

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМИ
ВІДЕОСПОСТЕРЕЖЕННЯ
SOFTWARE FOR A VIDEO SURVEILLANCE SYSTEM

Виконав(ла): студент(ка) 4 курсу, групи КНТ-131

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КУТИРСЬ С.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КУТИРЄВА Степана Вікторовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для системи відеоспостереження. Software for a Video Surveillance System.

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення розміщення камер відеоспостереження. 3. Опис програмного забезпечення. 4. Експлуатація та тестування програмного забезпечення.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Степан КУТИРЄВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Степан СКРУПСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 113 с., 17 табл., 44 рис., 3 дод., 24 джерела.

ВІДЕОСПОСТЕРЕЖЕННЯ, ОПТИМІЗАЦІЯ РОЗМІЩЕННЯ КАМЕР, СИСТЕМА БЕЗПЕКИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT.

Об'єкт - процес розроблення програмного забезпечення для оптимізації розміщення камер відеоспостереження в приміщеннях.

Предмет - методи та програмні засоби автоматизованого планування розміщення відеокамер на основі аналізу геометрії приміщень.

Мета роботи - підвищення ефективності проєктування систем відеоспостереження шляхом розробки програмного забезпечення для автоматичного визначення оптимального розташування камер.

Матеріали, методи та технічні засоби: документоорієнтована база даних MongoDB, мікросервісна архітектура, алгоритми оптимізації (жадібний алгоритм, метод імітації відпалу, градієнтний спуск), мови програмування JavaScript і TypeScript, фреймворки React і Node.js, комп'ютер з процесором Intel Core i7 під управлінням операційної системи Windows 10.

Результати. Створено програмний вебзастосунок для оптимізації розміщення камер відеоспостереження, який дозволяє автоматично визначати оптимальне розташування камер з урахуванням геометрії приміщення, наявних перешкод та технічних характеристик камер.

Висновки. Розроблено програмне забезпечення для системи відеоспостереження, яке забезпечує максимальне покриття приміщення при мінімальних витратах на обладнання.

Галузь використання - проєктування систем безпеки для приватних будинків, комерційних приміщень та промислових об'єктів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
113 pages, 17 tables, 44 figures, 3 appendixes, 24 sources.

VIDEO SURVEILLANCE, CAMERA PLACEMENT OPTIMIZATION,
SECURITY SYSTEM, SOFTWARE, REACT.

Object of study is the process of developing software for optimizing the placement of video surveillance cameras in premises.

Subject of study are methods and software tools for automated planning of camera placement based on the analysis of room geometry.

The purpose of the work is to improve the efficiency of video surveillance system design by developing software for automatic determination of optimal camera placement.

Materials, methods, and tools: document-oriented MongoDB database, microservice architecture, optimization algorithms (greedy algorithm, simulated annealing method, gradient descent), JavaScript and TypeScript programming languages, React and Node.js frameworks, computer with Intel Core i7 processor running Windows 10 operating system.

Results. A web application for optimizing the placement of video surveillance cameras has been created, which automatically determines the optimal camera placement taking into account room geometry, existing obstacles, and technical characteristics of cameras.

Conclusions. Software for a video surveillance system has been developed that provides maximum room coverage with minimal equipment costs. The system allows uploading room plans, configuring camera parameters, automatically optimizing their placement, and visualizing results.

Field of use is the design of security systems for private homes, commercial premises, and industrial facilities.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Аналіз сучасних систем відеоспостереження	10
1.2 Проблематика оптимізації розміщення камер відеоспостереження.....	15
1.3 Формулювання вимог до програмного забезпечення	22
1.4 Висновки до розділу 1	32
2 Проєктування програмного забезпечення розміщення камер відеоспостереження	34
2.1 Архітектура програмного забезпечення	34
2.2 Алгоритм оптимізації розміщення камер	40
2.3 Висновки за розділом 2	46
3 Опис програмного забезпечення	48
3.1 Розробка бази даних програмного забезпечення для оптимізації розміщення камер відеоспостереження	48
3.2 Проєктування інтерфейсу користувача	56
3.3 Реалізація алгоритму оптимізації розміщення камер.....	65
3.4 Висновки за розділом 3	67
4 Експлуатація та тестування програмного забезпечення	68
4.1 Призначення програмного забезпечення.....	68
4.2 Умови виконання програмного забезпечення.....	71
4.3 Виконання і тестування програмного забезпечення	77
4.4 Висновки за розділом 4	78
Висновки	80
Перелік джерел посилання	82
Додаток А Технічне завдання	84
Додаток Б Фрагмент тексту програми	88
Додаток В Слайди презентації.....	106

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CCTV	- Closed-Circuit Television;
DVR	- Digital Video Recorder;
FHD	- Full High Definition;
FPS	- Frames Per Second;
HD	- High Definition;
HTTPS	- Hypertext Transfer Protocol Secure;
IoT	- Internet of Things;
IP	- Internet Protocol;
JSON	- JavaScript Object Notation;
JWT	- JSON Web Tokens;
NVR	- Network Video Recorder;
ONVIF	- Open Network Video Interface Forum;
PTZ	- Pan-Tilt-Zoom;
REST	- Representational State Transfer;
RTSP	- Real Time Streaming Protocol;
SaaS	- Software as a Service;
SPA	- Single Page Application;
SSL	- Secure Sockets Layer;
TLS	- Transport Layer Security.

ВСТУП

У сучасному світі питання безпеки стають все більш актуальними, а системи відеоспостереження перетворились із простих пристроїв запису в інтелектуальні системи моніторингу з розширеними можливостями. Використання передових технологій програмування та штучного інтелекту відкриває нові горизонти для підвищення ефективності систем відеоспостереження та розширення їхнього функціоналу [1].

Зростання кількості користувачів систем безпеки та відеоспостереження супроводжується підвищенням вимог до функціональності, надійності та зручності таких систем. Сучасні програмні рішення для відеоспостереження не просто записують відео, але й здійснюють аналітику, розпізнають об'єкти та події, надсилають сповіщення, забезпечують віддалений доступ та інтеграцію з іншими системами розумного дому [2].

Розробка програмного забезпечення для системи відеоспостереження є комплексним завданням, що вимагає поєднання знань у галузі, мережових технологій, зберігання та обробки великих обсягів даних, а також створення зручного інтерфейсу користувача. Особливої актуальності набуває оптимізація розміщення камер відеоспостереження для максимального покриття приміщення при мінімальних витратах обладнання.

Сучасні користувачі систем відеоспостереження нерідко стикаються з проблемами неефективного розміщення камер, що призводить до появи "сліпих зон" або, навпаки, до надмірної кількості обладнання та перекриття зон спостереження. Тому автоматизація процесу проєктування системи відеоспостереження з оптимальним розміщенням камер є актуальним завданням, вирішення якого допоможе значно підвищити ефективність використання обладнання та зменшити витрати на його придбання та обслуговування [3].

Об'єктом дослідження є процес розроблення програмного забезпечення для оптимізації розміщення камер відеоспостереження в приміщеннях.

Предметом дослідження є методи та програмні засоби автоматизованого планування розміщення відеокамер на основі аналізу геометрії приміщень.

Метою роботи є підвищення ефективності проєктування систем відеоспостереження шляхом розробки програмного забезпечення для автоматичного визначення оптимального розташування камер.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- дослідити існуючі підходи до автоматизації проєктування систем відеоспостереження;
- розробити алгоритм оптимального розміщення камер з урахуванням геометрії приміщень та перешкод;
- створити вебзастосунок для взаємодії користувачів з розробленим алгоритмом;
- реалізувати інтерактивний інтерфейс для візуалізації результатів роботи алгоритму;
- провести тестування розробленого програмного забезпечення на різних типах приміщень.

Програмне забезпечення, що розробляється в рамках даної роботи, спрямоване на вирішення проблеми оптимального розміщення камер у приміщенні з урахуванням його конфігурації, наявності перешкод та характеристик наявного обладнання. На відміну від існуючих рішень, дане програмне забезпечення орієнтоване на використання непрофесійними користувачами, має інтуїтивно зрозумілий інтерфейс та не потребує спеціальних знань у галузі проєктування систем відеоспостереження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасних систем відеоспостереження

Сучасні системи відеоспостереження пройшли значний шлях розвитку від простих аналогових пристроїв до комплексних цифрових рішень з широкими можливостями аналітики та інтеграції [4]. Правильне розуміння еволюції та тенденцій розвитку таких систем є ключовим для створення ефективного програмного забезпечення в цій галузі.

Розвиток систем відеоспостереження можна умовно розділити на кілька етапів, кожен з яких характеризується впровадженням нових технологій та розширенням функціональних можливостей (табл. 1.1).

Таблиця 1.1 - Еволюція систем відеоспостереження

Період	Технологія	Характеристики	Можливості
1970-1990	Аналогові системи	Низька роздільна здатність, обмежений радіус дії	Базовий запис на плівку, відсутність віддаленого доступу
1990-2005	Цифрові DVR системи	Покращена якість зображення, локальне зберігання	Цифровий запис, базовий пошук за часом
2005-2015	IP-камери та NVR	Висока роздільна здатність, мережеве підключення	Віддалений доступ, базова аналітика
2015-2022	Хмарні системи з AI	HD/4K якість, інтеграція з IoT	Розширена аналітика, розпізнавання об'єктів
2022+	Системи з глибоким навчанням	Ultra HD, модульна архітектура	Предиктивна аналітика, автономне прийняття рішень

Важливо відзначити, що сучасний етап розвитку характеризується не лише покращенням технічних характеристик обладнання, але й впровадженням інтелектуальних алгоритмів обробки зображень [5]. Це створює нові вимоги до програмного забезпечення, яке має не лише забезпечувати базову функціональність запису та відтворення відео, але й надавати можливості для аналізу отриманих даних.

Сучасний ринок пропонує різноманітні типи камер відеоспостереження, які відрізняються за призначенням, технічними характеристиками та функціональними можливостями. Розуміння ключових відмінностей між ними є необхідним для розробки програмного забезпечення, яке враховуватиме особливості кожного типу обладнання (рис. 1.1).



Рисунок 1.1 – Класифікація сучасних камер відеоспостереження [4]

Кожен тип камер має специфічні характеристики, які впливають на алгоритми їх розміщення та налаштування. Аналіз технічних параметрів різних типів камер (табл. 1.2) дозволяє виявити ключові фактори, які необхідно враховувати при проєктуванні системи відеоспостереження.

Особливу увагу слід звернути на PTZ-камери (Pan-Tilt-Zoom), які можуть змінювати кут огляду та збільшення зображення [6].

Таблиця 1.2 – Порівняння технічних характеристик різних типів камер

Параметр	Купольні камери	Циліндричні камери	PTZ-камери	Панорамні камери
Кут огляду	90-120°	70-90°	360° (з поворотом)	180-360°
Максимальна дальність	15-30 м	30-100 м	100-300 м	15-20 м
Вплив освітлення	Середній	Високий	Високий	Низький
Захист від вандалізму	Високий	Середній	Низький	Низький
Вартість	Середня	Середня	Висока	Висока
Складність монтажу	Низька	Середня	Висока	Низька

Ці камери потребують специфічних алгоритмів розміщення, які враховуватимуть можливість керування їх положенням програмними засобами (рис. 1.2).



Рисунок 1.2 – Принцип роботи PTZ-камери та зони її покриття [6]

Аналіз існуючого програмного забезпечення для систем відеоспостереження дозволяє виділити кілька основних типів рішень, які домінують на ринку [7]:

- програмне забезпечення виробників обладнання, яке часто має обмежену сумісність з камерами інших виробників;
- універсальні платформи відеоменеджменту (VMS), які підтримують різноманітне обладнання;
- спеціалізовані рішення для конкретних галузей (банківський сектор, роздрібна торгівля);
- хмарні сервіси для віддаленого доступу та управління камерами;
- рішення з відкритим кодом, які можуть бути модифіковані під конкретні потреби.

Порівняльний аналіз популярних програмних рішень на ринку (табл. 1.3) демонструє їхні переваги та недоліки, а також допомагає виявити можливості для вдосконалення.

Таблиця 1.3 – Порівняння існуючих програмних рішень для систем відеоспостереження

Назва ПЗ	Підтримка обладнання	Аналітичні можливості	Оптимізація розміщення камер	Інтерфейс	Ціна
Milestone Xprotect	Широка	Високі	Відсутня	Складний	Висока
Avigilon Control Center	Обмежена	Високі	Обмежена	Середньої складності	Висока
Genetec Security Center	Широка	Високі	Відсутня	Складний	Висока
iSpy	Широка	Низькі	Відсутня	Простий	Безкоштовно
ZoneMinder	Середня	Середні	Відсутня	Складний	Безкоштовно

Аналіз існуючих рішень виявляє ключовий недолік більшості програмних продуктів на ринку – відсутність або обмежену функціональність оптимізації розміщення камер відеоспостереження [8]. Саме ця проблема є центральною для даної роботи, оскільки ефективність системи відеоспостереження значною мірою залежить від правильного розміщення камер.

Системи відеоспостереження використовуються в різноманітних сферах, кожна з яких має специфічні вимоги та обмеження (рис. 1.3).

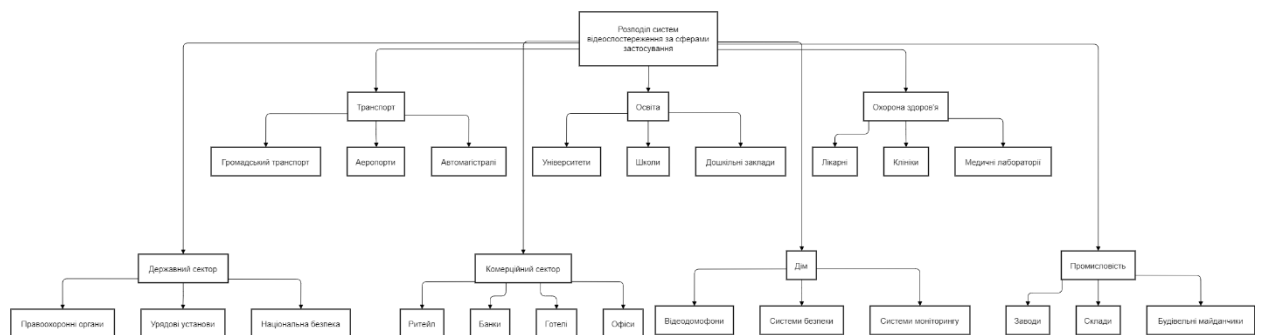


Рисунок 1.3 – Розподіл систем відеоспостереження за сферами застосування

Потреби різних категорій користувачів суттєво відрізняються як за масштабом системи, так і за функціональними вимогами (табл. 1.4).

Таблиця 1.4 – Потреби різних категорій користувачів систем відеоспостереження

Категорія користувачів	Масштаб системи	Ключові вимоги	Бюджетні обмеження
1	2	3	4
Приватні домогосподарства	2-10 камер	Простота використання, віддалений доступ	Значні
Малий бізнес	5-20 камер	Надійність, базова аналітика	Середні

Продовження таблиці 1.4

1	2	3	4
Великі підприємства	100+ камер	Інтеграція з іншими системами, висока надійність	Незначні
Об'єкти критичної інфраструктури	Різний	Максимальна надійність, специфічні функції	Незначні

Аналіз потреб різних категорій користувачів дозволяє зробити висновок, що найбільш незадоволеними залишаються потреби приватних домогосподарств та малого бізнесу, які мають обмежений бюджет, але потребують ефективних рішень [9]. Саме ці категорії користувачів є основною цільовою аудиторією для програмного забезпечення, що розробляється в рамках даної роботи.

Проведений аналіз сучасних систем відеоспостереження дозволяє сформулювати ключові вимоги до розроблюваного програмного забезпечення, яке має забезпечувати оптимальне розміщення камер з урахуванням їх технічних характеристик, особливостей приміщень та потреб користувачів різних категорій.

1.2 Проблематика оптимізації розміщення камер відеоспостереження

Оптимальне розміщення камер відеоспостереження є складною багатофакторною задачею, яка потребує врахування технічних обмежень обладнання, особливостей приміщень та специфічних вимог користувачів [10]. Дослідження цієї проблематики дозволяє виявити ключові фактори, які необхідно враховувати при розробці алгоритмів автоматичного розміщення камер.

Ефективність системи відеоспостереження значною мірою залежить від правильного врахування технічних характеристик використовуваних камер. Ключові параметри, які впливають на оптимальне розміщення обладнання (рис. 1.4).

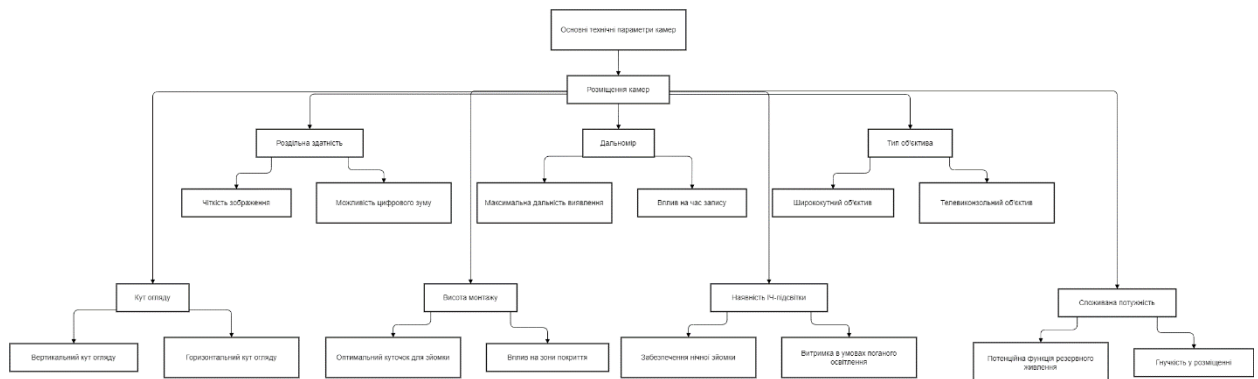


Рисунок 1.4 - Основні технічні параметри камер, що впливають на їх розміщення

Кут огляду є одним із найважливіших параметрів, який визначає зону покриття камери. Різні моделі камер мають різні кути огляду, які можуть варіюватися від 30° до 360° для панорамних камер [11]. При проєктуванні системи відеоспостереження необхідно враховувати, що камери з ширшим кутом огляду можуть покрити більшу площу, але з нижчою деталізацією зображення.

Глибина огляду (максимальна відстань, на якій камера може чітко розрізняти об'єкти) є ще одним критичним параметром. Цей показник залежить від роздільної здатності камери, чутливості матриці, якості оптики та інших факторів. Для різних сценаріїв використання необхідна різна глибина огляду (табл. 1.5).

Роздільна здатність камери визначає якість отриманого зображення та можливість розпізнавання деталей на ньому. Сучасні камери забезпечують роздільну здатність від базових 720p до 4K та навіть вище. Однак, важливо враховувати, що підвищення роздільної здатності збільшує вимоги до пропускної здатності мережі та обсягу сховища для запису відео [12].

Таблиця 1.5 - Рекомендовані значення глибини огляду для різних сценаріїв

Сценарій використання	Мінімальна глибина огляду	Рекомендована роздільна здатність
Ідентифікація особи	3-5 м	1080р або вище
Розпізнавання дій	10-15 м	720р або вище
Виявлення руху	20-50 м	480р або вище
Загальний огляд території	50-100 м	1080р або вище для PTZ-камер

Практика проєктування та експлуатації систем відеоспостереження виявляє ряд типових проблем, які можуть значно знизити ефективність системи (рис. 1.5).

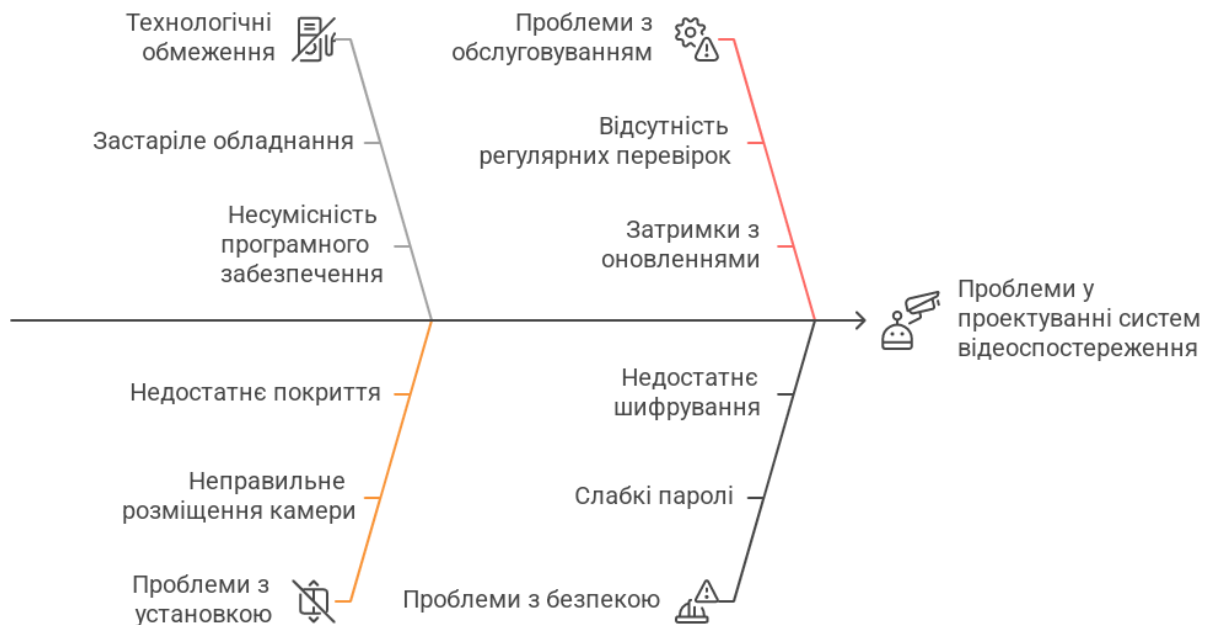


Рисунок 1.5 - Типові проблеми при проєктуванні систем відеоспостереження

Наявність «сліпих зон» є однією з найпоширеніших проблем, яка виникає через неправильне розміщення камер або неврахування перешкод у приміщенні. Дослідження показують, що в середньому до 30% площі

приміщень залишаються непокритими системами відеоспостереження через неоптимальне розміщення камер [13].

Надмірне перекриття зон огляду призводить до неефективного використання ресурсів і збільшення вартості системи. За оцінками експертів, до 25% камер у типових інсталяціях створюють надмірне перекриття зон огляду, що не дає значущого підвищення безпеки, але збільшує витрати на обладнання та обслуговування [14].

Неврахування особливостей освітлення може призвести до суттєвого зниження якості отриманого зображення. Камери, розміщені напроти вікон або яскравих джерел світла, можуть давати засвічене зображення, а недостатнє освітлення в зоні спостереження може зробити запис непридатним для ідентифікації об'єктів або осіб.

Аналіз типових проблем при проєктуванні систем відеоспостереження (табл. 1.6) дозволяє розробити алгоритми, які запобігатимуть цим проблемам.

Для вирішення задачі оптимального розміщення камер відеоспостереження можуть бути застосовані різні математичні методи та алгоритми. Аналіз наукової літератури дозволяє виділити кілька основних підходів до розв'язання цієї задачі [15]:

- генетичні алгоритми, які дозволяють знаходити приблизні рішення оптимізаційних задач через імітацію процесу природного відбору;
- метод динамічного програмування, який ефективний для задач з чітко визначеною структурою та обмеженнями;
- цілочисельне лінійне програмування, яке дозволяє знаходити точні рішення для задач з дискретними змінними;
- алгоритми комбінаторної оптимізації, які працюють у просторі дискретних рішень;
- евристичні методи, які дають швидкі, але не обов'язково оптимальні рішення.

Таблиця 1.6 - Типові проблеми при проектуванні систем відеоспостереження та їх наслідки

Проблема	Причини	Наслідки	Можливі рішення
Сліпі зони	Неврахування геометрії приміщення, перешкод	Неповне покриття простору	Алгоритмічна оптимізація розміщення
Надмірне перекриття	Відсутність координатії при розміщенні	Надлишкові витрати	Автоматичний розрахунок оптимальної кількості камер
Проблеми з освітленням	Неврахування джерел світла	Низька якість зображення	Аналіз освітлення при розміщенні
Недостатня роздільна здатність для цільових завдань	Неправильний вибір обладнання	Неможливість ідентифікації	Підбір камер під конкретні завдання
Обмеження пропускну здатності мережі	Неврахування обсягів даних	Втрата кадрів, затримки	Розрахунок мережевого навантаження

Кожен із цих методів має свої переваги та обмеження (табл. 1.7), які необхідно враховувати при розробці алгоритмів оптимізації розміщення камер.

Для задачі оптимізації розміщення камер відеоспостереження особливо ефективними є гібридні підходи, які поєднують різні методи для досягнення балансу між оптимальністю рішення та швидкістю обчислень [16].

Таблиця 1.7 - Порівняння методів оптимізації розміщення камер

Метод	Переваги	Обмеження	Складність реалізації	Швидкість роботи
Генетичні алгоритми	Здатність знаходити рішення для складних задач	Не гарантує знаходження глобального оптимуму	Середня	Середня
Динамічне програмування	Гарантоване знаходження оптимального рішення	Складно застосувати для задач з багатьма обмеженнями	Висока	Повільна
Цілочисельне лінійне програмування	Точні рішення для формалізованих задач	Обмеження на лінійність цільової функції	Висока	Повільна
Комбінаторна оптимізація	Ефективність для дискретних задач	Складність для задач великої розмірності	Середня	Середня
Евристичні методи	Швидкість роботи	Не гарантує оптимальності	Низька	Швидка

Однією з перспективних моделей є представлення приміщення у вигляді графа видимості (рис. 1.6), де вершини відповідають потенційним точкам розміщення камер, а ребра – зонам видимості.

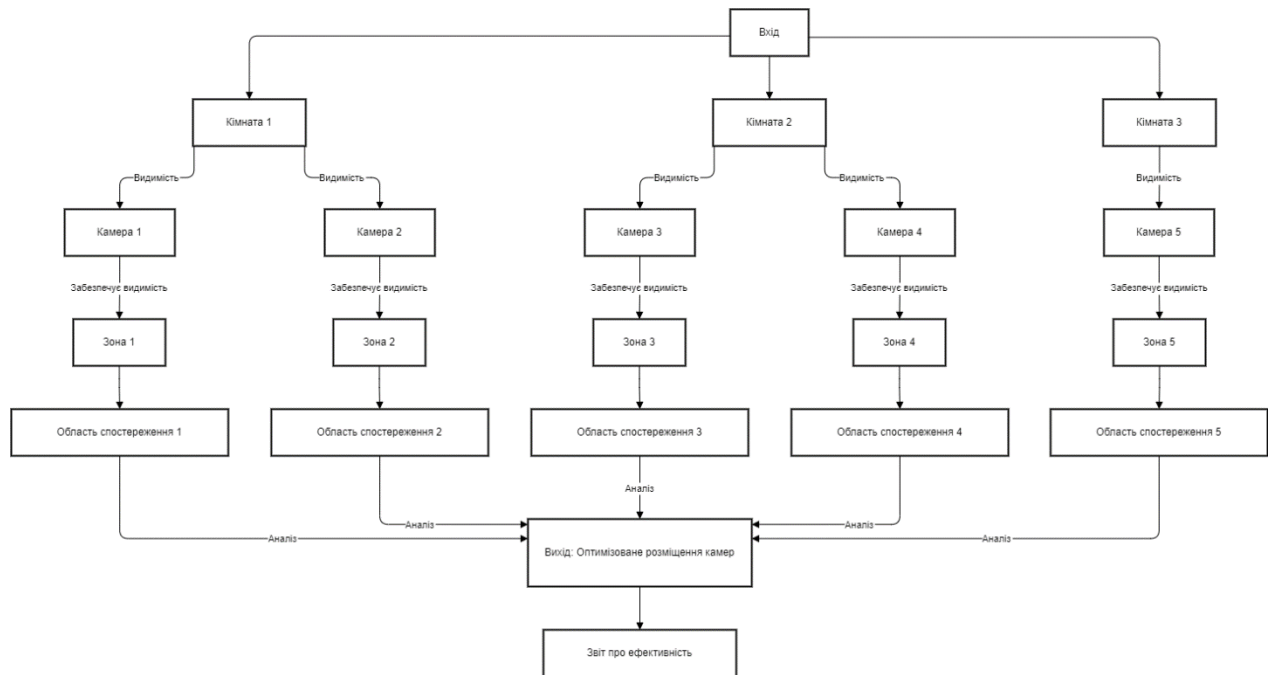


Рисунок 1.6 - Приклад представлення приміщення у вигляді графа видимості для оптимізації розміщення камер

Архітектурні особливості приміщень мають критичний вплив на ефективність системи відеоспостереження. При проєктуванні системи необхідно враховувати:

- геометрію приміщення (форма, розміри, висота стель);
- наявність та розташування перешкод (стіни, колони, меблі);
- розташування входів та виходів;
- зони особливої уваги (каси, сейфи, складські приміщення);
- особливості освітлення (вікна, штучне освітлення).

Різні типи приміщень потребують різних підходів до розміщення камер (табл. 1.8), оскільки мають специфічні вимоги та обмеження.

Для ефективного врахування архітектурних особливостей приміщень при автоматичному розміщенні камер необхідно розробити методи цифрового представлення приміщень, які дозволять алгоритмам аналізувати геометрію та перешкоди [17]. Одним із підходів є використання двовимірних планів приміщень з позначенням стін, дверей, вікон та інших елементів

Таблиця 1.8 - Особливості розміщення камер у різних типах приміщень

Тип приміщення	Особливості	Рекомендовані типи камер	Типові виклики
Офісні простори	Відкриті площі, перегородки	Купольні, панорамні	Баланс між приватністю та безпекою
Складські приміщення	Високі стелі, стелажі	Циліндричні, PTZ	Покриття вузьких проходів
Торгові зали	Відкриті простори з перешкодами	Купольні, панорамні	Відстеження руху клієнтів
Периметр будівлі	Відкритий простір	Циліндричні з ІЧ-підсвіткою	Погодні умови, дальність огляду
Приватні будинки	Компактні приміщення	Компактні купольні	Обмеження бюджету, естетика

Аналіз впливу архітектурних особливостей приміщень на ефективність системи відеоспостереження дозволяє визначити ключові фактори, які необхідно враховувати при розробці алгоритмів оптимізації розміщення камер.

1.3 Формулювання вимог до програмного забезпечення

Формулювання вимог до програмного забезпечення для оптимізації розміщення камер відеоспостереження вимагає всебічного аналізу функціональних та нефункціональних характеристик майбутньої системи. Це дозволить створити базис для успішної реалізації проєкту та перевірки його відповідності потребам кінцевих користувачів [1].

Функціональні вимоги визначають основні можливості та функції, які програмне забезпечення повинно виконувати. Для системи оптимізації розміщення камер відеоспостереження ключовими функціональними вимогами є забезпечення можливості завантаження та редагування планів приміщень; користувач повинен мати можливість завантажити плани приміщень у поширених форматах, таких як PNG, JPG, PDF, а також створювати плани з нуля за допомогою вбудованого редактора; важливим аспектом функціональності є можливість визначення масштабу та розмірів приміщення, а також додавання або видалення стін, дверей, вікон та інших елементів інтер'єру, що можуть впливати на розміщення камер та їх зони покриття [2]. Наступною важливою функціональною вимогою є визначення параметрів камер, що включає створення бібліотеки типових камер з різними характеристиками, включаючи кут огляду, дальність дії та роздільну здатність; користувачі також повинні мати можливість додавати власні моделі камер з налаштуванням специфічних параметрів, а також встановлювати обмеження щодо загальної кількості камер та бюджету проєкту [3].

Центральним елементом програмного забезпечення є алгоритм оптимізації розміщення, який повинен автоматично розраховувати оптимальне розташування камер з урахуванням геометрії приміщення, наявних перешкод та можливих "сліпих зон"; користувачі повинні мати можливість ручного коригування автоматично згенерованого розміщення, а система має надавати інформацію про відсоток покриття приміщення та якість спостереження в різних зонах [4]. Візуалізація результатів повинна забезпечувати 2D-відображення зон покриття на плані приміщення [5].

Діаграма прецедентів використання системи представлена на рисунку 1.7, де відображено основні сценарії взаємодії користувача з програмним забезпеченням.



Рисунок 1.7 - Діаграма прецедентів використання системи оптимізації розміщення камер

Нефункціональні вимоги визначають характеристики та обмеження програмного забезпечення, які не пов'язані з конкретними функціями. У контексті системи оптимізації розміщення камер відеоспостереження, важливими нефункціональними вимогами є вимоги до продуктивності; система повинна забезпечувати швидкий відгук при роботі з планами приміщень та камерами; час розрахунку оптимального розміщення камер не повинен перевищувати 30 секунд для приміщень середньої складності;

візуалізація результатів повинна відбуватися без затримок, що заважають роботі користувача [10].

Надійність системи є критичним аспектом; програмне забезпечення повинно стабільно працювати з різними типами та розмірами планів приміщень; система повинна коректно обробляти помилки введення користувача та неправильні формати даних; важливим аспектом надійності є збереження проєктів та запобігання втраті даних при непередбачуваних ситуаціях [11]. Безпека даних також є важливою нефункціональною вимогою; система повинна забезпечувати захист користувацьких даних та проєктів; для персональних даних користувачів має бути впроваджено шифрування; важливим аспектом безпеки є контроль доступу до проєктів [12].

Масштабованість системи забезпечується можливістю роботи з приміщеннями різного розміру та складності; система повинна підтримувати роботу з великою кількістю користувачів та проєктів без зниження продуктивності; архітектура програмного забезпечення повинна дозволяти легко додавати нові функції та модулі [13]. Важливим аспектом є зручність використання; інтерфейс користувача повинен бути інтуїтивно зрозумілим та не вимагати спеціальних технічних знань; всі елементи керування повинні мати підказки та бути згруповані за логічними категоріями; система повинна забезпечувати зворотний зв'язок про результати дій користувача [14].

Доступність системи забезпечується підтримкою різних пристроїв (десктоп, планшет) та операційних систем (Windows, macOS, Linux); інтерфейс повинен бути доступним для користувачів з різними потребами, включаючи підтримку високої контрастності та масштабування; важливим аспектом доступності є наявність детальної документації та навчальних матеріалів [15]. Локалізація та інтернаціоналізація також є важливими вимогами; система повинна підтримувати кілька мов інтерфейсу, включаючи українську, англійську та російську; всі повідомлення, підказки та документація повинні бути доступні на всіх підтримуваних мовах [16].

Діаграма якісних характеристик програмного забезпечення відповідно до ISO/IEC 25010 представлена на (рис. 1.8).



Рисунок 1.8 - Діаграма якісних характеристик системи оптимізації розміщення камер

Технічні обмеження та вимоги до інтерфейсу визначають технологічні аспекти реалізації програмного забезпечення. Для системи оптимізації розміщення камер відеоспостереження технічними обмеженнями та вимогами є вимоги до середовища розробки та розгортання; система повинна бути реалізована як вебзастосунок з використанням сучасних технологій для фронтенд-розробки (React, TypeScript); серверна частина повинна бути розроблена з використанням Node.js та Express.js; для зберігання даних рекомендується використовувати MongoDB [17].

Вимоги до апаратного забезпечення серверу включають процесор з не менше 4 ядрами; оперативну пам'ять не менше 8 Гб; дисковий простір не менше 100 Гб; стабільне підключення до інтернет з пропускнуою здатністю не менше 100 Мбіт/с [18]. Вимоги до клієнтського обладнання є наступними: сучасний браузер (Chrome, Firefox, Safari, Edge останніх версій); оперативна пам'ять не менше 4 Гб; стабільне підключення до інтернет [19].

Архітектура системи повинна базуватися на мікросервісному підході; окремі модулі повинні відповідати за різні аспекти функціональності (управління планами приміщень, параметрами камер, алгоритм оптимізації, візуалізація); комунікація між модулями повинна здійснюватися через чітко

визначені API [20]. Діаграма архітектури системи представлена на рисунку 1.9.

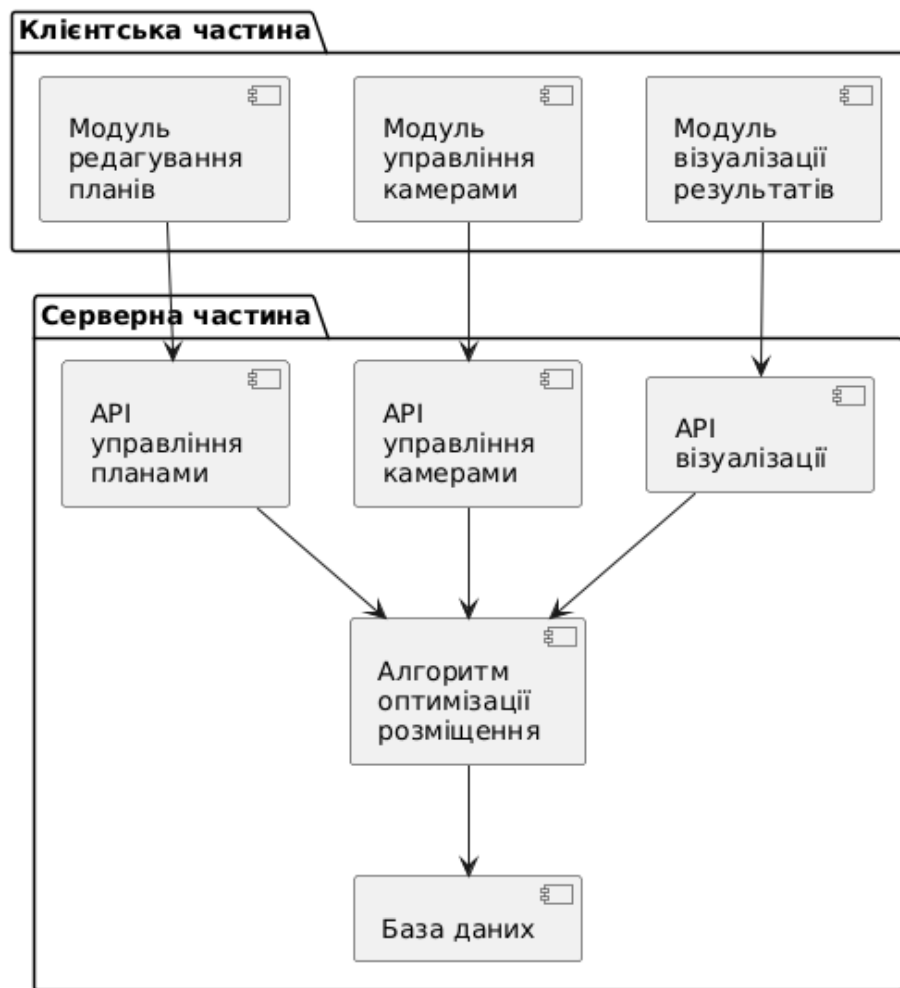


Рисунок 1.9 - Архітектура системи оптимізації розміщення камер

Технічні обмеження щодо використання сторонніх бібліотек та компонентів включають необхідність використання відкритих та безкоштовних рішень з активною підтримкою; всі залежності повинні бути задокументовані та мати відкритий код; важливим аспектом є мінімізація кількості залежностей для зменшення ризиків безпеки [21].

Вимоги до інтерфейсу користувача визначають необхідність дотримання сучасних принципів дизайну; інтерфейс повинен бути реактивним та адаптуватися до різних розмірів екрану; використання єдиного стилю оформлення для всіх елементів інтерфейсу; важливим аспектом є

забезпечення швидкого завантаження та відображення інтерфейсу [22].
Діаграма структури інтерфейсу користувача представлена на рисунку 1.10.

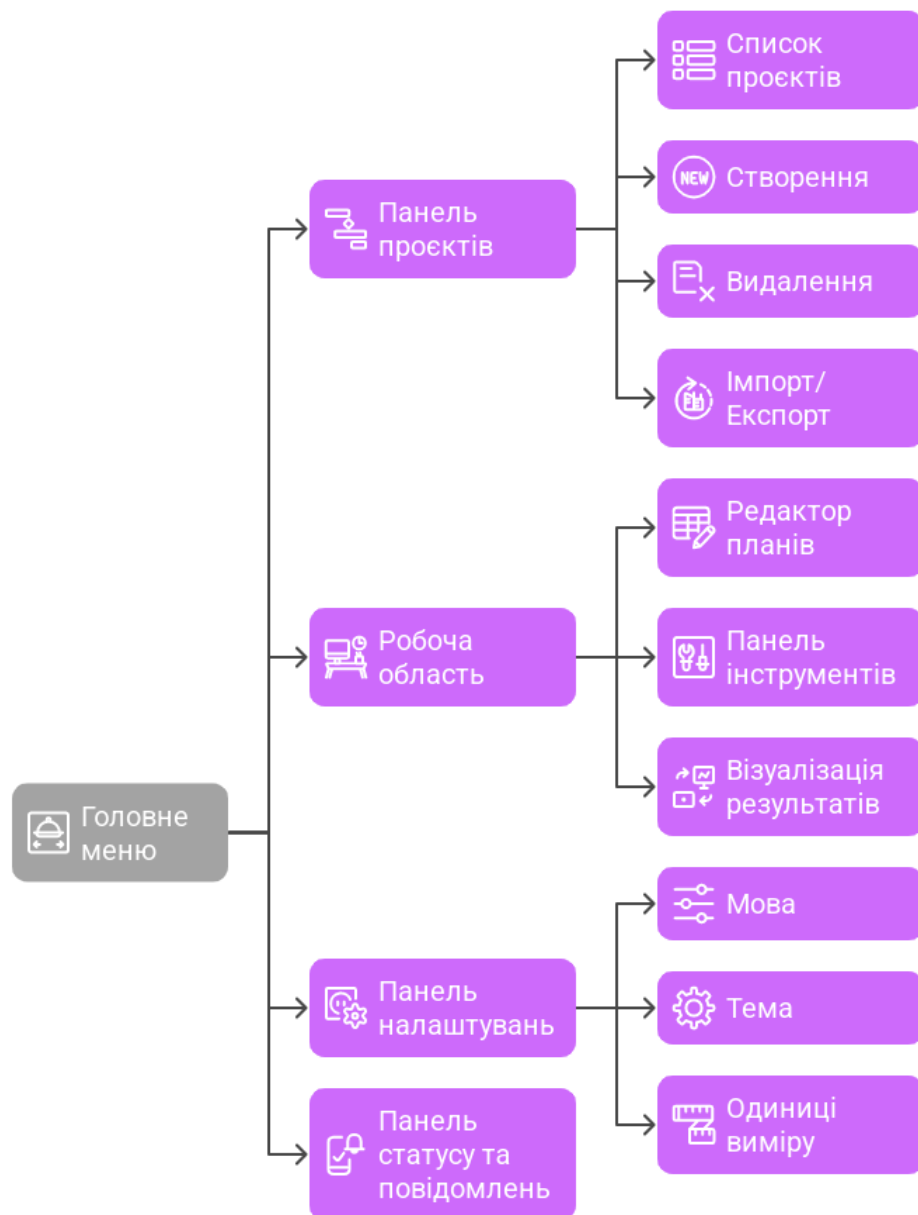


Рисунок 1.10 - Структура інтерфейсу користувача системи оптимізації розміщення камер

Вимоги до взаємодії з іншими системами включають можливість інтеграції з системами проектування інтер'єрів та архітектурними програмами через стандартні формати обміну даними (DXF, SVG); підтримку експорту даних у формати, сумісні з програмним забезпеченням

для управління проектами; інтеграцію з каталогами обладнання через API [23] - [24].

Діаграма класів основних компонентів системи представлена на рисунку 1.11, що демонструє структуру та взаємозв'язки між ключовими елементами програмного забезпечення.

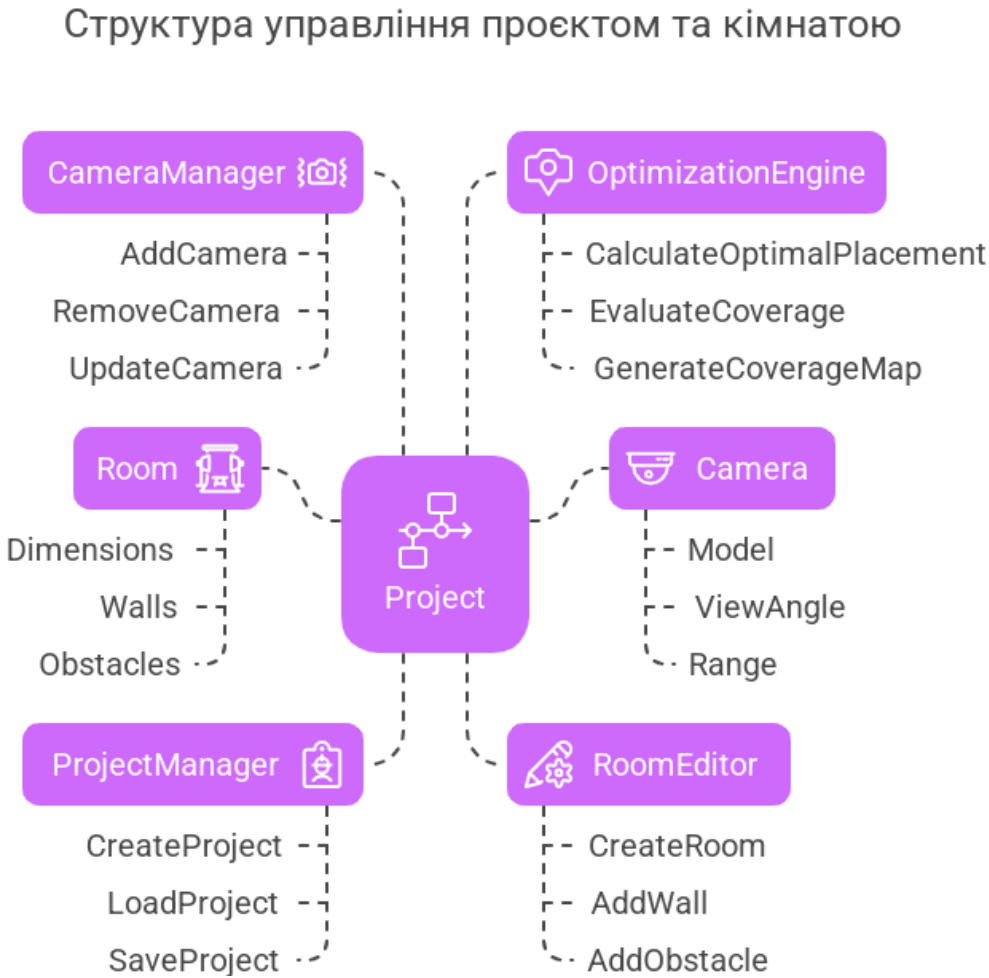


Рисунок 1.11 - Діаграма класів системи оптимізації розміщення камер

Сценарії використання програмного забезпечення описують типові взаємодії користувачів з системою та послідовність дій для досягнення певних цілей. Основні сценарії використання системи оптимізації розміщення камер відеоспостереження представлені в таблиці 1.9, де детально описані кроки взаємодії користувача з системою та результати цих взаємодій.

Таблиця 1.9 - Основні сценарії використання системи оптимізації розміщення камер

Сценарій	Актор	Кроки	Результат
Створення нового проєкту	Користувач	1. Користувач входить в систему; 2. Вибирає опцію "Створити новий проєкт"; 3. Вводить назву та опис проєкту; 4. Підтверджує створення.	Створено новий проєкт, користувач перенаправлений до редактора планів.
Завантаження плану приміщення	Користувач	1. У проєкті користувач вибирає опцію "Завантажити план"; 2. Вибирає файл з локального комп'ютера; 3. Система відображає план; 4. Користувач встановлює масштаб та орієнтацію; 5. Підтверджує завантаження.	План приміщення завантажено та відображено в редакторі.
Редагування плану приміщення	Користувач	1. Користувач вибирає інструмент редагування; 2. Додає або видаляє стіни, двері, вікна; 3. Визначає розміри елементів; 4. Змінює параметри приміщення; 5. Зберігає зміни.	План приміщення оновлено з урахуванням внесених змін.
Налаштування параметрів камер	Користувач	1. Користувач переходить до розділу "Параметри камер"; 2. Вибирає модель камери з каталогу або створює нову; 3. Налаштовує кут огляду, дальність, роздільну здатність; 4. Встановлює обмеження щодо кількості камер; 5. Зберігає налаштування.	Параметри камер налаштовані та збережені.

Продовження таблиці 1.9

Сценарій	Актор	Кроки	Результат
Запуск алгоритму оптимізації	Користувач	1. Користувач вибирає опцію "Розрахувати оптимальне розміщення"; 2. Система запускає алгоритм оптимізації; 3. Користувач отримує повідомлення про завершення розрахунків.	Система розрахувала оптимальне розміщення камер та готова відобразити результати.
Перегляд результатів оптимізації	Користувач	1. Після завершення розрахунків користувач переглядає візуалізацію результатів на плані; 2. Аналізує зони покриття та "сліпі зони"; 3. Переглядає симуляцію відображення з позиції камер; 4. Оцінює відсоток покриття приміщення.	Користувач отримав візуальне представлення оптимального розміщення камер та зон покриття.
Ручне коригування розміщення камер	Користувач	1. Користувач вибирає режим ручного коригування; 2. Переміщує, додає або видаляє камери на плані; 3. Система в реальному часі оновлює інформацію про зони покриття; 4. Користувач зберігає внесені зміни.	Розміщення камер скориговано вручну, система оновила інформацію про зони покриття.

Детальний опис одного з ключових сценаріїв - "Запуск алгоритму оптимізації" представлено на рисунку 1.12 у вигляді діаграми послідовності, що демонструє взаємодію між компонентами системи під час виконання цього сценарію.

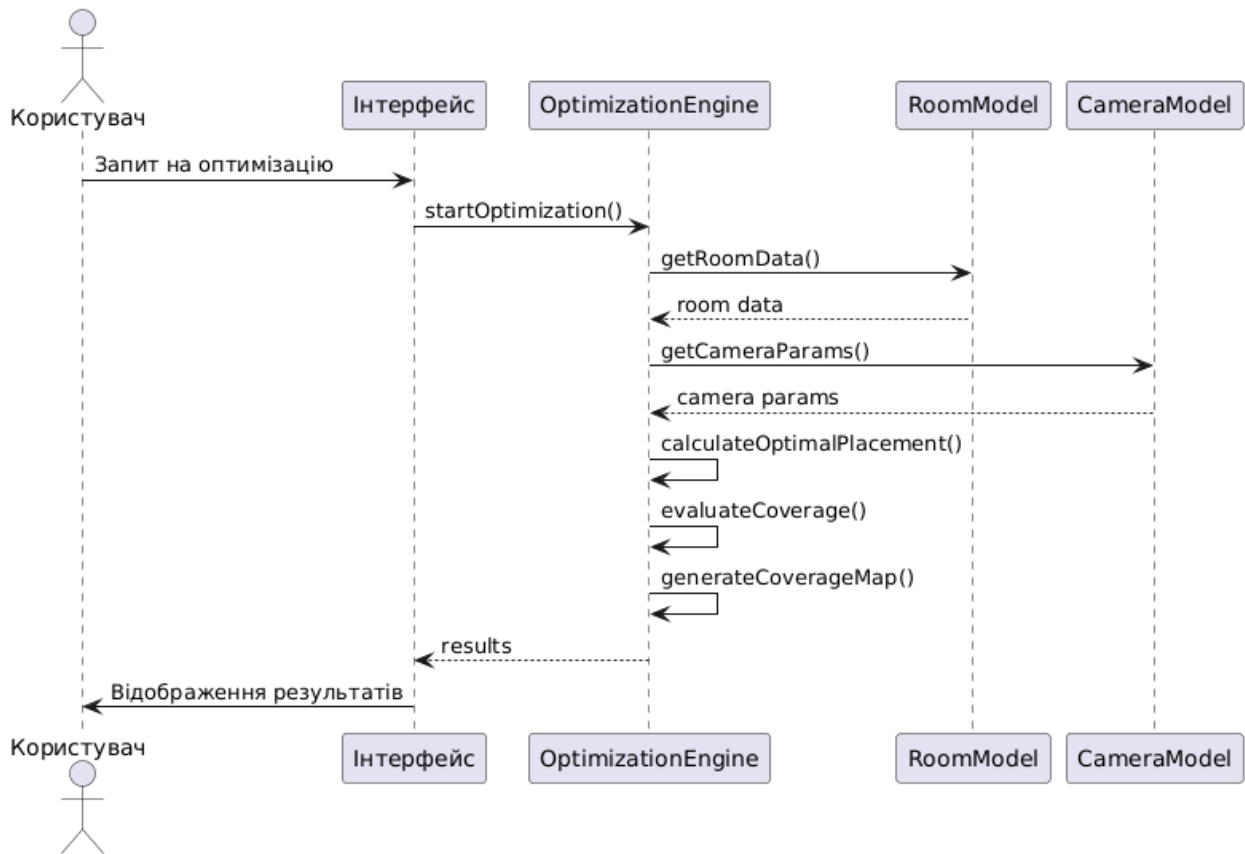


Рисунок 1.12 - Діаграма послідовності для сценарію "Запуск алгоритму оптимізації"

1.4 Висновки до розділу 1

У першому розділі було проведено комплексний аналіз предметної області систем відеоспостереження та проблематики оптимізації розміщення камер. Проведене дослідження дозволило сформулювати чітке розуміння сучасного стану технологій відеоспостереження, методів оптимізації розміщення камер та вимог до розроблюваного програмного забезпечення.

Аналіз сучасних систем відеоспостереження показав значну еволюцію від простих аналогових пристроїв до комплексних цифрових рішень з широкими аналітичними можливостями. Визначено основні типи камер, їх технічні характеристики та параметри, які критично впливають на ефективність системи спостереження в цілому. Дослідження існуючих програмних рішень виявило головний недолік більшості продуктів на ринку -

відсутність або обмежену функціональність оптимізації розміщення камер, що підтверджує актуальність розроблюваного програмного забезпечення.

На основі проведеного аналізу було сформульовано функціональні та нефункціональні вимоги до програмного забезпечення, технічні обмеження та вимоги до інтерфейсу. Розроблені діаграми прецедентів, якісних характеристик, архітектури системи та структури інтерфейсу користувача забезпечують чітке розуміння майбутньої системи та її взаємодії з користувачами. Детально описані сценарії використання системи дозволяють визначити основні потоки взаємодії користувачів з програмним забезпеченням та забезпечити його відповідність реальним потребам цільової аудиторії.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗМІЩЕННЯ КАМЕР ВІДЕОСПОСТЕРЕЖЕННЯ

2.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення для оптимізації розміщення камер відеоспостереження визначає організацію системи, її структурні елементи та їх взаємодію. Правильно спроектована архітектура забезпечує не лише ефективність роботи системи, але й її гнучкість, масштабованість та підтримуваність.

Для розроблюваної системи оптимізації розміщення камер відеоспостереження обрано багаторівневу клієнт-серверну архітектуру з мікросервісним підходом. Така архітектура дозволяє забезпечити незалежність окремих компонентів, їх масштабованість та можливість паралельної розробки. Основними рівнями архітектури є: рівень представлення (клієнтська частина), рівень бізнес-логіки (серверна частина) та рівень даних (база даних та файлове сховище). Кожен рівень має чітко визначену відповідальність та взаємодіє з іншими через визначені інтерфейси.

Рівень представлення реалізовано як одно-сторінковий вебзастосунок (SPA) з використанням бібліотеки React та TypeScript. Цей рівень відповідає за відображення даних користувачеві, отримання користувацьких введень та передачу їх на сервер для обробки. Використання React дозволяє створити реактивний інтерфейс, що швидко реагує на дії користувача та забезпечує плавний користувацький досвід.

Рівень бізнес-логіки, реалізований з використанням Node.js та Express.js, містить основну логіку системи та розподілений на мікросервіси. Основними мікросервісами є: сервіс автентифікації та авторизації, сервіс управління проєктами, сервіс управління планами приміщень, сервіс параметрів камер та сервіс оптимізації розміщення. Кожен мікросервіс є

незалежним компонентом з власною відповідальністю та API для взаємодії з іншими компонентами системи.

Рівень даних включає нереляційну базу даних MongoDB для зберігання інформації про користувачів, проекти, параметри камер, а також файлове сховище для зберігання графічних файлів планів приміщень та візуалізацій. Вибір MongoDB обумовлений гнучкістю схеми даних та можливістю масштабування, що відповідає вимогам розроблюваної системи.

Загальна структура архітектури системи представлена на рисунку 2.1, де показані основні компоненти та їх взаємодія.

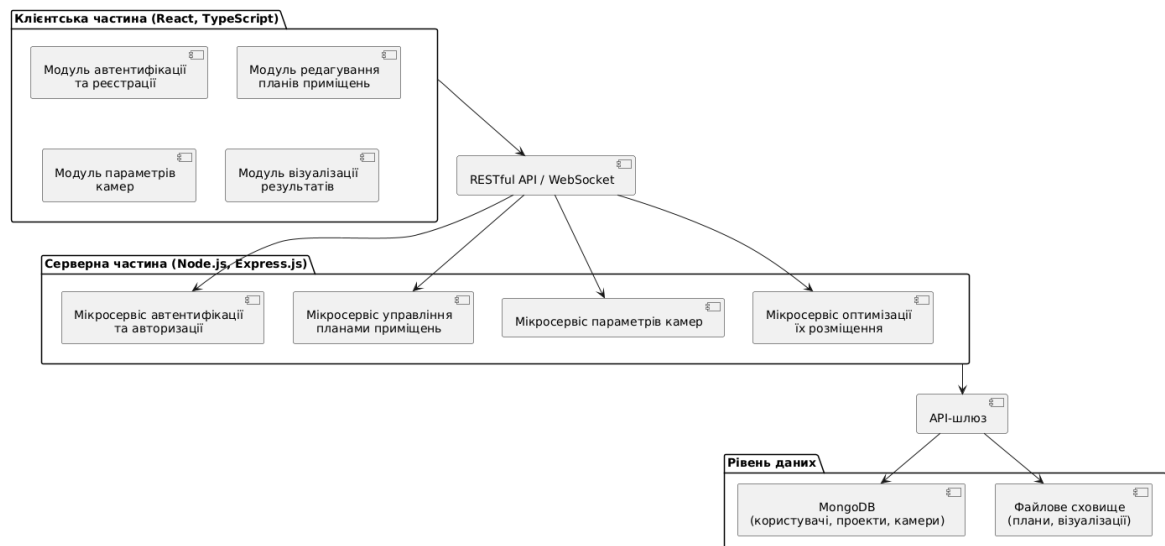


Рисунок 2.1 - Архітектура програмного забезпечення оптимізації розміщення камер відеоспостереження

Для забезпечення ефективної комунікації між компонентами системи використовується API-шлюз (API Gateway), який відповідає за маршрутизацію запитів до відповідних мікросервісів, агрегацію результатів, а також забезпечує додатковий рівень безпеки та моніторингу. Комунікація між клієнтською та серверною частиною здійснюється через RESTful API для більшості операцій та WebSocket для забезпечення реального часу оновлення даних при візуалізації результатів розміщення камер.

Одним з ключових компонентів системи є мікросервіс оптимізації розміщення, який реалізує алгоритми для розрахунку оптимального розміщення камер відеоспостереження з урахуванням геометрії приміщення, наявних перешкод та характеристик камер. Цей мікросервіс має найбільші вимоги до обчислювальних ресурсів, тому архітектура передбачає можливість його горизонтального масштабування.

Для забезпечення надійності системи та стійкості до відмов окремих компонентів впроваджено механізми відмовостійкості, які забезпечують збереження даних та можливість продовження роботи системи навіть при тимчасовій недоступності окремих мікросервісів. Це досягається шляхом реалізації патернів Circuit Breaker та Bulkhead.

Детальний опис структури мікросервісів та їх відповідальності представлено в таблиці 2.1, де визначено основні функції кожного мікросервісу та технології, що використовуються для їх реалізації.

Таблиця 2.1 - Структура мікросервісів системи оптимізації розміщення камер

Мікросервіс	Відповідальність	Використані технології
1	2	3
Автентифікації та авторизації	Реєстрація та автентифікація користувачів; управління сесіями; контроль доступу до ресурсів; інтеграція з зовнішніми провайдерами автентифікації.	Node.js, Express.js, JSON Web Tokens, OAuth 2.0, MongoDB.
Управління планами приміщень	Завантаження та редагування планів приміщень; конвертація між різними форматами файлів; масштабування та обробка зображень; зберігання та версіонування планів.	Node.js, Express.js, Sharp, SVG.js, MongoDB, GridFS.

Продовження таблиці 2.1

1	2	3
Параметрів камер	Управління бібліотекою камер; розрахунок зон покриття для різних моделей камер; валідація параметрів камер; інтеграція з каталогами обладнання.	Node.js, Express.js, MongoDB, Three.js.
Оптимізації розміщення	Реалізація алгоритмів оптимізації розміщення камер; розрахунок зон покриття та "сліпих зон"; оцінка якості покриття приміщення; оптимізація кількості камер відповідно до заданих обмежень.	Node.js, Express.js, TensorFlow.js, Three.js.
API-шлюз	Маршрутизація запитів до відповідних мікросервісів; агрегація результатів; кешування; моніторинг та логування; балансування навантаження; забезпечення безпеки API.	Node.js, Express.js, Redis, Nginx, Kong.

У розробленій архітектурі реалізовано ряд патернів проєктування, які забезпечують гнучкість, розширюваність та підтримуваність системи. Основними використаними патернами є: Model-View-Controller (MVC) для структурування коду, Repository для абстрагування доступу до даних, Factory для створення об'єктів, Strategy для реалізації різних алгоритмів оптимізації, Observer для реалізації реактивного оновлення інтерфейсу, Command для інкапсуляції операцій редагування планів приміщень.

Детальна структура патернів проєктування та їх застосування в системі представлена на рисунку 2.2, де показано основні патерни та компоненти, в яких вони використовуються.

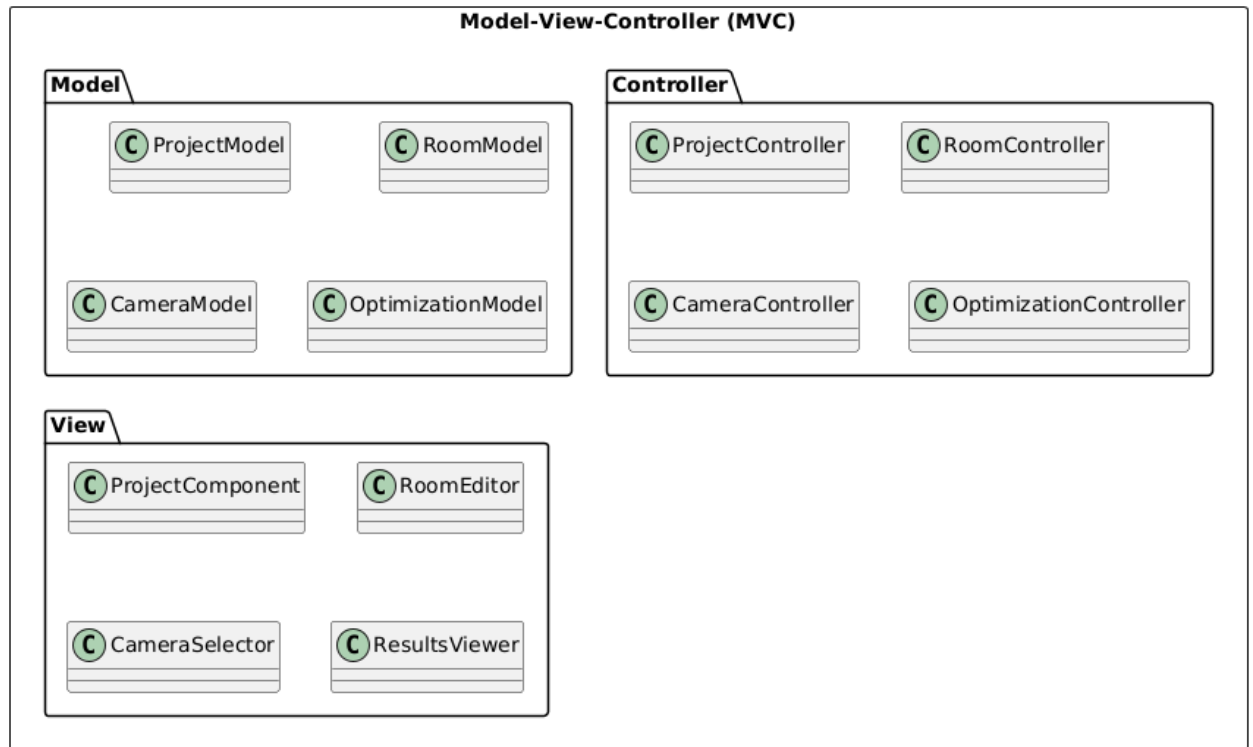


Рисунок 2.2 - Патерни проєктування в архітектурі програмного забезпечення

Для реалізації клієнтської частини системи використовується архітектурний підхід, заснований на компонентах, з використанням бібліотеки React та архітектури Flux/Redux для управління станом додатку. Такий підхід забезпечує передбачуваний потік даних, спрощує тестування та дозволяє ефективно керувати складним станом додатку.

Структура основних компонентів клієнтської частини та їх взаємодія представлена на рисунку 2.3, де показано ієрархію компонентів та потік даних між ними.

Серверна частина системи реалізована з використанням Node.js та Express.js, з організацією коду відповідно до принципів чистої архітектури (Clean Architecture). Це забезпечує незалежність бізнес-логіки від фреймворків, баз даних та інших зовнішніх компонентів, що спрощує тестування та підтримку системи.

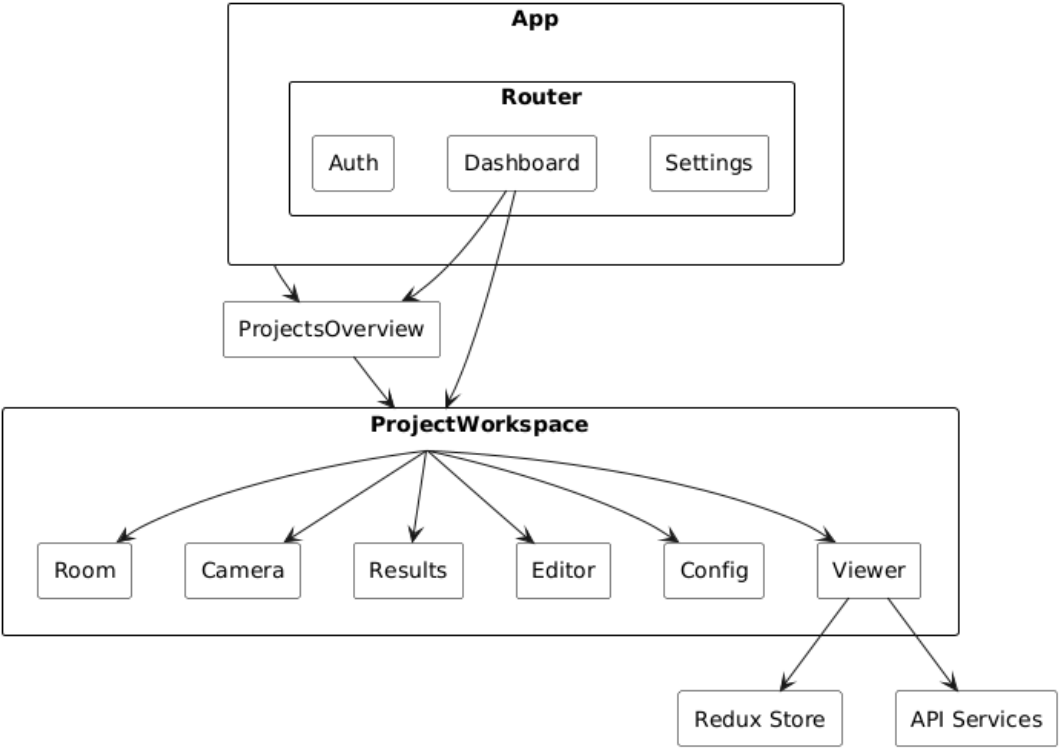


Рисунок 2.3 - Структура компонентів клієнтської частини системи

Структура шарів чистої архітектури в серверних компонентах представлена на рисунку 2.4, де показано основні шари та їх відповідальність.

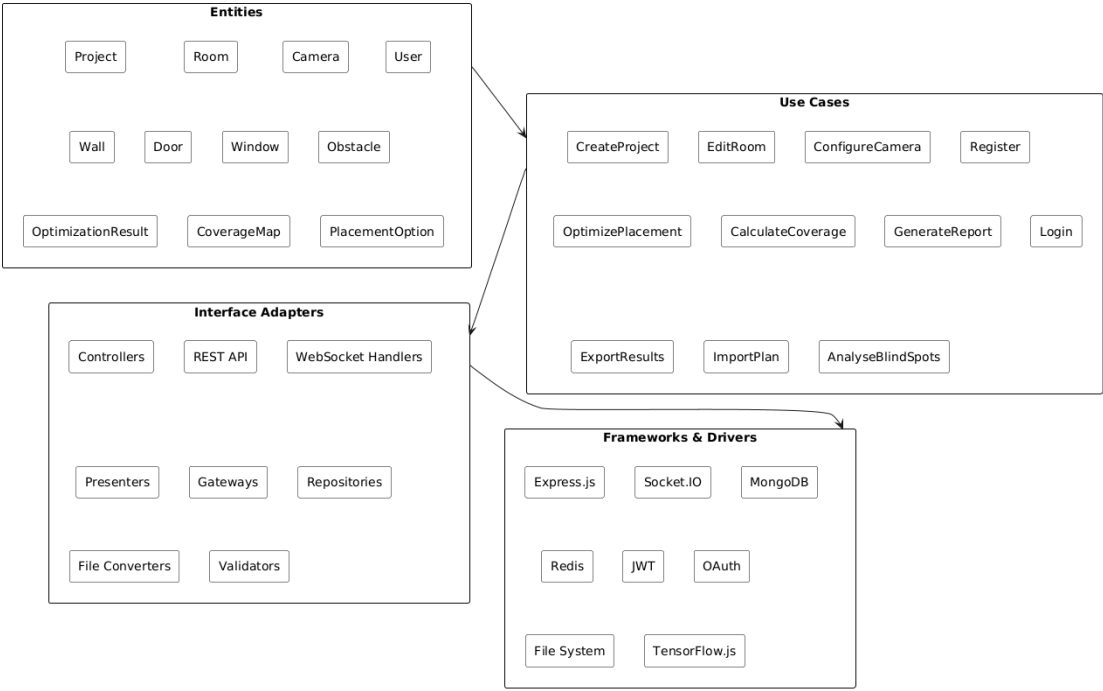


Рисунок 2.4 - Структура чистої архітектури в серверній частині системи

Для забезпечення безпеки системи впроваджено багаторівневий підхід, який включає захист від типових атак вебзастосунків (OWASP Top 10), шифрування чутливих даних, використання HTTPS для захисту передачі даних, впровадження механізмів автентифікації та авторизації на основі JWT (JSON Web Tokens), а також логування та моніторинг підозрілої активності.

Для розгортання та розробки системи використовується контейнеризація на основі Docker, що забезпечує ізоляцію компонентів, спрощує розгортання та масштабування системи, а також забезпечує однакове середовище для розробки, тестування та експлуатації.

Для забезпечення якості розроблюваного програмного забезпечення впроваджено практики Continuous Integration / Continuous Deployment (CI/CD), автоматизоване тестування (модульні, інтеграційні та end-to-end тести), статичний аналіз коду та автоматизоване збирання.

Таким чином, розроблена архітектура програмного забезпечення для оптимізації розміщення камер відеоспостереження забезпечує гнучкість, масштабованість, надійність та підтримуваність системи. Використання мікросервісного підходу, сучасних патернів проєктування та технологій дозволяє створити ефективне рішення, яке відповідає функціональним та нефункціональним вимогам до системи.

2.2 Алгоритм оптимізації розміщення камер

Реалізація ефективного алгоритму оптимізації розміщення камер відеоспостереження є найбільш складною та критичною частиною розроблюваної системи. Цей алгоритм повинен враховувати геометрію приміщення, розташування перешкод, параметри камер та різноманітні обмеження для забезпечення максимального покриття при мінімальних витратах.

Розробка алгоритму оптимізації розміщення камер починається з формалізації задачі у вигляді математичної моделі. Приміщення представляється як двовимірний багатокутник P , що містить набір перешкод $O = \{O_1, O_2, \dots, O_n\}$, які також є багатокутниками. Кожна камера характеризується набором параметрів: позицією $p_c = (x_c, y_c)$, кутом огляду α_c , напрямком d_c та дальністю видимості r_c . Зона огляду камери моделюється як сектор S_c , визначений параметрами камери. Основна мета оптимізації полягає у знаходженні набору камер $C = \{c_1, c_2, \dots, c_n\}$, які забезпечують максимальне покриття приміщення P за вирахуванням перешкод O , з урахуванням мінімізації кількості камер та їх вартості [2].

Формально задачу оптимізації можна визначити наступним чином (2.1):

$$\begin{aligned} \max U(C) &= \text{Area}(\text{Union}(S_{c_1}, S_{c_2}, \dots, S_{c_n}) \cap (P \setminus \text{Union}(O))) \\ \text{s.t. } |C| &\leq N_{\max} \text{ Cost}(C) \leq B_{\max}, \end{aligned} \quad (2.1)$$

де $U(C)$ - функція корисності, що відображає площу покриття набором камер C ; $\text{Area}()$ - функція обчислення площі; $\text{Union}()$ - операція об'єднання; " $\cap$ " - операція перетину; $N_{\max}$ - максимальна допустима кількість камер; $B_{\max}$ - максимальний бюджет; $\text{Cost}(C)$ - функція обчислення вартості набору камер C .

Для розв'язання цієї задачі розроблено комбінований алгоритм, що поєднує методи геометричного моделювання, жадібною оптимізації та метаевристичного пошуку. Основні етапи алгоритму представлені на рисунку 2.5.

Першим етапом алгоритму є дискретизація простору приміщення. Для цього використовується метод розбиття приміщення на сітку клітин з рівномірним кроком. Кожна клітина сітки містить інформацію про свій стан (вільна, зайнята перешкодою) та важливість з точки зору покриття (наприклад, клітини біля входів, цінних об'єктів можуть мати вищу

важливість). Крок сітки визначається залежно від розмірів приміщення та необхідної точності обчислень.



Рисунок 2.5 - Структура алгоритму оптимізації розміщення камер

Результатом дискретизації є матриця M розміром $W \times H$, де W та H - кількість клітин сітки по ширині та висоті відповідно.

Для визначення зони видимості камери використовується алгоритм променевого кидання (ray casting). З позиції камери у різних напрямках в межах кута огляду випускаються промені, і для кожного променя визначається перша точка перетину з перешкодою або стіною приміщення.

Це дозволяє визначити видимі та невидимі ділянки приміщення з позиції камери. Враховуються також параметри дальності видимості камери, що обмежує довжину променів.

Після дискретизації приміщення та визначення методу обчислення зон видимості, наступним етапом є генерація початкового розташування камер. Для цього використовується жадібний алгоритм, який послідовно додає камери в позиції, що забезпечують максимальне приросте покриття:

- ініціалізація пустого набору камер $C = \emptyset$ та непокритої області $U = P \setminus \text{Union}(O)$;
- поки $|C| < N_max$ та $\text{Area}(U) > 0$: а. Знайти позицію p^* і напрямок d^* , які забезпечують максимальне покриття непокритої області; б. Створити нову камеру c^* з параметрами p^* , d^* та додати її до набору C / Оновити непокриту область $U = U \setminus S_{c^*}$;
- повернути набір камер C .

Для пошуку оптимальної позиції камери на кожному кроці жадібного алгоритму використовується метод сканування дискретизованого простору. Для кожної можливої позиції камери обчислюється покриття, і вибирається позиція з максимальним значенням. Такий підхід забезпечує локально оптимальне рішення на кожному кроці, але не гарантує глобального оптимуму для всієї задачі.

Для покращення початкового розташування, отриманого жадібним алгоритмом, використовується метаевристичний метод імітації відпалу (simulated annealing). Цей метод дозволяє уникнути локальних оптимумів за рахунок прийняття з певною ймовірністю рішень, що тимчасово погіршують цільову функцію. Основні кроки алгоритму імітації відпалу:

- ініціалізація: поточний стан S та початкове розташування камер, температура $T = T_0$;
- поки $T > T_min$: для i від 1 до N_iter : *i*. Згенерувати сусідній стан S' шляхом випадкового переміщення, обертання або зміни типу однієї з камер; *ii*. Обчислити зміну цільової функції $\Delta E = U(S') - U(S)$; *iii*. Якщо $\Delta E > 0$ або

$\exp(\Delta E/T) > \text{random}(0,1)$, то $S = S'$; *b*. Зменшити температуру: $T = \alpha \times T$, де α - коефіцієнт охолодження ($0 < \alpha < 1$);

– повернути фінальний стан S .

Для генерації сусіднього стану використовуються три типи операцій: переміщення камери (зміна координат), обертання камери (зміна напрямку) та зміна типу камери (вибір іншої моделі з іншими параметрами). Ймовірність вибору кожної операції може налаштовуватися залежно від специфіки задачі.

Після застосування алгоритму імітації відпалу, отримане розташування камер додатково оптимізується методом градієнтного спуску для точного позиціонування камер. Для кожної камери обчислюються часткові похідні цільової функції по координатах та куту повороту, і камера зміщується у напрямку найбільшого зростання функції покриття. Цей процес повторюється ітеративно до досягнення стабільного положення.

Для оцінки якості покриття приміщення використовується комбінований критерій, що враховує не лише площу покриття, але й "якість" цього покриття - роздільну здатність камери на різних відстанях, кут спостереження та перекриття зон видимості різних камер. Кожна точка приміщення оцінюється за формулою (2.2):

$$Q(p) = \sum_{c \in C(p)} w_c * q_c(p), \quad (2.2)$$

де $C(p)$ - набір камер, що спостерігають точку p ; w_c - вага камери (залежить від її характеристик); $q_c(p)$ - якість спостереження точки p камерою c , яка обчислюється за формулою (2.3):

$$q_c(p) = r_factor \times a_factor \times d_factor, \quad (2.3)$$

де r_factor залежить від роздільної здатності камери та відстані до точки, a_factor залежить від кута спостереження, d_factor залежить від наявності та відстані до перешкод [9].

Для обчислення загальної якості покриття приміщення використовується інтеграл якості по всій площі приміщення (2.4):

$$Q_total = \int P Q(p) dp. \quad (2.4)$$

На практиці, при дискретизації простору, цей інтеграл апроксимується сумою (2.5):

$$Q_total = \sum_i \sum_j Q(i,j) \times A_cell, \quad (2.5)$$

де $Q(i,j)$ - якість покриття клітини (i,j) , A_cell - площа однієї клітини сітки.

Особливу увагу в алгоритмі приділено виявленню та мінімізації "сліпих зон" - ділянок приміщення, які не покриваються жодною камерою. Для цього використовується аналіз зв'язних компонент у матриці покриття та алгоритм мінімального домінуючого набору на графі видимості (табл. 2.2).

Таблиця 2.2 - Порівняння методів оптимізації розміщення камер

Метод	Переваги	Недоліки	Застосування в розробленому алгоритмі
Жадібний алгоритм	Швидкість виконання; простота реалізації; хороші результати для простих приміщень.	Знаходження локальних оптимумів; низька якість рішень для складних приміщень.	Генерація початкового розташування камер.
Повний перебір	Гарантоване знаходження глобального оптимуму.	Експоненційна складність.	Не використовується через обчислювальну складність.

Для оцінки ефективності розробленого алгоритму проведено порівняльне тестування на наборі тестових приміщень різної складності. Результати порівняння з іншими методами представлені на рисунку 2.6, де показано залежність якості покриття від кількості використаних камер для різних методів оптимізації.

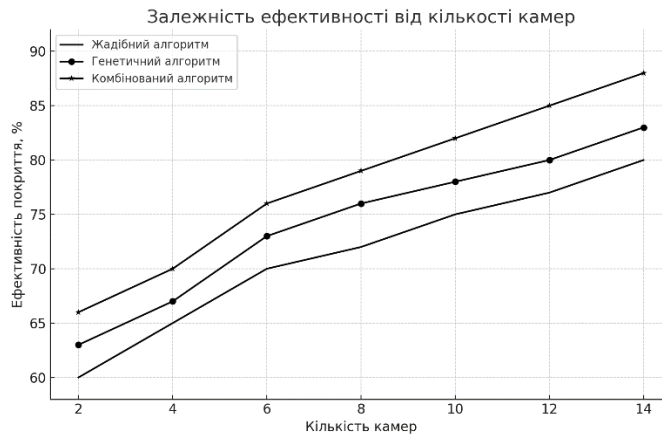


Рисунок 2.6 - Порівняння ефективності різних методів оптимізації розміщення камер

2.3 Висновки за розділом 2

У другому розділі було виконано детальне проєктування програмного забезпечення для оптимізації розміщення камер відеоспостереження з урахуванням архітектури системи, математичних алгоритмів оптимізації, структури бази даних, інтерфейсу користувача та взаємодії з зовнішніми системами.

Розроблена архітектура програмного забезпечення базується на мікросервісному підході, що забезпечує гнучкість, масштабованість та підтримуваність системи. Використання багаторівневої клієнт-серверної архітектури з чітким розподілом відповідальності між компонентами дозволяє забезпечити незалежність окремих модулів та можливість їх паралельної розробки. Реалізація різноманітних патернів проєктування

(MVC, Repository, Factory, Strategy, Observer та Command) підвищує гнучкість та розширюваність системи.

Запропонований алгоритм оптимізації розміщення камер поєднує жадібний алгоритм для генерації початкового розв'язку, метод імітації відпалу для глобальної оптимізації та градієнтний спуск для фінального уточнення результатів. Така комбінація методів дозволяє знаходити оптимальні рішення з урахуванням різноманітних обмежень: геометрії приміщення, наявних перешкод, бюджетних лімітів та мінімальних вимог до покриття. Порівняльне тестування алгоритмів показало, що запропонований комбінований підхід забезпечує найкраще співвідношення якості результатів та обчислювальної складності.

3 ОПИС ПРОГРАМНОГО ЗАБЕЩПЕЧЕННЯ

3.1 Розробка бази даних програмного забезпечення для оптимізації розміщення камер відеоспостереження

Розробка бази даних є одним із ключових етапів розробки програмного забезпечення для оптимізації розміщення камер відеоспостереження. Коректна організація даних забезпечує ефективну роботу системи, надійне зберігання інформації та зручний доступ до неї як з боку серверних компонентів, так і з боку користувацького інтерфейсу.

У контексті розроблюваної системи база даних повинна забезпечувати зберігання інформації про користувачів, проекти, плани приміщень, параметри камер, результати оптимізації та інші дані, необхідні для функціонування системи. Враховуючи специфіку задачі та вимоги до гнучкості схеми даних, для реалізації системи обрано документоорієнтовану СУБД MongoDB [2].

Проектування бази даних розпочинається з розробки концептуальної моделі, яка описує основні сутності системи та зв'язки між ними. Для системи оптимізації розміщення камер відеоспостереження основними сутностями є: користувач (User), проект (Project), приміщення (Room), камера (Camera), перешкода (Obstacle), результат оптимізації (OptimizationResult). Концептуальна модель даних представлена на рисунку 3.1, де показано основні сутності та зв'язки між ними.

На основі концептуальної моделі розроблено логічну модель бази даних, яка адаптована до особливостей документоорієнтованої СУБД MongoDB. У MongoDB дані зберігаються у вигляді JSON-подібних документів, об'єднаних у колекції. Основними колекціями в розроблюваній системі є: users, projects, rooms, cameras, optimizationResults. Враховуючи специфіку MongoDB, деякі зв'язки між сутностями реалізовані через вкладені документи, що дозволяє оптимізувати операції читання та зменшити кількість запитів до бази даних.

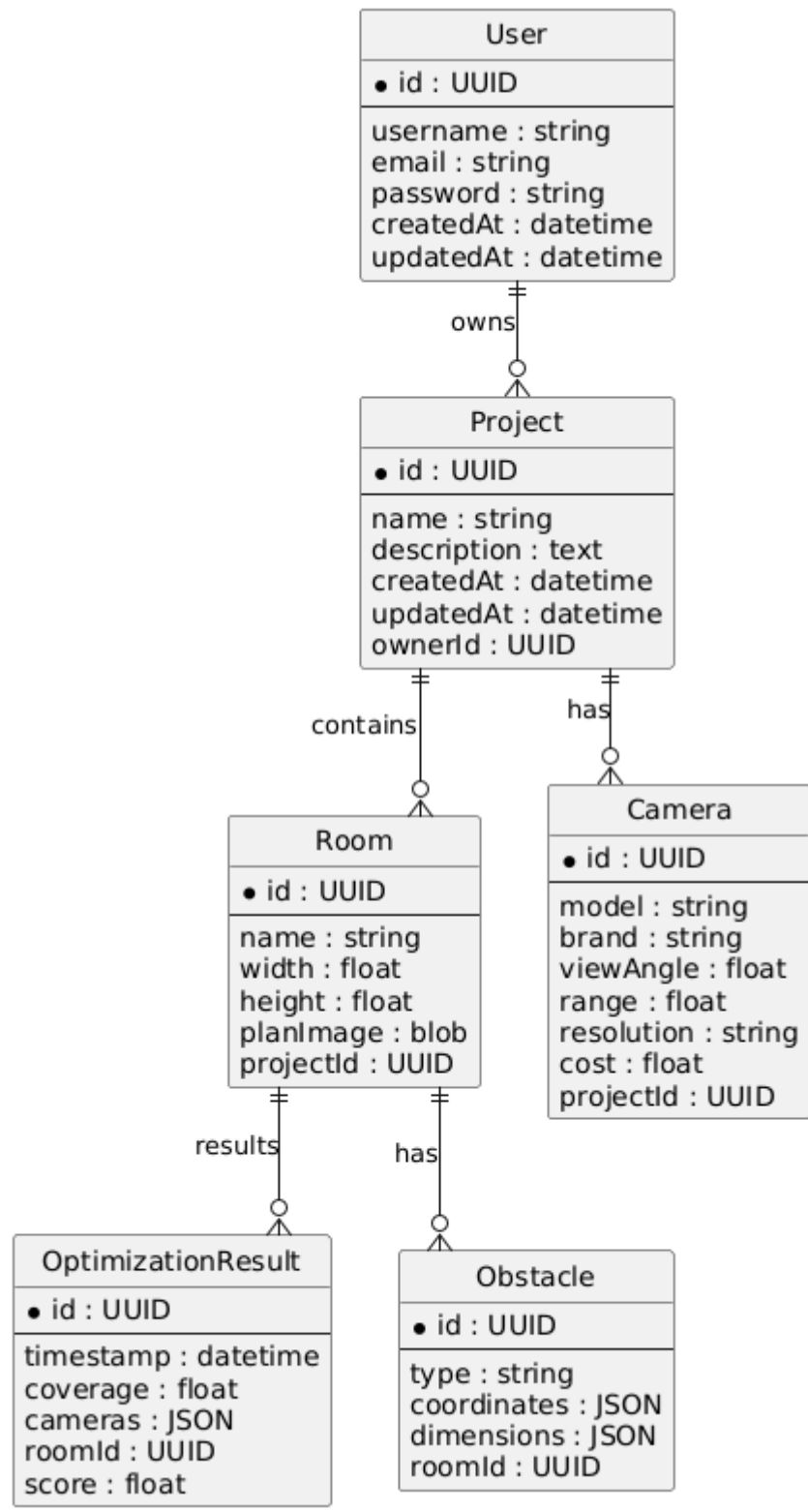


Рисунок 3.1 - Концептуальна модель даних системи

Логічна модель бази даних, адаптована для MongoDB, представлена в таблиці 3.1, де описано структуру основних колекцій.

Таблиця 3.1 - Логічна модель бази даних системи

Колекція	Поля	Опис	Індекси
users	_id: ObjectId	Унікальний ідентифікатор користувача	email: 1 (unique)
	username: String	Ім'я користувача	
	email: String	Електронна адреса користувача (унікальна)	
	passwordHash: String	Хеш пароля користувача	
	createdAt: Date	Дата створення облікового запису	
	updatedAt: Date	Дата останнього оновлення облікового запису	
	role: String	Роль користувача в системі (admin, user)	
	settings: Object	Налаштування користувача (мова, тема, одиниці виміру)	
projects	_id: ObjectId	Унікальний ідентифікатор проєкту	ownerId: 1
	name: String	Назва проєкту	createdAt: -1
	description: String	Опис проєкту	
	ownerId: ObjectId	Ідентифікатор власника проєкту	
	sharedWith: Array<ObjectId>	Список користувачів, з якими поділений проєкт	
	createdAt: Date	Дата створення проєкту	
	updatedAt: Date	Дата останнього оновлення проєкту	
	thumbnail: String	URL або Base64 мініатюри проєкту	
	tags: Array<String>	Теги для категоризації проєкту	
rooms	_id: ObjectId	Унікальний ідентифікатор приміщення	projectId: 1
	name: String	Назва приміщення	
	projectId: ObjectId	Ідентифікатор проєкту, до якого належить приміщення	
	width: Number	Ширина приміщення в метрах	
	height: Number	Довжина приміщення в метрах	

Продовження таблиці 3.1

Колекція	Поля	Опис	Індекси
rooms	walls: Array<Object>	Масив об'єктів, що описують стіни приміщення	
	planImage: String	URL або ObjectId файлу з планом приміщення	
	planScale: Number	Масштаб плану (пікселів на метр)	
	obstacles: Array<Object>	Масив об'єктів, що описують перешкоди у приміщенні	
	createdAt: Date	Дата створення приміщення	
	updatedAt: Date	Дата останнього оновлення приміщення	
cameras	_id: ObjectId	Унікальний ідентифікатор камери	projectId: 1
	model: String	Модель камери	brand: 1
	brand: String	Виробник камери	
	viewAngle: Number	Кут огляду камери в градусах	
	range: Number	Дальність дії камери в метрах	
	resolution: String	Роздільна здатність камери	
	cost: Number	Вартість камери	
	projectId: ObjectId	Ідентифікатор проєкту, до якого належить камера	
	isTemplate: Boolean	Ознака того, чи є камера шаблоном для вибору	
	features: Object	Додаткові характеристики камери	
optimizationResults	_id: ObjectId	Унікальний ідентифікатор результату оптимізації	roomId: 1
	roomId: ObjectId	Ідентифікатор приміщення	timestamp: -1
	timestamp: Date	Час проведення оптимізації	
	algorithm: String	Назва використаного алгоритму оптимізації	

Продовження таблиці 3.1

Колекція	Поля	Опис	Індекси
optimizationResults	coveragePercentage: Number	Відсоток покриття приміщення	
	quality: Number	Загальна оцінка якості покриття	
	cameras: Array<Object>	Масив об'єктів, що описують оптимальне розміщення камер	
	coverageMap: Object	Об'єкт, що описує карту покриття приміщення	
	totalCost: Number	Загальна вартість системи	
	isActive: Boolean	Ознака того, чи є даний результат активним для приміщення	

Для зберігання планів приміщень, які можуть мати значний обсяг та різні формати (PNG, JPG, SVG, PDF), використовується GridFS - функціональність MongoDB для роботи з великими файлами. GridFS розбиває файли на частини (chunks) та зберігає їх окремо, що дозволяє ефективно працювати з файлами розміром більше 16 МБ.

Для підвищення продуктивності запитів до бази даних створені індекси на основних полях пошуку та зв'язків між колекціями. Зокрема, індекси створені для полів email у колекції users, ownerId та createdAt у колекції projects, projectId у колекціях rooms та cameras, roomId та timestamp у колекції optimizationResults.

Фізична реалізація бази даних включає налаштування MongoDB для оптимальної продуктивності та надійності. Для забезпечення високої доступності та відмовостійкості використовується реплікація даних через механізм Replica Set. Для розподілу навантаження та масштабування системи при зростанні обсягу даних передбачено можливість шардингу основних колекцій.

Для забезпечення цілісності даних та перевірки правильності вводу використовується валідація на рівні схеми MongoDB. Приклад схеми валідації для колекції cameras представлений у лістингу 3.1.

Лістинг 3.1 - Схема валідації для колекції cameras

```
db.createCollection("cameras", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "model", "brand", "viewAngle",
"range", "resolution", "cost", "projectId" ],
      properties: {
        model: {
          bsonType: "string",
          description: "Модель камери (обов'язкове
поле) "
        },
        brand: {
          bsonType: "string",
          description: "Виробник камери (обов'язкове
поле) "
        },
        viewAngle: {
          bsonType: "number",
          minimum: 0,
          maximum: 360,
          description: "Кут огляду камери в градусах
(обов'язкове поле, від 0 до 360)"
        },
        range: {
          bsonType: "number",
          minimum: 0,
          description: "Дальність дії камери в метрах
(обов'язкове поле, більше 0)"
        },
        resolution: {
          bsonType: "string",
          pattern: "^[0-9]+x[0-9]+$",
          description: "Роздільна здатність камери у
форматі ШхВ (обов'язкове поле)"
        },
        cost: {
          bsonType: "number",
          minimum: 0,
          description: "Вартість камери (обов'язкове
поле, більше або дорівнює 0)"
        },
        projectId: {
          bsonType: "objectId",
```


дозволяє зменшити навантаження на базу даних та покращити швидкість системи.

Лістинг 3.2 - Агрегація для отримання статистики по проєктах користувача

```

db.projects.aggregate([
  {
    $match: {
      ownerId:
ObjectId("5f8423a1e4b0a2d870c5f8a1") } },
  { $lookup: {
    from: "rooms",
    localField: "_id",
    foreignField: "projectId",
    as: "rooms"
  }},
  { $lookup: {
    from: "optimizationResults",
    localField: "rooms._id",
    foreignField: "roomId",
    as: "results"
  }},
  { $project: {
    _id: 1,
    name: 1,
    description: 1,
    createdAt: 1,
    roomCount: { $size: "$rooms" },
    optimizationCount: { $size: "$results" },
    lastOptimization: { $max: "$results.timestamp" },
    averageCoverage: {
      $avg:
"$results.coveragePercentage" }
  }}
])

```

Для інтеграції з іншими системами та обміну даними реалізовано механізми експорту та імпорту даних у форматах JSON, CSV та XML. Це дозволяє забезпечити сумісність з іншими програмними рішеннями для проєктування приміщень та систем безпеки.

Для забезпечення відновлення даних у випадку збоїв реалізовано систему регулярного резервного копіювання бази даних. Резервні копії створюються щоденно та зберігаються у різних географічно розподілених сховищах, що забезпечує високий рівень надійності зберігання даних.

3.2 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача є критичним етапом розробки програмного забезпечення для оптимізації розміщення камер відеоспостереження, оскільки від якості та зручності інтерфейсу безпосередньо залежить ефективність взаємодії користувача з системою та її загальна корисність. Розроблюваний інтерфейс повинен бути інтуїтивно зрозумілим, забезпечувати швидкий доступ до всіх функцій системи та ефективно візуалізувати результати оптимізації розміщення камер.

Процес проєктування інтерфейсу користувача базується на принципах user-centered design (UCD), який ставить потреби, можливості та обмеження користувача в центр процесу проєктування. Цей підхід передбачає глибоке розуміння цільової аудиторії, аналіз сценаріїв використання, розробку прототипів та ітеративне тестування з реальними користувачами. Для програмного забезпечення оптимізації розміщення камер відеоспостереження цільовою аудиторією є власники нерухомості, малий та середній бізнес, які не мають спеціальних технічних знань у галузі проєктування систем безпеки.

При проєктуванні інтерфейсу користувача дотримуються принципів простоти, послідовності, зворотного зв'язку та ефективності. Простота передбачає мінімізацію кількості елементів інтерфейсу, їх логічне групування та виділення найважливіших функцій; послідовність забезпечує однаковий стиль та поведінку елементів інтерфейсу в різних частинах системи; зворотний зв'язок інформує користувача про результати його дій та стан системи; ефективність забезпечує мінімальну кількість дій для досягнення цілей користувача.

Інтерфейс користувача для системи оптимізації розміщення камер відеоспостереження реалізований як одно-сторінковий вебзастосунок (SPA) з використанням React, TypeScript та бібліотеки компонентів Material-UI. Такий підхід забезпечує плавну та реактивну взаємодію з користувачем, а також адаптивність інтерфейсу до різних розмірів екрану та пристроїв.

Загальна структура інтерфейсу користувача представлена на рисунку 3.2, де показано основні екрани системи та навігаційні зв'язки між ними.



Рисунок 3.2 - Структура інтерфейсу користувача системи

Головна сторінка системи є точкою входу для користувачів і містить інформацію про можливості системи, форми для входу та реєстрації, а також демонстраційне відео роботи з системою. Дизайн головної сторінки спрямований на залучення нових користувачів та надання їм базової інформації про систему.

Після автентифікації користувач потрапляє до особистого кабінету, де представлений список його проектів з можливістю фільтрації, сортування та пошуку. Кожен проєкт представлений у вигляді картки з мініатюрою,

назвою, описом та датою останнього редагування. Для зручності навігації при великій кількості проєктів реалізована пагінація та групування за тегами.

Створення нового проєкту відбувається через відповідну форму, де користувач вводить назву та опис проєкту, а також може вибрати шаблон для прискорення процесу проєктування. Після створення проєкту користувач автоматично перенаправляється до робочого середовища проєкту.

Робоче середовище проєкту є основним інтерфейсом для роботи з системою та складається з трьох основних модулів: редактора планів приміщень, налаштування камер та перегляду результатів оптимізації. Ці модулі представлені у вигляді вкладок для зручного перемикавання між ними.

Макет робочого середовища проєкту представлений на рисунку 3.3, де показано основні елементи інтерфейсу та їх розташування.

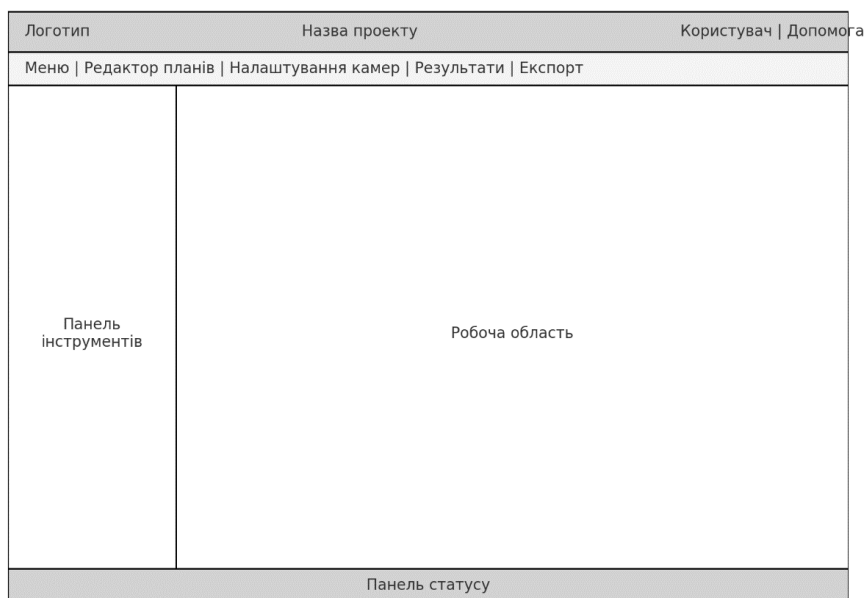


Рисунок 3.3 - Макет робочого середовища проєкту

Редактор планів приміщень є одним з ключових компонентів системи і призначений для створення, редагування та управління планами приміщень. Інтерфейс редактора включає робочу область для відображення та редагування плану, панель інструментів з засобами редагування, а також панель властивостей для налаштування параметрів вибраних елементів [9].

Основними функціями редактора планів є: створення нового плану з нуля за допомогою інструментів малювання; імпорт готових планів у форматах PNG, JPG, PDF, SVG; автоматичне розпізнавання стін та перешкод на імпортованих планах за допомогою алгоритмів комп'ютерного зору; масштабування та вирівнювання плану; додавання, видалення та редагування стін, дверей, вікон та перешкод; вимірювання відстаней та площ на плані; встановлення масштабу для точного відображення розмірів.

Для забезпечення зручності редагування та високої точності малювання у редакторі планів реалізовані такі функції як прилипання до сітки (grid snapping), прилипання до об'єктів (object snapping), вирівнювання об'єктів, групування та розгрупування елементів, а також можливість відміни та повторення дій (undo/redo).

Модуль налаштування камер призначений для вибору та конфігурації камер відеоспостереження, які будуть використовуватися в проєкті. Інтерфейс цього модуля включає бібліотеку типових камер, форму для створення користувацьких камер, а також панель для встановлення обмежень щодо кількості камер та бюджету.

Бібліотека типових камер представлена у вигляді каталогу з можливістю фільтрації за виробником, моделлю, ціновим діапазоном та технічними характеристиками. Кожна камера представлена карткою з зображенням, основними параметрами та можливістю додавання до проєкту. Інтерфейс бібліотеки камер представлений на рисунку 3.4.

Для створення користувацьких камер передбачена форма з полями для введення всіх необхідних параметрів: назва, виробник, модель, кут огляду, дальність дії, роздільна здатність, вартість та додаткові характеристики. Форма містить валідацію введених значень та підказки для користувача.

Панель обмежень дозволяє встановити максимальну кількість камер для проєкту, максимальний бюджет, а також пріоритети оптимізації: максимальне покриття при фіксованій кількості камер, мінімальна кількість

камер для забезпечення заданого рівня покриття, або баланс між покриттям та вартістю системи.

Пошук та фільтрація				Додати камеру			
---------------------	--	--	--	---------------	--	--	--

Камера 1 Параметри: - Кут: 90° - Дальн: 15м - Ціна: 100\$ [Додати]	Камера 2 Параметри: - Кут: 120° - Дальн: 20м - Ціна: 150\$ [Додати]	Камера 3 Параметри: - Кут: 180° - Дальн: 10м - Ціна: 200\$ [Додати]	Камера 4 Параметри: - Кут: 360° - Дальн: 5м - Ціна: 300\$ [Додати]
Камера 5 Параметри: - Кут: ... - Дальн: ... - Ціна: ... [Додати]	Камера 6 Параметри: - Кут: ... - Дальн: ... - Ціна: ... [Додати]	Камера 7 Параметри: - Кут: ... - Дальн: ... - Ціна: ... [Додати]	Камера 8 Параметри: - Кут: ... - Дальн: ... - Ціна: ... [Додати]

<< 1 2 3 4 5 ... >>	Показано 8 з 42
---------------------	-----------------

Рисунок 3.4 - Інтерфейс бібліотеки камер

Модуль результатів оптимізації є ключовим компонентом системи, який візуалізує результати роботи алгоритму оптимізації розміщення камер та дозволяє користувачеві аналізувати та коригувати отримані результати. Інтерфейс цього модуля включає візуалізацію плану приміщення з оптимальним розміщенням камер, карту покриття, панель статистики та інструменти для ручного коригування.

Візуалізація результатів оптимізації використовує кольорове кодування для відображення якості покриття різних ділянок приміщення: зелений колір означає оптимальне покриття (видимість з кількох камер з

високою якістю), жовтий - задовільне покриття (видимість з однієї камери або з низькою якістю), червоний - відсутність покриття ("сліпі зони"). Інтерфейс візуалізації результатів представлений на рисунку 3.5.

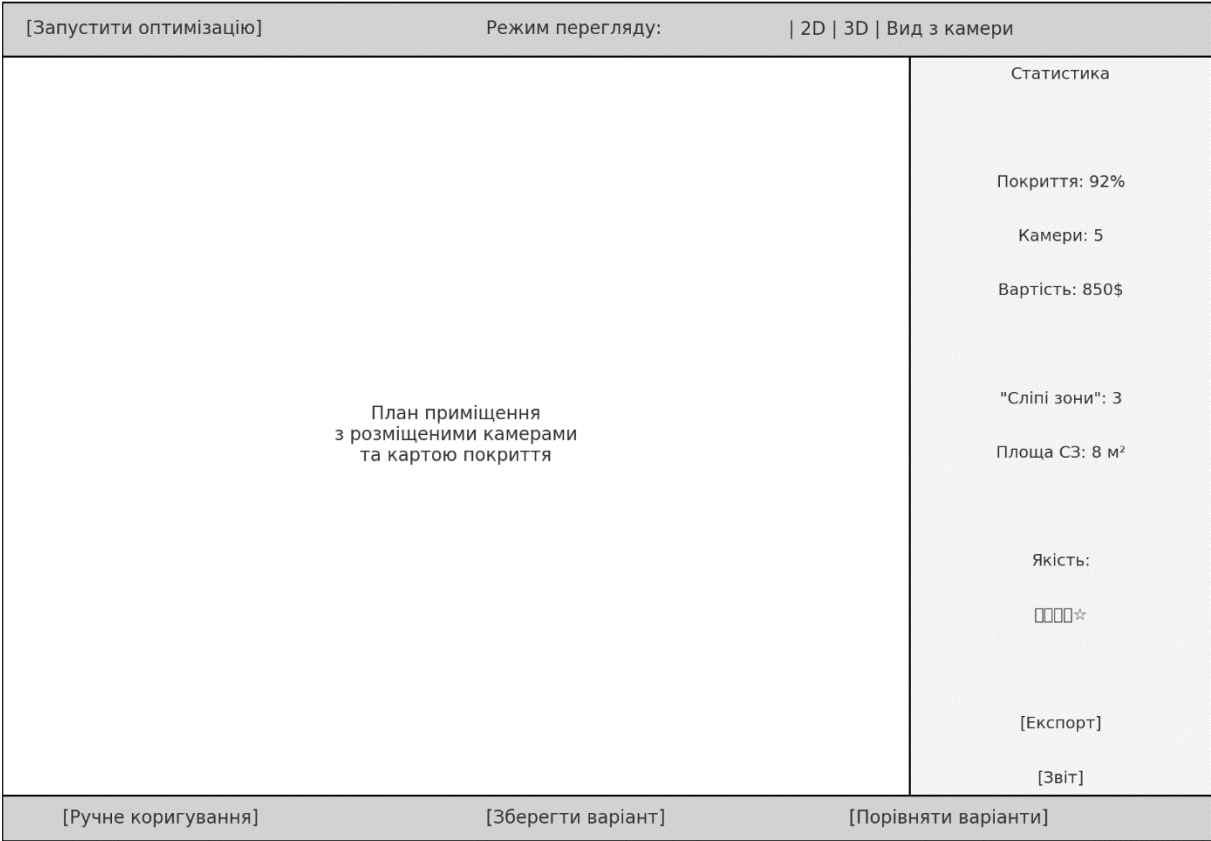


Рисунок 3.5 - Інтерфейс візуалізації результатів оптимізації

Для ручного коригування результатів оптимізації реалізований режим редагування, в якому користувач може переміщувати, додавати або видаляти камери, змінювати їх параметри та напрямок. При цьому система в реальному часі перераховує та відображає зміни в покритті приміщення, що дозволяє користувачеві миттєво бачити вплив своїх дій на результат.

Для порівняння різних варіантів розміщення камер реалізована функція збереження варіантів та їх порівняння у табличному вигляді за такими параметрами як покриття, кількість камер, вартість системи, кількість та площа "сліпих зон". Це дозволяє користувачеві вибрати оптимальний варіант, виходячи з своїх пріоритетів.

Для експорту результатів проєктування реалізований модуль експорту, який дозволяє зберігати результати у різних форматах: PDF-звіт з детальним описом системи, план розміщення камер у форматах PNG та SVG, таблиця з параметрами камер у форматах CSV та XLSX, а також спеціалізовані формати для інтеграції з іншими системами.

Для забезпечення зручності використання системи на різних пристроях реалізований адаптивний дизайн, який автоматично підлаштовується під розмір екрану. На мобільних пристроях інтерфейс спрощується, деякі елементи приховуються у випадаюче меню, а інші перегруповуються для забезпечення оптимального використання обмеженого простору екрану.

Для забезпечення зворотного зв'язку з користувачем реалізовано систему повідомлень та сповіщень: інформаційні повідомлення для сповіщення про успішне виконання операцій, попередження для звернення уваги на потенційні проблеми, та повідомлення про помилки з детальним описом проблеми та можливими шляхами її вирішення.

Для покращення користувацького досвіду реалізовано систему підказок та навчальних елементів: підказки при наведенні курсора на елементи інтерфейсу (tooltips), покрокові інструкції для складних операцій, інтерактивні навчальні матеріали для нових користувачів, а також контекстна довідка з детальним описом кожного елемента системи.

Для забезпечення швидкодії інтерфейсу використовуються такі техніки оптимізації як: віртуалізація списків для ефективного відображення великої кількості елементів, відкладене завантаження (lazy loading) для компонентів, які не відображаються на екрані, кешування даних для зменшення кількості запитів до сервера, ефективне управління станом додатку за допомогою Redux, оптимізація рендерингу компонентів за допомогою React.memo та useMemo.

Для забезпечення доступності інтерфейсу для користувачів з обмеженими можливостями реалізовані такі функції як: підтримка навігації

за допомогою клавіатури, сумісність з програмами читання з екрану, можливість збільшення розміру шрифту та елементів інтерфейсу, підтримка високої контрастності, альтернативний текст для зображень та інші рекомендації WCAG 2.1.

Таблиця 3.2 містить детальний опис основних екранів системи та їх ключових функцій.

Таблиця 3.2 - Основні екрани системи та їх функції

Екран	Ключові функції	Цільові користувачі
Головна сторінка	Інформація про систему; форми для входу та реєстрації; демонстраційне відео; відгуки користувачів; контактна інформація.	Нові та потенційні користувачі; відвідувачі сайту.
Сторінка входу	Автентифікація за допомогою логіна та пароля; відновлення пароля; авторизація через соціальні мережі; перевірка безпеки.	Зареєстровані користувачі, які хочуть отримати доступ до системи.
Сторінка реєстрації	Створення нового облікового запису; валідація введених даних; вибір типу облікового запису; прийняття умов використання.	Нові користувачі, які хочуть зареєструватися в системі.
Особистий кабінет	Перегляд та управління проектами; налаштування облікового запису; доступ до навчальних матеріалів; статистика використання системи.	Зареєстровані користувачі, які хочуть управляти своїми проектами.
Список проектів	Відображення списку проектів користувача; фільтрація, сортування та пошук проектів; створення, редагування та видалення проектів; доступ до архіву проектів.	Користувачі, які працюють з кількома проектами.
Налаштування користувача	Зміна особистих даних; налаштування мови та теми інтерфейсу; налаштування сповіщень; зміна пароля	Користувачі, які хочуть налаштувати систему під свої потреби.

Продовження таблиці 3.2

Екран	Ключові функції	Цільові користувачі
Редактор планів приміщень	Створення та редагування планів приміщень; імпорт планів з файлів; автоматичне розпізнавання стін та перешкод; масштабування та вирівнювання планів; додавання та редагування елементів приміщення.	Користувачі, які створюють або редагують плани приміщень.
Налаштування камер	Вибір та конфігурація камер відеоспостереження; створення користувацьких моделей камер; встановлення обмежень щодо кількості камер; вибір пріоритетів оптимізації.	Користувачі, які підбирають камери для своєї системи.

Для взаємодії з серверною частиною системи клієнтський інтерфейс використовує RESTful API, який забезпечує передачу даних у форматі JSON. Для операцій, що вимагають оновлення даних у реальному часі, таких як візуалізація результатів оптимізації, використовується WebSocket для забезпечення двосторонньої комунікації між клієнтом та сервером.

Для оцінки якості інтерфейсу користувача проведено серію тестувань з реальними користувачами, які представляють цільову аудиторію системи. За результатами тестування вдосконалено найбільш проблемні аспекти інтерфейсу, спрощено складні операції та покращено візуальне представлення інформації.

Таким чином, розроблений інтерфейс користувача для системи оптимізації розміщення камер відеоспостереження забезпечує інтуїтивно зрозумілу та ефективну взаємодію користувача з системою, дозволяє виконувати всі необхідні операції з мінімальними зусиллями та максимальною ефективністю, а також надає інструменти для аналізу та вдосконалення результатів оптимізації.

3.3 Реалізація алгоритму оптимізації розміщення камер

Реалізація алгоритму оптимізації розміщення камер відеоспостереження є центральним елементом розроблюваної системи, який визначає її ефективність та практичну цінність. Програмна реалізація алгоритму базується на математичній моделі, описаній у розділі 2.2, та включає реалізацію методів оптимізації, алгоритмів визначення зон покриття та механізми підвищення продуктивності.

Основою програмної реалізації алгоритму оптимізації є математична модель, яка формалізує задачу розміщення камер як оптимізаційну задачу з обмеженнями. Для реалізації цієї моделі використано об'єктно-орієнтований підхід, який дозволяє чітко розділити відповідальність між різними компонентами алгоритму та забезпечити гнучкість і розширюваність рішення. Основні класи, що реалізують математичну модель, представлені на рисунку 3.6.

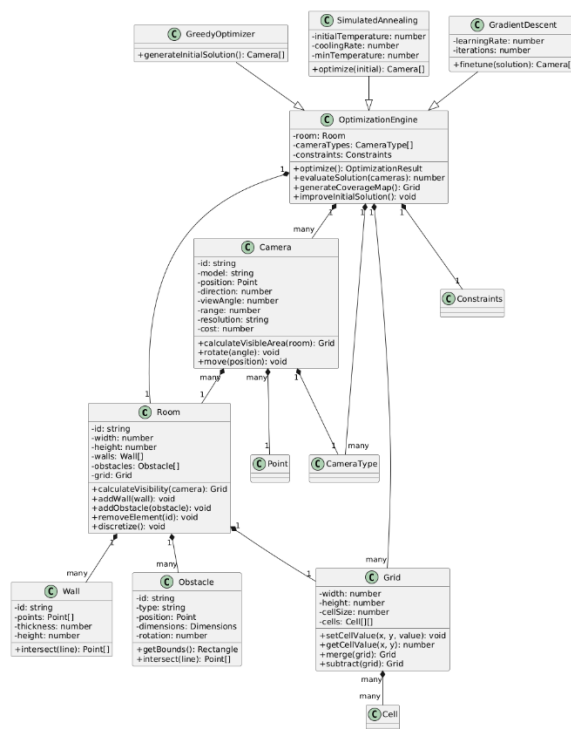


Рисунок 3.6 - Структура класів для реалізації алгоритму оптимізації

Для реалізації класів математичної моделі використано мову TypeScript, яка забезпечує статичну типізацію та об'єктно-орієнтовані можливості, що дозволяє виявляти потенційні помилки на етапі компіляції та покращити якість коду. Основні інтерфейси та типи даних представлені у лістингу Б.1.

Алгоритм оптимізації розміщення камер реалізовано як поєднання трьох основних методів: жадібного алгоритму для генерації початкового розв'язку, методу імітації відпалу для глобальної оптимізації та градієнтного спуску для локального покращення розв'язку. Така комбінація методів дозволяє отримати хороший баланс між якістю результату та обчислювальною складністю.

Реалізація жадібного алгоритму базується на ітеративному додаванні камер у позиції, які забезпечують максимальне приросте покриття. Цей підхід не гарантує глобально оптимального розв'язку, але дозволяє швидко отримати початкове наближення, яке потім можна покращити іншими методами. Реалізація жадібного алгоритму представлена у лістингу Б.2.

Після отримання початкового розв'язку за допомогою жадібного алгоритму, метод імітації відпалу використовується для глобальної оптимізації розміщення камер. Цей метод дозволяє уникнути локальних оптимумів за рахунок прийняття з певною ймовірністю рішень, що тимчасово погіршують цільову функцію. Реалізація методу імітації відпалу представлена у лістингу Б.3.

Для локального покращення розв'язку, отриманого методом імітації відпалу, використовується метод градієнтного спуску. Цей метод дозволяє точно позиціонувати камери, оптимізуючи їх розташування та напрямок на основі градієнта цільової функції. Реалізація методу градієнтного спуску представлена у лістингу Б.4.

Ключовим елементом алгоритму оптимізації є обчислення зон видимості камер та їх перетинів з перешкодами. Для ефективного обчислення зон видимості використовується алгоритм променевого кидання (ray casting).

Цей алгоритм моделює поширення променів світла від камери у різних напрямках і визначає перетини цих променів зі стінами та перешкодами. Реалізація алгоритму обчислення зон видимості представлена у лістингу Б.5.

Для оцінки якості розв'язку використовується комбінований критерій, який враховує площу покриття, якість покриття, кількість "сліпих зон", перекриття зон видимості та вартість системи. Реалізація класу для оцінки якості розв'язку використовується комбінований критерій, який враховує площу покриття, якість покриття, кількість "сліпих зон", перекриття зон видимості та вартість системи. Реалізація класу для оцінки якості розв'язку представлена у лістингу Б.6.

3.4 Висновки за розділом 3

У третьому розділі розроблена база даних на основі MongoDB забезпечує ефективне зберігання різномірної інформації про користувачів, проекти, приміщення, камери та результати оптимізації. Використання документоорієнтованого підходу забезпечує гнучкість схеми даних та можливість її адаптації до змінюваних вимог. Розроблені механізми валідації, індексування та кешування даних гарантують цілісність та високу продуктивність операцій з базою даних.

Проектування інтерфейсу користувача базується на принципах user-centered design, що забезпечує інтуїтивну зрозумілість та ефективність взаємодії. Розроблений інтерфейс надає користувачам зручні інструменти для редагування планів приміщень, налаштування параметрів камер, візуалізації результатів оптимізації та генерації звітів. Особлива увага приділена забезпеченню адаптивності інтерфейсу для різних розмірів екрану та доступності для користувачів з різними потребами.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Призначення програмного забезпечення

Розроблене програмне забезпечення призначене для автоматизації процесу проєктування систем відеоспостереження шляхом оптимізації розміщення камер у приміщеннях різного типу та складності. Основною метою програми є забезпечення максимального покриття приміщення при мінімальних витратах на обладнання та монтаж. Система орієнтована на широке коло користувачів, серед яких власники приватної нерухомості, які планують встановлення системи відеоспостереження власними силами; представники малого та середнього бізнесу, яким необхідно оптимізувати витрати на системи безпеки; спеціалісти з проєктування систем безпеки, які потребують інструменти для швидкого та ефективного проєктування; інтегратори систем безпеки, які надають послуги з встановлення систем відеоспостереження.

Функціональні можливості програмного забезпечення включають різноманітні компоненти та елементи взаємодії. Програмне забезпечення побудовано на мікросервісній архітектурі з чітким розподілом відповідальності між компонентами. Основні модулі системи включають модуль управління проєктами, що дозволяє створювати, редагувати та організовувати проєкти систем відеоспостереження; модуль роботи з планами приміщень, який забезпечує імпорт, редагування та масштабування планів різних форматів; модуль управління параметрами камер, що надає доступ до бібліотеки типових камер та можливість створювати користувацькі моделі; модуль оптимізації розміщення камер, що реалізує алгоритми автоматичного розміщення з урахуванням заданих обмежень; модуль візуалізації результатів, що відображає оптимальне розміщення камер та зони покриття; модуль експорту та звітності для генерації документації; модуль інтеграції з іншими системами. Взаємодія між модулями здійснюється через

API-шлюз, який забезпечує маршрутизацію запитів, автентифікацію та авторизацію користувачів.

Для оптимізації розміщення камер використовується комбінований алгоритм, що поєднує жадібний підхід для генерації початкового розміщення, метод імітації відпалу для глобальної оптимізації та градієнтний спуск для локального уточнення результатів. Алгоритм враховує геометрію приміщення, наявні перешкоди, характеристики камер та задані обмеження, такі як максимальна кількість камер, бюджет та мінімальне необхідне покриття.

Реалізація алгоритму базується на об'єктно-орієнтованому підході з використанням наступних основних класів: `OptimizationEngine`, що координує процес оптимізації; `Room`, що представляє приміщення з його геометрією та перешкодами; `Camera`, що моделює камери та їх параметри; `Grid`, що використовується для дискретизації простору та обчислення зон видимості.

Для зберігання даних використовується документоорієнтована база даних `MongoDB`, що забезпечує гнучкість схеми даних та можливість її адаптації до змінюваних вимог.

Інтерфейс користувача програмного забезпечення побудований на основі компонентного підходу та реалізований як одно-сторінковий додаток (SPA) з використанням `React` та `TypeScript`. Структура інтерфейсу включає головну сторінку з інформацією про систему та формою входу, сторінку проєктів з можливістю перегляду та управління проєктами, редактор планів приміщень, модуль конфігурації камер, модуль візуалізації результатів.

Редактор планів приміщень дозволяє користувачам створювати плани з нуля або імпортувати їх з різних форматів, додавати та редагувати стіни, двері, вікна та перешкоди, встановлювати масштаб та виконувати вимірювання. Модуль конфігурації камер надає доступ до каталогу типових камер та можливість налаштування їх параметрів. Модуль візуалізації

результатів відображає оптимальне розміщення камер, зони покриття та "сліпі зони" з використанням кольорового кодування.

Інтеграція з іншими системами забезпечується через API, реалізований за принципами REST. Основні ендпоінти включають методи для автентифікації та реєстрації користувачів, управління проєктами, роботи з планами приміщень, запуску оптимізації та отримання результатів (табл. 4.1).

Таблиця 4.1 - Основні ендпоінти API системи

Ендпоінт	Метод	Опис	Параметри запити	Відповідь
/api/auth/login	POST	Автентифікація користувача та отримання токена доступу	email, password	token, userId, expiresIn
/api/auth/register	POST	Реєстрація нового користувача	username, email, password	userId, status
/api/projects	GET	Отримання списку проєктів користувача	page, limit, sortBy, order, filter	projects[], totalCount, page, limit
/api/projects	POST	Створення нового проєкту	name, description, tags[]	projectId, status
/api/projects/{id}	GET	Отримання деталей проєкту	id	project
/api/optimization/run	POST	Запуск процесу оптимізації розміщення камер	roomId, constraints, algorithm	jobId, status
/api/optimization/results/{jobId}	GET	Отримання результатів	jobId	result

4.2 Умови виконання програмного забезпечення

Для забезпечення ефективної та стабільної роботи програмного забезпечення оптимізації розміщення камер відеоспостереження необхідно дотримуватись певних вимог до середовища виконання. Ці вимоги охоплюють апаратне та програмне забезпечення, мережеву інфраструктуру та систему безпеки.

Для початку роботи з системою користувачу необхідно пройти автентифікацію через екран входу, який забезпечує безпечний доступ до функціоналу програми. Інтерфейс входу розроблено з урахуванням сучасних принципів UX-дизайну та містить поля для введення облікових даних, а також можливість відновлення паролю (рис. 4.1).

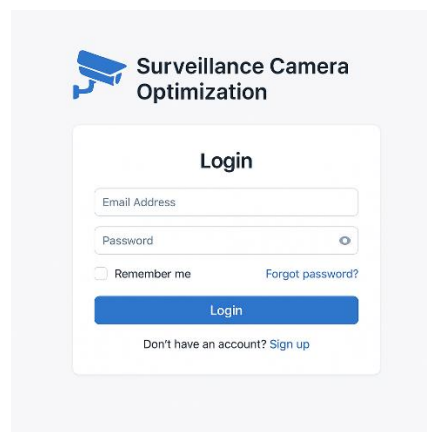


Рисунок 4.1 - Екран входу користувача до системи

Після успішної автентифікації користувач отримує доступ до панелі управління проєктами, яка є центральним хабом для навігації між різними проєктами систем відеоспостереження. На цьому екрані представлено список існуючих проєктів з можливістю фільтрації, пошуку та створення нових проєктів (рис. 4.2).

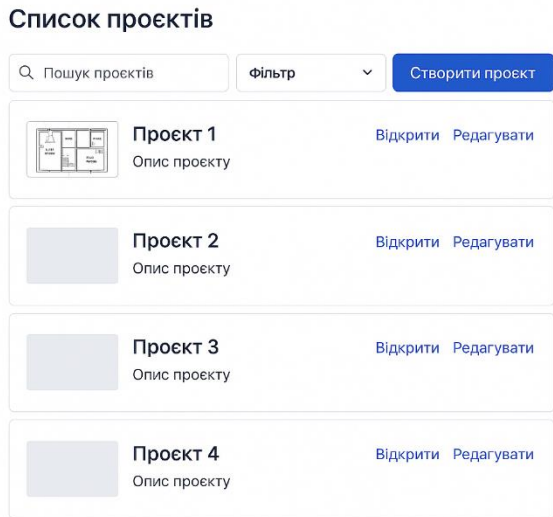


Рисунок 4.2 - Панель управління проектами системи відеоспостереження

Одним з ключових компонентів системи є інтерфейс завантаження та редагування планів приміщень, який дозволяє користувачам імпортувати плани у різних форматах (PNG, JPG, PDF, SVG), масштабувати їх та редагувати, додаючи або видаляючи елементи приміщення, такі як стіни, двері та перешкоди (рис. 4.3).

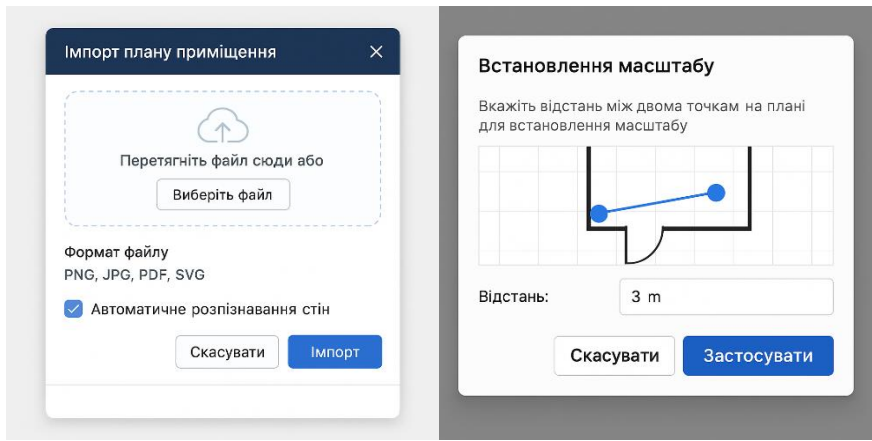


Рисунок 4.3 - Інтерфейс завантаження та редагування планів приміщень

Для налаштування параметрів камер відеоспостереження розроблено спеціальний модуль, який надає доступ до бібліотеки типових камер з можливістю фільтрації за різними характеристиками, а також створення та

налаштування власних моделей камер з урахуванням їх технічних параметрів (рис. 4.4).

Додавання камери

Назва камери

Модель

Датчики

Кут огляду ° Живлення В

Роздільна здатність ° Ціна

Додати

Налаштування камер

Назва	Кут огляду	Дальність	Роздільна здатність
Камера 1	90°	20 м	4 MP
Камера 2	60°	15 м	5 MP
Камера 3	110°	10 м	3 MP
Камера 4	100°	30 м	6 MP
Камера 5	70°	25 м	4 MP
Камера 6	120°	40 м	5 MP

Пріоритети оптимізації

☐ Максимальне покриття при фіксованій кількості камер

☐ Мінімальна кількість камер для забезпечення заданого рівня покриття

☐ Баланс між покриттям та вартістю системи

Рисунок 4.4 - Модуль налаштування параметрів камер відеоспостереження

Центральним елементом системи є візуалізація покриття камер на плані приміщення, яка в реальному часі демонструє зони видимості кожної камери та їх перекриття (рис. 4.5).

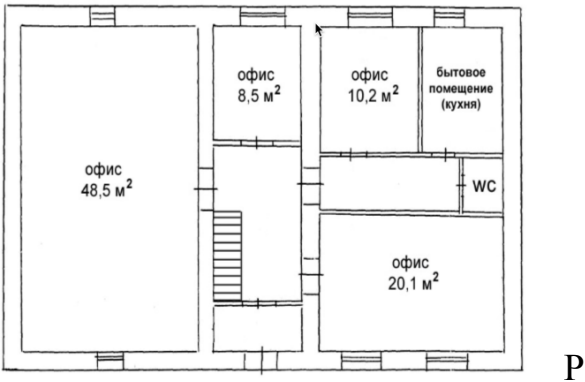


Рисунок 4.5 – Приклад плану для завантаження

Після запуску алгоритму оптимізації система відображає результати у вигляді оптимального розміщення камер на плані приміщення. Користувач має можливість переглянути детальну інформацію про кожну камеру, її зону покриття та внесок у загальне покриття приміщення (рис. 4.6).

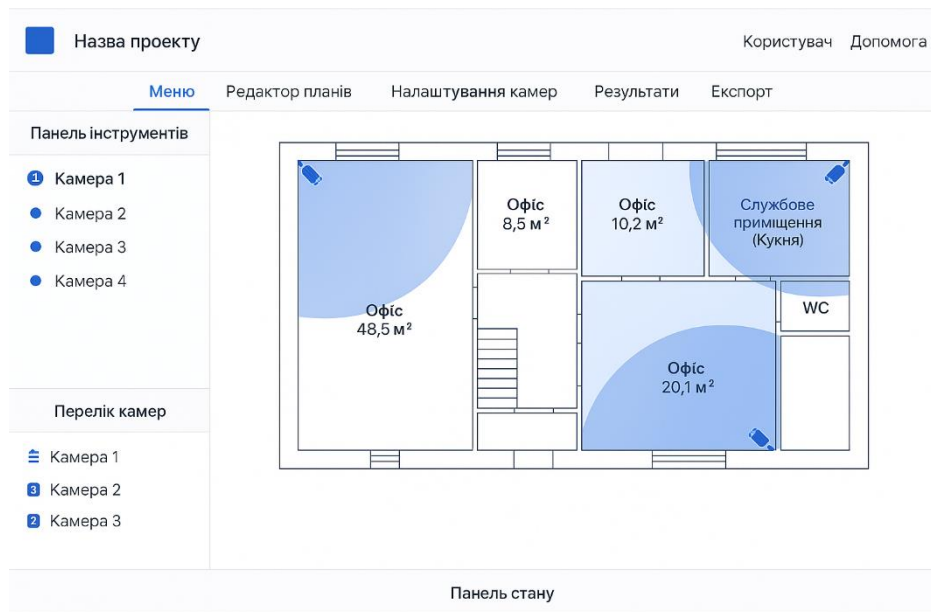


Рисунок 4.6 - Результати оптимізації розміщення камер

Для забезпечення ефективної та стабільної роботи програмного забезпечення оптимізації розміщення камер відеоспостереження необхідно дотримуватись певних вимог до середовища виконання. Ці вимоги охоплюють апаратне та програмне забезпечення, мережеву інфраструктуру та систему безпеки.

Апаратні вимоги до серверної частини системи включають наступні параметри: процесор з не менше ніж 4 ядрами та тактовою частотою 2.4 ГГц або вище; оперативна пам'ять обсягом не менше 16 Гб для забезпечення обробки складних алгоритмів оптимізації та одночасного обслуговування багатьох користувачів; дисковий простір обсягом не менше 100 Гб для зберігання бази даних, файлів планів приміщень та іншої необхідної інформації; мережева карта з пропускнуою здатністю не менше 1 Гбіт/с для забезпечення швидкої передачі даних. Для підвищення продуктивності та надійності рекомендовано використання серверів з SSD-накопичувачами для зберігання бази даних та інших даних, що часто використовуються.

Клієнтська частина системи має менш суворі вимоги до апаратного забезпечення: будь-який сучасний комп'ютер, ноутбук, планшет або смартфон з підтримкою сучасних веб-браузерів; мінімально рекомендована

роздільна здатність екрану – 1280x720 пікселів, оптимальна – 1920x1080 або вище. Оптимальний досвід використання системи забезпечується на пристроях з діагоналлю екрану не менше 10 дюймів.

Програмне забезпечення серверної частини системи має відповідати таким вимогам: операційна система – Linux (рекомендовано Ubuntu Server 20.04 LTS або новіше), Windows Server 2019 або новіше; Node.js версії 16.0 або новіше; MongoDB версії 5.0 або новіше; RabbitMQ версії 3.9 або новіше; Nginx або Apache для роботи як проксі-сервер.

Для клієнтської частини необхідно використовувати сучасні веб-браузери з підтримкою HTML5, CSS3, ES6 та WebGL: Google Chrome 90 або новіше; Mozilla Firefox 88 або новіше; Microsoft Edge 90 або новіше; Safari 14 або новіше. Браузери Internet Explorer та старі версії інших браузерів не підтримуються (табл. 4.2).

Таблиця 4.2 - Вимоги до програмного забезпечення серверної частини

Компонент	Мінімальна версія	Рекомендована версія	Примітки
Операційна система	Ubuntu Server 18.04, Windows Server 2016	Ubuntu Server 22.04, Windows Server 2022	Підтримуються
Node.js	14.0	18.0	Потрібен npm 6.0 або новіше
MongoDB	4.4	6.0	Рекомендовано
RabbitMQ	3.8	3.11	Потрібен erlang 23.0 або новіше
Nginx	1.18	1.22	Можна замінити на Apache
Redis	6.0	7.0	Використовується для кешування

Мережеві вимоги для нормальної роботи системи включають наступні параметри: для серверної частини – стабільне підключення до інтернету з пропускнуою здатністю не менше 100 Мбіт/с; для клієнтської частини – стабільне підключення з пропускнуою здатністю не менше 5 Мбіт/с. Рекомендовано використання статичної IP-адреси для серверної частини системи та налаштування DNS-запису для зручного доступу до системи. Для забезпечення безпеки даних весь обмін інформацією між клієнтською та серверною частинами повинен здійснюватися через шифроване з'єднання HTTPS з використанням SSL/TLS сертифікатів.

Для забезпечення захисту даних система використовує наступні механізми: аутентифікація користувачів з використанням JWT (JSON Web Tokens); шифрування паролів користувачів з використанням алгоритму bcrypt; захист від XSS та CSRF атак; валідація та санітизація всіх даних, що вводяться користувачем; логування всіх критичних операцій та спроб несанкціонованого доступу. Для додаткового захисту рекомендовано налаштування брандмауера для обмеження доступу до серверної частини системи, використання системи виявлення та запобігання вторгнень, регулярне оновлення програмного забезпечення та створення резервних копій бази даних (табл. 4.3).

Для роботи з великими обсягами даних (наприклад, при імпорті планів приміщень великого розміру або при оптимізації розміщення камер для складних приміщень) рекомендовано збільшення часових лімітів для HTTP-запитів та збільшення максимально допустимого розміру файлів, що завантажуються. Також рекомендовано налаштування пулів з'єднань до бази даних відповідно до очікуваного навантаження на систему.

Для забезпечення високої доступності та відмовостійкості системи рекомендовано використання кластерного режиму для Node.js додатків, реплікації даних MongoDB та налаштування балансувальника навантаження. Для забезпечення масштабованості системи при зростанні навантаження

можливе горизонтальне масштабування компонентів системи шляхом додавання нових серверів та розподілу навантаження між ними.

Таблиця 4.3 - Механізми захисту даних в системі

Тип захисту	Механізм	Опис
Аутентифікація	JWT	Використання токенів для авторизації запитів
Шифрування паролів	bcrypt	Одностороннє шифрування з сіллю
Захист від XSS	Санітизація даних	Видалення потенційно небезпечних HTML-тегів
Захист від CSRF	CSRF-токени	Валідація токенів для запитів, що змінюють дані
Контроль доступу	RBAC	Рольове розмежування доступу до ресурсів
Мережевий захист	HTTPS	Шифрування всього трафіку між клієнтом і сервером

4.3 Виконання і тестування програмного забезпечення

Тестування розробленого програмного забезпечення для оптимізації розміщення камер відеоспостереження проводилось з метою перевірки його функціональності, продуктивності та зручності використання. Для проведення тестів було розроблено спеціальні тестові сценарії та використано реальні плани приміщень різної складності.

Для тестування було використано набір з 20 планів приміщень, що включав: невеликі приватні приміщення (квартири, будинки); комерційні об'єкти середнього розміру (офіси, магазини); великі промислові приміщення (склади, виробничі цехи). Тестування проводилось на комп'ютері з

процесором Intel Core i7-10700, 16 Гб оперативної пам'яті під управлінням операційної системи Windows 10 Pro.

Функціональне тестування включало перевірку всіх основних функцій системи: завантаження та редагування планів приміщень; налаштування параметрів камер; запуск алгоритму оптимізації; візуалізацію результатів. Всі функції були перевірені на коректність роботи з використанням різних вхідних даних (табл. 4.4).

Таблиця 4.4 - Результати функціонального тестування

Функція	Результат тестування	Примітки
Імпорт планів приміщень	Успішно	Підтримка форматів PNG, JPG, PDF, SVG
Редагування елементів приміщення	Успішно	Додавання/видалення стін, дверей, вікон
Налаштування параметрів камер	Успішно	Робота з бібліотекою типових камер
Запуск оптимізації	Успішно	Коректна робота з усіма тестовими планами
Візуалізація результатів	Успішно	2D режими перегляду

4.4 Висновки за розділом 4

У четвертому розділі було проведено всебічне дослідження експлуатаційних характеристик та тестування розробленого програмного забезпечення для системи відеоспостереження. Розділ охоплює призначення програми, умови її виконання та результати тестування.

Визначено основне призначення програмного продукту - автоматизацію процесу проектування систем відеоспостереження через оптимізацію розміщення камер у приміщеннях різного типу та складності.

Програмне забезпечення орієнтоване на широке коло користувачів: від власників приватної нерухомості до професійних інтеграторів систем безпеки.

Детально описано умови виконання програми, включаючи апаратні та програмні вимоги для серверної та клієнтської частин. Визначено, що для стабільної роботи серверної частини необхідні: процесор з не менше 4 ядрами, оперативна пам'ять від 16 Гб, дисковий простір від 100 Гб, сучасні версії Node.js, MongoDB та інших компонентів. Для клієнтської частини встановлено мінімальні вимоги до пристроїв та перелік підтримуваних браузерів.

Тестування програмного забезпечення проводилося на наборі з 20 різноманітних планів приміщень - від невеликих приватних приміщень до великих промислових об'єктів. Результати функціонального тестування підтвердили коректність роботи всіх ключових функцій системи: імпорту та редагування планів приміщень, налаштування параметрів камер, запуску алгоритму оптимізації та візуалізації результатів.

Проведене тестування підтвердило відповідність розробленого програмного забезпечення поставленим вимогам та його готовність до практичного використання для проєктування ефективних систем відеоспостереження.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було успішно розроблено програмне забезпечення для системи відеоспостереження, яке дозволяє оптимізувати розміщення камер у приміщеннях різного типу та призначення.

Проведений у першому розділі аналіз предметної області виявив зростаючу потребу в інтелектуальних системах відеоспостереження та відсутність на ринку доступних програмних рішень для оптимального розміщення камер. Визначено, що неефективне проектування систем часто призводить до появи "сліпих зон" або надмірних витрат на обладнання через перекриття зон спостереження.

На етапі проектування програмного забезпечення було розроблено багаторівневу мікросервісну архітектуру, яка забезпечує гнучкість, масштабованість та високу продуктивність системи. Створено комбінований алгоритм оптимізації розміщення камер, що поєднує жадібний алгоритм для генерації початкового розміщення, метод імітації відпалу для глобальної оптимізації та градієнтний спуск для фінального уточнення результатів.

Програмна реалізація системи включає документоорієнтовану базу даних MongoDB для ефективного зберігання різномірної інформації та інтуїтивно зрозумілий вебінтерфейс, розроблений з використанням React і TypeScript. Особлива увага була приділена алгоритму обчислення зон видимості камер з використанням методу променевого кидання, що забезпечує точне моделювання покриття з урахуванням перешкод.

Проведене тестування підтвердило коректність роботи системи на різних типах приміщень та її відповідність вимогам користувачів. Результати тестування показали, що використання розробленого програмного забезпечення дозволяє досягти оптимального покриття приміщення при мінімальних витратах на обладнання.

Таким чином, розроблене програмне забезпечення вирішує актуальну проблему оптимізації розміщення камер відеоспостереження та може бути ефективно використане як індивідуальними користувачами для проєктування домашніх систем відеоспостереження, так і професіоналами для створення складних систем безпеки на комерційних об'єктах.

Перспективними напрямками подальшого розвитку розробленого програмного забезпечення є: удосконалення алгоритмів оптимізації для роботи з 3D-моделями приміщень, інтеграція з системами розумного дому та платформами управління відеоспостереженням, а також впровадження технологій штучного інтелекту для аналізу якості покриття та рекомендацій щодо розміщення камер.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Szeliski, R. Computer Vision: Algorithms and Applications / R. Szeliski. – London: Springer, 2022. – 812 p.
2. Goodfellow, I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge: MIT Press, 2016. – 775 p.
3. Redmon, J. YOLO9000: Better, Faster, Stronger / J. Redmon, A. Farhadi // IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2017. – P. 7263–7271.
4. Russakovsky, O. ImageNet Large Scale Visual Recognition Challenge / O. Russakovsky et al. // International Journal of Computer Vision. – 2015. – Vol. 115(3). – P. 211–252.
5. Ng, A. Machine Learning Yearning / A. Ng. – Stanford: DeepLearning.ai, 2018. – 111 p.
6. Melin, P. Analysis and Design of Intelligent Systems Using Soft Computing Techniques / P. Melin, O. Castillo, J. Kacprzyk. – Heidelberg: Springer, 2007. – 855 p.
7. Hansen, M. H. Sample Survey Methods and Theory / M. H. Hansen, W. N. Hurwitz, W. G. Madow. – New York: John Wiley & Sons, 1953. – Vol. 1. – 638 p.
8. Blanke, M. Diagnosis and Fault-Tolerant Control / M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki. – Berlin: Springer, 2016. – 672 p.
9. Buckley, J. J. Fuzzy Neural Networks: A Survey / J. J. Buckley, Y. Hayashi // Fuzzy Sets and Systems. – 1994. – Vol. 66. – P. 1–13.
10. Shoureshi, R. Intelligent Transformer Monitoring System Utilizing Neuro-Fuzzy Technique Approach: Final Project Report 04-26 / R. Shoureshi, T. Norick. – Golden: Colorado School of Mines, 2004. – 111 p.
11. Олійник, А. О. Еволюційні обчислення та програмування: навчальний посібник / А. О. Олійник, С. О. Субботін. – Запоріжжя: ЗНТУ, 2010. – 324 с.

12. Литвин, О. Архітектура програмного забезпечення: теорія і практика / О. Литвин, В. Кухта. – Львів: ЛНУ ім. І. Франка, 2023. – 292 с.
13. Шевченко, А. Системи технічного зору на основі нейромереж: навч. посібник / А. Шевченко. – Київ: КПІ ім. І. Сікорського, 2021. – 268 с.
14. Козлов, І. Основи проєктування інформаційних систем: навч. посібник / І. Козлов. – Харків: ХНУРЕ, 2020. – 204 с.
15. Іванченко, П. Системи відеоспостереження: від аналогових до IP-технологій / П. Іванченко. – Київ: ДІА, 2020. – 156 с.
16. Барабаш, В. Сучасні технології інтернету речей: навчальний посібник / В. Барабаш. – Львів: Новий Світ – 2000, 2022. – 216 с.
17. Zadeh, L. A. Fuzzy Sets / L. A. Zadeh // Information and Control. – 1965. – Vol. 8(3). – P. 338–353.
18. O'Reilly, T. Designing Distributed Systems / B. Burns. – Sebastopol: O'Reilly Media, 2018. – 180 p.
19. Sommerville, I. Software Engineering / I. Sommerville. – 10th ed. – Boston: Pearson, 2016. – 800 p.
20. Sarker, I. H. Machine Learning: Algorithms, Real-World Applications and Research Directions / I. H. Sarker // SN Computer Science. – 2021. – Vol. 2(3). – Article 160.
21. Охріменко, Г. Інтернет речей у розумних містах / Г. Охріменко. – Одеса: ОНПУ, 2021. – 128 с.
22. Luger, G. F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving / G. F. Luger. – 6th ed. – Boston: Pearson, 2008. – 784 p.
23. Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks / A. Karpathy. – 2015. – [Online]. Available: <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>.
24. Волкова, І. Оптимізація інформаційних систем: методичні підходи / І. Волкова. – Дніпро: ДНУ, 2022. – 184 с.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для оптимізації розміщення камер відеоспостереження у приміщеннях різного типу.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для системи відеоспостереження», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для автоматизації процесу проєктування систем відеоспостереження шляхом оптимізації розміщення камер у приміщеннях.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для системи відеоспостереження забезпечує виконання таких функцій:

- можливість завантаження та редагування планів приміщень у різних форматах (PNG, JPG, PDF, SVG);
- автоматичне розпізнавання стін та перешкод на імпортованих планах;
- можливість вибору та налаштування параметрів камер відеоспостереження;
- автоматична оптимізація розміщення камер з урахуванням геометрії приміщення та перешкод;

- візуалізація зон покриття та "сліпих зон" на плані приміщення;
- збереження та управління проєктами систем відеоспостереження.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для системи відеоспостереження повинен бути зручним для користувачів різного рівня підготовки. Повинна бути забезпечена можливість роботи в візуальному режимі з інтуїтивно зрозумілими елементами керування.

А.3.3 Вимоги до надійності

Програмне забезпечення для системи відеоспостереження повинно забезпечити надійне функціонування з коректною обробкою помилок введення користувача та неправильних форматів даних.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для системи відеоспостереження необхідна наявність комп'ютера з підключенням до мережі інтернет та сучасним веббраузером.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи програмного забезпечення для системи відеоспостереження потрібні такі технічні та програмні ресурси, як сервер з процесором не менше 4 ядер, оперативною пам'яттю не менше 16 Гб, дисковим простором не менше 100 Гб, та клієнтський пристрій з сучасним браузером (Chrome, Firefox, Safari, Edge останніх версій).

A.3.6 Вимоги до маркування і пакування

Програма для системи відеоспостереження може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми - «Програмне забезпечення для системи відеоспостереження».

A.4 Вхідні та вихідні дані

Вхідними даними до програмного забезпечення для системи відеоспостереження є плани приміщень у форматах PNG, JPG, PDF, SVG, параметри камер відеоспостереження, обмеження щодо кількості камер та бюджету.

Вихідними даними програмного забезпечення для системи відеоспостереження є схеми оптимального розміщення камер, візуалізація зон покриття, звіти про параметри системи та її ефективність.

ДОДАТОК Б
Фрагмент тексту програми

Лістинг Б.1 - Основні інтерфейси та типи даних для реалізації алгоритму оптимізації

```
// Типи даних та інтерфейси
interface Point {
  x: number;
  y: number;
}

interface Dimensions {
  width: number;
  height: number;
  depth?: number;
}

interface Rectangle {
  x: number;
  y: number;
  width: number;
  height: number;
}

interface Cell {
  isCovered: boolean;
  quality: number;
  visibleFrom: string[]; // ID камер, які бачать цю клітинку
  importance: number;    // Важливість цієї клітинки (для
                          // пріоритетних зон)
}

interface CameraType {
  id: string;
  model: string;
  viewAngle: number;
  range: number;
  resolution: string;
  cost: number;
}

interface Constraints {
  maxCameras: number;
  maxBudget: number;
  minCoverage: number;
  priorityAreas: PriorityArea[];
}

interface PriorityArea {
  id: string;
  points: Point[];
  importance: number;
}

interface OptimizationResult {
```

```

cameras: Camera[];
coveragePercentage: number;
qualityScore: number;
cost: number;
coverageMap: Grid;
blindSpots: Point[];
}

// Параметри оптимізації
interface SimulatedAnnealingParams {
  initialTemperature: number;
  coolingRate: number;
  minTemperature: number;
  iterationsPerTemperature: number;
}

interface GradientDescentParams {
  learningRate: number;
  iterations: number;
  momentum: number;
  tolerance: number;
}

```

Лістинг Б.2 - Реалізація жадібного алгоритму для генерації початкового розв'язку

```

class GreedyOptimizer {
  private room: Room;
  private cameraTypes: CameraType[];
  private constraints: Constraints;
  private grid: Grid;

  constructor(room: Room, cameraTypes: CameraType[],
constraints: Constraints) {
    this.room = room;
    this.cameraTypes = cameraTypes;
    this.constraints = constraints;
    this.grid = room.grid;
  }

  public generateInitialSolution(): Camera[] {
    const cameras: Camera[] = [];
    let uncoveredGrid = this.grid.clone();
    let totalCost = 0;

    // Поки не досягнуто максимальної кількості камер або
    // бюджету
    while (
      cameras.length < this.constraints.maxCameras &&
      totalCost < this.constraints.maxBudget &&
      uncoveredGrid.getCoveragePercentage() <
this.constraints.minCoverage
    ) {

```

```

let bestCamera: Camera | null = null;
let bestCoverage = 0;

// Перебір усіх можливих позицій камер та їх типів
for (const cameraType of this.cameraTypes) {
    // Якщо додавання цієї камери перевищить бюджет,
пропускаємо
    if (totalCost + cameraType.cost >
this.constraints.maxBudget) {
        continue;
    }

    // Дискретизуємо простір для пошуку позицій
    const potentialPositions = this.getPotentialPositions();

    for (const position of potentialPositions) {
45) {
        for (let direction = 0; direction < 360; direction +=

        const camera = new Camera(
            `cam_${cameras.length + 1}`,
            cameraType.model,
            position,
            direction,
            cameraType.viewAngle,
            cameraType.range,
            cameraType.resolution,
            cameraType.cost
        );

        // Обчислюємо зону видимості цієї камери
        const visibilityGrid =
camera.calculateVisibleArea(this.room);

        // Обчислюємо покриття непокритої області
        const newCoverage =
uncoveredGrid.calculateOverlap(visibilityGrid);

        // Якщо це покриття краще, зберігаємо цю камеру
        if (newCoverage > bestCoverage) {
            bestCoverage = newCoverage;
            bestCamera = camera;
        }
    }
}

// Якщо знайдено найкращу камеру, додаємо її до розв'язку
if (bestCamera && bestCoverage > 0) {
    cameras.push(bestCamera);
    totalCost += bestCamera.cost;

    // Оновлюємо непокриту область

```

```

        const visibilityGrid =
bestCamera.calculateVisibleArea(this.room);
        uncoveredGrid = uncoveredGrid.subtract(visibilityGrid);
    } else {
        // Якщо не знайдено камеру, яка покращує рішення,
виходимо з циклу
        break;
    }
}

return cameras;
}

private getPotentialPositions(): Point[] {
    // Реалізація генерації потенційних позицій для камер
    // Може використовувати рівномірну сітку або більш складну
стратегію
    const positions: Point[] = [];
    const step = this.room.grid.cellSize * 5; // Крок сітки для
розміщення камер

    for (let x = 0; x < this.room.width; x += step) {
        for (let y = 0; y < this.room.height; y += step) {
            // Перевіряємо, чи точка знаходиться всередині
приміщення і не всередині перешкоди
            if (this.room.isInsideRoom({x, y}) &&
!this.room.isInsideObstacle({x, y})) {
                positions.push({x, y});
            }
        }
    }

    // Додаємо кути приміщення та інші стратегічні точки
    this.addStrategicPoints(positions);

    return positions;
}

private addStrategicPoints(positions: Point[]): void {
    // Додавання кутів приміщення
    for (const wall of this.room.walls) {
        for (const point of wall.points) {
            if (this.room.isInsideRoom(point) &&
!this.room.isInsideObstacle(point)) {
                positions.push(point);
            }
        }
    }
}
}

```

Лістинг Б.3 - Реалізація методу імітації відпалу для глобальної оптимізації

```

class SimulatedAnnealing {
    private room: Room;
    private constraints: Constraints;
    private params: SimulatedAnnealingParams;
    private evaluator: SolutionEvaluator;

    constructor(
        room: Room,
        constraints: Constraints,
        params: SimulatedAnnealingParams,
        evaluator: SolutionEvaluator
    ) {
        this.room = room;
        this.constraints = constraints;
        this.params = params;
        this.evaluator = evaluator;
    }

    public optimize(initialSolution: Camera[]): Camera[] {
        let currentSolution = this.cloneCameras(initialSolution);
        let bestSolution = this.cloneCameras(currentSolution);

        let currentEnergy =
this.evaluator.evaluate(currentSolution);
        let bestEnergy = currentEnergy;

        let temperature = this.params.initialTemperature;

        // Основний цикл імітації відпалу
        while (temperature > this.params.minTemperature) {
            // Кілька ітерацій на поточній температурі
            for (let i = 0; i < this.params.iterationsPerTemperature;
i++) {
                // Генеруємо сусідній розв'язок
                const neighborSolution =
this.generateNeighbor(currentSolution);
                const neighborEnergy =
this.evaluator.evaluate(neighborSolution);

                // Обчислюємо зміну енергії
                const deltaEnergy = neighborEnergy - currentEnergy;

                // Приймаємо новий розв'язок, якщо він кращий або з
певною ймовірністю
                if (deltaEnergy > 0 || Math.random() <
Math.exp(deltaEnergy / temperature)) {
                    currentSolution = neighborSolution;
                    currentEnergy = neighborEnergy;

                    // Оновлюємо найкращий розв'язок, якщо поточний кращий

```

```

        if (currentEnergy > bestEnergy) {
            bestSolution = this.cloneCameras(currentSolution);
            bestEnergy = currentEnergy;
        }
    }
}

// Зменшуємо температуру
temperature *= this.params.coolingRate;
}

return bestSolution;
}

private generateNeighbor(solution: Camera[]): Camera[] {
    const neighbor = this.cloneCameras(solution);

    // Обираємо випадкову операцію модифікації
    const operation = Math.floor(Math.random() * 3);

    switch (operation) {
        case 0:
            // Переміщення випадкової камери
            this.moveRandomCamera(neighbor);
            break;
        case 1:
            // Обертання випадкової камери
            this.rotateRandomCamera(neighbor);
            break;
        case 2:
            // Заміна типу випадкової камери
            this.changeRandomCameraType(neighbor);
            break;
    }

    return neighbor;
}

private moveRandomCamera(cameras: Camera[]): void {
    if (cameras.length === 0) return;

    const index = Math.floor(Math.random() * cameras.length);
    const camera = cameras[index];

    // Генеруємо випадкове зміщення
    const deltaX = (Math.random() - 0.5) * 2 *
this.room.grid.cellSize * 5;
    const deltaY = (Math.random() - 0.5) * 2 *
this.room.grid.cellSize * 5;

    const newPosition: Point = {
        x: camera.position.x + deltaX,
        y: camera.position.y + deltaY
    }
}

```

```

};

// Перевіряємо, чи нова позиція допустима
if (this.room.isInsideRoom(newPosition)           &&
!this.room.isInsideObstacle(newPosition)) {
    camera.move(newPosition);
}
}

private rotateRandomCamera(cameras: Camera[]): void {
    if (cameras.length === 0) return;

    const index = Math.floor(Math.random() * cameras.length);
    const camera = cameras[index];

    // Генеруємо випадковий кут повороту
    const deltaAngle = (Math.random() - 0.5) * 2 * 45; // ±45
    градусів

    camera.rotate(deltaAngle);
}

private changeRandomCameraType(cameras: Camera[]): void {
    if (cameras.length === 0) return;

    const index = Math.floor(Math.random() * cameras.length);
    const camera = cameras[index];

    // Отримуємо доступні типи камер
    const availableCameraTypes =
this.getCompatibleCameraTypes();

    if (availableCameraTypes.length === 0) return;

    // Вибираємо випадковий тип камери
    const randomType =
availableCameraTypes[Math.floor(Math.random()
availableCameraTypes.length)];

    // Оновлюємо параметри камери
    camera.updateParameters(
        randomType.model,
        randomType.viewAngle,
        randomType.range,
        randomType.resolution,
        randomType.cost
    );
}

private getCompatibleCameraTypes(): CameraType[] {
    // Повертає типи камер, які сумісні з обмеженнями
    // Наприклад, за бюджетом та іншими параметрами
    return []; // Реалізація залежить від контексту
}

```

```

    }

    private cloneCameras(cameras: Camera[]): Camera[] {
        return cameras.map(camera => camera.clone());
    }
}

```

Лістинг Б.4 - Реалізація методу градієнтного спуску для локального покращення розв'язку

```

class GradientDescent {
    private room: Room;
    private params: GradientDescentParams;
    private evaluator: SolutionEvaluator;

    constructor(
        room: Room,
        params: GradientDescentParams,
        evaluator: SolutionEvaluator
    ) {
        this.room = room;
        this.params = params;
        this.evaluator = evaluator;
    }

    public finetune(solution: Camera[]): Camera[] {
        let currentSolution = this.cloneCameras(solution);
        let bestSolution = this.cloneCameras(currentSolution);

        let currentEnergy =
this.evaluator.evaluate(currentSolution);
        let bestEnergy = currentEnergy;

        const velocities: { position: Point; direction: number }[] =
currentSolution.map(() => ({
            position: { x: 0, y: 0 },
            direction: 0
        }));

        // Основний цикл градієнтного спуску
        for (let iteration = 0; iteration < this.params.iterations;
iteration++) {
            let improved = false;

            // Обчислюємо градієнт для кожної камери
            for (let i = 0; i < currentSolution.length; i++) {
                const camera = currentSolution[i];

                // Обчислюємо градієнт по позиції
                const gradPosition =
this.calculatePositionGradient(camera, currentSolution);

                // Обчислюємо градієнт по напрямку

```

```

        const gradDirection =
this.calculateDirectionGradient(camera, currentSolution);

        // Оновлюємо швидкість (з урахуванням моменту)
        velocities[i].position.x = this.params.momentum *
velocities[i].position.x +
                                this.params.learningRate *
gradPosition.x;
        velocities[i].position.y = this.params.momentum *
velocities[i].position.y +
                                this.params.learningRate *
gradPosition.y;
        velocities[i].direction = this.params.momentum *
velocities[i].direction +
                                this.params.learningRate *
gradDirection;

        // Оновлюємо параметри камери
        const newPosition: Point = {
            x: camera.position.x + velocities[i].position.x,
            y: camera.position.y + velocities[i].position.y
        };

        // Перевіряємо, чи нова позиція допустима
        if (this.room.isInsideRoom(newPosition) &&
!this.room.isInsideObstacle(newPosition)) {
            camera.move(newPosition);
            improved = true;
        }

        // Оновлюємо напрямок камери
        camera.rotate(velocities[i].direction);
        improved = true;
    }

    // Обчислюємо нову енергію
    currentEnergy = this.evaluator.evaluate(currentSolution);

    // Оновлюємо найкращий розв'язок, якщо поточний кращий
    if (currentEnergy > bestEnergy) {
        bestSolution = this.cloneCameras(currentSolution);
        bestEnergy = currentEnergy;
    }

    // Якщо не було покращень або досягнуто заданої точності,
    завершуємо
    if (!improved || Math.abs(currentEnergy - bestEnergy) <
this.params.tolerance) {
        break;
    }
}

return bestSolution;

```

```

    }

    private calculatePositionGradient(camera: Camera, solution:
Camera[]): Point {
        const epsilon = 0.1; // Малий крок для чисельного
диференціювання
        const originalPosition = { ...camera.position };

        // Обчислюємо початкову енергію
        const originalEnergy = this.evaluator.evaluate(solution);

        // Обчислюємо градієнт по x
        camera.move({ x: originalPosition.x + epsilon, y:
originalPosition.y });
        const energyDx = this.evaluator.evaluate(solution);

        // Обчислюємо градієнт по y
        camera.move({ x: originalPosition.x, y: originalPosition.y +
epsilon });
        const energyDy = this.evaluator.evaluate(solution);

        // Повертаємо камеру в початкову позицію
        camera.move(originalPosition);

        // Повертаємо градієнт
        return {
            x: (energyDx - originalEnergy) / epsilon,
            y: (energyDy - originalEnergy) / epsilon
        };
    }

    private calculateDirectionGradient(camera: Camera, solution:
Camera[]): number {
        const epsilon = 1; // Малий крок кута для чисельного
диференціювання
        const originalDirection = camera.direction;

        // Обчислюємо початкову енергію
        const originalEnergy = this.evaluator.evaluate(solution);

        // Обчислюємо градієнт по напрямку
        camera.rotate(epsilon);
        const energyDdir = this.evaluator.evaluate(solution);

        // Повертаємо камеру в початковий напрямок
        camera.direction = originalDirection;

        // Повертаємо градієнт
        return (energyDdir - originalEnergy) / epsilon;
    }

    private cloneCameras(cameras: Camera[]): Camera[] {
        return cameras.map(camera => camera.clone());
    }

```

```

    }
}

```

Лістинг Б.5 - Реалізація алгоритму обчислення зон видимості камер

```

class Camera {
    public calculateVisibleArea(room: Room): Grid {
        const visibilityGrid = new Grid(room.width, room.height,
            room.grid.cellSize);

        // Кількість променів для кидання
        const numRays = 360; // Один промінь на кожен градус

        // Кидаємо промені в різних напрямках
        for (let i = 0; i < numRays; i++) {
            const angle = (this.direction - this.viewAngle / 2) +
                (this.viewAngle * i / numRays);
            const rayDirection: Point = {
                x: Math.cos(angle * Math.PI / 180),
                y: Math.sin(angle * Math.PI / 180)
            };

            // Запускаємо промінь
            this.castRay(room, rayDirection, visibilityGrid);
        }

        return visibilityGrid;
    }

    private castRay(room: Room, direction: Point, grid: Grid):
    void {
        // Початкова точка променя - позиція камери
        let point: Point = { ...this.position };

        // Крок променя
        const step = grid.cellSize / 2;

        // Відстань, пройдена променем
        let distance = 0;

        // Кидаємо промінь, поки він не досягне максимальної
        // дальності або не перетне перешкоду
        while (distance < this.range) {
            // Перевіряємо, чи точка знаходиться всередині приміщення
            if (!room.isInsideRoom(point)) {
                break;
            }

            // Перевіряємо, чи промінь перетинає стіну або перешкоду
            if (room.isWallOrObstacle(point)) {
                break;
            }
        }
    }
}

```

```

// Позначаємо клітинку як видиму
const cellX = Math.floor(point.x / grid.cellSize);
const cellY = Math.floor(point.y / grid.cellSize);

if (cellX >= 0 && cellX < grid.width && cellY >= 0 &&
cellY < grid.height) {
    // Обчислюємо якість видимості в цій точці
    const quality =
this.calculateVisibilityQuality(distance);

    grid.setCellValue(cellX, cellY, {
        isCovered: true,
        quality: quality,
        visibleFrom: [this.id],
        importance: room.grid.getCellValue(cellX,
cellY)?.importance || 1
    });
}

// Переміщуємо точку вздовж променя
point.x += direction.x * step;
point.y += direction.y * step;

// Оновлюємо пройденої відстані
distance += step;
}
}

private calculateVisibilityQuality(distance: number): number {
    // Якість видимості зменшується з відстанню
    // Наприклад, лінійне зменшення
    return Math.max(0, 1 - distance / this.range);

    // Або більш складна модель, яка враховує роздільну
    здатність камери
    // const resolutionFactor = this.getResolutionFactor();
    // return Math.max(0, 1 - Math.pow(distance / this.range,
resolutionFactor));
}

private getResolutionFactor(): number {
    // Конвертує роздільну здатність у числовий фактор
    // Наприклад, для HD (1280x720) повертає 1.0, для Full HD
    (1920x1080) - 1.5, тощо
    switch (this.resolution) {
        case '1280x720': return 1.0;
        case '1920x1080': return 1.5;
        case '3840x2160': return 2.0;
        default: return 1.0;
    }
}
}
}

```

Лістинг Б.6 - Реалізація класу для оцінки якості розв'язку

```

class SolutionEvaluator {
    private room: Room;
    private constraints: Constraints;
    private weightCoverage: number;
    private weightQuality: number;
    private weightCost: number;
    private weightPriority: number;
    private weightOverlap: number;

    constructor(
        room: Room,
        constraints: Constraints,
        weights: {
            coverage: number;
            quality: number;
            cost: number;
            priority: number;
            overlap: number;
        }
    ) {
        this.room = room;
        this.constraints = constraints;
        this.weightCoverage = weights.coverage;
        this.weightQuality = weights.quality;
        this.weightCost = weights.cost;
        this.weightPriority = weights.priority;
        this.weightOverlap = weights.overlap;
    }

    public evaluate(cameras: Camera[]): number {
        // Обчислюємо загальну карту покриття
        const coverageMap = this.calculateCoverageMap(cameras);

        // Обчислюємо відсоток покриття
        const coveragePercentage =
this.calculateCoveragePercentage(coverageMap);

        // Обчислюємо середню якість покриття
        const qualityScore =
this.calculateQualityScore(coverageMap);

        // Обчислюємо покриття пріоритетних зон
        const priorityScore =
this.calculatePriorityScore(coverageMap);

        // Обчислюємо показник перекриття (оптимальність
        // використання камер)
        const overlapScore =
this.calculateOverlapScore(coverageMap);

        // Обчислюємо загальну вартість системи

```

```

const cost = this.calculateTotalCost(cameras);

// Нормалізуємо вартість (менша вартість -> вищий бал)
const costScore = this.constraints.maxBudget > 0
  ? 1 - Math.min(1, cost / this.constraints.maxBudget)
  : 0;

// Обчислюємо штрафи за порушення обмежень
const penalties = this.calculatePenalties(cameras,
coveragePercentage);

// Обчислюємо загальний бал (зважена сума показників)
const totalScore =
  this.weightCoverage * coveragePercentage +
  this.weightQuality * qualityScore +
  this.weightCost * costScore +
  this.weightPriority * priorityScore +
  this.weightOverlap * overlapScore -
  penalties;

return totalScore;
}

private calculateCoverageMap(cameras: Camera[]): Grid {
  // Створюємо порожню карту покриття
  const coverageMap = new Grid(this.room.width,
this.room.height, this.room.grid.cellSize);

  // Для кожної камери обчислюємо її зону видимості та додаємо
до загальної карти
  for (const camera of cameras) {
    const visibilityGrid =
camera.calculateVisibleArea(this.room);
    coverageMap.merge(visibilityGrid);
  }

  return coverageMap;
}

private calculateCoveragePercentage(coverageMap: Grid): number
{
  let coveredCells = 0;
  let totalCells = 0;

  // Підраховуємо кількість покритих клітинок
  for (let x = 0; x < coverageMap.width; x++) {
    for (let y = 0; y < coverageMap.height; y++) {
      const cell = coverageMap.getCellValue(x, y);

      // Якщо клітинка належить приміщенню (не стіна і не
перешкода)
      if (this.room.isValidCell(x, y)) {
        totalCells++;
      }
    }
  }

  return coveredCells / totalCells;
}

```

```

        if (cell && cell.isCovered) {
            coveredCells++;
        }
    }
}

return totalCells > 0 ? coveredCells / totalCells : 0;
}

private calculateQualityScore(coverageMap: Grid): number {
    let totalQuality = 0;
    let totalCells = 0;

    // Обчислюємо середню якість покриття
    for (let x = 0; x < coverageMap.width; x++) {
        for (let y = 0; y < coverageMap.height; y++) {
            const cell = coverageMap.getCellValue(x, y);

            // Якщо клітинка належить приміщенню і покрита
            if (this.room.isValidCell(x, y) && cell &&
cell.isCovered) {
                totalQuality += cell.quality;
                totalCells++;
            }
        }
    }

    return totalCells > 0 ? totalQuality / totalCells : 0;
}

private calculatePriorityScore(coverageMap: Grid): number {
    if (!this.constraints.priorityAreas ||
this.constraints.priorityAreas.length === 0) {
        return 1; // Якщо немає пріоритетних зон, повертаємо
максимальний бал
    }

    let totalPriorityScore = 0;

    // Для кожної пріоритетної зони обчислюємо її покриття
    for (const area of this.constraints.priorityAreas) {
        const areaCoverage =
this.calculateAreaCoverage(coverageMap, area);
        totalPriorityScore += areaCoverage * area.importance;
    }

    // Нормалізуємо результат
    const totalImportance =
this.constraints.priorityAreas.reduce(
        (sum, area) => sum + area.importance, 0
    );

```

```

    return totalImportance > 0 ? totalPriorityScore /
totalImportance : 0;
}

private calculateAreaCoverage(coverageMap: Grid, area:
PriorityArea): number {
    // Перетворюємо полігон зони в набір клітинок сітки
    const areaCells = this.polygonToCells(area.points);

    let coveredCells = 0;

    // Підраховуємо кількість покритих клітинок у зоні
    for (const cell of areaCells) {
        const cellValue = coverageMap.getCellValue(cell.x,
cell.y);

        if (cellValue && cellValue.isCovered) {
            coveredCells++;
        }
    }

    return areaCells.length > 0 ? coveredCells /
areaCells.length : 0;
}

private polygonToCells(points: Point[]): { x: number; y:
number }[] {
    // Реалізація алгоритму растеризації полігону
    // Повертає набір клітинок сітки, що належать полігону
    // ...
    return [];
}

private calculateOverlapScore(coverageMap: Grid): number {
    let totalOverlap = 0;
    let totalCells = 0;

    // Обчислюємо середнє перекриття зон видимості
    for (let x = 0; x < coverageMap.width; x++) {
        for (let y = 0; y < coverageMap.height; y++) {
            const cell = coverageMap.getCellValue(x, y);

            // Якщо клітинка належить приміщенню і покрита
            if (this.room.isValidCell(x, y) && cell &&
cell.isCovered) {
                // Кількість камер, що бачать цю клітинку
                const numCameras = cell.visibleFrom.length;

                // Оптимальне перекриття - 1 або 2 камери
                // Менше 1 - недостатнє покриття, більше 2 - надмірне
                // перекриття
                const optimalOverlap = numCameras <= 2

```

```

        ? numCameras / 2
        : 2 / numCameras;

        totalOverlap += optimalOverlap;
        totalCells++;
    }
}

return totalCells > 0 ? totalOverlap / totalCells : 0;
}

private calculateTotalCost(cameras: Camera[]): number {
    // Обчислюємо загальну вартість усіх камер
    return cameras.reduce((sum, camera) => sum + camera.cost,
0);
}

private calculatePenalties(cameras: Camera[],
coveragePercentage: number): number {
    let penalties = 0;

    // Штраф за перевищення максимальної кількості камер
    if (this.constraints.maxCameras > 0 && cameras.length >
this.constraints.maxCameras) {
        penalties += 10 * (cameras.length -
this.constraints.maxCameras);
    }

    // Штраф за перевищення бюджету
    const totalCost = this.calculateTotalCost(cameras);
    if (this.constraints.maxBudget > 0 && totalCost >
this.constraints.maxBudget) {
        penalties += 10 * (totalCost - this.constraints.maxBudget)
/ this.constraints.maxBudget;
    }

    // Штраф за недостатнє покриття
    if (this.constraints.minCoverage > 0 && coveragePercentage <
this.constraints.minCoverage) {
        penalties += 10 * (this.constraints.minCoverage -
coveragePercentage);
    }

    return penalties;
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для системи відеоспостереження.

Виконав студент групи КНТ-131
Керівник, к.т.н, доцент

Степан КУТИРЄВ
Степан СКРУПСЬКИЙ

Рисунок В.1 – Слайд 1

Мета та задачі

Об'єктом дослідження є процес розроблення програмного забезпечення для оптимізації розміщення камер відеоспостереження в приміщеннях.

Предметом дослідження є методи та програмні засоби автоматизованого планування розміщення відеокамер на основі аналізу геометрії приміщень.

Метою роботи є підвищення ефективності проектування систем відеоспостереження шляхом розробки програмного забезпечення для автоматичного визначення оптимального розташування камер.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- ▶ дослідити існуючі підходи до автоматизації проектування систем відеоспостереження;
- ▶ розробити алгоритм оптимального розміщення камер з урахуванням геометрії приміщень та перешкод;
- ▶ створити вебзастосунок для взаємодії користувачів з розробленим алгоритмом;
- ▶ реалізувати інтерактивний інтерфейс для візуалізації результатів роботи алгоритму;
- ▶ провести тестування розробленого програмного забезпечення на різних типах приміщень.

2

Рисунок В.2 – Слайд 2

Аналіз сучасних систем відеоспостереження

Параметр	Купольні камери	Циліндричні камери	PTZ-камери	Панорамні камери
Кут огляду	90-120°	70-90°	360° (з поворотом)	180-360°
Максимальна дальність	15-30 м	30-100 м	100-300 м	15-20 м
Вплив освітлення	Середній	Високий	Високий	Низький
Захист від вандалізму	Високий	Середній	Низький	Низький
Вартість	Середня	Середня	Висока	Висока
Складність монтажу	Низька	Середня	Висока	Низька



3

Рисунок В.3 – Слайд 3

Порівняння існуючих програмних рішень для систем відеоспостереження

Назва ПЗ	Підтримка обладнання	Аналітичні можливості	Оптимізація розміщення камер	Інтерфейс	Ціна
Milestone XProtect	Широка	Високі	Відсутня	Складний	Висока
Avigilon Control Center	Обмежена	Високі	Обмежена	Середньої складності	Висока
Genetec Security Center	Широка	Високі	Відсутня	Складний	Висока
iSpy	Широка	Низькі	Відсутня	Простий	Безкоштовно
ZoneMinder	Середня	Середні	Відсутня	Складний	Безкоштовно

4

Рисунок В.4 – Слайд 4

Архітектура програмного забезпечення оптимізації розміщення камер відеоспостереження

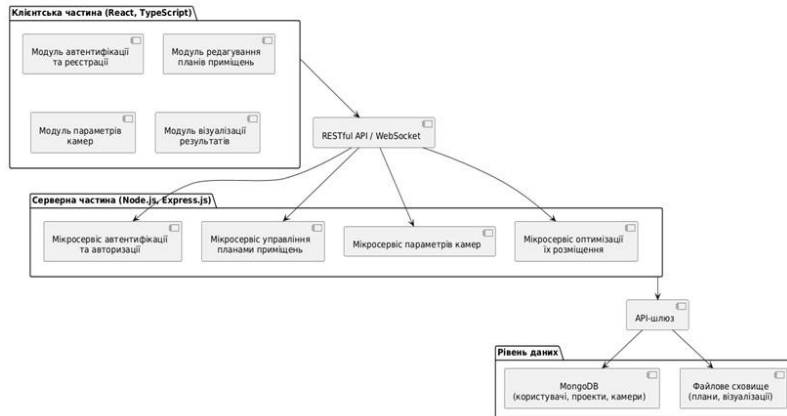


Рисунок В.5 – Слайд 5

Структура алгоритму оптимізації розміщення камер



Рисунок В.6 – Слайд 6

Структура класів для реалізації алгоритму оптимізації

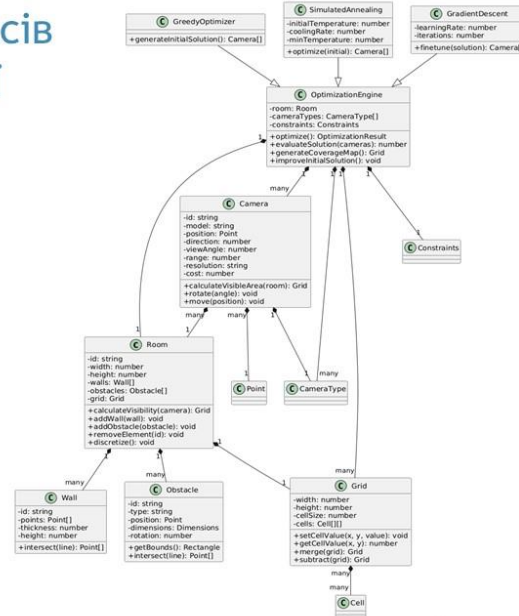


Рисунок В.7 – Слайд 7

Концептуальна модель даних системи

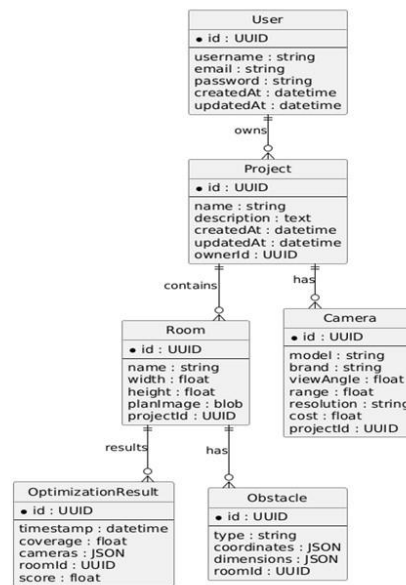


Рисунок В.8 – Слайд 8

Структура інтерфейсу користувача системи

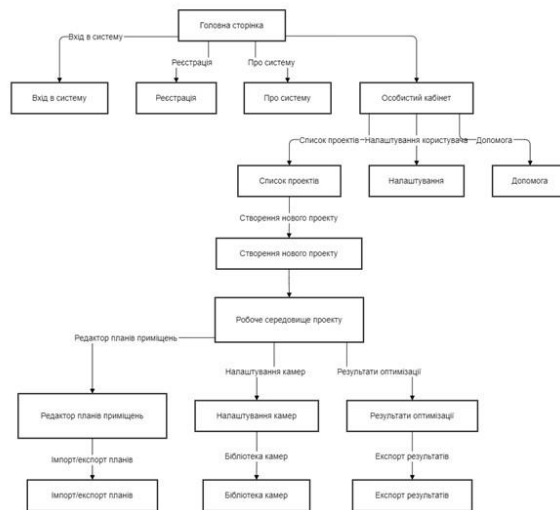


Рисунок В.9 – Слайд 9

Опис програмного забезпечення

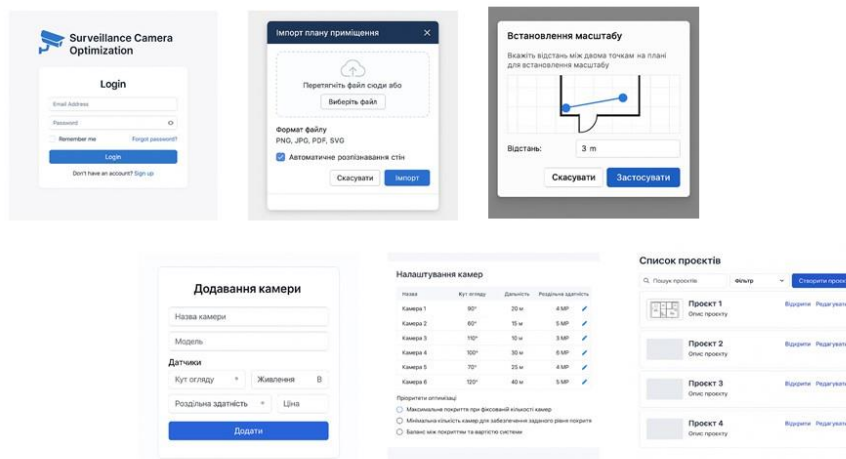
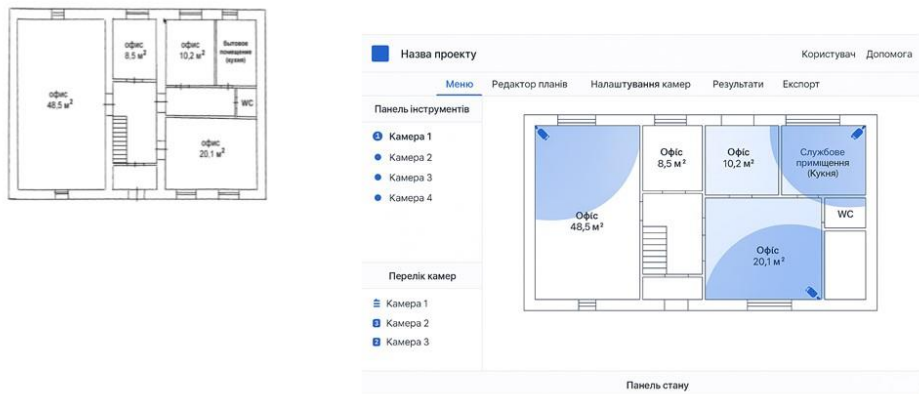


Рисунок В.10 – Слайд 10

Опис програмного забезпечення



11

Рисунок В.11 – Слайд 11

Вимоги до програмного забезпечення серверної частини

Компонент	Мінімальна версія	Рекомендована версія	Примітки
Операційна система	Ubuntu Server 18.04, Windows Server 2016	Ubuntu Server 22.04, Windows Server 2022	Підтримуються й інші дистрибутиви Linux
Node.js	14.0	18.0	Потрібен npm 6.0 або новіше
MongoDB	4.4	6.0	Рекомендовано використання replica set
RabbitMQ	3.8	3.11	Потрібен erlang 23.0 або новіше
Nginx	1.18	1.22	Можна замінити на Apache
Redis	6.0	7.0	Використовується для кешування

12

Рисунок В.12 – Слайд 12

Результати функціонального тестування

Функція	Результат тестування	Примітки
Імпорт планів приміщень	Успішно	Підтримка форматів PNG, JPG, PDF, SVG
Редагування елементів приміщення	Успішно	Додавання/видалення стін, дверей, вікон
Налаштування параметрів камер	Успішно	Робота з бібліотекою типових камер
Запуск оптимізації	Успішно	Коректна робота з усіма тестовими планами
Візуалізація результатів	Успішно	2D режими перегляду

13

Рисунок В.13 – Слайд 13

Висновки

- Розроблено програмне забезпечення для оптимізації розміщення камер відеоспостереження, що дозволяє автоматизувати процес проектування систем безпеки.
- Створена мікросервісна архітектура забезпечує гнучкість, масштабованість та високу продуктивність системи при роботі з приміщеннями різного типу.
- Впроваджений комбінований алгоритм оптимізації (жадібний алгоритм + імітація відпалу + градієнтний спуск) забезпечує оптимальне покриття при мінімальних витратах.
- Розроблений інтуїтивний вебінтерфейс робить систему доступною для широкого кола користувачів від власників приватної нерухомості до професіоналів.
- Тестування підтвердило ефективність системи на різних типах приміщень та її відповідність поставленим вимогам.
- Перспективи розвитку: інтеграція з 3D-моделями приміщень, сумісність з системами розумного дому, впровадження технологій штучного інтелекту для аналізу покриття.

14

Рисунок В.14 – Слайд 14