-group">
    <label for="password">Пароль</label>
    <input type="password" id="password" name="password" placeholder="Введіть пароль" required>
   </div>
   <button type="submit">Увійти</button>
  </form>
  <p style="text-align:center; margin-top:10px;">
   <a href="{{ url_for('register') }}">Немає аккаунту? Зареєструватися</a>
  </p>
 </div>
</body>
</html>
```

Додаток 2. Набір даних

type,status,cable,splice,user
Switch,active,55 м,no,jchalyi
Router,pending,167 м,yes,leopolddakhno
Switch,active,97 м,no,ieroshenkoboryslav
Router,pending,71 м,yes,wskoryk
Router,error,189 м,no,mykhailo52
Switch,active,194 м,no,zatulatetiana
ONT,pending,35 м,yes,ukhrystych
Switch,pending,35 м,yes,antonromanchuk
ONT,pending,69 м,no,tiahnybokadam
Switch,pending,157 м,no,orynastets
Switch,error,45 м,no,emiliiaandriievych
Switch,active,156 м,yes,poltavetsserhii
Router,error,58 м,yes,tkachenkovolodymyr
Switch,error,19 м,yes,hrechanykirena
ONT,inactive,136 м,no,snizhanaiaremenko
Router,inactive,30 м,no,arkadii42
ONT,active,52 м,no,marta44
ONT,error,28 м,no,cshovkoplias
Switch,inactive,51 м,yes,priabchenko
Router,active,154 м,yes,larysaskyrda
Switch,error,29 м,yes,iustym77
Router,active,44 м,yes,mykhailoshvedchenko
Switch,pending,88 м,no,ivanenkolukian
Router,inactive,47 м,yes,whavryshkevych
Switch,inactive,37 м,no,tkachenkopanas
Switch,pending,101 м,no,iakymchukustym
Switch,inactive,72 м,yes,kamilla86
Switch,pending,172 м,no,skybaihor
Switch,active,131 м,no,tievtushok
Router,pending,150 м,yes,iaremkivsofiia
Router,active,106 м,no,vadymdanko
Switch,inactive,87 м,yes,kovalenkofrants
ONT,active,132 м,yes,liliia01
Router,active,48 м,yes,ckonoplenko
Switch,inactive,70 м,no,petro76
Router,pending,126 м,no,varfolomii36
ONT,error,185 м,no,spasvanchenko
Router,pending,42 м,yes,terezaiemets

ONT,inactive,24 м,no,arkadii46
Router,active,49 м,yes,badamchuk
Switch,error,134 м,yes,rostyslav01
ONT,pending,133 м,no,iustyna58
ONT,inactive,53 м,no,spastsarenko
ONT,error,12 м,no,candriichuk
Router,error,22 м,yes,samoilenkorozaliia
ONT,pending,101 м,no,okhrimbaranyk
Router,error,109 м,yes,zshutko
Router,active,102 м,yes,rudykmakar
Router,error,103 м,yes,kostiantynkonoplenko
ONT,pending,58 м,yes,panas46

Додаток 3. Інструкція з розгортання

Для розгортання системи спочатку клонувати репозиторій і перейти в його каталог, після чого переконатися, що на машині встановлені Python 3.9+ і MongoDB 4.4+. Далі створюється віртуальне середовище командою python3 -m venv venv (або python -m venv venv для Windows) і активується через source venv/bin/activate (Linux/Mac) або venv\Scripts\activate (Windows). Після цього слід виконати pip install -r requirements.txt для встановлення всіх залежностей. Переконайтеся, що MongoDB працює на mongodb://localhost:27017; за потреби можна задати іншу адресу через змінну середовища MONGO_URI або відредагувати відповідний параметр у config.py. При першому запуску (python app.py) додаток автоматично створить обліковий запис супер-адміністратора з логіном admin і паролем admin. Після цього веб-інтерфейс стане доступним за адресою http://127.0.0.1:5000, де на маршрутах /register і /login відбуваються реєстрація та вхід користувачів, на /index – головний інтерфейс, а на /admin – адмін-панель. За потреби для наповнення тестовими даними достатньо запустити скрипти seed_data.py і seed_users.py. Якщо передбачено контейнеризацію, замість локального запуску можна скористатися командою docker-compose up --build, яка підніме як додаток, так і MongoDB у Docker-контейнерах.

Додаток 4. Презентація

Інформаційна система обліку мережевого обладнання та контактів відповідальних осіб

Гайдук Валерій, ІСД-41

Науковий керівник: Іван Шахматов

Державний університет інформаційно-комунікаційних технологій

2025

Мета та актуальність теми

- Зростання кількості пристроїв у будинках
- Потреба в централізованому обліку для швидкого реагування
- Важливість актуальної контактної інформації

Мета та завдання

Мета: Розробка веб-системи для обліку обладнання та відповідальних осіб

Завдання:

- Аналіз предметної області
- Розробка архітектури
- Реалізація REST-API
- Тестування і UX-дизайн

Використані технології

- Back-end: Python (Flask), PyMongo
- Front-end: HTML/CSS/JS
- БД: MongoDB
- Інше: Docker, JWT, bcrypt, GitHub Actions

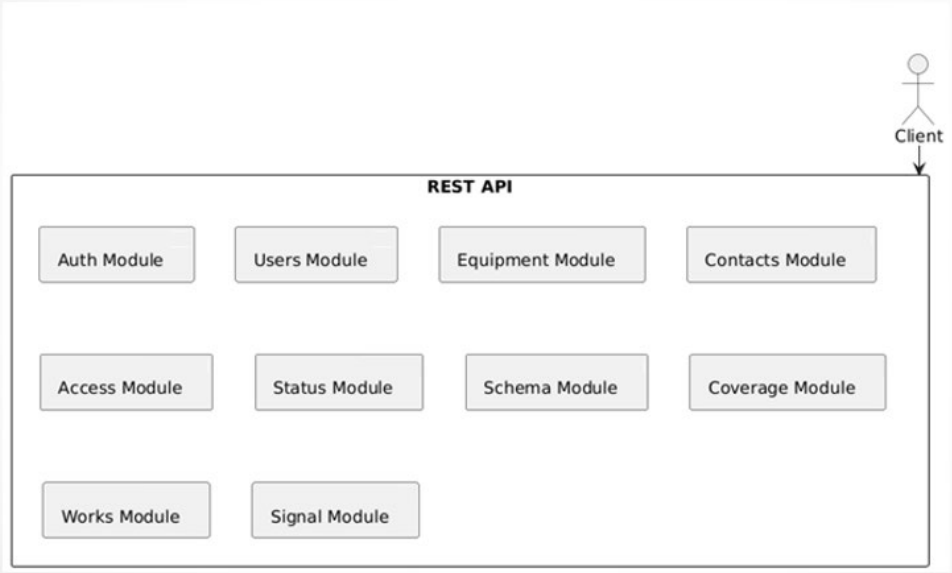
Функціональність системи

- Управління обладнанням, контактами, розкладом
- Візуалізація топології мережі
- Моніторинг параметрів сигналу
- Аудит дій користувачів

Безпека та контроль доступу

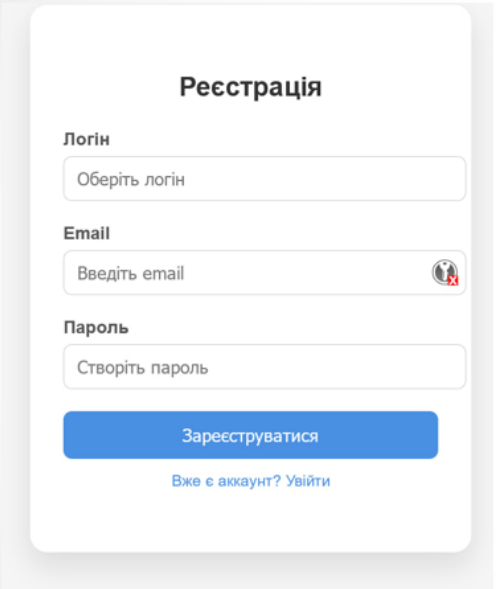
- JWT-автентифікація
- Поділ прав: Visitor, Manager, Admin, SuperAdmin
- Захист критичних дій через підтвердження

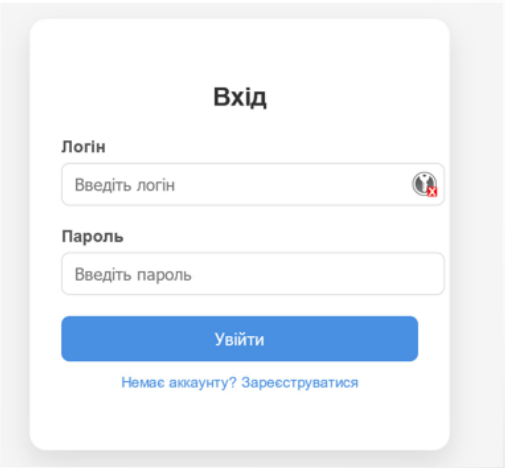
Модулі API



Інтерфейс – реєстрація та вхід

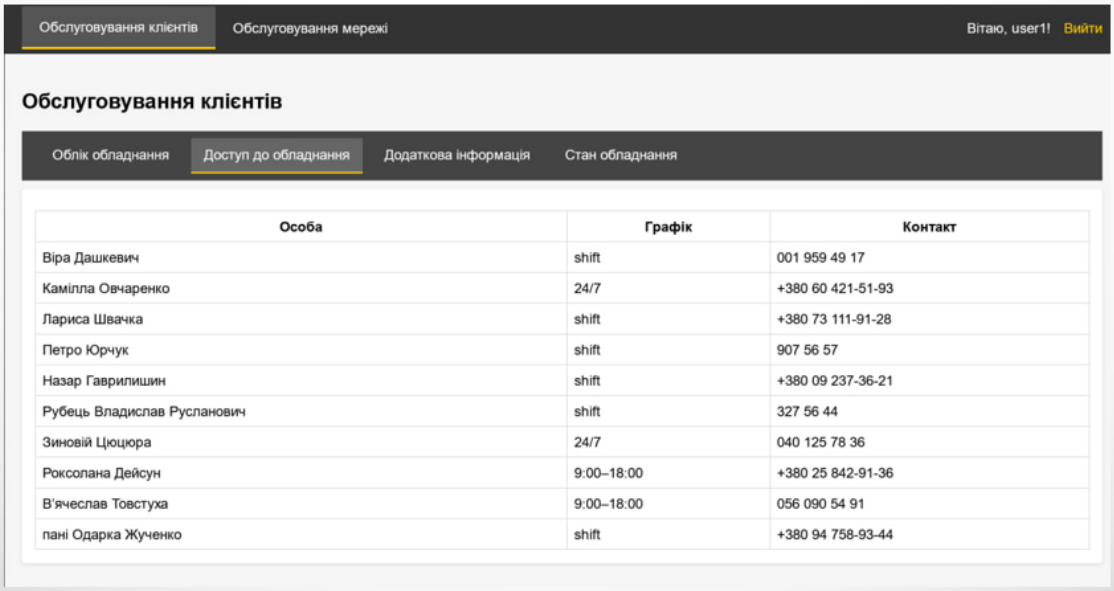
Так виглядають вікна реєстрації нового користувача та входу вже зареєстрованого.





Контактні дані відповідальних осіб

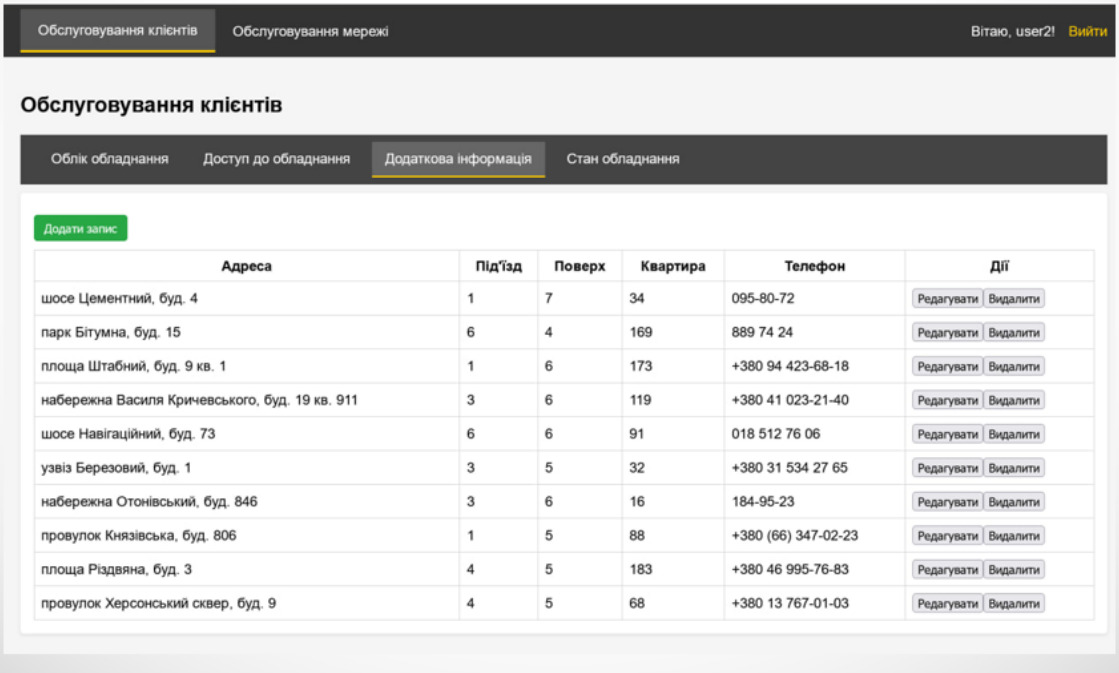
Вкладка де можна знайти контакти відповідальних осіб (наприклад голова правління ОСББ) та графік роботи для комунікації та попередніх домовленостей по запланованих ремонтних роботах та обслуговування лінії зв'язку.



Особа	Графік	Контакт
Віра Дашкевич	shift	001 959 49 17
Камілла Овчаренко	24/7	+380 60 421-51-93
Лариса Швачка	shift	+380 73 111-91-28
Петро Юрчук	shift	907 56 57
Назар Гаврилишин	shift	+380 09 237-36-21
Рубець Владислав Русланович	shift	327 56 44
Зиновій Цюцюра	24/7	040 125 78 36
Роксолана Дейсун	9:00–18:00	+380 25 842-91-36
В'ячеслав Товстуха	9:00–18:00	056 090 54 91
пані Одарка Жученко	shift	+380 94 758-93-44

Дані про абонентів

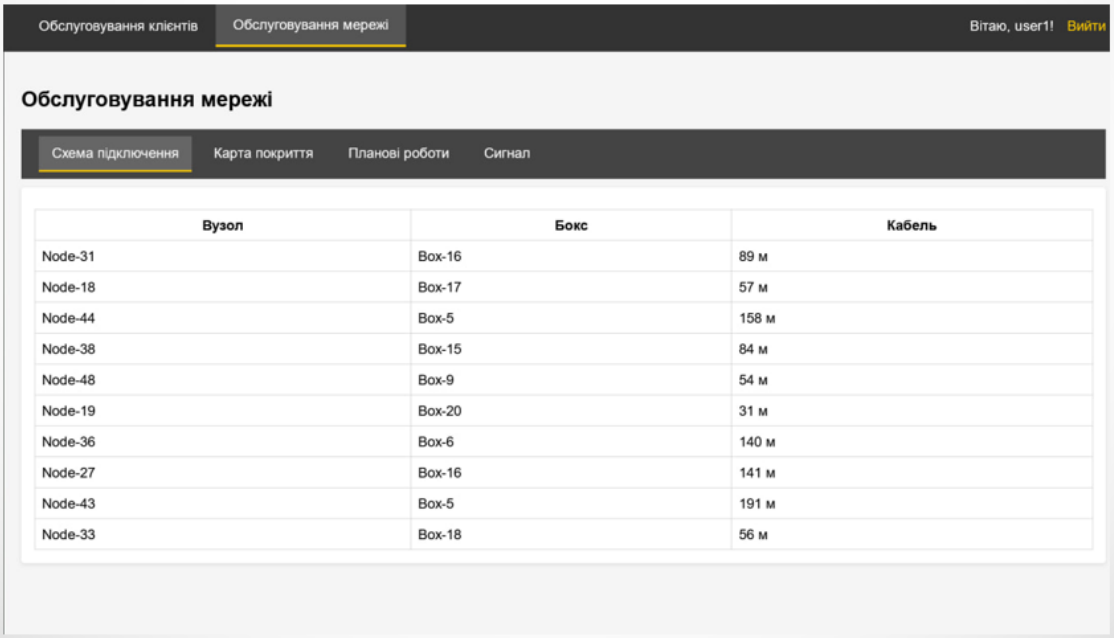
Вкладка на якій можна дізнатись необхідну інформацію про місцезнаходження абонентів та їх контактні дані для попередніх домовленностей.



Адреса	Під'їзд	Поверх	Квартира	Телефон	Дії
шосе Цементний, буд. 4	1	7	34	095-80-72	Редагувати Видалити
парк Бітумна, буд. 15	6	4	169	889 74 24	Редагувати Видалити
площа Штабний, буд. 9 кв. 1	1	6	173	+380 94 423-68-18	Редагувати Видалити
набережна Василя Кричевського, буд. 19 кв. 911	3	6	119	+380 41 023-21-40	Редагувати Видалити
шосе Навігаційний, буд. 73	6	6	91	018 512 76 06	Редагувати Видалити
узвіз Березовий, буд. 1	3	5	32	+380 31 534 27 65	Редагувати Видалити
набережна Отонівський, буд. 846	3	6	16	184-95-23	Редагувати Видалити
провулок Князівська, буд. 806	1	5	88	+380 (66) 347-02-23	Редагувати Видалити
площа Різдвяна, буд. 3	4	5	183	+380 46 995-76-83	Редагувати Видалити
провулок Херсонський сквер, буд. 9	4	5	68	+380 13 767-01-03	Редагувати Видалити

Схема підключення

Вкладка де можна переглянути схему підключення обладнання а саме відслідкувати який оптичний бокс з якого вузла отримує сигнал.



Результати та висновки

- Система реалізує поставлену мету
- Гнучка, масштабована архітектура
- Простий інтерфейс із підтримкою кількох ролей
- Можливість інтеграції з іншими сервісами

Апробація результатів роботи

Брав участь у IX Міжнародній студентській науковій конференції “АКТУАЛЬНІ ПИТАННЯ ТА ПЕРСПЕКТИВИ ПРОВЕДЕННЯ НАУКОВИХ ДОСЛІДЖЕНЬ”

з тезою

“Система обліку мережевого обладнання житлових будинків та контактів відповідальних осіб”

Дякую за увагу!