Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця»

Виконав: студент 2 курсу, групи 2АКІТР-23м
спеціальності 174 – Автоматизація,
комп'ютерно-інтегровані технології та
робототехніка
Артур НІКОНЧУК

Керівник: д.т.н., проф. каф. КСУ
В'ячеслав КОВТУН

«05» 12 2024р.

Опонент: к.т.н., доцент
Іванов Ю.Ю.

«05» 12 2024р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав КОВТУН
«05» 12 2024р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність – 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Зав. кафедри КСУ

В'ячеслав КОВТУН

"14" жовтня 2024 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Нікончуку Артуру Павловичу.
(прізвище, ім'я, по батькові)

1. Тема роботи. "Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця"
керівник роботи В'ячеслав Ковтун
затверджені наказом ВНТУ від "17" вересня 2024 року №310
2. Термін подання студентом роботи "1" грудня 2024 року
3. Вихідні дані до роботи: вихідні дані до роботи представлені у структурованому csv файлі. Формуючи вихідні дані автор використовував такі інформаційні джерела:

1) B. Hrytsenko, "Design of Cyber-Physical Security Systems Using IoT and Machine Learning Technologies," IEEE Access, vol. 9, pp. 65721-65730, 2021. DOI: 10.1109/ACCESS.2021.3079087. (10 pages).
2) I. Lebed, "Algorithms for Object Detection and Tracking in Security Surveillance Systems," Cybernetics and Systems Analysis, vol. 57, no. 3, pp. 227-234, 2021. DOI: 10.1007/s10559-021-00352-5. (8 pages).
3) O. Mishchenko, "Optimization of Image Processing Algorithms in Real-Time Surveillance Systems," International Journal of Electronics and Telecommunications, vol. 68, no. 2, pp. 177-183, 2022. DOI: 10.24425/ijet.2022.141732. (7 pages).

4. Зміст текстової частини: текстова частина включає анотацію, зміст, вступ, основну частину з першим, оглядовим, розділом, висновки, список посилань та додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових

запропонованої програмно-апаратної системи, алгоритми роботи програної складової розробки, результати тестування створеної системи.

1. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
4	Кавецький В.В. – доцент кафедри економіки підприємства і виробничого менеджменту	14.10.24	05.12.2024

2. Дата видачі завдання "14" жовтня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналітичний огляд переваг та недоліків, які характеризують досліджуваний технологічний процес	14.10.2024	15.10.2024	
2	Дослідження стану розв'язання технічної задачі	23.10.2024	25.10.2024	
3	Розробка та тестування системи	10.11.2024	15.11.2024	
4	Розрахунок економічної частини	24.11.2024	26.11.2024	
5	Оформлення матеріалів до захисту	27.11.2024	29.11.2024	

Студент Артур Нікончук

Анотація

УДК 669.05

Нікончук А. П. Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця. Магістерська кваліфікаційна робота зі спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка», освітня програма – Інтелектуальні комп'ютерні системи. Вінниця: ВНТУ, 2024. - 168 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 44; табл. 18.

У магістерській кваліфікаційній роботі розглянуто розробку інтелектуальної автоматизованої системи для опрацювання даних відеоканалу роботизованої охоронної платформи. Метою роботи є підвищення ефективності процесу передачі зображень між камерою роботизованої охоронної платформи та сервером через оптимізацію зображень, стиснення відео та зменшення кількості кадрів за секунду, що дозволяє зменшити навантаження на сервер і підвищити швидкість обробки даних без втрати якості. Запропоновано використання растрового формату для збереження зображень і стиснення відео в форматах MJPEG та WebP. Визначено технології для реалізації серверної та клієнтської частин, зокрема використано платформу Node, протокол WebSocket і базу даних MongoDB для зберігання інформації про користувачів. Розроблено архітектуру системи, встановлено взаємодію модулів і налаштовано процес стиснення та передачі зображень. Впровадження цієї системи дозволяє зменшити навантаження на сервер і підвищити швидкість передачі даних без втрати якості зображень. Також проведено маркетингове дослідження та визначено цільову аудиторію для виведення системи на ринок.

Ключові слова: Інтелектуальна автоматизована система, Оптимізація зображень, Роботизована охоронна платформа, 4. WebP, Стиснення відео, 6. WebSocket.

Annotation

UDC 669.05

Nikonchuk A. P. Intelligent automated data processing system of a robot security guard video channel. Master's qualification work in specialty 174 "Automation, computer-integrated technologies and robotics", educational program - Intelligent computer systems. Vinnytsia: VNTU, 2024. - 168 p.

In Ukrainian. Bibliography: 21 titles; fig.: 44; tab. 18.

The master's qualification work considers the development of an intelligent automated system for processing video channel data of a robotic security platform. The aim of the work is to increase the efficiency of the image transmission process between the camera of a robotic security platform and the server through image optimization, video compression and a reduction in the number of frames per second, which allows reducing the load on the server and increasing the speed of data processing without loss of quality. The use of a raster format for storing images and compressing video in MJPEG and WebP formats is proposed. Technologies for implementing the server and client parts are determined, in particular, the Node platform, the WebSocket protocol and the MongoDB database are used to store user information. The system architecture is developed, the interaction of modules is established and the image compression and transmission process is configured. The implementation of this system allows reducing the load on the server and increasing the speed of data transmission without loss of image quality. A marketing study is also conducted and the target audience for bringing the system to the market is determined.

Keywords: Intelligent automated system, Image optimization, Robotic security platform, 4. WebP, Video compression, 6. WebSocket.

ЗМІСТ

ВСТУП

Сучасні підприємства по всьому світу активно інтегрують робототехніку у різноманітні процеси, такі як підбір і упаковка замовлень, зварювання металевих деталей, нанесення фарб і герметиків, а також здійснення високоточних перевірок і багатьох інших операцій. Лише у 2023 році було зафіксовано рекордний обсяг постачань промислових роботів — 381 000 одиниць, що на 30% більше порівняно з попереднім роком.

Сектора, такі як електроніка, автомобілебудування і металургія, активно використовують робототехніку для автоматизації своїх виробничих процесів та стимулювання економічного зростання. Завдяки роботам компанії можуть підвищити ефективність виробництва і забезпечити стабільно високу якість продукції. Раніше роботи були обмежені в своїх функціях, працюючи лише в певних локаціях і не маючи можливості взаємодіяти з людьми. Однак сьогодні ці машини здатні оцінювати своє оточення та співпрацювати з людьми в реальному часі.

Хоча роботи вже займають значну роль у таких галузях, як логістика та виробництво, лише починається усвідомлення їхнього потенціалу в області безпеки. Роботи-охоронці поступово стають популярними у торгових центрах, офісах і громадських місцях. Вони все частіше використовуються не лише як заміна людським охоронцям, а й як мобільні машини для збору даних і здійснення моніторингу. Завдяки здатності робити це без постійної присутності людини, роботи можуть значно збільшити ефективність спостереження за об'єктами.

Завданням таких роботів є постійний збір інформації, яка включає в себе, серед іншого, розпізнавання номерних знаків, облич, а також виявлення мобільних пристроїв у межах їхнього досяжності. Це дозволяє підтримувати постійний контроль за ситуацією, що може бути надзвичайно важливим у забезпеченні безпеки. Замість того, щоб просто замінювати людину, роботи-

охоронці працюють як доповнення до традиційних систем безпеки, даючи змогу зменшити навантаження на персонал і забезпечити більш детальну та безперервну роботу.

На даний момент основною технологією відеоспостереження залишаються статичні камери, встановлені на стінах або стелях, які передають і зберігають інформацію на спеціалізованих серверах. Власники таких систем мають можливість дистанційно переглядати відео або слідкувати за ситуацією онлайн. Для ефективної роботи таких систем потрібна високоякісна камера, стабільний інтернет для передачі даних на сервер і достатній обсяг пам’яті для тривалого зберігання відеофайлів.

Для зменшення навантаження на сервери та підвищення швидкості передачі даних необхідно оптимізувати розмір зображень без суттєвої втрати якості. Також важливим є можливе зменшення кількості кадрів за секунду, що дозволяє знизити обсяг переданих даних, зберігаючи їхню інформативність.

Одним із перспективних рішень для забезпечення безпеки є роботизовані мобільні платформи, оснащені камерами для ведення фотозйомки. Така система дозволяє користувачеві дистанційно контролювати рух робота, здійснювати моніторинг стану приміщень або переглядати історію зйомок. Важливим аспектом для такої системи є ефективна оптимізація зображень, що передаються на сервер, для зменшення їхнього обсягу без значної втрати якості.

Таким чином, у цій роботі розглядається важлива задача захисту приміщень за допомогою мобільних роботизованих платформ, які проводять фотозйомку, оптимізують зображення та передають їх на сервер для подальшого зберігання. Одним з основних аспектів є оптимізація процесу передачі зображень для підвищення швидкодії і зменшення навантаження на сервери, а також для забезпечення більш ефективного використання каналів передачі даних.

Об'єкт дослідження:

Процес передачі оптимізованих зображень від камери роботизованої охоронної платформи до серверу для подальшого зберігання та обробки.

Предмет дослідження:

Технології, методи та ефективність оптимізації зображень для зменшення навантаження на сервери та покращення швидкості передачі даних у системах роботизованого відеоспостереження.

Мета дослідження:

Підвищення ефективності процесу передачі зображень між камерою роботизованої охоронної платформи та сервером через оптимізацію зображень, стиснення відео та зменшення кількості кадрів за секунду, що дозволяє зменшити навантаження на сервер і підвищити швидкість обробки даних без втрати якості.

Завдання дослідження:

1. Розробити та обґрунтувати методи оптимізації зображень для ефективної передачі даних у відеоканалі роботизованої охоронної платформи.

2. Аналізувати і порівняти різні формати стиснення відео (MJPEG, WebP) щодо їх ефективності у зменшенні розміру файлів та збереженні якості зображень.

3. Розробити архітектуру системи для передачі і зберігання зображень, що включає вибір технологій для серверної та клієнтської частин, таких як Node.js, WebSocket і MongoDB.

4. Дослідити вплив оптимізації на швидкість обробки даних і навантаження на сервер.

5. Провести маркетингове дослідження для визначення цільової аудиторії і можливостей виведення системи на ринок.

Практична цінність дослідження:

Запропоноване рішення дозволяє значно знизити навантаження на сервери та збільшити швидкість передачі даних без втрати якості зображень, що важливо для покращення роботи роботизованих охоронних систем у реальному часі. Реалізація розробленої інтелектуальної автоматизованої системи сприятиме підвищенню ефективності систем відеоспостереження в різних сферах застосування, таких як безпека в торгових центрах, офісах та інших громадських місцях.

Наукова новизна:

1. Вперше застосовано комплексний підхід до оптимізації зображень у роботизованих охоронних платформах з використанням растрових форматів та стиснення відео в MJPEG і WebP для зменшення навантаження на сервери та підвищення швидкості обробки.

2. Розроблено нову архітектуру інтелектуальної автоматизованої системи з використанням технологій Node.js, WebSocket і MongoDB, що дозволяє ефективно здійснювати передачу і зберігання зображень у реальному часі.

3. Визначено оптимальні параметри стиснення зображень і кадрів для забезпечення ефективної роботи системи з високими вимогами до швидкості і якості передачі даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд існуючих функціональних можливостей систем захисту

У сучасному світі системи безпеки відіграють важливу роль у забезпеченні безпеки людей, майна та інформації. Вони мають широкий набір функціональних можливостей, які дозволяють автоматизувати процеси моніторингу, зберігання та обробки даних. Сучасні технології значно полегшують роботу користувачів, дозволяючи їм отримувати необхідну інформацію в реальному часі та приймати оперативні рішення.

Однією з основних інновацій є датчики руху, які дозволяють знімати відео лише тоді, коли це необхідно. Це знижує навантаження на сервери та зменшує потребу в великих обсягах зберігання даних, оскільки відео записується тільки в момент виявлення руху в кадрі. Відповідно, система може надсилати сповіщення на комп'ютер або мобільний телефон, якщо виявлений рух у приміщенні, де це не очікується (Рис.1.1). Такий підхід значно покращує ефективність спостереження та знижує витрати на зберігання даних[6].

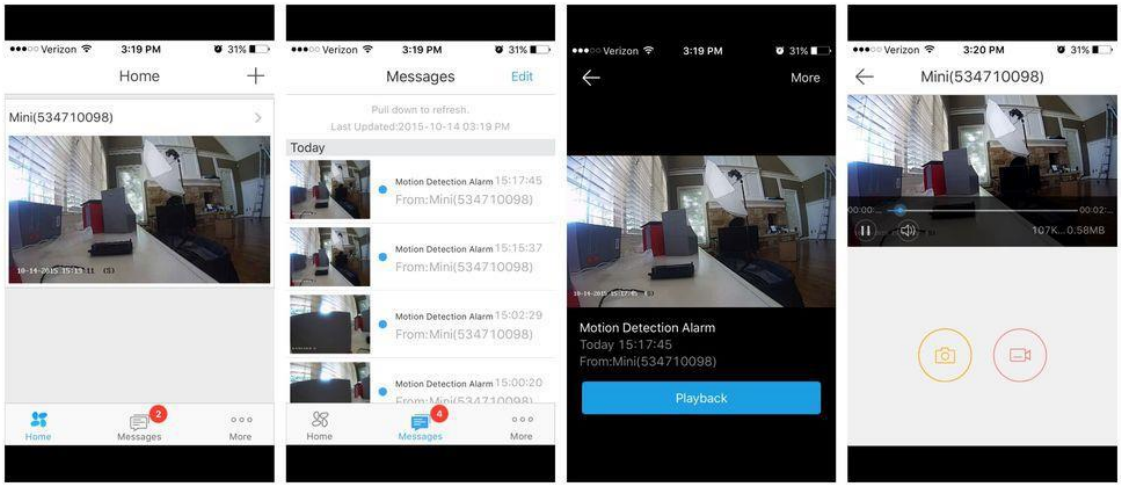


Рисунок 1.1 – Сповіщення про рух у приміщенні

Двостороннє аудіо є ще однією важливою функцією, яка дозволяє вести двосторонній зв'язок між людиною, що спостерігає за відео, і особою, що знаходиться перед камерою. Це може бути корисним для спілкування з відвідувачами або для виявлення непередбачених ситуацій, що вимагають оперативного втручання.

Зі зростанням вимог до якості відеозаписів, сучасні камери відеоспостереження пропонують значно кращу роздільну здатність. Вони здатні записувати відео з роздільною здатністю від 720p до 1080p або навіть вище, що дозволяє чітко ідентифікувати події на зображенні, навіть в умовах слабкого освітлення. Проте такі відео створюють проблему для інтернет-з'єднання, оскільки високоякісне відео потребує значної пропускної здатності для потокової передачі[3].

Одним з основних елементів системи безпеки є сховище даних, яке може бути локальним або хмарним. Локальне зберігання дозволяє зберігати відео безпосередньо на пристрої (жорсткому диску або SSD), проте обсяг зберігання обмежений розміром носія. Це створює проблему, якщо потрібно зберігати великий обсяг даних протягом тривалого часу. У такому випадку необхідно регулярно очищати накопичувач або використовувати більш ємні носії.

З хмарним сховищем можна вирішити проблему обмеженого місця, оскільки відео автоматично завантажується в хмару для зберігання. Однак, такі сервіси часто вимагають підписки з оплатою та накладають обмеження на кількість відеокліпів або термін їх зберігання. Крім того, при зберіганні даних у хмарі виникають питання безпеки та приватності даних, оскільки вони зберігаються на серверах сторонніх організацій.

Мобільні додатки (Рис.1.2) стали невід'ємною частиною сучасних систем безпеки. Вони дозволяють користувачам віддалено переглядати відео з камер спостереження, отримувати сповіщення про рух та інші важливі події.

Рисунок 1.2 – Мобільний додаток для доступу до сховища

Це дозволяє здійснювати моніторинг і приймати рішення навіть тоді, коли користувач знаходиться поза домом чи офісом[7].

Таким чином, інтеграція цих функціональних можливостей у сучасні системи захисту дозволяє значно підвищити рівень безпеки та забезпечити оперативне реагування на загрози в реальному часі. Однак, кожна з цих технологій має свої переваги та недоліки, що потрібно враховувати при виборі оптимальної системи для конкретного випадку.

1.2 Аналіз існуючих систем захисту приміщень

Хоча сучасні камери для захисту приміщень мають велику кількість функціональних можливостей, досі основною проблемою залишається збереження обмеженої кількості даних на внутрішньому накопичувачі пристрою, а для розширення – необхідно використовувати хмарні сервіси, за які потрібно буде платити щомісячно або щорічно. А враховуючи роздільну здатність сучасних камер, накопичувач буде заповнюватися неймовірно швидко[8].

Іншою проблемою є обмежена можливість доступу для віддаленого перегляду того, що відбувається в даний час у приміщенні, або для перегляду

історії, оскільки для доступу існують певні програми, якими можуть користуватися тільки користувачі певних мобільних пристроїв.

Розглянемо камеру для захисту приміщень Arlo Pro4, зображену на Рисунку 1.3.

Рисунок 1.3 – Камера Arlo Pro4

За допомогою спеціального додатку Arlo дозволяється отримувати доступ до камер і управляти ними за допомогою смартфона. Система виявлення руху дозволяє камерам економити заряд батареї та зменшувати завантаження сховища, записуючи тільки тоді, коли є рух перед камерою[13].

Певним недоліком системи є необхідність купівлі обов’язкової базової станції. Але разом з цим, вони покращують час автономної роботи в порівнянні з камерами безпеки, які використовують тільки Wi-Fi. Вони також забезпечують локальне зберігання відео, підключаючи запам'ятовуючий пристрій до концентратора. Базова станція – це найкращий спосіб записувати локальне відео без хмарної підписки, що в довгостроковій перспективі економить більше грошей. Камера надає вихідне зображення із роздільною якістю 1440 пікселів (2К) та кутом огляду в 160 градусів. Однак розмір вихідного файлу є досить великим, тому накопичувач доволі швидко

заповнюється, і користувачу необхідно видалити непотрібні файли з накопичувача.

Користувач має можливість переглядати відео в реальному часі в додатку Arlo – який необхідно завантажити на свій мобільний пристрій – в будь-який час без підписки, але також надає додаткові функції і більш високу якість зберігання відео тільки з його платними планами Arlo Smart.

Розглянемо ще одну камеру, призначену для захисту приміщень, Reolink Argus 2, яка зображена на Рисунку 1.4.

Рисунок 1.4 – Камера Reolink Argus 2

Reolink – це IP-камера, тобто користувач може мати доступ до прямої відео трансляції з будь-якого місця за допомогою програми Reolink, яку необхідно завантажити на смартфон. Камера має вбудований датчик руху, який економить час автономної роботи, записуючи на карту microSD тільки тоді, коли датчик виявляє певний рух, а не працює безперервно. Мобільний додаток дозволяє налаштувати сповіщення або змінити роздільну здатність зображення[6].

До переваг можна віднести можливість зберігати відео у хмарні сервіси безкоштовно та без необхідності у зовнішньому накопичувачі. Якщо ж інтернет-мережа не є доступною, у такому разі необхідно купити зовнішній накопичувач, щоб мати можливість зберігати відзнятий відеоматеріал.

Також за допомогою мобільного додатку є змога змінювати налаштування якості зображення для різних переваг, таких як збільшення роздільної здатності для більшої чіткості або зниження її для кращого часу автономної роботи і кращої продуктивності при більш низькій швидкості з'єднання. За допомогою програми також можна знімати відео і фотографії з прямої трансляції і зберігати їх на свій телефон, навіть без карти microSD, а також надавати іншим членам сім'ї доступ до прямої трансляції за допомогою QR-коду.

Великим недоліком є те, що при максимальній швидкості відео всього 15 кадрів в секунду, відео потік Reolink Argus 2 набагато більш переривчастий, ніж у інших наших камер безпеки. Існує також приблизно 2,5-секундна затримка, перш ніж камера викличе сигнал тривоги про рух, тому камера не встигне зняти необхідний матеріал.

У зв'язку з розвитком технологій в галузі безпеки, все більше компаній почали впроваджувати роботизовані системи для охорони приміщень. Роботи-охоронці є інтеграцією декількох технологій, включаючи штучний інтелект, автономні пересування та високоточні сенсори. Вони здатні не тільки спостерігати за об'єктами через камери відеоспостереження, але й реагувати на події в реальному часі, здійснювати патрулювання території, перевіряти підозрілі сигнали та навіть здійснювати автоматичні сповіщення користувачів[12].

Ці роботи можуть бути оснащені різними функціями, такими як автоматичне виявлення руху, розпізнавання осіб, голосове спілкування, а також інтеграція з іншими системами безпеки. Вони здатні ефективно працювати в умовах, де традиційні камери або людська охорона можуть не забезпечити достатньої присутності або своєчасної реакції.

Основним перевагою роботів-охоронців є їхня автономність, що дозволяє знижувати потребу в людському нагляді та знижує ризик людських помилок. Вони можуть працювати цілодобово, не вимагаючи перерв, і здатні

здійснювати гнучке управління на основі даних з камери відеоспостереження. Проте існують і деякі недоліки, такі як високі початкові витрати на придбання та технічне обслуговування таких пристроїв, а також необхідність інтеграції з іншими системами безпеки для забезпечення їх ефективності[15].

Таким чином, роботи-охоронці можуть стати важливою складовою частиною сучасних систем захисту приміщень, працюючи в комплексі з камерами спостереження, системами зберігання даних і хмарними сервісами, що забезпечує ефективний рівень безпеки при збереженні високої функціональності та автономності.

1.3 Інтеграція роботів-охоронців з відеокамерами в сучасні системи захисту та існуючі протоколи безпеки

Інтеграція роботизованих систем охорони з відеокамерами в сучасні комплекси безпеки є одним з найбільш інноваційних кроків для підвищення ефективності та автоматизації охорони об'єктів. Завдяки такій інтеграції створюються автономні системи, які забезпечують вищий рівень надійності, скорочення витрат та мінімізацію людської участі в процесах охорони. Відеокамери є невід'ємною частиною цієї інтеграції, оскільки вони забезпечують збирання та передачу відеоінформації, що дозволяє роботам вчасно реагувати на загрози[3].

1.3.1 Роботи-охоронці як частина сучасних систем безпеки

Роботи-охоронці в сучасних умовах — це не просто автономні пристрої, що патрулюють об’єкти. Вони є частиною складної інтегрованої системи, що включає в себе різноманітні сенсори, камери, мережі для обміну даними та програмне забезпечення для аналізу і прийняття рішень. Відеокамери, вмонтовані в роботів-охоронців, дозволяють збирати високоякісні відео- та аудіо дані для подальшого аналізу, допомагаючи роботам здійснювати

патрулювання, оцінювати ситуацію на об'єкті і здійснювати оперативні заходи на основі отриманих даних.

Автономні роботи можуть не лише виконувати патрулювання об'єкта, а й мати здатність до взаємодії з іншими елементами охоронної системи. Зокрема, роботи можуть передавати відеопотоки на сервери для збереження архіву або негайного опрацювання, а також здійснювати реєстрацію подій з автоматичним виявленням порушень — руху в заборонених зонах або спроби несанкціонованого доступу. Завдяки таким роботизованим рішенням знижуються помилки, пов'язані з людським фактором, і підвищується точність та оперативність виявлення загроз[3].

1.3.2 Інтеграція з системами відеоспостереження

Інтеграція роботів з відеокамерами є одним з найважливіших елементів системи відеоспостереження, що дозволяє здійснювати комплексний моніторинг об'єкта. Камери, розташовані на роботах, дозволяють здійснювати відеоспостереження в будь-який час доби, в тому числі в умовах низької освітленості або поганих погодних умов, завдяки використанню інфрачервоних або тепловізійних камер. Сучасні камери можуть працювати в реальному часі, передаючи відео в форматах високої роздільної здатності, таких як HD або 4K.

Інтеграція відеокамер з роботами також дозволяє здійснювати не лише запис і моніторинг, а й реалізувати функції розпізнавання образів. Наприклад, роботи можуть бути оснащені алгоритмами машинного навчання для розпізнавання облич, категоризації рухомих об'єктів або аналізу ситуацій для виявлення аномальних подій. Ці системи можуть бути інтегровані з іншими елементами охорони, такими як датчики руху, тривожні кнопки та інші пристрої, щоб здійснювати спільне реагування на порушення безпеки.

Крім того, відеокамери, що інтегровані в роботи, можуть автоматично коригувати свої кути огляду залежно від напрямку руху робота, що значно

підвищує ефективність спостереження на великих територіях. Такі камери також можуть мати функції зуму, що дозволяє зосередитися на конкретних об'єктах або деталях для їх більш точного аналізу[3].

1.3.3 Протоколи безпеки для захисту даних

Використання роботів для охорони та відеоспостереження підвищує вимоги до захисту даних, оскільки передача відео та команд між роботами і системами повинна бути надійно захищена від несанкціонованого доступу. Для цього використовуються сучасні протоколи безпеки, такі як TLS (Transport Layer Security) і SSL (Secure Sockets Layer), які забезпечують шифрування передачі даних між пристроями. Це дозволяє запобігти перехопленню відеопотоків або маніпуляціям з командами, що передаються до роботів.

Використання таких протоколів дає змогу створювати безпечні канали зв'язку, що критично важливо для систем, де дані можуть містити конфіденційну інформацію або бути основою для прийняття оперативних рішень. Крім того, для забезпечення більш високого рівня безпеки застосовуються механізми двухфакторної аутентифікації та системи управління доступом, що дозволяють гарантувати, що тільки авторизовані користувачі можуть мати доступ до управлінських функцій системи.

На додаток до цього, всі відеопотоки і дані, що надходять від роботів-охоронців, можуть бути захищені за допомогою битового шифрування для запобігання доступу до відеофайлів в разі їх перехоплення. Це дозволяє забезпечити цілісність інформації, а також запобігти можливим атакам на систему[14].

1.3.4 Перспективи розвитку та можливості інтеграції

Інтеграція роботів-охоронців з відеокамерами відкриває великі можливості для розвитку сучасних охоронних технологій. З появою нових алгоритмів на основі штучного інтелекту та машинного навчання роботизовані системи здатні не лише фіксувати відеоінформацію, а й здійснювати самостійний аналіз ситуації, приймати рішення про дії на місці та навіть передавати ці рішення в реальному часі до центральних серверів.

Одним з найбільш перспективних напрямків є впровадження систем розпізнавання осіб, об'єктів та аналітики поведінки на основі відео. Ці технології дозволяють роботам автоматично розпізнавати несанкціоновані вхід або підозрілі рухи та вживати відповідні заходи: включення сигналізації, блокування шляхів доступу або виклик охорони.

Інтеграція з інтернетом речей (IoT) відкриває нові можливості для з'єднання роботів з іншими системами будівлі, такими як системи освітлення, контроль доступу або пожежна сигналізація. Це дозволяє створювати інтегровані системи безпеки, де роботи можуть автоматично взаємодіяти з іншими компонентами, підвищуючи рівень безпеки та ефективності управління[3].

1.3.5 Виклики та обмеження

Попри всі переваги інтеграції роботів-охоронців з відеокамерами, існує низка викликів, які потребують вирішення. Одним із основних є висока потреба в обчислювальних потужностях для обробки відео та великих обсягів даних, що передаються в реальному часі. Для забезпечення високої якості відео і ефективності обробки необхідні потужні сервери та спеціалізоване програмне забезпечення для обробки відео та керування роботами.

Крім того, масштабування системи на великі об'єкти, що мають численні робочі платформи і камери, вимагає розробки нових[9].

1.4 Постановка задачі

Метою магістерської дисертації є вдосконалення процесу передачі зображень від камери роботизованої рухомої платформи до серверу, зокрема для підвищення швидкодії системи та зменшення навантаження на сервер. Це буде досягатися за допомогою оптимізації зображень, що дозволить зменшити обсяг даних, які передаються, зберігаючи при цьому необхідну якість зображення. Окрім того, зменшення кількості кадрів за секунду також позитивно позначиться на ефективності передачі даних і на зменшенні навантаження на систему, дозволяючи використовувати ресурси більш ефективно[17].

Основними завданнями, які мають бути виконані для досягнення цієї мети, є:

1. Аналіз існуючих функціональних можливостей і аналогів систем захисту приміщень: необхідно вивчити найпоширеніші технології відеоспостереження та охорони, виявити їх сильні та слабкі сторони, порівняти різні підходи, які використовуються в сучасних охоронних системах. Це дозволить зрозуміти, які технології є найбільш ефективними, а також виявити існуючі проблеми, які потребують вирішення.

2. Оцінка та вибір оптимальних форматів зображень для стиснення: для ефективної передачі зображень необхідно вибрати найбільш підходящий формат стиснення, що дозволить знизити розмір файлів без значних втрат у якості зображень. Вибір формату має враховувати швидкодію процесу стиснення, зручність у використанні та рівень компресії. Важливо знайти баланс між мінімальним розміром файлів і збереженням необхідної чіткості і деталізації зображень.

3. Експериментальне дослідження методів оптимізації зображень: для оцінки ефективності різних методів стиснення планується провести серію експериментів, у яких будуть порівнюватися різні алгоритми стиснення зображень. Кожен метод буде оцінюватися за кількома параметрами:

швидкість стиснення, розмір файлу після стиснення, а також якість відновленого зображення. Вибір найкращого методу буде ґрунтуватися на результатах тестів, де враховуватиметься баланс між мінімізацією розміру файлів та збереженням їх якості.

4. Розробка програмного забезпечення для оптимізації зображень: на основі результатів дослідження буде створено програмне забезпечення, яке здійснюватиме процес стиснення зображень перед їх передачею на сервер. Це програмне забезпечення повинно бути швидким, ефективним та надійним, дозволяючи автоматично оптимізувати зображення, що надходять від відеокамери, і передавати їх на сервер для подальшого зберігання та обробки.

5. Розробка веб-додатку для перегляду файлів, збережених на сервері: для забезпечення доступу до збережених файлів та відео, що надходять від камери, планується розробити зручний веб-додаток. Користувачі матимуть змогу через цей додаток переглядати зображення, що зберігаються на сервері, отримувати доступ до відеоархівів і здійснювати налаштування параметрів камери в режимі реального часу. Веб-додаток також повинен бути оптимізованим для роботи на різних пристроях, включаючи ПК, ноутбуки та мобільні телефони.

Для ефективної реалізації поставленої мети необхідно ретельно проаналізувати сучасні системи відеоспостереження. На сьогодні на ринку представлено чимало різних камер і систем, які мають широкий спектр функціональних можливостей. Однак жодна з них не є ідеальною, і кожна має свої переваги та недоліки, які варто врахувати.

Однією з таких систем є Arlo Pro4 — популярна камера відеоспостереження, яка має високу якість зйомки та багатий набір функцій. Проте вона має кілька значних недоліків. Основним є необхідність придбання базової станції, що збільшує загальні витрати на систему. Крім того, для доступу до хмарного сховища потрібно підписуватися на платні послуги, що також додає додаткові витрати на експлуатацію. Існує й обмеження на

використання системи лише через мобільний додаток, при цьому доступ до камер з комп'ютера неможливий, що створює певні незручності. Незважаючи на це, система Arlo Pro4 має високу якість зйомки, вбудовану систему виявлення руху та можливість нічної зйомки, що робить її ефективним рішенням для багатьох користувачів.

Інша система, яка була розглянута, забезпечує безкоштовне зберігання файлів у хмарному сховищі, що є великим плюсом у порівнянні з попередньою системою. Вона також має зручний мобільний додаток з можливістю перегляду відео та налаштування параметрів зйомки в реальному часі. Однак основний недолік цієї системи полягає в обмеженій швидкості відео в 15 кадрів за секунду, що може не бути достатньо для тих, хто потребує високої швидкості відеозапису для аналізу швидких подій. Крім того, наявність перерв у відео потоці може вплинути на якість спостереження.

Враховуючи переваги та недоліки цих систем, поставлені завдання передбачають оптимізацію процесу стиснення та передачі зображень від відеокамер, що допоможе значно полегшити навантаження на сервери і забезпечить високий рівень захисту приміщень без значних витрат на додаткові ресурси.

2 ВИБІР ФОРМАТУ ЗОБРАЖЕННЯ ТА РІВНЯ ЇХ СТИСНЕННЯ

2.1 Аналіз форматів зображень

Існує велика кількість різних форматів зображень, кожен з яких має свою унікальну структуру та специфічне призначення. Формати зображень можна поділити на дві основні категорії: растрові та векторні. Кожен із цих типів має свої переваги та недоліки в залежності від вимог до якості, розміру і типу контенту.

Векторні формати зображень, як правило, ідеально підходять для простих геометричних фігур, таких як логотипи, текстові елементи або значки. Вони створюються за допомогою математичних рівнянь, що описують лінії, криві та інші форми. Це дає змогу масштабувати зображення без втрати якості, що є їх основною перевагою[16]. Наприклад, збільшення розміру векторного зображення не призводить до розмиття або пікселізації, оскільки зображення визначається математичними формулами, а не набором пікселів. Така властивість робить векторні формати ідеальними для використання на екранах з високою роздільною здатністю та у випадках, коли необхідн забезпечити чіткість зображення при різних розмірах.

Проте, векторні зображення не підходять для передачі складних деталей, таких як фотографії або сцени з багатьма кольорами та текстурами. Вони можуть бути неефективними для таких типів зображень через складність опису кожного елемента у вигляді векторних примітивів, що в результаті призводить до великих обсягів файлів або навіть до того, що зображення не виглядатиме природно чи "фотореалістично". У таких випадках, більш підходящими будуть растрові формати, такі як PNG, JPEG або WebP[16].

Растрові зображення, на відміну від векторних, складаються з пікселів. Кожен піксель містить інформацію про колір і яскравість, що дозволяє створювати дуже детализовані зображення. Однак, як і будь-який растровий

формат, при збільшенні або зміні масштабу зображення можуть з'являтися артефакти, такі як пікселізація, коли пікселі стають видимими (Рис.2.1). Це обмежує їх використання при високих роздільних здатностях або коли потрібно змінювати розміри зображення без втрати якості.

Різниця між растровими та векторними форматами також виражається в обсязі даних. Векторне зображення зазвичай займає менше місця в порівнянні з растровим, особливо коли йдеться про прості геометричні форми. Однак для фотографій та складних зображень растрові формати можуть бути більш ефективними, оскільки вони здатні точно передати всі кольори та деталі, які є на зображенні[14].

Звичайно, вибір формату зображення залежить від вимог конкретного застосування. Наприклад, якщо необхідно зберігати або працювати з логотипами, схемами або графікою, яка повинна залишатися чіткою при будь-якому масштабі, векторні формати, такі як SVG або EPS, будуть кращим вибором. Якщо ж йдеться про фотографії або інші складні зображення, де важлива точність передачі кольору та детальності, варто вибрати растрові формати, такі як PNG або JPEG.

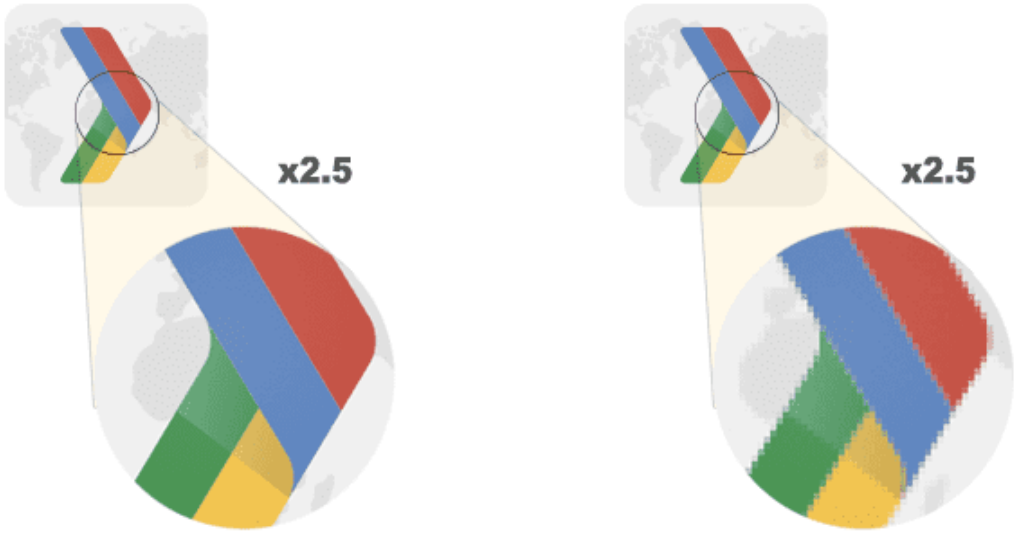


Рисунок 2.1 – Збільшене векторне та растрове зображення

З рисунку 2.1 можна зробити наступні висновки:

- Векторна графіка використовує лінії, точки та багатокутники для подання зображення;

- Растрова графіка зображує об'єкти шляхом кодування окремих значень кожного пікселя в прямокутній сітці.

У кожного формату зображення є свої сильні та слабкі сторони, які визначають їхню ефективність у різних умовах. Векторні формати є оптимальними для простих геометричних зображень, таких як логотипи, текст чи значки. Вони дозволяють зберігати чіткість і точність при будь-якому масштабуванні, що робить їх ідеальним вибором для високоякісних екранів та для графіки, що повинна зберігати чіткість при зміні розміру. Це також забезпечує високу ефективність для мультимедійних активів, які використовуються в мобільних додатках і веб-дизайні, оскільки такі формати не втрачають у якості навіть при великому збільшенні.

Проте, векторні формати не підходять для відтворення складних сцен, таких як фотографії, оскільки для їх точного опису потрібен значний обсяг даних. Наприклад, формат SVG може мати величезні файли при спробах описати фотографічні деталі, і навіть у цьому випадку зображення не буде мати "фотореалістичного" вигляду. В таких ситуаціях найкращим вибором є растрові формати, такі як PNG, JPEG або WebP, оскільки вони здатні передавати деталі з високою точністю та багатством кольорів[1].

Однак, растрові зображення мають суттєві обмеження в плані масштабування та збереження роздільної здатності. При спробах змінити розмір растрового зображення можуть виникати спотворення, пікселізація або розмитість, що значно погіршує якість. Це є наслідком того, що растрове зображення складається з пікселів, і при зміні масштабу ці пікселі можуть почати "розповзатися", знижуючи загальну чіткість. Щоб уникнути таких проблем, часто зберігають кілька версій одного і того ж зображення з різною

роздільною здатністю, забезпечуючи оптимальну якість для різних пристроїв (Рис.2.2).

Зараз дуже важливо також враховувати потреби безпеки у цифрових системах при обробці зображень. Наприклад, зображення в Інтернеті можуть бути вразливими до різних атак, таких як введення шкідливих кодів через зловмисно змінені метадані або навіть через самі зображення. Формати, такі як PNG або WebP, мають функції для інтеграції цифрових підписів або шифрування, що допомагає захистити зображення від несанкціонованого доступу або змін. Крім того, деякі системи безпеки використовують стеганографію для приховування важливої інформації в зображеннях, що дозволяє передавати дані в зашифрованій формі, що є важливим у випадках, коли необхідно захистити конфіденційність зображення[15].

Ще однією важливою особливістю сучасних систем безпеки є захист від атак, що спрямовані на зображення через їх метадані. У деяких випадках, метадані можуть містити важливу інформацію про зображення, таку як місце його створення, дата, модель камери тощо, що може бути використано для зловмисних цілей. Тому важливо забезпечити належний контроль над метаданими та використовувати спеціальні інструменти для їх очищення перед обробкою або публікацією зображень[12].

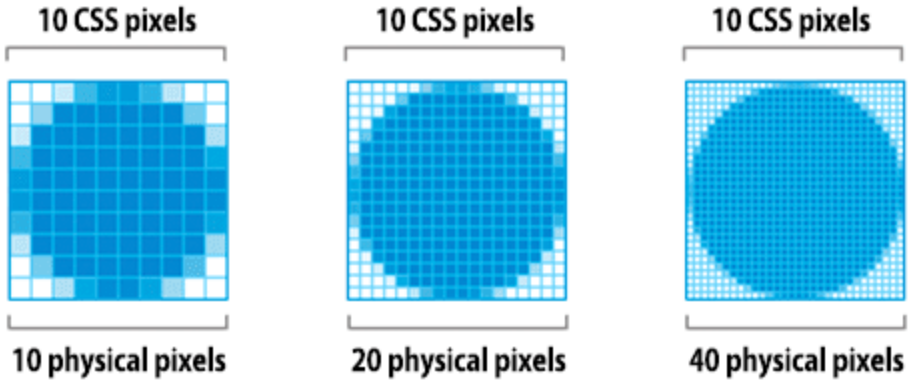


Рисунок 2.2 – Різниця між CSS-пікселями і пікселями пристрою.

Екрани з високою роздільною здатністю (HiDPI) забезпечують чудову якість відображення, що є особливо важливим для забезпечення чіткості зображень на різних пристроях. Однак цей тип екранів приносить із собою певні виклики, зокрема необхідність забезпечити достатню деталізацію зображень, щоб скористатися більшою кількістю пікселів пристрою[3]. Векторні зображення ідеально підходять для таких екранів, оскільки вони можуть бути масштабовані до будь-якої роздільної здатності без втрати якості або чіткості. Векторна графіка використовує математичні формули для представлення зображень, тому не має значення, скільки пікселів містить екран — зображення завжди буде мати чіткі контури та акуратний вигляд.

З іншого боку, растрові зображення створюються на основі пікселів, і при збільшенні роздільної здатності екрана потрібно створювати більш детальні зображення. Растрові формати мають обмеження, оскільки при збільшенні кількості пікселів для забезпечення високої якості зображень розмір файлу зростає, що збільшує вимоги до пам'яті та швидкості обробки даних[6]. Коли екран з високою роздільною здатністю подвоює кількість пікселів, це збільшує розмір зображення в чотири рази, і для того, щоб зберегти якість, потрібно підготувати кілька варіантів зображень з різною роздільною здатністю (Табл.2.1). Такий підхід дозволяє забезпечити високу якість відображення на екранах з різною кількістю пікселів, але одночасно потребує більше ресурсів для обробки і зберігання даних.

Таблиця 2.1 – Розмір растрового зображення

Роздільна здатність екрана	Загальна кількість пікселів	Розмір нестисненого файлу (4 байти на піксель)
1x	100 x 100 = 10,000	40 000 байт
2x	100 x 100 x 4 = 40,000	160,000 байт
3x	100 x 100 x 9 = 90,000	360,000 байт

Незважаючи на ці труднощі, важливо правильно вибирати формат зображень, щоб забезпечити оптимальний баланс між якістю і розміром файлів. Векторні формати є найкращими для елементів графіки, таких як логотипи чи іконки, оскільки вони не залежать від роздільної здатності екрана. Це дозволяє зберігати високу чіткість зображень незалежно від розміру екрана. У випадках, коли необхідно працювати з більш складними зображеннями, такими як фотографії або детальні графіки, використовуються растрові формати. Для таких зображень рекомендується створювати кілька варіантів з різною роздільною здатністю, що дозволить адаптувати їх до різних пристроїв, зберігаючи при цьому хорошу якість.

У контексті роботи з зображеннями для екранів з високою роздільною здатністю вибір між форматом PNG, JPEG або WebP також має важливе значення. Формат PNG підходить для зображень, які потребують прозорості або високої якості без втрат, але його розмір може бути значним. JPEG є більш ефективним для фотографій, оскільки дозволяє зменшити розмір файлів завдяки сильному стисненню, однак це може знизити якість зображень при високому рівні стиснення[7]. WebP, зі свого боку, пропонує чудове стиснення з мінімальними втратами якості, що робить його ідеальним вибором для веб-зображень, де важливо швидке завантаження сторінок.

Загалом, для оптимізації відображення зображень на екранах з високою роздільною здатністю важливо враховувати не тільки тип зображень, але й формат, а також необхідність адаптації їх до різних роздільних здатностей. Це дозволить досягти найкращого балансу між якістю та ефективністю використання ресурсів, що є критично важливим для забезпечення хорошого користувацького досвіду.

2.2 Особливості різних форматів растрових зображень

Різні формати растрових зображень не тільки застосовують різні алгоритми стиснення, що можуть бути з втратами або без втрат, але й підтримують різні функції, такі як анімація та прозорість (Альфа-канали). Вибір оптимального формату для конкретного зображення залежить від того, які візуальні результати та функціональні вимоги необхідно забезпечити для користувача або проекту[15]. Це поєднання візуальних ефектів і практичних вимог до обробки та відображення зображень, як показано в таблиці 2.2.

Таблиця 2.2 – Формати та візуальні ефекти

Формат	Прозорість	Анімація	Підтримка браузерів
PNG	Так	Ні	Усі
JPEG	Ні	Ні	Усі
WebP	Так	Так	Тільки сучасні

Серед найбільш поширених форматів для растрових зображень є два універсально підтримуваних: PNG та JPEG. Обидва ці формати використовуються у різних контекстах, кожен із яких має свої переваги. PNG особливо ефективний для зображень, що потребують прозорості або чітких контурів, таких як логотипи, іконки чи графіки з текстом[15]. Цей формат використовує стиснення без втрат, що дозволяє зберегти високу якість зображення. Однак файли PNG зазвичай мають більший розмір порівняно з іншими форматами.

JPEG, у свою чергу, є стандартом для фотографій та інших зображень, де важлива кольорова гама, але не настільки важлива прозорість або збереження кожної деталі без втрат. Формат JPEG використовує стиснення з втратами, що дозволяє зменшити розмір файлів, але це може призвести до зниження якості зображення, особливо при високому рівні стиснення.

Сучасні веб-браузери також підтримують новий формат WebP, який поєднує переваги обох попередніх форматів, пропонуючи краще стиснення,

зберігаючи при цьому високу якість зображення. WebP підтримує стиснення як з втратами, так і без втрат, а також дозволяє використовувати прозорість (як у PNG) та анімацію (як у GIF). Оскільки WebP забезпечує кращу оптимізацію розміру файлів без значних втрат у якості, він є бажаним вибором для веб-ресурсів, особливо коли важливо забезпечити швидке завантаження сторінок[15].

Використання формату WebP є рекомендованим у випадках, коли необхідно досягти максимальної компресії і зберегти хорошу якість зображень, зокрема для веб-ресурсів. Проте, у разі сумісності з більш старими браузерами, можна використовувати WebP як додатковий формат разом з іншими, наприклад PNG або JPEG, для забезпечення сумісності з усіма пристроями та платформами.

2.2.1 JPEG формат

JPEG (або JPG) – це один з найбільш популярних форматів для зберігання растрових зображень, особливо для фотографій. Він використовує метод стиснення з втратами, що дозволяє значно зменшити розмір файлів, але з певною втратою якості зображення. Формат JPEG ідеально підходить для зображень, де немає потреби зберігати всі деталі, що дозволяє зменшити обсяг файлу без суттєвих візуальних втрат. Під час стиснення видаляються малопомітні для людського ока деталі, що дозволяє значно зменшити обсяг файлу[8].

У форматі JPEG можна налаштовувати рівень якості зображення, що дає можливість вибирати оптимальний баланс між якістю та розміром файлу. Зниження якості веде до видалення дрібних деталей і появи шумів на зображенні, що в свою чергу дозволяє значно зменшити розмір файлу. В залежності від налаштувань компресії, JPEG може забезпечити стиснення з коефіцієнтом від 2:1 до 100:1. Однак слід пам'ятати, що чим вище коефіцієнт

стиснення, тим більше втрачається якості зображення, а також виникає ефект "пікселізації" і розмитості, особливо при різкому збільшенні зображення.

JPEG підтримує 24-бітні кольори для RGB (червоний, зелений, синій) і 8-бітний градаційний (Grayscale) режим. Це дозволяє створювати кольорові фотографії з великим діапазоном кольорів, що робить формат популярним для збереження високоякісних зображень, таких як портрети, пейзажі та інші фотографії. Однак формат не підтримує прозорість або альфа-канали, тому він не підходить для зображень, які вимагають прозорих ділянок, таких як графіка для веб-дизайну або іконки.

Ще однією важливою характеристикою JPEG є інтегрована підтримка EXIF (Exchangeable Image File Format), що дозволяє зберігати метадані зображення. EXIF включає в себе багато корисної інформації, такої як: марка та модель камери, дата та час зйомки, налаштування камери (витримка, діафрагма, баланс білого, чутливість ISO), географічні координати місця зйомки, використання спалаху та інші параметри. Це особливо корисно для фотографів, які хочуть зберегти всі деталі процесу зйомки, а також для організації та класифікації зображень[14].

Завдяки підтримці EXIF, зображення можуть зберігати інформацію про камеру та знімки, що полегшує їх подальшу обробку і аналіз. Наприклад, на основі метаданих можна швидко визначити параметри зйомки, що допомагає в корекції експозиції або при повторному використанні фотографій.

Розширення для файлів JPEG — .jpg і .jpeg — працюють абсолютно однаково, і вибір між ними зазвичай залежить від програмного забезпечення або переваг користувача. У різних операційних системах або програмах можна зустріти обидва варіанти, але вони позначають один і той самий формат.

Попри всі свої переваги, формат JPEG має свої обмеження. По-перше, стиснення з втратами призводить до постійної втрати якості зображення, навіть якщо файл відкривається і зберігається без змін. Крім того, формат не

підходить для зображень з високим рівнем деталізації або тих, що потребують високої якості, таких як деякі види графіки або зображення з великою кількістю тексту. В таких випадках краще використовувати інші формати, такі як PNG або TIFF, які підтримують стиснення без втрат.

В результаті JPEG є оптимальним вибором для фотографій та зображень, де важлива швидкість завантаження і зменшення розміру файлу, але при цьому не вимагається високої якості відтворення кожної деталі. Це робить формат дуже популярним для веб-графіки, фотозйомки та зберігання великих колекцій зображень[14].

2.2.2 PNG формат

PNG (Portable Network Graphics) – це формат растрових зображень, який здобув популярність завдяки своїм перевагам в області безвтратного стиснення та підтримки прозорості. Формат PNG широко використовується для зберігання зображень, де важливо зберегти якість без втрат, зокрема для графіки, логотипів, діаграм та елементів інтерфейсу, де важлива чіткість країв та прозорість.

PNG 24 підтримує повний спектр кольорів з глибиною 24 біти, що дозволяє зберігати до 16 мільйонів кольорів у зображенні. Це ідеальний варіант для фотографій і складних графічних зображень, де важлива точність відтворення кольору[8]. У PNG 24 неможливо налаштувати ступінь стиснення, що дає йому перевагу при зберіганні зображень без втрат якості, проте стиснення PNG не так ефективне, як у JPEG, що призводить до більших розмірів файлів у порівнянні з останнім. Однак PNG дозволяє зберігати прозорість завдяки використанню альфа-каналу, що робить його ідеальним для веб-дизайну та створення графіки з прозорим фоном.

Важливою особливістю формату PNG є його здатність адаптувати розмір файлу через зменшення кількості кольорів у зображенні. Якщо для конкретного зображення не потрібно більше ніж кілька сотень кольорів,

можна зменшити палітру і зменшити розмір файлу, при цьому зберігаючи якість. Для цього використовується формат PNG 8, який підтримує лише 256 кольорів, що робить його більш компактним у порівнянні з PNG 24. Однак, при застосуванні PNG 8 до зображень з великою кількістю кольорів, стиснення вже буде з втратами, що призводить до ефекту пастеризації (зниження якості зображення через обмежену кількість кольорів).

До 2017 року PNG не підтримував збереження метаданих у форматі EXIF, проте ця можливість була реалізована в останніх оновленнях стандарту, що дозволяє зберігати інформацію про зображення, як-от дані про камеру та налаштування зйомки, хоча в цілому формат не є популярним для фотографій. В основному PNG використовується для графічних елементів, що вимагають прозорості або високу якість при збереженні на малому розмірі[4].

У порівнянні з JPEG, PNG менш ефективний для фотографій через більший розмір файлів, але має перевагу в збереженні прозорості та безвтратному стисненні, що робить його ідеальним для зображень з необхідністю відображення прозорих областей, таких як іконки, логотипи або графіка для веб-сторінок. В результаті, PNG є найбільш підходящим для графіки, яка вимагає чітких ліній і прозорих ділянок, але менш оптимальним для фотографій або зображень з великою кількістю кольорів.

Файли в форматі PNG мають розширення .png, а бітність (24 або 8 біт) записується в метадані файлу, не визначаючи розширення. Вибір між PNG 24 і PNG 8 залежить від вимог до якості та розміру файлу: PNG 24 забезпечує високу якість зображень, але може бути більшим за розміром, тоді як PNG 8 є компактнішим, але має обмеження по кількості кольорів[8].

2.2.3 WebP формат

WebP — це сучасний формат зображень, розроблений компанією Google у 2010 році. Його створення стало можливим завдяки придбанню технологій компанії On2 Technologies. WebP поєднує ефективне стиснення зображень як з втратами, так і без втрат, зберігаючи при цьому високу якість[14]. Цей формат дозволяє досягти чудових результатів у зменшенні розміру файлів при збереженні візуальної якості зображень, що робить його популярним для використання в Інтернеті та мобільних додатках.

WebP має два основні варіанти: WebP Lossy та WebP Lossless:

1. WebP Lossy використовує стиснення з втратами, що дозволяє значно зменшити розмір файлу за рахунок часткового видалення інформації з зображення. Однак формат дозволяє регулювати ступінь стиснення, що дає змогу вибирати компроміс між розміром файлу та якістю зображення. Ступінь втрат можна налаштувати таким чином, щоб зберігати баланс між розміром файлу і прийнятною якістю зображення.

2. WebP Lossless використовує стиснення без втрат, подібно до формату PNG. Цей тип WebP забезпечує вищу якість зображення при більших розмірах файлів порівняно з стисненням з втратами, але все одно здатен досягати значного зменшення розміру файлу порівняно з іншими форматами без втрат, як-от PNG.

Однією з основних переваг формату WebP є його здатність зменшувати розмір файлів значно більше, ніж традиційні формати JPEG та PNG. Зображення в форматі WebP Lossless можуть бути на 26% меншими за аналогічні зображення в PNG при збереженні повної деталізації. У свою чергу, зображення WebP Lossy можуть бути на 25-35% меншими за розміром порівняно з JPEG з еквівалентною якістю. Це робить WebP дуже привабливим для використання в Інтернеті, де швидкість завантаження та обмеження по розміру файлів мають велике значення.

Проте, незважаючи на свої переваги, формат WebP має кілька значних недоліків, основним з яких є погана сумісність з деякими програмами та браузерами. Більшість сучасних веб-браузерів, таких як Google Chrome, Opera, Firefox і Edge, вже підтримують формат WebP, що дозволяє ефективно використовувати його для веб-ресурсів. Однак Internet Explorer взагалі не підтримує WebP, а Safari має лише часткову сумісність з цим форматом, що обмежує його використання в певних екосистемах. Крім того, більшість популярних графічних редакторів не підтримують відкриття або редагування зображень у форматі WebP, що може створити труднощі для дизайнерів та користувачів, які працюють з такими файлами.

Загалом, WebP є надзвичайно ефективним форматом для збереження зображень у веб-середовищі, де важливе зменшення часу завантаження сторінок та оптимізація використання смуги пропускання. Проте, його застосування залишається обмеженим через сумісність з деякими браузерами та програмами.

2.2.4 MJPEG формат

MJPEG (Motion JPEG) — це формат кодування відео, який складається з серії JPEG-зображень, де кожен кадр відео кодується незалежно один від одного з використанням стандарту JPEG. Відповідно, відео в цьому форматі є простою послідовністю окремих JPEG-зображень, кожне з яких є самостійним і не залежить від попередніх чи наступних кадрів[11].

Цей формат найбільше використовується в цифрових камерах, системах спостереження, а також в медичних і промислових відео-системах, де важлива висока якість кожного кадру. Однією з головних переваг MJPEG є те, що кожен кадр відео зберігається окремо, що дозволяє отримати стоп-кадри високої якості без значних втрат. Крім того, MJPEG вимагає набагато менше обчислювальних ресурсів у порівнянні з іншими форматами відео, такими як

MPEG-4, оскільки не потребує складних алгоритмів для зберігання часової залежності між кадрами.

Апаратна реалізація MJPEG є більш простою і дешевою, ніж у форматах, які використовують складніші алгоритми стиснення. Це робить MJPEG популярним у застосуваннях, де важлива доступність і зручність апаратного забезпечення, таких як відеоспостереження та прості відеокамери. Також формат MJPEG є зручним для редагування відео, оскільки кожен кадр незалежний, і зміни в одному кадрі не впливають на інші, що дозволяє легко маніпулювати відео без значних технічних складнощів.

Однак, попри всі переваги, MJPEG має низку недоліків. Одним із основних є відсутність єдиного стандарту для специфікації формату, що призводить до того, що різні розробники можуть реалізовувати MJPEG по-різному[11]. Це створює проблеми сумісності між різними програмами або пристроями, які використовують MJPEG, оскільки вони можуть мати різні інтерпретації формату.

Іншим важливим недоліком є відносно низька ефективність стиснення, оскільки MJPEG не передбачає зменшення часової надмірності між кадрами. Замість того, щоб враховувати інформацію, спільну між сусідніми кадрами (як це роблять формати на основі MPEG), MJPEG зберігає кожен кадр повністю. Це призводить до того, що ступінь стиснення в MJPEG значно нижчий — зазвичай він знаходиться в межах 5-20 разів, що є менш ефективним у порівнянні з іншими алгоритмами, такими як MPEG-4. Відповідно, відеофайли в форматі MJPEG займають більше місця на диску.

Попри це, MJPEG залишається популярним для простих відео-систем, де потрібна висока якість кожного кадру, а також для ситуацій, де важлива низька складність обробки та редагування відео.

2.3 Рівні стиснення зображень

Зображення часто становлять більшість завантажених байтів на веб-сторінці та займають значний візуальний простір, тому їх оптимізація може призвести до значних заощаджень в обсязі даних і покращення продуктивності веб-сайтів. Чим менше байтів потрібно завантажити браузеру, тим менше пропускної здатності використовується клієнтом, і тим швидше браузер може завантажити та відобразити корисний вміст на екрані.

Оптимізація зображень поєднує мистецтво та науку: з одного боку, немає однозначного рішення для ідеального стиснення кожного зображення, з іншого — існує велика кількість методів і алгоритмів, що дозволяють значно зменшити розмір файлів[11]. Для досягнення найкращих результатів необхідно враховувати кілька факторів, таких як формат зображення, зміст даних, якість та розмір пікселів. Растрове зображення — це двовимірна сітка пікселів, кожен з яких має значення для червоного (R), зеленого (G), синього (B) та прозорості (A) каналів. У браузері кожен з цих каналів може мати 256 відтінків, що дає 8 бітів на канал або 32 біти (4 байти) на піксель.

Наприклад, для зображення розміром 100x100 пікселів розмір файлу можна обчислити наступним чином:

- Кількість пікселів: 100x100 = 10,000 пікселів;
- Розмір файлу: 10,000 пікселів × 4 байти = 40,000 байт;
- 40,000 байт / 1024 = 39 КБ.

При цьому незалежно від формату стиснення зображення, що використовується для передачі з сервера на клієнт, кожен піксель в браузері завжди займає 4 байти пам'яті. Це може стати важливим обмеженням для великих зображень або для пристроїв з обмеженою пам'яттю, наприклад, для бюджетних мобільних пристроїв.

У таблиці 2.3 наведена залежність розміру файлу від розміру зображення в пікселях, що дозволяє візуалізувати цю закономірність і допомагає розраховувати орієнтовний розмір файлу для різних зображень[12].

Таблиця 2.3 – Залежність розміру файлу від розміру зображення

Розміри	Пікселі	Розмір файлу
100 x 100	10 000	39 КБ
200 x 200	40 000	156 КБ
300 x 300	90 000	351 КБ
500 x 500	250 000	977 КБ
800 x 800	640 000	2500 КБ

Таблиця 2.3 ілюструє, як збільшується розмір файлу із збільшенням кількості пікселів у зображенні. Це важливо враховувати при виборі оптимального розміру та формату зображення для веб-сторінки чи мобільного додатку, оскільки занадто великі зображення можуть значно сповільнити завантаження і відображення сторінок.

Зображення на веб-сторінках та в мобільних додатках можуть значно впливати на швидкість завантаження сторінок та загальну продуктивність. Оптимізація зображень є важливим кроком у покращенні користувацького досвіду, оскільки чим менший розмір файлу, тим швидше відбудеться завантаження та рендеринг контенту[9]. Однак, не завжди можна просто зменшити розмір файлу за рахунок компромісу з якістю. Тому важливо правильно налаштувати процес стиснення, враховуючи різні аспекти зображення, його використання та вимоги до якості.

Основні параметри стиснення зображень

1. Тип стиснення:

Зображення можуть бути стиснуті за допомогою двох основних методів:

- Стиснення без втрат — цей метод дозволяє зберігати всі дані зображення без втрат якості, що важливо для збереження точності кольорів і деталей. Однак, стиснення без втрат, як правило, забезпечує менше зменшення розміру файлу, ніж стиснення з втратами. Приклад таких форматів: PNG, WebP (без втрат).

- Стиснення з втратами — тут частина даних викидається, що дозволяє значно зменшити розмір файлу за рахунок деякої втрати якості. Це підходить для зображень, де точність не критична. Приклад: JPEG, WebP (з втратами).

2. Розмір зображення:

Розмір зображення в пікселях є одним з головних факторів, що впливає на його розмір у байтах. Як зазначено раніше, зображення 100x100 пікселів має розмір файлу близько 39 КБ при використанні стандартних 32 біт на піксель. Якщо зображення має великий розмір (наприклад, 2000x2000 пікселів), його розмір буде в десятки разів більшим, що призведе до значного навантаження на пропускну здатність та пам'ять пристрою. Важливо оптимізувати розмір зображень для зручного завантаження, особливо на мобільних пристроях з обмеженою пам'яттю.

3. Глибина кольору:

Глибина кольору визначає, скільки бітів виділено для збереження кожного пікселя зображення. Для стандартних зображень це зазвичай 24 біти (8 біт на кожен з трьох каналів: червоний, зелений, синій). Якщо зображення має меншу глибину кольору, наприклад, 8 біт на піксель (256 кольорів), розмір файлу буде значно меншим. Однак така оптимізація підходить лише для певних типів зображень, наприклад, іконок або простих графіків.

4. Тип зображення:

Існують різні типи зображень, для яких підходять різні методи стиснення:

- Фотографії зазвичай краще стискати за допомогою формату JPEG або WebP з втратами, оскільки це дозволяє значно зменшити розмір файлу при збереженні достатньої якості.

- Логотипи та графіки з чіткими лініями краще зберігати у форматах PNG або SVG для забезпечення високої якості та чітких контурів без втрат.

5. Методи оптимізації:

- Ретушування зображень: Видалення непотрібних пікселів (наприклад, прозорих пікселів, які не впливають на відображення) може зменшити розмір файлу.

- Використання більш ефективних алгоритмів стиснення: Наприклад, замість використання стандартного JPEG можна використовувати сучасніші формати, такі як WebP, які забезпечують більшу ефективність стиснення при збереженні високої якості.

- Зменшення розмірів зображень: Вибір оптимальних розмірів зображень для відображення на веб-сторінці (наприклад, для мініатюр або адаптивних зображень) може зменшити обсяг переданих даних. Стиснення великих зображень для використання на мобільних пристроях також знижує навантаження на пам'ять і пропускну здатність.

Стиснення зображень має вирішальне значення для продуктивності веб-сайтів. Чим менші зображення, тим швидше вони завантажуються, що покращує користувацький досвід. Це особливо важливо для мобільних користувачів, де з'єднання з Інтернетом може бути повільним або нестабільним. Крім того, стиснення зображень допомагає зменшити навантаження на сервери та зберігання даних.

Згідно з дослідженнями, зменшення розміру зображення на веб-сторінці може скоротити час завантаження на кілька секунд, що може значно підвищити швидкість відгуку та задоволеність користувачів. Це також має позитивний вплив на SEO, оскільки пошукові системи надають перевагу швидким веб-сайтам.

Оптимізація зображень є важливою частиною процесу поліпшення продуктивності веб-сайтів. Для досягнення найкращих результатів потрібно ретельно вибирати формат стиснення та параметри оптимізації залежно від типу зображення, його використання та вимог до якості. Це дозволяє значно зменшити обсяг передаваних даних, покращити швидкість завантаження та користувацький досвід.

Зменшення розміру файлу зображення може бути важливим аспектом для швидкості завантаження графічних ресурсів, особливо для великих зображень. Одна з простих стратегій для цього — зменшити "бітову глибину" зображення, зменшивши кількість кольорів у палітрі. Якщо стандартно використовувати 8 біт на канал (що дає 256 значень на канал та 16 777 216 кольорів загалом), зменшення палітри до 256 кольорів дозволяє зекономити 2 байти на піксель — це дає 50% економії порівняно з вихідним форматом, де використовуються 4 байти на піксель.

На рис. 2.3 зліва направо показано зображення з різною бітовою глибиною[6]:

- 32-бітне (16 мільйонів кольорів),
- 7-бітне (128 кольорів),
- 5-бітне (32 кольори).

Рисунок 2.3 – Зменшення бітової глибини зображення

Для складних сцен із плавними колірними переходами (наприклад, небо або градієнти) важливо зберегти велику палітру кольорів, щоб уникнути

візуальних артефактів, таких як піксельність на 5-бітних зображеннях. З іншого боку, якщо зображення використовує обмежену кількість кольорів, велика палітра лише даремно витрачає біти.

Після оптимізації даних, які зберігаються в окремих пікселях, можна звернути увагу на сусідні пікселі, оскільки на зображеннях, особливо фотографіях, часто зустрічаються великі ділянки з однаковими кольорами – наприклад, рівні поверхні неба чи повторювані текстури. Це дає можливість ефективно використовувати цю закономірність у процесі стиснення. Один із способів досягти цього – застосувати дельта-кодування. Замість того щоб зберігати значення кожного пікселя окремо, можна зберігати різницю між сусідніми пікселями. Якщо пікселі схожі, різниця буде "нульовою", що дозволить зберігати лише один біт.

Також людське око по-різному сприймає кольори, що дає змогу оптимізувати кодування кольорів, зменшуючи або збільшуючи палітру кольорів у залежності від чутливості ока до певних відтінків. Оскільки пікселі утворюють двовимірну сітку, де кожен піксель має кілька сусідів, можна використати цей факт для подальшого удосконалення дельта-кодування. Замість того щоб враховувати лише найближчі сусідні пікселі, можна обробляти більші блоки пікселів і застосовувати різні налаштування для кодування цих блоків.

Для деяких типів даних, таких як вихідний код чи виконувані файли, важливо зберігати точність кожного біта, оскільки навіть незначна помилка може повністю змінити зміст файлу або призвести до його пошкодження[7]. Натомість, для таких даних, як зображення, аудіо чи відео, допускається використання компресії з втратами, оскільки людське око та інші органи чуття не здатні помітити всі зміни, що можуть виникнути при відмові від деякої частини інформації.

Оскільки ми не завжди здатні помітити всі деталі зображення, компресія з втратами дозволяє зменшити розмір файлів, зберігаючи при

цьому достатню якість для сприйняття. Це особливо важливо при роботі з великими файлами, де зменшення розміру без суттєвих втрат у якості є необхідним для зручності зберігання та передачі даних. Компресія з втратами і без втрат комбінується в стандартному процесі оптимізації зображень. Зазвичай, для досягнення найкращих результатів, зображення проходить дві фази обробки: спочатку застосовується фільтр з втратами, що усуває незначні піксельні дані, а потім відбувається безвтратна компресія, що зменшує розмір.

У випадку форматів з втратами, таких як JPEG, користувач може налаштувати параметр "якість", який регулює компресію в межах від 1 до 100. Це дає змогу контролювати компроміс між розміром файлу та видимими артефактами, що можуть виникнути при стисненні. Важливо зазначити, що якість зображення у різних форматах не завжди порівнянна, оскільки різні формати використовують різні алгоритми стиснення.

Типові формати зображень можна поділити на два основні типи: векторні та растрові. Векторні формати ідеально підходять для зображень, що складаються з простих геометричних форм, таких як логотипи або текст, оскільки вони не втрачають якості при збільшенні роздільної здатності[14]. Проте, для складних зображень, таких як фотографії, де присутня велика кількість дрібних деталей і плавних переходів кольору, використовуються растрові формати. Серед них найбільш популярними є PNG, JPEG та WebP, кожен з яких має свої переваги та обмеження.

PNG пропонує високу якість і дуже велику роздільну здатність, однак має великий розмір файлів і не підтримує стиснення з втратами. JPEG використовує алгоритм стиснення з втратами, дозволяючи налаштувати рівень компресії для досягнення оптимального балансу між якістю та розміром файлу. WebP, розроблений для веб-застосунків, забезпечує кращу ефективність стиснення, зменшуючи розмір файлів на 25-35% порівняно з JPEG, зберігаючи при цьому схожу якість зображення.

3 ВИБРІ ТЕХНОЛОГІЙ ДЛЯ ПРОЦЕСУ ОПТИМІЗАЦІЇ ЗОБРАЖЕНЬ, РОЗРОБКИ СЕРВЕРУ ТА ВЕБ ДОДАТКУ

У даній роботі важливим аспектом є оптимізація розміру зображень, передача їх від роботизованої платформи до сховища та забезпечення доступу користувача до збережених файлів через веб-додаток. Цей процес включає кілька ключових етапів, таких як отримання зображень, їх стиснення, передача через сервер і зберігання на сервері, а також надання доступу до зображень кінцевому користувачу.

Для реалізації серверної частини проекту обрано платформу Node.js, що є потужним середовищем для розробки високопродуктивних веб-серверів. Однією з головних переваг Node.js є його асинхронний і подієвий характер, що дозволяє ефективно обробляти великі обсяги одночасних запитів, що особливо важливо для обробки медіа-файлів, таких як зображення. Використання Node.js також забезпечує інтеграцію з іншими сучасними технологіями, що полегшує реалізацію веб-додатків і сервісів.

Щоб керувати залежностями проекту та встановлювати необхідні бібліотеки, буде використано NPM (Node Package Manager). Це зручний менеджер пакетів для Node.js, що дозволяє швидко додавати й оновлювати модулі, необхідні для роботи серверу та клієнтської частини.

Для зберігання даних про користувачів та їх доступ до серверів обрано MongoDB. Це NoSQL база даних, яка дозволяє зберігати дані у вигляді документів (JSON-подібних об'єктів), що забезпечує високу гнучкість при зберіганні структурованої та неструктурованої інформації. MongoDB є ідеальним вибором для проектів, де дані можуть змінюватися або мати різні формати, а також для роботи з великими обсягами інформації, що є характерним для зберігання зображень та метаданих про користувачів.

Для створення клієнтської частини веб-додатку обрано стандартні веб-технології, зокрема:

- HTML (HyperText Markup Language) – мова розмітки документів, що дозволяє структурувати інформацію на веб-сторінках.

- CSS (Cascading Style Sheets) – каскадні таблиці стилів для надання веб-додатку візуальних властивостей і створення привабливого інтерфейсу користувача.

HTML та CSS дозволяють створити простий і зручний інтерфейс для взаємодії з користувачем.

Для отримання зображень від сервера та представлення їх у веб-додатку буде використовуватися JavaScript та WebSocket. JavaScript є основною мовою програмування для створення динамічного контенту на веб-сторінках, а WebSocket дозволяє здійснювати двосторонню комунікацію між клієнтом і сервером в реальному часі. Це ідеально підходить для передачі відео потоку або інших медіафайлів, що потребують негайної доставки та оновлення без необхідності постійно перезавантажувати сторінку.

Для забезпечення безпеки та контролю доступу до серверів буде використовуватися стандарт JWT (JSON Web Token). JWT дозволяє здійснювати автентифікацію та авторизацію користувачів, надаючи їм доступ до веб-додатка через передачу токенів, що містять закодовану інформацію про користувача. Це дозволяє безпечно передавати дані про сесії користувачів, забезпечуючи контроль доступу без необхідності зберігати сесії на сервері. Зазначимо:

1. Node.js забезпечує ефективне і масштабоване середовище для обробки запитів до серверу, що є важливим для роботи з медіафайлами.

2. MongoDB обрана через свою гнучкість у зберіганні структурованих і неструктурованих даних, що робить її відмінним вибором для зберігання даних користувачів і метаданих.

3. HTML/CSS/JavaScript використовуються для створення простого й ефективного інтерфейсу веб-додатку, що дозволяє користувачам переглядати зображення та відео потоки.

4. WebSocket дозволяє реалізувати зв'язок у реальному часі, що є критично важливим для передачі відео потоку.

5. JWT забезпечує безпечну верифікацію користувачів, що дозволяє здійснювати доступ до ресурсів тільки авторизованими особами.

Ці технології створюють основу для ефективної роботи з зображеннями, їх обробки, передачі та зберігання, а також забезпечують безпечний доступ до даних для користувачів.

3.1 Node.JS та npm

Основою для створення веб-сервера є програмна платформа Node.js.

Node.js — це потужна платформа з відкритим кодом, побудована на рушії Google V8, який компілює JavaScript у машинний код, що значно прискорює виконання програм. Node.js спеціалізується на створенні високопродуктивних мережевих застосунків, зокрема серверів і клієнтів, написаних на JavaScript. Вона надає можливість JavaScript взаємодіяти з пристроями введення-виведення за допомогою API (Application Programming Interface).

API — це набір інтерфейсів для взаємодії різних програм. Наприклад, API може бути частиною інтернет-протоколу, такого як HTTP, і визначає, як клієнт і сервер повинні взаємодіяти між собою.

Однією з ключових особливостей Node.js є її здатність працювати не лише з сервеними додатками для обробки веб-запитів, але й створювати клієнтські програми. В основі Node.js лежить асинхронна модель з неблокуючим введенням-виведенням (I/O), що дозволяє обробляти велику кількість запитів одночасно без затримок. Ця модель є дуже ефективною для високонавантажених веб-застосунків.

Подійно-орієнтоване програмування є ще однією основною концепцією платформи Node.js. У класичних синхронних системах програма виконує запити по черзі, чекаючи на завершення кожного перед тим, як перейти до

наступного. У Node.js застосовується асинхронне виконання запитів, де програма не чекає на результат попереднього запиту, а одразу переходить до виконання наступного блоку коду.

Приклад синхронного виконання:

```javascript
var result = db.query("select * from users");
```

У цьому випадку програма чекає завершення запиту до бази даних, перш ніж перейти до наступної операції.

Приклад асинхронного виконання:

```javascript
db.query("select * from users", function(result) {
 // Обробка результату
});
```

У цьому прикладі програма відразу переходить до виконання наступного коду, не чекаючи на результат запиту. Водночас результат обробляється в окремій функції, переданій у вигляді callback.

Callback-функції — це функції, що передаються в інші функції як параметри і викликаються після завершення асинхронних операцій. Вони дозволяють ефективно обробляти результати операцій, не блокуючи виконання основного потоку програми.

Таке асинхронне виконання дозволяє Node.js бути надзвичайно швидким і ефективним у обробці великої кількості запитів, що робить платформу ідеальною для створення масштабованих веб-систем і серверів реального часу.

Для полегшення керування залежностями та бібліотеками, Node.js використовує npm (Node Package Manager). npm дозволяє легко додавати до проекту різні пакети (бібліотеки), необхідні для роботи веб-сервера або

клієнтського додатку, зокрема для роботи з базами даних, обробки зображень та інших задач.

Наприклад, для підключення до MongoDB можна використовувати npm пакет:

```bash
npm install mongodb
```

Це дозволяє швидко інтегрувати необхідні бібліотеки і керувати їх версіями в проекті. npm також надає функціональність для оновлення пакетів і видалення непотрібних залежностей, що робить процес розробки більш зручним і керованим.

Таким чином, Node.js разом з npm є потужним інструментом для створення високопродуктивних веб-систем, що забезпечує ефективну обробку запитів та гнучке керування залежностями.

Callback-функція в JavaScript — це функція, яка передається іншій функції як аргумент і виконується після завершення роботи цієї функції. У JavaScript функції є об'єктами, тому вони можуть бути передані як параметри в інші функції або повернуті як результат. Така здатність дозволяє функціям взаємодіяти одна з одною і створює можливість організовувати асинхронне виконання коду.

Однією з важливих концепцій, яка використовує callback-функції, є асинхронне виконання. JavaScript підтримує модель подієво-орієнтованого програмування, де замість того, щоб чекати результату операції і блокувати виконання інших частин програми, можна передати функцію, яка буде викликана, коли операція завершиться. Це дозволяє програмі продовжувати виконання інших завдань без затримок, забезпечуючи ефективність та швидкодію.

```
function first(){
  console.log(1);
}
function second(){
  console.log(2);
}
first();
second();
```

Рисунок 3.1 – Приклад виконання синхронних команд

У JavaScript є два основні типи виконання коду: синхронне та асинхронне. При синхронному виконанні програма виконує кроки один за одним, чекаючи завершення кожної операції (Рис.3.1). Однак асинхронне виконання дозволяє здійснювати операції одночасно, не блокуючи основний потік програми. Для цього використовуються callback-функції, які спрацьовують тільки після того, як завершиться виконання певної операції або події.

Таке виконання надає більшу гнучкість у розробці, дозволяючи створювати програми, що працюють у реальному часі, без необхідності затримувати виконання основної логіки програми. Технічно, callback-функції можуть бути використані для обробки результатів запитів до баз даних, взаємодії з API, обробки користувацьких подій та інших довготривалих операцій, де важливо не блокувати виконання інших частин програми.

Якщо функція `first` містить код, який не може виконатися негайно, наприклад, запит до API, де потрібно чекати відповіді, ми можемо змоделювати таку ситуацію за допомогою функції `setTimeout`. Ця функція дозволяє відстрочити виконання коду на заданий час, таким чином імітуючи затримку, як це відбувається при виконанні запитів до зовнішніх серверів.

У цьому випадку, ми додаємо затримку на 500 мілісекунд, що імітує процес відправлення запиту та очікування відповіді від API. Завдяки цьому ми отримуємо ситуацію, де замість того, щоб виконувалася функція `first` і лише після неї — `second`, обидві функції працюють асинхронно. Проте, через затримку, виведення результатів буде не таким, як очікується, а порядок виведення буде порушений (Рис.3.2).

```
function first(){
  setTimeout( function(){
    console.log(1);
  }, 500 );
}
function second(){
  console.log(2);
}
first();
second();
```

Рисунок 3.2 – Виконання синхронних команд з часовою затримкою

Замість того, щоб обидві функції виконувалися послідовно, виведення виглядатиме так: спочатку виводиться результат від функції `second`, а лише після 500 мілісекунд — результат від `first`. Це свідчить про те, що програма не чекає на виконання функції `first`, а переходить до виконання наступного коду, поки чкає на відповідь від API.

Для того, щоб правильно обробити такі ситуації, де необхідно зберегти порядок виконання команд, використовуються callback-функції. Вони дозволяють точно вказати, яка функція має виконатися після того, як завершиться інша, що дає можливість забезпечити правильну послідовність дій у програмі.

Node.js, як платформа для виконання JavaScript на сервері, має велику кількість вбудованих модулів, що забезпечують функціональність для роботи з мережею, файловою системою та іншими ресурсами. Завдяки асинхронній природі Node.js, сервер може обробляти запити без необхідності блокувати виконання програми в очікуванні на результат кожної операції. Наприклад, запит до бази даних або стороннього API може зайняти певний час, але це не заважає серверу обробляти інші запити, поки чекає на відповідь.

У зв'язку з цим, асинхронне програмування стає важливим аспектом розробки веб-серверів на Node.js. Одним із способів обробки таких асинхронних операцій є використання callback-функцій (Рис.3.3), які виконуються після завершення основної операції, наприклад, отримання відповіді від API або виконання запиту до бази даних.

Ще одним важливим моментом є робота з подіями. Node.js використовує подієво-орієнтовану модель програмування, де різні частини програми можуть «слухати» події, що виникають в результаті певних дій користувача або операцій в системі. Це дозволяє значно скоротити час очікування на обробку запитів, оскільки програма може реагувати на події в реальному часі. Цей підхід також допомагає ефективно використовувати ресурси, оскільки програма не блокує виконання на час виконання однієї операції.

Для демонстрації роботи таких механізмів, зокрема обробки запитів користувачів, можна використовувати простий веб-сервер на Node.js, який реагує на HTTP запити і повертає відповідь (Рис.3.4). Це є основою для більш складних додатків, де потрібно обробляти різноманітні запити від клієнтів.

В результаті, Node.js стає ідеальним вибором для створення високопродуктивних веб-додатків і серверів, оскільки дозволяє ефективно працювати з великою кількістю одночасних підключень завдяки своїй асинхронній та подієво-орієнтованій архітектурі.

```
function first(callback){
  setTimeout( function(){
    console.log(1);
    callback();
  }, 500 );
}
function second(){
  console.log(2);
}
first(second);
```

Рисунок 3.3 – Використання callback-функції

```
var http = require('http'); // Завантажуємо модуль http

// Створюємо web-сервер і вказуємо функцію обробки запиту
var server = http.createServer(function (req, res) {
    console.log('Початок обробки запиту');
    // Передаємо код відповіді і заголовки
    res.writeHead(200, {
        'Content-Type': 'text/plain; charset=UTF-8'
    });
    res.end('Hello world!');
});

// Запускаємо web-сервер
server.listen(1991, "127.0.0.1", function () {
    console.log('Сервер запущено за адресою http://127.0.0.1:1991/');
});
```

Рисунок 3.4 – Приклад веб-серверу

Node.js включає в себе кілька вбудованих модулів, а також надає можливість працювати з власним менеджером пакунків NPM (Node Package Manager). NPM є основним інструментом для керування пакетами в екосистемі Node.js та JavaScript. Це дозволяє розробникам зручно встановлювати, оновлювати та видаляти пакунки, які містять корисний

функціонал, наприклад, для роботи з базами даних, HTTP протоколами, файловими системами та іншими завданнями.

Однією з основних переваг NPM є можливість зручного додавання сторонніх бібліотек і модулів, створених іншими розробниками. Для цього достатньо скористатися командою:

```
npm install <packagename>
```

Ця команда встановить модуль, вказаний в `<packagename>`, в проект.

Також для отримання інформації про доступні пакунки можна використовувати команду:

```
npm search
```

Ця команда дозволяє здійснити пошук модулів, що доступні для встановлення, і дізнатися короткий опис кожного з них. За допомогою цієї ж команди можна виконати фільтрацію за конкретними параметрами, наприклад, за назвами або описами модулів.

NPM дозволяє керувати як локальними залежностями, що потрібні для конкретного проекту, так і глобальними залежностями, що можуть бути використані в будь-якому проекті на системі. Важливою частиною кожного проекту є файл `package.json`, який містить інформацію про проект, зокрема його назву, версію, а також перелік залежностей, які потрібно встановити. Після налаштування цього файлу можна виконати команду:

```
npm install
```

Вона автоматично завантажить і встановить усі зазначені пакунки в папку проекту, що дозволяє швидко налаштувати середовище для розробки та використання необхідних бібліотек.

Таким чином, NPM є важливим інструментом для управління залежностями в проектах на Node.js, значно спрощуючи процес їх встановлення та оновлення.

3.2 MongoDB

MongoDB також має вбудовану підтримку для роботи з великими обсягами даних завдяки індексації та агрегованим запитам, які дозволяють виконувати складні операції з даними, не погіршуючи продуктивність. Наприклад, агрегатні операції в MongoDB дозволяють комбінувати, фільтрувати, сортувати та обчислювати дані без необхідності витягувати їх з бази в додаток, що значно підвищує ефективність обробки великих масивів інформації.

Для роботи з даними MongoDB використовує мову запитів, яка є схожою на JSON (Рис.3.5), з підтримкою складних операцій та фільтрів, що дозволяє зручно працювати з різноманітними типами даних. Можна здійснювати фільтрацію на основі умов, які стосуються вкладених об’єктів, або здійснювати складні операції, як-от пошук за регулярними виразами, використання логічних операторів, таких як AND, OR та ін.

MongoDB підтримує масштабованість на декілька серверів завдяки механізму реплікації, який дозволяє мати кілька копій одних і тих самих даних на різних серверах. Це забезпечує високий рівень доступності даних і дозволяє створювати більш стійкі до збоїв системи, де навіть якщо один з серверів виходить з ладу, інші продовжують обробляти запити.

Однією з найпоширеніших операцій у MongoDB є запит на вибірку даних. Як правило, вибірка даних може здійснюватися за допомогою методу `find()`, який дозволяє витягувати документи, що задовольняють певним

критеріям. Щоб зробити вибірку ефективною, MongoDB надає механізм індексації, за допомогою якого можна значно прискорити процес пошуку по великій кількості документів.

Іншою важливою особливістю MongoDB є її здатність працювати з бінарними даними, зокрема великими файлами, завдяки підтримці GridFS. Це дозволяє зберігати великі файли, такі як зображення чи відео, прямо в базі даних, замість того, щоб використовувати сторонні системи зберігання.

Крім того, MongoDB дозволяє здійснювати різноманітні операції зі сховищем, як-от вставка, оновлення, видалення документів, а також управління обсягами збережених даних через механізми очищення та архівування, що дозволяє ефективно керувати ресурсами системи.

Завдяки таким можливостям, MongoDB є дуже популярною серед розробників, які працюють з веб-додатками, аналітичними системами та іншими рішеннями, де важлива ефективність, масштабованість і гнучкість зберігання даних.

```
{
  "username" : "bob",
  "address" : {
    "street" : "123 Main Street",
    "city" : "Springfield",
    "state" : "NY"
  }
}
```

Рисунок 3.5 – JSON об’єкт з інформацією про користувача

3.3 HTML, CSS та JavaScript

HTML, або мова гіпертекстової розмітки, є основою для створення веб-сторінок і є стандартом для розмітки тексту, зображень та інших елементів на веб-сторінках. У самому HTML неможливо реалізувати складні алгоритми або математичні обчислення, тому він не є мовою програмування. Проте HTML дозволяє організувати інформацію на веб-сторінці, задаючи її

структуру і форматування, завдяки чому веб-браузери можуть коректно відображати контент користувачам. HTML є однією з основ веб-розробки і в поєднанні з іншими технологіями, такими як CSS і JavaScript, дозволяє створювати інтерактивні та привабливі веб-сторінки.

HTML-документи можуть зберігатися як на локальних комп'ютерах, так і на веб-серверах. Коли веб-браузер отримує запит на завантаження HTML-файлу, він інтерпретує цей файл і відображає його на екрані користувача. Структура документа в HTML є семантичною, що означає, що всі елементи документа взаємопов'язані і утворюють граф, у якому вузли — це об'єкти, а зв'язки між ними — це відносини між об'єктами. Це дозволяє легко розділяти контент на частини та виводити їх у вигляді, який буде зручний для користувача.

Кожен HTML-документ складається з кількох основних компонентів, таких як заголовок, тіло документа та декларація типу документа. Декларація типу документа (DOCTYPE) вказує браузеру, яку версію HTML потрібно використовувати для коректної інтерпретації даного документа. Це допомагає браузеру правильно інтерпретувати код і відображати сторінку так, як це передбачено стандартами. Шапка документа (head) містить метаінформацію, таку як посилання на стилі, JavaScript або інші ресурси, які не відображаються на веб-сторінці, але використовуються для її налаштування. Тіло документа (body) містить весь контент, який відображається на веб-сторінці, зокрема текст, зображення, форми та інші елементи.

Структура HTML-тегів є важливою частиною розмітки. Теги — це основні будівельні блоки веб-сторінки. Кожен тег може мати різні атрибути, які визначають його поведінку або вигляд на сторінці. Наприклад, тег `<img>` використовується для виведення зображень, і атрибут `src` вказує шлях до зображення. Теги завжди мають відкриваючий та закриваючий елементи, де закриваючий тег містить косу риску перед назвою тега. Атрибути тега можуть бути різними: деякі з них мають значення, інші — ні. Атрибути без значень,

як правило, вказуються просто за допомогою їхнього імені, наприклад, атрибут `checked` в елементі `<input>` для позначення стану "вибрано".

Атрибути зі значенням мають пару ім’я-значення, наприклад, `href="http://example.com"`, де `href` — це атрибут, а значенням є URL-адреса, куди веде цей тег. Завдяки атрибутам можна додавати різні властивості до елементів HTML. Ці атрибути можуть використовуватись для виведення динамічних елементів або зміни стилів за допомогою CSS або JavaScript. Каскадні таблиці стилів (CSS) використовуються для визначення візуального вигляду елементів на сторінці, наприклад, для зміни кольору тексту, шрифтів, розмірів елементів тощо. За допомогою JavaScript можна додавати інтерактивність до сторінки, наприклад, для обробки подій або зміни вмісту сторінки в реальному часі.

Завдяки такій структурі HTML дозволяє створювати дуже різноманітні та складні веб-сторінки, де кожен елемент має своє місце і функцію. Наприклад, структура документа може включати розділи з різними стилями для заголовків, абзаців, списків, таблиць, форм та зображень, що дає змогу чітко і логічно організувати контент на веб-сторінці (Рис.3.6).

```
<! DOCTYPE html>
<html>
   <head>
      <title>Назва</title>
   </head>
   <body>
      <p> Hello world!</p>
   </body>
</html>
```

Рисунок 3.6 – Структура HTML-документа

Так як HTML унаслідується від стандартної узагальненої мови розмітки SGML, то й усі типи даних у ній також присутні. Кожен елемент HTML-документу складається з атрибута та певного значення. Всі можливі варіації даних прописуються відповідно до типів, визначених у декларації типу

документа. Наприклад, такі атрибути, як ContentType, Charset, Script, StyleSheet, визначають особливості обробки даних у документі, наприклад, тип вмісту або набір символів.

Іноді виникає необхідність вставити в HTML-документ символи, які не описані в кодовій таблиці. У таких випадках використовуються символьні мнемоніки — SGML-посилання на символи. Існує два типи мнемонік: цифрові та сполучення символів. Цифрові мнемоніки вказують на позицію символу в таблиці кодів, а сполучення символів використовують псевдоніми для вказання символів. Наприклад, символи, які неможливо ввести безпосередньо на клавіатурі, можуть бути представлені у вигляді спеціальних кодів, що дозволяє браузеру правильно їх інтерпретувати та відображати.

Для покращення візуальної презентації веб-сторінки та додавання ефектів стилізації використовуються каскадні таблиці стилів (CSS). Це спеціальна мова, яка дозволяє описувати зовнішній вигляд елементів на сторінці. За допомогою CSS можна змінювати кольори, шрифти, розміри, а також додавати різноманітні візуальні ефекти, такі як спливаючі повідомлення при наведенні курсору на елемент, анімацію, зміну розмірів блоків тощо. Хоча CSS не дозволяє змінювати сам вміст документа або додавати логіку, він є потужним інструментом для зміни вигляду та організації інформації на сторінці.

Синтаксис CSS є досить простим та зручним. Кожне правило стилю складається з селектора і блоку властивостей. Селектор визначає, до яких елементів веб-сторінки застосовуватиметься стиль, а блок властивостей містить конкретні параметри, які змінюють вигляд елементів. Структура правила CSS виглядає наступним чином:

```
селектор { властивість: значення; }
```

Для застосування стилів до конкретного елементу необхідно вказати його селектор (наприклад, тег, id або class елементу). У середині фігурних дужок вказуються властивості, які змінюють вигляд елементів. Властивості задаються у вигляді пари "назва: значення", і кожне таке визначення розділяється крапкою з комою. Приклад правил CSS для зміни вигляду елемента наведено на рис. 3.7.

```
p {
  font-family: Verdana, sans-serif;
}
h2 {
  font-size: 110%;
  color: red;
  background: white;
}
.note {
  color: red;
  background: yellow;
  font-weight: bold;
}
p.warning {
  background: url(warning.png) no-repeat fixed top;
}
#paragraph1 {
  margin: 0;
}
a:hover {
  text-decoration: none;
}
#news p {
  color: red;
}
```

Рисунок 3.7 – Каскадні таблиці стилів

Для того, щоб зробити веб-сторінку більш інтерактивною, додати можливість взаємодії з користувачем, а також реалізувати функції авторизації, завантаження даних або обробки подій, використовують мову програмування JavaScript. JavaScript (JS) — це об’єктно-орієнтована мова програмування, що базується на стандарті ECMAScript. Програми, написані цією мовою,

називаються скриптами. Вони можуть вбудовуватися безпосередньо в HTML-документи і виконуватися автоматично під час завантаження веб-сторінки. JavaScript дозволяє створювати динамічний контент, взаємодіяти з користувачем через форми, створювати спливаючі вікна та багато інших ефектів, що значно покращує взаємодію користувача з сайтом.

JavaScript постійно еволюціонує, з кожним оновленням набуваючи нових можливостей та функціоналу. Протягом багатьох років мова розвивалася і здобула популярність завдяки своїй здатності працювати не тільки на клієнтській стороні, а й на сервері, завдяки таким технологіям, як Node.js. Це стало можливим через зміну парадигм програмування та розвиток нових інструментів і бібліотек, що дозволяють JavaScript використовувати для різних цілей, від управління базами даних до створення веб-серверів та запуску складних аналітичних систем.

Використання JavaScript в сучасних веб-додатках включає в себе різноманітні сценарії. Наприклад, можливість інтеграції з API для підключення до сторонніх сервісів робить JavaScript незамінним для створення інтерактивних додатків. Веб-сторінки тепер можуть безперервно отримувати та оновлювати дані з сервера, не перезавантажуючи сторінку. Це досягається через технології, такі як AJAX або WebSockets, які дозволяють передавати дані між клієнтом і сервером в режимі реального часу.

JavaScript також активно використовується для створення односторінкових веб-додатків (SPA), де користувачі можуть взаємодіяти з додатком без необхідності постійно перезавантажувати сторінки. Це значно покращує зручність використання, дозволяючи швидко відгукуватися на дії користувача, а також значно знижує навантаження на сервер, оскільки лише частини сторінки оновлюються.

Завдяки розвитку фреймворків, таких як React, Angular, Vue.js, JavaScript також став основною мовою для створення складних веб-інтерфейсів. Ці фреймворки надають потужні інструменти для розробки

компонентних структур, що дозволяє розбивати додатки на малі, легко керовані блоки, що спрощує тестування та підтримку проектів. Вони підтримують принципи реактивного програмування, що дозволяє автоматично оновлювати інтерфейс при зміні даних.

Однією з особливостей JavaScript є його асинхронний характер. Завдяки таким механізмам, як проміси (Promises) та async/await, програмісти можуть писати асинхронний код, який читається та підтримується так само, як і синхронний. Це дозволяє писати чистіший, більш зрозумілий і зручний для підтримки код, що особливо важливо при роботі з великими проектами, де існує велика кількість асинхронних запитів.

JavaScript також активно використовується для створення інтерактивних графіків та візуалізацій даних. Завдяки бібліотекам, таким як D3.js або Chart.js, можна створювати складні, анімовані графіки, діаграми та інші візуальні елементи, які дозволяють користувачам краще розуміти складні набори даних.

У той же час JavaScript має і свої обмеження, зокрема, в контексті безпеки. Оскільки мова була створена для роботи в браузерах, обмеження на доступ до системних ресурсів є важливою частиною її безпекової моделі. Наприклад, вона не має доступу до файлової системи або не може взаємодіяти з іншими веб-додатками, що працюють на різних доменах, без спеціальних дозволів. Ці обмеження дозволяють запобігти зловмисникам виконувати небезпечні операції, спрямовані на крадіжку або пошкодження даних користувачів (Рис.3.8).

Проте, ці ж обмеження можуть бути проблемою для деяких сценаріїв, особливо для складних систем, де потрібен доступ до локальних ресурсів або більш глибокі операції. У таких випадках розробники використовують Node.js, де JavaScript працює на сервері і може виконувати більш гнучкі завдання, як доступ до файлової системи, створення і збереження даних, обробка запитів від користувачів тощо. Крім того, за допомогою Node.js

можна реалізувати більш складні серверні додатки, де JavaScript керує логікою на сервері, а також обробляє запити від клієнтів.

JavaScript став також основою для розвитку нових технологій, таких як Internet of Things (IoT), де він використовується для управління розумними пристроями та взаємодії з ними через веб-інтерфейси. JavaScript дозволяє розробляти додатки, які працюють як на сервері, так і на пристрої, знижуючи складність коду та інтеграцію різних систем.

Завдяки постійному розвитку стандартів ECMAScript і нових фреймворків, JavaScript залишається однією з найбільш актуальних і важливих мов програмування для веб-розробки.

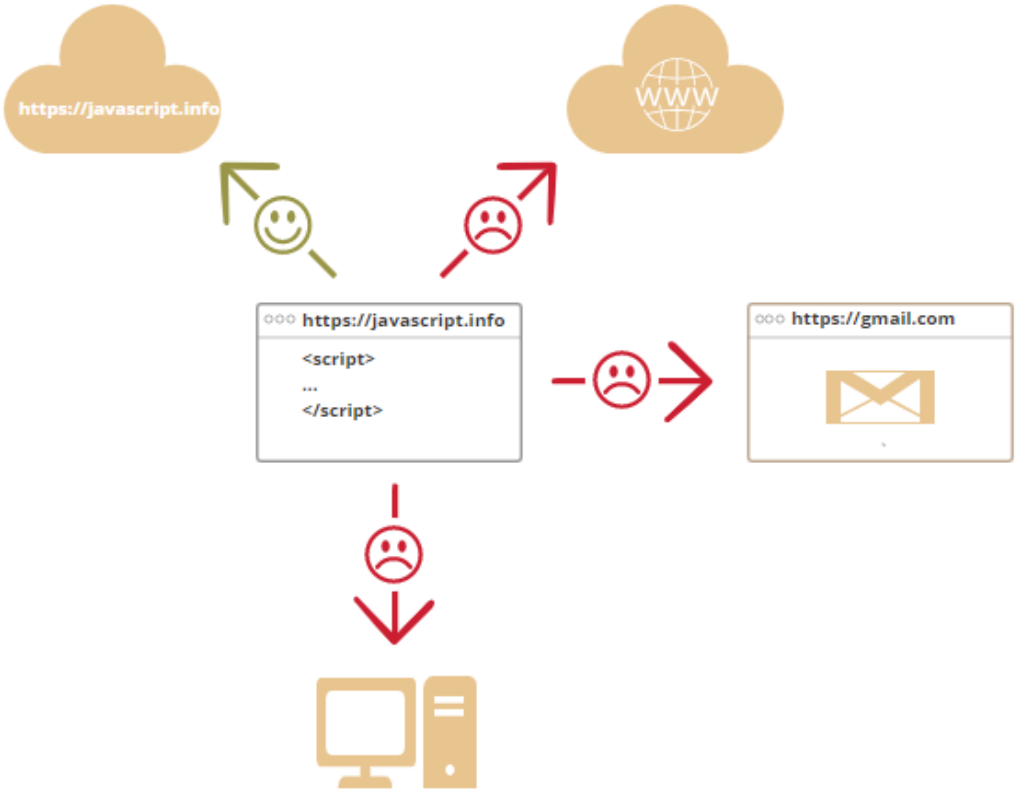


Рисунок 3.8 – Можливості та обмеження JS

3.4 WebScoket та JWT

Однією з важливих переваг використання WebSocket і JWT в сучасних веб-додатках є те, що вони дозволяють створювати динамічні та інтерактивні сервіси з мінімальними затримками при передачі даних. Наприклад, при використанні WebSocket для онлайн-ігор або чат-додатків, сервер може миттєво отримувати й передавати повідомлення між користувачами без необхідності постійно запитувати сервер, що значно покращує користувацький досвід. Оскільки з’єднання не закривається після кожного запиту, це також знижує навантаження на сервер і збільшує ефективність роботи додатка.

У випадку з JWT, додаткова безпека забезпечується завдяки використанню підпису, що дозволяє захистити інформацію від змін і підробок. Кожен токен містить зашифровану інформацію про користувача, а також строки дії токена, що дає змогу встановлювати обмеження на його використання. Наприклад, токен може мати обмежений час життя, після якого він стає недійсним, що допомагає знизити ризик використання старих або вкрадених токенів для несанкціонованого доступу.

Важливою перевагою використання JWT у поєднанні з WebSocket є можливість інтеграції з іншими сучасними технологіями, такими як мікросервіси або контейнеризація. Це дозволяє створювати масштабовані системи, де кожна частина додатка може окремо перевіряти токени для доступу до необхідних ресурсів. Наприклад, у великій системі, де є кілька серверів, кожен із яких обробляє окремі частини функціональності, використання JWT дозволяє централізовано здійснювати аутентифікацію користувачів без необхідності додаткових запитів до бази даних або стану сесії.

Протокол WebSocket забезпечує чудову підтримку реального часу, однак у великих системах необхідно враховувати певні аспекти масштабування та управління з’єднаннями. Для цього можна

використовувати спеціалізовані сервери або посередники (наприклад, Nginx або HAProxy), які дозволяють обробляти велику кількість одночасних з’єднань, надаючи можливість оптимізувати навантаження та забезпечити стійкість системи навіть при високих навантаженнях.

Використання JWT дозволяє ще більше підвищити безпеку в таких розподілених системах, оскільки кожен мікросервіс, що обробляє з’єднання WebSocket, може перевіряти токен без необхідності взаємодії з іншими сервісами чи базою даних, що значно прискорює процес авторизації і зменшує можливість помилок.

Таким чином, поєднання WebSocket і JWT дозволяє створити потужну, масштабовану і безпечну інфраструктуру для веб-додатків, що працюють в реальному часі, і забезпечити користувачам зручний та швидкий доступ до ресурсів без втрати безпеки. Це особливо важливо в сучасних умовах, коли користувачі очікують миттєвий доступ до даних і високий рівень захисту їх особистої інформації.

4 РОЗРОБКА СИСТЕМИ ТА ОПТИМІЗАЦІЯ ЗОБРАЖЕНЬ

4.1 Компоненти системи

Основою системи є комп’ютер Raspberry Pi 4 Model B (Рис.4.1), який є останньою моделлю в популярній серії одноплатних комп’ютерів Raspberry Pi. Цей пристрій пропонує значне покращення порівняно з попереднім поколінням Raspberry Pi 3 Model B+, забезпечуючи значно більшу швидкість процесора, підвищену продуктивність мультимедіа, більший обсяг оперативної пам'яті та кращі можливості підключення. Водночас він зберігає сумісність з попередніми версіями, а також має подібне енергоспоживання, що робить його ефективним і зручним для використання в різних проєктах. Raspberry Pi 4 Model B пропонує продуктивність, порівнянну з базовими ПК на платформі x86, що робить його оптимальним рішенням для безлічі завдань, де потрібна невелика потужність та компактність.

Комп’ютер, як зазначено, має революційні покращення в порівнянні з попередніми моделями. Основні характеристики Raspberry Pi 4 Model B включають високопродуктивний 64-розрядний чотириядерний процесор ARM Cortex-A72, що дозволяє значно підвищити швидкість обчислень у порівнянні з попереднім поколінням. Однією з ключових особливостей є підтримка подвійного дисплея з роздільною здатністю до 4K через два порти micro-HDMI, що дозволяє підключати монітори з високою роздільною здатністю для роботи з графікою або мультимедійними додатками. Також передбачене апаратне декодування відео до 4Kp60, що забезпечує відтворення високоякісного відео.

Крім того, Raspberry Pi 4 Model B може оснащуватися до 8 Гб оперативної пам’яті, що дозволяє використовувати його для більш складних і ресурсомістких завдань, ніж його попередники. Вбудовані модулі для підключення до мережі включають підтримку бездротових локальних мереж Wi-Fi (діапазони 2,4 / 5,0 ГГц) та Bluetooth 5.0, що дозволяє використовувати

Raspberry Pi для проектів, де потрібна підтримка бездротових з’єднань. Також є Gigabit Ethernet для забезпечення швидкого з’єднання через дротову мережу.

Raspberry Pi 4 Model B підтримує USB 3.0 порти для підключення зовнішніх пристроїв, що забезпечує швидку передачу даних між комп’ютером і периферійними пристроями. Для можливості живлення через Ethernet (PoE) передбачений додатковий модуль PoE HAT, що дозволяє спростити підключення Raspberry Pi до мережі та зменшити кількість необхідних кабелів.

Однією з важливих особливостей Raspberry Pi 4 є модуль сертифікації відповідності для бездротових мереж Wi-Fi та Bluetooth, що дозволяє розробникам зменшити час і витрати на тестування відповідності вимогам стандартів. Це важливе для впровадження в комерційні та промислові рішення, оскільки дозволяє швидше вивести продукт на ринок, знижуючи витрати на сертифікацію.

Рисунок 4.1 – Raspberry Pi 4 Model B

Наступним важливим компонентом системи є камера RASPBERRY PI INFRARED (Рис.4.2). Цей модуль камери є популярним вибором у застосунках, пов'язаних з домашньою безпекою та спостереженням, завдяки своїй високій ефективності та універсальності. Камера сумісна з усіма моделями Raspberry Pi, включаючи версії 1, 2, 3 та 4, що робить її зручним рішенням для різних типів проектів. Для доступу до камери можна використовувати API MMAL (Multi-Media Abstraction Layer) та V4L (Video for Linux), а також існують численні сторонні бібліотеки, серед яких бібліотека Picamera Python, яка спрощує інтеграцію камери з Python-сценаріями та розробкою програмного забезпечення.

Камера оснащена 8-мегапіксельним датчиком Sony IMX219, що забезпечує високу якість зображень та відео. Вона має кут огляду 75,7 градусів, що дозволяє охоплювати досить велику площу для моніторингу, що робить її ефективною для використання в системах спостереження та безпеки. Камера підтримує запис відео з роздільною здатністю 1080p, що дозволяє отримувати чітке відео з високою деталізацією. Крім того, камера оснащена можливістю підключення інфрачервоного світлодіода або світлодіодного спалаху, що дозволяє здійснювати зйомку в умовах низької освітленості або навіть в повній темряві, забезпечуючи стабільну роботу в будь-яких умовах освітлення.

Ця камера є важливим компонентом для реалізації проектів, де необхідна висока якість відео з можливістю роботи в умовах низької освітленості, таких як системи спостереження, інтелектуальні системи безпеки або інші автоматизовані системи.

Рисунок 4.2 – RASPBERRY PI INFRARED

4.2 Архітектура системи

Система складається з веб-додатку та двох серверних компонентів: один з яких розташований на мобільній платформі, а інший інтегрований у сховище. Всі ці компоненти об'єднані в єдину систему, яка функціонує завдяки дев'яти основним модулям, що забезпечують її ефективну роботу (Рис. 4.3).

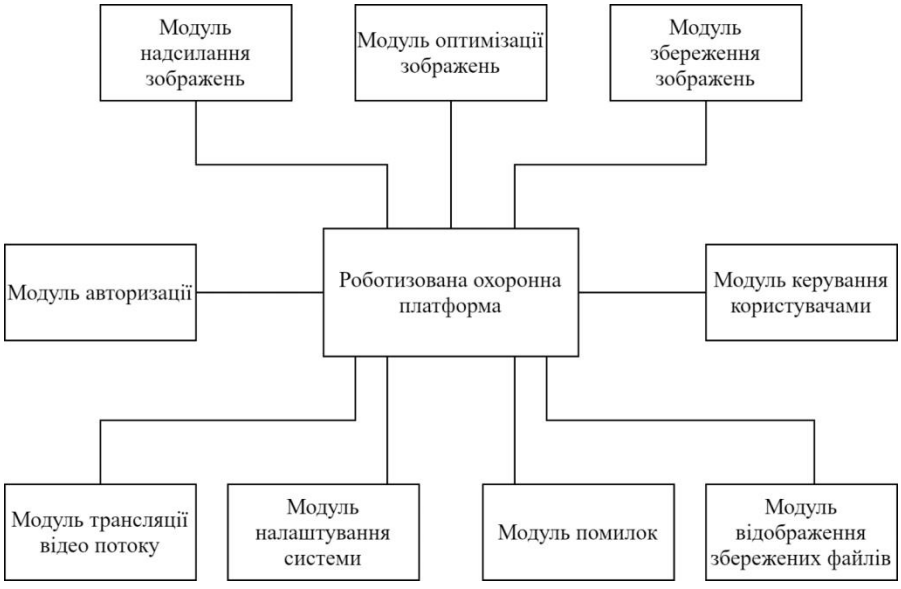


Рисунок 4.3 – Структура архітектури системи

Розглянемо детальніше архітектурну структуру. Почати слід з модуля керування користувачами. Цей компонент дозволяє користувачеві змінювати свої налаштування, додавати нових користувачів або видаляти існуючих. При першому вході в систему, користувач зобов'язаний створити обліковий запис, щоб мати можливість налаштувати параметри системи і отримати доступ до неї віддалено в подальшому. Цей користувач є основним в системі, адміністратором, і має доступ до всіх параметрів системи. Адміністратор може створювати нові облікові записи, переглядати та редагувати інформацію про існуючих користувачів. Для всіх користувачів, які будуть додаватись після цього, необхідно буде отримати підтвердження від адміністратора. Це підтвердження надсилається на електронну пошту адміністратора, і після його отримання, створюється профіль нового користувача, а на його пошту приходить повідомлення про успішне завершення цього процесу. Після авторизації новий користувач може переглядати історію збережених файлів, здійснювати онлайн-перегляд відео трансляцій, а також редагувати свій профіль (змінюючи поштову адресу, відновлюючи чи оновлюючи пароль, змінюючи ім’я користувача).

Після створення облікового запису, наступним важливим етапом є процес авторизації, за який відповідає окремий модуль авторизації. Для того, щоб увійти до системи, користувач має ввести свої персональні дані у відповідну форму входу. Після цього генерується унікальний зашифрований ключ, за допомогою якого користувач отримує доступ до функцій, що відповідають його ролі в системі. У разі введення невірних даних, модуль помилок сповістить користувача про це та запропонує повторити спробу. Якщо користувач забув свої облікові дані, він може скористатися функцією відновлення паролю, натиснувши на кнопку "Forgot your password? Click here", для відновлення доступу через електронну пошту, на яку був зареєстрований обліковий запис.

Наступний важливий компонент системи відповідає за оптимізацію зображень. Завдяки цьому модулю всі збережені та підготовлені для подальшої відправки зображення, а також кадри з відеопотоку, будуть проходити оптимізацію через конвертацію файлів у формат WebP і їх стиснення з урахуванням заданого параметру якості. Адміністратор має можливість налаштувати рівень якості вихідних зображень і визначити бажану кількість кадрів на секунду, які мають зберігатись у сховищі або передаватися в режимі реального часу через відеопотік. Для досягнення максимальної ефективності передачі файлів від роботизованої платформи до серверного сховища цей модуль конвертує фотографії в бінарний формат, що значно зменшує обсяг переданих файлів порівняно з форматом Base64.

Модуль передачі зображень від платформи до сховища функціонує відповідно до налаштувань, заданих адміністратором. Він періодично передає оброблені з камери фотографії із заданим інтервалом. Модуль прийому і збереження зображень працює безпосередньо на стороні сховища. Коли надходить запит на збереження, система перевіряє наявний обсяг вільного місця в сховищі. Якщо місце є, система продовжує процес збереження. Якщо простір обмежений, система автоматично звільняє місце шляхом видалення застарілих файлів. Далі дані запиту конвертуються з бінарного формату в WebP для подальшого збереження.

Для зручності подальшого доступу всі збережені файли організовуються у відповідні папки. Головними критеріями розподілу є дата запису, яка вказана у форматі DD-MM-YYYY (день-місяць-рік). В кожній папці по даті зберігаються підкаталоги з позначенням часу у форматі HH (година). Кожна година розділяється на інтервали по 10 хвилин, відповідні папки позначаються у форматі M-M (перша та десята хвилина). Всередині цих папок містяться всі зображення, зроблені протягом конкретної хвилини.

Модуль трансляції відеопотоку дозволяє користувачам здійснювати віддалений моніторинг приміщення в реальному часі з мінімальною

затримкою. Адміністратор має можливість налаштувати параметри відеопотоку, зокрема кількість кадрів, які передаються, а також якість зображень. У разі відсутності трансляції, наприклад, через заряджання робота або відсутність підключення до мережі Інтернет, система через модуль помилок сповістить користувача про відсутність доступу до робота в даний момент часу.

Останній модуль, що відповідає за відображення збережених файлів, складається з кількох доступних запитів, кожен з яких виконує конкретну задачу. Перший запит getFolders повертає список збережених папок у сховищі, що організовані за датою у форматі DD-MM-YYYY (день-місяць-рік) для зручності доступу. Наступний запит getDate надає користувачу список годин, протягом яких платформа надсилала зображення в сховище в обраний день. Третій запит getMinutes надає список 10-хвилинних інтервалів, в рамках яких зберігались зображення. Четвертий запит getImages дозволяє переглядати всі зображення, збережені на платформі на вибрану дату та час. Для відображення всіх фотографій у форматі відео, був реалізований додатковий запит getVideo, який конвертує завантажені зображення у послідовність кадрів, що потім транслюються користувачу як відеопотік.

4.3 Структура бази даних

База даних, що використовується в системі, включає дві основні колекції: users та configurations, які забезпечують нормальне функціонування всієї системи та зберігання ключової інформації.

Перша колекція, що містить дані про користувачів, називається users і складається з чотирьох основних полів: унікальна електронна пошта користувача (email), зашифрований пароль (password), роль користувача (role) та ім’я користувача (name). У системі визначені два типи ролей: user та admin. Для першого користувача, що реєструється в системі, автоматично

призначається роль admin, що надає йому доступ до всіх функціональних можливостей системи, включаючи можливість затверджувати реєстрацію нових користувачів. Детальніше збудовану структуру колекції користувачів можна побачити на рис. 4.4.

```
const UserSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
  },

  password: {
    type: String,
    required: true,
  },

  name: {
    type: String
  },

  role: {
    type: String,
    default: 'user',
  }
});
```

Рисунок 4.4 – Структура колекції користувачів

Друга колекція бази даних, configurations, містить параметри для налаштувань системи. Вона включає чотири важливих поля: кількість кадрів у секунду для відеопотоку, що передається (streamFPS), якість зображення для відеопотоку (streamQuality), кількість кадрів на секунду для збереження даних у сховищі (exportFPS), а також якість зображення для збережених даних (exportQuality). Адміністратор має можливість налаштувати ці параметри відповідно до вимог. Схему цієї колекції можна побачити на рис. 4.5.

```
const ConfigurationSchema = new mongoose.Schema({
  streamFPS: {
    type: Number,
    default: 24
  },

  streamQaulity: {
    type: Number,
    default: 1.0
  },

  exportFPS: {
    type: Number,
    default: 24
  },

  exportQaulity: {
    type: Number,
    default: 1.0
  }
});
```

Рисунок 4.5 – Структура колекції налаштувань системи

4.4 Розробка API

Розробка системи повинна починатися з визначення кінцевих точок доступу API. Оскільки ця система є обмеженою, доступ до неї мають лише зареєстровані користувачі. Для того, щоб отримати доступ до ресурсів системи, користувач має пройти процедуру входу, після чого йому буде надано згенерований веб-токен, який необхідний для виконання всіх подальших запитів. Під час переходу між різними сторінками здійснюється контроль за автентифікацією користувача та перевірка його прав для доступу до конкретної сторінки. Цей процес забезпечується за допомогою сервісу middlewares, який перевіряє наявність відповідних прав доступу на кожен запит. У разі успішної автентифікації користувачу відображається необхідна інформація, а в разі невдачі — виводиться повідомлення про помилку.

Система підтримує два основних типи запитів: HTTP та WebSocket. Для передачі відео потоку в реальному часі від камери до користувача переважно використовуються WebSocket запити, оскільки вони забезпечують більш

ефективну передачу великої кількості зображень, які формують відео потік. Відправка відео через звичайний HTTP запит вимагає додаткових інтервалів для коректного відображення зображень на стороні клієнта, що негативно позначається на швидкодії системи. З цього погляду, використання WebSocket значно покращує швидкість та ефективність обробки даних. Структура WebSocket сервера та процес авторизації користувача детально зображені на рис. 4.6 та рис. 4.7.

```
const WEBSOCKET_PORT = process.argv[4] || 8084,
const socketServer = new (require('ws').Server)({port: WEBSOCKET_PORT});

const WebSocketServer = require('ws').Server;
const socketServer = new WebSocketServer({
    verifyClient: function (info, cb) {
        var token = info.req.headers.token
        if (!token)
            cb(false, 401, 'Unauthorized')
        else {
            jwt.verify(token, 'secret-key', function (err, decoded) {
                if (err) {
                    cb(false, 401, 'Unauthorized')
                } else {
                    info.req.user = decoded //[1]
                    cb(true)
                }
            })

        }
    }
});
```

Рисунок 4.6 – Структура WebSocket сервера

```
socketServer.on('connection', function(socket) {
    // Send magic bytes and video size to the newly connected socket
    // struct { char magic[4]; unsigned short width, height;}
    var streamHeader = new Buffer(8);
    streamHeader.write(STREAM_MAGIC_BYTES);
    streamHeader.writeUInt16BE(width, 4);
    streamHeader.writeUInt16BE(height, 6);
    socket.send(streamHeader, {binary:true});

    console.log( 'New WebSocket Connection ('+socketServer.clients.length+' total)' );

    socket.on('close', function(code, message){
        console.log( 'Disconnected WebSocket ('+socketServer.clients.length+' total)' );
    });
});
```

Рисунок 4.7 – Авторизація користувача в WebSocket сервері

Щодо HTTP запитів, вони поділяються на дві основні категорії. Перша категорія відповідає за візуальне відображення контенту. При переході користувача на конкретну сторінку виконується запит, який повертає HTML-документ із підключеними JavaScript скриптами та стилями. Цей механізм забезпечує коректне відображення сторінок для користувачів, які успішно пройшли авторизацію. Приклад реалізації цього механізму можна побачити на рис. 4.8.

```
app.get('/login', function(req, res){
    res.sendFile('./login.html', {root: __dirname});
});

app.get('/cameraStream', function(req, res){
    res.sendFile('./cameraStream.html', {root: __dirname});
});

app.get('/storageHistory', function(req, res){
    res.sendFile('./storageHistory.html', {root: __dirname});
});

app.get('/systemConfiguration', function(req, res){
    res.sendFile('./systemConfiguration.html', {root: __dirname});
});

app.get('/users', function(req, res){
    res.sendFile('./users.html', {root: __dirname});
});

app.get('/usersSettings', function(req, res){
    res.sendFile('./usersSettings.html', {root: __dirname});
});
```

Рисунок 4.8 – HTTP запити для відображення веб-сторінки

Друга категорія HTTP запитів забезпечує інтерактивність системи, дозволяючи користувачам створювати, редагувати або видаляти свої облікові записи. Ці запити також використовуються для налаштування параметрів системи, передачі та збереження зображень у сховищі, а також для отримання файлів і їх подальшого відображення на веб-сторінках. Структура цих запитів для роботи з користувачами зображена на рис. 4.9.

```
app.post('/user/login', async (req, res) => {
  const email = req.body.user.email;
  const password = req.body.user.password;
  try {
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.Login(email, password);
    return res.status(200).json({ user, token }).end();
  } catch(e) {
    return res.json(e).status(500).end();
  }
});

app.post('/user/login-as', isAuth, attachCurrentUser, checkRole('admin'), async (req, res) => {
  try {
    const email = req.body.user.email;
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.LoginAs(email);
    return res.status(200).json({ user, token }).end();
  } catch(e) {
    console.log('Error in login as user: ', e);
    return res.json(e).status(500).end();
  }
});

app.post('/user/signup', async (req, res) => {
  try {
    const { name, email, password } = req.body.user;
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.SignUp(email, password, name);
    return res.json({ user, token }).status(200).end();
  } catch (e) {
    return res.json(e).status(500).end();
  }
});
```

Рисунок 4.9 – Структура HTTP запитів для взаємодії з користувачем

4.5 Розробка серверної частини системи

Наступним етапом розробки є створення серверної частини системи, яка забезпечує обробку запитів, взаємодію з базою даних та виконання бізнес-логіки. Серверна частина відповідає за прийом і обробку HTTP і WebSocket запитів, а також за авторизацію користувачів, управління їх правами доступу та забезпечення надійної взаємодії між компонентами системи.

Основним елементом серверної частини є веб-сервер, який обробляє запити користувачів і надає відповіді у відповідному форматі. Для обробки запитів використовуються сучасні технології та фреймворки, такі як Node.js та Express.js, що дозволяють ефективно створювати масштабовані веб-додатки. Веб-сервер взаємодіє з базою даних через ORM (Object-Relational Mapping) бібліотеки, що полегшує роботу з базою даних і знижує ймовірність помилок при виконанні запитів до неї.

Крім того, серверна частина включає в себе обробку WebSocket з'єднань, що дозволяє здійснювати передачу відео потоку в реальному часі. Для цього використовується Socket.io, що забезпечує ефективну двосторонню комунікацію між сервером і клієнтом. WebSocket запити дозволяють мінімізувати затримки при передачі зображень і, відповідно, покращити швидкодію системи.

Сервер також містить функціонал для автентифікації користувачів. Процес авторизації здійснюється через перевірку даних користувача в базі даних, де зберігається зашифрований пароль. Після успішної перевірки користувач отримує веб-токен, який використовується для доступу до всіх захищених частин системи. Веб-токен є тимчасовим і має обмежений термін дії, після закінчення якого користувач повинен пройти повторну авторизацію.

Для налаштування параметрів системи використовується окремий модуль, який дозволяє адміністраторам змінювати важливі параметри, такі як кількість кадрів у секунду для відео потоку, якість зображення, а також параметри для експорту даних. Ці налаштування зберігаються в окремій колекції бази даних і можуть бути змінені лише користувачами з правами адміністратора.

Розробка серверної частини передбачає також забезпечення високої надійності та безпеки системи. Для цього використовуються механізми захисту від атак, таких як CSRF, XSS та SQL Injection. Всі важливі дані, зокрема паролі користувачів, зберігаються у зашифрованому вигляді, що додає додатковий рівень безпеки при роботі з чутливою інформацією.

4.5.1 Модуль авторизації та керування користувачами

Оскільки система є закритого типу, що передбачає перевірку веб-токена для кожної дії користувача, важливо детально розглянути механізм авторизації.

Процес реєстрації нового користувача включає форму для введення його особистої інформації: електронної пошти, пароля та імені. Пароль, перед тим як бути збереженим у базі даних, повинен бути зашифрований. Для цього використовується метод Argon2, що є одним із найнадійніших для хешування паролів. Поряд з паролем в базу даних зберігаються електронна пошта та ім’я користувача. За замовчуванням роль нового користувача встановлюється як user, але адміністратор може змінити роль на admin після підтвердження облікового запису.

На рисунку 4.10 зображено процес створення нового користувача.

```
import * as argon2 from 'argon2';

class AuthService {
  public async SignUp(email, password, name): Promise<any> {
    const passwordHashed = await argon2.hash(password);

    const userRecord = await UserModel.create({
      password: passwordHashed,
      email,
      name,
    });

    const token = this.generateJWT(userRecord);

    return {
      user: {
        email: userRecord.email,
        name: userRecord.name,
      },
      token
    }
  }
}
```

Рисунок 4.10 – Створення нового користувача

Коли користувач намагається увійти в систему, він відправляє серверу свою електронну пошту та пароль. Алгоритм авторизації включає кілька етапів:

1. Клієнт відправляє серверу дані для входу (електронна пошта та пароль).

2. Сервер шукає користувача в базі даних за електронною поштою.

3. Якщо користувач існує, сервер порівнює введений пароль з тим, що зберігається в базі даних, шляхом його дешифрування за допомогою бібліотеки argon2.

4. Якщо перевірка успішна, сервер генерує JSON Web Token (JWT) — тимчасовий токен доступу для подальших запитів.

Цей процес продемонстровано на рисунку 4.11.

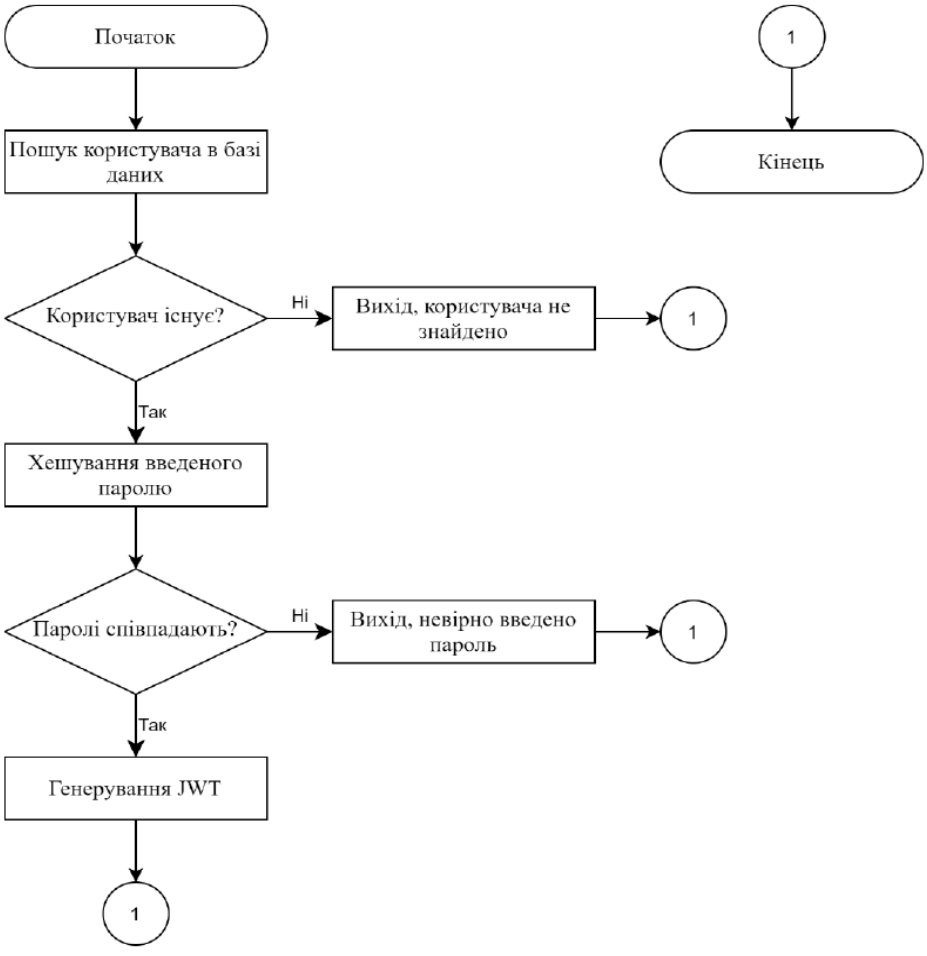


Рисунок 4.11 – Алгоритм авторизації користувача

JWT — це тимчасовий токен, який необхідно включати у заголовки HTTP-запитів до захищених кінцевих точок системи. Після того як користувач отримує JWT, він додається до кожного запиту у полі Authorization. Приклад цього процесу представлено на рисунку 4.12.

```
import * as argon2 from 'argon2';

class AuthService {
  public async Login(email, password): Promise<any> {
    const userRecord = await UserModel.findOne({ email });
    if (!userRecord) {
      throw new Error('User not found')
    } else {
      const correctPassword = await argon2.verify(userRecord.password, password);
      if (!correctPassword) {
        throw new Error('Incorrect password')
      }
    }


    return {
      user: {
        email: userRecord.email,
        name: userRecord.name,
      },
      token: this.generateJWT(userRecord)
    }
  }
}
```

Рисунок 4.12 – Авторизація користувача

Для генерації JWT використовується бібліотека jsonwebtoken для Node.js. Функція генерації токену продемонстрована на рисунку 4.13. Токен має обмежений термін дії, після закінчення якого користувач повинен пройти повторну авторизацію.

```
import * as jwt from 'jsonwebtoken'
class AuthService {
  private generateToken(user) {

    const data =  {
      _id: user._id,
      name: user.name,
      email: user.email
    };
    const signature = 'wgUsh20aG';
    const expiration = '6h';

    return jwt.sign({ data, }, signature, { expiresIn: expiration });
}
```

Рисунок 4.13 – Функція генерації JWT-токена

Щоб забезпечити доступ до захищених маршрутів, необхідно реалізувати проміжне програмне забезпечення (middleware), яке буде перевіряти наявність та валідність JWT у запитах. Це програмне забезпечення додасть логіку перевірки токену до кожного маршруту, що вимагає авторизації.

Приклад проміжного програмного забезпечення для перевірки токену наведений на рисунку 4.14.

```
import * as jwt from 'express-jwt';

const getTokenFromHeader = (req) => {
  if (req.headers.authorization && req.headers.authorization.split(' ')[0] === 'Bearer') {
    return req.headers.authorization.split(' ')[1];
  }
}

export default jwt({
  secret: 'wgUsh20aG',
  userProperty: 'token',
  getToken: getTokenFromHeader
});
```

Рисунок 4.14 – Проміжне програмне забезпечення для авторизації

Для того, щоб отримати детальну інформацію про користувача, який виконує запит, можна додати проміжне програмне забезпечення, яке буде підключати дані користувача до кожного запиту. Цей процес дозволяє маршрутам отримувати необхідну інформацію про користувача з бази даних

для виконання подальших операцій. Спрощений код для отримання відомостей про користувача представлений на рисунку 4.15.

```
export default (req, res, next) => {
  const decodedTokenData = req.tokenData;
  const userRecord = await UserModel.findOne({ _id: decodedTokenData._id });

  if(!userRecord) {
    return res.status(401).end('User not found')
  } else {
    req.currentUser = userRecord;
    return next();
  }
}
```

Рисунок 4.15 – Отримання відомостей про користувача

Після реалізації механізму авторизації, всі маршрути, окрім сторінки авторизації, стають захищеними. Для доступу до цих кінцевих точок користувач повинен мати валідний JWT. За допомогою проміжного програмного забезпечення, можна додавати дані про користувача до запитів і забезпечити їх правильну обробку на сервері.

4.5.2 Модуль оптимізації та передачі зображень

Модуль для стиснення зображень та конвертації файлів у бінарний вигляд є важливою частиною системи, яка дозволяє ефективно обробляти та передавати відео та зображення з камер на платформі.

Підключення до камери та запис відео. При запуску системи сервер підключається до камери для запису, конвертації та передачі відеопотоку у необхідному форматі. Для цього використовується бібліотека FFmpeg через NPM, яка дозволяє виконувати всі необхідні операції за допомогою однієї команди.

Налаштування параметрів зчитування з камери. При запуску системи, сервер зчитує налаштування з бази даних, зокрема максимальну кількість кадрів у секунду, з якою необхідно зчитувати зображення з камери. Ці параметри застосовуються для налаштування роботи модуля.

Запис файлів у пам'ять. Запис з камери здійснюється на дві частини:

1. Перша частина записує кадри з камери у пам'ять робота з вказаною частотою exportFPS для подальшої обробки. Ці файли передаються у форматі image2, що дозволяє зберігати зображення у форматі jpg.

Команда для запису фотографій у пам'ять робота:

```bash
ffmpeg -f dshow -rtbufsize 100M -i video=Camera abc.mp4 -r ${exportFPS} -f image2 ./savedImages/input.mp4_fr%7d.jpg
```

2. Друга частина займається записом відео з камери та конвертацією кадрів у відеопотік, який потім транслюється через WebSocket сервер. Вихідні файли передаються у форматі mjpeg, а параметр streamQuality визначає якість стиснення кадрів за допомогою алгоритму JPEG, що дозволяє швидко конвертувати зображення без затримок. Також в команду передається параметр StreamFPS, який регулює частоту кадрів у відеопотоці.

Цей підхід дозволяє ефективно оптимізувати передачу відео та зображень, зменшуючи навантаження на систему при збереженні високої якості зображення та відео.

Команда запуску трансляції відеопотою на WebSocket сервер:

```bash
ffmpeg -rtbufsize 100M -i /dev/video0 abc.mp4 -r ${streamFPS} -q:v ${streamQuality} -f mjpeg http://localhost:8082/12345/1920/1080
```

Ця команда використовується для трансляції відеопотоку на WebSocket сервер. Вона забезпечує стабільну передачу відео з камери у форматі MJPEG, який підтримує стиснення кожного кадру в реальному часі. Параметри streamFPS та streamQuality дозволяють налаштувати частоту кадрів та якість трансляції відповідно до потреб користувача та можливостей інтернет-

з'єднання, що дає змогу оптимізувати баланс між якістю, кількістю кадрів та затримками.

Оптимізація передачі кадрів. Для зменшення затримок і пропуску відеопотоку на WebSocket сервері важливо не лише стиснути кожен кадр у форматі MJPEG, а й оптимізувати саму передачу даних. Кожен кадр має бути записаний у буфер і перетворений в двійковий формат, що дозволяє значно зменшити розмір даних перед їх відправленням до клієнтів.

Розміри рядків у форматі Base64 (Табл.4.1). Використання формату Base64 для кодування кадрів зображень дозволяє передавати їх як текст, але це може збільшити розмір даних. В середньому, розмір вихідного рядка для кожного окремого кадру у форматі Base64 становить близько 42,500 байтів.

Таблиця 4.1 – Розміри рядків кожного кадру у форматі Base64:

Кадр	Довжина рядка
1	42455 (41.5 kB)
2	42735 (41.7 kB)
3	42751 (41.7 kB)
4	43075 (42.0 kB)
5	42451 (41.5 kB)

Це означає, що кожен кадр, який передається через WebSocket сервер у форматі Base64, займає значний обсяг пам'яті. Оскільки Base64 збільшує розмір даних на 33%, важливо оцінювати вплив цього на пропускну здатність мережі, особливо коли транслюється велика кількість кадрів одночасно для кількох користувачів.

Таким чином, для ефективної оптимізації передачі відео через WebSocket сервер необхідно враховувати такі фактори, як швидкість передачі, стиснення кадрів і оптимізація формату передачі даних (наприклад, через двійкове представлення замість Base64), щоб мінімізувати затримки і забезпечити стабільний потік для всіх користувачів.

Процес передачі даних через WebSocket сервер з використанням бінарного формату для кадрів відео є ефективним способом оптимізації передачі великих обсягів даних (Рис.4.16). У даному підході створено сервер, який приймає кожен кадр з відео, конвертує його в бінарний формат і відправляє цей кадр на WebSocket сервер. Далі цей сервер передає бінарні дані всім підключеним користувачам у вигляді відео потоку.

Цей метод значно зменшує розмір даних порівняно з використанням формату Base64, який, хоча й забезпечує простоту кодування, вимагає значно більше місця для зберігання даних. Кожен кадр, конвертований у бінарний формат, займає майже втричі менше місця, що робить його ідеальним для передачі великих обсягів відео інформації без помітних затримок.

```
const WEBSOCKET_PORT = process.argv[4] || 8084,
const socketServer = new (require('ws').Server)({port: WEBSOCKET_PORT});

const WebSocketServer = require('ws').Server;
const socketServer = new WebSocketServer({
    verifyClient: function (info, cb) {
        var token = info.req.headers.token
        if (!token)
            cb(false, 401, 'Unauthorized')
        else {
            jwt.verify(token, 'secret-key', function (err, decoded) {
                if (err) {
                    cb(false, 401, 'Unauthorized')
                } else {
                    info.req.user = decoded //[1]
                    cb(true)
                }
            })

        }
    }
});
socketServer.on('connection', function(socket) {
    // Send magic bytes and video size to the newly connected socket
    // struct { char magic[4]; unsigned short width, height;}
    var streamHeader = new Buffer(8);
    streamHeader.write(STREAM_MAGIC_BYTES);
    streamHeader.writeUInt16BE(width, 4);
    streamHeader.writeUInt16BE(height, 6);
    socket.send(streamHeader, {binary:true});

    console.log( 'New WebSocket Connection ('+socketServer.clients.length+' total)' );

    socket.on('close', function(code, message){
        console.log( 'Disconnected WebSocket ('+socketServer.clients.length+' total)' );
    });
});
```

Рисунок 4.16 – Код серверу для передачі та конвертації

Використання бінарного формату дозволяє не лише зменшити обсяг передаваних даних, але й підвищити ефективність мережевої передачі, що критично важливо в умовах обмеженої пропускної здатності каналу. Завдяки такій оптимізації система здатна обробляти та передавати відео в режимі реального часу навіть для великої кількості користувачів, що підключаються одночасно.

Для зручності роботи з бінарними даними розміри рядків кожного кадру значно зменшуються, як показано в таблиці 4.2. Це робить передачу даних значно більш ефективною, порівняно з іншими методами передачі, такими як Base64, де розміри даних значно більші.

Таблиця 4.2 – Розміри рядків кожного кадру в бінарному форматі

Кадр	Розмір рядка
1	27.5 kB
2	27.7 kB
3	28.0 kB
4	27.7 kB
5	27.8 kB

Цей підхід забезпечує значні переваги для систем, які потребують швидкої та стабільної передачі відео потоку, гарантуючи низьку затримку і збереження високої якості відео при мінімальному навантаженні на мережу.

При зчитуванні відео кадрів з камери для їх збереження часто використовують формат JPG, оскільки він дозволяє отримати зображення доброї якості. Однак одним з основних недоліків цього формату є значний розмір файлів, що ускладнює їх зберігання та передачу. Для вирішення цієї проблеми, була розроблена система стиснення зображень, що використовує бібліотеки NPM — imagemin і imagemin-webp. Завдяки цим інструментам можна конвертувати зображення з формату JPG у більш ефективний формат WebP без значної втрати якості.

WebP є форматом зображень, який дозволяє значно зменшити розмір файлів при збереженні високої якості зображень. Це досягається завдяки використанню ефективніших алгоритмів стиснення. Крім того, в процесі конвертації можна регулювати кінцеву якість зображення, що дає можливість досягти оптимального балансу між розміром файлу та його якістю. На рисунку 4.17 показано приклад реалізації цього процесу конвертації та стиснення.

```
const imagemin = require('imagemin');
const imageminWebp = require('imagemin-webp');

const exportQuality = mongo.get('exportQuality');

imagemin(['images/*'], {
  destination: 'compressed_images',
  plugins: [imageminWebp({quality: exportQuality})]
}).then(() => {
  sendFiles();
});
```

Рисунок 4.17 – Конвертація та стиснення зображення

Для детальнішого порівняння розглянемо результати зміни розмірів файлів до та після конвертації. Спочатку розглянемо зображення у форматі JPG без стиснення і порівняємо їх із файлами у форматі WebP після процесу конвертації. Крім того, здійснимо стиснення WebP файлів за допомогою різних налаштувань якості. Таблиця 4.3 містить детальні результати таких порівнянь.

Таблиця 4.3 – Порівняння розмірів зображень

JPG	WebP (100%)	WebP (80%)	WebP (50%)
108 kB	90 kB	44 kB	29 kB
108 kB	90 kB	44 kB	29 kB
106 kB	89 kB	43 kB	29 kB
106 kB	89 kB	43 kB	29 kB
107 kB	93 kB	45 kB	30 kB

Як видно з таблиці 4.3, після конвертації файли в форматі WebP займають значно менше місця порівняно з оригінальними JPG зображеннями. При використанні налаштувань якості 80% різницю між оригінальним

зображенням і стиснутим майже неможливо помітити, хоча розмір файлу зменшується майже вдвічі. Зменшення якості до 50% дає ще більше зменшення розміру файлу — майже втричі, при цьому різниця в якості вже дещо помітна, але вона не є критичною.

Такий підхід дозволяє значно зменшити розмір файлів, зберігаючи при цьому достатній рівень якості для подальшої обробки і передачі, що є важливим для забезпечення ефективної роботи системи в умовах обмежених ресурсів зберігання і пропускної здатності каналу передачі.

Наступним важливим етапом є оптимізація процесу передачі стиснених і перетворених зображень у форматі WebP від робота до сховища. Оскільки ці зображення вже були значно зменшені за розміром, важливо забезпечити ефективну та швидку передачу даних. Для цього, як і в процесі передачі відео потоку, всі зображення конвертуються у бінарний формат, оскільки бінарні запити значно менші за розмірами, що дозволяє зменшити навантаження на мережу.

Метод конвертації даних у бінарний вигляд для зображень в WebP схожий до того, що використовувався при передачі відео потоку. Однак у цьому випадку використовується інший метод передачі, а саме асинхронна передача. Кожну секунду всі оптимізовані зображення зчитуються, конвертуються в бінарний формат, після чого кожен файл асинхронно відправляється на сервер-сховище. Асинхронне надсилання дозволяє значно підвищити швидкість передачі, оскільки наступний файл відправляється без очікування завершення передачі попереднього файлу. Це суттєво зменшує час затримки і підвищує ефективність передачі, що ілюструється на рисунку 4.18.

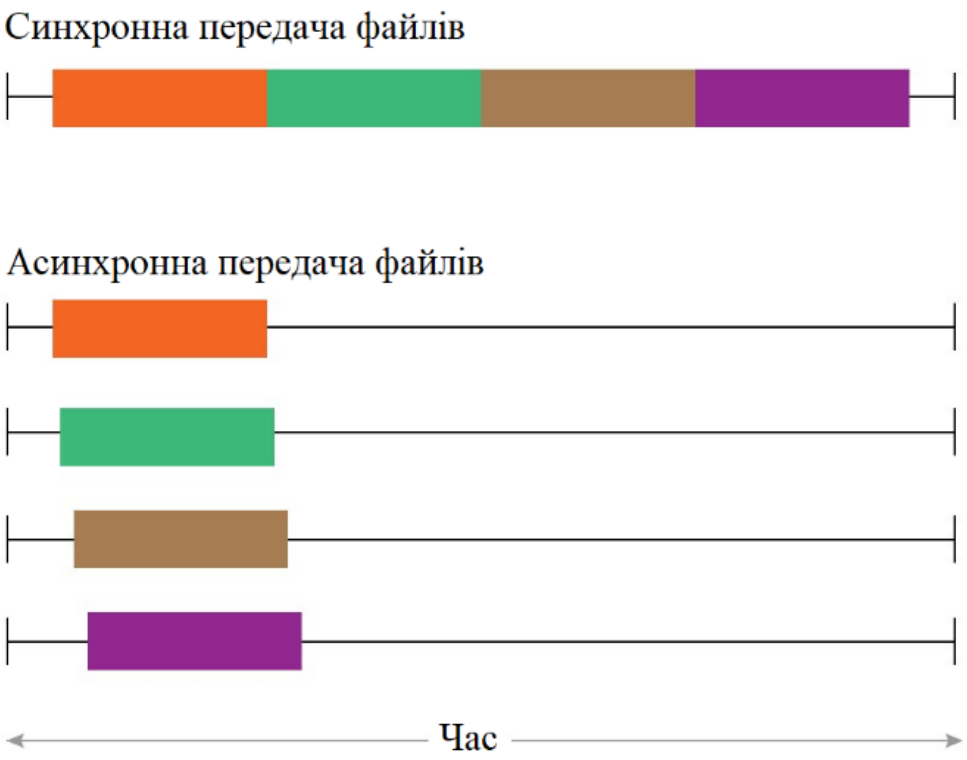


Рисунок 4.18 – Порівняння синхронної та асинхронної передачі файлів

Для забезпечення збереження фотографій на сервері створено спеціальну кінцеву точку, яка обробляє запити на збереження зображень. Коли зображення надходить на сервер, воно спочатку конвертується в необхідний формат і потім записується у відповідну папку на сервері. Залежно від дати та часу створення зображення, файл розподіляється по відповідним папкам. Якщо папки для певної дати або часу не існують, вони автоматично створюються. Цей алгоритм забезпечує впорядковане збереження зображень і дозволяє легко організувати доступ до них, що продемонстровано на рисунку 4.19.

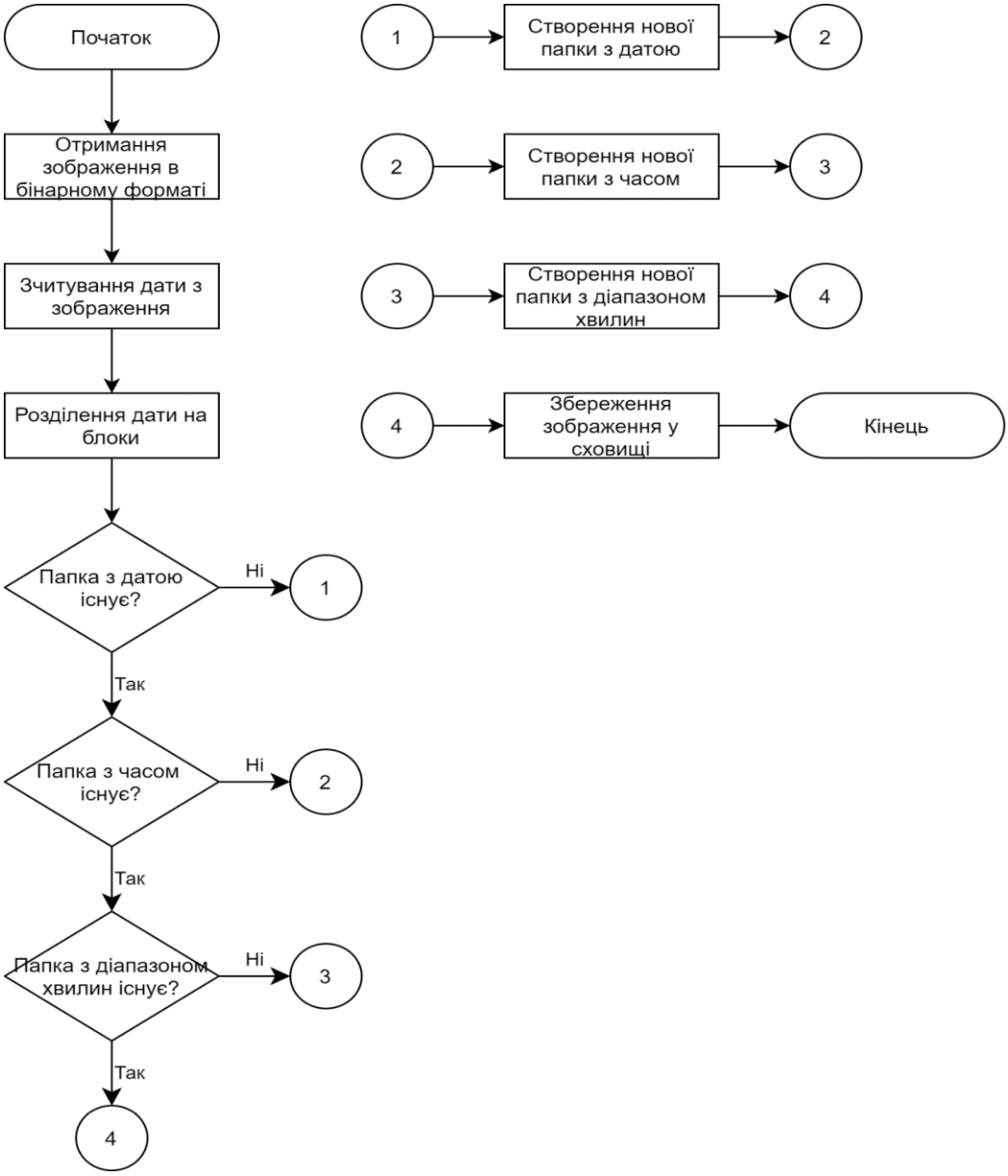


Рисунок 4.19 – Алгоритм отримання та зберігання зображення

Завдяки такій організації, процес збереження зображень на сервері стає не тільки швидким, а й надійним, оскільки кожен файл обробляється асинхронно та зберігається в систематизованому вигляді. Це дозволяє значно знизити навантаження на сервер і оптимізувати використання ресурсів, при цьому підвищуючи загальну ефективність системи.

4.6 Розробка клієнтської частини

Розробка клієнтської частини додатку починається з організації структури програмного забезпечення, що включає розподіл додатку на окремі компоненти, кожен з яких відповідає за конкретну частину функціоналу. Це дозволяє значно спростити подальше обслуговування програми, оскільки кожен компонент можна оновлювати або замінювати без суттєвих змін у решті системи. Крім того, така структура полегшує інтеграцію нових функціональних можливостей у майбутньому.

Формально можна розділити клієнтську частину додатку на дві основні складові: візуальну частину та функціональні можливості. Візуальна частина відповідає за відображення сторінки користувачеві, включаючи стилістику та наповнення, зокрема, розташування елементів на екрані. Завданням візуальної складової є забезпечення комфортного користування веб-системою, що досягається через правильне компонування інформації та мінімізацію непотрібного вмісту на сторінці. Всі елементи інтерфейсу мають бути поділені на логічні блоки, що робить взаємодію з системою зрозумілою та зручною.

Функціональні можливості в свою чергу відповідають за реакцію системи на дії користувача. Це включає обробку введених даних та виконання запитів до сервера, наприклад, при натисканні на кнопку відправляється запит, і при отриманні відповіді від сервера, дані відображаються на екрані.

Першим етапом взаємодії користувача з системою є авторизація, тому розглянемо сторінку входу. На цій сторінці користувач має можливість ввести свої дані для доступу до системи, а також є кнопка для відновлення паролю і логотип компанії. У разі введення некоректних даних, таких як неправильна поштову скриньку або пароль, на екрані з’являється відповідне повідомлення про помилку. Для цього система здійснює перевірку введених даних на відповідність збереженим у базі даних.

Сторінка авторизації має чистий і зрозумілий дизайн, не перевантажений зайвою інформацією, що робить її інтуїтивно зрозумілою та зручною для користувачів (рис. 4.20).

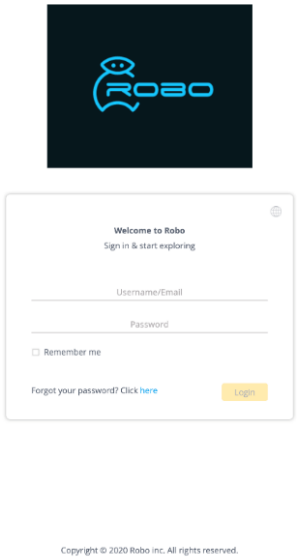


Рисунок 4.20 – Сторінка авторизації користувача

Після успішної авторизації, користувач автоматично переходить на головну сторінку системи, де в верхній частині розташовані три основні розділи: Storage History, Robot Stream, Management Page. Кожен з цих розділів має свою специфічну функціональність, яка варіюється в залежності від ролі користувача. Для різних ролей деякі компоненти можуть бути відсутніми, що забезпечує персоналізацію доступу до функцій системи.

Розглянемо кожен розділ більш детально. На сторінці Robot Stream користувач має змогу слідкувати за діяльністю робота в режимі реального часу, якщо робот увімкнений і має підключення до Інтернету. У випадку, якщо робот з певних причин не передає сигнал, на екрані з’являється модальне вікно, яке інформує користувача про причину помилки. Для цього система здійснює підключення до сервера через WebSocket, очікуючи

відповідь протягом кількох секунд. Якщо зв’язок з роботом успішний і робот дійсно передає відео потік, користувач побачить трансляцію на екрані.

На Рис. 4.21 представлено інтерфейс сторінки, де користувач може спостерігати за трансляцією відео потоку з робота в реальному часі.

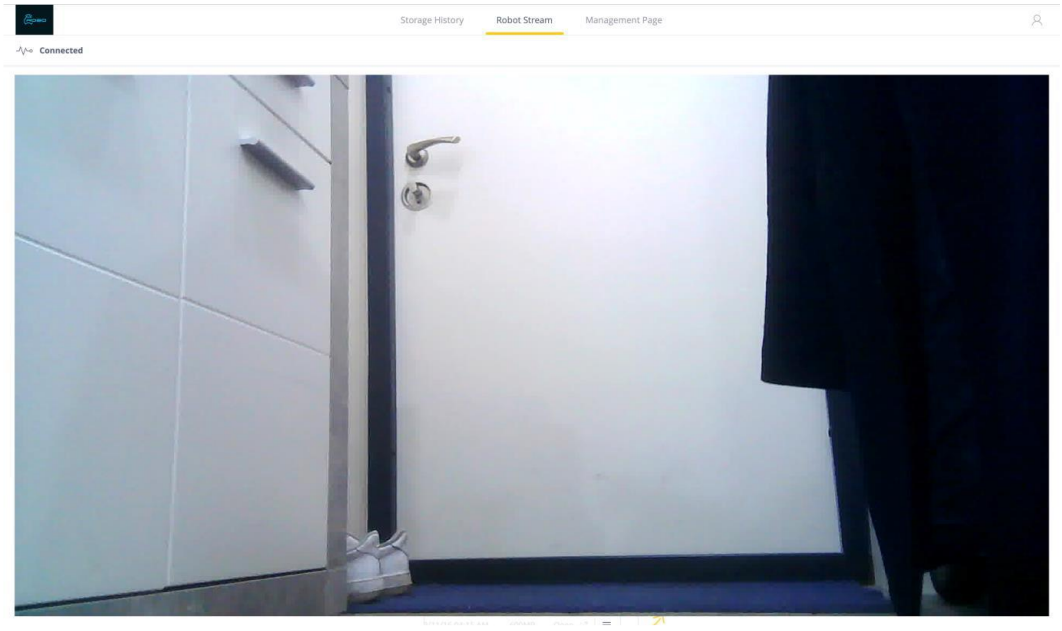


Рисунок 4.21 – Сторінка трансляції відео потоку

Сторінка Storage History надає користувачеві можливість переглядати історію усіх кадрів, які були збережені у сховищі. У разі, коли немає підключення до сховища або ж не знайдено збережених файлів, на екрані з'являється відповідне повідомлення для користувача. Якщо ж файли знайдені, на лівій панелі сторінки формуються елементи списку, які відображають дату збереження файлів у форматі DD-MM-YYYY. Кожен елемент цього списку відповідає окремій папці у сховищі, що містить відповідні файли.

При натисканні на певний елемент списку відправляється запит для отримання усіх папок, що відповідають вибраній даті. Після отримання відповіді, в тому ж елементі списку з'являється новий список, де відображається час у форматі HH, що відповідає конкретним періодам

збереження. Натискаючи на елемент, що відображає потрібний час, користувач переходить до списку десятихвилинних відрізків. Кожен з цих відрізків містить кадри, збережені протягом цього часу. Вибір відповідного відрізка викликає формування списку усіх збережених кадрів для цього періоду у головній частині сторінки.

На Рис. 4.22 зображено інтерфейс сторінки для відображення вмісту сховища, де користувач може легко переглядати і вибирати збережені кадри на основі дати та часу.

Рисунок 4.22 – Сторінка відображення вмісту сховища

Для полегшення перегляду всіх кадрів, створена кнопка Show as a Video, яка відкриває модальне вікно і дозволяє з усіх завантажених кадрів за певний проміжок часу створити неперервне відео. Зчитується інтервал між кожним окремим кадром, після чого викликається функція showVideo, яка підставляє наступний кадр у список з відповідним інтервалом. Для користувача цей процес буде непомітним, і йому здаватиметься, що транслюється відео. Щоб користувач міг перемотувати відео вперед або назад,

додано кнопки, що змінюють індекс наступного кадру на необхідну кількість кадрів, тим самим дозволяючи здійснювати перемотування відео.

Для розрахунку інтервалу між кадрами використовується наступна формула:

$$
\text{frameInterval} = \frac{\text{frame}_n - \text{frame}_{n-1}}{n}
$$

де:

- frame_n — дата поточного кадру в мілісекундах,
- n — загальна кількість кадрів.

Для обчислення зміщення індексу кадру при перемотуванні використовується така формула:

$$
\text{frameIndexOffset} = \frac{\text{timeOffset}}{\text{frameInterval}}
$$

де:

- frameIndexOffset — кількість кадрів, на яку потрібно змінити індекс,
- timeOffset — час у секундах, на який потрібно перемотати відео (за замовчуванням 5 секунд),
- frameInterval — інтервал між кадрами.

Останньою сторінкою є Management Page, яка залежить від ролі користувача. У звичайного користувача є лише можливість редагувати свій профіль через розділ User Management. Адміністратор системи має додаткові функції, зокрема перегляд і редагування профілів всіх користувачів, а також налаштування системи через System Configuration.

У розділі User Management для адміністратора відображається інформація про всіх користувачів системи. Для додавання нового користувача існує кнопка Add User, при натисканні на яку відкривається модальне вікно з формою для введення даних нового користувача. Для редагування інформації про користувача, поряд з кожним профілем є значок редагування, при натисканні на який також відкривається модальне вікно з формою редагування. Для видалення користувача потрібно натискати на значок видалення, після чого потрібно підтвердити свій вибір. Сторінка User Management продемонстрована на рис. 4.23.

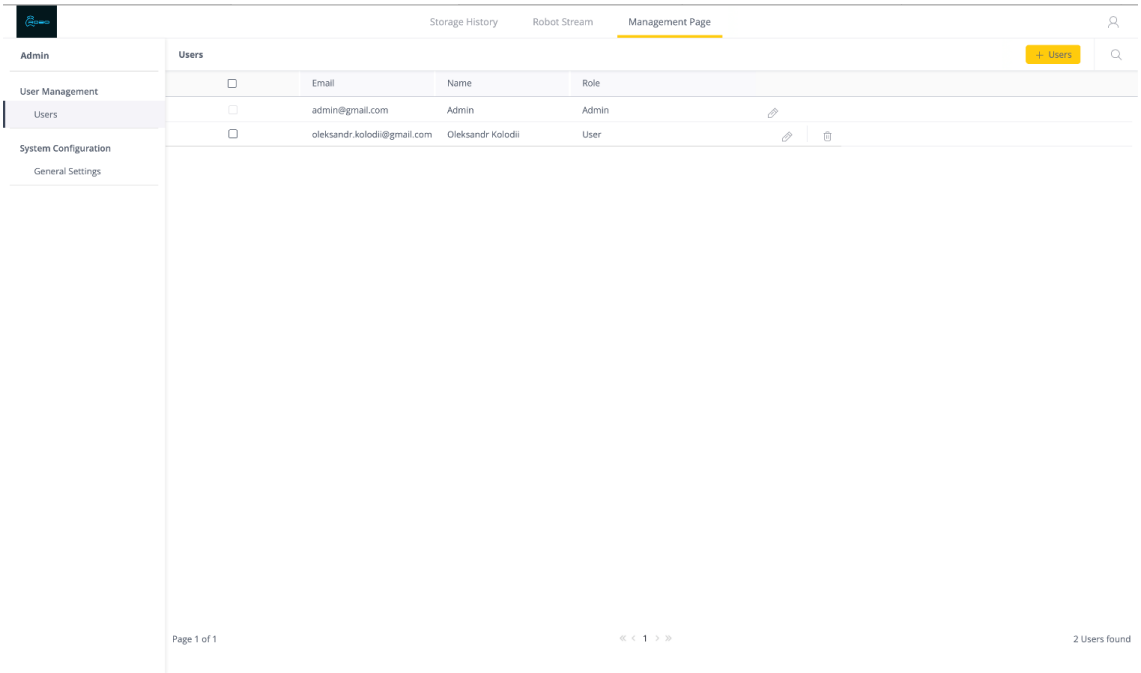


Рисунок 4.23 – Сторінка керування користувачами

У розділі System Configuration адміністратор може змінювати налаштування системи, зокрема налаштовувати якість зображення та кількість кадрів на секунду для відео трансляцій або збереження кадрів до сховища.

Особливо важливим аспектом є адаптивний веб-дизайн, створений для забезпечення оптимального відображення та взаємодії сайту з користувачем, незалежно від пристрою, на якому здійснюється перегляд сторінки.

Це дозволяє додатку коректно працювати на різних пристроях, таких як смартфони, планшети та комп'ютери, з різними роздільними здатностями екрану.

У цьому розділі також розглянута архітектура системи, описано основні компоненти, і визначено необхідні модулі для нормального функціонування системи, а також встановлена логіка взаємодії між ними.

Структура бази даних була розроблена для зберігання налаштувань системи та інформації про користувачів. Для взаємодії з базою даних створено необхідні кінцеві точки API, а також для зв'язку веб-додатку з сервером. Було розроблено серверну частину для роботи з роботом і сховищем, а також оптимізовано процес передачі зображень від камери робота до кінцевого користувача та сховища. Це включає в себе стиснення зображень та кадрів відео потоку, що покращує швидкодію системи та дозволяє збільшити обсяг збереженої інформації в сховищі.

Розробка клієнтської частини включає відображення всіх збережених фотографій у сховищі та трансляцію відео потоку з камери робота в режимі реального часу.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [19].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	5	4
2. Ринкові переваги (наявність аналогів)	4	4	3
3. Ринкові переваги (ціна продукту)	4	5	4
4. Ринкові переваги (технічні властивості)	3	4	5
5. Ринкові переваги (експлуатаційні витрати)	3	3	4
6. Ринкові перспективи (розмір ринку)	2	2	2
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4

9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	2	2	2
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	37	42	40
Середньоарифметична сума балів $СБ_с$	39,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [19].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» становить 39,7 бала, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок узагальненого коефіцієнта якості розробки

Окрім комерційного аудиту розробки доцільно також розглянути технічний рівень якості розробки, розглянувши її основні технічні показники. Ці показники по-різному впливають на загальну якість проектної розробки.

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [20]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \tag{5.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Відносні значення β_i для різних випадків розраховуємо за такими формулами:

- для показників, зростання яких вказує на підвищення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{ні}}{I_{аі}}, \tag{5.2}$$

де $I_{ні}$ та $I_{на}$ – чисельні значення конкретного *і*-го технічного показника якості відповідно для нової розробки та аналога;

- для показників, зростання яких вказує на погіршення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{аі}}{I_{ні}}; \tag{5.3}$$

Використовуючи наведені залежності можемо проаналізувати та порівняти техніко-економічні характеристики аналогу та розробки на основі отриманих наявних та проектних показників, а результати порівняння зведемо до таблиці 5.4.

Таблиця 5.4 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової	Питома вага показника

				розробки до аналога	
Точність розпізнавання об'єктів	%	89	97	1,09	0,15
Швидкість візуалізації об'єкта	мс	0,2	0,1	2	0,2
Кількість допоміжних функцій	од.	2	3	1,5	0,25
Підтримка баз даних	од.	1	2	2	0,25
Адаптація інтерфейсу	бал	8	9,5	1,2	0,15

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 1{,}09 \cdot 0{,}15 + 2 \cdot 0{,}2 + 1{,}5 \cdot 0{,}25 + 2 \cdot 0{,}25 + 1{,}2 \cdot 0{,}15 = 1{,}62.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,62 рази.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням

конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [19]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.4)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 18500{,}00 \cdot 21 / 21 = 18500{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці 5.5.

Таблиця 5.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	18500,00	880,95	21	18500,00
Інженер-розробник автоматизованих систем управління	16350,00	778,57	15	11678,57
Консультант (менеджер безпеки)	15200,00	723,81	3	2171,43
Технік	8000,00	380,95	15	5714,29
Всього				38064,29

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.5)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.6)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [19];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000{,}00 \cdot 1{,}10 \cdot 1{,}15 / (21 \cdot 8) = 60{,}24$ *грн.*
$З_{р1} = 60{,}24 \cdot 6{,}00 = 361{,}43$ *грн.*

Таблиця 5.6 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Організація робочого місця інженера-розробника інтелектуальної системи	6,00	2	1,10	60,24	361,43
Підготовка дослідного обладнання	4,00	3	1,35	73,93	295,71
Інсталяція програмного забезпечення моделювання поведінки системи	5,00	4	1,50	82,14	410,71
Монтаж дослідного	3,20	5	1,70	93,10	297,90

обладнання					
Формування бази даних результатів експерименту	8,20	2	1,10	60,24	493,95
Всього					1859,71

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.7)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (38064,29 + 1859,71) · 11 / 100% = 4391,64 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.8)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (38064,29 + 1859,71 + 4391,64) · 22 / 100% = 9749,44 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}\,, \qquad (5.9)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$M_1 = 2 \cdot 210{,}00 \cdot 1{,}03 - 0 \cdot 0 = 432{,}60$ *грн.*

Проведені розрахунки зведемо до таблиці 5.7.

Таблиця 5.7 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір 500-8 А4	210,00	2	0	0	432,60
Папір для записів А5 Bic	150,00	4	0	0	618,00
Органайзер офісний	310,00	3	0	0	957,90
Набір офісного працівника	210,00	3	0	0	648,90
Картридж для принтера	2400,00	2	0	0	4944,00
Диск оптичний	25,00	3	0	0	77,25
Flesh-пам'ять 32 GB	169,00	2	0	0	348,14
Тека для паперів А4	120,00	3	0	0	370,80
Інше	250,00	1,000	0	0	257,50
Всього					8655,09

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (5.10)$$

де $Н_j$ – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*).

$К_в = 3 \cdot 2780{,}00 \cdot 1{,}03 = 8590{,}20$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Датчики прискорення (акселерометри)	3	2780,00	8590,20
Датчики положення (гіроскопи)	3	4850,00	14986,50
Інтерфейс передачі даних	2	1400,00	2884,00
Відеокамера спостереження	2	1600,00	3296,00
Всього			29756,70

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$В_{спец} = \sum_{i=1}^{k} Ц_i \cdot С_{пр.i} \cdot К_i \ , \qquad (5.11)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.і}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k – кількість найменувань устаткування.

$B_{спец}$ = *1850,00 · 1 · 1,03 = 1905,50 грн.*

Отримані результати зведемо до таблиці 5.9:

Таблиця 5.9 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Програмний інтерфейс	1	1850,00	1905,50
Осцилограф цифровий MAX-250-JIR	1	15400,00	15862,00
Всього			17767,50

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.12)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг}$ = *40,58 · 1 · 1,02 = 41,39 грн.*

Отримані результати зведемо до таблиці 5.10:

Таблиця 5.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Програмне забезпечення середовища моделювання процесів	1	40,58	41,39
Абонплата доступу до мережі Internet	1	350,00	357,00
Прикладний пакет моделювання процесів MatLab	1	8800,00	8976,00
Всього			9374,39

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.13)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (35699{,}00 \cdot 1) / (3 \cdot 12) = 991{,}64$ *грн.*

Проведені розрахунки зведемо до таблиці 5.11.

Таблиця 5.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер	35699,00	3	1	991,64
Робоче місце розробника	7300,00	5	1	121,67
Пристрої виводу	7770,00	4	1	161,88

інформації				
Оргтехніка	7320,00	5	1	122,00
Приміщення лабораторії	410000,00	25	1	1366,67
ОС Windows 11	7320,00	3	1	203,33
Прикладний пакет Microsoft Office 2019	6500,00	3	1	180,56
Всього				3147,74

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.14)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 10,98 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,25 \cdot 160,0 \cdot 10,98 \cdot 0,95 / 0,97 = 439,20$ *грн.*

Проведені розрахунки зведемо до таблиці 5.12.

Таблиця 5.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер	0,25	160,0	439,20
Робоче місце розробника	0,01	160,0	17,57
Пристрої виводу інформації	0,10	2,5	2,75
Оргтехніка	0,32	1,4	4,74
Пристрої передачі інформації	0,02	160,0	35,14
Осцилограф цифровий MAX-250-JIR	0,25	15,0	41,18
Всього			540,57

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.15)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (38064,29 + 1859,71) · 20 / 100% = 7984,80 грн.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.16)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 55%.

$I_в$ = (38064,29 + 1859,71) · 55 / 100% = 21958,20 грн.

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.17)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (38064,29 + 1859,71) · 100 / 100% = 39924,00 грн.

Витрати на проведення науково-дослідної роботи на тему «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.18)$$

$В_{заг}$= 38064,29 +1859,71 +4391,64 +9749,44 +8655,09 +29756,70 +17767,50 + 9374,39 + 3147,74 +540,57 +7984,80 +0,00 +21958,20 +39924,00 = 193174,07 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.19)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 193174,07 / 0,95 = 203341,12 грн.

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів пристрою, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	35	50	80	70

N – кількість споживачів які використовували аналогічний пристрій у році до впровадження результатів нової науково-технічної розробки, приймемо 2000 осіб;

$Ц_б$ – вартість пристрою у році до впровадження результатів розробки, приймемо 86000,00 грн;

$\pm\Delta Ц_o$ – зміна вартості пристрою від впровадження результатів науково-технічної розробки, приймемо 3191,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [19]:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.20)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =35%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(3191,00·2000,00+89191,00·35)·0,83·0,35·(1-0,18/100%)=2263872,80 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(3191,00·2000,00+89191,00·85)·0,83·0,35·(1-0,18/100%)=3326182,21 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(3191,00·2000,00+89191,00·165)·0,83·0,35·(1-0,18/100%)=5025877,26 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4$ =(3191,00·2000,00+89191,00·235)·0,83·0,35·(1-0,18/100%)=6513110,43 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.21)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =2263872,80/(1+0,15)^1+3326182,21/(1+0,15)^2+5025877,26/(1+0,15)^3+
+6513110,43/(1+0,15)^4=1968585,05+2515071,61+3304595,88+3723892,02=11512
144,57 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.22)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 203341,12 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 203341{,}12 = 406682{,}24$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.23}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 11512144,57 грн;

PV – теперішня вартість початкових інвестицій, 406682,24 грн.

$E_{абс} = ПП - PV$ = 11512144,57 - 406682,24 = 11105462,32 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.24}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 11105462,32 грн;

PV – теперішня вартість початкових інвестицій, 406682,24 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+11105462,32/406682,24)$^{1/4}$= 1,31.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{5.25}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 1,31 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.26}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,31 = 0,77 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця» становить 39,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,62 рази.

Також термін окупності становить 0,77 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця».

ВИСНОВКИ

У магістерській дисертації «Розробка інноваційної системи відео- та фото-спостереження для забезпечення безпеки приміщень» проведено всебічне дослідження та розробку сучасної системи відео- та фото-спостереження. У першому розділі визначено об'єкт і предмет дослідження, а також проведено аналіз функціональних можливостей існуючих систем захисту приміщень, таких як Arlo Pro4 та Reolink Argus 2. Це дозволило виявити їх переваги та недоліки, на основі чого були сформульовані основні задачі дослідження: розробка ефективної системи, що забезпечує високу якість відео, зменшення обсягу даних та інтеграція з різними пристроями.

У другому розділі оцінено формати зображень та їхні особливості. Здійснений аналіз підтвердив доцільність використання формату MJPEG для відео-трансляцій, що дозволяє зберігати високу якість при оптимальному розмірі файлів. Для збереження статичних зображень обрано формат WebP, що забезпечує високий рівень стиснення без втрати якості.

Третій розділ присвячено вибору технологій для створення серверної та клієнтської частин системи. Обрано платформу Node.js з менеджером пакунків NPM для обробки зображень і відео. Базу даних реалізовано на MongoDB, яка забезпечує високу продуктивність при роботі з неструктурованими даними. Клієнтська частина створена з використанням HTML, CSS та JavaScript, а передача відео реалізована через WebSocket, що забезпечує стабільну роботу системи.

У четвертому розділі детально описано архітектуру системи, взаємодію її компонентів, структуру бази даних та API. Серверна частина забезпечує обробку та стиснення зображень, а клієнтська частина надає доступ до онлайн-трансляцій і збережених файлів.

Таким чином, інноваційна система відео- та фото-спостереження має значний потенціал для впровадження, забезпечуючи високий рівень безпеки, ефективність та зручність використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Al-Naymat, G., & Alsamara, K. (2023). Image Compression Techniques for Surveillance Systems: An Evaluation of Modern Approaches. Electronics, 12(17), 3567. MDPI.

2. Amini, Z., & Mohammadi, M. (2022). A Comparative Study of JPEG and WebP Formats in Video Surveillance Applications. International Journal of Advanced Computer Science, 11(2), 45–63.

3. Bai, X., Zhang, Y., & Li, H. (2023). Real-time Video Streaming Using MJPEG: Performance Analysis and Use Cases in Security Systems. Journal of Imaging Science, 40(3), 256–274. Springer.

4. Baker, S., & Collinson, R. (2024). Modern Challenges in Video Surveillance Systems for Smart Cities. IEEE Transactions on Multimedia, 31(2), 112–129.

5. Chen, Y., & Liu, J. (2021). Development of Real-Time Surveillance Systems with Node.js and MongoDB: A Case Study. International Journal of Web Information Systems, 17(1), 34–53. Emerald.

6. Duan, M., & Liu, G. (2023). WebSocket Protocol in Modern Surveillance Systems: Enhancing Real-time Communication. Journal of Network and Computer Applications, 77, 221–238. Elsevier.

7. Fang, H., & Wang, J. (2020). Data Storage Optimization in Surveillance Systems: Transition to NoSQL Databases. Journal of Database Management, 31(4), 65–82. IGI Global.

8. Gan, W., & Yang, M. (2023). Integrating AI and Machine Learning in Video Surveillance Systems for Enhanced Security. Journal of Artificial Intelligence Research, 14(1), 56–74.

9. Hsu, K., & Lee, S. (2022). Edge Computing in Surveillance Systems: Reducing Latency and Increasing Reliability. IEEE Communications Surveys & Tutorials, 24(1), 102–121.

10. Jackson, T., & Smith, P. (2021). Comprehensive Analysis of Video Surveillance Formats and Compression Techniques. Multimedia Tools and Applications, 80(4), 689–706.

11. Kim, W. (2023). A Survey of Video Surveillance Systems in Smart Cities. Electronics, 12(17), 3567. MDPI.

12. Lin, T., & Zhou, F. (2020). Design and Implementation of Secure Web APIs in Surveillance Systems. International Journal of Secure Software Engineering, 11(3), 25–43.

13. Martin, E., & Rodriguez, J. (2023). Evaluation of Video Surveillance Technologies in Urban Safety. Safety Science, 134, 1–19. Elsevier.

14. Patel, K., & Johnson, L. (2021). Developing Client-side Applications for Surveillance Systems Using Modern Web Technologies. International Journal of Human–Computer Interaction, 37(12), 1134–1153.

15. Qureshi, Z., & Shamsi, Z. (2022). Video Surveillance for Small Businesses: System Design and Implementation. International Journal of Innovative Computing and Applications, 14(2), 125–143. Inderscience.

16. Smith, A., & Taylor, D. (2023). Cloud-based Storage Solutions for Surveillance Systems: A Review. Future Generation Computer Systems, 133, 22–41. Elsevier.

17. Wang, Y., & Zhao, X. (2024). Optimizing Surveillance System Performance with Adaptive Streaming Protocols. IEEE Transactions on Circuits and Systems for Video Technology, 34(5), 905–924.

18. Zhang, L., & Zhou, Y. (2023). The Role of Modern Compression Algorithms in Enhancing Video Surveillance Systems. International Journal of Digital Multimedia Broadcasting, 2023, 1–20. Hindawi.

19.Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : Козловський В.О., Лесько О.Й., Кавецький В.В.. – Вінниця : ВНТУ, 2021. – 42 с.

20. Кавецький В.В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

21. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» (кафедра комп'ютерних систем управління) [Електронний ресурс] / уклад. В. М. Дубовой. – Вінниця: ВНТУ, 2023 – (PDF, 45 с.).

ДОДАТКИ

Додаток А
(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: _« **Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця** »__

Тип роботи: _____магістерська кваліфікаційна робота_____
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ _____кафедра комп'ютерних систем управління_____
(кафедра, факультет (інститут), навчальна група)

Керівник_____Ковтун В. В., професор, зав. кафедри КСУ_____
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	88%
Загальна схожість	12%

Аналіз звіту подібності (відмітити потрібне)

√ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ _____Артур НІКОНЧУК _____
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____
_____
_____
_____

Особа, відповідальна за перевірку_ _____ __ _Володимир ДУБОВОЙ _

Додаток Б

(обов'язковий)

ЗАТВЕРДЖЕНО

Зав. кафедри КСУ ВНТУ,

д.т.н., професор

_____В'ячеслав КОВТУН

" 05 " 12 2024р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

"Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця"

08-33.МКР.007.00.000 ТЗ

Студент групи: 2АКІТР-23м

Артур НІКОНЧУК

Керівник: проф. каф. КСУ

В'ячеслав КОВТУН

1. Назва та галузь застосування

1.1. Назва – Інтелектуальна автоматизована система опрацювання даних відео каналу робота-охоронця.

1.2. Галузь застосування – опрацювання даних відео каналу робота-охоронця.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від
№310 від 17-09-2024р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення ефективності процесу передачі зображень від камери роботизованої рухомої платформи до серверу для підвищення швидкодії та зменшення навантаження на сервер, за рахунок оптимізації розміру зображень та можливості зменшення кількості зроблених кадрів за секунду.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. I. Lebed, "Algorithms for Object Detection and Tracking in Security Surveillance Systems," Cybernetics and Systems Analysis, vol. 57, no. 3, pp. 227-234, 2021. DOI: 10.1007/s10559-021-00352-5.

2. I. Koval, "Video Analytics in Robot-Assisted Surveillance Systems," Ukrainian Journal of Information Technology, vol. 4, no. 2, pp. 74-81, 2023. DOI: 10.18372/ujit.v4i2.

3. V. Vostriakova, O. Rubanenko, N. Burennikova, M. Belik, and O. Lesko, "Prosuming Business Models in Transition to a Sustainable

Bioeconomy," IEEE 4th KhPI Week on Advanced Technology, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10312899.

5. Вимоги до розробки.

Для досягнення поставленої мети в роботі визначені наступні завдання:

- Проаналізувати існуючі функціональні можливості та аналоги систем захисту приміщень, виявити їх переваги та недоліки;

- На основі аналізу сучасного стану різних типів форматів зображень та можливості їх оптимізації, обрати найкращий тип для подальшого стиснення розміру зображення;

- Провести експериментальне дослідження швидкодії різних методів оптимізації зображень та обрати той метод, в результаті якого, отримане зображення буде одним з найменших по розміру та нас буде задовольняти його якість;

- Розробити програмну реалізацію по оптимізації зображень, які надходять з відеокамери та передача оптимізованих файлів до серверу;

- Розробка веб додатку для надання можливості користувачеві переглядати файли, збережені на сервері.

В даному розділі розглянуто основні функціональні можливості сучасних систем захисту приміщень, а також проведено аналіз існуючих найпопулярніших систем, виявлено основні їх переваги та недоліки. Першою розглянутою системою була камера Arlo Pro4, недоліками якої була необхідність у придбанні базової станції, на яку здійснюється запис файлів, а для розширення і доступу до хмарного сховища необхідна щомісячна або річна підписка. Також, не дуже зручним є необхідність завантажувати мобільний додаток на свій смартфон, щоб отримати доступ до віддаленого перегляду за

камерою. Критичним є те, що за допомогою комп’ютера, можливості підключитись віддалено не існує. До переваг даної системи можна віднести дуже хорошу якість зйомки та велику кількість функціональних можливостей, таких як система виявлення руху або нічна зйомка. Інша розглянута система має досить схожі функціональні можливості, але на відміну від першої – файли можна зберігати у хмарне сховище безкоштовно, без необхідності у зовнішніх накопичувачах інформації, що є дуже хорошим плюсом. В даній системі так само необхідно завантажувати мобільний додаток, для доступу до камери, хоча функціональних можливостей у даному додатку значно більше, оскільки користувач має змогу не тільки переглядати файли і дивитись, що відбувається в приміщенні в даний момент, а й налаштувати якість зйомки. Основним недоліком системи є обмеженість відео в 15 кадрів за секунду та велика переривчастість відео потоку.

Врахувавши усі проаналізовані переваги та недоліки обох систем визначено мету, предмет та об’єкт дослідження. Також визначено основні задачі, які необхідно виконати для досягнення зазначеної мети.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1) Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «15».10. 2024 р.

2) Удосконалення технології прийняття рішень при автоматизації об’єкту управління «24».10. 2024 р.

3) Визначення технічних характеристик системи
«25».11. 2024 р.

4) Розробка програмного забезпечення системи
«15».11. 2024 р.

6.2 Графічні матеріали:

1) Розробка UML-діаграм системи «15».11. 2024 р.
2) Розробка моделі бази даних системи «15».11.2024р.
3) Тестування програмного забезпечення «15».11.2024 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28».11. 2024 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «01».12.2024 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «10».12. 2024 р.

ДОДАТОК В

(вибірковий)

Лістинг коду

service_predictor/service/predictor/metrics_article.py

Base

```
import logging
from datetime import datetime, timedelta from collections import defaultdict
from typing import Any

 Internal
from service_predictor.service.predictor._predictor import Predictor
from service_predictor.service.connection import ConnectionElastic, ConnectionSQL

 External
from tqdm import tqdm import pandas as pd import numpy as np
from prophet import Prophet



class TelegramPredictorMetrics(Predictor): type = 'telegram'

progress_bar = False

 Metrics to predict metrics = [
'views',
]

 Fields to save from Elasticsearch
 fields = [ 'id',

'chat_id', 'message_id',

'delta',

'date', 'loading_date',
]

 Minimum articles to predict on minimum_articles = 100
```

```
 Quantile to cut outliers for better model fit (no anomalies in the train data)
cut_quantile = 0.95

 Prediction step (minutes between two predictions) step = 5

 Size of uncertainty intervals (bigger value == bigger intervals) uncertainty_interval_width = 0.95

def init (self, connection_elastic: ConnectionElastic, connection_sql: ConnectionSQL | None = None):

"""
Telegram predictor

:param connection_elastic: ConnectionElastic object
:param connection_sql: ConnectionSQL object """

 Disable annoying logging from Prophet and cmdstanpy logging.getLogger('prophet').setLevel(logging.WARNING) logging.getLogger('cmdstanpy').disabled = True

self. connection_elastic = connection_elastic self. connection_sql = connection_sql

def run(self, index: str, from_date: datetime, to_date: datetime, predict_to: datetime) -> (list[dict], list[dict]):
"""
Grabs articles and create predictions for given index and dates

:param index: Elastic index (only with supported data)
:param from_date: Starting date of train data
:param to_date: Ending date of train data and starting date for prediction
:param predict_to: Ending date of prediction
:return: """

articles = self. connection_elastic.get_articles( index=index,
from_date=from_date, to_date=to_date,
fields=self. fields + self.metrics
)

grouped_articles = self._group_articles(articles) del articles

predictions, statistics = self._predict_all(grouped_articles, to_date, predict_to)
return predictions, statistics

def _group_articles(self, articles: list[dict]) -> defaultdict[Any, defaultdict[Any, list]]:
```

```
"""
Group articles by chat_id and delta

:param articles: List of articles
:return: Dict with articles """

grouped_articles = defaultdict(lambda: defaultdict(list)) for article in articles:
grouped_articles[article['chat_id']][article['delta']].append(article) return grouped_articles
def _predict_all(self, grouped_articles: defaultdict[Any, defaultdict[Any, list]], predict_from: datetime,
predict_to: datetime) -> (list[dict], list[dict]):
"""
Predict all metrics for all articles in grouped_articles

:param grouped_articles: Grouped articles dict

:param predict_from: Prediction start date
:param predict_to: Prediction end date
:return: """




60))

 Create future dataframe
delta_minutes = int(np.ceil((predict_to - predict_from).total_seconds() /



future_df = pd.DataFrame([ predict_from.replace(minute=0, second=0) +
timedelta(minutes=x) for x in range(0, delta_minutes + 1, self.step)
], columns=['ds'])

 Predict for each chat_id and delta pair predictions = []
statistics = []

for chat_id in tqdm(grouped_articles, disable=not self.progress_bar, desc='Predicting'):
for delta in grouped_articles[chat_id]: articles = grouped_articles[chat_id][delta]

if len(articles) < self.minimum_articles:
logging.info(f'Not enough articles for chat_id {chat_id} and delta {delta}, skipping...')
continue
```

```
future_df)

prediction, statistic = self._predict_metrics(articles,


for row in prediction: predictions.append({
'chat_id': chat_id, 'delta': delta, 'date': row,

prediction[row]
})

for row in statistic: statistics.append({
'chat_id': chat_id, 'delta': delta, 'metric': row,

'date_from': predict_from, 'date_to': predict_to,

statistic[row]
})

return predictions, statistics

def _predict_metrics(self, articles: list[dict], future_df: pd.DataFrame) -> (dict, dict):
"""
Predict all metrics for given articles and future dataframe

:param articles: List of articles to fit model on
:param future_df: List of dates to predict on
:return: Prediction dict and statistic dict """

prediction = defaultdict(dict) statistics = defaultdict(dict)

for metric in self.metrics:
 Prepare data for model fitting
data = [[article['loading_date'], article[metric]] for article in

articles]
```

```
df = pd.DataFrame(data, columns=['ds', 'y'])

 Cut biggest outliers to not screw up the model
df = df[df['y'] < df['y'].quantile(self.cut_quantile)]

 Fit model
model = Prophet(interval_width=self.uncertainty_interval_width) model.fit(df)

 Form past dataframe for score calculation past_df = pd.DataFrame([
article['loading_date'] for article in articles
], columns=['ds'])

 Group all dataframes to speed up the prediction process all_df = pd.concat([past_df, future_df])

 Predict
forecast = model.predict(all_df)

split_date = future_df['ds'].min()
forecast_future = forecast[forecast['ds'] >= split_date] forecast_old = forecast[forecast['ds'] < split_date]



 Calculate deviation from real data for past data metric_statistic = []
for article in articles:
predicted_row = forecast_old[forecast_old['ds'] == article['loading_date']]

try:
predicted_value = predicted_row['yhat'].values[0]




cut

 Skip articles without predicted value due to biggest outliers

except IndexError: continue

 Skip articles with predicted value <= 0 (it's impossible) if predicted_value <= 0:
continue

upper_predicted_value = predicted_row['yhat_upper'].values[0] actual_value = article[metric]
```

```
 Skip articles without metric if actual_value is None:
continue

 Skip articles with metric outside of uncertainty interval if actual_value < upper_predicted_value:
continue

deviation = article[metric] / predicted_value metric_statistic.append(deviation)

 Add prediction to results

for index, row in forecast_future.iterrows(): prediction[row['ds']].update({
metric: row['yhat'], f'{metric}_lower': row['yhat_lower'], f'{metric}_upper': row['yhat_upper']
})

statistics_valid = len(metric_statistic) > 0
 Add statistic to results statistics[metric] = {
'std': np.std(metric_statistic) if statistics_valid else None, 'mean': np.mean(metric_statistic) if statistics_valid else None, 'min': np.min(metric_statistic) if statistics_valid else None, 'max': np.max(metric_statistic) if statistics_valid else None,
}

return prediction, statistics



service_predictor/service/predictor/metrics_index.py
 Base
import logging
from datetime import datetime, timedelta

 Internal
from service_predictor.service.predictor._predictor import Predictor
from service_predictor.service.connection import ConnectionElastic, ConnectionSQL

 External
from tqdm import tqdm import pandas as pd import numpy as np
from prophet import Prophet



class IndexPredictorMetrics(Predictor): type = 'index'

progress_bar = False
```

```
 Article grouping interval interval = timedelta(hours=1)

 Size of uncertainty intervals (bigger value == bigger intervals) uncertainty_interval_width = 0.95

def init (self, connection_elastic: ConnectionElastic, connection_sql: ConnectionSQL | None = None):
"""
Telegram predictor

:param connection_elastic: ConnectionElastic object
:param connection_sql: ConnectionSQL object """

 Disable annoying logging from Prophet and cmdstanpy logging.getLogger('prophet').setLevel(logging.WARNING) logging.getLogger('cmdstanpy').disabled = True

self. connection_elastic = connection_elastic self. connection_sql = connection_sql

def run(self, index: str, from_date: datetime, to_date: datetime, predict_to: datetime) -> (list[dict], list[dict]):
"""
Grabs articles and create predictions for given index and dates

:param index: Elastic index (only with supported data)
:param from_date: Starting date of train data
:param to_date: Ending date of train data and starting date for prediction
:param predict_to: Ending date of prediction
:return: """

articles = self. connection_elastic.get_articles( index=index,
from_date=from_date, to_date=to_date, fields=['date'], date_field='date',
)

grouped_articles = self._group_articles(articles, from_date, to_date) del articles

predictions, statistics = self._predict_all(index, grouped_articles, to_date, predict_to)
return predictions, statistics

def _group_articles(self, articles: list[dict], from_date: datetime, to_date: datetime) \


"""
```

```
-> dict[(datetime, datetime), int]:

Count articles per given interval

:param articles: List of articles
:param from_date: Starting date of train data
:param to_date: Ending date of train data and starting date for prediction
:return: Dict with articles """

 Calculate number of intervals between from_date and to_date interval_count = int(np.ceil((to_date - from_date).total_seconds() /
self.interval.total_seconds()))

counted_articles = {}
for i in range(interval_count):
interval_start = from_date + i self.interval interval_end = interval_start + self.interval interval_pair = (interval_start, interval_end)

counted_articles[interval_pair] = 0

 Suboptimal, but moderately good implementation
for article in tqdm(articles, disable=not self.progress_bar, desc='Article processing'):
article_date = datetime.strptime(article['date'], '%Y-%m-
%dT%H:%M:%S')

for interval_pair in counted_articles:
if interval_pair[0] <= article_date < interval_pair[1]: counted_articles[interval_pair] += 1
break return counted_articles

def _predict_all(self, index: str, grouped_articles: dict[(datetime, datetime), int], predict_from: datetime,
predict_to: datetime) -> (list[dict], list[dict]):
"""
Predict all metrics for all articles in grouped_articles

:param grouped_articles: Grouped articles dict
:param predict_from: Prediction start date
:param predict_to: Prediction end date
:return: """

 Create future dataframe
interval_count = int(np.ceil((predict_to - predict_from).total_seconds()
```

```
/ self.interval.total_seconds()))

future_df = pd.DataFrame([
predict_from.replace(minute=0, second=0) + x self.interval for x in range(interval_count)
], columns=['ds'])

 Predict for each chat_id and delta pair df = pd.DataFrame(
[(interval_pair[0], count) for interval_pair, count in grouped_articles.items()], columns=['ds', 'y']
)

 Fit model
model = Prophet(interval_width=self.uncertainty_interval_width) model.fit(df)

past_df = pd.DataFrame([
interval_pair[0] for interval_pair in grouped_articles
], columns=['ds'])

 Group all dataframes to speed up the prediction process all_df = pd.concat([past_df, future_df])

 Predict
forecast = model.predict(all_df)

 Uncomment to plot prediction in Jupyter Notebook (for debugging)
 model.plot(forecast)

split_date = future_df['ds'].min()
forecast_future = forecast[forecast['ds'] >= split_date] forecast_old = forecast[forecast['ds'] < split_date]

 Calculate deviation from real data for past data metric_statistic = []
for interval_pair, count in grouped_articles.items():
predicted_row = forecast_old[forecast_old['ds'] == interval_pair[0]]



try:



predicted_value = predicted_row['yhat'].values[0]



 Skip articles without predicted value due to biggest outliers cut except IndexError:
continue
```

```
 Skip articles with predicted value <= 0 (it's impossible) if predicted_value <= 0:
continue

upper_predicted_value = predicted_row['yhat_upper'].values[0]

 Skip articles with metric outside of uncertainty interval if count < upper_predicted_value:
continue

deviation = count / predicted_value metric_statistic.append(deviation)

statistics = [{ 'destination': index,

'date_from': predict_from, 'date_to': predict_to,

'std': np.std(metric_statistic), 'mean': np.mean(metric_statistic), 'min': np.min(metric_statistic), 'max': np.max(metric_statistic),
}]

 Add prediction to results prediction = []
for _, row in forecast_future.iterrows(): row = {
'destination': index, 'date': row['ds'],



else 0,

else 0,

'count': int(row['yhat']) if row['yhat'] > 0 else 0, 'count_lower': int(row['yhat_lower']) if row['yhat_lower'] > 0

'count_upper': int(row['yhat_upper']) if row['yhat_upper'] > 0

}

prediction.append(row) return prediction, statistics

service_predictor/container_predictor/container_predictor.py
 Base import json import os
import logging
from datetime import datetime, timedelta
```

```
 Import from project from pathlib import Path import sys

working_dir = Path( file ).parents[2].resolve() sys.path.append(str(working_dir))

 Internal
from service_predictor.service.connection import ConnectionSQL, ConnectionElastic from service_predictor.service.predictor import (
TelegramPredictorMetrics, IndexPredictorMetrics
)

 External
from mnv_data_container_template.container_template import Container

from mnv_queue_adapter.types import ACK, QueueMessage


class ContainerPredictor(Container):
 version = '1.0.0'
_service_name = 'predictor' def init (self):
"""
Grabber container object """
 Required config groups for container self._config_services_groups = ['elasticsearch']

 Setup params self._setup_pubsub = True self._setup_rabbitmq = False self._setup_versioning = False

super(). init ()

def _setup_container(self): """
Additional setup for grabber container

:return: """

self.input_subscription = 'demo-pipeline-predictor-sub'

 Setup Elasticsearch self._setup_elastic()

 Setup metrics database self._setup_database_art()

 Setup predictor self._setup_predictor()
```

```
def _setup_elastic(self): """
Setup Elastic connection """
self.elastic_url = self._service_values['elasticsearch_url']
self. elastic_api_key = self._gcloud_secrets.get_secret('pipeline-data- key-elasticsearch')

self.connection_elastic = ConnectionElastic(self.elastic_url, self. elastic_api_key)

def _setup_database_art(self): """
Setup database connection to metrics database """
db_hostname = os.environ.get('ART_MYSQL_HOSTNAME') if 'ART_MYSQL_PORT' in os.environ:

port = os.environ['ART_MYSQL_PORT'] db_hostname = f'{db_hostname}:{port}'

db_database = os.environ.get('ART_MYSQL_DATABASE') db_login = os.environ.get('ART_MYSQL_LOGIN')
db_password = self._gcloud_secrets.get_secret('pipeline-data-password- sql-art')

self.connection_sql = ConnectionSQL(db_login, db_password, db_hostname, db_database)

def _setup_predictor(self): """
Setup predictor object """
self.predictor_telegram_metrics =
TelegramPredictorMetrics(self.connection_elastic, self.connection_db) self.predictor_index_metrics =
IndexPredictorMetrics(self.connection_elastic, self.connection_db)

def _telegram_prediction(self, message: dict): """
Run prediction for one index with telegram engagement metrics

:param message: Message from queue with run params """

logging.info(f"Telegram prediction for {message['index']} started")

 Message example:
 {
 "type": "telegram_metrics"
 "index": "diploma_demo_tg_engagement",
 "anchor_date": "2023-12-11T12:34:56"
 }

index = message['index']
```

```
Check if index is valid, cause we cant use general telegram index assert 'engagement' in index, 'Index must contain engagement in name, e.g.
telegram_metrics_engagement'

Set anchor date to today if not specified
anchor_date = datetime.now().replace(hour=0, minute=0, second=0, microsecond=0)
if message.get('anchor_date'):
anchor_date = datetime.strptime(message['anchor_date'], '%Y-%m-
%dT%H:%M:%S')

Setup secondary dates
from_date = anchor_date - timedelta(days=7) to_date = anchor_date
to_predict = anchor_date + timedelta(days=1)

Run prediction
predictions, statistics = self.predictor_telegram_metrics.run(index, from_date, to_date, to_predict)

self.connection_sql.upsert_data('metrics_telegram_statistic', statistics)
self.connection_sql.upsert_data('metrics_telegram_prediction', predictions)


logging.info(f"Telegram prediction for {message['index']} finished")

def _index_prediction(self, message: dict): """
Run prediction for one index with index metrics

:param message: Message from queue with run params """

logging.info(f"Index prediction for {message['index']} started")

Message example:
{
"type": "index_metrics"
"index": "diploma_demo_tg",
"anchor_date": "2023-12-11T12:34:56"
}

index = message['index']

Check if index is valid, because we cant use engagement index due to its instability,
two different dates and inflated count
```

```
        assert 'engagement' not in index, 'This predictor does not support engagement index'

        anchor_date = datetime.now().replace(hour=0, minute=0, second=0, microsecond=0)
        if message.get('anchor_date'):
        anchor_date = datetime.strptime(message['anchor_date'], '%Y-%m-
        %dT%H:%M:%S')

         Setup secondary dates
        from_date = anchor_date - timedelta(days=7) to_date = anchor_date
        to_predict = anchor_date + timedelta(days=1)

         Run prediction
        predictions, statistics = self.predictor_index_metrics.run(index, from_date, to_date, to_predict)

        self.connection_sql.upsert_data('metrics_index_statistic', statistics)
self.connection_sql.upsert_data('metrics_index_prediction', predictions)
        logging.info(f"Index prediction for {message['index']} finished") def _run_predictions(self, message:
QueueMessage):
        """
        Run prediction from config message

        :param message: Message from queue with run params """

        message = json.loads(message.message)

        if message['type'] == 'telegram_metrics': self._telegram_prediction(message)

        elif message['type'] == 'index_metrics': self._index_prediction(message)

        else:
        raise ValueError(f"Unknown message type {message['type']}")



        def run(self): """
        Start grabbing tasks """
        with self._pubsub_client.publisher() as self._queue_pub:
        with self._pubsub_client.subscriber(self.input_subscription) as (queue_sub, queue_ack):
        while True:
        message = queue_sub.get() self._run_predictions(message) queue_ack.put((ACK.ACK, message))
```

if name == ' main ':
processor = ContainerPredictor() processor.run()

service_detector/detector/detectors/doc_metrics.py
Base
from datetime import datetime

Internal
from service_detector.detector.detectors._detector import Detector from service_detector.detector.utils import TelegramPredictionCache from service_detector.detector.types.anomaly import Anomaly

External
import numpy as np

class DocMetricAnomalyDetector(Detector):
must_have_fields = ['chat_id', 'delta', 'loading_date', 'views', 'forwards', 'reaction_count']

anomaly_weights = { 'predicted_metrics': 200,
'static_metrics': 100
}

def init (self, telegram_prediction_cache: TelegramPredictionCache): """
Telegram metric_name anomaly detector

:param telegram_prediction_cache: Telegram prediction cache object """

self._telegram_prediction_cache = telegram_prediction_cache

def run(self, metadata: dict, article: dict, anomalies: list[Anomaly]): """
Detect anomalies in Telegram metrics for a article based on metrics

:param metadata: Article metadata
:param article: Input article
:param anomalies: Anomalies list
:return: """

If article is not valid, we can't detect anomalies for this article if not self._check_article_fields(article):
return

```
            If article has minimal threshold for anomaly detection if not self._check_article_minimal_threshold(article):
            return

            Check static metrics self._check_static_metrics(article, anomalies)

            Check predicted metrics self._check_predicted_metrics(article, anomalies)

        def _check_static_metrics(self, article: dict, anomalies: list[Anomaly]): """
        Check static metrics in article

        :param article: Input article
        :param anomalies: Anomalies list
        :return: """

            Get coefficients for the chat
        coefficients = self._telegram_prediction_cache.get_coefficients( chat_id=article['chat_id'],
        )

            If no coefficients found, we can't detect anomalies for this article if coefficients is None:
            return

            Calculate metrics
        forwards_by_views = article['forwards'] / article['views'] reaction_count_by_views =
article['reaction_count'] / article['views']

        if forwards_by_views > coefficients['forwards_by_views']: anomalies.append(
        Anomaly(
        metric_name='forwards_by_views', metric_value=forwards_by_views,
expected_value=coefficients['forwards_by_views'], score=self._score_static_metrics(
        metric_value=forwards_by_views, expected_value=coefficients['forwards_by_views']
        ),

        _weight=self.anomaly_weights['static_metrics']
        )
        )

        if reaction_count_by_views > coefficients['reaction_count_by_views']: anomalies.append(
        Anomaly(
        metric_name='reaction_count_by_views', metric_value=reaction_count_by_views,
expected_value=coefficients['reaction_count_by_views'], score=self._score_static_metrics(
        metric_value=reaction_count_by_views, expected_value=coefficients['reaction_count_by_views']
```

```
            ),

            _weight=self.anomaly_weights['static_metrics']
            )

            )

            def _check_predicted_metrics(self, article: dict, anomalies: list[Anomaly]): """
            Check predicted metrics in article

            :param article: Input article
            :param anomalies: Anomalies list
            :return: """

             Get nearest prediction for the article
            prediction = self._telegram_prediction_cache.get_prediction( date=datetime.strptime(article['loading_date'],
            self._datetime_format),
            chat_id=article['chat_id'], delta=article['delta']
            )

            statistics = self._telegram_prediction_cache.get_statistics( date=datetime.strptime(article['loading_date'],
            self._datetime_format),
            chat_id=article['chat_id'], delta=article['delta'], metric='views'
            )

             If no prediction found, we can't detect anomalies for this article if prediction is None:
            return

            if article['views'] > prediction['views_upper']: anomalies.append(
            Anomaly(
            metric_name='views',              metric_value=article['views'],              expected_value=prediction['views'],
score=self._score_predicted_metrics(
            metric_value=article['views'], expected_value=prediction['views'], statistics=statistics
            ),

            _weight=self.anomaly_weights['predicted_metrics']
            )
            )

            def _score_predicted_metrics(self, metric_value: float, expected_value: float, statistics: dict) -> float:
            """
```

```
Calculate anomaly score for predicted metrics

:param metric_value: Metric value
:param expected_value: Expected value
:param statistics: Statistics dict
:return: Anomaly score """

x_offset = 2
x_stretch = 1.5

deviation = metric_value / expected_value if statistics:

z_score = (deviation - statistics['mean']) / statistics['std'] else:
z_score = deviation

return 1 / (1 + np.exp(-z_score / x_stretch + x_offset))

def _score_static_metrics(self, metric_value: float, expected_value: float) -
> float:
"""
Calculate anomaly score

:param metric_value: Metric value
:param expected_value: Expected value
:return: Anomaly score """

deviation = metric_value / expected_value
return 1 / (1 + np.exp((-deviation + expected_value) 10))

def _check_article_minimal_threshold(self, article: dict) -> bool: """
Check if article has minimal threshold for anomaly detection

:param article: Input article
:return: True if article has minimal threshold for anomaly detection """

 Get coefficients for the chat
coefficients = self._telegram_prediction_cache.get_coefficients( chat_id=article['chat_id'],
)

 If no coefficients found, we can't detect anomalies for this article if coefficients is None:
return False
```

```
 Check if article views are too low
if article['views'] < coefficients.get('minimal_views_threshold', 0): return False

return True

def _check_article_fields(self, article: dict) -> bool: """
Check if detector should check an article based on field presence

:param article: Input article
:return: True if detector should check an article """

return all(article.get(key) for key in self. must_have_fields)



service_detector/detector/detectors/text_narrative.py
 Base
from collections import defaultdict

 Internal
from service_detector.detector.detectors._detector import Detector from service_detector.detector.utils import NarrativeCache
from service_detector.detector.types.anomaly import Anomaly

 External
from sklearn.metrics.pairwise import cosine_similarity



class TextNarrativeAnomalyDetector(Detector):
_datetime_format = '%Y-%m-%dT%H:%M:%S'
 must_have_fields = ['translated_text', 'embeddings']

def init (self, narrative_cache: NarrativeCache): """
Telegram metric_name anomaly detector

:param narrative_cache: Narrative cache object """

self._narrative_cache = narrative_cache

def run(self, metadata: dict, article: dict, anomalies: list[Anomaly]): """
Detect anomalies in Telegram metrics for an article based on metrics
```

```
:param metadata: Article metadata
:param article: Input article
:param anomalies: Anomalies list
:return: """

 If article is not valid, we can't detect anomalies for this article if not self._check_article_fields(article):
return

self._check_narrative_similarity(metadata, article, anomalies)

def _check_narrative_similarity(self, metadata: dict, article: dict, anomalies: list) -> bool:
"""
Check if original chat country and forward chat country are different

:param metadata: Article metadata
:param article: Input article
:param anomalies: Anomalies list to append anomalies
:return: True if countries are different """

 Get text vectors and check if it fits the model text_vectors = article.get('embeddings', {})
if not text_vectors: return False

 Get all text vector model names text_vector_models = list(text_vectors.keys())

project = metadata['project']
narratives_for_destination =
self._narrative_cache.get_narratives(project)

 Get all narrative model names
narratives_models = [narrative['model'] for narrative in narratives_for_destination]
narratives_per_model = defaultdict(list)
for narrative in narratives_for_destination: narratives_per_model[narrative['model']].append(narrative)

 Get all common model names

common_models = set(text_vector_models) & set(narratives_models)

for model in common_models:
narratives_vectors = [narrative['vector'] for narrative in narratives_per_model[model]]
text_vectors = [text_vectors[model]]
```

```
 Calculate cosine similarity
sim = cosine_similarity(text_vectors, narratives_vectors)

 Check if cosine similarity is higher than threshold
 If it is, add an anomaly
for i, narrative in enumerate(narratives_per_model[model]): if sim[0][i] > narrative['threshold']:
score = float(sim[0][i]) anomalies.append(
Anomaly(
metric_name='narrative_similarity', metric_value=score, expected_value=narrative['narrative'], score=score,
)
)

def _check_article_fields(self, article: dict) -> bool: """
Check if detector should check an article based on field presence

:param article: Input article
:return: True if detector should check an article """

return all(article.get(key) for key in self. must_have_fields)



service_detector/detector/utils/metric_prediction_cache.py
 Base
from cachetools import TTLCache
from datetime import datetime, timedelta

 Internal
from mnv_data_package_database.connections.mysql import ConnectionDB

 External



class MetricPredictionCache:
telegram_prediction_table = 'metrics_telegram_prediction' telegram_statistic_table = 'metrics_telegram_statistic'
telegram_coefficient_table = 'metrics_telegram_coefficient' def init (self,
connection_db_metrics: ConnectionDB, prediction_step: int = 5,

cache_big_size: int = 500000,
```

cache_big_update_window: timedelta = timedelta(hours=1), cache_big_retention_window: timedelta = timedelta(hours=1),

cache_small_size: int = 10000,
cache_small_update_window: timedelta = timedelta(hours=1),

"""

cache_small_retention_window: timedelta = timedelta(hours=1),
):

Prediction cache object

:param connection_db_metrics: ConnectionDB object to ART database
:param prediction_step: """

self._connection_db_metrics = connection_db_metrics self._prediction_step = prediction_step

self.cache_big_update_window = cache_big_update_window self.cache_big_retention_window =
cache_big_retention_window.total_seconds() self.cache_small_update_window = cache_small_update_window
self.cache_small_retention_window =
cache_small_retention_window.total_seconds()

self._predictions = TTLCache(maxsize=cache_big_size, ttl=self.cache_big_retention_window)
self._statistics = TTLCache(maxsize=cache_small_size, ttl=self.cache_small_retention_window)
self._coefficients = TTLCache(maxsize=cache_small_size, ttl=self.cache_small_retention_window)

now = datetime.now()

self._get_predictions_from_db(now - self.cache_big_update_window, now + self.cache_big_update_window)
self._get_statistics_from_db(now) self._get_coefficients_from_db()

def get_prediction(self, date: datetime, chat_id: int, delta: int) -> dict | None:
"""
Get prediction for date, chat_id and delta combination from cache or None if not found

:param date: Date to predict on

```
        :param chat_id: Chat ID to predict on
        :param delta: Delta to predict on
        :return: Prediction dict """

        date = self._get_nearest_date(date)

         Check if prediction is in cache
        prediction = self._predictions.get((date, chat_id, delta)) if prediction is not None:
        return prediction

         If prediction is not in cache, try to get it from database with cache update window
        self._get_predictions_from_db(date - self.cache_big_update_window, date +
self.cache_big_update_window)

         Try to get prediction from cache again
         If prediction is not in cache, return None
        prediction = self._predictions.get((date, chat_id, delta)) return prediction

        def get_statistics(self, date: datetime, chat_id: int, delta: int, metric: str) -> dict | None:

        """
        Get statistic for date, chat_id, delta and metric combination from cache or None if not found

        :param date: Date for statistic
        :param chat_id: Chat ID for statistic
        :param delta: Delta for statistic
        :param metric: Metric name
        :return: """

        def get_latest_statistics():
         Check if statistic is in cache rows_for_date = {}
        for row in self._statistics:
        if row[0] < date < row[1] and row[2] == chat_id and row[3] == delta and row[4] == metric:
        rows_for_date[row[1]] = self._statistics[row]

        if len(rows_for_date) > 0:
        return sorted(rows_for_date.items(), key=lambda x: x[0])[-1][1]

        else:
        return None
```

```
 Check if statistic is in cache statistic = get_latest_statistics() if statistic is not None:
return statistic

 If statistic is not in cache, try to get it from database with cache update window
self._get_statistics_from_db(date)

 Try to get statistic from cache again
 If statistic is not in cache, return None statistic = get_latest_statistics()
return statistic

def get_coefficients(self, chat_id: int) -> dict | None: """
Get coefficients for chat_id from cache or None if not found

:param chat_id: Chat ID to get coefficients for
:return: """

 Check if coefficients are in cache coefficients = self._coefficients.get(chat_id) if coefficients is not None:
return coefficients

 If coefficients are not in cache, try to get them from database with cache update window
self._get_coefficients_from_db()

 Try to get coefficients from cache again
 If coefficients are not in cache, return None coefficients = self._coefficients.get(chat_id) return coefficients

def _get_predictions_from_db(self, from_date: datetime = None, to_date: datetime = None):
"""

Get all predictions from database

:return: """

table =
self._connection_db_metrics.metadata.tables[self.telegram_prediction_table] for row in self._connection_db_metrics.session.query(table).filter(
table.c.date >= from_date, table.c.date <= to_date
).all():
self._predictions[(row.date.replace(second=0, microsecond=0), row.chat_id, row.delta)] = {
column_name: value for column_name, value in row._asdict().items()
if column_name not in ['date', 'chat_id', 'delta']
}
```

```
        def _get_statistics_from_db(self, date: datetime): """
        Get statistics from the database for given date

        :param date: Date to get statistics for
        :return: """

        table =
        self._connection_db_metrics.metadata.tables[self.telegram_statistic_table] for row in
self._connection_db_metrics.session.query(table).filter(
        table.c.date_from <= date, table.c.date_to >= date,
        ).all():
        self._statistics[(row.date_from, row.date_to, row.chat_id, row.delta, row.metric)] = {
        column_name: value for column_name, value in row._asdict().items()
        if column_name not in ['chat_id', 'delta', 'metric', 'date_from',
        'date_to']
        }

        def _get_coefficients_from_db(self): """
        Get all coefficients from database

        :return: """

        table =
        self._connection_db_metrics.metadata.tables[self.telegram_coefficient_table] for row in
self._connection_db_metrics.session.query(table).all():
        self._coefficients[row.id] = {
        column_name: value for column_name, value in row._asdict().items()
        if column_name not in ['id']
        }

        def _get_nearest_date(self, dt: datetime) -> datetime: """
        Get the nearest date to predict on, rounded to prediction step minutes

        :param dt: Date to predict on
        :return: """

        minutes = dt.minute
        subtract_minutes = minutes % self._prediction_step

        return dt.replace(minute=minutes - subtract_minutes, second=0)
```

service_detector/container_detector/container_detector.py
Base import json
import logging

Import from project from pathlib import Path import sys
working_dir = Path(file).parents[2].resolve() sys.path.append(str(working_dir))

Internal
from service_detector.detector.detector import AnomalyDetector from service_detector.detector.detectors import (
TelegramMetricAnomalyDetector, TelegramRepostAnomalyDetector, TextNarrativeAnomalyDetector,
)

from service_detector.detector.utils import (TelegramPredictionCache, KibanaObjectCache,
KibanaLinkGenerator, NarrativeCache
)

External
from mnv_data_container_template.container_template import Container from mnv_queue_adapter.types import ACK, QueueMessage
from redis import Redis, ConnectionPool

class ContainerDetector(Container):
version = '1.0.0'
_service_name = 'detector' def init (self):
"""
Unifier container object """
Required config groups for container self._config_services_groups = ['elasticsearch', 'redis'] self._config_routing_groups = ['detector', 'versioning']

Setup params self._setup_pubsub = True self._setup_rabbitmq = False self._setup_versioning = True
self._versioning_target = 'pubsub' super(). init ()
def _setup_container(self):

"""
Additional setup for service_unifier container

:return: """

Setup routing keys
self.input_subscription = self._routing_values['detector_pubsub_subscription']
self.input_subscription = 'demo-pipeline-detector-sub' data-pipeline- detector-attached

Setup Redis connection self._setup_redis_connection()

Setup additional db connection self._setup_connection_db_metrics()

Setup anomaly detector self._setup_anomaly_detector()

def _setup_redis_connection(self): """
Setup Redis connection

:return: """

hostname = self._service_values['redis_hostname'] port = self._service_values['redis_port']
db = self._service_values['redis_database']

redis_connection_pool = ConnectionPool(host=hostname, port=port, db=db) self.connection_redis = Redis(connection_pool=redis_connection_pool)

def _setup_connection_db_metrics(self): """
Setup connection to metrics database

"""
import os
from mnv_data_package_database.connections.mysql import ConnectionDB

assert 'ART_MYSQL_HOSTNAME' in os.environ, 'ART_MYSQL_HOSTNAME is not set' assert 'ART_MYSQL_DATABASE' in os.environ, 'ART_MYSQL_DATABASE is not set' assert 'ART_MYSQL_LOGIN' in os.environ, 'ART_MYSQL_LOGIN is not set'
db_hostname = os.environ['ART_MYSQL_HOSTNAME'] if 'ART_MYSQL_PORT' in os.environ:
port = os.environ['ART_MYSQL_PORT']
db_hostname = f'{db_hostname}:{port}'

db_database = os.environ['ART_MYSQL_DATABASE'] db_login = os.environ['ART_MYSQL_LOGIN']
db_password = self._gcloud_secrets.get_secret('pipeline-data-password- sql-art')

self.connection_db_metrics = ConnectionDB(db_login, db_password, db_hostname, db_database)

```
        def _setup_anomaly_detector(self): """
        Setup anomaly detector


        :return: """

        narrative_cache = NarrativeCache(self.connection_db_metrics) telegram_prediction_cache =
        TelegramPredictionCache(self.connection_db_metrics) kibana_object_cache = KibanaObjectCache(
        elastic_api_key=self._gcloud_secrets.get_secret('pipeline-data-key- elasticsearch')
        )

        detectors = [

        TelegramMetricAnomalyDetector(telegram_prediction_cache=telegram_prediction_cach e),

        TelegramRepostAnomalyDetector(redis_connection=self.connection_redis),
TextNarrativeAnomalyDetector(narrative_cache=narrative_cache)
        ]

        kibana_link_generator =
        KibanaLinkGenerator(kibana_object_cache=kibana_object_cache)          self.anomaly_detector          =
AnomalyDetector(detectors=detectors,
        kibana_link_generator=kibana_link_generator)

        def _run_anomaly_detection(self, message: QueueMessage): """
        Run anomaly detection on article

        :param message: Input message
        :return: """

        message = json.loads(message.message, strict=False) alert = self.anomaly_detector.run(message)
        if not alert:
        logging.info(f"Message {message['metadata']['id']} has no alerts") return

        logging.info(f"Potential alert {alert.id} created") self._publish(json.dumps(alert.to_dict()), 'demo-pipeline-
anomalies')

        def run(self): """
        Start consumer """
        with self._pubsub_client.publisher() as self._queue_pub:
        with self._pubsub_client.subscriber(self.input_subscription) as (queue_sub, queue_ack):
```

while True:
message = queue_sub.get() self._run_anomaly_detection(message) queue_ack.put((ACK.ACK, message))

if name == ' main ': processor = ContainerDetector() processor.run()

alert_generator/utils/anomaly_retriever.py
Base
from datetime import datetime

Internal

External
from elasticsearch import Elasticsearch from elasticsearch.helpers import scan from tqdm import tqdm

class AnomalyRetriever: progress_bar = False
anomaly_index = 'anomalies_diploma_demo'

def init (self, elastic_url: str, elastic_api_key: str): """
Article retriever class

:param elastic_url: Elasticsearch URL
:param elastic_api_key: Elasticsearch API key """

self._elastic_url = elastic_url self._elastic_api_key = elastic_api_key

def get_anomalies(self, from_date: datetime, to_date: datetime) -> list[dict]: """
Retrieve anomalies from Elasticsearch

:param from_date: From date
:param to_date: To date
:return: List of anomalies """

es:

anomalies = []

```
with Elasticsearch(self._elastic_url, api_key=self._elastic_api_key) as

for anomaly in tqdm( scan(
es, index=self.anomaly_index, query={
'query': {
'range': {
'date': {
'gte': from_date, 'lte': to_date

}
}
}
}
),
desc='Retrieving anomalies', disable=not self.progress_bar
):
anomalies.append(anomaly['_source'])

return anomalies
```

ДОДАТОК Г

(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

"ІНТЕЛЕКТУАЛЬНА АВТОМАТИЗОВАНА СИСТЕМА ОПРАЦЮВАННЯ ДАНИХ ВІДЕО КАНАЛУ РОБОТА-ОХОРОНЦЯ"

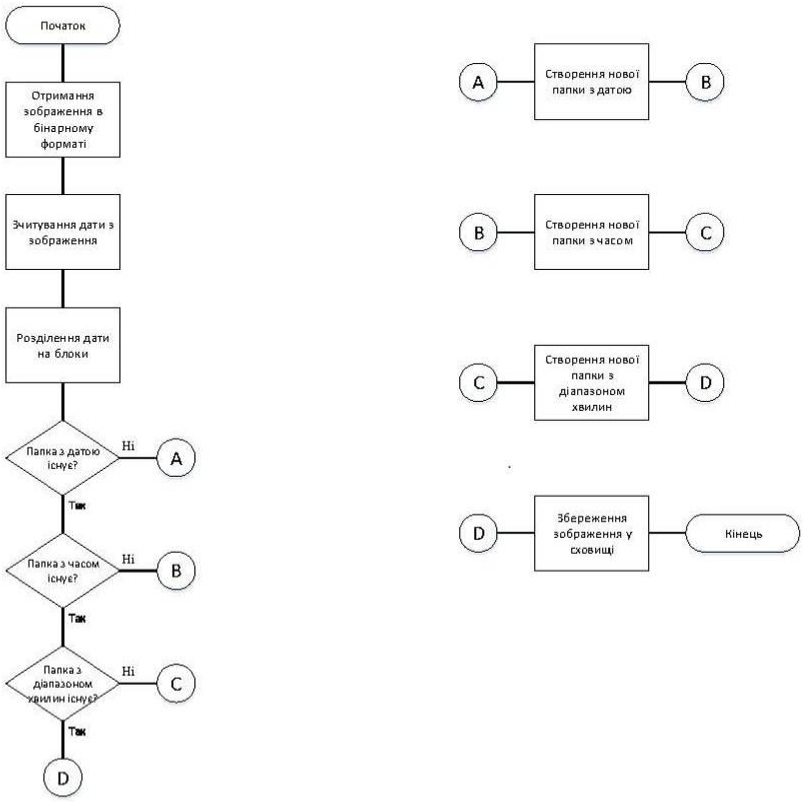
Студент групи 2АКІТР-23м

Артур НІКОНЧУК

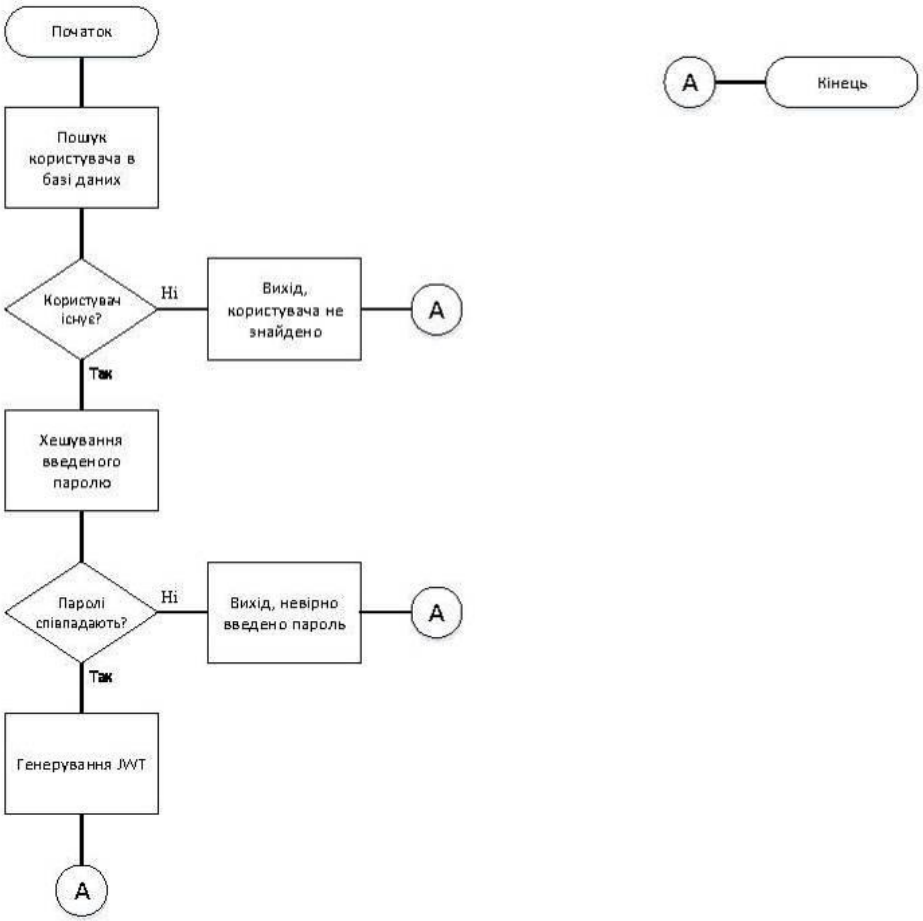
Керівник д.т.н., проф. каф. КСУ

В'ячеслав КОВТУН

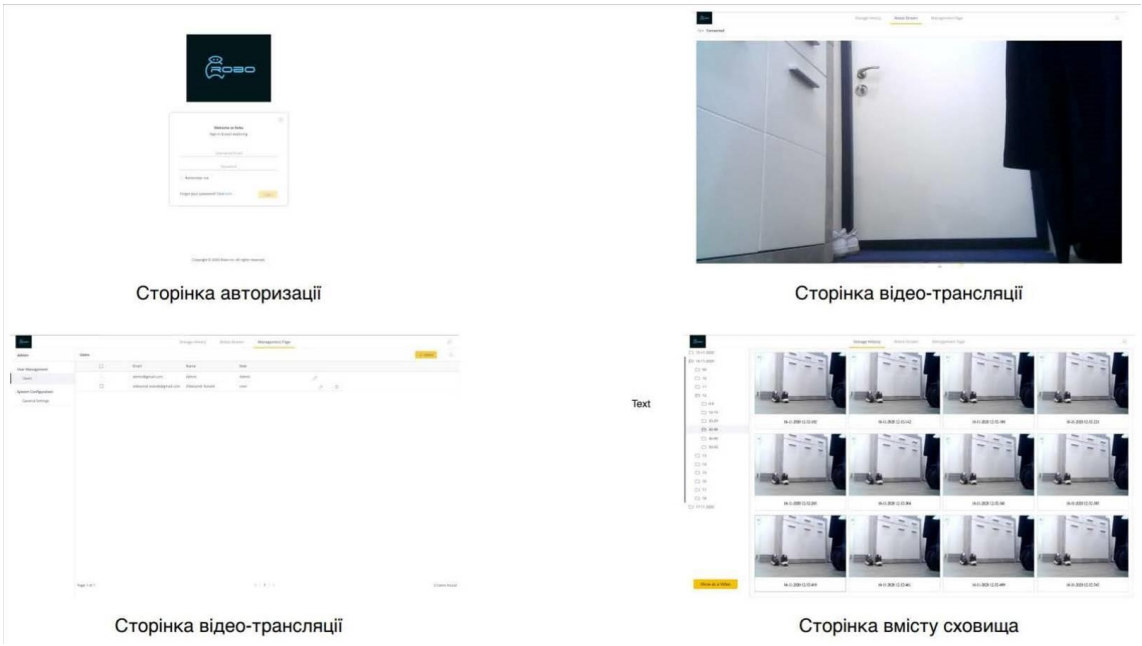
Архітектура системи



Алгоритм отримання та збереження зображень



Алгоритм авторизації користувачів

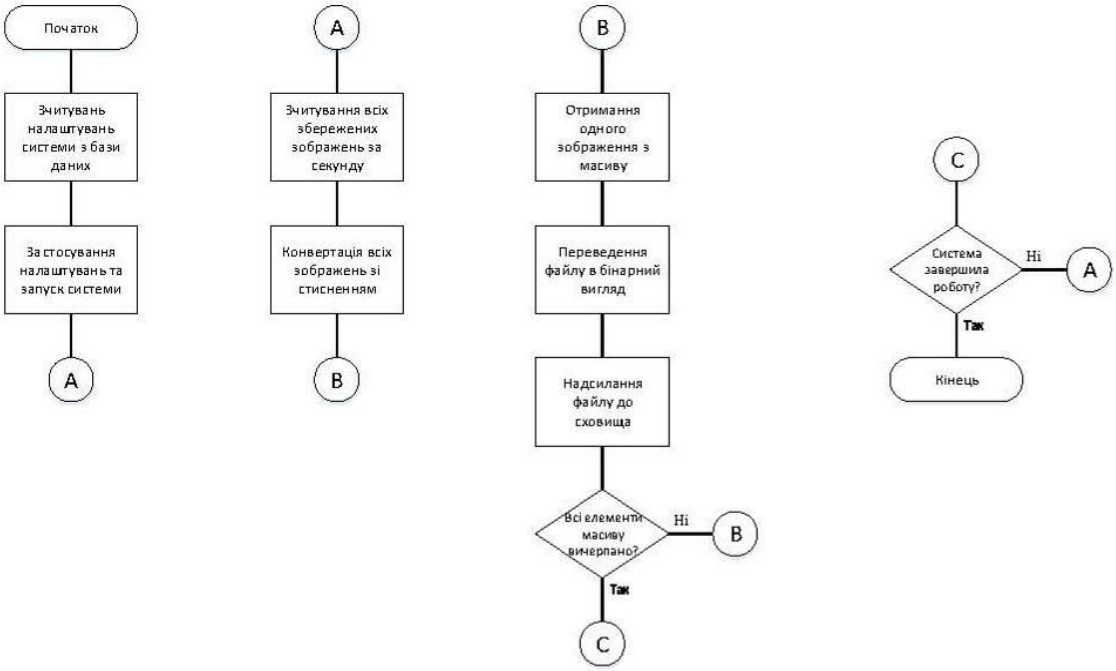


Інтерфейс користувача

№ кадру	Base64	Binary
1	41.5 kB	27.5 kB
2	41.7 kb	27.9 kB
3	41.7 kb	27.7 kB
4	42.0 kB	28.0 kB
5	41.5 kB	27.7 kB
6	41.8 kB	27.8 kB
7	41.6 kB	27.6 kB

№ кадру	JPG	WebP (100%)	WebP (80%)	WebP (50%)
1	108.3 kB	90.2 kB	44.5 kB	29.2 kB
2	108.2 kB	90.2 kb	44.9 kB	29.5 kB
3	106.4 kB	89.3 kb	43.1 kB	29.1 kB
4	106.9 kB	89.6 kB	43.0 kB	29.9 kB
5	107.1 kB	93.8 kB	45.2 kB	30.8 kB
6	107.6 kB	91.0 kB	44.1 kB	30.0 kB
7	108.9 kB	90.5 kB	43.6 kB	29. 6 kB

Аналіз ефективності стискання даних



Алгоритм оптимізації та надсилання зображень