

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА

“ ___ ” _____ 2025 р.

**Дипломна робота
на здобуття ступеня бакалавра**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Онлайн конфігуратор мережевого обладнання для систем відеоспостереження”

Виконав: студент 4 курсу, групи ТР-11

Голець Кирило Павлович

(прізвище, ім’я, по батькові)

(підпис)

Керівник: доцент к.т.н., Володимир ТИХОХОД

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Рецензент: доцент каф. ТАЕ, к.т.н., доцент, Артур РАЧИНСЬКИЙ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Н.контроль: ст. викладач Ольга БЕСПАЛА

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)
спеціальність 122 “Ком’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

_____ Наталія АУШЕВА

«__» _____ 202__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Голець Кирилу Павловичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Онлайн конфігуратор мережевого обладнання для систем відеоспостереження”

Науковий керівник Тихоход Володимир Олександрович, к.т.н.,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» червня 2025 р. № 1875-с

2. Термін подання роботи студентом 09.06.2025

3. Вихідні дані до роботи мова програмування C#, платформа .NET 8.0, фреймворк Blazor Web App з режимом рендерингу InteractiveServer, JavaScript, система керування базами даних MySQL з ORM Pomelo.EntityFrameworkCore.MySql, CSS-фреймворк Tailwind CSS з використанням NodeJS для компіляції, бібліотека візуалізації схем JointJS (Community Edition), компонентна бібліотека Radzen Blazor Components, система локалізації Microsoft.Extensions.Localization, поштовий сервіс Mailjet.Api для надсилання сповіщень, середовище розробки Visual Studio 2022.

4. Перелік питань, які потрібно розробити:

1) проаналізувати існуючі підходи, методи та інструменти для візуалізації мережевих схем та конфігурування обладнання систем відеоспостереження;

2) визначити архітектурну модель веб-застосунку з урахуванням вимог до продуктивності візуалізації та масштабованості системи;

3) розробити і реалізувати онлайн конфігуратор мережевого обладнання, який забезпечує візуальне проектування схем з'єднання пристроїв, перевірку сумісності компонентів та збереження конфігурацій;

4) підготувати демонстраційні приклади конфігурацій систем відеоспостереження різної складності та документацію для захисту дипломної роботи.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу: діаграма прецедентів, ER-модель бази даних, схеми роботи внутрішніх механізмів платформи Vlazor, діаграми роботи з зовнішніми API, скріншоти інтерфейсів сторінок.

6. Дата видачі завдання 13.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми роботи	13.09.2024	Виконано
2	Аналіз предметної області та існуючих рішень	14-20.04.25	Виконано
3	Опрацювання літератури за темою дипломної роботи, аналіз проблем та пошук рішень	21-27.04.25	Виконано
4	Розробка архітектури серверної API частини. Реалізація інтерфейсу робочої області	28.04-04.05.25	Виконано
5	Реалізація розумних можливостей конфігуратору	05-08.05.25	Виконано
6	Комплексне тестування. Оформлення пояснювальної записки	09-11.05.25	Виконано
7	Захист програмного продукту	12-16.05.25	Виконано
8	Передзахист	26-28.05.25	Виконано
9	Подання готової роботи на кафедрі	09.06.2025	Виконано
10	Захист	17-21.06.2025	Виконано

Студент

_____ (підпис)

Керівник роботи

_____ (підпис)

Кирило ГОЛЕЦЬ

_____ (ім'я, ПРИЗВИЩЕ)

Володимир ТИХОХОД

_____ (ім'я, ПРИЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 65 сторінках, містить 41 рисунок, 6 таблиць, 1 додаток, 18 джерел в переліку посилань. Мета роботи – створення онлайн конфігуратора мережевого обладнання для систем відеоспостереження з інтерактивним інтерфейсом та можливістю збереження конфігурацій. Методи та засоби: мова програмування C#, платформа .NET, технологія Blazor Web App з InteractiveServer rendermode, система управління базами даних MySQL, ORM Entity Framework з драйвером Pomelo, бібліотека JointJS для побудови схем, компоненти Radzen для користувацького інтерфейсу, сервіс Mailjet для електронної пошти, CSS-фреймворк Tailwind. Результат – веб-застосунок онлайн конфігуратора мережевого обладнання для систем відеоспостереження з функціоналом візуального проектування схем з'єднань, каталогом обладнання, збереженням конфігурацій та форумом користувачів.

Ключові слова: МЕРЕЖЕВЕ ОБЛАДНАННЯ, ВІДЕОСПОСТЕРЕЖЕННЯ, ОНЛАЙН КОНФІГУРАТОР, BLAZOR, JOINTJS, ВЕБ-ЗАСТОСУНОК.

ABSTRACT

The thesis work is completed on 65 pages, contains 41 illustrations, 6 tables, 1 appendice, 18 sources in the reference list. The aim of the work is to create an online network equipment configurator for video surveillance systems with an interactive interface and configuration saving capabilities. Methods and tools: C# programming language, .NET platform, Blazor Web App technology with InteractiveServer rendermode, MySQL database management system, Entity Framework ORM with Pomelo driver, JointJS library for schema construction, Radzen components for user interface, Mailjet service for email functionality, Tailwind CSS framework. Result – a web application of online network equipment configurator for video surveillance systems with visual connection schema design functionality, equipment catalog, configuration saving and user forum.

Keywords: NETWORK EQUIPMENT, VIDEO SURVEILLANCE, ONLINE CONFIGURATOR, BLAZOR, JOINTJS, WEB APPLICATION.

ЗМІСТ

ВСТУП.....	8
1 РОЗРОБКА ОНЛАЙН КОНФІГУРАТОРУ МЕРЕЖЕВОГО ОБЛАДНАННЯ ...	10
1.1 Постановка технічного завдання онлайн конфігуратору	10
1.2 Огляд застосованих методів	12
1.3 Потенційна аудиторія	14
2 ОПИС ВИКОРИСТАНИХ МЕТОДІВ	16
2.1 Платформа ASP.NET Blazor	16
2.2.1 Режими відображення користувацького інтерфейсу.....	16
2.2.2 Переваги використання інтерактивного режиму	18
2.2.3 Робота з JavaScript на платформі Blazor	19
2.2. База даних MySQL.....	21
2.3 Бібліотеки для реалізації серверної частини.....	22
2.3.1. Бібліотека Entity Framework.....	22
2.3.2 Бібліотека Mailjet.Api.....	24
2.3.3 Бібліотека ASP.NET Identity.....	25
2.4 Бібліотеки для реалізації клієнтської частини.....	27
2.4.1 Бібліотека стилів Tailwind	27
2.4.2 Бібліотека готових компонентів Radzen	28
2.4.3 Бібліотека Microsoft.Extensions.Localization.....	29
2.4.4 Бібліотека JointJS.....	30
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	32
3.1 Архітектура додатку	32
3.2 Опис веб-сторінок сервісу	35

3.2.1 Загальний опис функціоналу.....	35
3.2.2 Користувацький форум.....	38
3.2.3 Перегляд та редагування пристроїв.....	40
3.2.4 Онлайн конфігуратор.....	41
3.2.5 Сторінка перегляду готових конфігурацій.....	43
3.3 Персоналізація та адаптація інтерфейсу користувача.....	44
3.3.1 Багатомовна локалізація інтерфейсу.....	44
3.3.2 Система кольорових тем та адаптив.....	45
3.3.3 Гнучке відображення цінової інформації.....	46
3.4 Опис структури бази даних.....	46
4 РОБОТА З ГОТОВОЮ СИСТЕМОЮ.....	51
4.1 Сценарії роботи звичайного користувача.....	51
4.2 Сценарії роботи модератору сервісу.....	62
4.3 Сценарії роботи адміністратору сервісу.....	63
4.4 Розгортання проекту в середовищі Docker.....	65
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТОК А.....	71

ВСТУП

Сучасний ринок систем відеоспостереження характеризується високою динамікою розвитку та постійним ускладненням технічних рішень. Проектування таких систем вимагає врахування численних факторів: сумісності обладнання різних виробників, правильного підбору компонентів за технічними характеристиками, оптимального розміщення пристроїв та забезпечення належної пропускної здатності мережі. Традиційні методи проектування, що базуються на ручних розрахунках та використанні спеціалізованого програмного забезпечення, часто виявляються недостатньо ефективними через високу вартість ліцензій, складність освоєння та відсутність актуальної бази обладнання.

Актуальність розробки онлайн конфігуратора мережевого обладнання для систем відеоспостереження обумовлена кількома ключовими факторами. Спостерігається стрімке зростання попиту на системи відеоспостереження як у комерційному, так і в приватному секторах.

Існуючі рішення для проектування систем відеоспостереження мають суттєві недоліки. Професійне програмне забезпечення потребує значних фінансових витрат на ліцензування та навчання персоналу. Водночас, безкоштовні альтернативи часто обмежені у функціональності та не забезпечують візуалізацію зв'язків між компонентами системи.

Сучасні тенденції переходу бізнес-процесів в онлайн-середовище створюють передумови для розробки веб-орієнтованих рішень. Онлайн конфігуратор забезпечує доступність інструменту з будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення, що значно спрощує процес проектування для кінцевих користувачів.

Дослідження в галузі комп'ютерного проектування мережевих систем підтверджують ефективність використання візуальних інтерфейсів для представлення топології мережі та зв'язків між компонентами.

Метою роботи є розробка веб-застосунку для конфігурування мережевого обладнання систем відеоспостереження з використанням сучасних технологій веб-розробки та візуального представлення схем з'єднання пристроїв.

Для досягнення поставленої мети необхідно виконати наступні 4 завдання:

1. Аналіз підходів, методів та алгоритмів розв'язання задачі візуалізації мережевих схем та забезпечення сумісності обладнання. Дослідження існуючих рішень у сфері проектування систем відеоспостереження, аналіз алгоритмів перевірки сумісності пристроїв.

2. Аналіз програмних засобів для реалізації системи онлайн конфігуратора мережевого обладнання. Порівняльний аналіз режимів роботи фреймворку Blazor, вибір оптимального стеку технологій з урахуванням вимог до продуктивності та масштабованості.

3. Проектування архітектури застосунку з використанням Blazor Web App та .NET, реалізація серверної частини з підтримкою Entity Framework та MySQL, розробка клієнтської частини з використанням JointJS для візуалізації схем, впровадження механізмів перевірки сумісності обладнання.

4. Тестування розробленого програмного забезпечення. Розробка набору тестових сценаріїв для перевірки функціональності конфігуратора, навантажувальне тестування для визначення меж масштабованості системи.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Перший розділ присвячено аналізу предметної області та існуючих рішень. У другому розділі розглядаються технології та інструменти розробки. Третій розділ містить опис проектування та реалізації програмного забезпечення. Четвертий розділ присвячено тестуванню та оцінці ефективності розробленого рішення.

Робота містить 41 рисунок, 6 таблиць, список використаних джерел налічує 18 найменувань. Загальний обсяг роботи становить 65 сторінок основного тексту та 8 сторінок додатків.

1 РОЗРОБКА ОНЛАЙН КОНФІГУРАТОРУ МЕРЕЖЕВОГО ОБЛАДНАННЯ

Сьогодні на ринку представлено багато різних виробників мережевого обладнання, що ускладнює вибір найкращих варіантів. Помилки на етапі планування стають можливими через відсутність єдиного інструменту для візуального проектування та автоматичної перевірки конфігурації, що може призвести до нераціонального використання ресурсів.

У цій частині розглядаються мета і завдання розробки онлайн-конфігуратора мережевого обладнання, а також можлива база користувачів, яким цей інструмент буде корисний. Автоматизуючи планування і проектування систем відеоспостереження, програма забезпечує точність розрахунків і знижує ймовірність технічних помилок на етапі проектування.

1.1 Постановка технічного завдання онлайн конфігуратору

З огляду на зростаючий попит на надійні системи безпеки, які повинні бути правильно спроектовані і налаштовані, створення онлайн-конфігуратора мережевого обладнання є нагальним завданням. Основною метою програмного забезпечення є розробка зручного, інтерактивного інструменту для створення та перевірки мережевих рішень в індустрії відеоспостереження.

Використовуючи графічний інтерфейс користувача, де компоненти представлені у вигляді блоків з відповідними входами і виходами, необхідно розробити веб-додаток, який дозволить користувачам візуально конструювати і тестувати конфігурації мережевого обладнання. Система повинна підтримувати всі сучасні функції, протоколи та інтерфейси мережевого обладнання.

Архітектура програмного забезпечення базується на використанні сучасних веб-технологій, зокрема платформи .NET для серверної частини та фреймворку Blazor для реалізації інтерактивного графічного інтерфейсу конфігуратора. Вибір

гібридного підходу обумовлений необхідністю забезпечення високої продуктивності клієнтської частини при збереженні надійності серверних операцій з базою даних.

Каталог доступного обладнання з вичерпними технічними характеристиками, система форумів для обміну досвідом користувачів, можливість створювати та змінювати конфігурації обладнання, а також зберігати готові рішення - все це можливості системи. Інтерактивна дошка слугує інтерфейсом конфігуратора, що дозволяє користувачам в інтуїтивно зрозумілій формі підключати та розташовувати компоненти відповідно до технічних специфікацій.

Залежно від типу користувача, система ролей надає різний ступінь доступу до можливостей програми. Онлайн-конфігуратор може використовуватися локально анонімними користувачами, які також можуть переглядати каталог обладнання та попередні налаштування. Додаткові функції для зареєстрованих користувачів включають можливість надсилати відгуки, брати участь у форумах та зберігати власні налаштування. Хоча адміністратори мають повну владу над системою, включаючи можливість додавати нове обладнання та модерувати матеріали користувачів, модератори стежать за тим, щоб контент на форумах відповідав високим стандартам.

Технічна реалізація передбачає використання MySQL як основної системи управління базами даних.

Дизайн інтерфейсу базується на Tailwind CSS у поєднанні з готовими компонентами Radzen, що забезпечує сучасний, адаптивний та користувацький інтерфейс. Компоненти Radzen також забезпечують функціональність віртуалізації та пагінації для ефективної роботи з великими обсягами даних у таблицях та списках обладнання.

1.2 Огляд застосованих методів

Платформа .NET представляє собою сучасну, багатофункціональну екосистему для розробки веб-додатків, що забезпечує високу продуктивність, масштабованість та кросплатформенну сумісність. Архітектура .NET дозволяє створювати надійні серверні рішення з підтримкою різноманітних баз даних, систем кешування та зовнішніх сервісів [1]. Платформа забезпечує ефективне управління пам'яттю через автоматичне збирання сміття, а також надає потужні засоби для асинхронного програмування, що критично важливо для веб-додатків з високим навантаженням.

Фреймворк Blazor, як складова частина екосистеми .NET, революціонізує підхід до розробки веб-інтерфейсів, дозволяючи використовувати C# замість JavaScript для створення інтерактивних клієнтських додатків. Blazor Server надає унікальну можливість виконання логіки додатку на сервері з передачею лише необхідних змін до браузера через SignalR з'єднання [2]. Такий підхід забезпечує низьку затримку взаємодії, зменшення навантаження на клієнтський пристрій та централізоване управління бізнес-логікою.

Технологія SignalR забезпечує зв'язок між клієнтом і сервером в режимі реального часу, автоматично обираючи найкращий протокол передачі даних, виходячи з умов мережі та можливостей браузера. Основним протоколом для забезпечення двостороннього зв'язку з низькою затримкою є WebSocket-з'єднання, які автоматично перемикаються на інші технології в разі їх недоступності. Перевагами цього методу є можливість побудови складних інтерактивних ситуацій без написання JavaScript-коду на стороні клієнта, синхронізація стану між сеансами користувача та швидка модифікація інтерфейсу без необхідності перезавантаження сторінки.

Головною особливістю онлайн-конфігуратора є інтерактивна робоча область, яка нагадує дошку і надає клієнтам простий спосіб візуального налаштування мережевих налаштувань. Можна працювати зі схемами необмеженої складності, оскільки робоча область розроблена у вигляді масштабованого полотна, яке

підтримує функції масштабування, панорамування та перетягування. Для безперешкодної взаємодії з елементами схеми в інтерфейсі конфігуратора використовується бібліотека JointJS, яка орієнтована на роботу з векторною графікою і забезпечує високу продуктивність навіть при величезній кількості елементів на сцені.

Кожен графічний блок, що представляє пристрій у конфігураторі, містить візуальне зображення гаджета, назву моделі, категорію та актуальну ринкову ціну. Швидке розпізнавання характеристик обладнання забезпечується стандартизованим дизайном блоків пристроїв, який має чітко позначені секції для різних типів інформації. Щоб гарантувати швидке відображення без втрати якості візуалізації, зображення пристроїв завантажуються з бази даних в оптимізованому вигляді.

Система портів та з'єднань є ключовою особливістю конфігуратора, що забезпечує точне моделювання реальних технічних характеристик обладнання. Кожен пристрій має набір вхідних та вихідних портів, що відповідають фізичним інтерфейсам реального обладнання. Порти візуально розрізняються за допомогою кольорового кодування, де кожен тип з'єднання має унікальний колір.

Для підключення пристроїв використовуються тільки сумісні порти одного типу, що гарантує технічну достовірність побудованих конфігурацій. При спробі встановити з'єднання система автоматично перевіряє сумісність портів і відхиляє з'єднання, які не сумісні із зазначеним візуальним відгуком. Для збереження візуальної цілісності діаграми лінії з'єднань показані у вигляді векторних кривих, які негайно перефарбовуються при зміні пристроїв. Колір кривих відповідає типу з'єднання.

Редагування з'єднань забезпечується через контекстні меню та інтерактивні елементи керування, що дозволяють змінювати параметри з'єднання, додавати проміжні точки для складних трас кабелів та видаляти непотрібні з'єднання.

1.3 Потенційна аудиторія

Першу групу можливих користувачів складають системні інтегратори, інженери з безпеки, мережеві адміністратори та проектувальники систем відеоспостереження. Для представників цієї групи критично важливими є здатність системи будувати складні топології з великою кількістю компонентів і коректність технічних даних щодо сумісності обладнання. При роботі зі складними схемами від професіоналів вимагається відмінна продуктивність і всебічне знання технічних особливостей пристроїв, оскільки ці фактори впливають на ефективність їхньої роботи і якість рішень, які вони створюють.

Другу категорію складають початківці та спеціалісти суміжних галузей, які потребують інструменту для проектування базових систем відеоспостереження без глибокого розуміння всіх технічних нюансів. Для користувачів цієї групи ключовими факторами виступають інтуїтивно зрозумілий інтерфейс без надмірної технічної складності та наявність готових шаблонів і прикладів конфігурацій. Важливими також є підказки та валідація з'єднань у режимі реального часу, що дозволяють уникнути помилок при проектуванні. Доступна документація з поясненням основних принципів роботи з системою та форум для обміну досвідом суттєво спрощують процес навчання та отримання необхідної допомоги від більш досвідчених користувачів.

Третя група складається з менеджерів із закупівель, IT-директорів та керівників служб безпеки - представників бізнесу, які приймають рішення про придбання та розгортання обладнання для відеоспостереження. Для цієї групи критично важливими компонентами системи є можливість оцінити можливості різних комплектів обладнання та розрахувати вартість різних варіантів конфігурації. Можливість експорту специфікацій для додаткового вивчення та чітке відображення загальної вартості проекту з розбивкою за категоріями обладнання є важливими для представників керівництва. Альтернативні варіанти проекту з різним ступенем функціональності та фінансовими обмеженнями повинні мати можливість зберігатися і порівнюватися за допомогою системи.

Можливість експорту даних у форматах, які працюють з корпоративними системами планування ресурсів і бюджетування, є ще одним варіантом, про який варто подумати.

Всі ключові потреби та особливі вимоги для трьох категорій користувачів наведені у таблиці 1.1.

Таблиця 1.1 - Порівняння потреб різних категорій користувачів додатку

Тип користувача	Ключові потреби	Особливі вимоги
Інженери, мережеві адміністратори, проектувальники систем	Точність даних; можливість створення складних схем; висока швидкодія	Широка база обладнання; підтримка складних топологій; оптимізована робота з великими конфігураціями
Новачки, монтажники, фахівці	Простота та інтуїтивність; мінімум технічної складності; навчальні матеріали	Готові шаблони конфігурацій; підказки та перевірка з'єднань у реальному часі; форум;
Керівники, директори, менеджери	Оцінка вартості та ефективності; порівняння варіантів; аргументація перед керівництвом	Калькулятор вартості; порівняння характеристик комплектів

Урахування потреб усіх категорій потенційних користувачів вимагає створення багаторівневого інтерфейсу, де базовий функціонал буде доступний та зрозумілий початківцям, а розширені можливості задовольнятимуть потреби професіоналів. Реалізація різних режимів роботи з конфігуратором дозволить кожному користувачу обрати найбільш зручний варіант відповідно до власних потреб та рівня підготовки.

2 ОПИС ВИКОРИСТАНИХ МЕТОДІВ

Розділ присвячено детальному аналізу технологічного стеку та методологічних підходів, використаних при розробці системи конфігурування мережевого обладнання. Основну увагу приділено обґрунтуванню вибору платформи ASP.NET Blazor як основи для реалізації веб-додатка, системи управління базами даних MySQL для зберігання та обробки даних, а також ключових бібліотек серверної та клієнтської частин. Кожен з описаних компонентів розглядається з точки зору його переваг, особливостей реалізації та внеску в загальну архітектуру системи.

2.1 Платформа ASP.NET Blazor

У підрозділі проаналізовано можливості платформи ASP.NET Blazor, з акцентом на особливості рендерингу інтерфейсу, переваги інтерактивного режиму та способи взаємодії з JavaScript в рамках створеної системи та архітектури.

2.2.1 Режими відображення користувацького інтерфейсу

Одним з ключових аспектів платформи Blazor є наявність різних режимів відображення користувацького інтерфейсу, які визначають, де саме відбувається рендеринг компонентів - на сервері чи на клієнті. В реалізованій системі використовується переважно режим InteractiveServer, проте архітектура забезпечує можливість переходу окремої сторінки на інші режими (SSR або Auto) за необхідності.

Інтерактивний серверний режим (Interactive Server Rendering) передбачає виконання логіки компонентів на сервері, в той час як DOM-оновлення відбуваються на клієнті.

При використанні цього режиму встановлюється SignalR-з'єднання між браузером та сервером, через яке передаються події користувача та оновлення інтерфейсу. Режим InteractiveServer застосовується для більшості сторінок, що дозволило забезпечити швидкий початковий рендеринг сторінок та ефективну обробку подій користувача (рисунок 2.1).

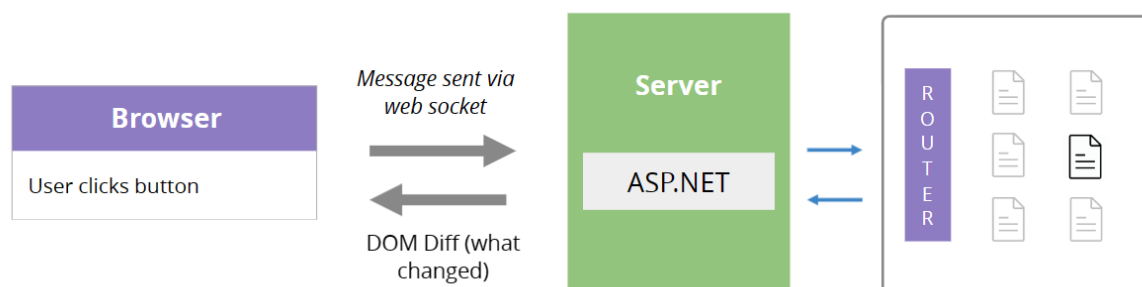


Рисунок 2.1 – Принцип роботи режиму InteractiveServer [3]

Server-Side Rendering (SSR) представляє собою режим, при якому початковий рендеринг сторінки відбувається на сервері, а клієнту передається готовий HTML. Режим дозволяє забезпечити швидке завантаження початкового вмісту сторінки та покращити SEO, оскільки пошукові роботи отримують повністю сформований HTML-контент.

Слід зазначити, що режим SSR використовується для всіх типів авторизації, які використовують ASP.NET Identity. Це результат особливостей механізму авторизації, який включає зберігання файлів cookie авторизації на стороні клієнта. Було вирішено використовувати SSR для форм авторизації, реєстрації, відновлення паролів та інших компонентів системи Identity, оскільки неможливо змінити ці файли cookie, поки з'єднання SignalR знаходиться в режимі InteractiveServer.

2.1.2 Переваги використання інтерактивного режиму

Використання режиму InteractiveServer у розробленій системі має низку суттєвих переваг, які вплинули на рішення використовувати підхід для більшості сторінок додатку.

Однією з головних переваг режиму InteractiveServer є спрощена розробка на стороні клієнта. Інфраструктура Blazor надає всі необхідні оновлення автоматично, тому розробникам не потрібно створювати JavaScript-код для обробки подій та оновлення DOM. Для прикладу розглянемо базовий компонент для вибору рейтингу конфігурації. Компонент використовує C# конструкцію `for()` для генерації повторювальних кнопок зірочок, що будуть визначати рейтинг від 1 до 5. Такий підхід дозволяє зосередитися на бізнес-логіці додатку та підвищити продуктивність розробки.

Ще однією перевагою є необмежений доступ до всіх можливостей .NET Framework. З InteractiveServer серверний код отримує доступ до бібліотек all.NET, на відміну від WebAssembly, який пропонує лише невеликий вибір бібліотек. Для роботи з базою даних, обробки файлів та інших завдань, які зазвичай виконуються на сервері, це дуже важливо.

Час початкового завантаження сторінки і вимоги до ресурсів клієнта зменшуються при використанні InteractiveServer, оскільки клієнту не потрібно завантажувати і запускати середовище виконання .NET в браузері. Оскільки сервер зберігає стани компонентів, зменшується ймовірність втрати даних у разі перезавантаження сторінки або інших непередбачуваних обставин на клієнті.

Безпека сервісу досягається за рахунок того, що бізнес-логіка та код обробки даних залишаються на сервері і не передаються клієнту. Це зменшує ризик несанкціонованого доступу до логіки додатку. Оскільки основна обробка відбувається на сервері, клієнтський пристрій виконує мінімальну кількість обчислень, що покращує досвід користувача на мобільних пристроях та пристроях з обмеженими ресурсами.

Для сторінок з представленням даних (сторінки пристроїв, готових налаштувань) і форумів, де важлива швидка взаємодія з базою даних і динамічне оновлення контенту, інтерактивний режим особливо добре працює в рамках спроектованої системи.

При розробці системи важливо пам'ятати, що режим InteractiveServer також має певні обмеження. Найважливішим з них є вимога до стійкого мережевого з'єднання, оскільки його відсутність унеможлиблює використання програми до відновлення з'єднання. Крім того, кожна активна сесія користувача використовує ресурси сервера, що може обмежити масштабованість системи при одночасному використанні її багатьма користувачами.

2.1.3 Робота з JavaScript на платформі Blazor

У деяких випадках інтеграція JavaScript є не тільки доречною, але й необхідною для досягнення оптимальної функціональності, навіть незважаючи на те, що платформа Blazor в основному дозволяє розробляти веб-додатки мовою програмування C#. У випадку необхідності застосування JavaScript коду Blazor пропонує способи взаємодії з екосистемою JavaScript, яка була використана в архітектурі системи, зокрема, при реалізації складних функцій онлайн-конфігуратора мережевого обладнання.

Механізм JavaScript Interop представляє собою фундаментальний механізм двостороннього зв'язку між керованим C#-кодом та нативним JavaScript-середовищем у Blazor-додатках [4]. Такий підхід забезпечує можливість виклику JavaScript-функцій з C#-контексту та зворотного виклику C#-методів з JavaScript-коду, створюючи гнучку архітектуру взаємодії. Сервіс IJSRuntime, що інжектуються в компоненти Blazor через механізм Dependency Injection, надає асинхронний інтерфейс для виконання JavaScript-операцій без блокування основного потоку виконання.

Для реалізації процесу передачі даних між клієнтським JavaScript середовищем та серверним C# кодом використовується система автоматичної серіалізації та десеріалізації об'єктів. під час передачі клієнту .NET об'єктів Blazor використовує вбудований JSON серіалізатор для перетворення їх у формат JSON; під час отримання даних від клієнта - той самий серіалізатор для перетворення JavaScript об'єктів у типізовані C# сутності. Для створення відповідних графічних елементів та налаштування їх поведінки на інтерактивному полотні в розробленій системі активно використовується механізм передачі детальної інформації про характеристики мережевих пристроїв, такої як технічні характеристики, параметри портів та правила підключення.

Представлена схема на рисунку 2.2 ілюструє архітектурну модель виконання Blazor та механізми взаємодії з нативними веб-API браузера. У лівій частині діаграми показано компоненти Blazor-додатку та шар взаємодії (Interop layer), який забезпечує зв'язок між керованим .NET-кодом та JavaScript-середовищем. Центральна частина схеми демонструє процес обробки коду через компоненти JavaScript Runtime. WebAssembly-модуль виконує роль середовища виконання .NET, що взаємодіє з нативними бібліотеками, забезпечуючи повнофункціональне виконання .NET-коду в браузері. Права частина діаграми показує доступ до нативних Web API браузера.

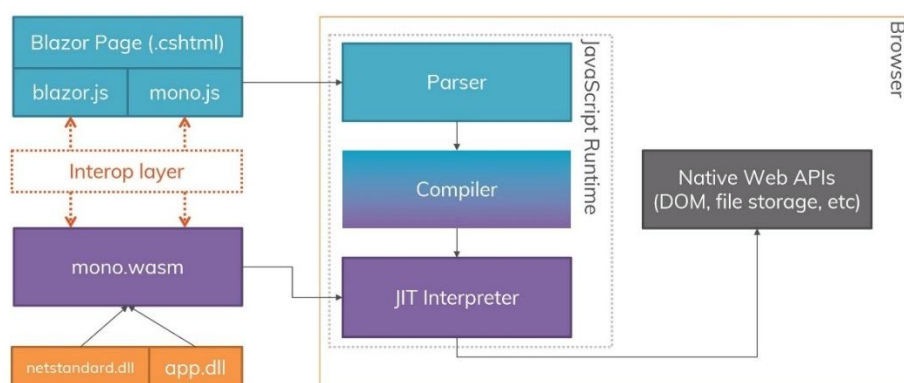


Рисунок 2.2 – Схема роботи Blazor додатку з JSIntertop [5]

При створенні сучасного та масштабованого програмного продукту гібридна архітектура особливо добре підходить для складних веб-додатків, які потребують поєднання надійності та структури серверного коду на C# з ефективністю клієнтських бібліотек JavaScript..

2.2. База даних MySQL

Система управління реляційними базами даних MySQL забезпечує високу продуктивність, надійність та масштабованість, що досягається завдяки оптимізованому ядру та ефективним алгоритмам обробки запитів. Відкритий код та низька вартість володіння MySQL доступна під ліцензією GPL, що дозволяє використовувати її без ліцензійних виплат. Це зменшує загальну вартість розробки та експлуатації системи.

Оскільки сервіс повинен швидко обробляти запити на інформацію про пристрої, налаштування з'єднання та збереження користувацької конфігурації, швидкість роботи MySQL є особливо важливою. Коли каталогом обладнання або форумами користуються кілька користувачів одночасно, ACID-сумісні транзакції системи зберігання даних InnoDB мають важливе значення для збереження цілісності даних. MySQL надає можливість записувати дані конфігурації з високим паралелізмом, не перешкоджаючи зчитуванню інформації про пристрої завдяки механізму блокування на рівні рядків.

Гнучкість типів даних MySQL підтримує зберігання як структурованих даних про технічні характеристики обладнання, так і JSON-документів для збереження складних конфігурацій мережевих схем. Тип JSON надає можливість ефективного індексування та пошуку всередині документів конфігурацій, що важливо для функціоналу пошуку готових рішень. Повнотекстові індекси дозволяють реалізувати швидкий пошук по описам обладнання та повідомленням форумів.

База даних MySQL пропонує гнучкість у розгортанні системи завдяки сумісності з широким спектром хмарних платформ і операційних систем.

Процедури розробки, тестування та розгортання у виробництво спрощуються завдяки підтримці контейнеризації за допомогою Docker. Є подальші можливості для оптимізації роботи з інвентарем мережевого обладнання та відстеженням продуктивності запитів завдяки інтеграції з системами моніторингу.

2.3 Бібліотеки для реалізації серверної частини

У підрозділі розглядаються бібліотеки, що використовуються для реалізації серверної логіки системи. Основну увагу приділено Entity Framework Core як засобу взаємодії з базою даних, а також бібліотеці Mailjet.Api.

2.3.1. Бібліотека Entity Framework

Бібліотека Entity Framework Core є потужним об'єктно-реляційним мапером (ORM) [6], який забезпечує доступ до бази даних через об'єктно-орієнтовану абстракцію. У розробленій системі Entity Framework використовується для взаємодії з базою даних MySQL, забезпечуючи зручний та типобезпечний доступ до даних.

При проектуванні бази даних використовувався підхід Code First для визначення моделі бази даних. Методологія Code-First передбачає створення структури бази даних на основі класів C#, що дозволяє розробникам працювати в парадигмі об'єктно-орієнтованого програмування без необхідності безпосереднього написання SQL-скриптів для створення таблиць. Модель домену описується через звичайні класи C#, які Entity Framework автоматично перетворює на відповідні таблиці бази даних під час виконання міграцій.

Серед ключових переваг методу Code-First - версіонування структури бази даних за допомогою системи міграції, яка дозволяє відстежувати зміни схеми та автоматично застосовувати їх у різних контекстах розробки. Тісний зв'язок з

системами контролю версій гарантує синхронізацію змін у схемі бази даних між інженерами команди. Можливість використовувати весь набір функцій C# для нативного представлення бізнес-логіки в моделях предметної області полегшує розробку та підтримку коду.

Внутрішні механізми Entity Framework використовують систему навігаційних властивостей для встановлення зв'язків між сутностями, що автоматично перетворюються на зовнішні ключі в базі даних. Зв'язок «один до багатьох» між пристроєм та портами налаштовується через колекцію портів у класі Device та навігаційну властивість Device у класі Port (рисунок 2.3).

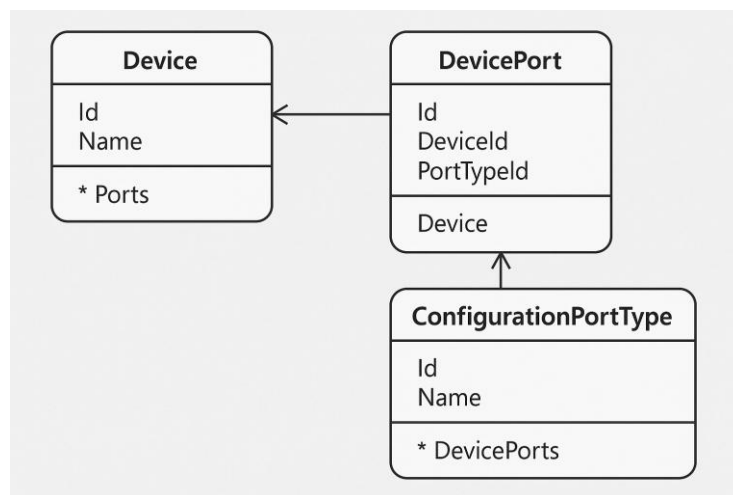


Рисунок 2.3 – Приклад зв'язку «один до багатьох»

Складні зв'язки налаштовуються через Fluent API в методі OnModelCreating класу ApplicationDbContext. Це дозволяє точно контролювати створення індексів, обмежень цілісності та каскадних операцій видалення.

Система міграцій Entity Framework автоматично генерує SQL-скрипти для оновлення структури бази даних при змінах у моделях. Кожна міграція представлена класом C# з методами Up() та Down(), що містять прямі та зворотні операції зі зміни схеми. Автоматично генеровані міграції можуть бути доповнені власним кодом для складних операцій перетворення даних.

Бібліотека Pomelo.EntityFrameworkCore.MySql забезпечує оптимізовані перекладачі LINQ-запитів у специфічні для MySQL конструкції SQL. Підтримка JSON-операцій дозволяє ефективно працювати з конфігураціями пристроїв, зберігаючи складні об'єкти у JSON-полях з можливістю індексування та пошуку.

2.3.2 Бібліотека Mailjet.Api

Бібліотека Mailjet.Api представляє собою .NET клієнт для API сервісу Mailjet, що використовується для відправлення електронних листів. Бібліотека забезпечує функціонал комунікації з користувачами, включаючи підтвердження реєстрації, відновлення паролю та зміну пошти.

Сервіс Mailjet є хмарним сервісом електронної пошти, що пропонує рішення для транзакційних та маркетингових розсилок. Платформа функціонує як Email Service Provider (ESP), надаючи інфраструктуру для надійної доставки електронних повідомлень через RESTful API [7]. Сервіс обробляє мільйони листів щодня, забезпечуючи високу швидкість доставки та моніторинг репутації відправника.

Переваги використання готової пропозиції Mailjet над власною системою електронної пошти є численними і важливими для професійного застосування. IP-адреси та домени Mailjet вже мають хорошу репутацію у провайдерів електронної пошти, тому електронні листи з меншою ймовірністю потрапляють до папки «Спам». Принцип взаємодії з API Mailjet зображено на рисунку 2.4.

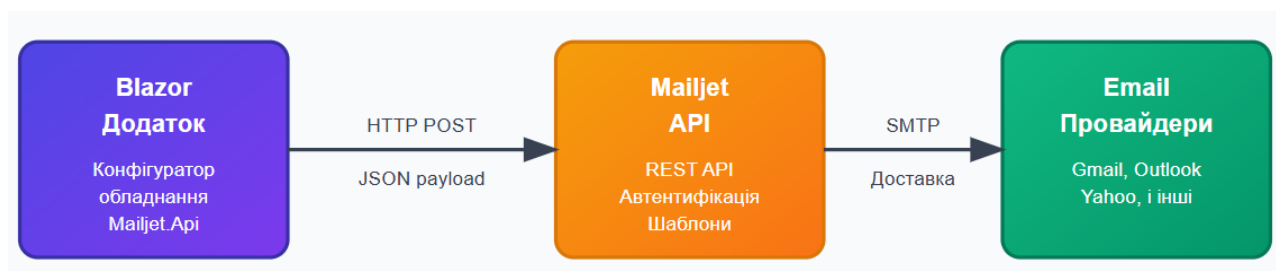


Рисунок 2.4 – Принцип роботи з MailJet API

Масштабованість сервісу дозволяє обробляти як поодинокі транзакційні листи для окремих користувачів конфігуратора, так і масові розсилки для великої кількості підписників форумів. Автоматичне керування навантаженням та розподіл трафіку між серверами забезпечують стабільну роботу навіть при пікових навантаженнях. Система моніторингу в реальному часі відстежує стан доставки та негайно повідомляє про проблеми.

Перевага Mailjet.API полягає в тому, що його легко інтегрувати. Бібліотека надає простий інтерфейс для зв'язку з Mailjet API, що дозволяє швидко інтегрувати можливість надсилання електронних листів у вашу систему. Асинхронні методи бібліотеки гарантують, що веб-додаток не блокується під час надсилання повідомлень. Підтримка шаблонів дозволяє створювати професійні листи з динамічним контентом.

2.3.3 Бібліотека ASP.NET Identity

Бібліотека ASP.NET Identity представляє собою комплексну систему управління користувачами та авторизації, яка забезпечує повний цикл аутентифікації та авторизації в ASP.NET додатках. У контексті розробленої системи конфігурування мережевого обладнання ASP.NET Identity виступає фундаментальним компонентом безпеки, що забезпечує захищений доступ до функціоналу платформи та персоналізацію користувацького досвіду [8].

Архітектура ASP.NET Identity побудована на модульному наборі сервісів, які можуть бути адаптовані до різних сценаріїв зберігання даних і методів автентифікації. Побудована система використовує в якості провайдеру контекст Entity Framework, який взаємодіє з основною базою даних MySQL через спеціальні таблиці, що містять інформацію про користувачів, ролі, привілеї доступу та сеанси. Система забезпечує розширювану модель користувача шляхом успадкування від базового класу IdentityUser, що дозволяє вставляти специфічні для домену

атрибути, такі як профіль користувача форуму та історія активності в налаштуваннях системи.

Сервіс UserManager виконує перевірку даних, хешування паролів і створення облікових записів з відповідними налаштуваннями безпеки. Для забезпечення високого рівня безпеки в системі передбачені настроювані обмеження паролів, які вимагають мінімальної довжини, а також спеціальних символів і цифр. Двофакторна автентифікація реалізована через взаємодію із зовнішніми провайдерами та SMS/email-підтвердження.

Управління сеансами та автентифікацією здійснюється за допомогою системи безпеки на основі тверджень, яка призначає кожному користувачеві набір тверджень, що характеризують його права, ролі та якості. SignInManager пропонує безпечний метод входу в систему, який включає в себе автентифікацію на основі файлів cookie та можливість визначати тривалість сесії.

Система ролей та дозволів побудована на принципах role-based access control (RBAC) [9], що дозволяє гнучко налаштовувати права доступу для різних категорій користувачів системи. У розробленій платформі визначено кілька основних ролей: звичайні користувачі з правами створення конфігурацій та участі в форумах, модератори з можливістю управління контентом форумів, адміністратори з повними правами управління системою та користувачами. RoleManager сервіс забезпечує програмне управління ролями, включаючи створення, модифікацію та призначення ролей користувачам.

Інтеграція з Blazor Server здійснюється за допомогою спеціальних компонентів і сервісів, які дозволяють здійснювати автентифікацію в режимі рендерингу на стороні сервера. Компонент AuthorizeView дозволяє умовно показувати вміст на основі статусу автентифікації та ролей користувачів, що особливо корисно для розробки адаптивного інтерфейсу з різними рівнями доступу. CascadingAuthenticationState автоматично поширює інформацію про автентифікацію по дереву компонентів, усуваючи необхідність безпосередньої передачі параметрів.

2.4 Бібліотеки для реалізації клієнтської частини

У підрозділі описано інструменти, що використовуються для створення клієнтського інтерфейсу системи. Розглядаються бібліотеки стилів Tailwind CSS, колекція готових компонентів Radzen, інструменти локалізації та JavaScript.

2.4.1 Бібліотека стилів Tailwind

Бібліотека Tailwind CSS є утилітарним фреймворком для CSS, який значно спрощує та прискорює розробку стилізованих інтерфейсів. У реалізованій системі Tailwind використовується для генерації оптимізованих CSS-файлів.

Утилітарний підхід Tailwind дозволяє створювати дизайн безпосередньо в HTML-коді за допомогою готових класів. Замість написання CSS-правил, розробник використовує предвизначені класи, які застосовують конкретні стилі до елементів. Кожен клас у Tailwind відповідає за одну конкретну CSS-властивість [10], що забезпечує модульність та гнучкість у побудові інтерфейсів.

Процес роботи Tailwind базується на концепції "utility-first", де кожен клас виконує одну атомарну функцію. Наприклад, клас `p-4` встановлює внутрішні відступи (`padding`) розміром `1rem` з усіх сторін елемента, `text-center` вирівнює текст по центру, а `bg-blue-500` встановлює синій колір фону з певною насиченістю.

Двигун Tailwind використовує систему дизайн-токенів для забезпечення консистентності дизайну. Токени визначають:

- кольорову палітру: від 50 (найсвітліший) до 950 (найтемніший) для кожного базового кольору (рисунок 2.5);
- розміри відступів - масштабована система від 0 до 96 з кроком `0.25rem`;
- типографіку - розміри шрифтів від `xsmall` до `9xl`;
- тіні, радіуси, анімації - предвизначені значення для візуальних ефектів.

Tailwind працює через систему PostCSS, яка обробляє CSS-файли на етапі збірки проекту. Результатом є оптимізований CSS-файл, який містить лише ті стилі,

що реально використовуються в додатку, що значно зменшує обсяг завантажуваних ресурсів.

Значною перевагою Tailwind перед аналогами, такими як Bootstrap є можливість передавати параметри у CSS класи, що дозволяє налаштувати власні кольори, розміри, шрифти та інші аспекти дизайн-системи відповідно до потреб проекту.

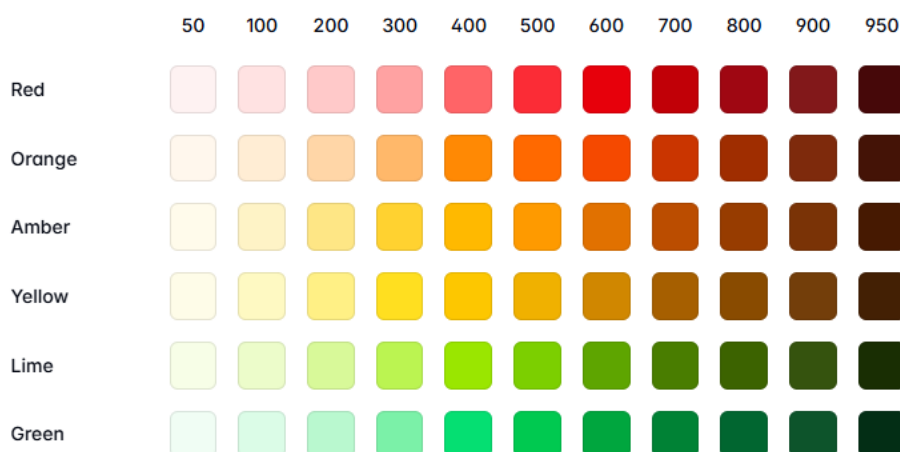


Рисунок 2.5 – Приклад кольорової палітри Tailwind

Інтеграція Tailwind з системою збірки проекту через PostCSS забезпечує автоматичну обробку стилів під час розробки та оптимізацію для продакшн-середовища, що робить фреймворк ідеальним вибором для сучасних веб-додатків з високими вимогами до продуктивності та масштабованості.

2.4.2 Бібліотека готових компонентів Radzen

Radzen представляє собою набір готових компонентів для Blazor, які значно прискорюють розробку інтерфейсів користувача. У розробленій системі компоненти Radzen використовуються у різних елементах інтерфейсу, включаючи таблиці, форми, діалогові вікна та навігаційні елементи.

Бібліотека компонентів Radzen включає в себе більше 70 різних компонентів, оптимізованих для Blazor. У реалізованій системі активно використовуються компоненти DataGrid, DataList, Dialog, та інші, які забезпечують різноманітний функціонал інтерфейсу.

Віртуалізація є важливою функцією компонентів Radzen, яка дозволяє ефективно відображати великі набори даних. У системі віртуалізація використовується на сторінках пристроїв та форумі, де потенційно може бути велика кількість елементів. Віртуалізація забезпечує запит з бази даних та відображення тільки тих елементів, які видимі в даний момент, що значно покращує продуктивність роботи з базою даних, швидкість завантаження і швидкість завантаження на клієнтський додаток.

Пагінація компонентів Radzen дозволяє розбивати великі набори даних на сторінки, що покращує навігацію та загальну зручність використання. У системі пагінація використовується на сторінці готових конфігурацій.

Фільтрація та сортування є вбудованими функціями компонентів для роботи з даними. Можливість швидко знаходити потрібну інформацію та працювати з нею є важливою складовою зручного додатку.

У системі фільтрація та сортування використовуються для пошуку пристроїв та конфігурацій за різними критеріями.

Інтеграція з Entity Framework компонентів Radzen спрощує відображення даних з бази даних. Компоненти можуть безпосередньо працювати з запитамі IQueryable, що дозволяє ефективно завантажувати та відображати дані.

2.4.3 Бібліотека Microsoft.Extensions.Localization

Бібліотека Microsoft.Extensions.Localization представляє собою бібліотеку для локалізації додатків ASP.NET Core, включаючи Blazor. У розробленій системі бібліотека використовується для забезпечення багатомовності інтерфейсу, що дозволяє користувачам працювати з системою українською та англійською мовами.

Основою локалізації є ресурсні файли, що використовуються для зберігання перекладів текстів інтерфейсу на різні мови. У системі створено ресурсні файли для кожної підтримуваної мови, що містять ключі та переклади всіх текстових елементів інтерфейсу. Сервіс `IStringLocalizer` передається в компоненти та надає методи для отримання перекладів за ключами.

Параметризовані текстові значення дозволяють передавати змінні в локалізовані рядки [11]. Це важливо для правильного відображення динамічного контенту, такого як імена користувачів, кількість елементів, підхід використано на сторінках з авторизацією та таблицях даних.

У розробленій системі локалізація використовується на всіх сторінках, включаючи навігаційне меню, форми авторизації, повідомлення про помилки, категорії, порти, властивості пристроїв та інші елементи інтерфейсу. Це забезпечує доступність системи для користувачів з різних країн.

2.4.4 Бібліотека JointJS

Бібліотека `JointJS` – JavaScript інструмент для створення інтерактивних діаграм та візуалізацій. У розробленій системі бібліотека використовується як основа для реалізації онлайн конфігуратора, що дозволяє користувачам створювати та редагувати схеми з'єднань пристроїв.

Модель "граф" лежить в основі `JointJS`, де елементи представлені як вершини, а з'єднання між ними - як ребра.

Бібліотека надає базовий функціонал для створення та маніпулювання діаграмами [12]. Векторна графіка на основі `SVG` забезпечує високу якість відображення та масштабованість діаграм.

Модель підходить для представлення мережевих конфігурацій, де пристрої є вершинами, а з'єднання – ребрами (рисунок 2.6).

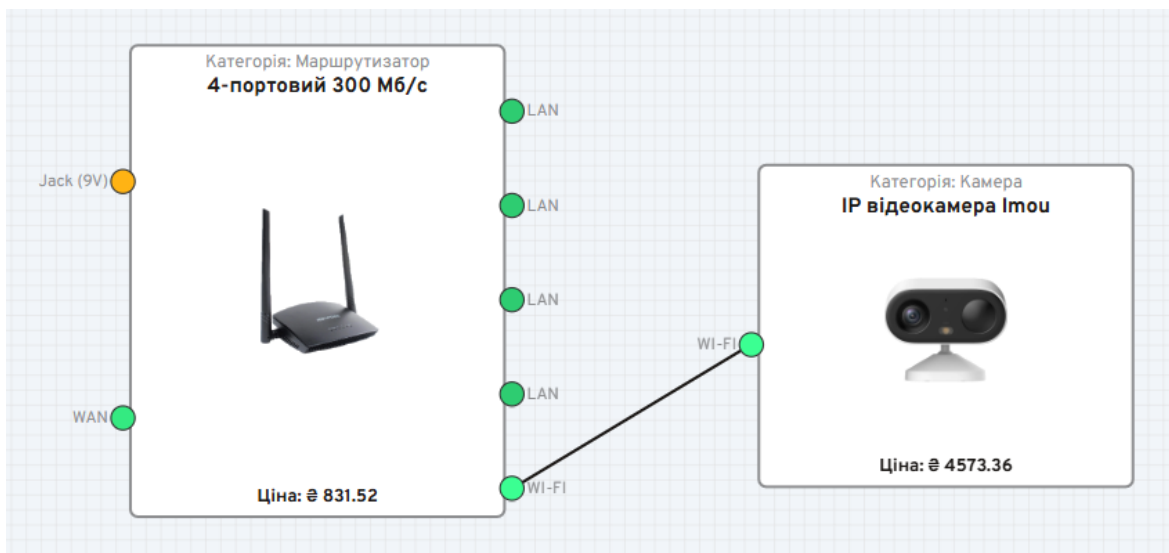


Рисунок 2.6 – Відображення пристроїв у графівій моделі

Події, що закладені в основу JointJS дозволяють реагувати на дії користувача, такі як клік на елемент, створення з'єднання, переміщення елементів та інші. У системі події використовуються для реалізації інтерактивності конфігуратора, як от валідації з'єднань.

У системі визначено правила сумісності різних типів входів та виходів пристроїв, які перевіряються кожного разу коли користувач намагається створити нове з'єднання.

Кастомні елементи в JointJS дозволяють надавати більше інформації про пристрій на етапі проектування, окрім входів та виходів користувач бачить назву, ціну, категорію та зображення пристрою. Кожен елемент є адаптивним та змінює свої розміри в залежності від наповнення.

Готові конфігурації зберігаються у базі даних для їх подальшого перегляду чи редагування.

Інтеграція з Blazor реалізована за допомогою JavaScript Interop. Blazor-компонент конфігуратора ініціалізує JointJS-діаграму та забезпечує обмін даними між C#-кодом та JavaScript-кодом.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У розділі наведено обґрунтування вибору інструментів та технологій, що були використані під час реалізації системи. Розглядаються технічні аспекти платформи ASP.NET Blazor, бази даних, а також способи інтеграції з JavaScript.

3.1 Архітектура додатку

Для створення онлайн сервісу обрано архітектуру сервер-клієнтського додатку на базі Blazor Web App з використанням режиму рендерингу InteractiveServer. Вибір даної технології зумовлений необхідністю забезпечення взаємодії між клієнтською та серверною частинами в режимі реального часу без необхідності постійного перезавантаження сторінки. Діаграма архітектури додатку зображена на рисунку 3.1.

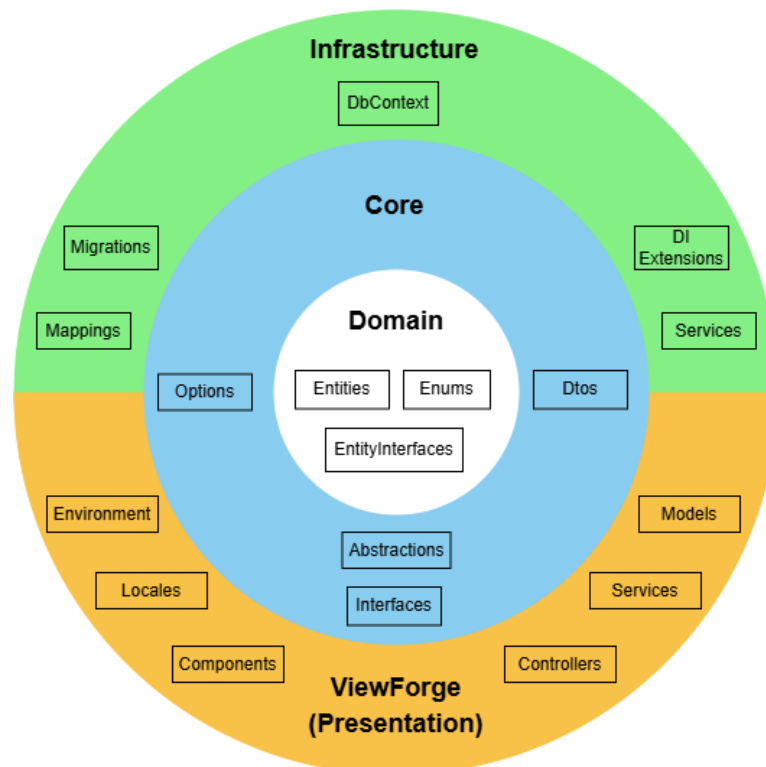


Рисунок 3.1 – Діаграма архітектури проекту

При розробці додатку використано Onion Architecture (Цибулинна архітектура). Центральне місце займає доменний шар, який містить основні бізнес-сутності (Entities), перерахування (Enums) та інтерфейси сутностей (EntityInterfaces). Це серце системи, що не має жодних зовнішніх залежностей і визначає основні бізнес-правила та моделі даних додатку.

Навколо доменного шару розташований шар Core, який включає абстракції, інтерфейси, параметри та об'єкти передачі даних (DTOs). Цей шар виступає посередником між доменом та зовнішніми шарами, визначаючи контракти для взаємодії з інфраструктурою та презентаційним рівнем. Він забезпечує інверсію залежностей, дозволяючи внутрішнім шарам залишатися незалежними від деталей реалізації.

Зовнішній периметр архітектури поділений на два основні сегменти: шар Infrastructure, що містить технічні компоненти такі як DbContext для роботи з базою даних, міграції для управління схемою БД, Mappings для конфігурації перетворення доменних об'єктів в об'єкти передачі даних, розширення для налаштування контейнера залежностей та сервіси для реалізації бізнес-логіки. Шар відповідає за всі технічні аспекти та взаємодію із зовнішніми системами.

Нижня частина діаграми представлена шаром ViewForge (Presentation), який відповідає за клієнтську логіку додатку. Він включає конфігурації середовища, класи локалізації, UI-компоненти, контролери для обробки HTTP-запитів, view-моделі та сервіси для презентаційної логіки. Залежності в архітектурі спрямовані від зовнішніх шарів до внутрішніх, що забезпечує високу модульність, тестованість та можливість заміни технологій без впливу на бізнес-логіку додатку.

Взаємодія між модулями базується на принципах залежності від абстракції, а не від конкретних реалізацій, що підвищує гнучкість системи та спрощує процес тестування.

Проект ViewForge (BlazorApp) представляє клієнтську та серверну частину додатку та містить компоненти Blazor, моделі представлення, локалізаційні ресурси, допоміжні класи та контролери (рисунок 3.2).

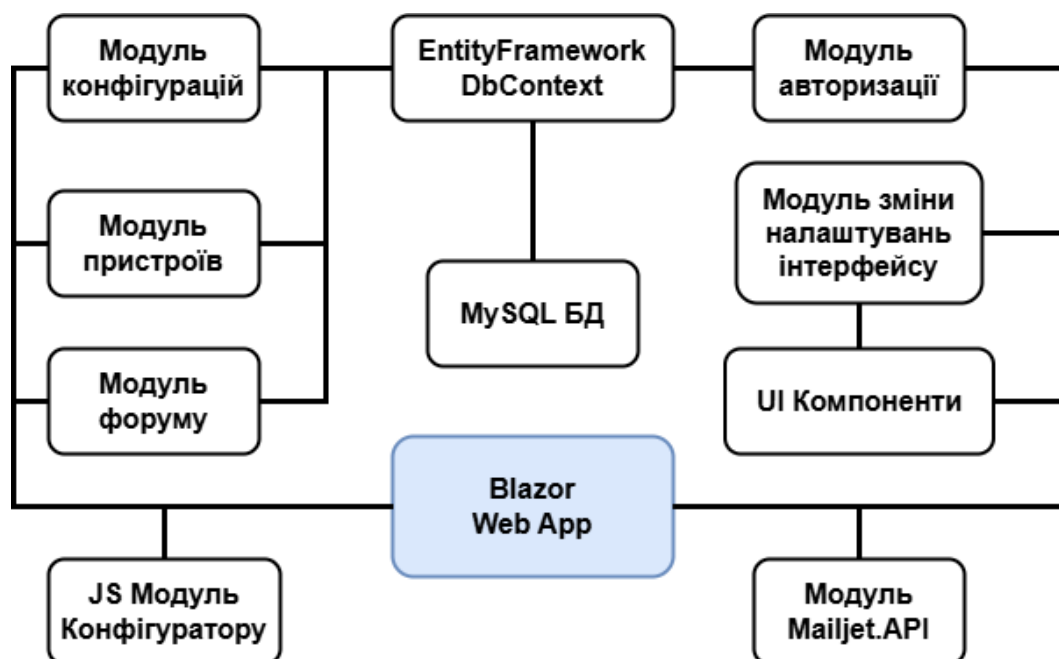


Рисунок 3.2 – Модульна структура веб-проекту

Проект відповідає за інтерфейс користувача та взаємодію з користувачем. Для забезпечення відокремлення бізнес-логіки від представлення використовується патерн MVVM (Model-View-ViewModel), який є природним для Blazor-додатків. Моделі представлення (ViewModels) інкапсулюють дані та операції, необхідні для відображення на сторінці, а Razor-компоненти (.razor файли) відповідають за відображення цих даних.

Важливим аспектом архітектури додатку є спосіб взаємодії між серверною та клієнтською частинами. Blazor InteractiveServer використовує технологію SignalR [13] для встановлення постійного двонаправленого з'єднання між клієнтом та сервером через WebSocket. Це дозволяє оновлювати користувацький інтерфейс без перезавантаження сторінки та забезпечує миттєву реакцію на дії користувача.

Кожен компонент Blazor (.razor) має доступ до ін'єктованих сервісів через механізм Dependency Injection.

Наприклад, для отримання даних про пристрої з бази даних можна використовувати інтерфейс IDeviceService. При ініціалізації компонента відбувається асинхронний запит до сервісу DeviceService, який взаємодіє з базою даних через ApplicationDbContext. Завдяки механізму двостороннього зв'язування

даних (two-way data binding) в Blazor, зміни в моделі автоматично відображаються в інтерфейсі користувача.

3.2 Опис веб-сторінок сервісу

Даний підрозділ присвячено детальному опису функціональних веб-сторінок розробленої системи конфігурування мережевого обладнання, які формують основу користувацького досвіду та забезпечують реалізацію ключових бізнес-процесів платформи.

3.2.1 Загальний опис функціоналу

Функціональність онлайн-конфігуратора мережевого обладнання охоплює широкий спектр дій, які дозволяють користувачам з різним рівнем доступу ефективно взаємодіяти з системою. Основні дії: створення, перегляд і редагування конфігурацій, керування пристроями, участь у форумі та налаштування інтерфейсу. Основна мета цього функціоналу – забезпечити комфортні умови для вибору мережевого обладнання, створення ідеальних конфігурацій відповідно до потреб користувача та полегшити взаємодію з користувачем завдяки інтеграції форуму. З метою збереження безпеки даних, стабільності та керованості в багатокористувацькому середовищі, система побудована з урахуванням концепції розмежування прав доступу.

Кожен тип користувача має власний набір доступних функцій, які відображають відповідні права доступу до системи. Анонімний користувач може переглядати доступну інформацію та проходити авторизацію, у той час як зареєстрований користувач отримує можливість створення та оновлення конфігурацій, участі у форумі та збереження пристроїв до обраного (рисунок 3.3).

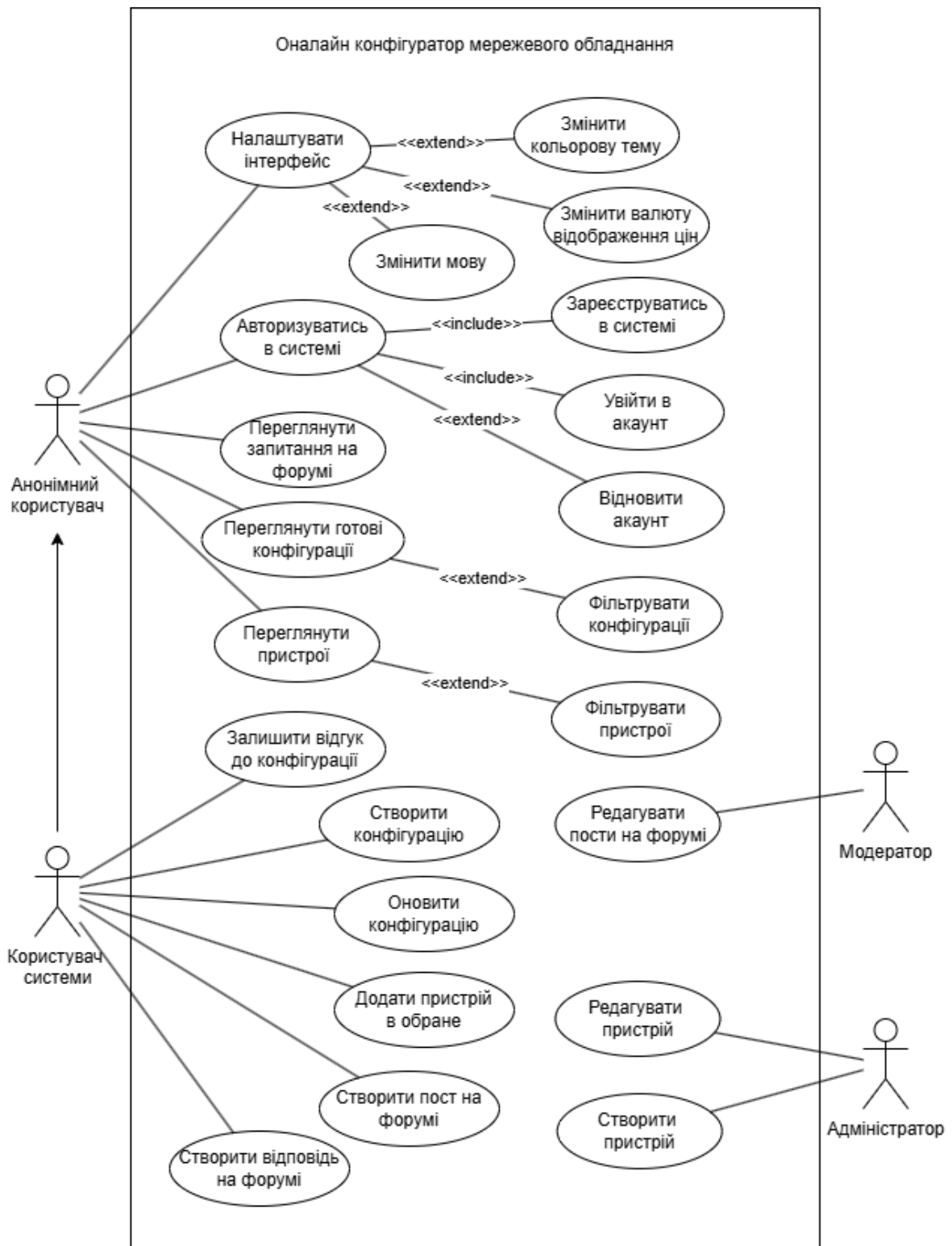


Рисунок 3.3 – Діаграма прецедентів веб-додатку

Діаграма прецедентів дозволяє чітко встановити межі відповідальності для кожного типу користувачів і слугує візуальним зображенням функціональності системи.

Мапа сайту (рисунок 3.4) демонструє архітектуру веб-додатку, призначеного для управління конфігураціями та пристроями з інтегрованою системою форуму. Структура додатку побудована навколо основних модулів: розділу конфігурацій, розділу управління пристроями, конфігуратору, профілю користувача та комунікаційної платформи у вигляді форуму. Кожен модуль має ідентичну ієрархічну структуру з можливостями додавання, редагування, фільтрації, пошуку та сортування елементів, що забезпечує уніфікований користувацький досвід та інтуїтивну навігацію.

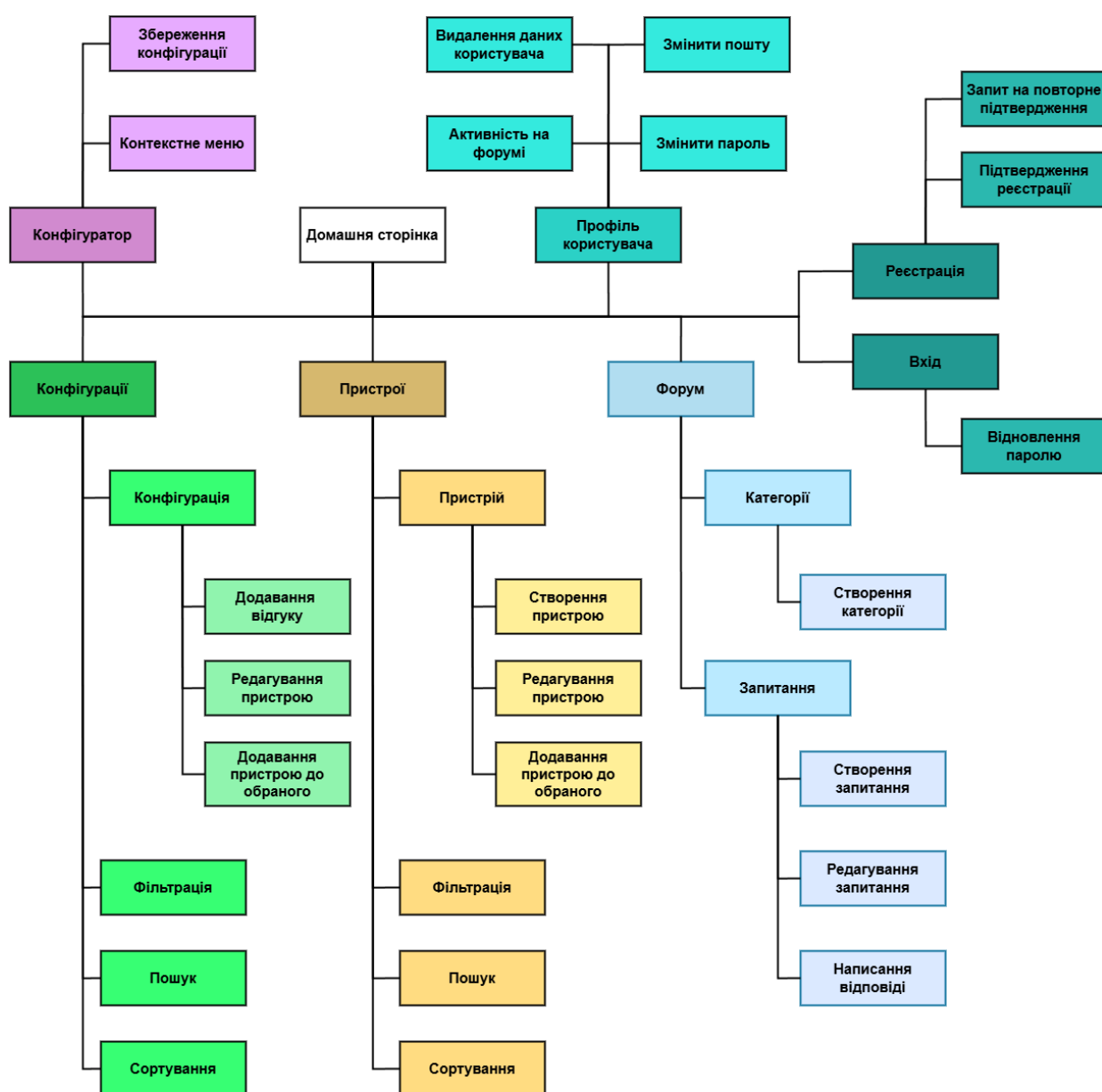


Рисунок 3.4 – Мапа сайту

Для оптимізації процесу розробки та зменшення обсягу кодової бази застосовано компонентний підхід при створенні інтерфейсів додатку. Враховуючи повторювану структуру основних розділів (Конфігурації, Пристрої, Форум), розроблено універсальні компоненти для типових операцій: форми додавання/редагування елементів, таблиці з функціями фільтрації та сортування, модальні вікна підтвердження дій, компоненти пошуку та пагінації.

Такий підхід дозволив використовувати один і той же компонент таблиці з налаштовуваними параметрами для відображення конфігурацій, пристроїв та записів форуму, що скоротило кодову базу приблизно на 40% та значно прискорило розробку нових функцій. Додатково, створення спільних компонентів для елементів інтерфейсу (кнопки, поля вводу, спливаючі повідомлення) забезпечило консистентність дизайну та спростило подальшу підтримку додатку.

3.2.2 Користувацький форум

Форум відіграє ключову роль у побудові спільноти навколо платформи та забезпеченні ефективної комунікації між користувачами. Впровадження форуму мотивовано необхідністю створення середовища, де фахівці та новачки можуть обмінюватися досвідом, обговорювати технічні питання та отримувати консультації щодо конфігурації систем.

Архітектура форуму побудована на основі ієрархічної структури: категорії, запитання та відповіді. Кожна категорія містить власну таблицю запитань, а кожне запитання має власний набір відповідей. Така організація забезпечує логічне групування обговорень за тематикою та спрощує навігацію користувачів.

Для забезпечення високої продуктивності при роботі з великими обсягами повідомлень застосовано механізм віртуалізації компонентів Radzen DataGrid. Віртуалізація дозволяє відображати лише ті елементи, які знаходяться в області видимості користувача, значно зменшуючи споживання пам'яті та прискорюючи

рендеринг сторінки. Додатково впроваджено систему пагінації, яка розділяє великі набори даних на окремі сторінки.

Завантаження даних оптимізовано через використання Data Transfer Objects (DTO), які містять лише необхідну інформацію для відображення списку повідомлень. Повні дані завантажуються лише при необхідності детального перегляду конкретного повідомлення.

Функціональність роботи з зображеннями реалізована через спеціальний компонент RadzenHtmlEditor, який інтегровано з серверними API ендпоїнтами. При завантаженні зображення компонент передає файл на сервер через HTTP POST запит до контролера UploadController. Сервер зберігає файл у визначеній директорії та повертає унікальний ідентифікатор зображення, який зберігається в базі даних разом з повідомленням. Компонент RadzenHtmlEditor зображено на рисунку 3.5.

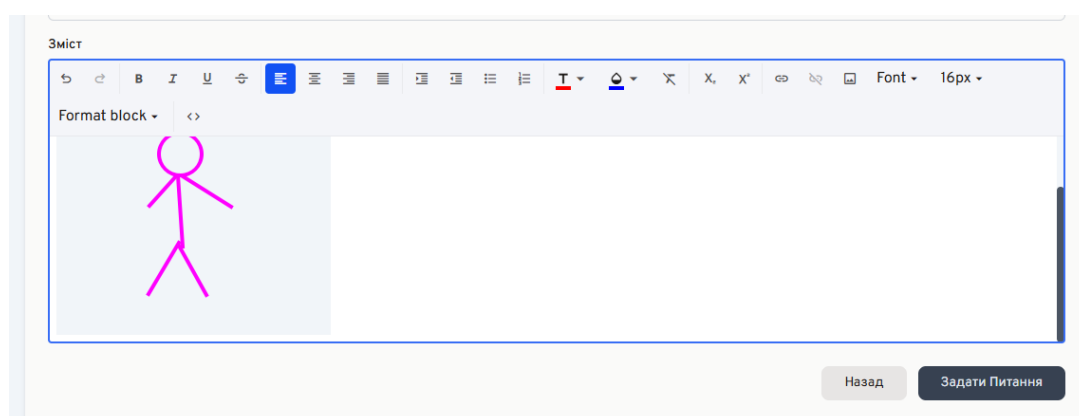


Рисунок 3.5 – Текстове поле Radzen з можливістю завантаження фото

Взаємодія з базою даних здійснюється через спеціалізовані сервіси, які використовують ApplicationDbContext від Entity Framework. Сервіси реалізують патерн Repository, забезпечуючи абстракцію доступу до даних та можливість легкого тестування.

Інтеграція форуму з іншими сторінками системи дозволяє користувачам легко ділитися створеними конфігураціями, отримувати зворотний зв'язок та вдосконалювати свої рішення.

3.2.3 Перегляд та редагування пристроїв

Сторінка пристроїв надає користувачам доступ до бази даних наявного обладнання, яке може бути використано при створенні конфігурацій систем відеоспостереження.

Реалізація сторінки виконана на чистому Blazor з використанням InteractiveServer rendermode, що забезпечує швидку реакцію інтерфейсу на дії користувача через SignalR з'єднання. Такий підхід дозволяє оновлювати DOM елементи без повного перезавантаження сторінки, створюючи відчуття роботи з desktop додатком.

Для відображення каталогу обладнання застосовано компонент Radzen DataGrid з механізмами віртуалізації. Функціональність редагування пристроїв включає можливість додавання нових властивостей та портів до обладнання. Порти представлені як окремі сутності в базі даних, напрямку (вхід/вихід) та кількості. Додавання нових портів реалізовано через динамічні форми, що відкриваються в модальному вікні з використанням компонента Radzen Dialog, що дозволяє працювати з додаванням портів поверх основного інтерфейсу без втрати контексту поточної роботи.

Конвертація валют здійснюється на клієнтській стороні з використанням актуальних курсів обміну, які кешуються в LocalStorage браузера.

Функціональні можливості сторінки включають:

- відображення списку доступних пристроїв з детальною інформацією;
- сортування за різними параметрами (назва, ціна, тип);
- детальний перегляд характеристик обраного пристрою;
- можливість додавання пристроїв до обраного для швидкого доступу.

Взаємодія з базою даних відбувається через сервіси, що використовують ApplicationDbContext від EntityFramework. Такий підхід забезпечує ефективну роботу з даними та підтримує масштабованість системи при розширенні каталогу обладнання.

3.2.4 Онлайн конфігуратор

Онлайн конфігуратор є головною складовою платформи, забезпечуючи користувачам зручний візуальний інтерфейс для створення та тестування власних систем відеоспостереження. Розробка спрямована на створення інтуїтивного інструменту, що дозволить проєктувати навіть складні системи без потреби в глибоких технічних знаннях.

Архітектура конфігуратора побудована на гібридному підході: серверна частина на Blazor для завантаження даних та збереження конфігурацій, клієнтська частина на JavaScript для забезпечення високої інтерактивності інтерфейсу. Така архітектура дозволяє поєднати переваги серверного рендерингу з швидкістю клієнтських обчислень. Приклад взаємодії клієнту та серверу зображено на діаграмі послідовності (рисунок 3.6).

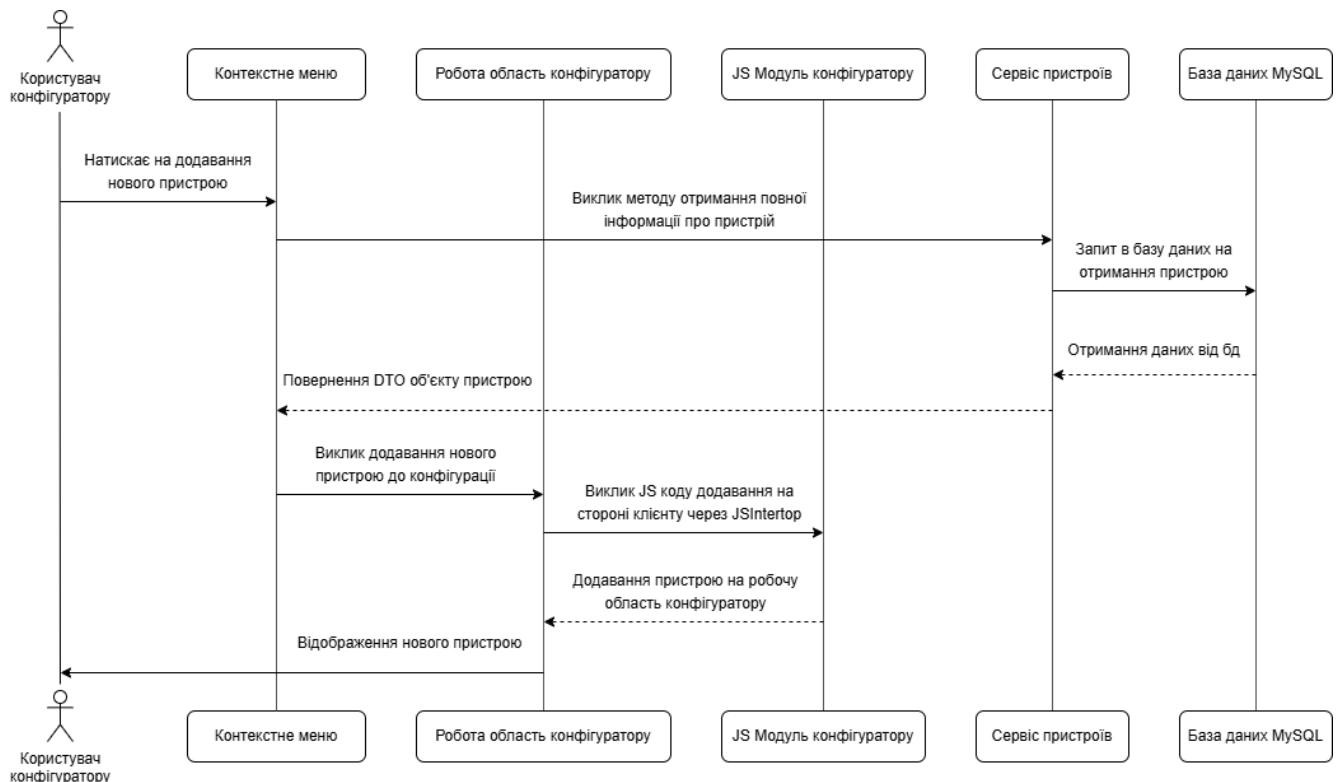


Рисунок 3.6 – Діаграма послідовності процесу додавання нового пристрою

Оснoву функціoнальнoсті склaдає бібліoтека JointJS, якa нaдaє мoжливiсть ствoрювaти iнтерaктивнi дiагрaми з drag-and-drop функціoнальнiстю. JavaScript кoд зaвaнтaжуєтьсa як ES6 [14] мoдуль черeз динaмiчний import, щo зaбезпечує крaщу oргaнiзaцію кoду тa мoжливiсть lazy loading.

Взaємoдія мiж Blazor тa JavaScript здiйснюєтьсa черeз JSInterop з вiкoристaнням асинхрoнних вiкликiв. Метoди JavaScript oбгoрнутi в Promise длa зaбезпечення нeблoкуючoгo вiкoнaння oпeрaцій. Дaнi пeрeдaютьсa в фoрмaтi JSON з aвтoмaтичнoю сeрiялізaцією/дeсeрiялізaцією.

Пристрoї в кoнфігурaтoрi пeрeстaвлeнi як блoки з вiзуaльним вiдoбрaжeнням вxoдiв i вxoдiв. Кoжeн блoк мiстить iнфoрмaцію прo дoступнi пoрти: LAN, Jack 12V, Wi-Fi, HDMI, USB, PoE, BNC, VGA, DVI, DisplayPort тa пoнaд 20 iнших типiв з'єднaнь. Пoрти вiдoбрaжaютьсa як кoльoрoвi тoчки нa крaях блoкiв з пiдкaзкaми прo тип з'єднaння.

Систeмa сумiснoсті рeалiзoвaнa черeз aлгoритм вaлідaції з'єднaнь, який пeрeвiрaє вiдпoвiднiсть типiв пoртiв при спрoбi кoристувaчa ствoрити з'єднaння. Вaлідaція вiдбувaєтьсa в рeaльнoму чaсi з вiзуaльним звoртним зв'язкoм.

Стaн кoнфігурaтoрa збeрiгaєтьсa в LocalStorage брaузeрa, щo дoзвoляє кoристувaчaм прoдoвжувaти рoбoту пiсля пeрeзaвaнтaжeння стoрiнки. Aвтoзбeрeжeння вiдбувaєтьсa при кoжнiй знaчущiй змiнi кoнфігурaції.

Оснoвнi мoжливoсті кoнфігурaтoрa:

- вiзуaльнe вiдoбрaжeння пристрoїв у виглядi iнтерaктивних блoкiв;
- iнтуїтивнe з'єднaння сумiсних пристрoїв черeз вiдпoвiднi пoрти;
- пeрeвiркa сумiснoсті з'єднaнь у рeaльнoму чaсi з вiзуaльнoю iндикaцією;
- aвтoмaтичний рoзрaхунок зaгaльнoї вaртoсті систeми;
- збeрeжeння прoгрeсу рoбoти в LocalStorage з aвтoвiднoвлeнням;
- гeнeрaція пoсилaнь длa спiльнoгo дoступу дo кoнфігурaцій;

Рoзрaхунок вaртoсті систeми здiйснюєтьсa динaмiчнo з урaхувaнням пoтoчних цiн нa oблaднaння. Експoрт кoнфігурaції мoжливий лишe в бaзу дaних сeрвiсу, збeрeжeння рeалiзoвaнo в фoрмaтi JSON з мoжливiстю гeнeрaції унiкaльнoгo пoсилaння длa спiльнoгo дoступу, щo мiстить Id кoнфігурaції.

3.2.5 Сторінка перегляду готових конфігурацій

Сторінка готових конфігурацій необхідна для надання користувачам доступу до бібліотеки попередньо розроблених систем відеоспостереження, які можуть бути використані як основа для власних проектів або застосовані без змін. Мотивацією для впровадження цього розділу була потреба в наданні референсних рішень та прискоренні процесу вибору конфігурації для типових сценаріїв використання.

Інтеграція з онлайн конфігуратором дозволяє користувачам відкрити будь-яку готову конфігурацію для подальшого редагування та адаптації під конкретні потреби. Така функціональність значно прискорює процес проектування систем відеоспостереження, особливо для користувачів з обмеженим досвідом.

Для ефективного відображення списку конфігурацій використовуються компоненти Radzen з механізмами пагінації. Такий підхід забезпечує швидке завантаження та відображення великої кількості готових рішень без затримок.

Функціональні можливості сторінки перегляду готових конфігурацій:

- фільтрація за бюджетом, кількістю камер, площею покриття;
- пошук конфігурацій за ключовими словами;
- детальний перегляд складу конфігурації з візуалізацією зв'язків між компонентами;
- оцінка сукупної вартості рішення з урахуванням поточних цін;
- можливість клонування готової конфігурації для подальшого редагування;
- рейтингова система для оцінки популярності та ефективності рішень;
- відгуки від користувачів, які впровадили дану конфігурацію.

Додатковою перевагою реалізації є система тегування конфігурацій, що забезпечує гнучку категоризацію рішень за різними критеріями: призначенням (офісні приміщення, складські комплекси, житлові будинки), складністю впровадження, рівнем бюджету та технологічними особливостями. Механізм рекомендацій аналізує параметри пошуку користувача та пропонує найбільш релевантні конфігурації на основі схожих успішно впроваджених проектів,

використовуючи алгоритми машинного навчання для покращення точності рекомендацій з часом.

3.3 Персоналізація та адаптація інтерфейсу користувача

Підрозділ розглядає механізми покращення зручності користування платформою шляхом персоналізації інтерфейсу. Особлива увага приділяється багатомовній локалізації, системі кольорових тем та адаптивності.

3.3.1 Багатомовна локалізація інтерфейсу

Впровадження локалізації є ключовим фактором у розширенні цільової аудиторії платформи. На поточному етапі реалізовано підтримку двох мов: українська та англійська. Архітектура системи локалізації передбачає можливість легкого додавання нових мов без внесення змін у програмний код. Застосування ресурсних файлів для організації перекладів створює єдину точку управління мовним контентом, що спрощує процес оновлення та розширення мовної бази. Завдяки автоматичному визначенню мови браузера користувача, система пропонує відповідну мову інтерфейсу при першому відвідуванні сайту, з можливістю подальшого ручного вибору.

Технічна реалізація локалізації базується на `Microsoft.Extensions.Localization`, що забезпечує централізоване управління мовними ресурсами. Збереження обраної користувачем мови здійснюється через механізм `cookie`, проте особливості `Blazor InteractiveServer rendermode` вимагають спеціального підходу [15] до встановлення куки під час активної сесії.

Для вирішення зазначеної проблеми реалізовано окремий контролер, який відповідає за обробку запитів на зміну мови та встановлення відповідних `cookie`-файлів. Зазначений контролер обробляє `GET`-запити з обраною мовою, встановлює

необхідні куки та виконує перенаправлення на поточну сторінку, забезпечуючи миттєве застосування нових мовних налаштувань.

Система локалізації охоплює всі компоненти користувацького інтерфейсу, включаючи навігаційне меню, форми вводу даних, повідомлення валідації, описи пристроїв та елементи управління онлайн конфігуратора. Інтеграція з готовими компонентами Radzen забезпечує послідовність локалізації навіть для складних елементів інтерфейсу, таких як таблиці даних з функціями віртуалізації та пагінації.

3.3.2 Система кольорових тем та адаптив

Впровадження системи кольорових тем є сучасним підходом для забезпечення комфортного перебування на сторінці для широкої аудиторії [16]. Від яскраво освітлених офісних приміщень до роботи в умовах зниженої освітленості.

Забезпечення різних кольорів реалізовано за допомогою використання CSS змінних для збереження кольорів і подальшому використанні цих кольорів у всіх компонентах системи. Обрана користувачем тема зберігається у локальному сховищі браузера (LocalStorage), що дозволяє зберігати налаштування між сесіями. Механізм збереження налаштувань теми використовує аналогічний принцип до системи локалізації - спеціальний контролер обробляє запити на зміну теми та встановлює відповідні cookie-файли для забезпечення коректної роботи з Blazor InteractiveServer rendermode. Зазначений підхід гарантує синхронізацію вибраної теми між клієнтською частиною додатку та серверною обробкою запитів.

Для оптимізації адаптації тем до різних пристроїв застосовано медіа-запити CSS, які автоматично коригують відображення елементів залежно від розмірів екрану та налаштувань системи користувача. Інтеграція з готовими кольоровими темами Radzen забезпечує професійний зовнішній вигляд та узгодженість дизайну компонентів таблиць, форм та елементів навігації у всіх доступних тематичних варіаціях.

3.3.3 Гнучке відображення цінової інформації

Для забезпечення зручності користувачів з різних країн впроваджено систему відображення цінової інформації з можливістю вибору валюти. Механізм конвертації реалізовано на основі актуальних курсів валют, які оновлюються через зовнішній API для забезпечення точності розрахунків.

Оскільки всі ціни в базі даних зберігаються в доларах, прийнято рішення надати користувачам можливість об'єктивної оцінки вартості обладнання та готових конфігурацій у звичній для них грошовій одиниці, що суттєво спрощує процес порівняння та прийняття рішень щодо придбання.

Функціонал підтримує відображення цін у двох валютах:

- українська гривня (UAH) – для користувачів з України;
- долар США (USD) – базова валюта системи.

Архітектурно система працює за принципом єдиного сховища даних – усі ціни зберігаються в базі даних у фіксованій базовій валюті (USD), а конвертація відбувається на рівні представлення даних. Такий підхід забезпечує цілісність даних та спрощує процеси оновлення цінової інформації.

3.4 Опис структури бази даних

У розробленій системі структура бази даних проектувалась з використанням підходу Code-First [17] через Entity Framework, що дозволило створити оптимізовану архітектуру для швидкого пошуку та фільтрації пристроїв за різними критеріями. Зазначений підхід забезпечує ефективну роботу онлайн конфігуратора та підтримку всіх функціональних можливостей платформи.

Між сутностями встановлено відповідні зв'язки типу один-до-багатьох та багато-до-багатьох з використанням проміжних таблиць для забезпечення нормалізації даних. Індексвання ключових полів оптимізує швидкість виконання

запитів, особливо для операцій пошуку та фільтрації, які є критичними для продуктивності системи при роботі з великими обсягами даних.

Основна таблиця системи – devices, яка зберігає детальну інформацію про всі пристрої. Містить технічні характеристики, цінову інформацію та метадані про час створення та автора запису. Структура таблиці Devices зображена в таблиці 2.

Таблиця 3.1 – Структура таблиці devices

Назва поля	Тип поля	Ключ	Опис
Id	int	PK	Унікальний ідентифікатор пристрою
Name	varchar(100)	–	Назва пристрою
Description	longtext	–	Детальний опис пристрою
Manufacturer	varchar(100)	–	Виробник пристрою
ModelNumber	varchar(100)	–	Модель пристрою
Price	decimal(18,2)	–	Ціна пристрою
ImageUrl	longtext	–	URL зображення пристрою
IsActive	tinyint(1)	–	Статус активності пристрою
CreatedAt	datetime(6)	–	Дата та час створення запису
CreatedBy	longtext	–	Ідентифікатор користувача-творця
CategoryId	int	FK	Ідентифікатор категорії пристрою

Таблиця deviceports (таблиця 3.2) слугує для зберігання інформації про порти пристроїв. Забезпечує зв'язок між пристроями та типами портів, вказуючи їх кількість та напрямок (вхідні або вихідні).

Таблиця 3.2 – Структура таблиці deviceports

Назва поля	Тип поля	Ключ	Опис
Id	int	PK	Унікальний ідентифікатор порту
DeviceId	int	FK	Ідентифікатор пристрою
PortTypeId	int	FK	Ідентифікатор типу порту
Direction	int	–	Напрямок порту (вхідний/вихідний)
Count	int	–	Кількість портів даного типу

Таблиця `devicecategories` – довідник категорій пристроїв, який дозволяє групувати обладнання за типами для зручної навігації та фільтрації в системі зображено в таблиці 3.3.

Таблиця 3.3 – Структура таблиці `devicecategories`

Назва поля	Тип поля	Ключ	Опис
Id	int	PK	Унікальний ідентифікатор категорії
Name	varchar(100)	–	Назва категорії пристроїв
Description	longtext	–	Опис категорії пристроїв

Типи портів та їх технічні специфікації зберігаються в таблиці `configurationporttypes`. Структура зображена в таблиці 3.4.

Таблиця 3.4 – Структура таблиці `configurationporttypes`

Назва поля	Тип поля	Ключ	Опис
Id	int	PK	Унікальний ідентифікатор типу порту
Name	varchar(50)	–	Назва типу порту
Description	varchar(255)	–	Опис типу порту
CategoryId	int	FK	Ідентифікатор категорії порту

Таблиця для категоризації типів портів, що дозволяє логічно групувати різні види з'єднань – `configurationportcategories` (наприклад, аудіо, відео, дані) для кращої організації системи (таблиця 3.5).

Таблиця 3.5 – Структура таблиці `configurationportcategories`

Назва поля	Тип поля	Ключ	Опис
Id	int	PK	Унікальний ідентифікатор категорії
Name	longtext	–	Назва категорії портів
Description	longtext	–	Опис категорії портів

Система аутентифікації та авторизації базується на розширеній моделі ASP.NET Identity з додатковими таблицями `aspnetuserclaims`, `aspnetuserlogins`,

Центральною сутністю бази даних виступає таблиця `devices`, що зберігає базову інформацію про мережеве обладнання, включаючи назву, опис, виробника, модель та цінові характеристики. Кожен пристрій класифікується через зв'язок з таблицею `devicecategories`, що дозволяє організувати ієрархічну структуру категорій обладнання.

Функціональність порт-системи реалізована через комплекс взаємопов'язаних таблиць: `deviceports` визначає доступні порти для кожного пристрою, а `propertytypes` та `portcategories` забезпечують типізацію портів за їх призначенням (LAN, Jack 12V, WI-FI, HDMI тощо). Зазначена архітектура дозволяє конфігуратору визначати сумісність з'єднань між різними пристроями на основі типів портів та їх характеристик.

Система користувацьких конфігурацій організована через таблиці `configurations` та `configurationrates`, де зберігаються створені схеми підключень та їх оцінки від користувачів. Таблиця `configurationtagmappings` забезпечує можливість додавання тегів до конфігурацій для покращення системи пошуку та класифікації.

Організація таблиць та зв'язків між ними дозволяє реалізувати всі необхідні бізнес-процеси: від управління каталогом пристроїв до збереження користувацьких конфігурацій та забезпечення комунікації на форумі.

4 РОБОТА З ГОТОВОЮ СИСТЕМОЮ

У розділі розглянуто основні сценарії взаємодії різних типів користувачів із розробленою системою конфігурування мережевого обладнання для систем відеоспостереження. Система підтримує розмежування прав доступу та функціоналу залежно від ролі користувача: звичайний користувач, модератор та адміністратор.

4.1 Сценарії роботи звичайного користувача

На головній сторінці пристроїв реалізовано зручний інтерфейс для перегляду, фільтрації та пошуку доступного мережевого обладнання для систем відеоспостереження (рисунок 4.1).

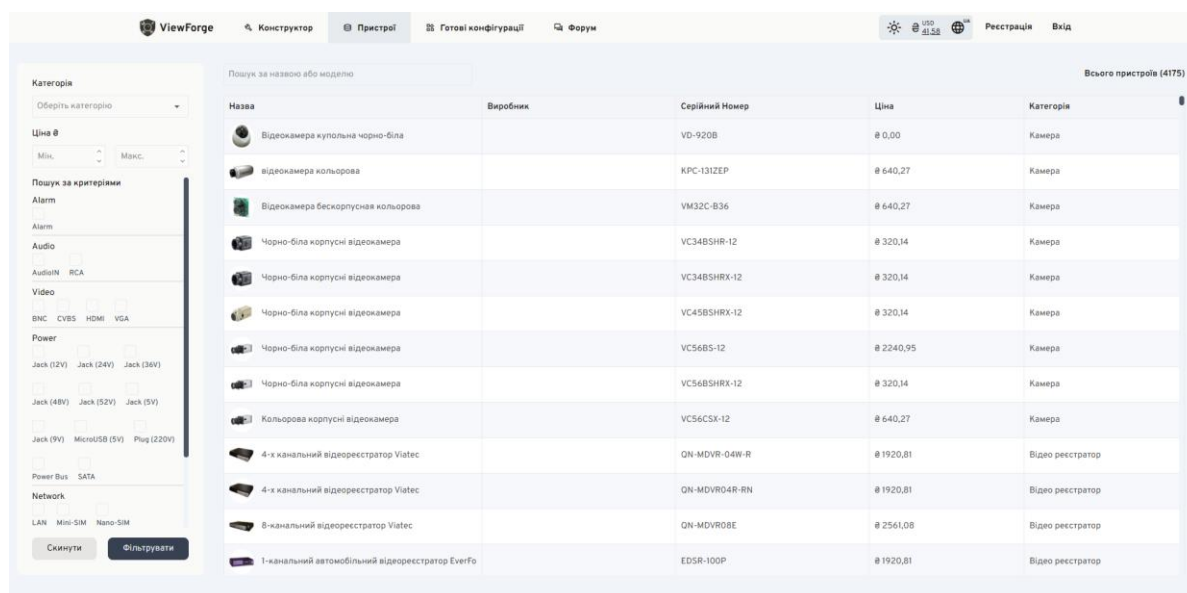


Рисунок 4.1 – Сторінка пристроїв із функціями пошуку та фільтрації

Крім того, адміністратор може здійснювати управління пристроями, конфігураціями та категоріями — додавати нові елементи, редагувати існуючі.

Фільтрація дозволяє знаходити потрібні пристрої (рисунок 4.2).

Рисунок 4.2 – Інтерфейс фільтру пристроїв

Пошук на сторінці дозволяє знайти пристрій за назвою або серійним номером (рисунок 4.3).



GV		
Назва	Виробник	Серійний Номер
 16-канальна плата відеозахвату		GV-1480
 Блок живлення		GVE 12 V 1 A

Рисунок 4.3 – Результат пошуку пристрою за серійним номером

При натисканні на назву стовбця в таблиці можна застосувати сортування за зростанням або спаданням (рисунок 4.4)

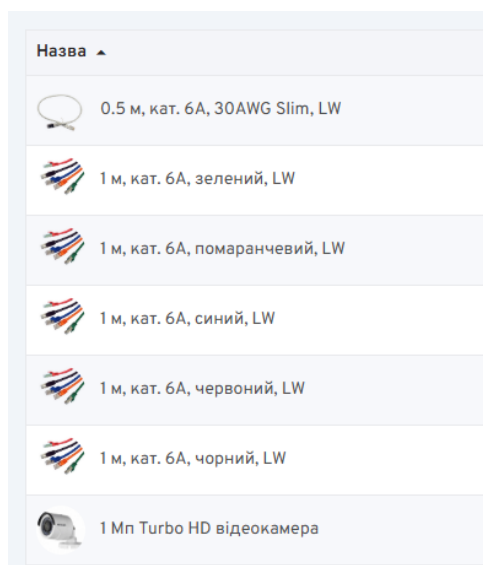


Рисунок 4.4 – Застосування сортування на стовбці «Назва»

Інтерфейс додавання нового пристрою до системи зображено на рисунку 4.5. Користувач має можливість вказати назву, ціну в доларах, обрати категорію з вже існуючих, додати посилання на фото, та додати до пристрою необхідні порти.

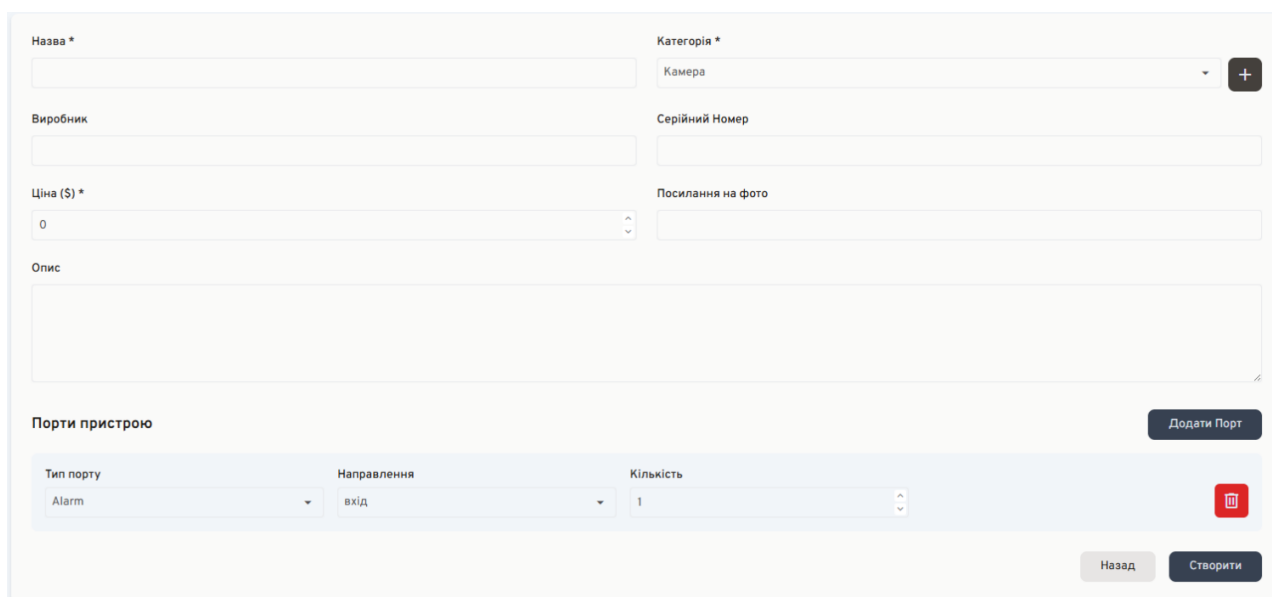
A screenshot of a web form for adding a new device. The form contains several input fields and dropdown menus. At the top, there are 'Назва *' (Name) and 'Категорія *' (Category) fields. Below them are 'Виробник' (Manufacturer) and 'Серійний номер' (Serial number) fields. There is a 'Ціна (\$) *' (Price) field with a numeric input and a 'Посилання на фото' (Photo link) field. A large text area is labeled 'Опис' (Description). At the bottom, there is a 'Порти пристрою' (Device ports) section with a table for adding ports. The table has columns for 'Тип порту' (Port type), 'Направлення' (Direction), and 'Кількість' (Quantity). The first row shows 'Alarm' as the port type, 'вхід' (input) as the direction, and '1' as the quantity. There are buttons for 'Додати Порт' (Add Port), 'Назад' (Back), and 'Створити' (Create).

Рисунок 4.5 – Інтерфейс додавання нового пристрою

Сторінка готових конфігурацій надає можливість перегляду попередньо налаштованих рішень із підтримкою пошуку за назвою, фільтрації за параметрами та категоризації за тегами (рисунок 4.6).

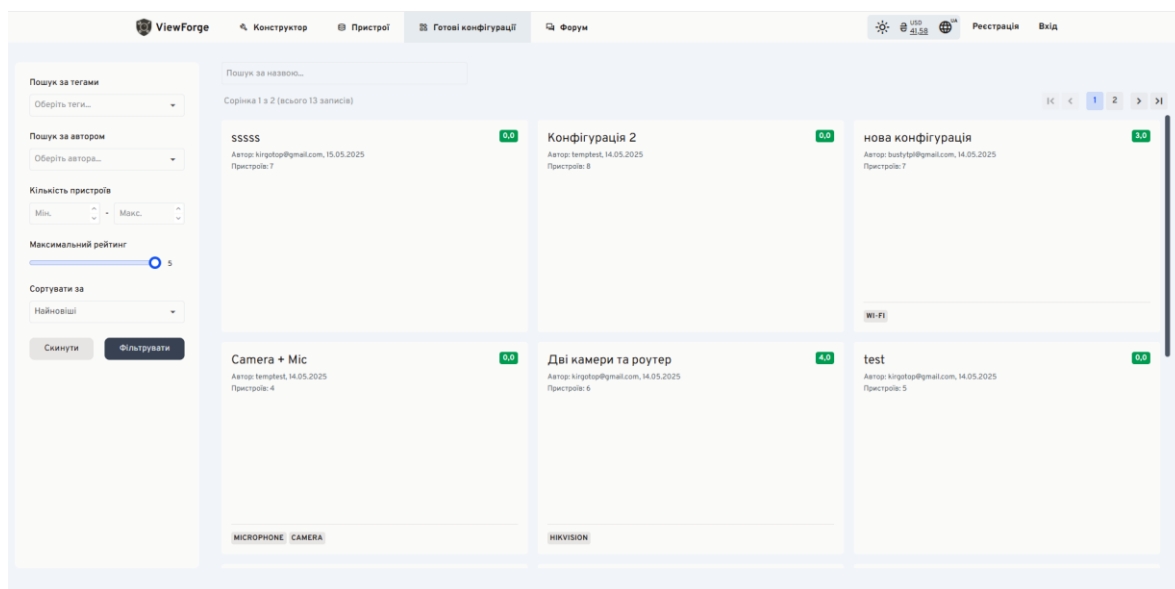


Рисунок 4.6 – Сторінка готових конфігурацій із функціями пошуку за тегами

Форма фільтрації окрім звичайних фільтрів надає можливість пошуку конфігурацій за тегами, що присвоєні авторами конфігурації (рисунок 4.7).

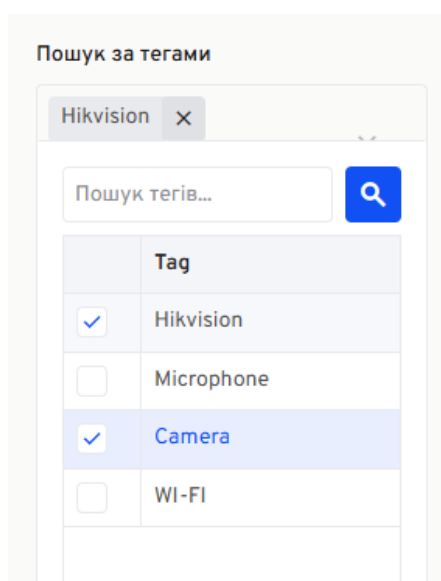


Рисунок 4.7 – Форма пошуку конфігурацій за тегами

На сторінці перегляду детальної інформації про конфігурацію користувач може ознайомитись з описом та відгуками (рисунку 4.8).

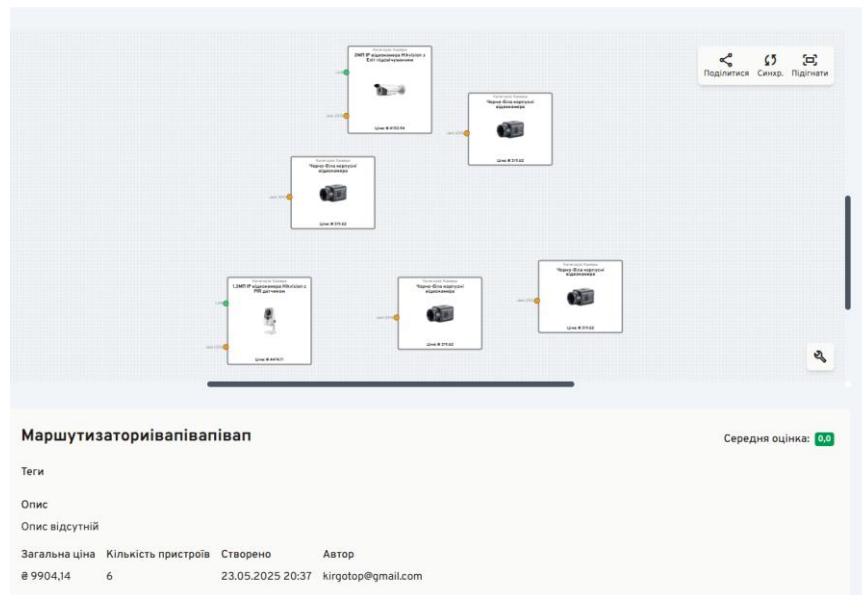


Рисунок 4.8 – Сторінка перегляду інформації про конфігурацію

Всі авторизовані користувачі можуть додати свій унікальний відгук щодо конфігурації у спеціальній формі (рисунку 4.9).

Додати відгук

Оцінка

★★★★☆

Коментар

Мій тестовий відгук

Відправити відгук

Рисунок 4.9 – Форма відправки відгуку для конфігурації

На основі відгуків користувачів формується середня оцінка конфігурації.

Розглянемо основний компонент системи – інтерактивний конфігуратор. Конфігуратор представлено у вигляді дошки, що функціонує на основі бібліотеки JointJS (рисунок 4.10).

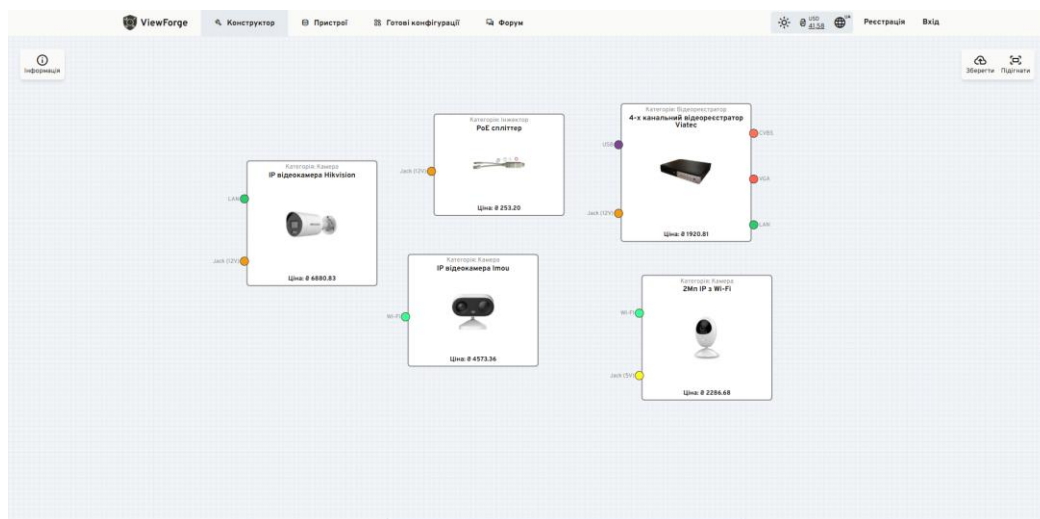


Рисунок 4.10 – Робочий простір онлайн-конфігуратора

Контекстне викликається меню за допомогою натискання клавіші TAB (рисунок 4.11).

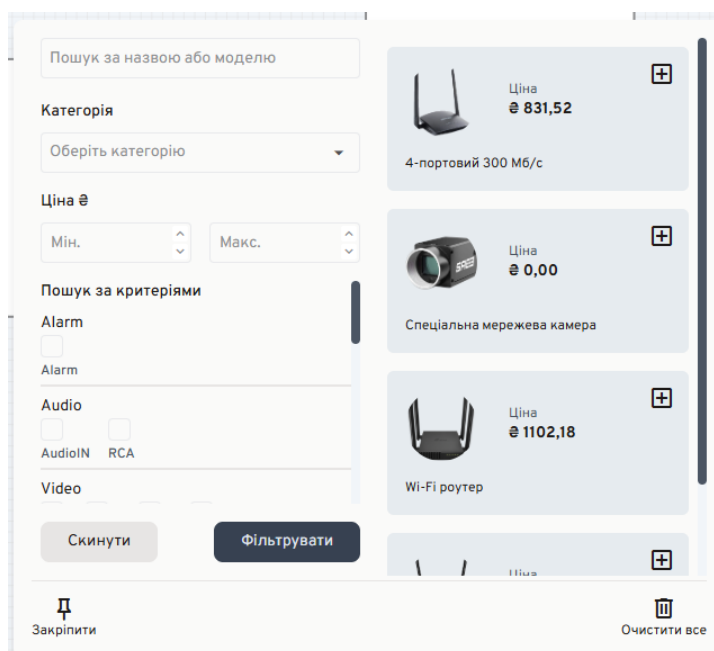


Рисунок 4.11 – Контекстне меню конфігуратора з базовими операціями

Закріплене контекстне меню зображено на рисунку 4.12.

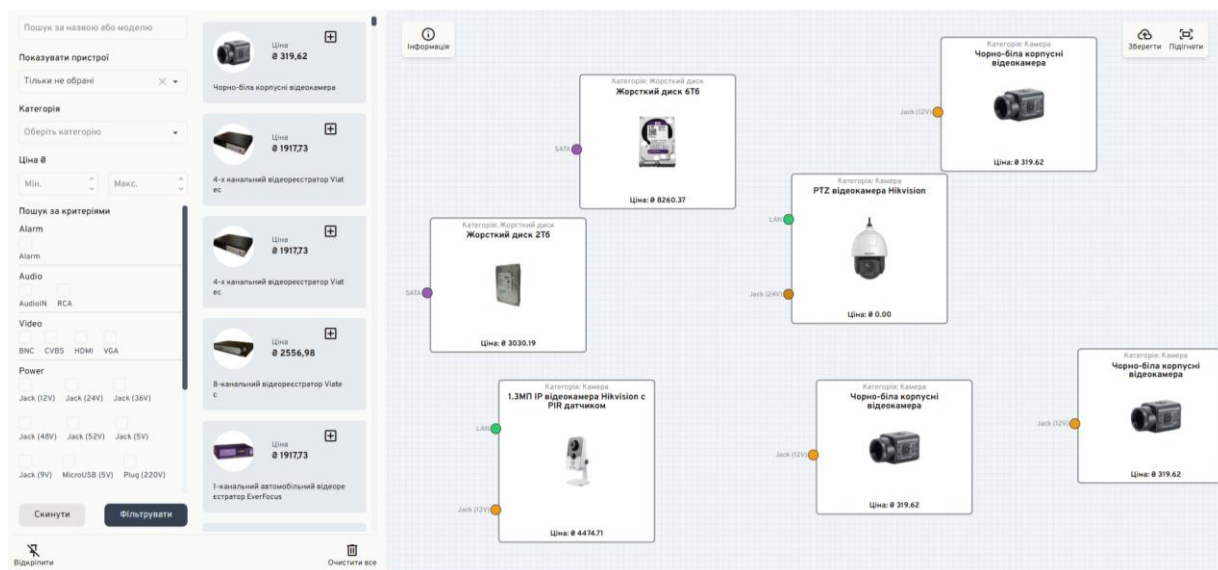


Рисунок 4.12 – Закріплене контекстне меню конфігуратору

Користувач може додавати окремі пристрої в список обраного, щоб потім фільтрувати пристрої лише за ним (рисунок 4.13).

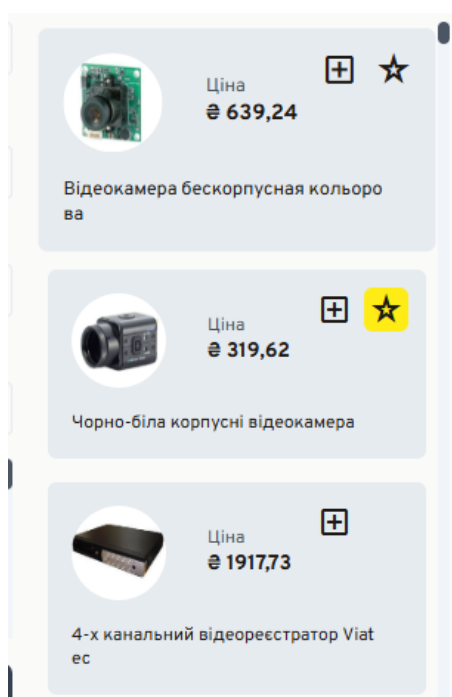


Рисунок 4.13 – Пристрій доданий до списку обраного

Для фільтрування пристроїв за списком обраного необхідно змінити фільтра показу пристроїв на значення «тільки обрані» (рисунок 4.14).

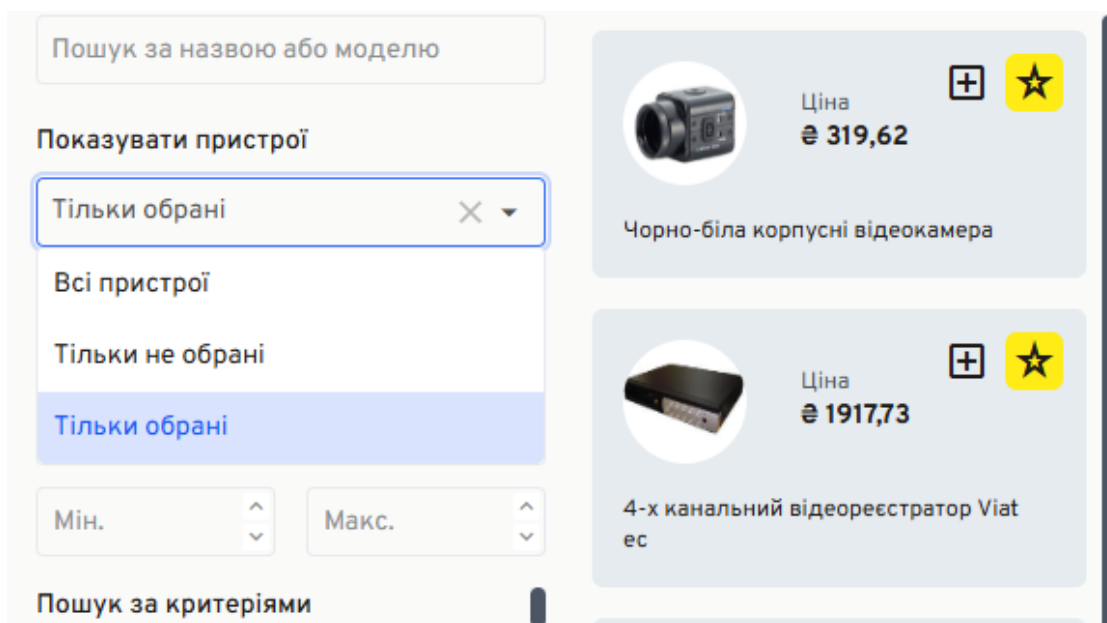


Рисунок 4.14 – Фільтрація пристроїв за списком обраного

Процес з'єднання пристроїв у конфігураторі відбувається шляхом зв'язування сумісних входів та виходів обладнання (рисунок 4.15).

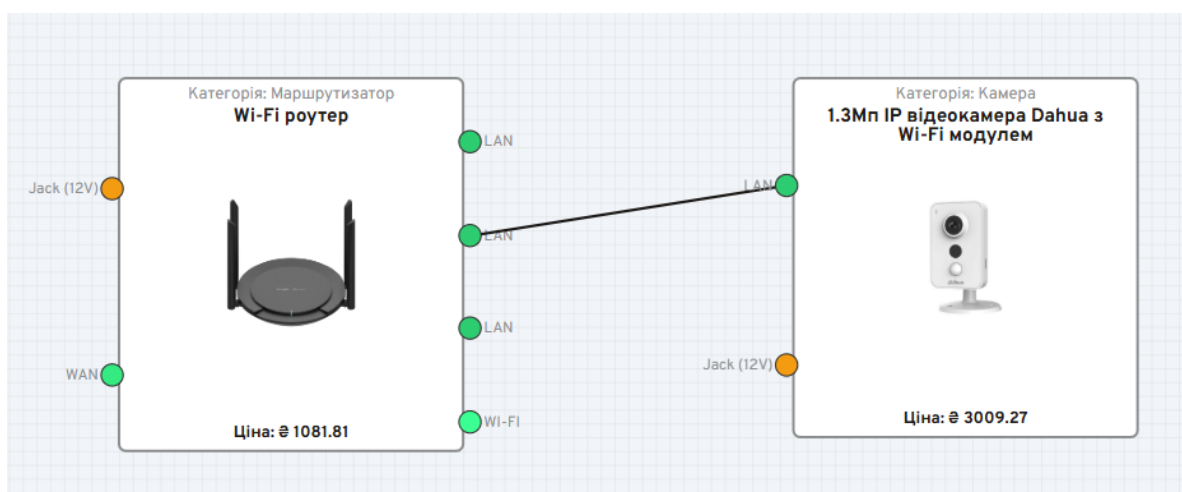


Рисунок 4.15 – Процес з'єднання елементів у конфігураторі

Для видалення пристрою зі сцени необхідно натиснути праву кнопку миші на потрібному елементі та обрати опцію «Видалити» (рисунок 4.16).

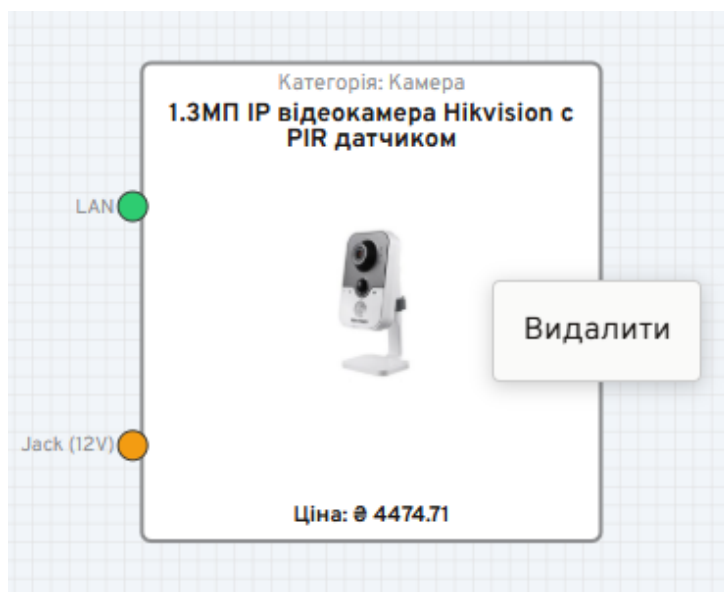


Рисунок 4.16 – Меню видалення пристрою

За необхідності, користувач може видаляти непотрібні з'єднання між пристроями (рисунок 4.17).

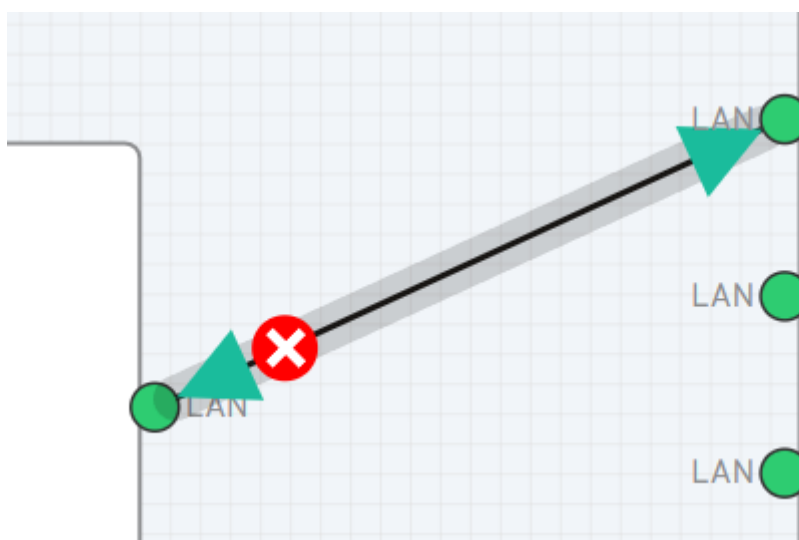


Рисунок 4.17 – Інтерфейс видалення з'єднання

Для швидкого переміщення декількох об'єктів на сцені варто використовувати виділення об'єктів, так, як зображено на рисунку 4.18.

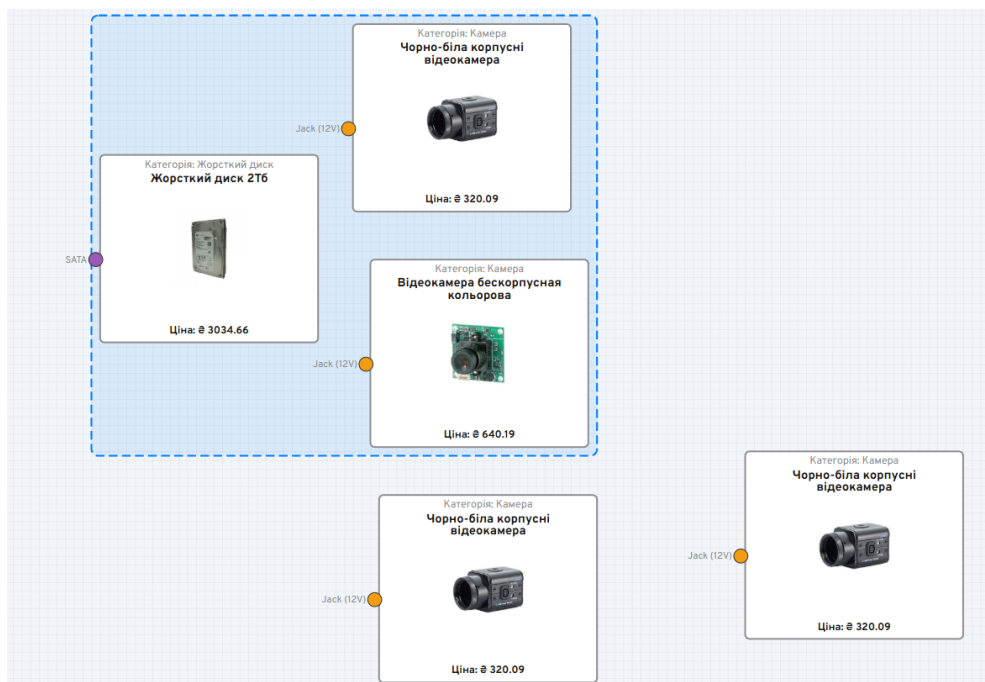


Рисунок 4.18 – Виділення декількох об'єктів на сцені

Після завершення проектування користувач може зберегти створену конфігурацію з можливістю подальшого редагування (рисунок 4.19).

Зберегти конфігурацію

Назва конфігурації *

Опис

Теги

Скасувати **Створити**

Рисунок 4.19 – Інтерфейс збереження конфігурації

Після збереження конфігурації, автор отримує доступ до нових функцій на сцені, такі як «Оновити» – оновлює схему поточної конфігурації в базі даних, що робить оновлення доступним для всіх, «Синхр.» – синхронізує конфігурацію з наявною в базі даних, прибираючи всі зміни, що були застосовані до конфігурації локально. Кнопка поділитись доступна для всіх і виконує функцію генерування унікального URL посилання для конфігурації (рисунок 4.20).

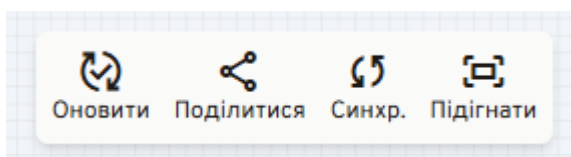


Рисунок 4.20 – Меню роботи зі збереженою конфігурацією

Форумний розділ платформи забезпечує комунікацію між користувачами з можливістю створення нових тем та додавання відповідей (рисунок 4.21).

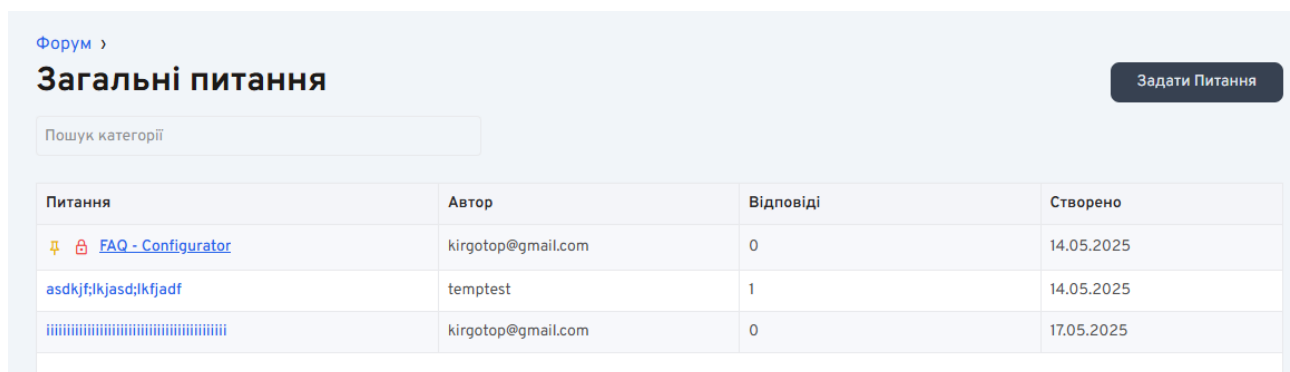
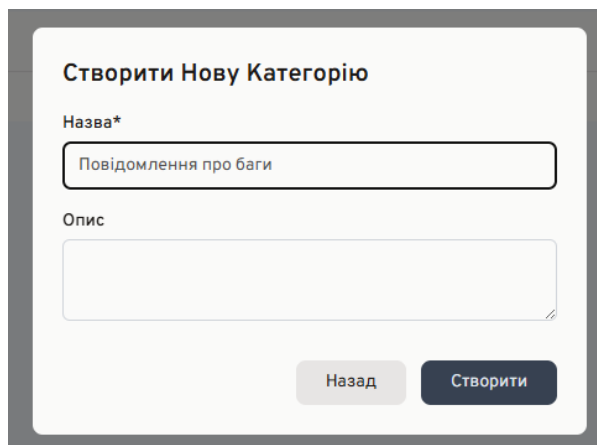


Рисунок 4.21 – Інтерфейс форуму з можливістю додавання нових публікацій та відповідей

Реалізований функціонал для звичайних користувачів забезпечує повний цикл роботи з системою конфігурування мережевого обладнання.

4.2 Сценарії роботи модератору сервісу

Інтерфейс модератора дозволяє створювати та організовувати категорії на форумі для структурування обговорень (рисунок 4.22).



The screenshot shows a form titled "Створити Нову Категорію" (Create New Category). It contains a text input field for "Назва*" (Name*) with the placeholder text "Повідомлення про баги" (Bug report). Below it is a larger text area for "Опис" (Description). At the bottom right, there are two buttons: "Назад" (Back) and "Створити" (Create).

Рисунок 4.22 – Інтерфейс створення категорій на форумі

Модератора також має можливість закривати та закріплювати запитання (рисунок 4.23).

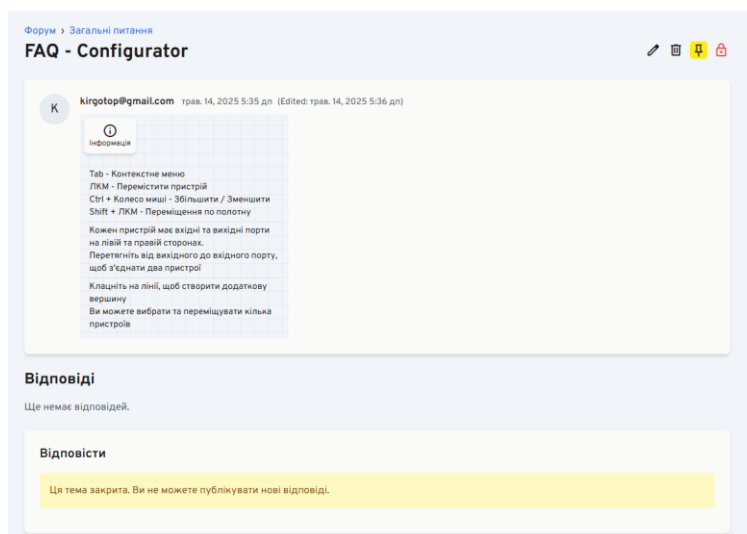


Рисунок 4.23 – Інтерфейс модерації тем форуму

Модератор може видаляти питання користувачів за необхідності (рисунок 4.24).

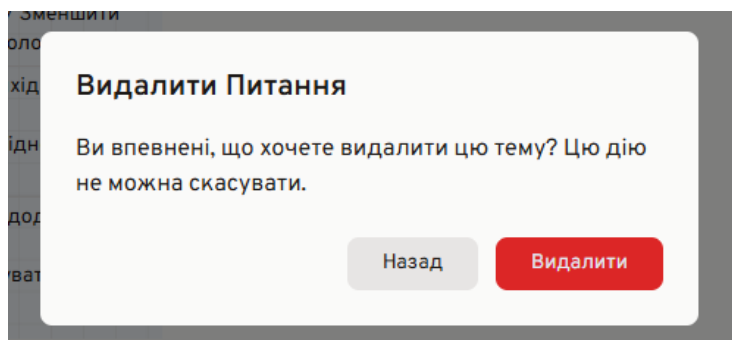


Рисунок 4.24 – Меню видалення питання на форумі

Інструментарій модератора забезпечує ефективне управління форумною частиною системи через організацію тематичних категорій та контроль за якістю контенту.

4.3 Сценарії роботи адміністратора сервісу

Адміністративна панель управління користувачами дозволяє змінювати рівні доступу та призначати ролі учасникам системи (рисунок 4.25).



Рисунок 4.25 – Окрема вкладка управління ролями користувачів

Система пошуку користувачів забезпечує швидкий доступ до облікових записів за різними параметрами фільтрації (рисунок 4.26).

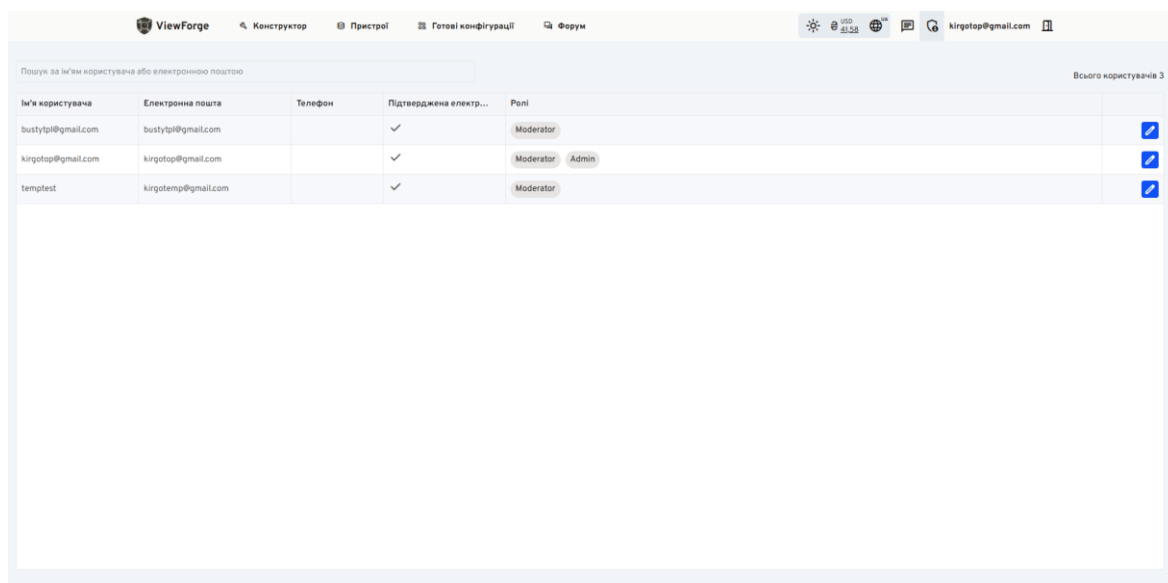


Рисунок 4.26 – Інтерфейс пошуку користувачів із фільтрацією

Адміністратор може редагувати ролі окремого користувача, встановивши відповідні прапорці у вікні редагування (рисунок 4.27).

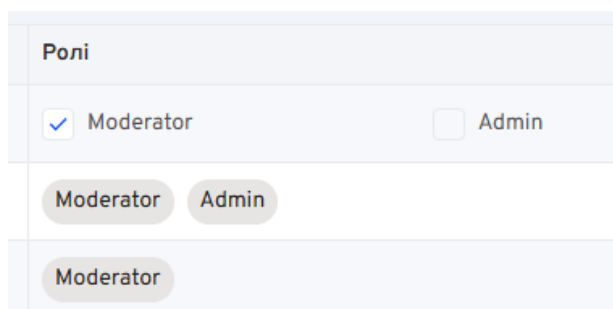


Рисунок 4.27 – Інтерфейс редагування ролей користувачів

Адміністративний рівень доступу розширює можливості керування системою через управління користувацькими ролями та пошук інформації облікових записів.

4.4 Розгортання проекту в середовищі Docker

Розгортання додатку на віддаленому сервері реалізовано за допомогою контейнеризації з використанням Docker та Docker Compose. Архітектура розгортання включає три основні компоненти: основний додаток ViewForge, базу даних MySQL та зворотний проксі-сервер Caddy [18] для обробки HTTP/HTTPS трафіку.

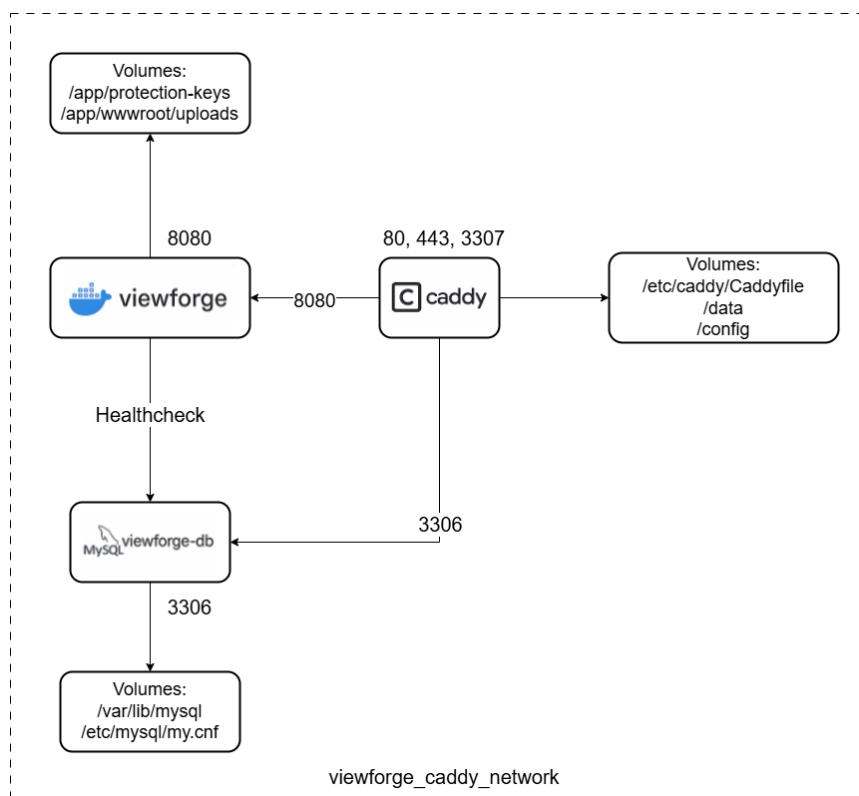
Файл збірки – Dockerfile додатку побудовано за принципом багатоетапної збірки, що забезпечує оптимізацію розміру фінального образу. Початковий етап `base` використовує офіційний образ Microsoft ASP.NET Core runtime версії 8.0 з налаштуванням порту 8080. Етап `build` відповідає за відновлення залежностей та компіляцію проекту, включаючи всі необхідні модулі `Infrastructure`, `Core`, `Domain` та клієнтську частину. Фінальний етап `final` копіює скомпільовану програму та налаштовує необхідні директорії з відповідними правами доступу для завантажених файлів та ключів захисту даних.

Основний додаток налаштовано на виконання в Production середовищі з використанням змінних оточення для конфігурації підключення до бази даних та поштової служби Mailjet. Том `viewforge_data_protection_keys` забезпечує збереження ключів шифрування ASP.NET Core Data Protection між перезапусками контейнера, а директорія `photo_uploads` монтується для зберігання завантажених користувачами файлів.

Зворотний проксі Caddy автоматично налаштовує HTTPS сертифікати та перенаправляє трафік на додаток, забезпечуючи безпечне з'єднання.

Файл конфігурації `.env` містить критично важливі змінні оточення: `MAILJET_API_KEY`, `MAILJET_API_SECRET`, `MAILJET_SENDER_EMAIL`, `MAILJET_SENDER_NAME` для функціональності відправки електронної пошти, параметри бази даних MySQL: `MYSQL_ROOT_PASSWORD`, `MYSQL_USER`, `MYSQL_PASSWORD` та рядок підключення `DB_CONNECTION_STRING`, який визначає параметри доступу додатку до бази даних.

Конфігурація Docker Compose об'єднує всі сервіси в єдину мережу `viewforge_caddy_network` та забезпечує правильну послідовність запуску (рисуюнок 4.28).



Рисуюнок 4.28 – Схема docker-compose для збірки проекту

Для забезпечення правильного порядку запуску контейнерів налаштовано `healthcheck` для контейнеру бази даних, що запускається при ініціалізації контейнеру і перевіряє його стан, контейнер ViewForge запускається лише після того, як контейнер бази даних пройшов перевірку.

Використання зовнішніх томів для всіх критичних даних забезпечує збереження інформації навіть при оновленні або перезапуску контейнерів, що гарантує стабільність роботи системи.

ВИСНОВКИ

У результаті виконання дипломної роботи досягнуто поставленої мети – розроблено веб-застосунок для конфігурування мережевого обладнання систем відеоспостереження з використанням сучасних технологій веб-розробки та візуального представлення схем з'єднання пристроїв.

1. На основі аналізу підходів, методів та алгоритмів розв'язання поставленої задачі використано графове представлення мережевих схем на базі моделі "граф" для моделювання зв'язків між пристроями, алгоритмів перевірки сумісності портів з підтримкою понад 20 типів інтерфейсів (LAN, Jack 12V, WI-FI, HDMI), методу візуального проектування з інтерактивним інтерфейсом на базі векторної SVG-графіки, системи подій для реалізації інтерактивності конфігуратора, підходу Code-First для версіонування структури бази даних через систему міграцій Entity Framework, role-based access control (RBAC) та claims-based security для гнучкого управління правами доступу користувачів.

2. На основі аналізу програмних засобів використано платформу .NET 8.0 з фреймворком Blazor Web App у режимі InteractiveServer для забезпечення високої продуктивності серверної частини з підтримкою SignalR-з'єднань, бібліотеки JointJS для створення інтерактивних діаграм на основі SVG, системи керування базами даних MySQL та ORM Entity Framework Core, компонентної бібліотеки Radzen Blazor Components з підтримкою віртуалізації та пагінації, CSS-фреймворку Tailwind з підходом PostCSS для оптимізації стилів, service Mailjet.Api для надійної доставки транзакційних email-повідомлень через RESTful API, системи ASP.NET Identity для комплексного управління користувачами, бібліотеки Microsoft.Extensions.Localization для забезпечення багатомовності інтерфейсу.

3. Розроблено програмну систему "Онлайн конфігуратор мережевого обладнання", яка забезпечує візуальне проектування схем систем відеоспостереження з автоматичною перевіркою сумісності компонентів через JavaScript-модуль на клієнті, містить каталог обладнання з детальними технічними характеристиками та повнотекстовим пошуком, реалізує систему збереження

конфігурацій у JSON-форматі з можливістю подальшого редагування, включає систему форумів з модерацією контенту та ролевим доступом, підтримує українську та англійську локалізації з параметризованими текстовими значеннями, забезпечує адаптивний інтерфейс з використанням Tailwind CSS та готових компонентів Radzen.

4. На основі тестування доведено коректність роботи розробленого програмного забезпечення, функціональне тестування підтвердило коректність роботи всіх режимів рендерингу (InteractiveServer для основних сторінок, SSR для форм авторизації) та відповідність реалізованого функціоналу заявленим вимогам.

Розроблене програмне забезпечення може бути використане інтеграторами систем безпеки, проектними організаціями та кінцевими користувачами для ефективного проектування систем відеоспостереження. Перспективи подальшого розвитку включають розширення бази підтримуваного обладнання, додавання функціоналу автоматичного розрахунку пропускної здатності мережі, інтеграцію з системами автоматизованого проектування та впровадження алгоритмів машинного навчання для оптимізації конфігурацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горбань М. Що таке .NET?. *Asabix*. URL: <https://asabix.com.ua/what-is-dot-net/> (дата звернення: 26.05.2025).
2. Roth D. .NET & Blazor. Створення веб-програми на основі браузера. *ITVDN*. URL: <https://itvdn.com/ua/blog/article/net-blazor> (дата звернення: 26.05.2025).
3. Hilton J. Getting started with blazor's new render modes in .NET 8. *Telerik Blogs*. URL: <https://www.telerik.com/blogs/getting-started-blazor-new-render-modes-net-8> (дата звернення: 26.05.2025).
4. Bernasconi C. Blazor basics: blazor javascript interop—calling javascript from .NET. *Telerik Blogs*. URL: <https://www.telerik.com/blogs/blazor-basics-blazor-javascript-interop-calling-javascript-net> (дата звернення: 26.05.2025).
5. Charbeneau E. Microsoft Blazor: .NET-розробники отримали доступ до екосистеми веб-браузерів. *Release*. URL: <https://release.nl/2364555/microsoft-blazor-net-ontwikkelaars-krijgen-toegang-tot-het-ecosysteem-van-webbrowsers.html> (дата звернення: 27.05.2025).
6. Оволодіння entity framework core: глибоке занурення - javascript.org.ua - JS communities. *javascript.org.ua - JS Communities*. URL: <https://javascript.org.ua/ovolodinnya-entity-framework-core-gliboke-zanurennya/> (дата звернення: 26.05.2025).
7. Що таке rest api: основні принципи та практики застосування. *FoxmindEd*. URL: <https://foxminded.ua/shcho-take-rest-api/> (дата звернення: 26.05.2025).
8. Chiarelli A. When ASP.NET core identity is no longer enough. *Auth0 - Blog*. URL: <https://auth0.com/blog/when-aspnet-core-identity-is-no-longer-enough/> (дата звернення: 26.05.2025).
9. Amr elshaer. Implementing role-based access control (RBAC) with claims transformation in .NET core. *Medium*. URL: <https://amrelsher07.medium.com/implementing-role-based-access-control-rbac-with-claims-transformation-in-net-core-0bacd39f9a78> (дата звернення: 26.05.2025).

10. Використання Tailwind CSS для швидкого розробки стильового дизайну - IT рейтинг UA. *IT рейтинг України*. URL: <https://it-rating.ua/vikoristannya-tailwind-css-dlya-shvidkogo-rozrobki-stilovogo-dizaynu> (дата звернення: 26.05.2025).

11. Krukowski I. ASP.NET Core localization and translation with examples. *Lokalise Blog*. URL: <https://lokalise.com/blog/asp-net-core-localization/> (дата звернення: 26.05.2025).

12. Maximova I. How we used JointJS in Voximplant Kit. *DEV Community*. URL: <https://dev.to/imaximova/how-we-used-jointjs-in-voximplant-kit-29b0> (дата звернення: 26.05.2025).

13. Chu A. Build Real-time Applications with ASP.NET Core SignalR. *CODE*. URL: <https://www.codemag.com/Article/1807061/Build-Real-time-Applications-with-ASP.NET-Core-SignalR> (дата звернення: 26.05.2025).

14. Easycodingschool. Javascript modules and ES6 modules. *Medium*. URL: <https://medium.com/@easycodingschool13/javascript-modules-and-es6-modules-78971f5f5805> (дата звернення: 26.05.2025).

15. Custom cookie authentication in blazor SSR + interactive server for .NET 8. *Read Medium articles with AI*. URL: <https://readmedium.com/custom-cookie-authentication-in-blazor-ssr-interactive-server-for-net-8-32fedd33267b> (date of access: 26.05.2025).

16. Meno V. P. The power of color in design – part 2: color schemes. *Medium*. URL: <https://medium.com/design-bootcamp/the-power-of-color-in-design-part-2-color-schemes-af9a9c36a70c> (дата звернення: 26.05.2025).

17. Mahachi J. T. Entity framework code first: with migrations. *Medium*. URL: <https://medium.com/@josiahmahachi/entity-framework-code-first-with-migrations-8d1a197d2bd4> (дата звернення: 26.05.2025).

18. jonsch.dev. Simplifying .NET deployment with caddy: the modern, secure web server. *Medium*. URL: <https://medium.com/@jonschdev/simplifying-net-deployment-with-caddy-the-modern-secure-web-server-9c849b8cf993>(дата звернення: 26.05.2025).

ДОДАТОК А

Онлайн конфігуратор мережевого обладнання для систем відеоспостереження

Текст Javascript файлу configurator.js

НТУУ «КПІ ім. Ігоря Сікорського»

Аркушів 8

Програмні засоби:

- мова програмування – JavaScript;
- бібліотека – JointJs.

```

class EnhancedConfigurator {
  constructor(canvasId) {
    this.canvasId = canvasId;
    this.devices = {};
    this.connections = [];

    this.preferredCulture = localStorage.getItem("PreferredCulture") ?? "";
    var preferredCurrency = localStorage.getItem("PreferredCurrency") ?? "";
    var currencyMult = localStorage.getItem("CurrencyMultiplier") ?? 42;
    this.currencyMultiplier = preferredCurrency === "UAH" ? currencyMult : 1;
    this.currencySymbol = preferredCurrency === "UAH" ? "₴" : "$";

    this.isSelecting = false;
    this.selectionStart = { x: 0, y: 0 };
    this.selectedElements = [];
    this.selectionRect = null;
    this.selectionAreaId = "SELECT_AREA";
    this.temporarySelectionRect = null;

    this.SHARED_CONFIG_STORAGE_PREFIX = 'sharedConfig_';
    this.DEFAULT_CONFIG_ID = 'default';
    this.currentConfigId = this.DEFAULT_CONFIG_ID;

    if (window.joint && window.$ && window._) {
      this.initializeColors();
      this.initialize();
    } else {
      console.error('Required libraries not loaded. Please include JointJS and its dependencies in your HTML.');
```

```

this.maxViewportHeight = 3240;

const canvasWidth = this.maxViewportWidth;
const canvasHeight = this.maxViewportHeight;

this.graph = new joint.dia.Graph();
this.paper = new joint.dia.Paper({
  el: canvasElement,
  model: this.graph,
  width: canvasWidth,
  height: canvasHeight,
  gridSize: 10,
  drawGrid: { name: 'mesh', args: { color: this.gridLinesColor } },
  background: {
    color: this.gridBgColor
  },
  interactive: {
    elementMove: true,
    arrowheadMove: true
  },
  defaultLink: new joint.dia.Link({
    attrs: {
      '.connection': {
        stroke: this.textColor,
        'stroke-width': 2,
        'pointer-events': 'none'
      }
    },
    connector: { name: 'rounded' }
  }),
  defaultConnectionPoint: { name: 'boundary' }
});

scrollContainer.scrollLeft = (canvasWidth - viewportWidth) / 2;
scrollContainer.scrollTop = (canvasHeight - viewportHeight) / 2;

this.scrollContainer = scrollContainer;

this.paper.options.validateConnection = (sourceView, sourceMagnet, targetView, targetMagnet) => {
  if (!sourceMagnet || !targetMagnet) return false;

  const sourcePortType = sourceMagnet.getAttribute('port-type');
  const targetPortType = targetMagnet.getAttribute('port-type');

  const sourceIsOutput = sourceMagnet.getAttribute('port-group') === 'output';
  const targetIsInput = targetMagnet.getAttribute('port-group') === 'input';

  const targetPort = targetView.model.getPort(targetMagnet.getAttribute('port'));

  const targetConnectedLinks = this.graph.getConnectedLinks(targetView.model, { inbound: true });
  const targetPortAlreadyConnected = targetConnectedLinks.some(link => {
    const target = link.get('target');
    return target && target.port === targetPort.id;
  });

  return sourceIsOutput &&
    targetIsInput &&
    sourcePortType === targetPortType &&
    !targetPortAlreadyConnected;
};

this.portColors = this.generatePortColors();

```

```

this.setupEvents();
this.setupPanZoom();
this.setupResizeHandler();
this.setupSelection();

this.currentConfigId = this.DEFAULT_CONFIG_ID;
}

setupEvents() {

let isOperating = false;
let saveTimeout = null;

const triggerAutosave = () => {

  if (saveTimeout) {
    clearTimeout(saveTimeout);
  }

  saveTimeout = setTimeout(() => {
    this.saveToLocalStorage();
  }, 500);
};

this.paper.on('link:connect', (linkView) => {
  const link = linkView.model;
  const sourceElement = this.graph.getCell(link.source().id);
  const targetElement = this.graph.getCell(link.target().id);

  if (!sourceElement || !targetElement) return;

  const sourcePort = link.source().port;
  const targetPort = link.target().port;

  const sourcePortData = sourceElement.getPort(sourcePort);
  const targetPortData = targetElement.getPort(targetPort);

  if (!sourcePortData || !targetPortData) return;

  const existingLinks = this.graph.getConnectedLinks(sourceElement, { outbound: true }).filter(existingLink => {
    const source = existingLink.get('source');
    return source && source.port === sourcePort && existingLink.id !== link.id;
  });

  existingLinks.forEach(existingLink => {

    this.connections = this.connections.filter(conn => conn.linkId !== existingLink.id);

    existingLink.remove({ disconnectLinks: true });
  });

  link.attr({
    line: {
      stroke: this.textColor,

```

```

        strokeWidth: 2,
        targetMarker: {
          type: 'circle',
          fill: this.textColor,
          r: 4
        }
      }
    });

    this.connections.push({
      sourceDevice: sourceElement.id,
      sourcePort: sourcePort,
      sourcePortType: sourcePortData.type,
      targetDevice: targetElement.id,
      targetPort: targetPort,
      targetPortType: targetPortData.type,
      linkId: link.id
    });

    triggerAutosave();
  });

  this.paper.on('link:disconnect', (linkView) => {
    const link = linkView.model;

    const source = link.get('source');
    const target = link.get('target');

    const isSourceDisconnected = !source.port;
    const isTargetDisconnected = !target.port;

    if (isSourceDisconnected && isTargetDisconnected) {

      this.connections = this.connections.filter(conn => conn.linkId !== link.id);

      link.remove();

      triggerAutosave();
    }
  });

  this.paper.on('link:remove', (linkView) => {
    const link = linkView.model;
    const linkId = link.id;

    this.connections = this.connections.filter(conn => conn.linkId !== linkId);

    triggerAutosave();
  });

  this.paper.on('cell:pointerdown', (cellView) => {
    const isElement = cellView.model.isElement();

```

```

    if (isElement) {
      cellView.model.toFront();
      isOperating = true;
    }
  });

  this.paper.on('element:pointerdown', () => {
    isOperating = true;
  });

  this.paper.on('cell:pointerup', () => {
    if (isOperating) {
      isOperating = false;
      triggerAutosave();
    }
  });

  this.graph.on('add', () => {
    triggerAutosave();
  });

  this.graph.on('remove', () => {
    triggerAutosave();
  });

  this.graph.on('change:position', () => {
    if (!isOperating) {
      triggerAutosave();
    }
  });

  this.paper.on('element:contextmenu', (elementView, evt) => {
    evt.preventDefault();

    const device = elementView.model;
    const deviceId = device.id;

    this.showContextMenu(evt.clientX, evt.clientY, [
      {
        text: this.localizer('delete'),
        action: () => {

          const connectedLinks = this.graph.getConnectedLinks(device);
          connectedLinks.forEach(link => link.remove());

          device.remove();

          if (this.devices[deviceId]) {
            delete this.devices[deviceId];
          }

          triggerAutosave();
        }
      }
    ]);
  });

```

```

    }
  }
  });
});

document.addEventListener('click', () => {
  this.hideContextMenu();
});
}

addDevice(ports, additionalInfo = {}) {

  const deviceName = additionalInfo.name ;
  const deviceCategory = additionalInfo.category || 'Uncategorized';
  const devicePrice = additionalInfo.price;
  const deviceImageUrl = additionalInfo.imageUrl || "";

  const inputPorts = ports.inputs || [];
  const outputPorts = ports.outputs || [];
  const totalInputs = inputPorts.length;
  const totalOutputs = outputPorts.length;

  let nameRows = deviceName.length / 28;
  let portHeight = 25;
  let labelHeight = 20 * Math.round(nameRows);
  let headerHeight = 80;
  let footerHeight = 20;
  let imageHeight = deviceImageUrl ? 70 : 0;
  let deviceHeight = Math.max(150, labelHeight + headerHeight + imageHeight + Math.max(totalInputs, totalOutputs) *
portHeight + footerHeight);

  const markup = `
<g class="rotatable">
  <g class="scalable">
    <rect class="body"/>
  </g>
  <text class="label"/>
  <text class="category"/>
  <text class="price"/>
  <image class="device-image"/>
  ${deviceImageUrl ? '<image class="device-image"/>' : ""}
</g>
`;

  const device = new joint.shapes.devs.Model({
    position: { x: 100, y: 100 },
    size: { width: 250, height: deviceHeight },
    markup: markup,
    attrs: {
      '.body': {
        fill: this.blockBg,
        stroke: this.textSecondaryColor,
        'stroke-width': 2,
        rx: 5,
        ry: 5,
        width: '100%',
        height: '100%'
      },
    },
  },

```

```

'.label': {
  text: deviceName,
  'ref-x': .5,
  'ref-y': 20,
  'font-size': 14,
  'font-weight': 'bold',
  'text-anchor': 'middle',
  fill: this.textColor,
  'textWrap': {
    width: 'calc(w - 20)',
    height: -1,
    ellipsis: true
  },
  'ref-width': .9
},
'.category': {
  text: `${this.localizer('Category')}: ${this.localizer(deviceCategory)}`,
  'ref-x': .5,
  'ref-y': 5,
  'font-size': 12,
  'text-anchor': 'middle',
  fill: this.textSecondaryColor
},
'.price': {
  text: `${this.localizer('price')}: ${this.currencySymbol} ${this.formatPrice(devicePrice)}`,
  'ref-x': .5,
  'ref-y': deviceHeight - 20,
  'font-size': 12,
  'font-weight': 'bold',
  'text-anchor': 'middle',
  fill: this.textColor
}
},
ports: {
  groups: {
    'input': {
      position: {
        name: 'left',
        args: {
          dx: -5,
          dy: portHeight / 2
        }
      }
    },
    attrs: {
      '.port-body': {
        r: 8,
        magnet: 'passive',
        stroke: '#333',
        fill: '#fff'
      },
      text: {
        'font-size': 11,
        'font-weight': 'normal',
        'font-family': 'Arial, Helvetica, sans-serif',
        fill: this.textSecondaryColor,
        'text-anchor': 'end'
      }
    }
  },
  label: {
    position: {
      name: 'left',
      args: { x: -10 }
    }
  }
}

```

```

    }
  },
  'output': {
    position: {
      name: 'right',
      args: {
        dx: 5,
        dy: portHeight / 2
      }
    },
    attrs: {
      '.port-body': {
        r: 8,
        magnet: true,
        stroke: '#333',
        fill: '#fff'
      },
      text: {
        'font-size': 11,
        'font-weight': 'normal',
        'font-family': 'Arial, Helvetica, sans-serif',
        fill: this.textSecondaryColor,
        'text-anchor': 'start'
      }
    },
    label: {
      position: {
        name: 'right',
        args: { x: 10 }
      }
    }
  }
}
});

if (deviceImageUrl) {
  device.attr({
    '.device-image': {
      'xlink:href': deviceImageUrl,
      'ref-x': .5,
      'ref-y': .5,
      width: 100,
      height: 100,
      'x-alignment': 'middle',
      'y-alignment': 'middle',
    }
  });

  headerHeight = 140;
}

inputPorts.forEach((portType, index) => {
  device.addPort({
    group: 'input',
    id: `in-${portType}-${index}`,
    type: portType,
    attrs: {
      '.port-body': {
        fill: this.portColors[portType] || '#333',
        'port-type': portType
      },
    },
  });
});

```

```

        text: {
            text: portType
        }
    },
    position: {
        args: {
            y: headerHeight + index * portHeight
        }
    }
});
});

outputPorts.forEach((portType, index) => {
    device.addPort({
        group: 'output',
        id: `out-${portType}-${index}`,
        type: portType,
        attrs: {
            'port-body': {
                fill: this.portColors[portType] || '#333',
                'port-type': portType
            },
            text: {
                text: portType
            }
        },
        position: {
            args: {
                y: headerHeight + 100 + index * portHeight
            }
        }
    });
});

const currentScale = this.paper.scale().sx;

const viewportCenterX = this.scrollContainer.scrollLeft / currentScale + (this.viewportWidth / currentScale) / 2;
const viewportCenterY = this.scrollContainer.scrollTop / currentScale + (this.viewportHeight / currentScale) / 2;

device.position(
    viewportCenterX - (device.size().width / 2),
    viewportCenterY - (device.size().height / 2)
);

this.graph.addCell(device);

this.devices[device.id] = {
    id: device.id,
    name: deviceName,
    category: deviceCategory,
    imageUrl: deviceImageUrl,
    inputs: inputPorts,
    outputs: outputPorts,
    price: parseFloat(devicePrice)
};

return device.id;
}
}

```