Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня ____магістра___
(бакалавра, спеціаліста, магістра)

студента _____Муштата Олександра Олександровича_____
(ПІБ)

академічної групи _____123м-20-1_____
(шифр)

спеціальності _____123 «Комп'ютерна інженерія»_____
(код і назва спеціальності)

за освітньо-професійною програмою ___«Комп'ютерна інженерія»___
(офіційна назва)

на тему «Комп'ютерна система контролю наповненості фітнес клубу «Спортлайф» в умовах COVID-19»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Соколова Н.О.			
розділів:				
теоретичний розділ	доц. Соколова Н.О.			
синтез комп'ютерної системи	доц. Ткаченко С.М.			
розроблення програмного забезпечення	ас. Бешта Л.В.			
Рецензент	доц. Реута О.В.			
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2022

ЗАТВЕРДЖЕНО:

завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ __Гнатушенко В.В._
(підпис) (прізвище, ініціали)
«___»_____ 202__ року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____магістр _____
(бакалавра, спеціаліста, магістра)

студенту _____Муштату О. О.____ **академічної групи** ____123м-20-1____
(прізвище та ініціали) (шифр)

спеціальності _____123 «Комп'ютерна інженерія»_____

за освітньою-професійною програмою _123 «Комп'ютерна інженерія»__
(офіційна назва)

на тему «Комп'ютерна система контролю наповненості фітнес клубу «Спортлайф» в умовах COVID-19»,

затверджену наказом ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. №1036

Розділ	**Зміст**	**Термін виконання**
Стан питання і постановка задачі	На основі матеріалів виробничих практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	20.09.2021
Теоретичний розділ	Аналіз хмарних сервісів та технології розпізнавання об'єктів на зображеннях	13.10.2021
Синтез комп'ютерної системи	Розробка апаратної частини комп'ютерної системи контролю наповненості	01.12.2021
Розроблення програмного забезпечення	Розробка програмного забезпечення	15.12.2021
Експериментальний розділ	Тестування розробленої комп'ютерної системи контролю наповненості	20.12.2021
Графічна частина	Графічні результати роботи подати у вигляді рисунків схем таблиць на 10 арк. формату А4.	20.12.2021

Завдання видано _____ доц. Соколова Н.О.
(підпис керівника) (прізвище, ініціали)

Дата видачі 06 вересня 2021 р.

Дата подання до екзаменаційної комісії 10.01.2022 р.

Прийнято до виконання _____ Муштат О. О.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Об’єкт дослідження: фітнес клуб “Спортлайф” в умовах карантину в зв'язку з COVID-19.

Предмет дослідження: контроль наповненості території фітнес клубу “Спортлайф” методами програмних та хмарних технологій.

Мета кваліфікаційної роботи: комп'ютерна система наповненості фітнес клубу “Спортлайф” в умовах COVID-19.

Кваліфікаційна робота присвячена актуальній задачі контролю наповненості території фітнес клубу в карантинних умовах.

Наукова новизна виконаної роботи полягає в:

— дослідженні і обґрунтуванні застосування вибраних технологій для реалізації контролю наповненості фітнес клубу

— розробка програмного забезпечення для відстежування наповненості та переміщень на території клубу за допомогою програмного забезпечення на базі Python, AWS, Meraki та Node-Red.

— реалізація технологій, побудованих на машинному навчанні AWS Rekognition

Практична цінність результатів полягає в тому, що розроблена програмно-інформаційна технологія розширює можливості стандартного процесу відстежування і контролю наповненості та робить його автоматизованим.

Список ключових слів: Python, AWS, Meraki, Node-Red, AWS Rekognition.

ABSTRACT

Object of research: fitness club “Sportlife” in quarantine in connection with COVID-19.

Subject of research: control of the filling of the territory of the fitness club “Sportlife” with the methods of software and cloud technologies.

The purpose of the qualification work: computer system of filling of the fitness club “Sportlife” in the conditions of COVID-19.

Qualification work is devoted to the urgent task of controlling the fullness of the fitness club in quarantine conditions.

The scientific novelty of the work is:

- research and justification of the use of selected technologies for the control of the fitness club

- Development of software for tracking fullness and movement on the territory of the club using software based on Python, AWS, Meraki and Node-Red.

- implementation of technologies based on machine learning AWS Rekognition

The practical value of the results is that the developed software and information technology expands the capabilities of the standard process of tracking and control of filling and makes it automated.

Keyword list: Python, AWS, Meraki, Node-Red, AWS Rekognition.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS Rekognition - сервіс, що дозволяє легко додати до програми потужні інструменти для аналізу зображень.

Meraki - це найбільший у галузі сервіс, що надає доступ до хмарних технологій.

Node-Red - це інструмент візуального програмування для інтернету речей, що дозволяє підключати один до одного пристрої, API та онлайн-сервіси.

Python - це скриптова мова, що активно розвивається, яку використовують для вирішення великого обсягу найрізноманітніших проблем і завдань.

Amazon Elastic Compute Cloud (Amazon EC2) – це веб-сервіс, що надає безпечні масштабовані обчислювальні ресурси у хмарі.

ООП - об’єктно-орієнтоване програмування.

URL (Uniform Resource Locator) – це унікальна адреса веб ресурсу

HTTP – Hyper Text Transfer Protocol

REST – Representational State Transfer — «передача стану уявлення» — архітектурний стиль взаємодії компонентів

АКС – автоматизована комп’ютерна система

АРМ – автоматизоване робоче місце

CRM (Customer Relationship Management) - «управління взаємовідносинами з клієнтами» - створення єдиної екосистеми по залученню нових і розвитку існуючих клієнтів.

ВСТУП

Актуальність роботи. В нинішніх реаліях досить важко дотримуватись правил поведінки в громадських місцях, таких як фітнес-клуби. Умови, задані нам карантинними нормами COVID-19 досить сильно впливають на зручне та практичне ведення бізнес-процесами будь-якої компанії. Велика кількість компаній по обслуговуванню клієнтів стрімко переходять на автоматизовані системи. Це обумовлюється тим, що зменшення попиту на послуги компанії несе за собою збитки. Таким чином розробка та впровадження автоматизованих систем в бізнес-процес компанії є вигідним рішенням як для заощадження бюджетних коштів, так і для подальшого поліпшення якості послуг, які надає компанія.

Об’єкт дослідження: фітнес клуб “Спортлайф” в умовах карантину в зв'язку з COVID-19.

Предмет дослідження: контроль наповненості території фітнес клубу “Спортлайф” методами програмних та хмарних технологій.

Мета й завдання дослідження. Мета роботи полягає у створенні автоматизованої системи контролю наповненості фітнес клубу “Спортлайф”.

Відповідно до мети й предмета дослідження у кваліфікаційній роботі необхідно вирішити наступні завдання:

— налаштування вибраних технологій для взаємозв'язку

— розробити та реалізувати програмне забезпечення для відстежування наповненості та переміщень на території клубу за допомогою обраних технологій.

Робота складається з трьох розділів. Перший розділ присвячено аналізу теми дослідження та постановці задачі. Наведено огляд хмарного сервісу Amazon Elastic Compute Cloud, коротко розглянутий протокол передачі даних MQTT.

В другому розділі наведено проектну складову вирішення завдання. Розглянуто деякі відомі методи побудови архітектури автоматизованої системи

контролю наповненості фітнес клубу “Спортлайф”. Обґрунтовано вибір інструментів програмної реалізації інформаційної системи.

Третій розділ присвячено розробці програмного забезпечення Виконано реалізацію інформаційної технології. Створено програмний продукт на мові Python.

Практична цінність результатів полягає у тому, що дана побудована система з вибраними технологіями розширює можливості стандартного процесу відстежування і контролю наповненості та робить його автоматизованим.

Висновки. Створено автоматизовану систему для відстежування переміщень людей по території фітнес-клубу. Розроблений програмний додаток за допомогою Python для реалізації створеної та налаштованої системи.

Наприкінці продемонстровані знімки, які містять в собі отриману інформацію про стан переміщень людей на момент їх створення.

1. СТАН ПИТАННЯ І ПОСТАНОВКА ЗАДАЧІ

1.1 Вплив COVID-19 на підприємства

За останні п’ятнадцять років світ охопило декілька пандемій: атипова пневмонія (SARS-1, 2002р.), пташиний грип (вірус H5N1, 2003р.), свинячий грип (вірус A/H1N1, 2009р.), вірус Ебола (EVD, 2013р.). За різними джерелами найбільша смертність була викликана свинячим грипом, а найбільших збитків (1,5 трлн. доларів) світовій економіці завдала пандемія пташиного грипу [15]. Однак справжнім викликом людству став COVID-19 (SARS-2 Cov, 2019р.). За даними ВООЗ захворіли понад 179 млн. людей [16], а втрати світової економіки тільки у 2020 році за різними оцінками складають 20 трлн.доларів. У нещодавно опублікованому спільному з Bank Pekao звіті (було опитано тисячі компаній по всьому світу) відзначається, що більшість (55%) з них дуже постраждали від пандемії. Однак ці ефекти розподілені нерівномірно. Невеликі компанії, як правило, більше страждають від Covid-19, ніж великі [17].

Комп’ютерні технології почали набувати велике значення в багатьох секторах бізнесу ще до 2019 р. До початку пандемії почала спостерігатися більша зацікавленість на залучення програм, направлених на автоматизацію бізнес-процесів компаній і підвищення важливості електронної комерції, а пандемія зміцнила їх позиції ще більше. Багато підприємств було вимушено змінити бізнес-моделі, щоб забезпечити виконання замовлень і надання послуг через цифрові платформи та комп’ютерні системи. В результаті комп’ютерні технології стали ключем до забезпечення безперервності роботи бізнесу. Імовірно, після пандемії бізнес, і надалі буде працювати як у звичайному режимі роботи та і використовувати комп’ютерні технології для ведення підприємницької діяльності.

Проаналізувавши (рис.1), які сфери діяльності найбільше постраждали під час пандемії, можна зробити висновок, що найбільше постраждали ті організації,

які пов’язані із залученням та створенням масових зібрань, зустрічей декількох та більше людей [18].

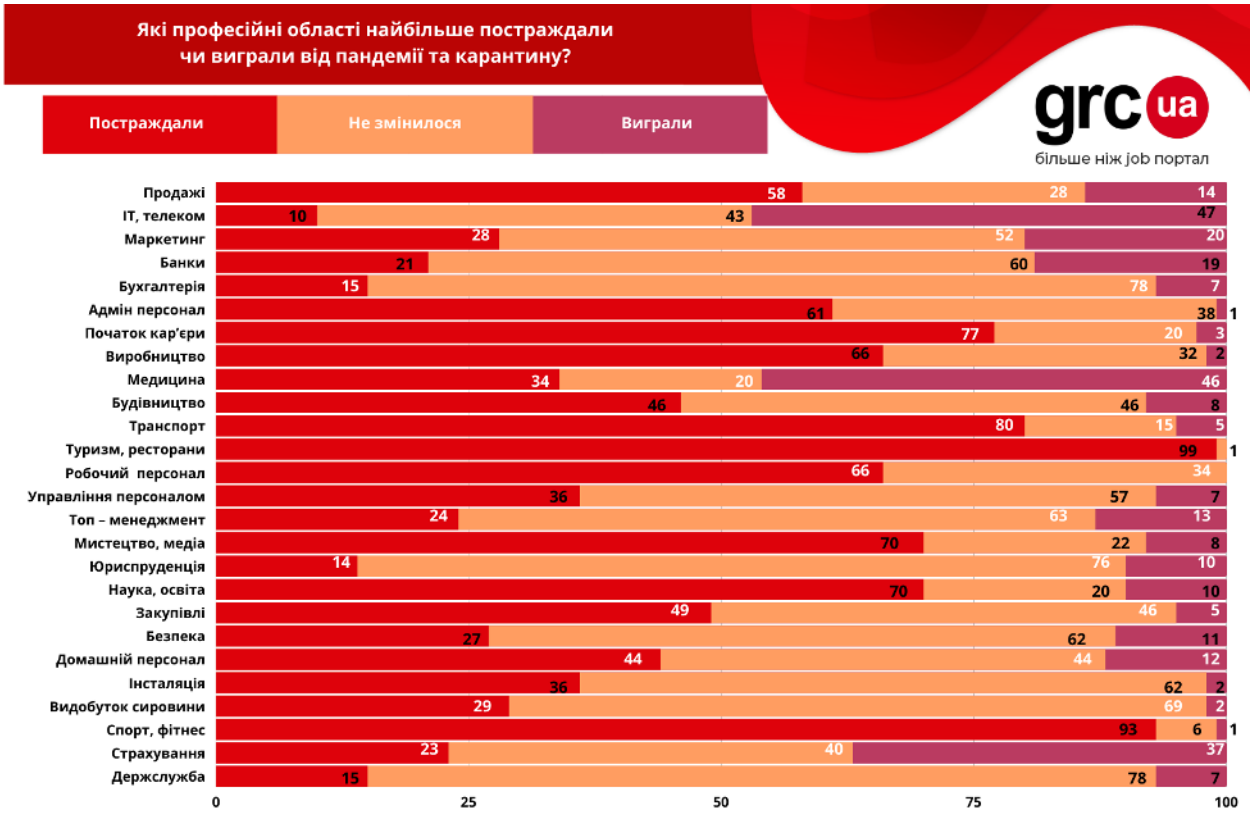

Рисунок 1.1 — Найбільш постраждалі професійні області від пандемії та карантину.

Карантинні обмеження зменшують та/або заморожують економічну активність підприємств, призводять до зменшення користуванням послугами, зменшення продажів товарів, користуванням послугами, спричиняють зменшенню прибутків, що безпосередньо впливає на оцінку економічної ситуації підприємств і їх подальшої перспективи [19]. Карантинні обмеження, передбачені постановою № 392 кабміну про запровадження суворіших правил безпеки та санітарії, передбачає посилення контролю наповненістю для організації роботи підприємства оф лайн. Відповідальність за дотримання і виконання карантинних умов лежить на власниках підприємств.

Вимоги щодо роботи закладів фізичної культури та спорту відповідно до Постанови Головного державного санітарного лікаря України №54 від

18.09.2020р. «Протиепідемічні заходи в закладах фізичної культури та спорту на період карантину у зв'язку з поширенням коронавірусної хвороби» [20] передбачають наступні дії:

– перед початком зміни проводиться температурний скринінг працівникам;

– усім відвідувачам проводиться температурний скринінг при вході до закладу;

– дозволяється одночасне перебування клієнтів у закладі з розрахунку не більше однієї особи на 5 кв.м., але за умов встановлення «зеленого» та «жовтого» рівні епідемічної небезпеки;

– на території на якій установлено «помаранчевий» рівень у закладі можуть перебувати не більше ніж 220 осіб та не більше 1 особи на 10 кв.м.;

– проведення групових занять проводиться виключно за попереднім записом;

– на вході до закладу організовується місце для обробки рук спиртовмісними антисептиками. Доцільно розмістити яскравий вказівник про необхідність дезінфекції рук;

– у зонах проведення занять розміщуються додаткові місця для обробки рук антисептиками, паперові рушники, контейнери для їх утилізації;

– заходи індивідуальної профілактики доцільно періодично транслювати на екранах;

– на період карантину забороняється розміщення в доступі клієнтів рекламних брошур;

– на період карантину забороняється функціонування спа-зон та надання послуг масажу;

– на період карантину забороняється функціонування кулерів з водою. Продаж їжі та напоїв в індивідуальній упаковці дозволяється;

– функціонування басейнів дозволено для індивідуального відвідування;

– кількість клієнтів, що одночасно перебуває у закладі, повинна враховувати та не допускати потенційне скупчення в зонах роздягалень, санітарних та адміністративних зонах;

– тренажери для кардіо-занять, що передбачають тривалий цикл використанням одним клієнтом, повинні бути розміщені з інтервалом не менше 2,5 м один від одного;

– працівники повинні бути одягнені в медичну маску;

– після кожного зняття ЗІЗ та одягання працівник повинен ретельно помити руки з милом або обробити антисептиком;

– працівники закладу повинні мати п’ятиденний запас ЗІЗ;

– забезпечити постійну наявність рідкого мила, паперових рушників в санвузлах;

– інформування щодо встановлених обмежень при вході;

– вологе прибирання з використанням дезінфекційних засобів кожні 2 години;

– провітрювання приміщень не менше 15 хвилин перед відкриттям та кожні 2 години протягом дня;

– тимчасове відсторонення від роботи осіб з групи ризику;

– самоізолюватись у разі виникнення симптомів респіраторних захворювань.

1.2 Робота фітнес клубів.

В звичайному режимі роботи фітнес клуби надають серед всіх інших і такі послуги своїм клієнтам, як інформаційні данні, такі як режим роботи закладу, інформування про те, які є спортивні зали (басейн, тренажерних зал, тенісні корти, футбольні поля, масажні та SPA кімнати, роздягальні та інші), які існують заняття у фітнес клубі (аквааеробіка, плавання, дитяче плавання, тренажерний зал, бокс, фітнес, йога, масаж, танці та інші), розклад занять індивідуальних та

групових занять, інформація про тренерів, новини закладу, акції, контактні данні та зворотній зв’язок.

Для організації зручного ведення бізнесу власникам необхідно організувати:

- налагоджений процес контролю відпрацювання свого робочого часу обслуговуючим персоналом на робочих місцях;
- ведення клієнтської бази даних, в якій міститься така інформація, як ПІБ, вік, зацікавлена спортивна секція, оплата послуг, кількість відвідувань, можливість додання бонусів, для заохочення клієнта;
- корпоративний чат для швидкого вирішення важливих питань;
- ведення документообігу, бухгалтерії, аналіз даних;
- постійний облік присутніх на території закладу.

Особливо важливо організувати налагоджену систему роботи фітнес клубу в умовах COVID- 19, оскільки підприємству необхідно дотримуватись санітарних норм, які вимагає законодавство і до звичайного режиму роботи додаються додаткові завдання та обов’язки для співробітників.

За таким умов стає доцільним організувати автоматизоване ведення бізнесу, для швидкого, правильного та коректного вирішення запитання.

1.3 Поняття про CRM та її види

Автоматизація бізнес-процесів дозволяє організувати та удосконалити структуру у процесах підприємства. Особливо актуальне питання запровадження автоматизації бізнес-процесів в умовах пандемії COVID-19. Для автоматизації застосовують CRM системи. CRM-системи (Customer Relationship Management, або управління відносинами з клієнтами) – це прикладне програмне забезпечення для організацій, призначене для автоматизації стратегій взаємодії з потенційними та наявними клієнтами. Застосування CRM на різних видах підприємств набирає обертів, оскільки CRM бере під контроль всі канали комунікацій з клієнтами, підказує, що робити і автоматизує роботу бізнесу.

Варіацій CRM-систем так само багато, як і сфер бізнесу. Кожна програма прагне врахувати особливості певних функцій бізнесу. Все розмаїття видів CRM можна розділити на чотири групи: операційні, аналітичні, колаборaційні та комбіновані [25].

Операційні с. З назви видно, що CRM даного виду націлені насамперед на спрощення операційної сторони роботи. В її функціонал входить автоматизація процесів, організація клієнтської бази, контактів та фіксація даних проєкту на всіх етапах продажів продукту або послуги, постановка завдань і контроль роботи співробітників.

Аналітичні CRM. У CRM-системах цього виду основний наголос зроблений не тільки на зборі, але й на наступному аналізі зібраних даних. Такі CRM-системи використовуються, коли необхідно згенерувати звітність, провести аналіз інформації , наприклад, продажів, маркетингових заходів чи в залежності від регіонів клієнтів; визначити рентабельність, проаналізувати воронку продажів, простежити поведінку клієнтів на різних етапах. Отже, аналітичні CRM допомагають у складанні прогнозів і оцінки ефективності маркетингової стратегії.

Колаборaційна CRM. Це рішення для тих продуктів, які розробляються за безпосередньої участі споживачів із використанням різних каналів зв'язку (через інтернет-портал, телефонію, особисті контакти та ін.). Проведення опитувань для зміни якості продукту або порядку обслуговування, веб-сторінки для відстеження клієнтами стану замовлень, розсилка повідомлень по SMS, надання клієнтам можливості самостійно вибирати і замовляти продукти і послуги, а також інші інтерактивні можливості.

Комбіновані CRM – це найбільш універсальні CRM системи, оскільки поєднують автоматизацію і аналітику процесів. Більшість компаній потребують запровадження саме систем такого типу. Також ці системи можуть інтегруватися з іншими програмними продуктами. Наприклад, з Gmail, завдяки чому CRM

з'являється прямо в інтерфейсі вашої пошти. Це забезпечує максимальну зручність та комфорт роботи.

Для того щоб обрати CRM систему недостатньо вибрати тільки її вид. Потрібно вирішити це буде універсальна готова система чи розроблена під конкретне підприємство; де буде зберігатися інформація, на власних серверах організації чи на хмарному сервіс (який ще називають SaaS- продуктом); з якими сервісами є інтеграція та інше.

CRM системи які працюють на власних серверах компанії, потребують більше часу на розгортання, їх адміністрування, налаштування, працездатність і зберігання даних повністю лежить на співробітниках компанії. Підприємство купує програмних продукт та встановлює його на власний комп'ютер. Даний вид CRM є більш витратним і підходить в основному великим компаніям, оскільки включає в себе наявність спеціального обладнання та утримання технічного персоналу.

З іншого боку, програмне забезпечення, як послуга (SaaS) або хмарна CRM зберігає дані на серверах сертифікованого хмарного провайдера. У цьому випадку користувачі мають повний доступ до бази даних, де б вони не знаходилися. Важливим є те, що хмарна CRM має високу масштабованість. Це означає, що організація може легко розширювати не тільки функціональні можливості системи, але і її продуктивність по мірі зростання вимог бізнесу [28]. Єдиною вимогою до цієї хмарної системи є надійне підключення до інтернету.

1.4 Порівняння існуючих CRM систем

Основними продуктами серед CRM-систем на ринку України відзначається Бітрікс24, Amocrm, Мегаплан, NetHunt CRM, Terrasoft Salesforce, Zoho, OneBox.

Бітрікс24. Бітрікс це цілий комплекс інструментів для розвитку бізнесу, один з них своя CRM. Її можна поєднати з 1С, а також з інтернет-магазином, створеним у сервісі Бітрікс24.Магазини.

Тестовий період – ні, але є безкоштовний тариф.

Робота в хмарі - є хмарна та коробкова версія

Інтеграція з сервісами - інтеграція з усіма інструментами Бітрікс.

Коллтрекінг - оренда номера або підключення АТС.

Сервіси документообігу - створення рахунків та накладних, шаблони документів через сервіс 1С: Бекофіс.

Управління персоналом – можна скласти план для кожного співробітника та аналізувати продажі, є чат для співробітників, є розмежування прав доступу до CRM.

Можливості CRM:

— зрозумілий інтерфейс — картки як канбана, як і Trello, видно рух по угодам.

— детальна картка клієнта – історія від першого запиту до оцінки клієнтом роботи працівника.

— чати між співробітниками можна зробити чат всередині картки.

— телефонія — оренда віртуального номера або підключення АТС, всі дзвінки та записи розмов зберігаються в системі.

— автоматизація продажів - роботи надсилають клієнтам листи, голосові повідомлення, SMS.

— CRM термінал — для оплати покупки за QR-кодом, посиланням у sms або готівкою.

— генератор продажів - для повторних угод (клієнту, який здійснив покупку, надходять пропозиції з особливими умовами).

— центр продажів - інструменти, що дозволяють пропонувати клієнтам найзручніший варіант оплати, наприклад, через месенджери або в соціальних мережах.

— інструменти маркетингу – розсилки та запуск рекламних кампаній прямо з CRM.

— AI Скоринг - аналізує дані і передбачає вихід угоди.

— Наскрізна аналітика — для аналізу ефективності рекламних кампаній вже вбудована в CRM.

Плюси:

— мобільний додаток;

— великий вибір тарифів;

— є готові CRM-системи по галузях бізнесу – для інтернет-торгівлі, юридичних компаній, оптовиків тощо;

— конструктор інтернет-магазину із вбудованою CRM;

— безкоштовний онлайн-чат для сайту;

— інструмент продажу через соцмережі та месенджери;

— обмін даними з 1С;

— є API для підключення сторонніх сервісів;

— рейтинг клієнтів - для більш щільної роботи з постійними клієнтами та тими, хто приносить найбільше прибутку;

Мінуси:

— може здатися складною, потрібен час вивчення всіх функцій;

— громіздка для малого бізнесу, більше підходить для середнього та великого бізнесу.

Amocrm. AmoCRM - протилежність Бітрікс24, розробники позиціонують її як найлегшу у використанні CRM. Це хмарна система з основними функціями – воронка продажів, картки клієнтів, інтеграція з сервісами телефонії.

Тестовий період – 14 днів.

Робота в хмарі – так.

Інтеграція з сервісами – понад 20 сервісів для інтеграції.

Коллтрекінг є інтеграція з сервісами телефонії.

Сервіси документообігу – підключення зовнішніх сервісів.

Управління персоналом - контроль роботи менеджерів, вбудовані месенджери для спілкування співробітників між собою.

Можливості CRM:

— digital воронка - для автоматизації продажів. Клієнти одержують розсилки, бачать рекламу, роблять дії. На основі їхньої реакції можна розробити подальшу стратегію продажів.

— сканер візиток — дозволяє вносити контакти в CRM через мобільний додаток.

— інтеграція з IP-телефонією дозволяє здійснювати дзвінки прямо з карток і вести статистику дзвінків.

— вбудований месенджер для спільної роботи співробітників.

Плюси:

— легка у використанні - не потрібно звертатися до сторонніх фахівців для налаштування та навчання;

— можна працювати із планшета;

— можна настроїти систему під себе;

— є мобільний додаток.

Мінуси:

— немає вбудованого коллтрекінгу, необхідно підключатися до зовнішнього сервісу.

Мегаплан. Мегаплан – проста CRM для бізнесу, працює понад 12 років. Безперечна перевага даної CRM — продумана робота з персоналом, наприклад, у Мегаплані відображається графік відпусток, є свій сервіс для відеоконференцій, повідомлення про прострочені завдання.

Тестовий період – 14 днів.

Робота в хмарі є хмарна і коробкова версія.

Інтеграція з сервісами – 70+ інтеграцій.

Коллтрекінг - вбудована телефонія + інтеграція із зовнішніми сервісами.

Сервіси документообігу — через зовнішні послуги є шаблони документів.

Управління персоналом – таск-менеджер для розподілу завдань, відеодзвінки з персоналом.

Можливості CRM:

— модуль продажів - єдина клієнтська база, воронка продажів, картки клієнтів.

— інструменти для контролю за співробітниками та спільної роботи - таск-менеджер, відеоконференції, контроль термінів виконання завдань, повідомлення, звітність, облік часу, витраченого на роботу, корпоративне листування, робочий календар, графік відпусток.

— інтеграція з WhatsApp - можна здійснювати продаж через месенджер.

— спрощена робота з документами та рахунками – автозаповнення реквізитів, шаблони документів.

NetHunt CRM. NetHunt CRM - система автоматизації процесів продажу, повністю інтегрована в Gmail та інші сервіси системи Google. NetHunt дозволяє вести базу контактів, управляти продажами, візуалізувати воронку продажів, спілкуватися з клієнтами, автоматизувати процеси, аналізувати дані та ставити завдання прямо зі своєї поштової скриньки Gmail.

Тестовий період – 14 днів.

Робота в хмарі – так.

Інтеграція із сервісами — пряма інтеграція із 10 сервісами, необмежена через Zapier.

Коллтрекінг - підключення зовнішніх сервісів.

Сервіси документообігу – підключення зовнішніх сервісів.

Управління персоналом – так.

Можливості CRM:

— інтеграція з Gmail та іншими програмами Google Workspace (Google Контакти, Google Календар, Google Диск тощо)

— додавання лідів в один клік або автоматично з багатьох джерел, таких як листи, соціальні мережі, повідомлення в чатах на сайті, веб форми на сайті та ін.

— детальні картки клієнтів з полями, що настроюються, і повною історією взаємодій, коментарями менеджера, супутніми документами.

— сегментація клієнтів для налаштування персоналізованих листів. Можливість зберігати фільтри.

— візуалізація воронки продажів та відстеження угод на кожному етапі.

— розширений функціонал для відправки email-розсилок з CRM безкоштовно (шаблони листів, функція "відписатися від розсилки", перегляд листа, тестовий лист, і т.ін.)

— автоматизація безлічі бізнес-процесів і шляхи клієнта від першого контакту до закриття угоди на підставі різних тригерів (відповідь на лист, зміни в картці клієнта і т.ін.)

— можливість налаштовувати серії листів для розсилок, які поступово підігрівають клієнтів для укладання угод.

— звіти для відстеження результатів команди продажу та досягнення цілей.

Плюси:

— надійність: NetHunt CRM - сертифікований партнер Google.

— відсутність обмежень на кількість записів у CRM у будь-якому тарифі.

— зрозумілий інтерфейс, для освоєння якого не потрібне спеціальне навчання. Підтримка та допомога з налаштуванням.

— швидка технічна підтримка.

— автоматична моментальна прив'язка вхідних та вихідних листів до картки ліда чи клієнта.

— висока налаштованість системи під конкретний бізнес та масштабованість.

— взаємодія команди всередині CRM за допомогою @згадувань.

— зручний імпорт даних із інших систем.

— автоматичне створення завдань для менеджера у межах функціоналу автоматизації.

— є мобільний додаток Android/iOs, веб-версія та версія для Gmail, які синхронізовані між собою.

Мінуси:

— відсутня пряма інтеграція із 1-С.

— немає інтеграції зі складом.

1.5 Висновки до розділу

Таким чином, проаналізувавши функціонал CRM систем, зроблено висновок, що існуючи CRM системи не відстежують дистанцію між клієнтами . а також не підраховують кількість відвідувачів на території організацій, що унеможливлює контроль в реальному часі за наповненістю окремих приміщень із розрахунку необхідної кількості людей на визначену квадратуру приміщення, що є необхідністю для роботи підприємств в умовах пандемії COVID-19.

2. ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Хмарний сервіс Amazon Elastic Compute Cloud.

Хмарні сервіси, або «хмари» - це мережа потужних комп'ютерів - серверів, які дозволяють клієнтам користуватися своїми ресурсами через інтернет: зберігати файли та обмінюватися ними, працювати в онлайн-офісах, проводити обчислення [11].

У вузькому сенсі хмарні сервіси - це онлайн-програми, які допомагають організувати віддалену роботу та вирішувати бізнес-завдання. Співробітники отримують доступ до загальної бази даних із будь-якої точки світу і можуть керувати проектами. Кожен працівник бачить результат у реальному часі, може вносити коментарі, правки та виконувати персональні чи спільні завдання. Приклад хмарного сервісу - Google Документи.

Хмарні послуги приходять на зміну традиційним «коробковим» офлайн-програмам, які встановлюють на окремі комп'ютери. Приклад “коробки” - Microsoft Office.

Хмарна мережа складається із вузлів зберігання інформації, які називаються дата-центрами. Це цілі будівлі, заповнені величезними шафами із серверним обладнанням. Вони розташовані по всьому світу та пов'язані через інтернет. Будують та обслуговують обладнання хмарні провайдери. Вони ж розподіляють ресурси «залізних» серверів на окремі віртуальні машини та здають їх в аренду.

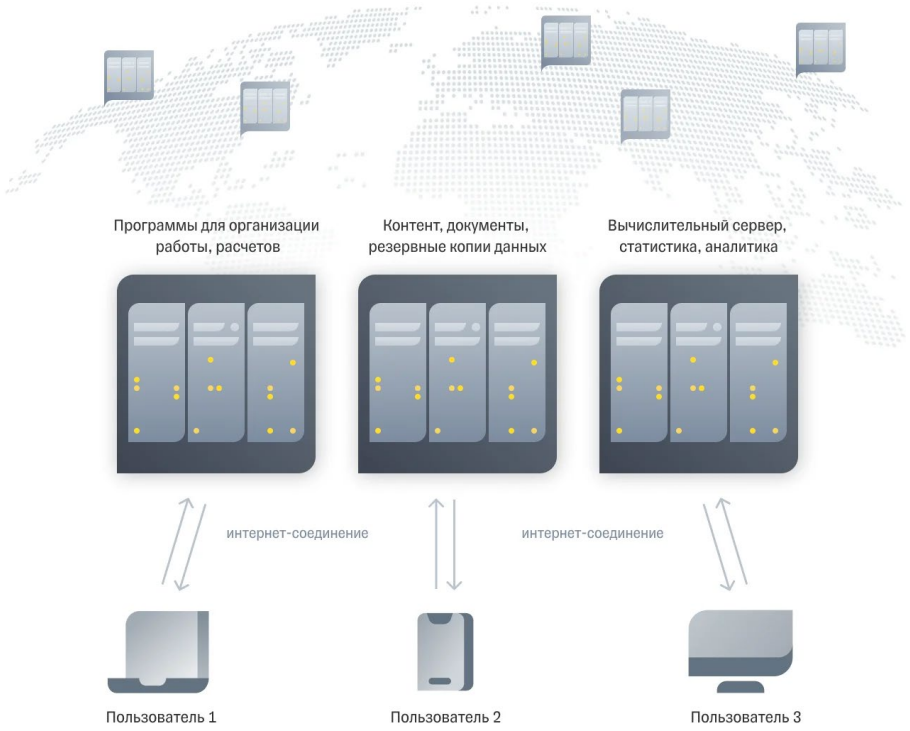

Рисунок 2.1 — Схема роботи хмарних мереж

2.1.1 Принцип роботи хмарних сервісів

Користувачі підключаються до хмарних серверів через інтернет і через них надсилають та отримують інформацію, використовують програми та інші онлайн-інструменти для роботи.

Обчислювальне облако Amazon Elastic Compute Cloud (Amazon EC2) – це веб-сервіс, що надає безпечні масштабовані обчислювальні ресурси в хмарі [11]. EC2 пропонує безліч варіантів віртуальної розробки та використання будь-якої програми. Він допомагає розробникам, спрощуючи проведення обчислень у хмарі масштабу всього Інтернету. Простий веб-інтерфейс Amazon EC2 дозволяє отримати доступ до обчислювальних ресурсів і налаштувати їх з мінімальними зусиллями. Він надає користувачам повний контроль над обчислювальними ресурсами, а також перевірене обчислювальне середовище Amazon для роботи.

Amazon EC2 пропонує обчислювальну платформу з найбільш широкими та досконалими функціональними можливостями, яка дозволяє вибрати процесор,

сховище, мережу, операційну систему та модель покупки. Дана платформа пропонує найшвидші процесори в хмарі і є єдиним простором хмари з мережею Ethernet, пропускна здатність якої становить 400 Гбіт/с, найпотужніші інстанси на базі графічних процесорів для організації курсів з машинного навчання та графічного відображення робочих навантажень, а також інстансами з найнижчими витратами на кожен логічний висновок інстансів у хмару [11]. На AWS виконується більше робочих навантажень SAP, HPC, машинного навчання та Windows, ніж у будь-якій іншій хмарі.

Управління AWS здійснюється як за допомогою веб-інтерфейсу (AWS console), так і за допомогою Command Line Tools. У консолі зібрані всі послуги AWS, але функціональність налаштування трохи обрізана. У командній рядку можна гнучкіше настроїти той чи інший сервіс, так само доступні закриті в консолі функції. EC2 дозволяє запускати вже заздалегідь налаштовані сервери з встановленими ОС: Amazon Linux, Red Hat EL, Suse ES, Windows 2008, Oracle EL. Балансування навантаження та автомасштабування є дуже важливими функціями EC2. Можна створити правила, при яких можна автоматично збільшити кількість серверів, наприклад, якщо один або кілька серверів не справляються з навантаженням. Контроль за здоров'ям серверів веде ще один сервіс AWS – Amazon Cloud Watch. За допомогою цього сервісу можна створювати різні перевірки - checks - за допомогою яких контролюються найважливіші показники роботи ОС.

Користувачі знаходять достатньо причин зі згаданими вище визначальними функціями. Тим не менш, є ще деякі переваги, які пропонує EC2:

— На відміну від інших сервісів, EC2 не складний для запуску, тому що він дуже простий та швидкий. Користувачі можуть відвідати Marketplace всередині AMI та вибрати всі попередньо налаштовані програми. Це можна зробити, обравши Launch або через консоль.

— Екземпляри EC2 можна розміщувати в різних місцях залежно від зон доступності та регіонів. Ця функція корисніша, коли відбувається збій в одному

з регіонів. У цей час Програми можуть бути захищені шляхом запуску екземплярів до інших

— Примірники з дуже великим обсягом пам'яті можуть допомогти користувача, якщо він / вона намагається шукати високий послідовний введення-виведення для додатків та щільність зберігання в кожному

— Високі екземпляри введення/виводу допомагають користувачам, яким потрібен довільний та високий доступ до даних із низькою затримкою. Вони більш застосовні для користувачів, які використовують NoSQL та реляційні бази даних.

— Високопродуктивні кластери обчислювальні спроектовані в Amazon EC2 для забезпечення відносно високої пропускної спроможності і виконання певних операцій, пов'язаних з мережею.

2.1.2 AWS Management Console

Одним із способів керування хмарними ресурсами є веб-консоль, де можна увійти в систему та вибрати потрібну послугу. Це може бути найпростішим способом створення ресурсів і керування ним [4]. Нижче наведено знімок екрана, який показує цільову сторінку під час першого входу в Консоль керування AWS.

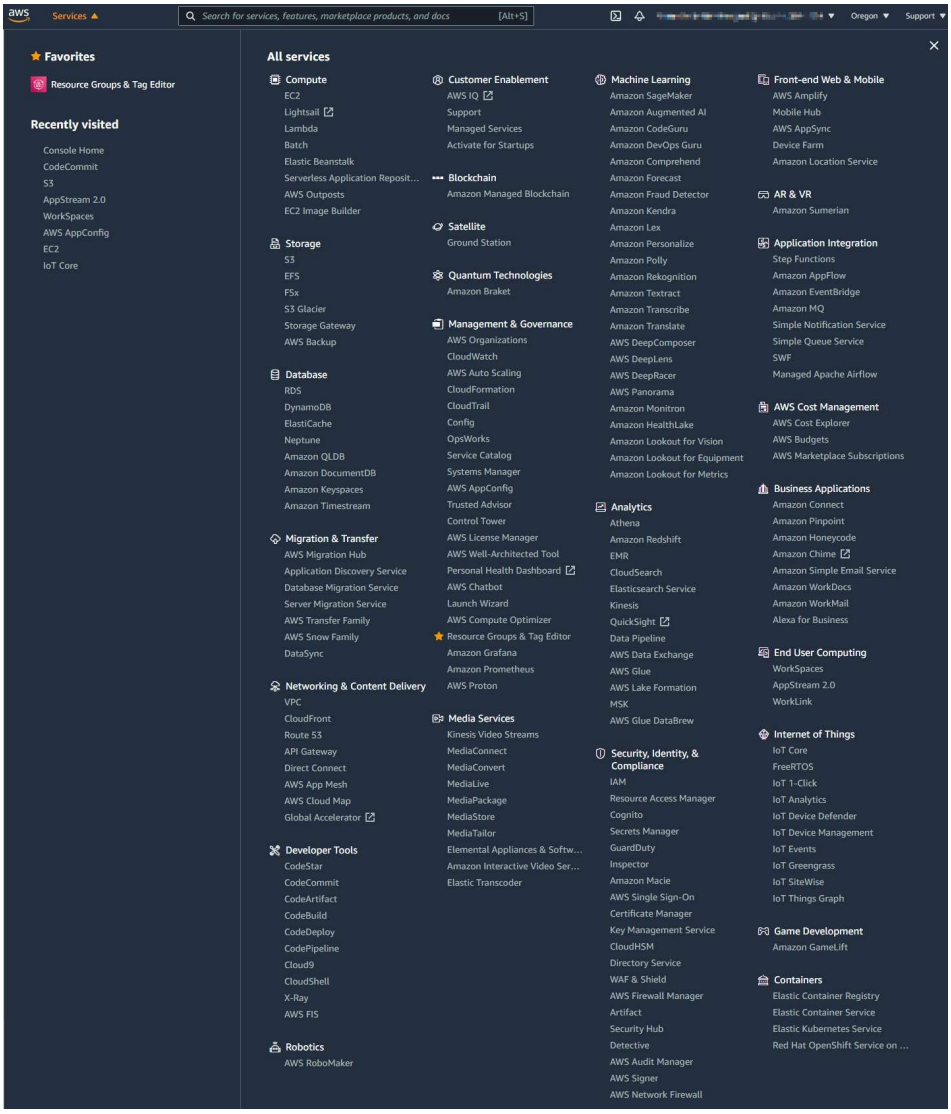

Рисунок 2.2 — Початкова сторінка консолі керування AWS

2.1.3 AWS SDKs

Виклики API до AWS також можна виконувати за допомогою коду з мовами програмування. Це можно зробити за допомогою комплектів розробки програмного забезпечення AWS (SDK). Пакет SDK є відкритим вихідним кодом і підтримується AWS для найпопулярніших мов програмування, таких як C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python і Ruby.

Розробники зазвичай використовують пакети SDK AWS для інтеграції вихідного коду програми зі службами AWS. Наприклад, розглянемо програму з інтерфейсом, що працює на Python. Кожного разу, коли програма отримує фотографію людини, вона завантажує файл у службу зберігання. Цю дію можна

виконати у вихідному коді за допомогою AWS SDK для Python [13]. Ось приклад коду, який можна застосувати для роботи з ресурсами AWS за допомогою пакета SDK Python AWS:

```
import boto3
ec2 = boto3.client('ec2')
response = ec2.describe_instances()
print(response)
```

2.1.4 AWS responsibility

AWS відповідає за безпеку хмари. Це означає, що AWS захищає та захищає інфраструктуру, яка запускає послуги, що пропонуються в AWS Cloud [13]. AWS відповідає за:

— Захист і безпека регіонів AWS, зон доступності та центрів обробки даних, аж до фізичної безпеки будівель

— Керування апаратним, програмним та мережевими компонентами, які запускають послуги AWS, такими як фізичні сервери, операційні системи хоста, рівні віртуалізації та мережеві компоненти AWS

Рівень відповідальності AWS залежить від сервісу. Сервіс класифікує послуги на три категорії. У наступній таблиці наведено інформацію про кожен із них, включаючи відповідальність сервісу AWS.

2.2 Віртуальні сервери Amazon EC2

Першим будівельним блоком, який потрібен для розміщення програми, є сервер. Сервери зазвичай можуть обробляти запити Hypertext Transfer Protocol (HTTP) і надсилати відповіді клієнтам за моделлю клієнт-сервер, хоча будь-яке спілкування на основі API також підпадає під цю модель. Клієнт - це людина або комп'ютер, який надсилає запит [11]. Сервером, який обробляє запити, є

комп'ютер або набір комп'ютерів, підключених до Інтернету, що обслуговує веб-сайти для користувачів Інтернету.

Сервери забезпечують роботу вашої програми, забезпечуючи процесор, пам’ять і мережеві потужності для обробки запитів користувачів і перетворення їх у відповіді. Для контексту загальні сервери HTTP включають:

— Параметри Windows, такі як Internet Information Services (IIS)

— Параметри Linux, такі як веб-сервер Apache HTTP, Nginx і Apache Tomcat

Щоб запустити HTTP-сервер на AWS, потрібно знайти службу, яка забезпечує обчислювальну потужність, у Консолі керування AWS. Можемо увійти на консоль і переглянути повний список обчислювальних служб AWS.

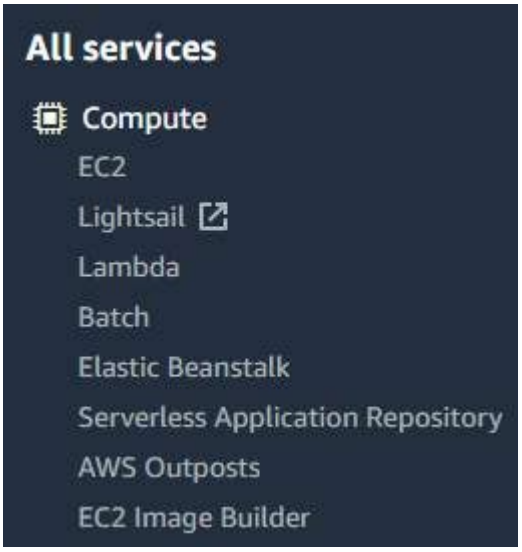

Рисунок 2.3 — Список обчислювальних служб AWS

У AWS віртуальні машини називаються Amazon Elastic Compute Cloud або Amazon EC2. За лаштунками AWS керує і керує головними машинами та рівнем гіпервізора. AWS також встановлює операційну систему віртуальної машини, яка називається гостьовою операційною системою.

Деякі обчислювальні служби AWS використовують Amazon EC2 або використовують концепції віртуалізації під капотом.

Amazon EC2 - це веб-сервіс, який забезпечує безпечні обчислювальні потужності в хмарі зі змінним розміром. Це дозволяє надавати віртуальні сервери, які називаються екземплярами EC2. Хоча AWS використовує фразу «веб-сервіс» для її опису, ми не обмежуємось запуском лише веб-серверів на своїх екземплярах EC2.

Можна створювати екземпляри EC2 і керувати ними за допомогою Консолі керування AWS, інтерфейсу командного рядка AWS (CLI), комплектів розробки програмного забезпечення AWS (SDK), інструментів автоматизації та служб оркестрування інфраструктури.

Щоб створити екземпляр EC2, потрібно визначити наступне:

— Специфікації обладнання, як-от процесор, пам’ять, мережа та сховище

— Логічні конфігурації, як-от розташування мережі, правила брандмауера, аутентифікація та операційна система на вибір.

Під час запуску екземпляра EC2 першим параметром при налаштуванні є операційна система, вибравши зображення машини Amazon (AMI).

У традиційному світі інфраструктури процес розгортання сервера складається з встановлення операційної системи з інсталяційних дисків або майстрів встановлення через мережу. У AWS Cloud установка операційної системи не є вашою відповідальністю. Замість цього він вбудований в AMI.

Крім того, при використанні AMI, можна вибрати відображення пам’яті, тип архітектури (наприклад, 32-розрядний, 64-розрядний або 64-розрядний ARM) та встановлене додаткове програмне забезпечення.

2.2.1 Зв'язок між екземплярами AMI та EC2

Екземпляри EC2 - це живі екземпляри того, що визначено в AMI. Можна провести аналогію з об’єктно-орієнтованими мовами програмування (ООП). Екземпляр EC2 в даному порівнянні виступає як об’єкт класу.

Клас - це те, що ви моделюєте та визначаєте, тоді як об’єкт - це те, з чим ви взаємодієте. У цьому випадку AMI - це те, як ви моделюєте та визначаєте свій

екземпляр, тоді як екземпляр EC2 - це об’єкт, з яким ви взаємодієте, де ви можете встановити свій веб-сервер і подавати свій вміст користувачам.

При старті нового екземпляру AWS виділяє віртуальну машину, яка працює на гіпервізорі. Потім вибраний вами AMI копіюється на кореневий том пристрою, який містить образ, який використовується для завантаження тома. Зрештою, отримуємо сервер, до якого можна підключатися та встановлювати пакети та додаткове програмне забезпечення.

2.2.2 Життєвий цикл екземпляра EC2

Примірник EC2 переходить між різними станами від моменту його створення до завершення.

Коли запускається екземпляр, він переходить у стан очікування. Коли екземпляр очікує на розгляд, виставлення рахунків не розпочато. На цьому етапі екземпляр готується до переходу в робочий стан. У очікуванні AWS виконує всі дії, необхідні для налаштування екземпляра, наприклад копіює вміст AMI на кореневий пристрій і розподіляє необхідні мережеві компоненти [13].

Коли екземпляр запущено, він готовий до використання. Це також етап, на якому починається виставлення рахунків. Щойно екземпляр запущено, ми можемо виконувати інші дії з екземпляром, наприклад перезавантажити, завершити, зупинити та зупинити сплячий режим.

При перезавантаженні екземпляру, процесс відрізняється від виконання дії зупинки, а потім дії запуску. Перезавантаження екземпляра еквівалентно перезавантаженню операційної системи. Примірник залишається на тому ж комп’ютері і зберігає свою публічну та приватну IP-адресу, а також будь-які дані в сховищі екземплярів.

Зазвичай перезавантаження триває кілька хвилин. При зупинці та запускі екземпляру, даний екземпляр може бути розміщений на новому базовому фізичному сервері. Таким чином, ми втрачаємо всі дані зі сховища екземплярів,

які були на попередньому хост-комп'ютері. При зупинці екземпляру, він отримує нову публічну IP-адресу, але зберігає ту саму приватну IP-адресу.

При закриванні екземпляру, сховища екземплярів стираються, і втрачається як загальнодоступна IP-адреса, так і приватна IP-адреса машини. Припинення екземпляра означає, що ви більше не можете отримати доступ до машини.

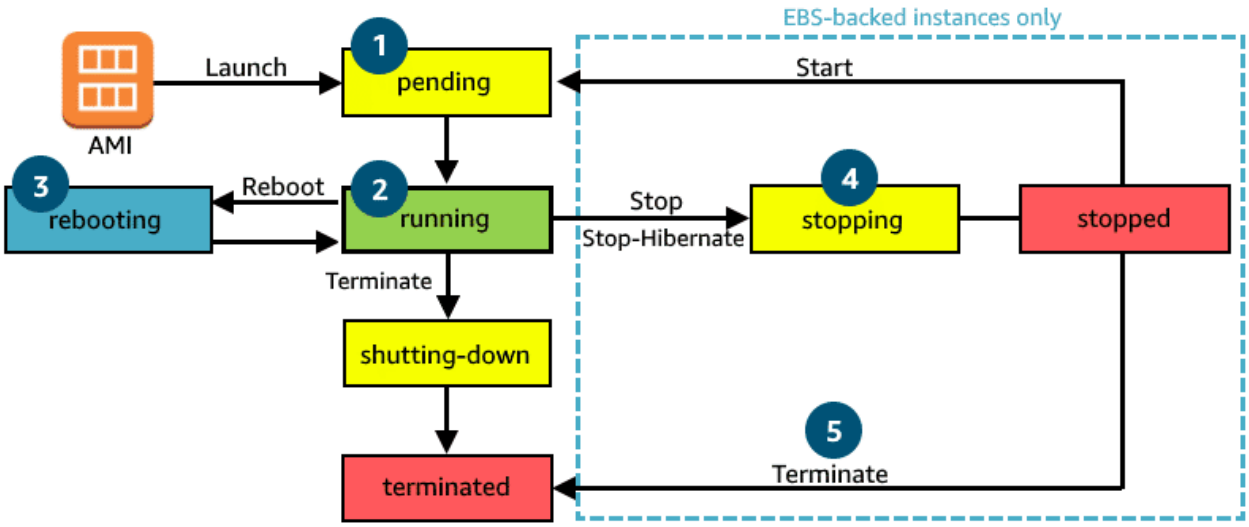

Рисунок 2.4 — Життєвий цикл екземпляра EC2

2.3 Протокол MQTT

MQTT - це стандартний протокол обміну повідомленнями OASIS для Інтернету речей (IoT) [6]. Він розроблений як надзвичайно легкий транспорт для обміну повідомленнями публікації/підписки, який ідеально підходить для підключення віддалених пристроїв з невеликим обсягом коду та мінімальною пропускною спроможністю мережі. Сьогодні MQTT використовується в різних галузях, таких як автомобілебудування, виробництво, телекомунікації, нафта і газ і т. д.

Чому б замість цього не використовувати веб-перехоплювач? Оскільки MQTT широко застосовується в індустрії Інтернету речей поскільки це дуже легкий протокол, і завдяки мінімальним витратам накладних пакетів, MQTT перевершує, наприклад, передачу даних по дроту, в порівнянні з такими протоколами, як HTTP [6]. Інший важливий аспект протоколу полягає в тому, що

MQTT дуже легко реалізувати на стороні клієнта. Простота використання була ключовим завданням розробки MQTT і робить її ідеальною для сьогоднішніх обмежених пристроїв з обмеженими ресурсами.

Програми MQTT використовують архітектуру видавця / передплатника, і наша також. Ось приклад цієї архітектури:

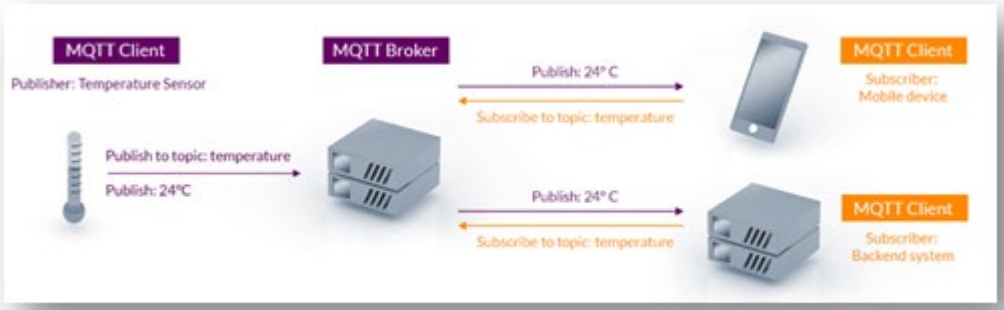

Рисунок 2.5 — Приклад архітектури MQTT

MQTT протокол складається з MQTT-брокера, MQTT-агентів передплатників та виконавців. Усі вони чітко знають та виконують запрограмоване завдання, працюючи чітко та злагоджено. Виконавці займаються публікацією даних, призначених для передплатників. Це їхня основна функція, без якої з'єднання не працюватиме.

Обчислювальні потреби для протоколу MQTT дуже малі, тому що він розрахований на вмонтовані пристрої з низькою потужністю. Навіть якщо в мережах низька пропускна можливість, MQTT зберігає високу якість зв'язку та практично не перевантажує роботу системи. Це один із основних плюсів цього протоколу. У структурі даних, що передає протокол, майже немає функціональної інформації, порівняно з іншими протоколами зв'язку. Що характеризує его з якісного боку. Наприклад, HTTP передає всі службові дані, але в цьому немає жодної термінової необхідності.

Зробивши вимірювання в 3G-мережах і детально перевіривши всі процеси, стало відомо, що MQTT має в 93 рази більшу пропускну можливість, ніж REST (Representational State Transfer), який виконує свою роботу поверх HTTP.
Приклади можливостей MQTT:

— connect (встановлення зв'язку з брокером)
— disconnect (переривання зв'язку з брокером)
— subscribe (підписка на тему на брокері)
— unsubscribe (відписка від теми на брокері)
— publish (опублікування своєї теми на брокері)

Джерелом інформації є публікатор, завдання якого з'єднається з брокером, а потім якісно і швидко передати необхідні дані. Потім передплатник, який є споживачем, працює за аналогічною схемою та здійснює підписку на тему, якою ми тут можемо побачити “/home/alarms/1/status”. Приклад цієї теми здійснюється публікація даних про становище домашньої сигналізації в деякій “зоні 1”. Якщо до видавця надходить нова інформація він передає її брокеру та здійснює публікацію повідомлення на цю тему. А завдання брокера, розповсюдити повідомлення всім передплатникам цієї теми.

Структура теми наведеної нами є ієрархічною. Цим вона має схожість з шляхом файлової системи, і можливість спростити організацію об'єктів. А ще такі ієрархічні структури дуже популярні і популярні також і в інших протоколах, наприклад в REST.

Програмне забезпечення MQTT можна встановити автономно на сервері або як частина Node-RED. Ми будемо використовувати MQTT в Node-RED, його легко налаштувати.

2.4 MV Camera

Сімейство розумних камер MV несе в собі простоту та розум до світу камер відеоспостереження. Кожна модель MV поставляється з потужним процесором — таким же, як і в багатьох сучасних смартфонах — та інноваційна архітектура,

яка мінімізує фізичну інфраструктуру, а також вимоги до програмного забезпечення [2].

Ці розумні камери не лише допомагають забезпечити фізичну безпеку, але також забезпечують передові бізнес-аналітики. MV містить швидку обчислювальну потужність, надійні функції безпеки та складну аналітику в надзвичайно простому пакеті. В даному проекті було обрано модель MV72.

Серія Meraki MV72 - частина сімейства розумних камер MV, об’єднує фізичну безпеку та розширену аналітику в компактному форм-факторі. Завдяки потужному апаратному забезпеченню та архітектурній простоті сімейства MV, розумні камери MV72 створені не лише для фізичної безпеки та безпеки, але й для бізнес-аналітики.

Архітектура розумної камери MV розміщує високоякісне сховище безпосередньо на камері, усуваючи потребу в мережевому відеореєстраторі (NVR). Це не тільки різко спрощує встановлення та масштабування, але й усуває серйозну вразливість мережевої безпеки в IT-інфраструктурі. Такі функції, як аналіз LLDP, сповіщення про автономні пристрої та вбудовані віддалені інструменти, скорочують час на усунення несправностей, звільняючи IT-ресурси. А оскільки MV керується за допомогою інформаційної панелі Meraki на основі браузера та працює за моделлю ліцензування, немає необхідності купувати, завантажувати та підтримувати будь-яке додаткове програмне забезпечення. Приладова панель Meraki забезпечує оновлення мікропрограми, а нові функції будуть постійно впроваджуватися протягом терміну служби продукту, збільшуючи загальну цінність [2].

Переваги MV72:

— Фіксований об’єктив, міні-купольна камера з компактним форм-фактором

— Можливість машинного навчання камери для інтелектуального виявлення об'єктів

— Не потрібно спеціального програмного забезпечення або плагінів для браузера

— Хмарне доповнене граничне сховище мінімізує фізичну інфраструктуру

— Інструментальна панель Meraki спрощує роботу

— Безпечна, зашифрована архітектура керування та зберігання

— Детальний контроль доступу користувачів

— Інтегрований бездротовий зв’язок 802.11ac

— Локальний або віддалений потік — хмара автоматично направляє відеопотік

Основні характеристики MV72:

— Вбудований твердотільний накопичувач ємністю 256 ГБ високої надійності

— Запис відео 1080p із кодуванням Smart Codec

— 73° горизонтальне поле зору

— 802.11ac бездротовий, 1x 10/100/1000 Base-T Ethernet (RJ45)

— Мікрофон для запису звуку

— Температурний режим 0°C - 40°C (32°F - 104°F)

— Розміри 106mm x 74mm (діаметр x висота), вага 286g

2.5 Інструмент Node-Red

Node-Red – інструмент програмування на основі потоків. По суті це Low-code програмування для подійно-керованих додатків [10]. Це дозволяє визначати поведінку програми як мережу чорних ящиків, або «вузлів», як їх називають Node-Red. Кожен вузол має чітко визначену мету - йому надаються деякі дані, він щось робить із цими даними, та передає ці дані. Мережа відповідає за потік даних між вузлами.

Це модель, яка дуже добре піддається візуальному уявленню і робить її доступнішою для ширшого кола користувачів. Якщо хтось може розбити

проблему на окремі кроки, він може подивитися на потік та зрозуміти, що він робить.

Node-Red складається з середовища виконання на основі Node.js, на яке ви вказуєте веб-браузер для доступу до редактора потоку. У браузері створюємо програму, перетягуючи вузли з палітри в робочу область та починаєте зв'язувати їх разом. Одним клацанням миші програма розгортається назад у середу виконання, де вона була запущена.

Палітру вузлів можна легко розширити, встановивши нові вузли, створені спільнотою, а потоки, які ми створюємо, можна легко передати у вигляді файлів JSON.

Приклад графічного інтерфейсу Node-Red після його налаштування:

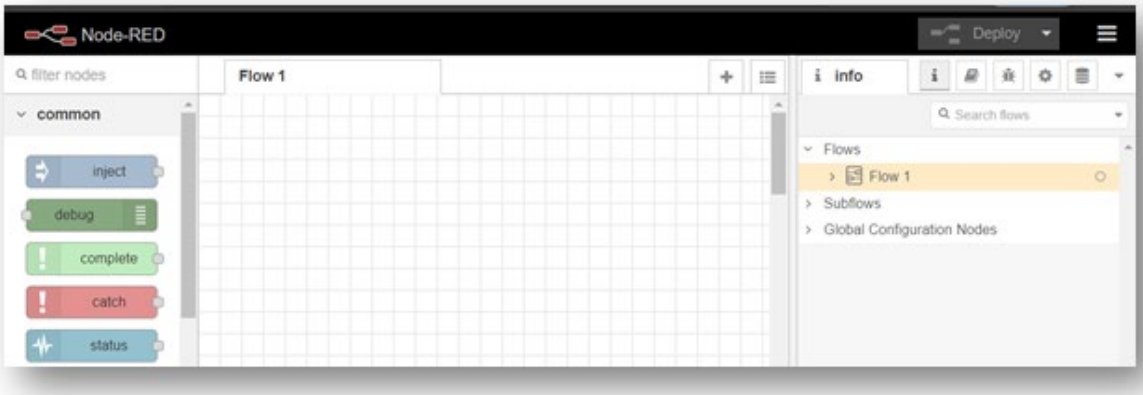

Рисунок 2.6 — Адміністративний інтерфейс Node-RED

Node-Red можна налаштувати, розмістити та запустити на кількох сучасних платформах, включаючи Raspberry Pi, Azure, AWS, Docker, Android, Windows та інші. Ми будемо використовувати його в Linux [10].

2.6 AWS Rekognition

AWS Rekognition (Amazon Rekognition) - спрощує додавання аналізу зображень та відео до ваших програм за допомогою перевіреної, добре

масштабованої технології глибокого навчання, що не потребує досвіду машинного навчання [13].

За допомогою Amazon Rekognition ми можемо ідентифікувати:

— Об'єкти

— Людей

— Текст

— Сцени та дії із зображеннями та відео, а також виявляти неприйнятний контент.

Рисунок 2.7 Приклад розпізнавання об’єктів за допомогою Amazon Rekognition

За допомогою міток Amazon Rekognition, що настроюються, ми можемо ідентифікувати об'єкти та сцени на зображеннях, які відповідають потребам бізнесу. Завдяки даній технології ми можемо виявити серед об’єктів людей, що сильно допоможе нам для подальшого збору та обробки інформації. Amazon Rekognition Custom Labels бере на себе важку роботу з розробки моделі, тому досвід машинного навчання не потрібний. Просто потрібно надати зображення об'єктів або сцен, які хочемо ідентифікувати, а решту зробить сервіс.

Рисунок 2.8 — Розпізнавання характеристик людини на знімку за допомогою Amazon Rekognition

3. СИНТЕЗ КОМП'ЮРЕНОЇ СИСТЕМИ

3.1 Загальні системні вимоги

3.1.1 Вимоги до структури та функціональності системи

АСКН призначена для підрахунку людей на території об'єкту та обчислення дистанції між ними.

АСКН потребує впровадження наступних функцій:

— віртуалізація мережі за допомогою апаратного і програмного забезпечення на платформі Meraki;
— швидке розгортання з оптимізацією апаратних ресурсів;
— можливість виявлення фізичного місця розташування користувачів;
— можливість обчислення дистанцій між клієнтами шляхом виконання Python скрипта;
— можливість відправки повідомлень на електронну адресу за допомогою Python скрипта;
— отримання даних про клієнтів клубу в текстовому форматі шляхом розпізнання технологією AWS Rekognition і її зв'язку з розробленим додатком, Node-Red та віртуальним сервером Amazon EC2;
— правильне розпізнавання людей від інших об'єктів в кадрі технологією AWS Rekognition.

3.1.2 Вимоги до складу технічних засобів системи

АКСН побудована на продуктах Cisco з управлінням з хмарної платформи Meraki.

Вимоги до технічних засобів системи:

— самоконфігурація, розгортання за принципом «включай і працюй»;
— стандартний протокол обміну повідомленнями OASIS – MQTT
— адміністрування на основі ролей і автоматичні, заплановані оновлення прошивки, що надаються через Інтернет;
— привабливий дизайн і компактні розміри.

— попередження на електронну пошту чи текстові повідомлення про збої живлення, простоях або зміни конфігурації;
— високошвидкісний обмін даними між розеткою підключення обладнання і/або серверами за технологією Ethernet (Fast Ethernet, Gigabit Ethernet) та Wi-Fi 802.11n частотою 2,4 та 5 ГГц;
— ефективна робота всіх складових елементів локальної обчислювальної мережі ЛОМ (АРМ, серверів, принтерів і т.п.).

Технічне забезпечення системи повинно максимально і найбільш ефективним чином використовувати існуючі технічні засоби.

3.1.3 Вимоги до надійності

— безперервна робота серверного обладнання;
— в разі відключення пристроїв від електроенергії мережа повинна працювати напротязі 30 хв;
— повинні використовуватись технічні засоби, що мають середній час відпрацювання не менще ніж 25000 годин;
— всі можливі дані повинні розміщуватись на основному сервері;
— мати допоміжний диск для резервного копіювання даних;
— мінімальна степінь захисту IP64

3.1.4 Вимоги безпеки

Для персоналу, що обслуговує електрообладнання системи, обов'язковими до виконання є «Правила технічної експлуатації та безпеки обслуговування електротехнічних засобів». Особи, які обслуговують електроприлади, зобов'язані пройти навчання безпечним методам роботи та перевірку знань кваліфікаційною комісією.

Співробітники повинні дотримуватися «Правил охорони праці під час експлуатації електронно-обчислювальних комп'ютерів».

3.1.5 Вимоги до ергономіки та технічної естетики

Головний комп’ютер для доступу до серверу повинен знаходитись в окремому приміщенні. Робоча зона повинна складатись з комп’ютерного стола та кріслом з можливістю регуляції. Робоче місце повинно мати доступ до інформаційної розетки RJ-45 в кількості 2 штуки.

Зовнішній вигляд елементів приміщення повинен гармоніювати з інтер'єром робочих місць.

3.1.6 Вимоги до експлуатації, обслуговування, ремонту та консервації

Повинно надаватись резервне джерело безперебійного живлення для моментального підключення до ЕОМ в разі аварії.

Вимоги для співробітників, обслуговуючих електротехнічних засобів:

— візуальний огляд цілісності обладнання не менше 1 разу на зміну;
— ознайомлення з протоколом регулярних і надзвичайних;
— діагностика обладнання автоматизованої системи з метою постійного виявлення несправностей під час робочої зміни;
— планові ремонтні роботи для запобігання зносу обладнання не менше 1 разу на квартал;
— планове обслуговування системи не рідше 1 разу на місяць.

Під час обслуговування системи контролюється наступне:

— надійність провідникових кріплень, системних блоків;
— цілісність роз'ємів, вилк, джерел живлення;

3.1.7 Вимоги до захисту інформації від несанкціонованого доступу

— забезпечити можливість безпечного та захищеного віддаленого адміністрування через мережу Інтернет;
— забезпечити обмеження доступу до серверного обладнання з корпоративної мережі та мереж загального користування;

— забезпечити мережевий захист від вірусних атак, DDoS;

— забезпечення надійним паролем головного комп’ютера;

— зміна паролю доступу не рідше ніж 1 раз на місяць.

3.1.8 Вимоги до схоронності інформації при аваріях

Забезпечити резервне копіювання даних, які підлягають довготривалому зберіганню.

3.1.9 Вимоги до захисту від впливу зовнішніх чинників

Корпуси технічних засобів повинні мати наступні характеристики:

— степінь захисту не менше ніж IP64;

— всі кабельні з’єднання повинні знаходитися в кабель каналах для захисту від зовнішнього впливу;

— стійкість до кліматичних впливів навколишнього середовища при робочих температурах повітря від -20 до +85 ° С;

— стійкість до механічних зовнішніх факторів;

— умови експлуатації при верхньому значенні відносної вологості при 25 ° С, до 100%;

— висота до 1000 м.

3.1.10 Вимоги до стандартизації

Усе активне обладнання повинно мати сертифікат відповідності. Для створення локальної обчислювальної мережі необхідно використовувати тільки високоякісні компоненти, які пройшли стовідсоткове тестування відповідно до вимог ISO 9001 (ГОСТ 40.9001-88).

Для реалізації автоматизованої системи повинен використовуватись персональний комп’ютер з наступними характеристиками:

— об’єм оперативної пам’яті – не нижче 4 Гбайт;

— частота процесора - не нижче 2.2 ГГц;

— обсяг жорсткого диску – не менше 500 Гбайт;

— операційна система – від Windows 7;

— для користування сервером потрібен будь-який браузер на основі движка Chromium – для роботи з віртуальним сервером Meraki;

— відеокарта Nvidia – не нижче моделі 610;

— програмний драйвер Nvidia Cuda для більшої обчислювальної спроможності.

3.2 Вимоги до функцій системи

3.2.1 Вимоги до функцій, які виконує КС

Побудова єдиного інформаційного середовища фітнес-центру для забезпечення наступних функцій:

— безперервне підключення камер Meraki MV до точок доступу;

— підписка камер Meraki MV на брокери протоколу MQTT;

— обмін даними між MQTT брокерами та додатком Node-Red;

— збір інформації про місцезнаходження відвідувачів в режимі реального часу;

— обмін даними між Node-Red та Meraki Snapshot API;

— можливість залежності Python скрипта з утилітою AWS Recognition та MQTT брокером;

управління мережним обладнанням з хмарної платформи Meraki.

3.2.2 Вимоги до інформаційного забезпечення

— Інформаційний обмін між компонентами КС повинен здійснюватися за допомогою захищених протоколів MQTT.

— Доступність інформаційних ресурсів між компонентами підсистем повинна забезпечуватися за рахунок надійного доступу до Інтернет.

— Програмне супроводження та обслуговування мережевого варіанту програмного комплексу через хмару в веб-браузері або API-інтерфейси програмування.

3.3 Вибір технологій та технічних пристроїв для системи керування

Основні технології та технічні пристрої для впровадження автоматизованої системи контролю наповненості фітнес клубу, далі АСКН:

— Meraki Dashboard;

— Cisco MV72 Camera;

— Node-Red;

— MQTT протокол та брокери;

— AWS Recognition;

— додаток написаний на мові Python;

— REST API;

— точки доступу Cisco Meraki MR34;

— комутатори доступу Cisco MS320;

— комутатори рівня розподілу Cisco MS420;

— міжмережний екран Cisco Meraki MX400.

3.4 Обґрунтування вибору технологій

Система призначена для:

— відстеження переміщень відвідувачів по території фітнес клубу;

— підрахунку кількості відвідувачів в певних зонах території;

— виявлення дистанції між відвідувачами;

— підрахунку кількості порушень дистанційних норм між відвідувачами;

— оповіщення про перевищення максимальної кількості дистанційних порушень між відвідувачами способом відправки попереджувального листа на електронну пошту адміністратора фітнес клубу;

— аналізу та описання характеристик розпізнаної людини (настрій, вік, стать, тощо) в вигляді текстового представлення.

Більшість сучасних фітнес-клубів використовують CRM системи для контролю доступу. В даних систем є як свої плюси, так і мінуси. Задача даної роботи стоїть в тому, що автоматизована система повинна розпізнавати людей через камери відеоспостереження, мати функцію підраховування розпізнаних людей, та обчислювання дистанції між ними. Такого функціонала немає в жодній з CRM систем.

Згідно з вимог розроблене рішення повинно бути впроваджено на платформі Meraki. Meraki – це 100% хмарне рішення для побудови IT-інфраструктури невеликої або середньої компанії, яким характерно швидке розгортання, спрощене адміністрування і багатофункціональність. Оскільки була вибрана архітектура Meraki – це означає, що практично вся система буде розміщена в хмарі.

Для організації повноцінного Wi-Fi рішення слід встановити точки доступу Meraki і під'єднати до центральної cloud-системи управління з використанням надійних Інтернет каналів зв'язку. Cloud – це рішення, яке не вимагає наявності апаратного контролера точок доступу і зменшує початкові витрати на розгортання системи.

Панель управління Meraki (Meraki Dashboard) являється частиною архітектури хмарного управління Cisco Meraki, яка дозволяє візуалізувати мережеву інфраструктуру з демонстрацією даних про всі підключені пристрої, статистики мережевого трафіку і моніторингу стану мережі. За допомогою панелі керування адміністратори можуть додавати нові пристрої і автоматизувати такі рутинні завдання, як знаходження і установку останніх версій прошивок. Всі пристрої Cisco Meraki підключаються до хмари в автоматичному режимі, використовуючи протокол SSL.

Користувачі можуть розгортати, контролювати і налаштовувати свої пристрої Meraki через веб-інтерфейс панелі інструментів Meraki або через API.

Як тільки користувач вносить зміну в конфігурацію, запит на зміну відправляється в хмару Meraki, а потім вирушає з хмари Meraki на відповідний пристрій (пристрої).

Керуючі дані (наприклад, конфігурація, статистика, моніторинг і т. д.) передаються з пристроїв Meraki (точок бездротового доступу, комутатори та пристроїв безпеки) в хмару Meraki по захищеному інтернет-з'єднання.

Данні користувача (перегляд веб-сторінок, внутрішні програми і т. д). не передаються через хмару Meraki, а прямують безпосередньо до місця призначення в локальній мережі або через глобальну мережу.

Пристрої для зв'язку з хмарою Meraki використовують запатентований легкий зашифрований тунель, який використовує шифрування AES256. У самому тунелі Meraki використовує HTTPS і протокольні буфери для безпечного і ефективного вирішення, обмеженого 1 кбіт/с на пристрій, коли пристрій активно не керується.

Описана структура задовольняє вимогам до синтезованої системи, тож можна приступати до розробки схеми функціональної структури.

3.5 Розробка схеми функціональної структури

АКСН повинна виконувати наступні функції:

— Камери MV повинні підключатися до мережі інтернет за допомогою точок дступу;

— камера MV повинна надавати потік MQTT брокеру MQTT;

— кожні кілька секунд скрипт Python повинен запитувати URL-адресу знімка з камери MV;

— хмара Meraki повертатиме URL-адресу до сценарію Python;

— URL-адреса пересилатиметься до служби AWS Rekognition;

— AWS Rekognition повинна завантажувати знімок за допомогою URL-адреси та виконувати на знімку розпізнавання об’єктів і обличчя на основі машинного навчання;

— розпізнавання AWS повертатиме результат аналізу об’єкта та обличчя до сценарію Python;

— результати повинні пересилатися в MQTT брокер за допомогою MQTT разом з URL-адресою моментального знімка;

— брокер MQTT пересилатиме це (разом з попередніми деталями) до графічного інтерфейсу користувача Node-Red;

— інформація повинна передаватись до Node-Red, включаючи URL-адресу знімка зображення.

Судячи зі списку схема функціональної структури матиме вигляд, який показано на рисунку 3.1.

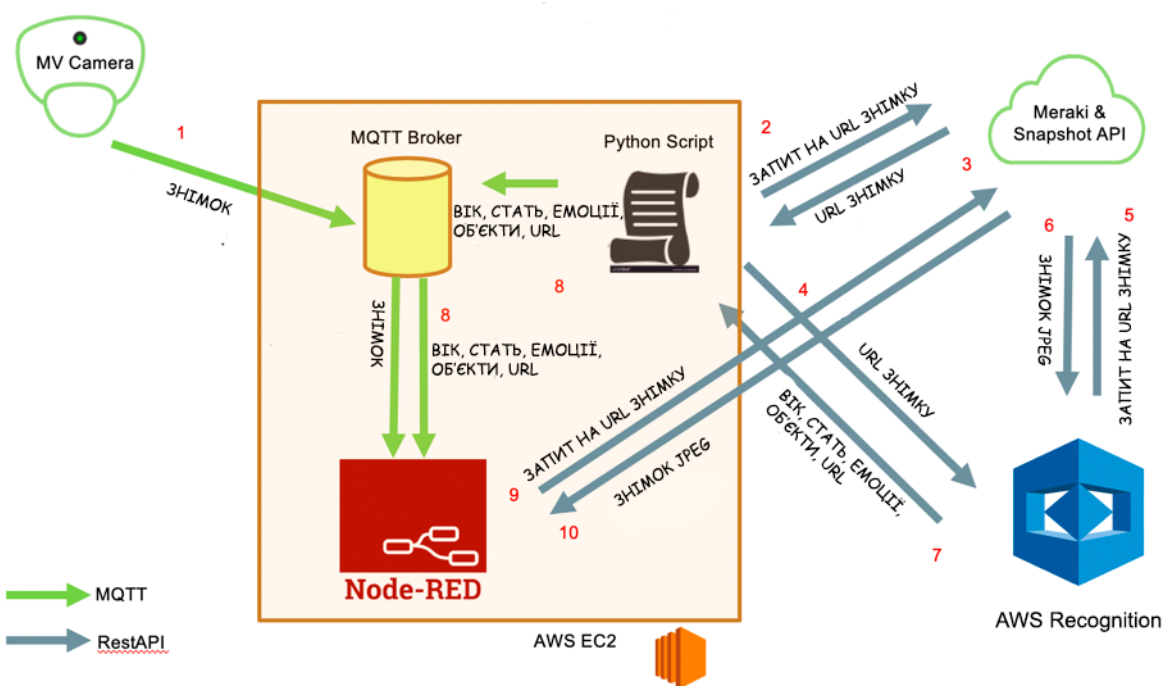

Рисунок 3.1 — Схема функціональної структури

3.6 Розробка функціональної схеми автоматизації

Для точного побудування функціональної схеми автоматизації АКСН потрібен план будівлі спорт клубу «Спортлайф». План зображений на рисунку 3.2.

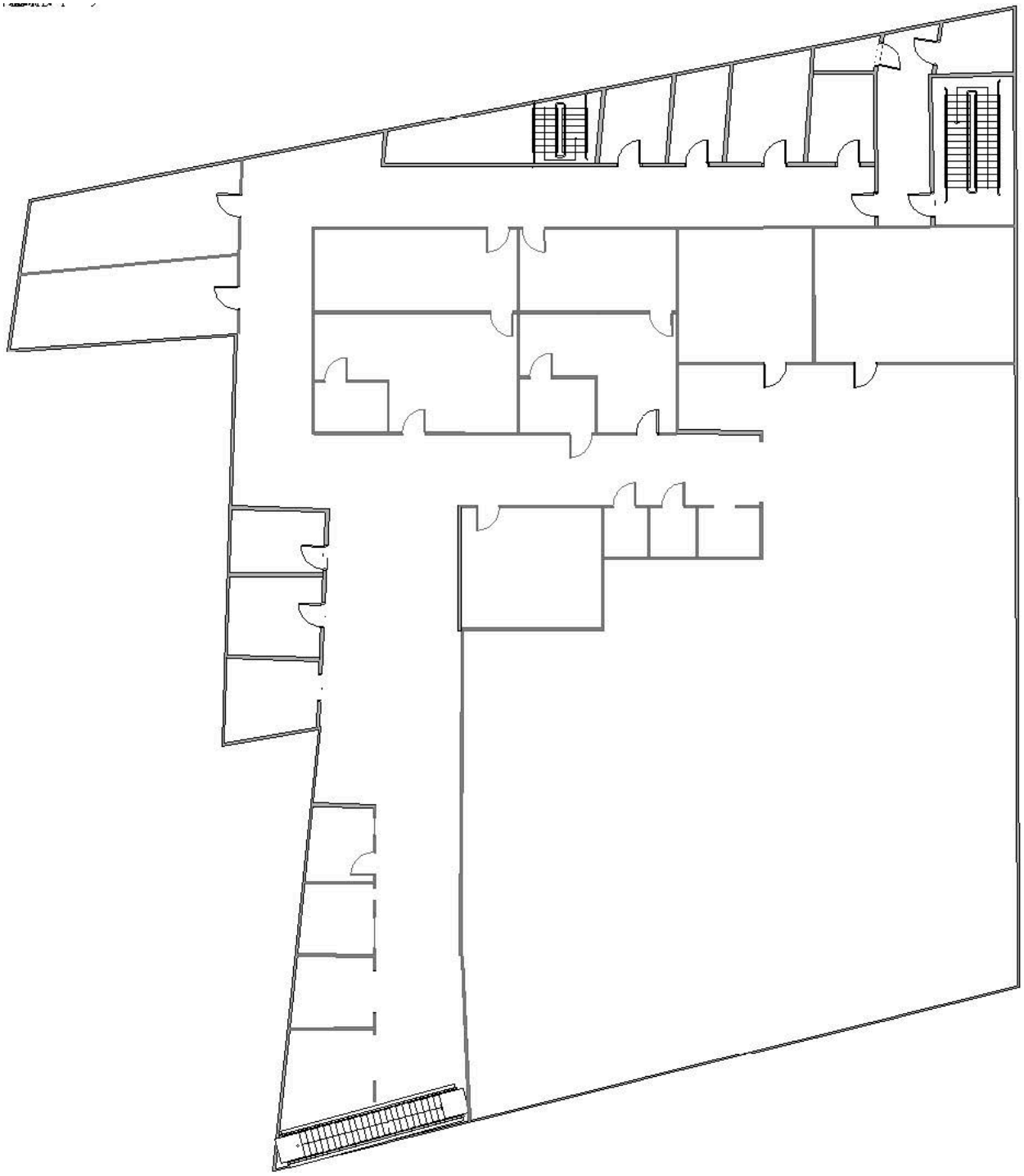

Рисунок 3.2 — План будівлі спорт клубу «Спортлайф»

Згідно з вимог та переліку обраних пристроїв була побудована функціональна схема автоматизації, яка зображена на рисунку 3.3.

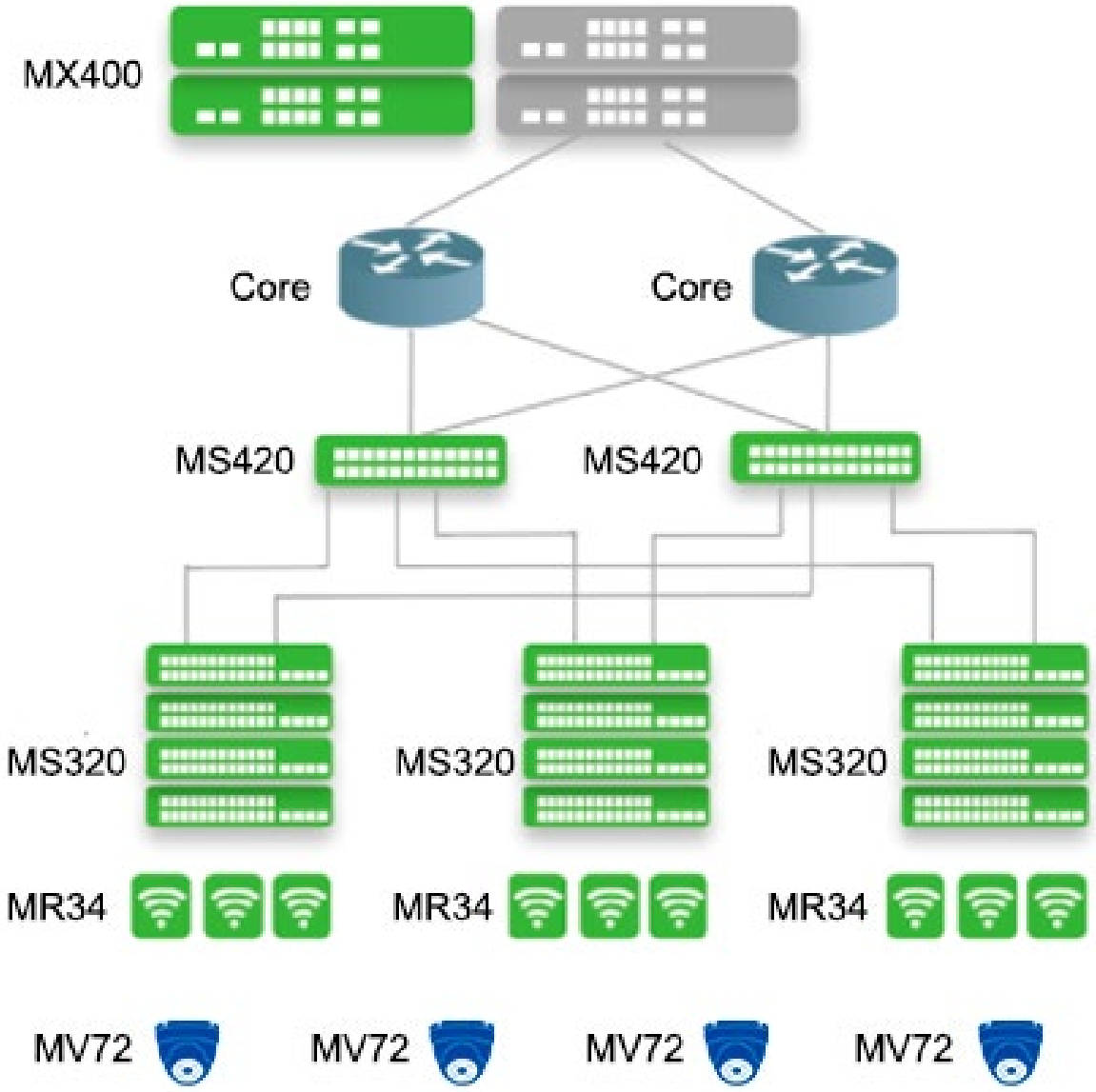

Рисунок 3.3 — Функціональна схема автоматизації

На основі вимог, схеми функціональної структури та схеми будівлі фітнес клубу «Спортлайф» побудована схема розміщення апаратних пристроїв на території фітнес клубу «Спортлайф» (рисунок 3.4).

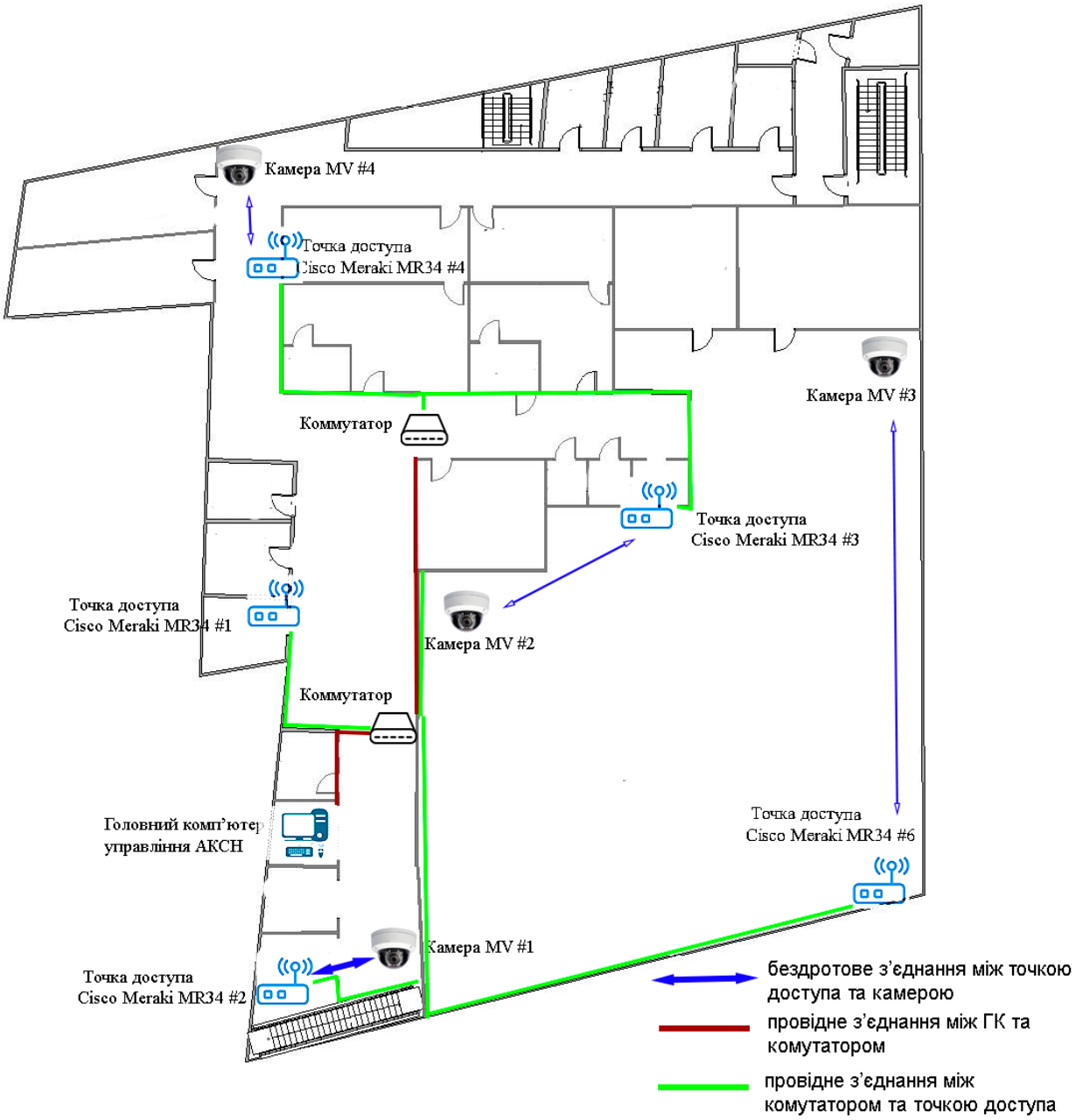

Рисунок 3.4 — Схема розміщення апаратних пристроїв на території фітнес клубу «Спортлайф»

3.7 Аналіз входів та виходів АСКН

Для визначення входів та виходів АКСН потрібно провести аналіз і класифікацію входів та виходів датчиків та виконуючих пристроїв технічного

обладнання. Для отримання точних даних потрібно дотримуватись певних вимог для обчислень. В результаті аналізу отримана класифікація входів, яка зображена в таблиці 3.1 та таблиці 3.2.

Вид активності	**Задачі та можливості**	**% від висоти кадру**	**Альтернативний параметр, мм/1пкс**	**Кількість пікселей на 1м по горизонталі**
Моніторинг	Моніторинг та контроль потоку людей	5%	80	12
Виявлення	Гарантоване виявлення людей в кадрі	10%	40	25
Спостереження	Виявлення характерних якостей людини (настрій, вік, ріст)	25%	16	62
Розпізнавання	Розпізнавання людей серед інших об’єктів	50%	8	125
Ідентифікація	Достатня якість для виявлення людини	100%	4	250
Інспектування	Точна ідентифікація людини	400%	1	1000

Таблиця 3.1 — Перелік вхідних та вихідних даних для програмного додатку

Фокусна відстань (мм)	**Дистанція ідентифікації(м)**	**Дистанція розпізнавання(м)**	**Дистанція виявлення об’єкту(м)**
2,8	**1,86**	**4,66**	**23,33**
3,3	**2,20**	**5,50**	**27,50**
3,6	**2,40**	**6,00**	**30,00**
4,2	**2,80**	**7,00**	**35,00**
6	**4,00**	**10,00**	**50,00**
8	**5,33**	**13,33**	**66,66**
9	**6,00**	**15,00**	**75,00**

Таблиця 3.2 — Перелік вхідних та вихідних даних дистанцій для програмного додатку

3.8 Висновки до розділу

На основі обзору процесу роботи автоматизованої комп’ютерної системи контролю наповненості були построєні функціональні схеми системи управління КС: схема функціональної структури та схема автоматизації. Також були побудовані таблиці вхідних та вихідних даних на основі аналізу функціональної структури. Отримані дані в ході синтезу системи дозволяють перейти до розробки програмного забезпечення.

4. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Призначення й сфера застосування програми

Суть задачі стоїть у відстеженні переміщень по території фітнес-клубу, та обчислення дистанції між людьми. Дана програма буде виконувати частину, яка необхідна для обчислень дистанційних норм, характеристик об'єктів в кадрі, правильне розпізнання людей серед інших об'єктів, а також надсилання попереджень про порушення дистанційних норм.

Сферою застосування є будь-яка компанія, яка зацікавлена в вище описаних функціях, тобто це доволі універсальний додаток.

4.2 Обґрунтування технічних характеристик програми

4.2.1 Постановка завдання на розробку програми

Програма повинна виконувати такі функції:

— підключення до серверу;

— взаємодія з сервером для отримання та відправки інформації;

— виявляння об'єктів в кадрі;

— виявлення людей серед інших об'єктів в кадрі;

— обчислення дистанцій між людьми;

— відправка повідомлень на електронну пошту;

4.2.2 Опис алгоритму програми з обґрунтуванням вибору схеми алгоритму

Для розробки програмного додатку потрібна схема алгоритму, котра містить в собі опис взаємодії програми з іншим програмним забезпеченням.

Схема алгоритму наведена на рисунку 4.1.

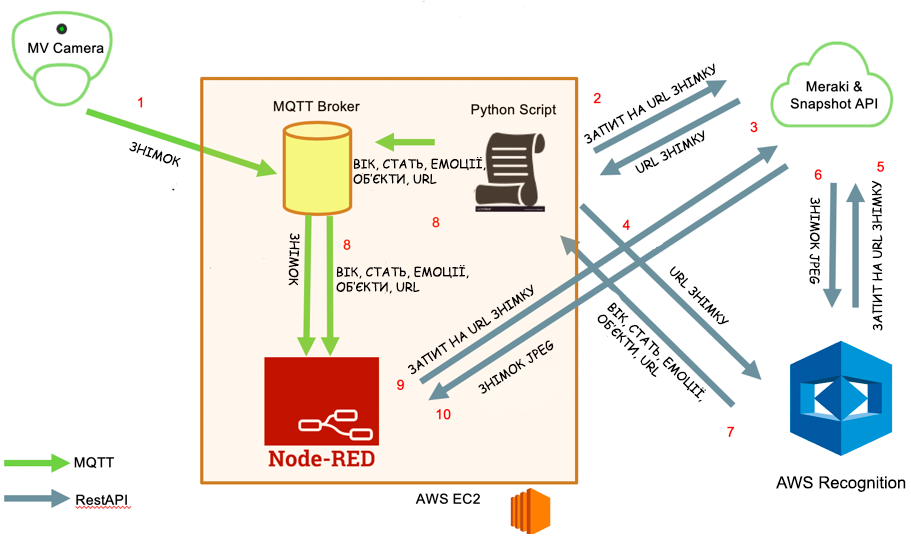

Рисунок 4.1 — Схема алгоритму взаємодії додатку з іншим програмним забезпеченням

4.2.3 Опис і обґрунтування вибору методу організації вхідних і вихідних даних

Програмний додаток має такі вхідні дані:

— URL кадру з камери;

— IP серверу;

— ключ, для підключення до камери MV;

— ID мережі;

Вихідні дані:

— картинка з розширенням JPEG;

— повідомлення на електронну пошту;

— дані про розпізнану персону в текстовому вигляді;

Принцип по якому були розроблені вхідні та вихідні дані був обраний по вимогам до системи в цілому. В цих даних немає нічого того, що би не використовувалося в процесі виконання програми, тобто являється необхідним мінімумом для працездатності додатку.

4.2.4 Опис і обґрунтування вибору складу технічних і програмних засобів

Для працездатності системи були обрані такі технології та технічні пристрої:

— Meraki Dashboard;

— Node-Red;

— MQTT протокол та брокери;

— AWS Recognition;

— REST API;

4.3 Опис розробленої програми

4.3.1 Загальні відомості

Для потрібних обчислень потрібен потік відеозапису, з якого буде отриманий кадр. При успішному імпортуванні кадру в скрипт потрібно виявити об’єкти на ньому та сортувати їх по признаку “людина”. Після сортування об’єктів в отриманому кадрі потрібно обчислити дистанцію між ними. Для коректного обчислення було введено константи для максимальної та мінімальної дистанції. Наступним кроком буде порівняння обчисленої дистанції с константами, в яких ми мануально указали порогові значення. Фінальним етапом буде отримання результатів при попередніх обчисленнях.

Рисунок 4.2 — Схема розробки додатку

Вхідною точкою в програму є блок коду **main**. Він буде відповідати за підключення до серверу. Для підключення до серверу потрібні дані, за допомогою яких і буде створене підключення. За цей процес буде відповідати блок **gather_credentials**. Далі потрібно підписати камери на брокер MQTT, а також здійснити підключення до серверу з уже відомими даними. Для цього потрібні блоки **on_connect** та **connect.** Наступний блок буде призначений для аналізу та розпізнавання об’єктів в кадрі. Даними подіями буде займатись блок **analyze.** В блоці **analyze** буде декілька залежностей. Блок **detect_text_detections** генерує текстове представлення виявленого об'єкта в отриманому скріншоті. **detect_labels** буде сортувати розпізнані об’єкти по типам. В блокі **detect_moderation** будуть виставлені рамки, по яким будуть розпізнані об’єкти. Для того, щоб додаток мав доступ до знімків, потрібно виконати запит до камери для отримання кадрів. Цим буде займатись модуль **get_meraki_snapshots.** Оброблені кадри повинні також надсилатись до серверу, це реалізовано в блоці **send_snap_to_aws.** Останнім блоком в даній системі буде **loop_forever,** в ньому буде реалізований безперервний цикл двосторонніх запитів.

4.3.2 Опис логічної структури

Для більшої зручності була побудована схема зв’язків блоків. Її вигляд можна побачити на рисунку 4.3.

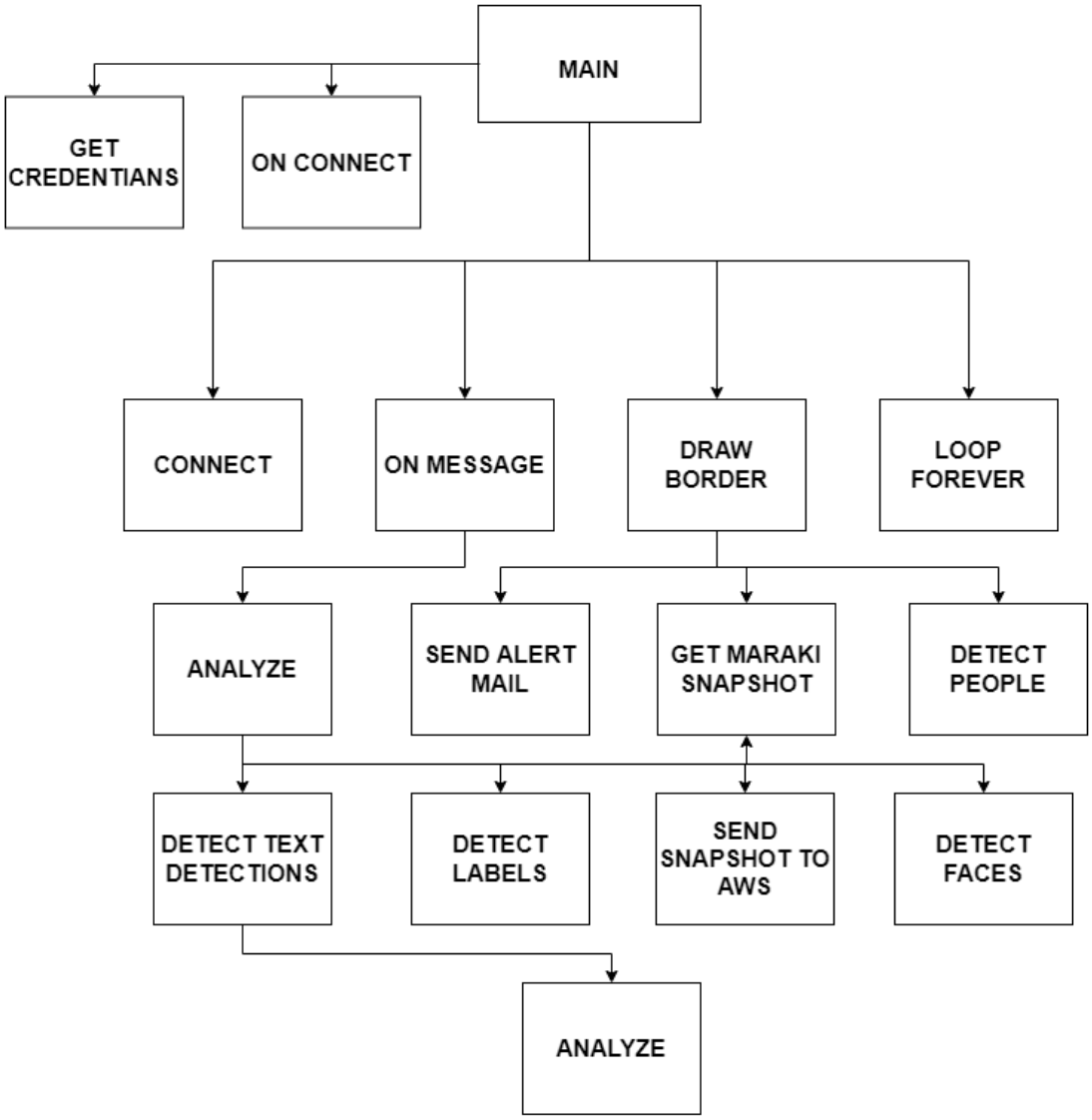

Рисунок 4.3 — Схема зв’язку блоків додатку

Далі будуть приведені частини коду, які описують логіку блоків коду показаних на рисунку 4.3.

Підключення до серверу та підписка камер на брокер MQTT. На рисунку 4.4 показана програмна реалізація блоку ON CONNECT зі схеми зв’язку блоків на рисунку 4.3.

```
def on_connect(clnMQ, usrData, rsltCode):
    print(f'Connected with result code {rsltCode}')
    serialMV = usrData['MV_SERIAL']
    clnMQ.subscribe(f'/merakimv/{serialMV}/0')
    sessionClient.subscribe(f'/merakimv/{serialMV}/light')
```

Рисунок 4.4 — Реалізація блоку ON CONNECT

Отримання ключів всіх пристроїв мережі. На рисунку 4.5 описано отримання всіх пристроїв мережі за ключом api-key, що зберігається в JSON форматі в локальному репозиторії. Виконується фільтрація пристроїв, вибираючи тільки MV камери.

В циклі йде прохід по відфільтрованим пристроям, в яких залишилися тільки камери, і робиться перевірка на відповідність серійному номеру. Якщо пристрій проходить по всіх фільтрах, то робиться запит POST на сервер MERAKI з командою зробити скріншот.

```
def get_meraki_snapshots(keyAPI, netID, TIME=None):
    global snapURL
    hdrs = {
        'X-Cisco-Meraki-API-Key': keyAPI,
    }
    rspns = currentSession.get(f'https://api.meraki.com/api/v0/networks/{netID}/dvcList',
                               headers=hdrs)
    dvcList = rspns.json()
    camerasList = [dvc for dvc in dvcList if dvc['model'][:2] == 'MV']

    for nextCamera in camerasList:
        if nextCamera["serial"] == serialMV:
            if TIME:
                hdrs['Content-Type'] = 'application/json'
                rspns = currentSession.post(
                    f'https://api.meraki.com/api/v0/networks/{netID}/camerasList/{nextCamera["serial"]}/snapshot',
                    headers=hdrs,
                    data=json.dumps({'timestamp': TIME}))
            else:
                rspns = currentSession.post(
                    f'https://api.meraki.com/api/v0/networks/{netID}/camerasList/{nextCamera["serial"]}/snapshot',
                    headers=hdrs)
            if rspns.ok:
                snapURL = rspns.json()['url']

    return snapURL
```

Рисунок 4.5 — Код блоку GET MERAKI SNAPSHOT

Отримання реквізитів входу на сервер Meraki. Для отримання реквізитів входу на сервер потрібно мати конфігураційний файл «credentials.ini». Він описує залежності, необхідні для успішного підключення. Якщо немає конфігураційного файлу або містить некоректні дані, то буде отримана помилка. Нижче можна побачити вміст файлу «credentials.ini».

```
[camera]

#RTSP source

host = rtsp://192.168.128.29:9000/live

[mqtt]

broker = broker.hivemq.com

port = 1883

[ssd]

prototxt = MobileNetSSD_deploy.prototxt.txt

model = MobileNetSSD_deploy.caffemodel

conf = 0.80
```

Рисунок 4.6 — Конфігураційний файл «credentials.ini»

```
def gather_credentials():
    configParcer = configparser.ConfigParser()
    try:
        configParcer.read('credentials.ini')
        cameraKEY = configParcer.get('meraki', 'key')
        netID = configParcer.get('meraki', 'network')
        serialMV = configParcer.get('sense', 'serial')
        serverIP = configParcer.get('server', 'ip')
    except:
        print('Something went wrong!')
        print('Credentials is not found or missing input file!')
        sys.exit(2)
    return cameraKEY, netID, serialMV, serverIP
```

Рисунок 4.7 — Код блоку GATHER CREDENTIALS

Імпорт URL-адресу знімка в AWS Rekognition. Amazon Rekognition пропонує можливості попередньо навченого та настроюваного машинного зору (CV), яке дозволяє отримувати корисну інформацію із зображень та відеозаписів.

У цьому ж блоці робиться пошук по особам за допомогою “detect_faces”.

```
def send_snap_to_aws(img):
    print("sending snapshot_url to AWS rekognition")
    botoSession = boto3.Session(profile_name='default')
    rkgn = botoSession.client('rekognition')
    rspns = requests.get(img)
    rekognitionResp = {}
    rspnsText = str(rspns)
    imgBts = rspns.content
    try:
        rekognitionResp = rkgn.detect_faces(Image={'Bytes': imgBts}, Attributes=['ALL'])
    except:
        pass

    return (rekognitionResp, rspnsText)
```

Рисунок 4.7 — Код блоку SEND SNAP TO AWS

Вибірка певних типів об’єктів. Цей блок відповідає за правильне розпізнавання людей з інших рухомих об'єктів. Програма розпізнаватиме будь-який об'єкт, потім за допомогою машинного інтелекту сортуватиме об’єкти по заданим критеріям.

```
def detect_labels(img, maxLbls=10, minConfdnc=90):
    rkgn = boto3.client("rkgn")
    rspns = requests.get(img)
    imgBts = rspns.content
    lblRspns = rkgn.detect_labels(
        Image={'Bytes': imgBts},
        MaxLabels=maxLbls,
        MinConfidence=minConfdnc,
    )
    print(str(lblRspns))
    return lblRspns['Labels']
```

Рисунок 4.8 — Код блоку DETECT LABELS

Установочні рамки для точного виявлення об’єкта. Вказання установочних рамкок для точного виявлення того чи іншого об'єкта в отриманому скрині. Всі параметри можна настроїти як завгодно. Змінна minConfdnc буде відповідати за збіг розпізнаного об’єкта з тим чи іншим лейблом. Для точності результату було виставлене значення 90, але це значення можна буде перенастроїти.

```
def detect_moderation(img, maxLbls=10, minConfdnc=90):
    rspns = requests.get(img)
    imgBts = rspns.content
    mdrtnRspnc = sessionClient.detect_moderation_labels(
        Image={'Bytes': imgBts},
        MaxLabels=maxLbls,
        MinConfidence=minConfdnc,
    )
    print(mdrtnRspnc)
    return mdrtnRspnc
```

Рисунок 4.9 — Код блоку DETECT MODERATION

Текстове представлення опису об’єкта. Отримання текстового представлення опису виявленого об'єкта в отриманому скріншоті.

```
def detect_text_detections(img):
    rkgn = boto3.client("rekognition")
    rspns = requests.get(img)
    imgBts = rspns.content
    rspnsText = rkgn.detect_text(
        Image={'Bytes': imgBts},
    )
    return rspnsText['TextDetections']
```

Рисунок 4.10 — Код блоку DETECT TEXT DETECTIONS

Звертання до Meraki для отримання Url-адреси. У цьому блоці коду описане періодичне звертання до Meraki, отримання від нього URL на скріншот та надсилання інформації на AWS Rekognition. У разі помилки підключення до сервера (error 404), у циклі йде підключення по URL до сервера.

Після звернення до сервера з успішним отриманням даних виконується надсилання їх на AWS. Також, виводиться отримана інформація в консоль та на панель Меракі.

За допомогою підключених бібліотек отримується інформація про її стан (стаття, вік, настрій). Дана інформація про клієнта дасть можливість в майбутньому доповнити функціональність даної утиліти, що допоможе компанії у розвитку інфраструктури, логіки та покращення бізнес-процесу. На основі цієї вибірки можна надалі контролювати якість послуг, що надаються, і т.д.

Після отримання та заповнення полів інформацією про клієнта йде публікація їх через MQTT у NodeRed.

```
def analyze():
    flag = True
    if flag:
        print("Requesting the Url of Snapshot..")
        snpshtURL = get_meraki_snapshots(keyAPI, netID, None)
        rspnsText = "404"
        while "404" in rspnsText:
            rekresp, rspnsText = send_snap_to_aws(snpshtURL)

        for faceDetails in rekresp['FaceDetails']:
            print('Face Analytic: Detected face is probably between ' +
                  str(faceDetails['AgeRange']['Low']) +
                  'and ' + str(faceDetails['AgeRange']['High']) + ' years old')
            age = (((faceDetails['AgeRange']['Low']) + (faceDetails['AgeRange']['High'])) / 2)
            emotionState = max(faceDetails['Emotions'], key=lambda x: x['Confidence'])
            emotion = emotionState['Type']
            sex = (faceDetails['Gender']['Value'])
            print(sex)
            print(emotion)
            print(age)

            sessionClient.publish("Age", age)
            sessionClient.publish("EmotionalState", emotion)
            sessionClient.publish("Gender", sex)
```

Рисунок 4.11 — Код блоку ANALYZE

Блок запуску додатку. Запуск MQTT client із заданими параметрами для підключення.

```
if __name__ == '__main__':
    (keyAPI, netID, serialMV, serverIP) = gather_credentials()
    sessionUserData = {
        'API_KEY': keyAPI,
        'NET_ID': netID,
        'MV_SERIAL': serialMV,
        'SERVER_IP': serverIP
    }
    serious = set()
    abnormal = set()

    args = vars(ap.parse_args())
    currentSession = requests.Session()
    sessionClient = mqtt.Client()
    sessionClient.user_data_set(sessionUserData)
    sessionClient.on_connect = on_connect
    boto3.client.on_message = on_message
    sessionClient.connect(serverIP, 1883, 300)
    sessionClient.loop_forever()
    draw_border(get_meraki_snapshots(keyAPI, netID, TIME=None))
```

Рисунок 4.12 — Код блоку MAIN

Отримання даних про об’єкти в кадрі та їх сортування. Суть завдання полягає у моніторингу пересування людей територією фітнес-клубу. У цьому блоці коду описується робота алгоритму розпізнавання людей території, охопленої камерами.

У цьому блоці через потоки запускається запис даних з камер змінну блоку “capturedVideo”. У змінну “frame” робиться запис кадру з отриманого потоку з відео. Після форматування розміру кадру йде перехід в метод “detect_people” з необхідними змінними. В результаті отримано змінну “results_list” з типом List, яка містить інформацію про людей, виявлених камерою на території фітнес-клубу.

Для виконання наступних обчислень робиться перевірку на кількість записів у змінній списку “results_list” - вона еквівалентна числу людей, яких

вдалося виявити в заздалегідь зробленому кадрі “frame”. Якщо число записів більше 1, то буде виконане обчислення відстані між виявленими людьми у кадрі.

Вимірювання відбувається на основі заданих початкових даних “MIN_DISTANCE”, “MAX_DISTANCE” та “THRESHOLD”, які можна налаштовувати у файлі конфігурації “config”.

MIN_DISTANCE – мінімально допустима дистанція між людьми.

MAX_DISTANCE - максимально допустима дистанція для людей.

THRESHOLD - граничне значення для ліміту загальної кількості порушень.

Після обчислень дистанцій описується контуром виявлених людей квадратом, а також поміщається коло по центру квадрата. Для зручності записуються дані поверх кадру в текстовому поданні, а також дублюються у консоль програми.

```
    for i in range(0, D.shape[0]):
        for j in range(i + 1, D.shape[1]):
            if D[i, j] < config.MIN_DISTANCE:
                serious.add(i)
                serious.add(j)
            if (D[i, j] < config.MAX_DISTANCE) and not serious:
                abnormal.add(i)
                abnormal.add(j)

for (i, (prob, bbox, centroid)) in enumerate(results):
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)
    if i in serious:
        color = (0, 0, 255)
    elif i in abnormal:
        color = (0, 255, 255)  # orange = (0, 165, 255)

    cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
    cv2.circle(frame, (cX, cY), 5, color, 2)
```

Рисунок 4.13 — Частина коду блоку DRAW BORDER

Надсилання повідомлення на пошту при порушенні норм дистанції. Передбачено випадок, коли кількість порушень виходить за межі допустимого заданого порогу. У такому разі використовується бібліотека “Mailer”, яка

дозволяє надсилати повідомлення на вказану пошту. Всю інформацію крок за кроком виводиться в консоль.

```
def send_alert_mail():
    if len(serious) >= config.Threshold:
        cv2.putText(frame, "-ALERT: Violations over limit-", (10, frame.shape[0] - 80),
                    cv2.FONT_HERSHEY_COMPLEX, 0.60, (0, 0, 255), 2)
        if config.ALERT:
            print("")
            print('[INFO] Sending mail...')
            Mailer().send(config.MAIL)
            print('[INFO] Mail sent')
```

Рисунок 4.13 — Частина коду блоку SEND ALERT MAIL

4.3.3 Використовувані технічні засоби

Для виконання додатку потрібна наявність таких технічних засобів в системі:

— Cisco MV72 Camera;

— точки доступу Cisco Meraki MR34;

— комутатори доступу Cisco MS320;

— комутатори рівня розподілу Cisco MS420;

— міжмережний екран Cisco Meraki MX400.

4.3.5 Виклик і завантаження

Щоб скрипт Python працював, знадобиться файл, що містить всю інформацію про налаштоване середовище, аналогічний файлам “.env”. Він буде називатися “credentials.ini”. Цей файл розміщений у тому самому місці, де знаходиться скрипт Python.

Створення віртуального середовища:

$ virtualenv venv

Клонування з репозиторію GitHub розробленого проекту:

$ git clone https://github.com/damnjohny/Meraki-Computer-Vision

$ cd Meraki-Computer-Vision

Активація готового віртуального середовищя Python:

$ source env/bin/activate

Потрібно переконатись що в папці Meraki-Computer-Vision є всі файли, з якими працювали досі:

— credentials.ini

— meraki_cv.py

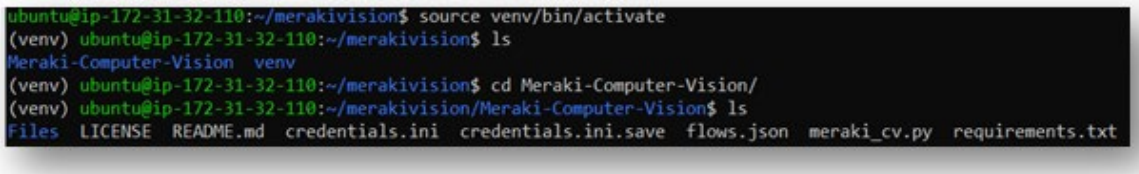

Рисунок 3.1 — Виконання активації віртуального середовища “venv”

Установка всіх вимог Python із файлу “requirements.txt” у віртуальне середовище Python:

$ sudo pip3 install -r requirements.txt

Запуск скрипту Python:

$ sudo python3 meraki_cv.py

3.2. Висновки до розділу

У ході виконання кваліфікаційної роботі були успішно вирішені наступні завдання:

— проведено огляд та вибір інструментарію для розробки;

— створено додаток за допомогою мови програмування Python;

— продукт наділений необхідним функціоналом;

— проведений приклад використання розробленого програмного забезпечення.

5. ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Сутність експерименту

Сутність даного експерименту полягає в тому, щоб перевірити працездатність вибраної налаштованої системи та отримати результати виконання розробленого додатку.

5.2 Опис процесу експерименту

Розміщення камер. Правильне розміщення камер рекомендується для того, щоб повною мірою використовувати переваги точності визначення місця розташування. Важливо відзначити, що всі камери не повинні бути згруповані усередині плану поверху. Замість цього AP можуть бути розміщені по периметру плану поверху, щоб забезпечити постійне покриття на всьому протязі. Додаткові камери можуть бути розміщені в кутах плану поверху, щоб підвищити точність визначення місця розташування для клієнтських пристроїв. Ці кутові точки доступу грають життєво-важливу роль в забезпеченні високої точності визначення місця розташування клієнтів, які перебувають всередині периметра.

В прикладі камери розміщені по куткам залів задля більш обширної зони видимості. Так ми не тільки захоплюємо більшу частину території, але й маємо можливість заощадити на сумарній кількості камер. Було вирішено поставити 4 камери в різних зонах території фітнес клубу. Таке розміщення камер було обумовлене тим, що дані зони мають найвищий показник переміщень.

Розташування камер може вплинути на відстеження місця розташування клієнта, а також на продуктивність бездротового зв'язку в цілому. Тому рекомендується, щоб камери не розташовувалися занадто близько один до одного або занадто далеко один від одного.

Приклади розміщення показані на зображеннях в плані нижче. Розміщення камер на території фітнес клубу “Спортлайф” див. рис. 5.1.

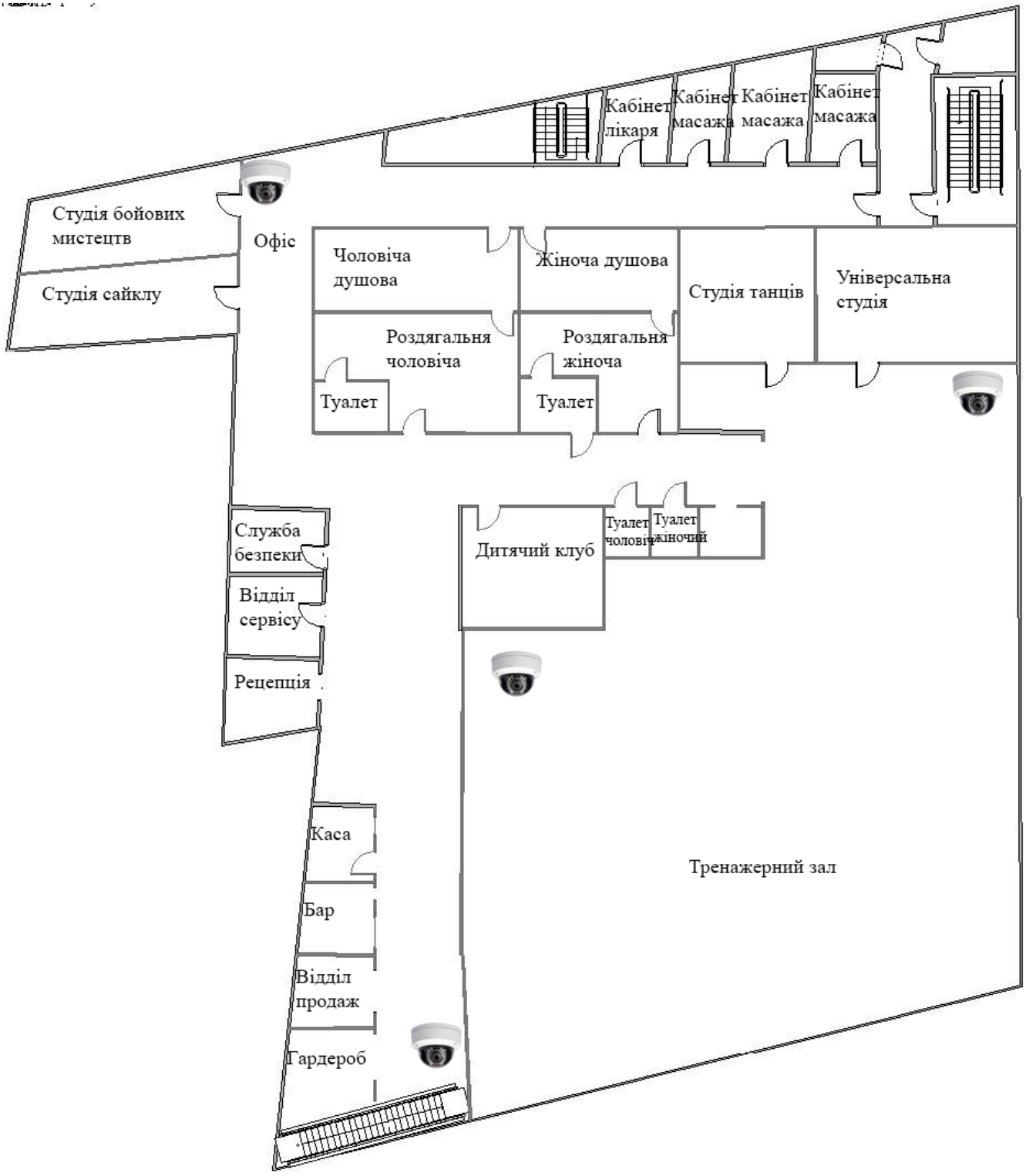

Рисунок 5.1 — Розміщення камер на території фітнес клубу «Спортлайф»

Налаштування зони камери. Зони - корисний інструмент для визначення конкретних областей, що цікавлять поле зору камери. Ці зони можна настроїти на панелі моніторингу, а потім запросити їх на API MV Sense.

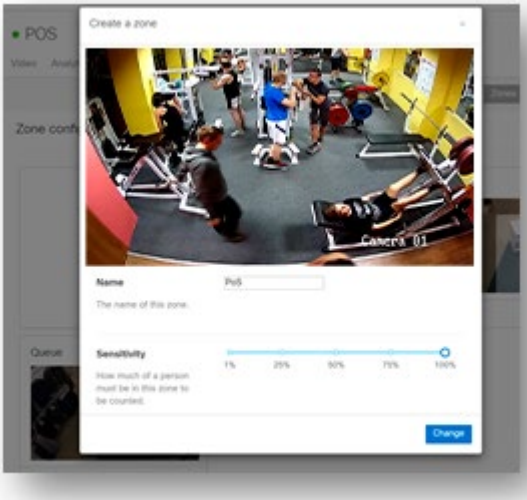

Рисунок 5.2 — Приклад налаштування зони в Meraki Camera

Налаштування зон на панелі інструментів. Щоб створити зону виконуються наступні кроки:

1. Камери > Монітор > Камери > Вибір камери.

2. Установки > Зони. Кнопка “New Zone”.

3. Вибирається область, яку було визначено, як зону для подальшої роботи, визначається назва для зони та вказується чутливість підрахунку виявлення людини.

4. Кнопка “Зберегти”.

Після створення зони можна надавати запити для будь-яких подій виявлення людей, зафіксованих у зоні.

Потрібно буде знайти номер ID зони. Це просто можна зробити двома способами:

— через приладову панель Meraki;

— через API.

ZoneID = 0 для повного кадру.

Зони – повертає всі налаштовані аналітичні зони для цієї камери.

Щоб знайти це через панель керування потрібно зробити такі дії:

— ввійти в панель керування Meraki, перейти до розділу Камери > Монітор > Камери > Ім'я камери > Установки;

— прокрутити вниз, на діаграмі знайти діаграму, у стовпці “Тема” вказані всі шляхи MQTT, доступні для цієї камери, включаючи створені зони.

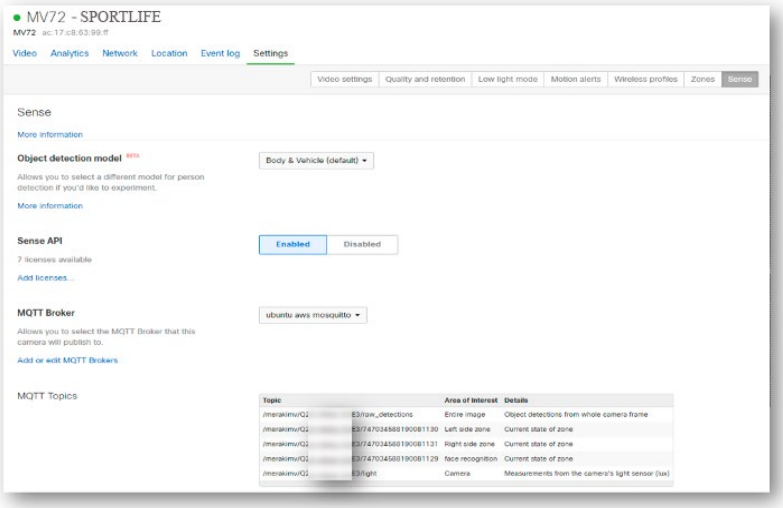

Рисунок 5.3 — Сторінка налаштування камери MV72

Щоб далі використовувати зони в Node-Red потрібно їх запам'ятати, або можна знайти їх через API.

Виконується наступний виклик API з POSTMAN, використовуючи серійний номер камери:

GET https://api.meraki.com/api/v0/devices/[serial]/camera/analytics/zones

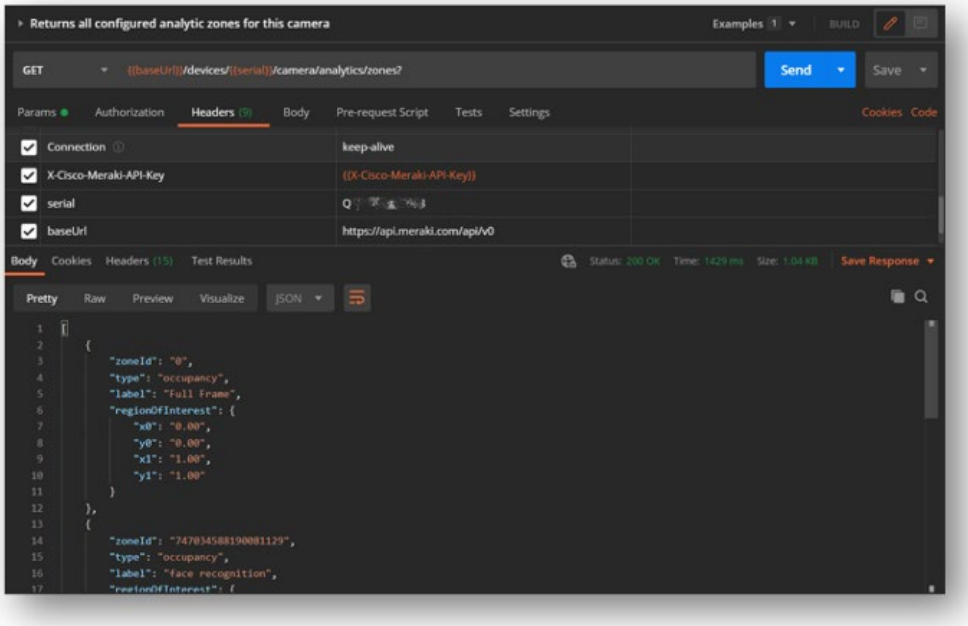

Рисунок 5.4 — Виконання GET запиту в програмі Postman

Створення сервера MQTT на панелі управління Meraki.

1. Камери > Монітор > Камери та вибирається камера, для якої потрібно увімкнути MV Sense.
2. Після вибору камери потрібно перейти до “Налаштування” > “Sense”.
3. Вибирається опція “Увімкнено”.

Щоб увімкнути MQTT на камері та створити нову конфігурацію брокера MQTT, потрібно натиснути на кнопку “Додати або відредагувати брокери MQTT”.

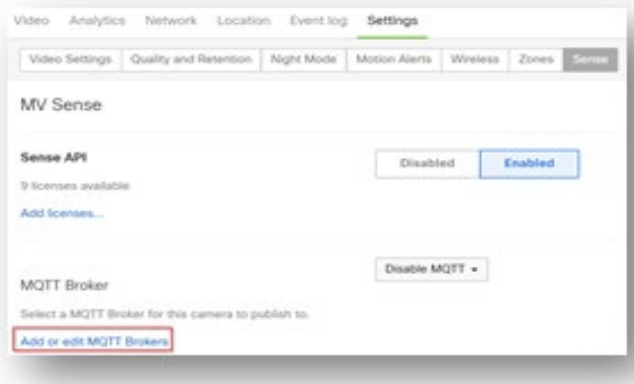

Рисунок 5.5 — Підключення камери MV до MQTT

Вводиться наступна інформація про налаштований брокер:

— ім'я брокера - ім'я нашого брокера;

— хост - це може бути IP-адреса або ім'я хоста;

— порт - це номер TCP-порту для MQTT. Найчастіше використовуються порти 1883 для TCP та 8883 для TLS.

—

Налаштування брокерського сервера. Потрібен запущений сервер Ubuntu:

— Python 3

— Node-Red

— AWS Rekognition SDK.

Python 3.8 не входитиме за замовчуванням в Ubuntu 18.04 та вище, але доступний у репозиторії Universe. Щоб встановити версію 3.8, потрібно відкрити термінал і ввести наступні команди:

$ sudo apt-get update

$ sudo apt-get install python3.8 python3-pip

Налаштування віртуального середовища venv:

$ sudo apt-get install -y python3-venv

Налаштування Node-Red.

$ sudo apt-get install -y nodejs

$ sudo apt-get install npm

$ sudo npm install -g --unsafe-perm node-red

Запуск Node-Red. Важливо переконатися в тому, що цей додаток знаходиться у фоновому режимі, тому що доведеться працювати з ним за допомогою терміналу [10].

$ node-red

Натиснути комбінацію клавіш “ctrl + z”, щоб зупинити виконання node-red, а після цього ввести команду:

$ bg

Також можна скористатися командою нижче, щоб Node-Red запускався у фоновому режимі при запуску системи:

$ sudo npm install -g pm2

$ pm2 start /usr/local/bin/node-red - -v

Далі потрібно відкрити наступні вхідні TCP-порти в групі безпеки в ньому. Дозволити доступ тільки зі своєї IP-адреси.

1880 – для графічного інтерфейсу Node-Red

1883 – для MQTT

Далі перейти на сторінку конфігурації Node-Red за адресою http://192.168.0.1:1880. Переконатися у тому, що графічний інтерфейс Node-Red з'явився на екрані.

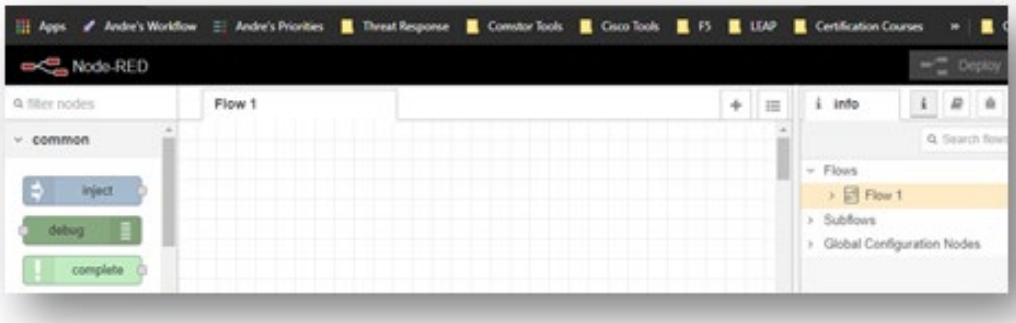

Рисунок 5.6 — Графічний інтерфейс Node-Red

Імпорт файла конфігурації Node-Red. Для подальшого налаштування знадобиться конфігураційний файл “flow.json”. Потрібно перейти до інтерфейсу користувача Node-Red і виконати наступні дії:

— Deploy > Import;

— вибрати файл для імпорту > Import;

— для завершення натиснути Confirm deploy.

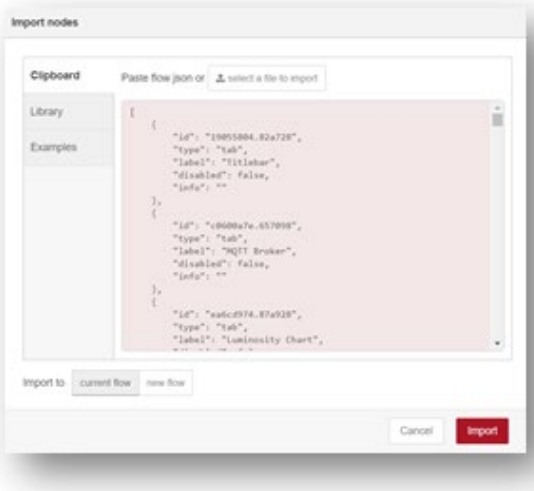

Рисунок 5.7 — Імпортування файлу конфігурації до Node-Red

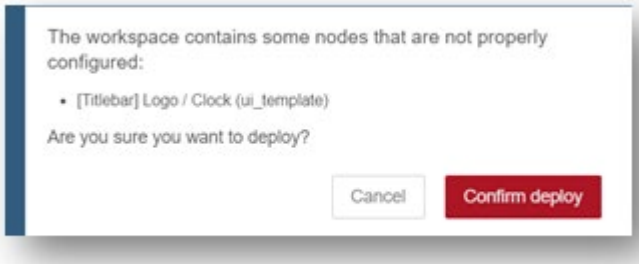

Рисунок 5.8 — Підтвердження налаштування

Після розгортання буде отримано доступ до інтерфейсу Node-Red проекту за адресою: https://192.168.0.1:1880/ui.

Передплата камер на брокера MQTT від Node-Red. Для початку потрібно переконатися, що Node-Red знає шляхи підписки MQTT камери:

— визначити шляхи MQTT нашої камери або “ZoneID” (можна зробити це за допомогою викликів API, або за допомогою панелі керування Meraki);

— додати їх у конфігурацію Node-Red на кожній вкладці з введенням MQTT;

— розгорнути оновлену конфігурацію Node-Red.

Налаштування AWS Rekognition. Ввійти в консоль AWS (https://console.aws.amazon.com/console/home). Потрібно буде настроїти AWS CLI на сервері, щоб він міг вимагати AWS Rekognition [7]. Спочатку завантажуємо та встановлюємо zip:

$ sudo apt-get install zip

Потім завантажити та встановити AWS CLI для Linux:

$ sudo curl https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip -o .zip»

$ unzip awscliv2.zip

$ sudo ./aws/install

Підтвердити встановлення:

$ aws - версія

Тепер потрібно налаштувати облікові дані в інтерфейсі командного рядка AWS на хості. Спочатку потрібно буде отримати облікові дані API, а потім додати їх на свій хост. Щоб створити обліковий запис IAM під кореневим обліковим записом AWS:

1. Заходимо в консоль управління IAM (amazon.com), використовуючи свій обліковий запис root.
2. Дозволяється доступ до платіжних даних для адміністратора IAM.
3. Ввійти в https://console.aws.amazon.com з обліковим записом Адміністратора IAM
4. Створити ключ доступу для нового користувача.
5. Використати створені ключі для створення файлів облікових даних та конфігурації за замовчуванням на сервері Linux. Перемістити їх у папку “~/.aws”.

5.3 Результат експерименту в цифрах і фактах.

В ході експерименту були отримані зображення, які відображають результати процесу виконання програмного додатку.

На рисунку 5.9 показаний результат успішного розпізнання людини на зображенні, отриманого з камери, установленої на території фітнес клубу. В даній панелі відображається інформація про стан людини в кадрі: вік, стать, та інше.

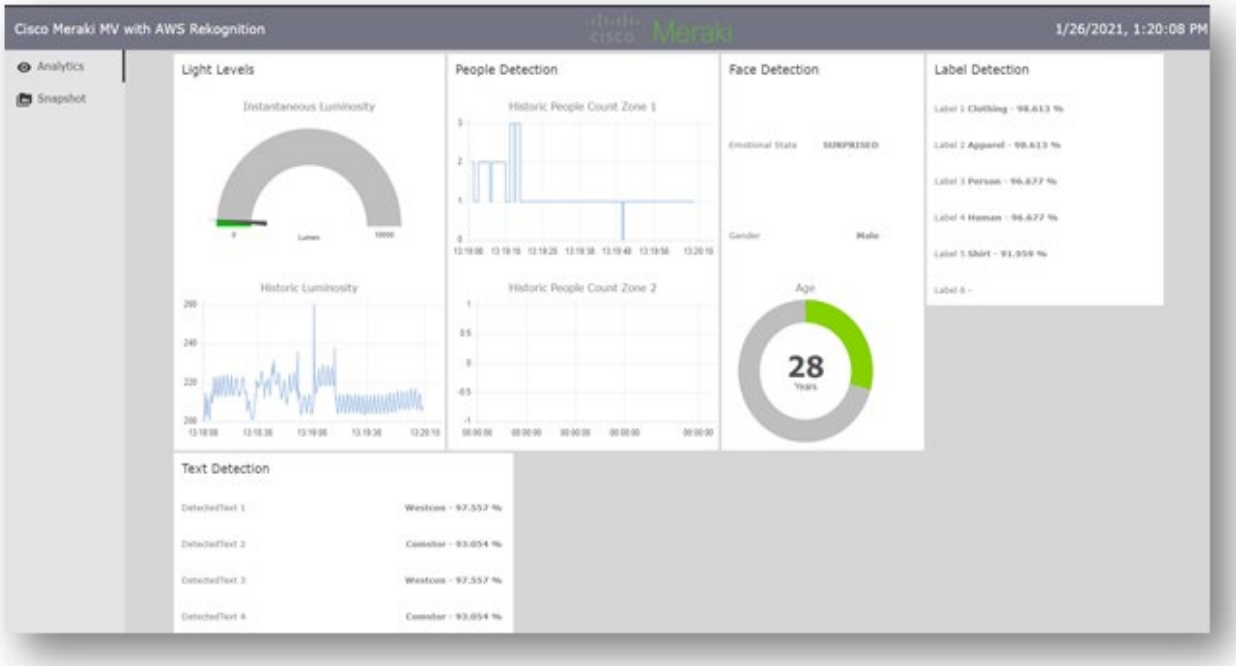

Рисунок 5.9 — Сторінка AWS Rekognition с отриманими розпізнаними даними

В ході роботи скрипта також були отримані зображення з камери. Ці зображення за допомогою машинного навчання та логіки додатку були доповнені текстом, що описує поведінку об’єктів в даному кадрі. Розглянемо перше зображення на рисунку 5.10

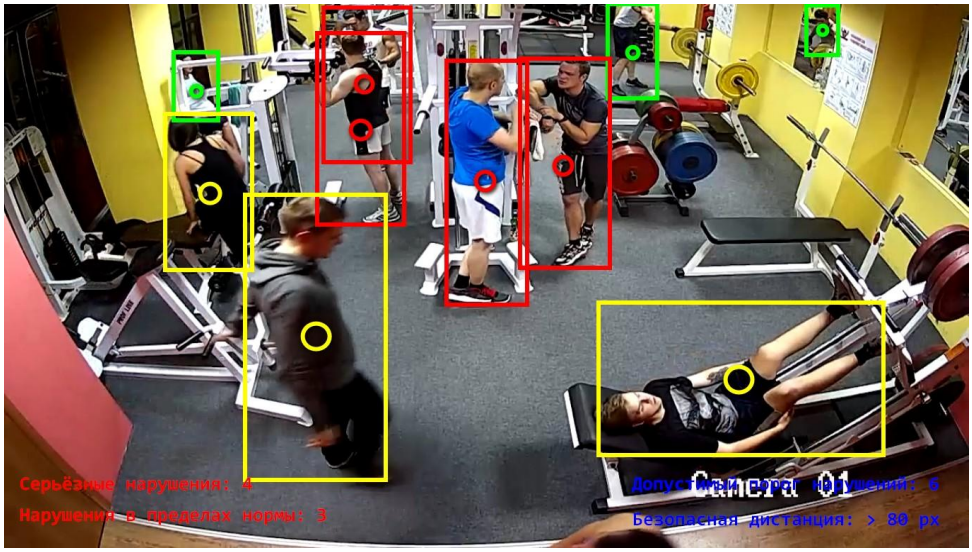

Рисунок 5.10 — Зображення з описом поведінки об’єктів в кадрі

Також можемо розглянути оригінальний кадр на рисунку 5.10 перед його обробкою нашим додатком. Можна побачити, що скрипт успішно виконав свою роботу на даному етапі.

Поля, які були добавлені скриптом в ході роботи:

— інформаційне поле кількості серйозних порушень;

— інформаційне поле кількості порушень, які входять в рамки норми;

— інформаційне поле кількості допустимих серйозних порушень в момент захвату кадру;

— інформаційне поле безпечної дистанції між об’єктами.

Рисунок 5.10 — Зображення отримане з потоку камери, без обробки додатком

Розглянемо ще один приклад обробки зображення додатком. В даному випадку на зображення накладається інформація, котра містить в собі повідомлення про перевищення допустимої границі кількості порушень в кадрі. Дана інформація добавляється по аналогії з попередніми прикладами. Приклад такого зображення можемо розглянути на рисунку 5.11.

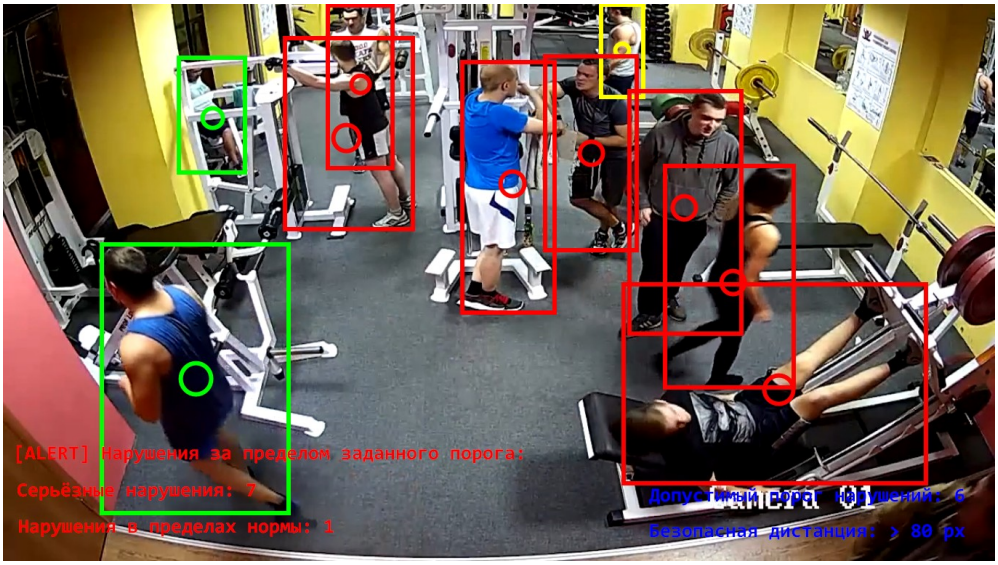

Рисунок 5.11 — Зображення з описом поведінки об’єктів в кадрі та текстом про перевищення допустимої границі порушень

В додатку була передбачена функція попередження про перевищення кількості допустимих порушень дистанційних норм на території фітнес-клубу, тож маємо результат приведений на рисунку 5.12.

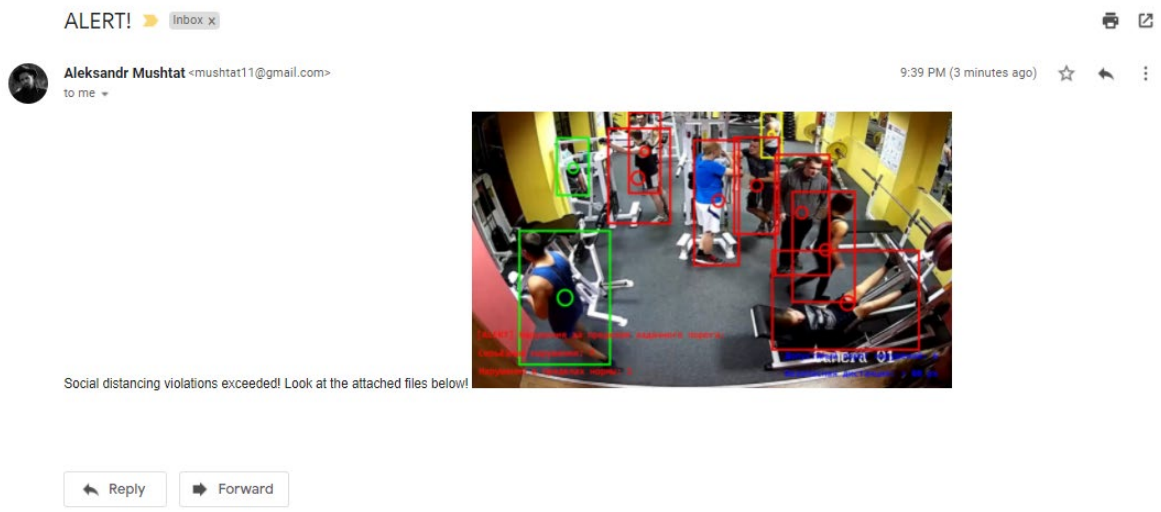

Рисунок 5.12 — Приклад повідомлення про перевищення заданої границі допустимих порушень.

ВИСНОВКИ

У даній роботі була запропонована інформаційна технологія побудови комп'ютерної системи контролю наповненості території фітнес клубу “Спортлайф” методами програмних та хмарних технологій. Були розглянуті апаратна і програмна складові запропонованої технології.

Основні принципи побудови запропонованої інформаційної технології побудови комп'ютерної системи контролю наповненості території фітнес клубу:

— підбір технологій та технічних пристроїв для реалізації

— налаштування обраних технологій для комплектування в цілісну автоматизовану систему

— створення додатку на мові Python для обробки необхідних даних в автоматизованій системі

— отримання експериментальних результатів

Наукова новизна отриманих результатів дипломної роботи визначається тим, що впровадження хмарних технологій та машинного навчання в автоматизовані системи не тільки спрощують відстежування та контроль наповненості території, але й розширюють можливості бізнесу.

Практична цінність результатів полягає у тому, що дана побудована система з вибраними технологіями розширює можливості стандартного процесу відстежування і контролю наповненості та робить його автоматизованим.

Був розроблений додаток, а також отримані результати в ході виконання роботи розробленого додатку для налаштованої автоматизованої системи в виді діаграм та зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MV Object Detection. URL:https://documentation.meraki.com/MV/Video_Analytics/MV_Object_Detection
2. MV 72 Outdoor varifocal dome camera. URL:https://meraki.cisco.com/product/security-cameras/outdoor-security-cameras/mv72
3. Adrian Rosebrock .YOLO object detection with OpenCV. URL:https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
4. Общие вопросы по Amazon. URL:https://aws.amazon.com/ru/ec2/faqs/
5. Аналитика через REST. URL:https://developer.cisco.com/meraki/mv-sense/#!rest-api/enable-api
6. MQTT и его важность. URL: https://developer.cisco.com/meraki/mv-sense/#!mqtt/what-is-mqtt
7. Начало работы с AWS. URL: https://aws.amazon.com/rekognition/getting-started/?nc=sn&loc=5
8. Andre Camillo. Начало работы с Python для сетевых инженеров. URL: https://medium.com/@andrecamillo/python-scripting-and-it-automation-30b80a3b3f32
9. Настройка зоны на панели инструментов. URL: https://developer.cisco.com/meraki/mv-sense/#!zones/configuring-zones-in-the-dashboard
10. Running Node-RED. URL: https://nodered.org/docs/getting-started/local
11. What is Amazon EC2. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
12. Basic Internet Concepts. URL: http://esate.ru/page/osnovnie-koncepcii-internet/

13. Офіційний сайт Amazon. URL: https://aws.amazon.com/

14. Офіційний сайт Meraki. URL: https://meraki.cisco.com/

15. Васильєва Т.А., Лєонов С.В. COVID-19, SARS, H5N1, A/H1N1, EVD: порівняльний аналіз впливу пандемій на економічний та соціальний розвиток у національному, світовому та регіональному контекстах. Науково-пратичний журнал «Науковий погляд: економіка та управління». - №3 (69), 2020. – С.24-28. URL: https://doi.org/10.32836/2521-666X/2020-69-4.

16. COVID-19 Health System Response Monitor (HSRM). URL: https://www.covid19healthsystem.org/mainpage.aspx

17. Геннадій Андрощук. Ефект пандемії Covid-19: цифрові технології — ключ до розвитку бізнесу. URL:https://yur-gazeta.com/golovna/efekt-pandemiyi-covid19-cifrovi-tehnologiyi--klyuch-do-rozvitku-biznesu.html

18. Вплив пандемії Covid 19 та карантину на професійну діяльність. URL: https://eba.com.ua/vplyv-pandemiyi-covid-19-ta-karantynu-na-profesijnu-diyalnist/

19. Громадська організація «Центр прикладних досліджень» Представництво Фонду Конрада Аденауера в Україні «Вплив COVID-19 та карантинних обмежень на економіку України.», с.38, 2020р. URL: https://www.kas.de/documents/270026/8703904/%D0%92%D0%BF%D0%BB%D0%B8%D0%B2+COVID-19+%D1%82%D0%B0+%D0%BA%D0%B0%D1%80%D0%B0%D0%BD%D1%82%D0%B8%D0%BD%D0%BD%D0%B8%D1%85+%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D1%8C+%D0%BD%D0%B0+%D0%B5%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D1%96%D0%BA%D1%83+%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B8.+%D0%9A%D0%B0%D0%B1%D1%96%D0%BD%D0%B5%D1%82%D0%BD%D0%B5+%D0%B4%D0%BE%D1%81%D0%BB%D1%96%D0%B4%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F+%D0%A6%D0%9F%D0%94.+%D0%9B%D0%B8%D0%BF%D0%B5%D0%BD%D1%8C+2020.pdf/b7398098-a602-524d-7f88-6189058f69d3?version=1.0&t=1597301028775

20. Вимоги щодо роботи фітнес-клубів, тренажерних залів. URL: https://dpssmk.gov.ua/vymohy-shchodo-roboty-fitnes-klubiv-trenazhernykh-zaliv/

21. Карантин економіки. URL:https://ucci.org.ua/uploads/files/602f9ce03135f554471158.pdf

22. Національний інститут стратегічних досліджень: Щодо розвитку туризму в Україні в умовах підвищених епідемічних ризиків. URL: https://niss.gov.ua/sites/default/files/2020-06/turyzm-v-ukraini.pdf

23. Marcin Łukasik. Pandemia ujawniła nierówności i słabości międzynarodowego systemu handlowego. URL: https://msp.money.pl/wiadomosci/drugi-rok-z-covid-19-pandemia-ujawnila-slabosci-miedzynarodowego-systemu-handlowego-i-obnazyla-nierownosci-6606636227852960a.html

24. МОЗ Постанова №54 від 18.09.2020р. «Протиепідемічні заходи в закладах фізичної культури та спорту на період карантину у зв'язку з поширенням коронавірусної хвороби». URL: https://zakon.rada.gov.ua/rada/show/v0054488-20#Text

25. Що таке CRM-система: повний гід по вибору CRM. URL: https://nethunt.ua/blog/shcho-takie-crm-sistiema-povnii-ghid-po-viboru-crm-dlia-pochatkivtsiv-v-2020/

26. Програма для фітнес-клубу. URL: https://appointer.ua/programa-dlya-fitnes-centriv/

27. Результати дослідження ринку CRM в Україні. URL: https://www.bitrix24.ua/crmresearch2018/

28. Що таке CRM (Customer Relationship Management)?. URL: https://www.terrasoft.ua/page/definition-crm

ДОДАТОК А

Текст програми

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КОНТРОЛЮ НАПОВНЕНОСТІ

Текст програми

804.02070743.22015-01 12 01

Листів 18

2022

АНОТАЦІЯ

Даний документ містить програмне забезпечення реалізації системи контролю наповненості.

Програма була розроблена для відстеження переміщень по території фітнес-клубу, та обчислення дистанції між людьми. Дана програма буде виконувати частину, яка необхідна для обчислень дистанційних норм, характеристик об’єктів в кадрі, правильне розпізнання людей серед інших об’єктів, а також надсилання попереджень про порушення дистанційних норм.

Дане рішення розроблене на мові програмування Python, що є великою перевагою серед інших, так як Python має велику кількість відкритих бібліотек для імпорту, що означає спрощений процес розробки.

Особливістю розробленого додатку є інтеграція його з віртуальним сервером та унікальним процесом розпізнавання людей на зображенні.

ЗМІСТ

1. Текст програми

```
import configparser
import os
import sys
import json
import time

import argparse
import cv2
import imutils
import numpy as np
import requests
import boto3
import paho.mqtt.client as mqtt
from imutils.video import FPS
from scipy.spatial import distance as dist

import meraki
from mylib import config, thread
from mylib.detection import detect_people
from mylib.mailer import Mailer

frame = None

def set_frame(frame):
    meraki.frame = frame

def get_meraki_snapshots(keyAPI, netID, TIME=None):
    hdrs = {
        'X-Cisco-Meraki-API-Key': keyAPI,
    }
    rspns                                                    =
currentSession.get(f'https://api.meraki.com/api/v0/networ
ks/{netID}/dvcList',
                                       headers=hdrs)
    dvcList = rspns.json()
    camerasList  =  [dvc  for  dvc  in  dvcList  if
dvc['model'][:2] == 'MV']
    for nextCamera in camerasList:
        if nextCamera[“serial”] == serialMV:
            if TIME:
                hdrs['Content-Type']                    =
'application/json'
                rspns = currentSession.post(
```

```
f'https://api.meraki.com/api/v0/networks/{netID}/camerasL
ist/{nextCamera[“serial”]}/snapshot',
                        headers=hdrs,
                        data=json.dumps({'timestamp':
TIME}))
                else:
                    rspns = currentSession.post(

f'https://api.meraki.com/api/v0/networks/{netID}/camerasL
ist/{nextCamera[“serial”]}/snapshot',
                        headers=hdrs)
                if rspns.ok:
                    snapURL = rspns.json()['url']

        return snapURL


    def gather_credentials():
        configParcer = configparser.ConfigParser()
        try:
            configParcer.read('credentials.ini')
            cameraKEY = configParcer.get('meraki', 'key')
            netID = configParcer.get('meraki', 'network')
            serialMV = configParcer.get('sense', 'serial')
            serverIP = configParcer.get('server', 'ip')
        except:
            print('Something went wrong!')
            print('Credentials is not found or missing
input file!')
            sys.exit(2)
        return cameraKEY, netID, serialMV, serverIP


    def send_snap_to_aws(img):
        print(“sending snapshot_url to AWS rekognition”)
        botoSession                                    =
boto3.Session(profile_name='default')
        rkgn = botoSession.client('rekognition')
        rspns = requests.get(img)
        rekognitionResp = {}
        rspnsText = str(rspns)
        imgBts = rspns.content
        try:
```

```
            rekognitionResp                                    =
rkgn.detect_faces(Image={'Bytes':                        imgBts},
Attributes=['ALL'])
        except:
            pass

        return (rekognitionResp, rspnsText)


    def detect_labels(img, maxLbls=10, minConfdnc=90):
        rkgn = boto3.client("rkgn")
        rspns = requests.get(img)
        imgBts = rspns.content
        lblRspns = rkgn.detect_labels(
            Image={'Bytes': imgBts},
            MaxLabels=maxLbls,
            MinConfidence=minConfdnc,
        )
        print(str(lblRspns))
        return lblRspns['Labels']


    def detect_moderation(img, maxLbls=10, minConfdnc=90):
        rspns = requests.get(img)
        imgBts = rspns.content
        mdrtnRspnc                                             =
sessionClient.detect_moderation_labels(
            Image={'Bytes': imgBts},
            MaxLabels=maxLbls,
            MinConfidence=minConfdnc,
        )
        print(mdrtnRspnc)
        return mdrtnRspnc


    def detect_text_detections(img):
        rkgn = boto3.client("rekognition")
        rspns = requests.get(img)
        imgBts = rspns.content
        rspnsText = rkgn.detect_text(
            Image={'Bytes': imgBts},
        )
        return rspnsText['TextDetections']
```

```
    def on_connect(clnMQ, usrData, rsltCode):
        print(f'Connected with result code {rsltCode}')
        serialMV = usrData['MV_SERIAL']
        clnMQ.subscribe(f'/merakimv/{serialMV}/0')
    sessionClient.subscribe(f'/merakimv/{serialMV}/light'
)


    def on_message(cln, usrData, msg):
        analyze()


    def analyze():
        flag = True
        if flag:
            print("Requesting the Url of Snapshot..")
            snpshtURL    =    get_meraki_snapshots(keyAPI,
netID, None)
            rspnsText = "404"
            while "404" in rspnsText:
                rekresp,            rspnsText            =
send_snap_to_aws(snpshtURL)

            for faceDetails in rekresp['FaceDetails']:
                print('Face  Analytic:  Detected  face  is
probably between ' +
                      str(faceDetails['AgeRange']['Low'])
+
                      'and               '               +
str(faceDetails['AgeRange']['High']) + ' years old')
                age = (((faceDetails['AgeRange']['Low']) +
(faceDetails['AgeRange']['High'])) / 2)
                emotionState                              =
max(faceDetails['Emotions'],        key=lambda         x:
x['Confidence'])
                emotion = emotionState['Type']
                sex = (faceDetails['Gender']['Value'])
                print(sex)
                print(emotion)
                print(age)

                sessionClient.publish("Age", age)
```

```
                sessionClient.publish("EmotionalState",
emotion)
                sessionClient.publish("Gender", sex)

            lbls = []
            object = 0
            detectedObjects = detect_labels(snpshtURL)
            quantityObjects = len(detectedObjects)
            for lbl in detectedObjects:
                trnctdCnfdnc     =     str('%.3f'     %
round((lbl["Confidence"]), 3))
                detectedObjects                          =
str("{Name}".format(**lbl))
                entry  =  detectedObjects  +  "  -  "  +
trnctdCnfdnc + " %"
                lbls.append(entry)
                lbl = ("Label" + str(object))
                print(lbl + " " + entry)
                sessionClient.publish(lbl, entry)
                object = object + 1
            sessionClient.publish("Snap", snpshtURL)
            while quantityObjects < 6:
                entry = " - "
                lbl = ("Label" + str(quantityObjects))
                quantityObjects = quantityObjects + 1
                sessionClient.publish(lbl, entry)
            dtctnsText = []
            txtCount = 0
            dtctdText = detect_text_detections(snpshtURL)
            qnttTextDtctns = len(dtctnsText)
            for DetectedText in dtctdText:
                trnctdCnfdnc     =     str('%.3f'     %
round((DetectedText["Confidence"]), 3))
                object                                   =
str("{DetectedText}".format(**DetectedText))
                entryText = object + " - " + trnctdCnfdnc
+ " %"
                dtctnsText.append(entryText)
                DetectedText    =    ("DetectedText"    +
str(txtCount))
                print(DetectedText + " " + entryText)
                sessionClient.publish(DetectedText,
entryText)
                txtCount = txtCount + 1
```

```
            sessionClient.publish("Snap", snpshtURL)
            while qnttTextDtctns < 6:
                entryText = " - "
                DetectedText    =    ("DetectedText"    +
str(qnttTextDtctns))
                qnttTextDtctns = qnttTextDtctns + 1
                sessionClient.publish(DetectedText,
entryText)


    def draw_border(snap_url):
        if not args.get("input", False):
            print("[INFO] Starting the live stream..")
            URL   =   get_meraki_snapshots(keyAPI,   netID,
None)
            vs = cv2.VideoCapture(URL)
            if config.Thread:
                cap = thread.ThreadingClass(URL)
            time.sleep(2.0)

        # otherwise, grab a reference to the video file
        else:
            print("[INFO] Starting the video..")
            vs = cv2.VideoCapture(args["input"])
            if config.Thread:
                cap                                          =
thread.ThreadingClass(args["input"])

        writer = None
        # start the FPS counter
        fps = FPS().start()

        # loop over the frames from the video stream
        while True:
            # read the next frame from the file
            if config.Thread:
                frame = cap.read()
                set_frame(frame)

            else:
                (grabbed, frame) = vs.read()
                # if the frame was not grabbed, then we
have reached the end of the stream
                if not grabbed:
```

```
                    break

            # resize the frame and then detect people (and
only people) in it
            frame = imutils.resize(frame, width=700)
            results = detect_people(frame, net, ln,

personIdx=LABELS.index("person"))

            # ensure there are *at least* two people
detections (required in
            # order to compute our pairwise distance maps)
            if len(results) >= 2:
                # extract all centroids from the results
and compute the
                # Euclidean distances between all pairs of
the centroids
                centroids = np.array([r[2] for r in
results])
                D = dist.cdist(centroids, centroids,
metric="euclidean")

                # loop over the upper triangular of the
distance matrix
                for i in range(0, D.shape[0]):
                    for j in range(i + 1, D.shape[1]):
                        # check to see if the distance
between any two
                        # centroid pairs is less than the
configured number of pixels
                        if D[i, j] < config.MIN_DISTANCE:
                            # update our violation set
with the indexes of the centroid pairs
                            serious.add(i)
                            serious.add(j)
                        # update our abnormal set if the
centroid distance is below max distance limit
                        if (D[i, j] < config.MAX_DISTANCE)
and not serious:
                            abnormal.add(i)
                            abnormal.add(j)

            # loop over the results
```

```
            for (i, (prob, bbox, centroid)) in
enumerate(results):
                # extract the bounding box and centroid
coordinates, then
                # initialize the color of the annotation
                (startX, startY, endX, endY) = bbox
                (cX, cY) = centroid
                color = (0, 255, 0)

                # if the index pair exists within the
violation/abnormal sets, then update the color
                if i in serious:
                    color = (0, 0, 255)
                elif i in abnormal:
                    color = (0, 255, 255)  # orange = (0,
165, 255)

                # draw (1) a bounding box around the person
and (2) the
                # centroid coordinates of the person,
                cv2.rectangle(frame, (startX, startY),
(endX, endY), color, 2)
                cv2.circle(frame, (cX, cY), 5, color, 2)

            # draw some of the parameters
            Safe_Distance = "Safe distance: >{}
px".format(config.MAX_DISTANCE)
            cv2.putText(frame, Safe_Distance, (470,
frame.shape[0] - 25),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.60,
(255, 0, 0), 2)
            Threshold = "Threshold limit:
{}".format(config.Threshold)
            cv2.putText(frame, Threshold, (470,
frame.shape[0] - 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.60,
(255, 0, 0), 2)

            # draw the total number of social distancing
violations on the output frame
            text = "Total serious violations:
{}".format(len(serious))
            cv2.putText(frame, text, (10, frame.shape[0] -
55),
```

```
                        cv2.FONT_HERSHEY_SIMPLEX,   0.70,
(0, 0, 255), 2)

            text1   =   "Total   abnormal   violations:
{}".format(len(abnormal))
            cv2.putText(frame, text1, (10, frame.shape[0]
- 25),
                        cv2.FONT_HERSHEY_SIMPLEX,   0.70,
(0, 255, 255), 2)

            if len(serious) >= config.Threshold:
                send_alert_mail()

    def send_alert_mail():
            cv2.putText(frame,  "-ALERT:  Violations  over
limit-", (10, frame.shape[0] - 80),
                        cv2.FONT_HERSHEY_COMPLEX,   0.60,
(0, 0, 255), 2)
            if config.ALERT:
                print("")
                print('[INFO] Sending mail...')
                Mailer().send(config.MAIL)
                print('[INFO] Mail sent')

    def detect_people(frame, net, ln, personIdx=0):

        (H, W) = frame.shape[:2]
        results = []

        blob  =  cv2.dnn.blobFromImage(frame,  1  /  255.0,
(416, 416),
            swapRB=True, crop=False)
        net.setInput(blob)
        layerOutputs = net.forward(ln)
        boxes = []
        centroids = []
        confidences = []
        for output in layerOutputs:
            for detection in output:
                scores = detection[5:]
                classID = np.argmax(scores)
                confidence = scores[classID]
```

```
                if classID == personIdx and confidence >
MIN_CONF:
                    box = detection[0:4] * np.array([W,
H, W, H])
                    (centerX, centerY, width, height) =
box.astype("int")

                    x = int(centerX - (width / 2))
                    y = int(centerY - (height / 2))

                    boxes.append([x,    y,    int(width),
int(height)])
                    centroids.append((centerX, centerY))

    confidences.append(float(confidence))

        idxs   =   cv2.dnn.NMSBoxes(boxes,   confidences,
MIN_CONF, NMS_THRESH)

        if People_Counter:
            human_count       =       "Human       count:
{}".format(len(idxs))
            cv2.putText(frame,     human_count,     (470,
frame.shape[0] - 75), cv2.FONT_HERSHEY_SIMPLEX, 0.70, (0,
0, 0), 2)

        if len(idxs) > 0:
            for i in idxs.flatten():
                # extract the bounding box coordinates
                (x, y) = (boxes[i][0], boxes[i][1])
                (w, h) = (boxes[i][2], boxes[i][3])

                r = (confidences[i], (x, y, x + w, y +
h), centroids[i])
                results.append(r)
        return results


    if __name__ == '__main__':
        (keyAPI,    netID,    serialMV,    serverIP)    =
gather_credentials()
        sessionUserData = {
            'API_KEY': keyAPI,
            'NET_ID': netID,
```

```
            'MV_SERIAL': serialMV,
            'SERVER_IP': serverIP
        }
        serious = set()
        abnormal = set()
        labelsPath = os.path.sep.join([config.MODEL_PATH,
"coco.names"])
        LABELS                                                =
open(labelsPath).read().strip().split("\n")
        weightsPath                                           =
os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
        configPath = os.path.sep.join([config.MODEL_PATH,
"yolov3.cfg"])

        net    =    cv2.dnn.readNetFromDarknet(configPath,
weightsPath)
        ln = net.getLayerNames()
        ln    =    [ln[i[0]    -    1]    for    i    in
net.getUnconnectedOutLayers()]
        ap = argparse.ArgumentParser()
        ap.add_argument("-i",    "--input",    type=str,
default="",
                        help="path  to  (optional)  input
video file")
        ap.add_argument("-o",    "--output",    type=str,
default="",
                        help="path  to  (optional)  output
video file")
        ap.add_argument("-d",    "--display",    type=int,
default=1,
                        help="whether or not output frame
should be displayed")
        args = vars(ap.parse_args())
        currentSession = requests.Session()
        sessionClient = mqtt.Client()
        sessionClient.user_data_set(sessionUserData)
        sessionClient.on_connect = on_connect
        boto3.client.on_message = on_message
        sessionClient.connect(serverIP, 1883, 300)
        sessionClient.loop_forever()

        draw_border(get_meraki_snapshots(keyAPI,   netID,
TIME=None))
```