КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки на тему:

РОЗРОБКА СИСТЕМИ ІНТЕГРАЦІЇ SECURITY-КАМЕР ТА ЗАСТОСУНКУ ДЛЯ ВІДЕОСПОСТЕРЕЖЕННЯ

Виконав студент 2-го курсу магістратури

Ілля СЕМЕНОВ _____
(підпис)

Науковий керівник: асистент кафедри математичної інформатики, кандидат технічних наук

Олексій ФЕДОРУС _____
(підпис)

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

Студент_____
(підпис)

Роботу розглянуто і допущено до захисту

На засіданні кафедри математичної інформатики

«___» _____20__р. Протокол № ____

Завідувач кафедри

Василь ТЕРЕЩЕНКО _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 43 сторінки 24 ілюстрації та 8 джерел.

Ключові слова: ВІДЕОСПОСТЕРЕЖЕННЯ, КІБЕРБЕЗПЕКА, ВЕБ, ВЕБ-ДОДАТОК, ВІДЕО, РОЗПІЗНАВАННЯ КАРТИНОК, ASP.NET CORE, ANGULAR, TYPESCRIPT, PYTHON.

Об’єктом розробки є веб-доданок, а також декілька серверів, що забезпечують його роботу.

Метою дипломної роботи є проведення дослідження проблем, які можуть трапитись у стані розробці веб-додатку або під час роботи з відео; реалізація розпізнавання відео та зображень; розробка та імплементація архітектури проекту; та реалізація проекту.

Методи розроблення: проектування веб-застосунка з використанням фреймворку Angular, проектування архітектури сервера, веб-дизайн, конвертація веб-застосунку у desktop-версію за допомогою Electron.

Інструменти розроблення: Для розробки даної програмної реалізації було обрано мови програмування C# .NET 7, Angular 14, Typescript, HTML, CSS та фреймворк Electron. Під час розробки у якості середовищ було обрано вільно поширювані VS Code для веб-частини та Visual Studio Community Edition для серверної частини. Мною було використано таку додаткову бібліотеку, як RxJs, яка облегшую розробку асинхронних проектів. Для полегшення створення нейронних мереж було обрано бібліотеку TensorFlow

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE – Integrated Design Environment, інтегроване середовище розробки;

API (application programming interface або інтерфейс програмування застосунків) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення.

стек

ORM (Object Relational Mapping) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних.

JWT (JSON Web Token) — це відкритий стандарт (RFC 7519) для створення токенов доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи.

OPC—сервер (OPClient Server) — сервер, також однойменна внутрішня бібліотека для менеджменту відео

MVC — це архітектурний шаблон програмного забезпечення, що використовується для розробки веб-додатків. MVC розшифровується як Model-View-Controller (Модель-Вид-Контролер)

HMACSHA — це алгоритм хеш-повідомлення з кодом аутентифікації на основі ключа (HMAC) з криптографічно стійкою функцією хешу SHA. HMACSHA використовується для генерації та перевірки інтегритету

повідомлення шляхом обчислення коду автентифікації на основі вказаного ключа та хешу повідомлення.

HEVC або H265 (High Efficiency Video Coding) — високоефективне кодування відеозображень

ВСТУП

Оцінка сучасного стану об’єкта розробки.

У сучасному світі комерційне відеоспостереження є однією з багатьох сфер, у якій дуже бракує універсальних та зручних рішень. Багато з проектів не підтримують сучасні стандарти безпеки, а ті, що підтримують, мають досить малу швидкодію та можуть бути дуже вимогливими до железа користувача

Актуальність роботи та підстави для її виконання. Постійне зростання кількості магазинів, торгівельно-розважальних центрів та такого іншого лише збільшує потребу ринку у сучасних та надійний рішеннях для відеоспостереження **Мета й завдання роботи**. Метою курсової роботи є розробка веб-додатку, інфраструктури, яка його підтримує та проведення дослідження проблем, які можуть трапитись у цей час. Для досягнення цієї мети поставлено такі завдання:

- Спроектувати сервер авторизації та аутентифікації.
- Розробити сервер, який би віддавав потрібні дані для веб-частини.
- Розробити веб-клієнт, який був би зручним для користувача та простим у використанні

Об'єкт, методи й засоби розроблення.

Перед розробкою ПЗ був проведений аналіз ринку на предмет схожих рішень та актуальність даної теми. Виявилось, що тема я дуже актуальною у наш час и все ще набирає популярність. Після цього було спроектовано структуру всього на бумазі для полегшення подальшої розробки.

Для розробки серверної частини було обрано C# ASP.NET Core 3. Мова є дуже ефективною в реалізації програм, особливо, використовуючи шаблон MVC. Її переваги перед іншими язиками, які використовують для бекенд розробки: кроссплатформенність, легкість у написанні і підтримки програм та швидкодія.

Для веб частини було обрано стек Angular + RxJs + Typescript. Це дуже популярний вібір у цій сфері, обумовлений простотою та зручністю розробки, гарною документацією, повним циклом розробки без використання додаткових засобів та швидкістю алгоритму пошуку змін у дереві DOM у angular'і.

Для розробки нейронних мереж було обрано мову Python та бібліотеку Pytorch для полегшення їх проектування і використання.

Під час розробки даного програмного засобу використовувалося два інтегрованих програмних середовища від Microsoft: Visual Studio (для серверної частини) та Visual Studio Code (для вебу та споміжний застосунків мовою Python). Для усього цього ПЗ розробник надає безкоштовну ліцензію для студентів вищих навчальних закладів та розробки поза межами крупних компаній.

Можливі сфери застосування. Розроблений програмний додаток може бути використаний для комерційного застосування з монетизацією через продаж ліцензій.

В загальному можна зробити висновок, що тема відеоспостереження є досить актуальною у наш час та розроблений веб-додаток буде зручним для користувача.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ПРОЕКТІВ

1.1 Surveillance station

Surveillance station - проект компанії synology, який реалізує систему для управління відеоспостереженням

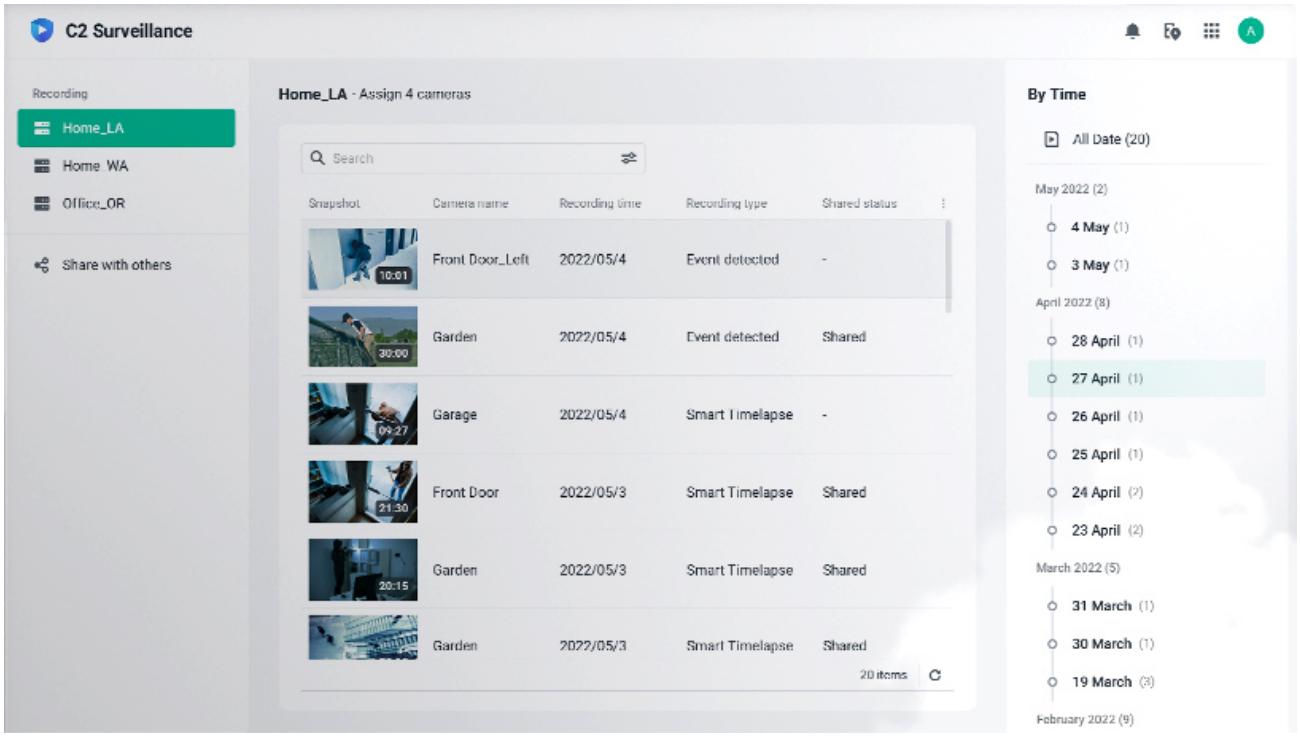


Рисунок 1.1 – приклад інтерфейсу Surveillance station

Переваги:

- Розробляється великою командую професіоналів, тому є досить стабільною
- Дуже багатофункціональна

Недоліки:

- Коштує дуже багато

  Розглянувши та проаналізувавши ринок, було вирішено створити свій застосунок для системи відеоспостереження

1.2 NetCam Studio

Netcam Studio Client - проект однойменної компанії netcam, який дозволяє юзеру керувати камерами відеоспостереженням

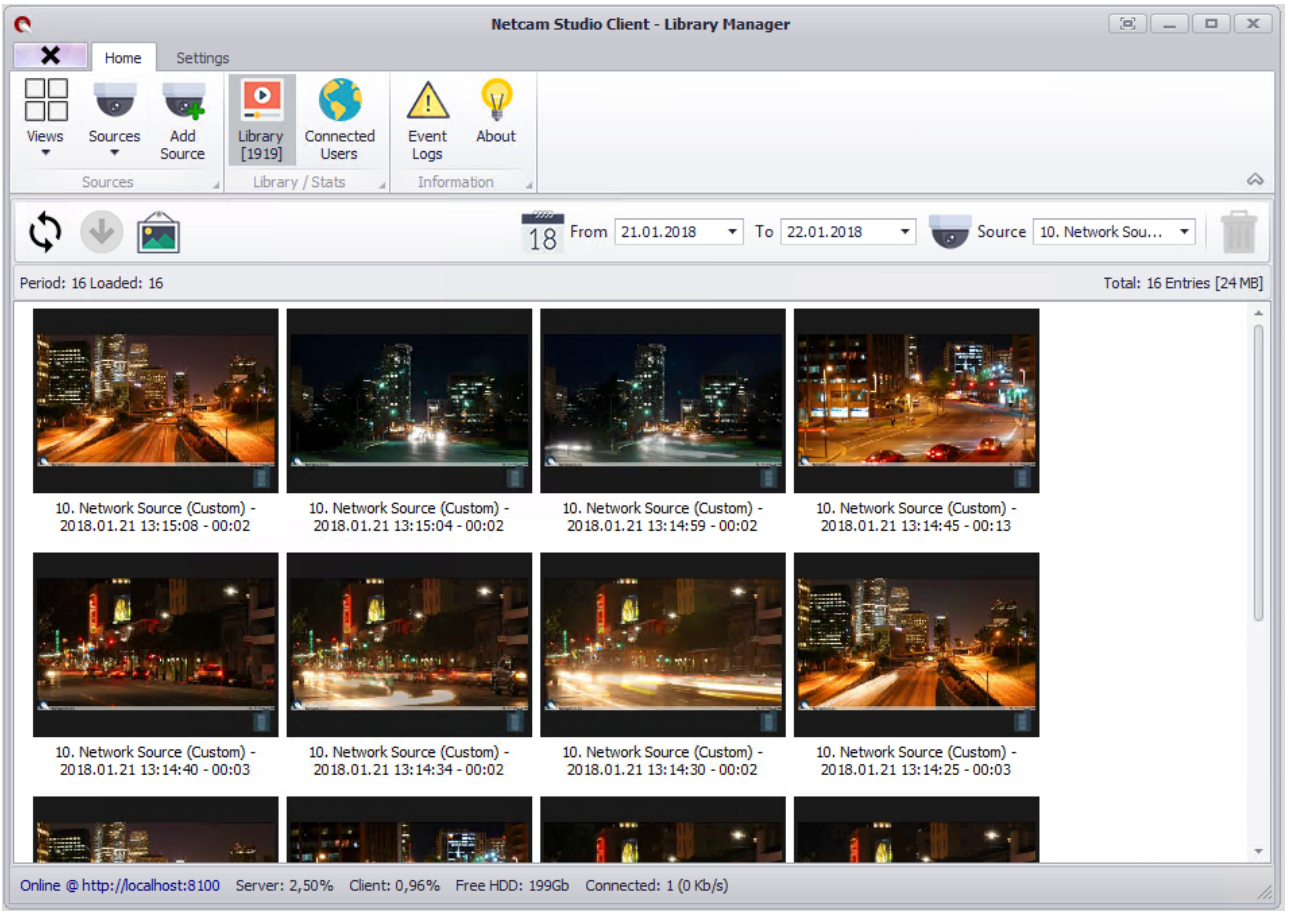


Рисунок 1.2 – приклад інтерфейсу NetCam studio

Цей продукт інтеграції CCTV-камер пропонує ряд переваг, які роблять його зручним і потужним інструментом для відеоспостереження. Однак, він також має свої недоліки, які варто врахувати при його використанні. Огляд основних переваг та недоліків цього продукту наведено нижче:

Переваги:

- Простий інтерфейс: Продукт має інтуїтивно зрозумілий і легкий у використанні інтерфейс, що дозволяє оперативно орієнтуватись у системі і здійснювати потрібні налаштування.
- Широкий спектр підтримуваних камер: Він підтримує інтеграцію з багатьма моделями і типами CCTV-камер, що дозволяє забезпечити сумісність з різними виробниками та моделями камер.

Недоліки:

- Обмежені можливості масштабування: При великому обсязі системи або необхідності інтеграції з великою кількістю камер, цей продукт може зазнавати обмежень у плані масштабування. Це може створювати проблеми при розширенні системи або додаванні нових камер до існуючої інфраструктури.
- Відсутність підтримки деяких додаткових функцій: Деякі додаткові функції, які можуть бути важливими для певних сценаріїв відеоспостереження, можуть бути відсутні у цьому продукті. Наприклад, підтримка аналітики поведінки, розпізнавання номерних знаків або інтеграція з системами контролю доступу можуть бути обмеженими або відсутніми.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СТРУКТУРИ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Розробка структури серверу

Було розроблено повну структуру застосунку, яку можна побачити на рисунку 2.1

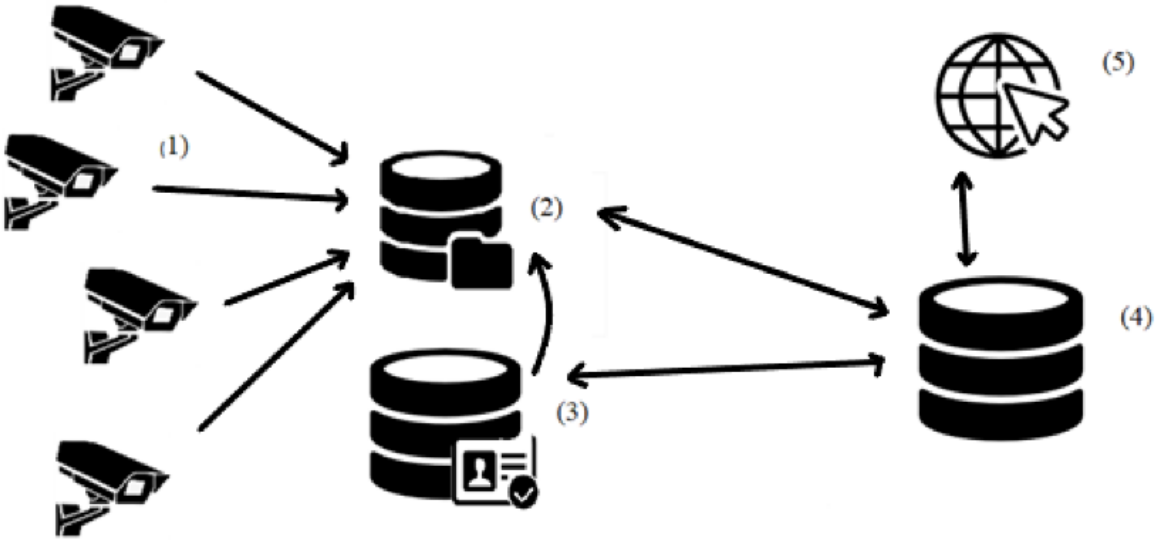


Рисунок 2.1 – схематична структура застосунку

Тут можна побачити такі сутності:

1) Камери відеоспостереження, можуть мати різних виробників, різні налаштування, різні кодеки
2) Глобальний (для системи, фактично один або декілька для кожної компанії та мережі) сервер для зберігання та обміну інформації з усіма клієнтами
3) Сервер автентифікації
4) Локальний для кожного юзера сервер. Може бути проксі-сервером, може бути electron-сервером
5) Веб або десктоп застосунок

Розглянемо трохи детальніше:

Було реалізовано мульти-серверну структуру. Тобто для кожного користувача є можливість додавати різні глобальні сервери, кожен з яких поєднує багато камер. Користувач заходить у застосунок, який звертається до локального сервера. Якщо це десктоп-версія, то вона буде на комп'ютері користувача, але може й до локального сервера іншого користувача у цьому ж домені, якщо обрано веб-версію. Локальний (він же проксі) сервер намагається авторизуватись у (3), якщо спроба вдала – отримує збереженні конфігурації глобальних серверів та налаштування для під'єднання до них. Після цього локальний сервер під'єднується до кожного з глобальних серверів, кожен з яких може керувати своїм пулом камер. Якщо користувач переходить у вкладку з стрімінгом, то додатково застосунок перевіряє, чи він десктопний. Якщо так - намагається під'єднатися до node-модуля, який дає можливість більш швидкого стрімінгу у форматах, що не підтримуються браузером через прямий доступ до відеопам'яті.

Локальний сервер – це є обгортка над API глобального сервера, який не має HTTP інтерфейсу. Локальний сервер може отримати стан будь-якої змінної з глобального серверу, або викликати будь-яку команду:

```
Ссылок: 1
public void Connect() => _connectService.Connect(OPClient);
Ссылок: 0
public Dictionary<string, object> Execute(Dictionary<string, object> request) => _proxyService.Execute(OPClient, request);
Ссылок: 0
public object GetProperty(string propertyName) => _proxyService.GetProperty(OPClient, propertyName);
Ссылок: 0
public object Method(string methodName, object[] methodArgs) => _proxyService.CallMethod(OPClient, methodName, methodArgs);

Ссылок: 4
private OPClient OPClient => _opClientService.Get(Context.ConnectionId);
Ссылок: 1
private void AddOPClient() => _opClientService.Add(Context.ConnectionId);
Ссылок: 1
private void RemoveOPClient() => _opClientService.Remove(Context.ConnectionId);
Ссылок: 1
private void AddHandlers() => _opClientEventService.Add(Context.ConnectionId);
Ссылок: 1
private void RemoveHandlers() => _opClientEventService.Remove(Context.ConnectionId);
```

Рисунок 2.2 – код для звернення для глобального серверу

Тут OPClient – це суперклас з однойменної внутрішньої бібліотеки для комунікації з глобальним сервером за TCP\IP протоколом. Тобто можна побачити три функції –

GetProperty/ Method – отримати значення з бібліотеки будь-якої змінної– викликати будь-який стандартний метод (наприклад, увімкнути камеру)

CallExecute – викликати будь-який не-стандартний метод з будь-якими параметрами. Бібліотека підтримує свою скриптову мову, тому можливо, наприклад, вимкнути усі камери, у назві яких міститься слово “Indoor” на 8 годин і почати записувати одразу після того, як вони увімкнуться. Має схожу структуру з командами на язику bash.

Add / RemoveHandler – управління підписками на івенти глобального сервера

Для зв’язку клієнта та локального сервера було обрано бібліотеку SignalR від Microsoft, яка є простою у використанні та являє собою обгортку над усими можливими методами зв’язку клієнта та сервера у реальному часі (наприклад, веб-сокети або лонгполлінг). Для більше зручного користування кастомними командами було реалізовано патерн Command:

```
export class GetCamerasCommand extends ApiCommand {
  request = [
    {
      action: 'executeinscript',
      command: 'get_cameras_info',
      enabled: 1,
      properties: 'camera_id;camera_name;ip_address;'
    }
  ] as const;

  mapResponse(response: GetCamerasPropertyBag): CameraDto[] {
    const cameras: CameraDto[] = [];
    const cameraCount = Object.keys(response).filter(key => key.startsWith('camera_id')).length;

    for (let index = 0; index < cameraCount; index++) {
      const id = response[`camera_id${index}`];
      const ipAddress = response[`ip_address${index}`]?.split(':')[0];
      const name = response[`camera_name${index}`];
      const camera = new CameraDto({ id, ipAddress, name });
      cameras.push(camera);
    }

    return cameras;
  }
}
```

Рис 2.3 – Реалізація команди для отримування бажаної інформації про вімкнені камери

Для пришвидшення роботи непідтримуємих браузером форматів було реалізовано т.н. “аддон”. Це обгортка на c++ бібліотеку за допомогою утіліти gyp, яка робить з неї node-js бібліотеку (Native Node Module).

2.2 Авторизація

Систему авторизації реалізовано на основі специфікації JSON Web Token (JWT) RFC7519. Основна перевага даного підходу полягає у тому, що токен створюється один раз на сервері, а потім зберігається на клієнті, що дозволяє швидко аутентифікувати користувача. Більшість полів, які можуть знадобитися у роботі, чітко описано у специфікації, але можна також створювати нові поля.

У NET Core вже реалізовано більшість функціоналу для обробки, парсингу та створення токенів. JSON Web Token (JWT) [7] – це структура, яка отримана з простого об’єкту у форматі JSON з деякими трансформаціями. Наприклад, візьмемо такий об’єкт:

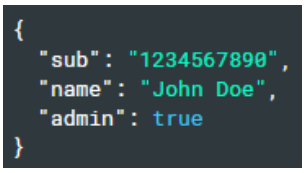


Рисунок 2.4 – приклад початкового JSON об’єкту для створення токену.

Кінцевий токен має формат xxxx.yyyy.zzzz, де xxxx – заголовок, yyyy – вміст, та zzzz – сигнатура (підтвердження цілісності токену. Заголовок можна отримати через застосування алгоритму Base64Url до наступного JSON об'єкту:

Рисунок 2.5 – приклад JSON для створення заголовку токена

Сигнатуру можна отримати за формулою :

HMACSHA256(base64UrlEncode(xxxx) + "." +base64UrlEncode(yyyy), secret)

де secret - ключ, який зберігається на сервері і за допомогою якого відбувається кодування. Отже, не маючи цього ключа, користувач не може підробити токен [8].

RefreshTokens – таблиця для токенів поновлення Json Web Token`ів:

Через деякий час виявилось, що потрібно поновлювати JWT доволі часто для того, щоб інформацію у них можна було змінювати на сервері. При цьому в токенах, які зберігаються у користувачів, вона теж має оновлюватись

Ілюстрацію проблеми можна побачити на рис 2.5:

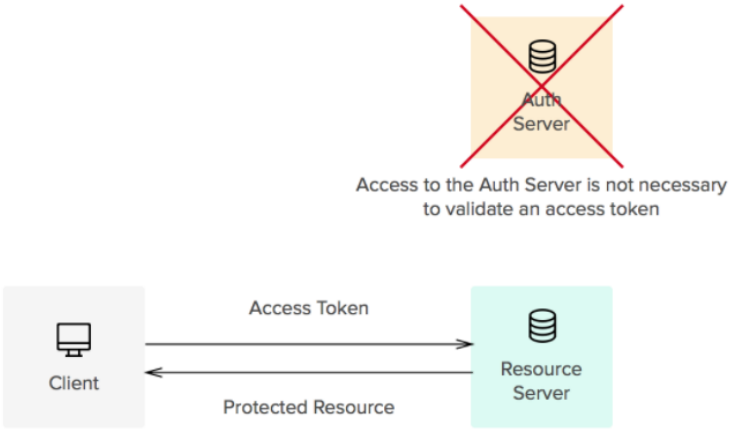


Рисунок 2.6 – проблема використання JWT

Діграмма вирішення проблеми з використанням Refresh Token`ів (рис. 2.6)

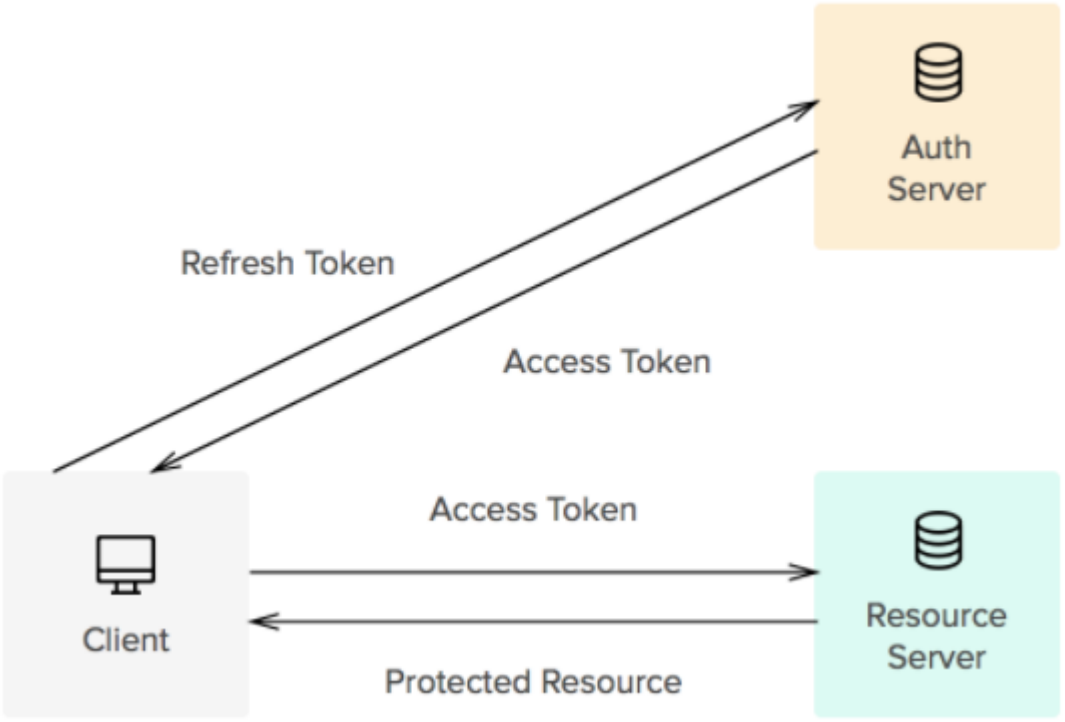


Рисунок 2.7 Діаграма використання Refresh Token`ів

Токени оновлення містять інформацію, яка необхідна для отримання нового JWT (обраного нами реалізацією токену доступу). Іншими словами, кожного разу, коли потрібен маркер доступу для отримання певного ресурсу, клієнт може використовувати маркер оновлення, щоб отримати новий маркер доступу від серверу автентифікації. Розповсюдженим використанням є отримання нових маркерів доступу після закінчення терміну дії старих або отримання доступу до нового ресурсу вперше. Токени оновлення можуть також закінчитись, але зазвичай вони мають досить довгий термін дії. Для токенів оновлення застосовуються суворі вимоги щодо зберігання, щоб гарантувати їх надійність та запобігти просоченню. Сервер авторизації може також внести їх до чорного списку.

2.3 Проблема кодеків

Під час розробки була виявлена серйозна проблема, пов'язана зі старими камерами, які постачають відео в застарілих форматах. Ці формати можуть бути передані до браузера в стислому вигляді та без проблем переглядаються, але нові камери використовують новий алгоритм кодування відео H265. Для кращого розуміння цього формату потрібно зазначити декілька слів про нього.

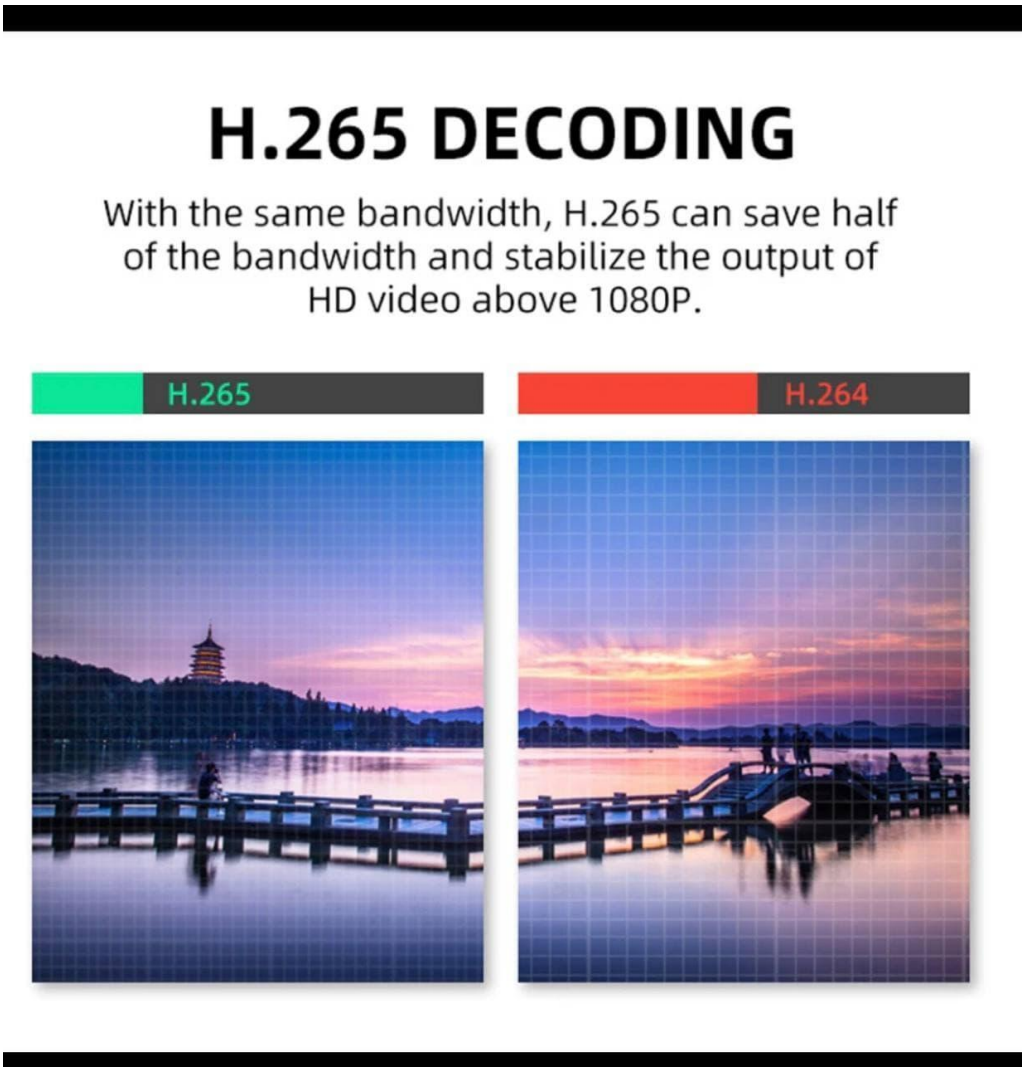


Рис 2.8 Схематичне зображення порівняння H264 та H265

H.265 або HEVC— формат відеостиснення із застосуванням більш ефективних алгоритмів порівняно з H.264/MPEG-4 AVC. Рекомендація МККТТ H.265, а також стандарт ISO/IEC 23008-2 MPEG-H Частина 2 — спільна розробка експертної групи з відеокодування МККТТ (ITU-T Video Coding Experts Group —

VCEG) та експертної групи з рухомих зображень MPEG . Рекомендація стандарту розроблена у зв'язку з зростаючою потребою у більш високому ступені стиснення рухомих зображень для різних програм, таких як потокова передача в Інтернеті, передача даних, відеоконференц-зв'язок, цифрові пристрої та телевізійне мовлення. Підтримуються формати кадру до 8K (UHDTV) з роздільною здатністю 8192×4320 пікселів.

Цей алгоритм є стандартним для усіх найновіших камер відеоспостереження, але бразуер не підтримує можливість декодування його з використанням відеокарти клієнта (наприклад, як у попереднього H264). Тому було розглянуто такі можливості обходу проблеми:

1) Декодування (або перекодування) відео на глобальному сервері

2) Декодування відео у бразуері користувача без використання апаратного прискорення

3) Відмова від підтримки H265 до того моменту, як його не почнуть підтримувати браузери

4) Декодування (або перекодування) відео на локальному сервері

Розглянемо плюси та мінуси кожного підходу.

Перший варіант: Хоча перекодування відео буде відбуватися лише раз, але у сервера, який обслуговує деяке підприємництво, може бути, наприклад, сотня камер, і якщо багато охоронців одночасно дивляться на усі камері (наприклад, 20 охоронців, кожен спостерігає за 5 камерами), то перекодування одночасно сотні відео буде неможливим на більшості існуючих машин.

Другий варіант: без підтримки апаратного прискорення, перекодування навіть 2-3 відео одночасно у реальному часі у браузері не можливо за технічних обмежень

Третій варіант: підтримка цього кодеку у браузері може не настати, а продукт без цієї фічі не буде конкурентоспроможним на ринку.

Четвертий варіант: хоча й є мінус того, що одне й те саме відео може перекодовуватись багато разів, але це є єдиний можливий варіант з урахуванням існуючих апаратних обмежень.

Тому з оглядом на причини було обрано четвертий варіант.

Для передачі даних з локального сервера у браузер було розглянуто такі варіанти:

1) Передача дескриптора вікна локальному серверу, який буде рендерити відео у цьому вікні. Проблема – таке використання заборонено в усіх браузерах, окрім Internet Explorer.
2) Передача даних за допомогою розширень браузеру (Browser Extensions) та використання shared process memory. Проблема – штучне обмеження ширини каналу в 1МБ.
3) Підняття локального серверу у форматі апі, який буде віддавати дані у браузер.
4) Обрнення застосунку в десктопний варіант за допомогою Electron

Як можна було побачити, перші два варіанта неможливі з урахуванням обмежень, третій вирішує проблему в основному, але залишається проблема недостатньої швидкості. Такий варіант буде робочим для одного відео у FullHD (розмір декодованого фрейму приблизно 10МБ), але зовсім не підійде для стрімов в форматі мозаіки (тобто, 2-16 стрімів в форматі до FullHD) або 4К стримінгу.

Тому було обрано поєднання третього та четвертого варіанту, і використання четвертого, якщо він доступен, а якщо ні – то повернення до третього.

2.4 Стрімінг за допомогою Native Node Modules

Для вирішення проблеми декодування формату H.265 в браузері, було проаналізовано можливість використання electron modules та модулів на платформі Node.js. Перевагою використання цих модулів є можливість використання нативних бібліотек оперуючи з рівнем операційної системи, що забезпечує більш високу ефективність обробки відеоданих. Однак, для веб-додатків, які мають широкий охоплювати шар користувачів, використання нативних модулів може бути обмежене через необхідність встановлення спеціального ПЗ на більшості пристроїв користувачів. У цьому випадку, модулі на платформі Node.js можуть бути більш практичним варіантом, оскільки вони працюють в звичайних браузерах і не вимагають встановлення додаткового ПЗ на більшості пристроїв користувачів.

Трохи детальніше про структуру цього модуля (див. рис. 2.9). OpClientAddon - це модуль на рівні native Node.js, який експортує інтерфейси OPClient та OPClientStream, розроблені для роботи з глобальним OPClient – сервером (далі – OPC-сервер).

OPClientAddon містить два класи OPClient та OPClientStream, які служать для з'єднання з OPC-сервером та створення відтворювального та живого потоків відео. Для підключення до OPC-сервера OPClient містить метод Connect, який приймає адресу сервера, логін, пароль та домен, щоб забезпечити відповідну автентифікацію. Ця автентифікація забезпечується і повністю покладається на схеми Windows автентифікації

OPClientStream містить набір методів, що дозволяють керувати відтворенням відео. Це включає зупинку, паузу, відтворення, швидкість відтворення, перехід до певного часового маркера та отримання параметрів зображення відео, таких як розмір та формат.

Також OPClientStream містить методи OnFrameConsumed та FrameStep, що служать для взаємодії з відеофреймами. OnFrameConsumed викликається при кожному споживанні відеофрейма, а FrameStep дозволяє крокувати вперед або назад на задану кількість фреймів.

Даний модуль є необхідним для забезпечення підтримки відтворення відео в форматі H.265, ефективного використання ресурсів системи та забезпечення користувачам безперебійний досвід перегляду відео.

Цей модуль представляє собою нативний модуль для Node.js і, на жаль, не має вбудованої підтримки для типізації в середовищі Electron. Це означає, що для використання всіх переваг TypeScript з цим модулем, потрібно вручну визначити його типи. (рис 2.9)

Для забезпечення належної типізації можна створити відповідні інтерфейси для класів і функцій, що надаються модулем. У випадку цього модуля можна створити інтерфейси *OpClientAddon*, *OpClientCredentials*, *OPClient*, *OPClientStream*, а також визначити типи *OnStreamEventHandler* і *ConsumeFrame*. (рис 2.10)

```
You, 4 weeks ago | 1 author (You)
export const OpClientAddon = (window as any).require('@electron/remote').require('/../../../unicorn.node') as OpClientAddon;


You, 4 weeks ago | 1 author (You)
export interface OpClientAddon {
    OPClient: { new(): OPClient }
    OPClientStream: { new(): OPClientStream }
}
    You, 4 weeks ago • Add electron …
You, 4 weeks ago | 1 author (You)
export interface OpClientCredentials {
    tcpAdress: string;
    login: string;
    password: string;
    domain: string;
}

export type OnStreamEventHandler = (eventId: number) => void;
export type ConsumeFrame = (uint8array: ArrayBuffer, frameSize: number, timestamp: number) => void;
```

Рис 2.9 – Додаткова типізація для Native Node Module

```
You, 4 weeks ago | 1 author (You)
export interface OPClient {
    Connect: (tcpAddress: string, login: string, password: string, domain: string) => void;
    CreateLive: (
        cameraId: string,
        onStreamEventHandler: OnStreamEventHandler,
        consumeFrame: ConsumeFrame
    ) => OPClientStream;
    CreatePlayback: (
        cameraId: string,
        startOADate: number,
        endOADate: number,
        onStreamEventHandler: OnStreamEventHandler,
        consumeFrame: ConsumeFrame
    ) => OPClientStream;
}

You, 4 weeks ago | 1 author (You)
export interface OPClientStream {
    FrameStep: (ms: number) => string;
    GetPictureParameters: () => {
        width: number,
        height: number,
        subtype: string
    };
    Pause: () => string;
    Play: () => string;
    Seek: (oaDate: number) => string;
    Stop: () => string;
    SetPlayRate: (playrate: number) => string;
    OnFrameConsumed: () => string;
}
```

Рис 2.10 – Основна типізація для Native Node Module

Ці типи дозволять використовувати модуль з TypeScript і отримати всі переваги типізації, такі як перевірка типів під час компіляції і підказки IDE. Що надає можливість впевнено взаємодіяти з методами модуля, передавати параметри правильного типу і отримувати очікувані результати.

Хоча це може бути трохи незручно, оскільки потребує додаткової ручної роботи для типізації модуля, але це необхідний крок, щоб гарантувати коректну роботу проекту з TypeScript та Electron.

РОЗДІЛ 3 РОЗПІЗНАВАННЯ ЛЮДЕЙ, ЯКІ НЕ НОСЯТЬ МАСКУ ТА ВИЯВЛЕННЯ АКТИВНОСТІ

3.1 Збір даних

Набір даних, використаний у цьому дослідженні, був зібраний з різних форматів зображень, таких як JPEG, PNG та інших. Отримано різні набори даних і зображення з відкритих джерел, включаючи набір даних Kaggle «Face Mask Detection». Тому вдалось отримати різні види зображень з різницею в розмірі та роздільній здатності, з яких деякі нагадують реальні фотографії, а інші були зі штучно накладеними масками. У наборі даних Face Mask Detection зібрані основні зображення облич та нанесені орієнтири обличчя, щоб виявити характеристики обличчя людини на зображенні. Основні орієнтири обличчя включають очі, ніс, підборіддя, губи та брови. Загалом отримали датасет розміром приблизно 4000 зображень.

3.2 Попередня обробка даних

Точність моделі залежить від якості набору даних. Початкове очищення даних виконується для усунення дефектних зображень, виявлених у наборі даних. Розмір зображень було змінено до фіксованого розміру 96x96, що допомагає зменшити навантаження на модель під час тренування та забезпечити оптимальні результати. Далі, набір даних було розділено на два класи «with_mask» і «without_mask». Потім масив зображень перетворено на масив NumPy для більш швидкого обчислення. Поряд з цим також використовується функція preprocess_input з MobileNetV2. Після цього було використано техніку збільшення даних. Функція ImageDataGenerator використовується з відповідними значеннями обертання, масштабування, горизонтального або вертикального перевороту для створення численних версій одного і того ж зображення. Навчальні вибірки були збільшені, щоб уникнути перенавчання. Це спрямовано на покращення надійності навченої моделі. Потім весь набір даних було поділено на навчальні та тестові дані у співвідношенні 80:20 шляхом випадкового вибору зображень із набору даних. Параметр stratify використовується для збереження тієї ж частки даних, що й у вихідному наборі даних, як у наборах даних для навчання, так і для тестування.

3.3 Створення моделі

Архітектура моделі, прийнята для дослідження, описана в Таблиці 3.1.

Шар	Вихідна розмірність	Структура
Conv2D	96 x 96	f = 16, dim f = 3 x 3, Крок = 1
MaxPooling	48 x 48	f = 16, dim f = 2 x 2, Крок = 1
Conv2D	48 x 48	f = 32, dim f = 3 x 3, Крок = 1
MaxPooling	24 x 24	f = 32, dim f = 2 x 2, Крок = 1
Conv2D	24 x 24	f = 64, dim f = 3 x 3, Крок = 1
MaxPooling	12 x 12	f = 64, dim f = 2 x 2, Крок = 1
Conv2D	12 x 12	f = 128,dim f = 3 x 3,Крок = 1
MaxPooling	6 x 6	f = 128,dim f= 2 x 2, Крок = 1
Conv2D	6 x 6	f = 256,dim f = 3 x 3,Крок = 1
MaxPooling	3 x 3	f = 256,dim f = 2 x 2,Крок = 1
Classification Layer	–	Повнозв’язний, Softmax

Таблиця 3.1 – Структура згорткової нейронної мережі

Тут f – кількість фільтрів, dim f - розмірність фільтрів. Основними компонентами архітектури є двовимірні згорткові шари (conv2D), рівень об’єднання, функції активації та повнозв’язані шари. Запропонована модель містить загалом 5 шарів Conv2D із заповненням «однаковим» і кроком 1. На кожному шарі conv2D карта характеристик двовимірних вхідних даних витягується шляхом «ковзного введення» через фільтр або ядро та виконує наступну операцію:

$$C(Z) = (P \times Q)(x) = (\int_{-\infty}^{\infty} P(Z) \times Q(Z - x) dZ),$$

де P – матриця вхідного зображення,

Q – згорткове ядро,

C – вихідні дані.

Об’єднання шарів зменшує розмір карти об’єктів. Таким чином, кількість параметрів, які можна навчати, зменшується, що призводить до швидших обчислень без втрати суттєвих ознак. Можна виконувати два основних типи операцій об’єднання: MaxPooling та AveragePooling. MaxPooling означає створення найбільш важливого значення в конкретному місці, де знаходиться ядро. З іншого боку, AveragePooling обчислює середнє значення кожного значення в цьому регіоні.

Функції активації – це вузли, які розміщуються на кінці або серед нейронних мереж (шарів). Вони вирішують, спрацьовує нейрон чи ні. Вибір функції активації на прихованих шарах, а також на вихідному шарі важливий, оскільки він контролює якість навчання моделі. Функція активації ReLU в основному використовується для прихованих шарів; тоді як Softmax використовується для вихідного шару та обчислює розподіл ймовірності з вектора реальних чисел. Останній є кращим вибором для багатокласових завдань класифікації. Що стосується ReLU, то він пропонує кращу продуктивність і широке вивчення глибини в порівнянні з функцією сигмоїди та гіперболічного тангенсу.

Після того, як усі згорткові шари були реалізовані, застосовуються шари FC. Ці шари допомагають класифікувати зображення як за мультикласовими, так і за бінарними категоріями. У цих шарах функція активації softmax є гарним вибором для отримання імовірнісних результатів.

TensorFlow і Keras є основними підходами для реалізації запропонованої моделі. У цьому дослідженні 80% набору даних вноситься в навчальний набір, а решта — на набір для тестування. Вхідне зображення попередньо обробляється та доповнюється за допомогою кроків, описаних вище. Всього існує 5 шарів Conv2D з функціями активації ReLu з фільтром 3x3 і 5 шарами MaxPooling з розміром фільтра 2 x 2. Flatten і Dense використовуються як повністю підключені шари. Вихідний рівень використовує softmax як функцію активації. Це генерує 2 818 658 параметрів,

які можна навчати в цій згортковій нейронній мережі. У Таблицях 3.2 та 3.3 показано процес навчання, який реалізується за допомогою SGD і Adam. Він порівнює їх за різними параметрами якості.

Epochs	loss	accuracy	val_loss	val_accuracy
10	0.6913	0.5000	0.6888	0.5100
20	0.6902	0.5025	0.6872	0.5100
30	0.6890	0.5075	0.6857	0.5100
40	0.6882	0.5075	0.6841	0.5250
50	0.6873	0.5138	0.6826	0.5300
60	0.6858	0.5400	0.6808	0.5400
70	0.6851	0.5562	0.6789	0.5600
80	0.6836	0.5738	0.6769	0.5800
90	0.6823	0.5875	0.6747	0.6100
100	0.6807	0.5962	0.6722	0.6250

Таблиця 3.2 – Результати тренування моделі звикористанням оптимізатора SGD

Epochs	loss	accuracy	val_loss	val_accuracy
10	0.2938	0.8612	0.2520	0.9050
20	0.2172	0.9125	0.2506	0.9100
30	0.1524	0.9375	0.1783	0.9300
40	0.1688	0.9350	0.2488	0.9300
50	0.1223	0.9600	0.2259	0.9450
60	0.1204	0.9575	0.3382	0.9150
70	0.0760	0.9800	0.2224	0.9400
80	0.0478	0.9825	0.2671	0.9450
90	0.0541	0.9825	0.2407	0.9500
100	0.0288	0.9937	0.2102	0.9500

Таблиця 3.3 – Результати тренування моделі з використанням оптимізатора ADAM

З таблиць видно, що Adam Optimizer працює краще, ніж оптимізатор SGD, оскільки забезпечує кращі результати зі збільшенням епох. На Рисунках 3.1 і 3.2 показано, що оптимізатор Adam забезпечує кращу продуктивність, ніж оптимізатор SGD. Гіперпараметри, використані в цій моделі, описані нижче в Таблиці 3.4. Binary_crossentropy використовується для розрахунку втрат від класифікації для моделі. Для задачі класифікації це дає значення від 0 до 1 (значення ймовірності).

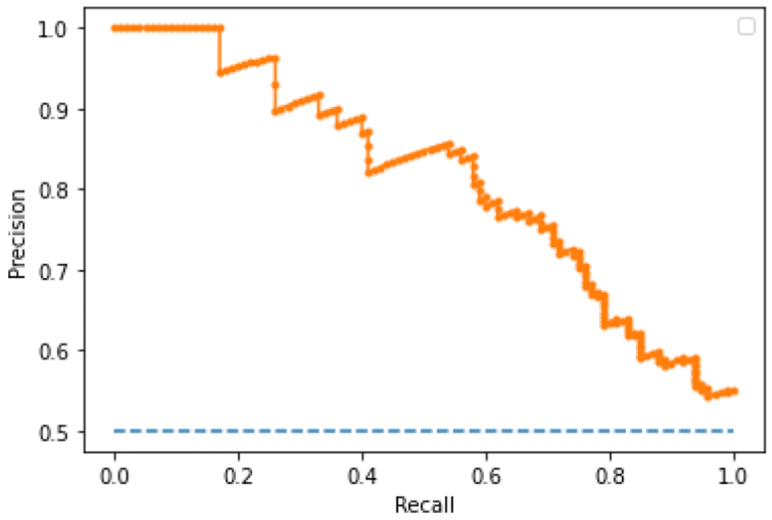


Рисунок 3.1 - Показники точності для оптимізатора SGD

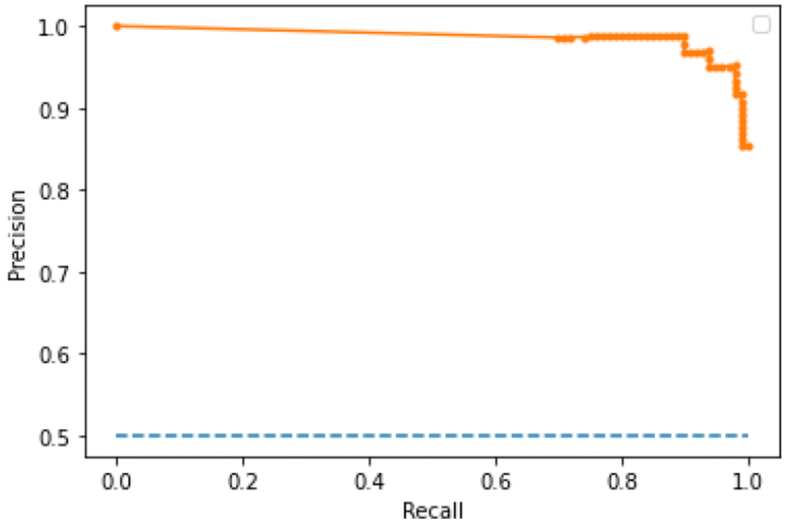


Рисунок 3.2 – Показники точності для оптимізатора ADAM

Параметр	Значення
Learning Rate	0.0005
Epochs	100
Batch Size	32
Optimizer	Adam
Loss Function	binary_crossentropy

Таблиця 3.4 – Підбір гіперпараметрів моделі

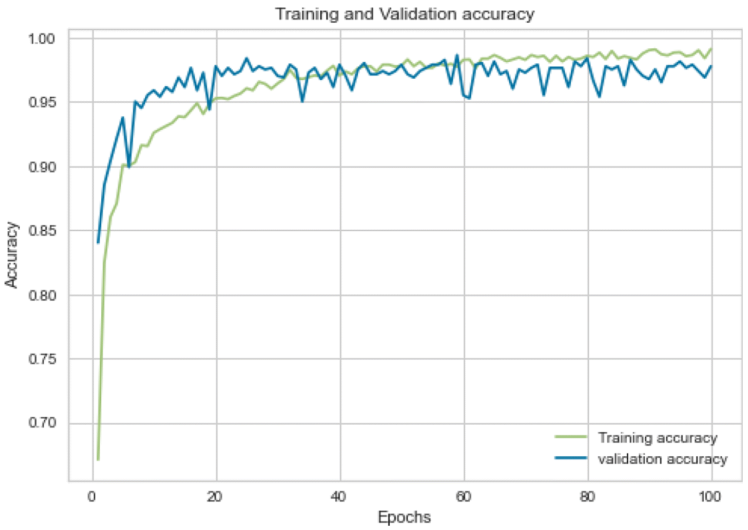
3.4 Результати та тестування моделі

Після тестування моделі на тестовій вибірці було отримано наступні показники класифікації.

	Precision	Recall	F1-score	Support
dataset/with_mask	0.98	0.97	0.98	400
dataset/without_mask	0.97	0.98	0.98	400
accuracy			0.98	800
macro avg	0.98	0.98	0.98	800
weighted avg	0.98	0.98	0.98	800

Таблиця 3.5 – Результати моделі

Точність особи в масці, визначеної розробленою моделлю, забезпечує хороший стандарт прогнозу. На Рисунку 3.3 видно, що якість навчання збільшувалась поки не досягла 40 епох, після чого крива залишалася стабільною. Точність становила близько 98% після 100 епох. Зелена крива відображає точність навчання, а синя – набір даних перевірки. Крім того, на Рисунку 3.4, де показано криву втрат для навчання та перевірки, зелена лінія позначає втрати в наборі навчальних даних, менші за 0,1, а втрати в наборі



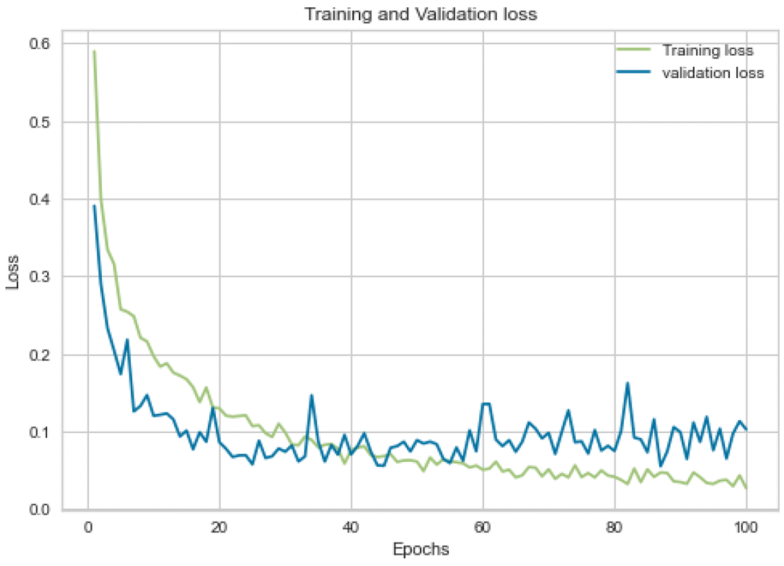
даних перевірки представлені синьою лінією, яка також менша. ніж 0,2 через 100 епох.

Рис 3.3 – Залежність точності від епохи

Рис 3.4 – Залежність функції витрат від епохи

3.5 Виявлення активності

Було реалізовано просте виявлення активності на відео задля економії місця на сервері, який зберігає записи відео. Очевидно, що нема сенсу зберігати відео з камер, якщо на них нічого не відбувається.



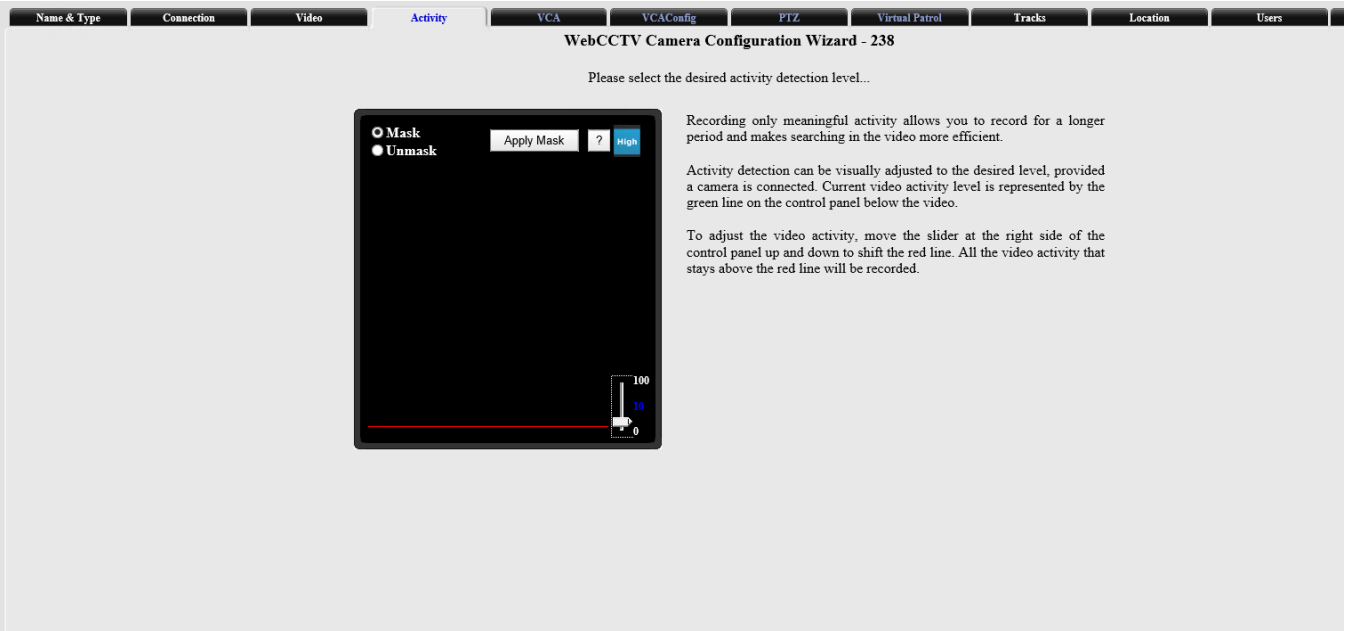


Рисунок 3.6 Приклад інтерфейсу налаштування виявлення активності

Було використано дуже простий алгоритм, який з порівнянніює два фрейми за модулем різниці між пікселями і починає зберігати відео тільки, якщо сумма модулів різниці пікселів більше за константу. Можна побачити інтерфейс для налаштування цього на рисунку 3.1.1. Можна було би використати машинне навчання, але замовники просили простий та передбачуваний механізм, тому було зроблено так.

```
public bool CheckActivity(byte[][] frame1, byte[][] frame2, int frameWdith, int frameheight, int diffConst)
{
    var diff = 0;
    for (int i=0; i< frameWdith; i++)
    {
        for (int j=0; j< frameheight; j++)
        {
            diff += Math.Abs(frame1[i][j] - frame2[i][j]);
        }
    }
    return diff >= diffConst;
}
```

Рисунок 3.7 Приклад реалізації такого методу виявлення активності на відео

РОЗДІЛ 4 АРХІТЕКТУРА ВЕБ-ЧАСТИНИ

4.1 Структурування проекту за допомогою NX

Для розробки даного проекту було обрано технологію NX для Angular. NX - це надбудова над фреймворком Angular, яка надає інструменти для розробки великих та складних додатків. Використання NX забезпечує розподіл додатку на окремі модулі (бібліотеки), які можуть бути легко перевикористані та перерозподілені для розвитку додатку у майбутньому.

У даному проекті було створено наступні бібліотеки:

- auth - відповідає за автентифікацію та авторизацію користувачів.
- core - містить загальний функціонал, який використовується в різних частинах додатку.
- identity - відповідає за роботу зі стороною сервера щодо ідентифікації користувачів та отримання інформації про них.
- sdk - надає API для роботи з серверною частиною додатку.
- shared - містить загальний функціонал, який використовується у декількох бібліотеках.
- sidebar - відповідає за відображення бокової панелі додатку.
- viewer - відповідає за відображення та роботу з відеопотоками.
- webapi - надає API для роботи з Web-сервером.

Кожна з цих бібліотек містить відповідний функціонал та може бути використана окремо або у поєднанні з іншими бібліотеками для створення більш складних функціональних можливостей додатку.

4.2 Використання абстракцій для покращення архітектури

Для того, щоб архітектура підтримувала багато різних способів комунікації та стрімінгу (в планах додавання технології WebRTC після її імплементації на OPC-сервері) було створено абстрактні класи, які мають бути реалізовано для використання нового способу комунікації. На рис 4.1 можна побачити клас, який необхідно реалізувати для додавання нового способу стрімінгу

```
export abstract class StreamSocket extends ApiSocket {
  abstract createLive(cameraId: string): Observable<StreamDto>;
  abstract createPlayback(cameraId: string, start: Date, end: Date): Observable<StreamDto>;
  abstract play(): Observable<void>;
  abstract pause(): Observable<void>;
  abstract stop(): Observable<void>;
  abstract seek(position: Date): Observable<void>;
  abstract extractFrame(): Observable<Frame>;
  abstract onChangeState: Observable<number>;
}
```

Рисунок 4.1 клас StreamSocket

Підхід з абстрагуванням дає можливість створювати класи і методи, які дозволяють працювати з API, не знаючи деталей його реалізації (див рис. 4.2) . Клас ApiProvider (рис 4.2 та рис 4.3) використовує абстрактні класи ApiSocket, CommandSocket і EventSocket, щоб взаємодіяти з різними типами сокетів. Він також містить методи send та on, які дозволяють відправляти запити на сервер та підписуватись на події, відповідно.

```
export class ApiProvider<TSocket extends ApiSocket & CommandSocket<TSocket> & EventSocket<TSocket>> {
  private socket$ = this.buildSocket();
  private events = new Map<EventKeys<TSocket>, Observable<unknown>>();

  constructor(
    private socket: TSocket | ApiUntypedSocket
  ) { }

  protected send<
    K extends CommandKeys<TSocket>,
    P extends unknown[],
    T,
    U extends InferFromObservable<ReturnType<TSocket[K]>>,
    R extends Parameters<TSocket[K]>
  >(
    endpoint: K,
    commandCtor: new (...params: P) => ApiCommand<T, U, [...R]>
  ): (...params: P) => Observable<T> {
    return (...params: P) => {
      const command = new commandCtor(...params);
      return this.socket$.pipe(
        switchMap(socket => socket instanceof ApiUntypedSocket
          ? socket.send<U>(endpoint, ...command.request)
          : socket[endpoint](...command.request) as Observable<U>),
        map(response => command.mapResponse(response))
      );
    };
  }
```

Рисунок 4.2 Клас ApiProvider

Однією з головних переваг цього коду є те, що він використовує абстрактні класи, які дозволяють розширювати його функціональність без необхідності змінювати сам код. Це робить його дуже гнучким і масштабованим. Крім того, ApiProvider може працювати з будь-якою реалізацією ApiSocket, незалежно від того, як вона реалізована, що є дуже зручним і дозволяє легко змінювати та оновлювати реалізацію сокетів без необхідності змінювати логіку в самому ApiProvider.

```
  private buildSocket(): Observable<TSocket | ApiUntypedSocket> {
    return this.socket.open().pipe(
      map(() => this.socket),
      share({
        connector: () => new ReplaySubject(1),
        resetOnError: false,
        resetOnComplete: false,
        resetOnRefCountZero: () => this.socket.close()
      })
    );
  }

  private buildEvent<K extends EventKeys<TSocket>, T, U extends InferFromObservable<TSocket[K]>>(
    endpoint: K, event: ApiEvent<T, U>
  ): Observable<T> {
    return this.socket$.pipe(
      switchMap(socket => socket instanceof ApiUntypedSocket
        ? socket.on<U>(endpoint)
        : socket[endpoint]() as Observable<U>),
      map(data => event.mapEvent(data)),
      finalize(() => this.events.delete(endpoint)),
      share()
    );
  }
}
```

Рис 4.3 Методи класу ApiProvider

4.3 Angular, rxjs та Observable<T>

Для отримання більш чіткого та ефективного коду в проекті використовується патерн "наблюдач". Цей патерн дозволяє підписуватись на певні івенти, які повідомлятимуть усіх підписників про зміни. Використання бібліотеки RxJS дозволяє реалізувати цей патерн у більшості випадків. Гарним прикладом такого використання може бути така функція:

```
private mosaicSizeChanged$ = this.mosaicForm.valueChanges.pipe(
  filter(() => this.mosaicForm.controls.rows.valid && this.mosaicForm.controls.cols.valid),
  distinctUntilChanged((v1, v2) => v1.rows === v2.rows && v1.cols === v2.cols),
  takeUntil(this.ngUnsubscribe$)
);
```

Рисунок 4.4 – приклад написання кода з використанням RxJs

Основні поняття RxJS, які вирішують управління асинхронними подіями, включають Observable, Observer, Subscription, Operators та Subject. RxJS поєднує в собі схему Observer з схемою Iterator та функціональне програмування з колекціями, що дозволяє керувати послідовностями подій ефективніше. Бібліотека RxJS надає можливість обробляти асинхронні події як колекції, використовуючи оператори, такі як map, filter, reduce тощо. Крім того, бібліотека містить супутникові (Observer, Schedulers, Subjects), які допомагають управляти асинхронними подіями і роблять код більш ефективним та легким у розумінні. Це дозволяє виконувати проекти на Angular з використанням RxJS більш швидко та ефективно.

Rx поєднує схему Observer із схемою Iterator та функціональне програмування з колекціями, щоб заповнити потребу в ідеальному способі управління послідовностями подій.

Основними поняттями RxJS, які вирішують управління асинхронними подіями, є:

- Observable: представляє ідею виклику колекції майбутніх значень чи подій.

- Observer: це сукупність зворотних викликів, яка вміє слухати значення, доставлені спостерігачем.
- Subscription: представляє виконання спостережуваного, в першу чергу корисна для скасування виконання.
- Operators: це чисті функції, які забезпечують функціональний стиль програмування колекцій з такими операціями, як map, filter, concat, reduce тощо.
- Subject: є еквівалентом EventEmitter, і єдиним способом мультикастингу значення або події для кількох спостерігачів.
- Schedulers: це централізовані диспетчери для контролю багатопочності, що дозволяють нам координувати, коли обчислення відбуваються, наприклад, setTimeout або requestAnimationFrame або інші.

РОЗДІЛ 5 ПРИКЛАДИ ІНТЕРФЕЙСУ

5.1 Інтерфейс веб-додатка

Цей проект містить складові частини, що дозволяють користувачам переглядати відеостріми в режимі прямого ефіру або відтворення, а також створювати мозаїчний вигляд декількох відеострімів одночасно. На рис 5.1 та рис 5.2 можна це побачити

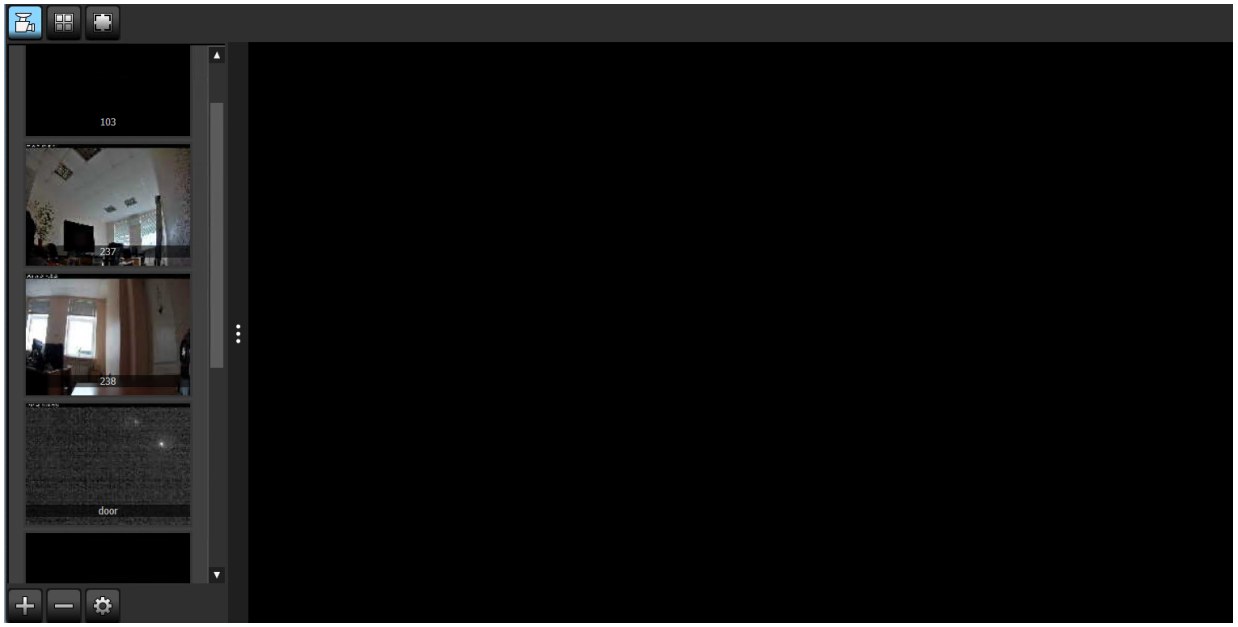


Рисунок 5.1 Приклад інтерфейсу перегляду підключених камер одного глобального серверу

Користувачі також мають можливість самостійно вибирати камери, які будуть включені в мозаїку або окремий стрім. Це дозволяє індивідуалізувати перегляд відео відповідно до потреб користувача, вибираючи лише необхідні джерела камер для спостереження.

Всі ці функції і можливості забезпечують більш гнучкий і зручний підхід до керування відеострімами та камерами в системі. Користувачі можуть легко відстежувати багато різних джерел відео і переключатися між ними з легкістю, що забезпечує зручність та ефективність в роботі з системою.

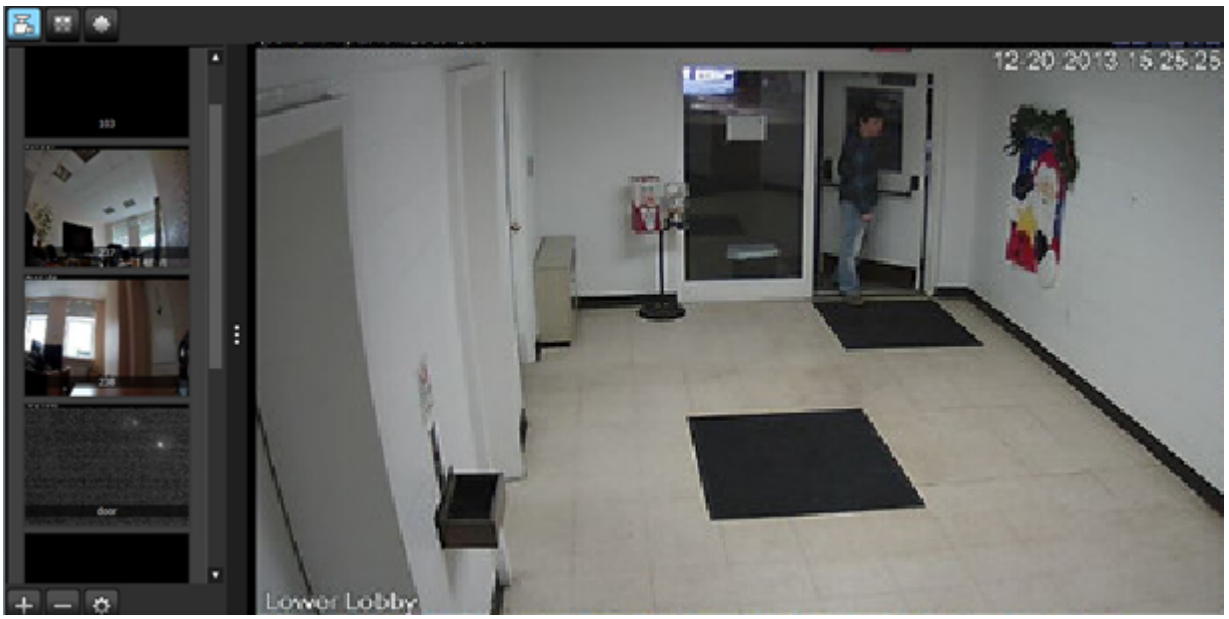


Рис 5.2 Приклад інтерфейсу перегляду лайв-стріму

Ця сторінка (рис 5.3) налаштувань камери дозволяє користувачам встановлювати різноманітні параметри для підключення до відеостріму. Вона надає зручний інтерфейс для налаштування IP-адреси, DNS-імені, порту, імені користувача та пароля для підключення до камери.

Додатково, користувачам доступні параметри доставки відео: унікаст або мультикаст. У режимі унікаст, відеопотік передається безпосередньо до користувача, використовуючи його унікастову IP-адресу. У режимі мультикаст, відеопотік транслюється до групової IP-адреси та порту, що дозволяє багатьом користувачам одночасно отримувати відео з одного джерела.

Цей розділ налаштувань надає користувачам гнучкість вибору і зміни параметрів підключення та доставки відео залежно від їх потреб і вимог. Користувачі можуть настроїти камеру зручним для них способом, забезпечуючи надійне та ефективне з'єднання для отримання відеостріму.

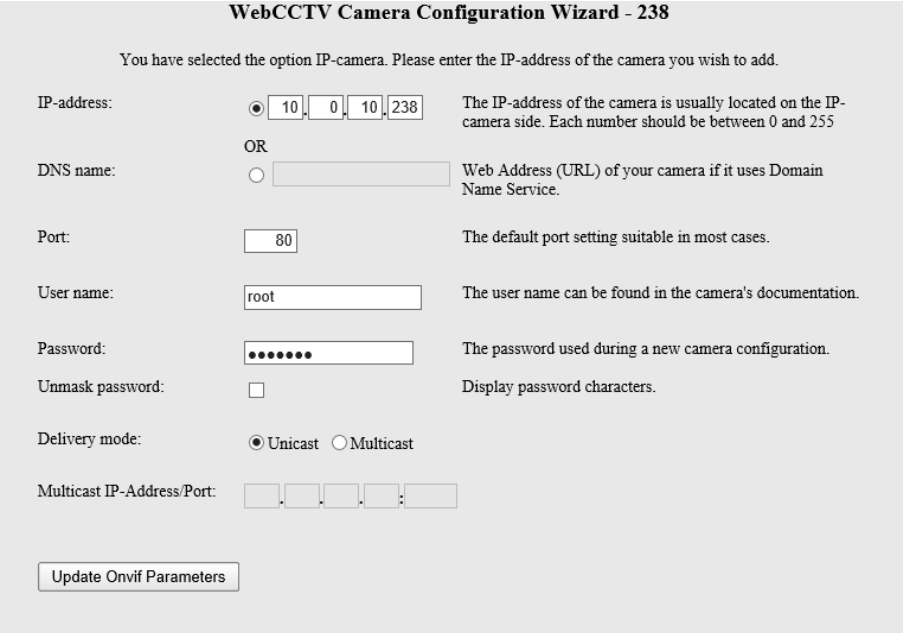


Рисунок 5.2 Додавння камери та її налаштування

Функціональність мозаїк (рис. 5.3) дозволяє користувачам створювати мозаїчний вигляд з відеострімів за допомогою готових шаблонів. В проекті доступні різноманітні шаблони, такі як 2x2, 3x3, 3x2, одне велике відео та три менші, та багато інших.

Вибір шаблону для мозаїки дозволяє користувачам організувати та спостерігати за декількома відеострімами одночасно на одному екрані. Це створює зручну можливість перегляду багатьох джерел відео одночасно і забезпечує гнучкість відстеження подій з різних камер без необхідності переключатися між окремими вікнами або екранами. Завдяки цій функціональності, користувачі можуть налаштовувати мозаїку згідно своїх потреб та вимог. Вони можуть вибрати оптимальний шаблон, який найкраще відповідає їх вимогам до перегляду відеострімів, або створити свій.

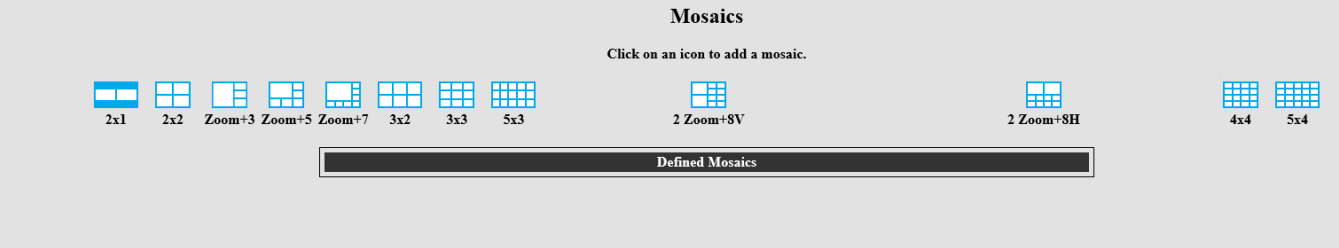


Рисунок 5.3 Мозайки (перегляд декількох камер одночасно)

ВИСНОВКИ

Результат роботи: проведено докладний аналіз деяких проблем, які було зустрінено під час розробки. Розроблено програму, що дозволяє здійснювати захищену трансляцію відео у реальному часі з камер відеоспостереження, спроектовано архітектуру сервера, яка буде легко підтримованою та масштабованою.

Рекомендації щодо використання результатів роботи: Оновити дизайн проекту.

Сфера застосування: розроблено для комерційного застосування з продажем пожиттєвих ліцензій та\або механізму підписки.

У даній роботі було розібрано ключові етапи проектування додатку та проаналізовано деякі проблеми, які траплялись під час розробки. Результати роботи:

1. Було спроектовано, розроблено та розглянуто структуру серверної частини.
2. Було реалізовано розпізнавання людей в масках та відслідковування активності для економії місця.
3. Був розроблен веб-додаток, який дозволяє користувачам додавати та користуватись камерами.
4. Було проведено оптимізації для стрімінгу лайв-відео або записів відео
5. Проведено аналіз ринку і визначено стратегію подальшого розвитку.

Отже, даний застосунок є придатним для комерційного застосування.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Surveillance Station | Synology Inc. [Електронний ресурс] - Режим доступу до ресурсу: https://www.synology.com/en-global/surveillance/

2. Специфікація Json Web Tokens [Електронний ресурс] - Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc7519/

3. Ozer J. Video Encoding by the Numbers: Eliminate the Guesswork from your Streaming Video / Jan Ozer., 2016. – 332 с.

4. Документація з C# [Електронний ресурс] - Режим доступу до ресурсу:

https://docs.microsoft.com/en-us/dotnet/csharp/

5. The H.264 Advanced Video Compression Standard, Second Edition, 2010.

6. Документація RxJS [Електронний ресурс] - Режим доступу до ресурсу:

https://rxjs-dev.firebaseapp.com/

7. Long B. Understanding Video Codecs: From MPEG to H.264 (and Beyond) / Long Ben.

8. Стандарти для security-камер [Електронний ресурс] - Режим доступу до ресурсу: https://www.embedded.com/ip-video-surveillance-standards/.

9. Документація з Angular [Електронний ресурс] - Режим доступу до ресурсу:

https://angular.io/docs

10. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures / N.Ford, M. Richards, P. Sadalage, Z. Dehghani., 2021.