Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.4

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: Програмно-апаратна система «Інтелектуальний сейф»

Спеціальність 121 «Інженерія програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
Радченко Микита Андрійович

Науковий керівник
к.т.н. доцент кафедри ПСТ
Меркулова Катерина Володимирівна

Київ 2022
(рік)

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____Олексій БИЧКОВ
“_____”_____20__р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТУ

Радченко Микита Андрійович
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної роботи:
Програмно-апаратна система «Інтелектуальний сейф»

затверджені наказом вищого навчального закладу від „__”_____20__р.

№_____

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані для роботи
Програмна реалізація системи, пояснювальна записка до дипломної роботи, презентація дипломного проекту

4. Зміст пояснювальної записки (перелік питань, що належить розробити)
Аналіз предметної галузі, аналіз та вибір методів аутентифікації користувача по його зображенню, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень та тестування

5. Перелік графічного матеріалу
Етапи доставки товару поштою, етапи доставки товару кур’єром, отримання купленого товару самовивозом, SVM класифікація, схема алгоритму розпізнавання обличчя, архітектура FaceNet мережі, співвідношення рівнів ZF-Net та вимог до пам’яті, співвідношення різних рівнів початкової моделі та вимог до пам’яті, діаграма розгортання системи, діаграма, що ілюструє взаємодію програмних компонентів системи, схема бази даних сервісу автентифікації, схема бази даних сервісу, що обслуговує камери схову, схема бази даних основного сервісу, взаємодія компонентів між собою за шаблоном

MVC, приклад міграції структури бази даних, приклад коду, що реалізує валідації полів, приклад коду, що реалізує контролер, приклад реалізації перетворення екземпляру класу «Box» у JSON за допомогою бібліотеки «ActiveModel Serializer», компонента «Orders», сторінка замовлень, демонстрація роботи вбудованої компоненти Google Maps, прототип сейфу у стадії MVP

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Огляд методів розпізнавання облич	К.В. Меркулова		
Визначення оптимального методу розпізнавання облич	К.В. Меркулова		
Структура пояснювальної записки	К.В. Меркулова		
Підготовка презентації до захисту дипломної роботи	К.В. Меркулова		

7. Дата видачі завдання _____

Керівник _____ Катерина МЕРКУЛОВА

(підпис)

Завдання прийняв до виконання _____ Микита РАДЧЕНКО

(підпис)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
Отримання завдання від дипломного керівника		

Підготовка матеріалів для розділу про аналіз предметної галузі		
Підготовка матеріалів для розділу про аналіз методів аутентифікації користувача		
Підготовка матеріалів для розділу про проектування системи		
Підготовка матеріалів для розділу про опис прийнятих програмних рішень		
Підготовка пояснювальної записки до кваліфікаційної роботи		

Студент-магістр _____ Микита РАДЧЕНКО
(підпис)

Керівник роботи _____ Катерина МЕРКУЛОВА
(підпис)

АНОТАЦІЯ

Обсяг звіту – 102 сторінок. Кількість рисунків – 23, 7 додатків.

Об’єкт дослідження – методи та підходи реалізації програмно-апартної системи, реалізація взаємодії апаратною частини з програмною, дослідження та реалізація автентифікації користувача за фотографією обличчя.

Мета роботи – проектування та реалізація програмно-апаратної системи керування автоматичною камерою схову (сейфом) з веб додатку, дослідження та реалізація автентифікації користувача за фотографією обличчя.

Методи дослідження – USE-CASE моделювання, моделювання за допомогою діаграми компонент системи, ER-моделювання, нейро-мережеві методи.

Розроблено програмну систему, що дозволяє зберігати, видаляти, змінювати інформацію про наявні в системі камери схову, бронювати їх та керувати доступом до них з можливістю автентифікуватись за фотографією обличчя.

Для розробки вікористовувалися: Ruby 2.6.0, Ruby on Rails 5.2.3, PostgresQL 12.2, Vue.js 2.0, RapberryPI 3, Heroku, Keras, Tensorflow.

ЗАМОК, БЕКЕНД, БАЗА ДАНИХ, ФРОНТЕНД, КОНТРОЛЛЕР , СИТЕМА З ДЕКІЛЬКОХ ПРОГРАМНИХ КОМПОНЕНТ, ЗАПИТ, RASPBERRYPI3, СИГНАЛ, МІКРОКОМП’ЮТЕР, НЕЙРОННА МЕРЕЖА, АВТЕНТИФІКАЦІЯ ПО ОБЛИЧУ.

ANNOTATION

The volume of the report is 102 pages. Number of drawings - 23, 7 applications.

The object of research - methods and approaches to the implementation of software and hardware system, the implementation of the interaction of hardware with the software, research and implementation of user authentication with a photo of the face.

The purpose of the work - design, modeling and implementation of software and hardware system that controls automatic storage units using a web application, research and implementation of user authentication by user's photo.

Research methods - USE-CASE modeling, system component modeling, ER modeling, RpiGpio library to control microcomputer voltage pins, neural network training.

A software system has been developed that allows storing, deleting, changing information about existing storage cameras in the system, booking them and controlling access to them with the ability to authenticate by face photo.

The following were used for development: Ruby 2.6.0, Ruby on Rails 5.2.3, PostgresQL 12.2, Vue.js 2.0, RapberryPI 3, Heroku, Keras, Tensorflow.

LOCK, BACKEND, DATABASE, FRONTEND, CONTROLLER, DESTRIBUTED SYSTEM, REQUEST, RASPBERRYPI3, SIGNAL, MICROCOMPUTER, NEURAL NETWORK, FACE AUTHENTICATION.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

Use-case діаграма	У розробці програмного забезпечення та системному проектуванні це опис поведінки системи, як вона відповідає на зовнішні запити.
ER-моделювання	Модель даних, що дозволяє описувати концептуальні схеми предметної галузі
SVM	Машина опорних векторів
ICA	Аналіз незалежних компонентів
PCA	Аналіз основних компонентів
LDA	Лінійний дискримінантний аналіз
HOG	Гістограма орієнтованих градієнтів
MTCNN	Multi-task Cascaded Convolutional Networks
LFW	Датасет Labeled Faces in the Wild

ВСТУП

З кожним днем автоматизується все більше і більше процесів. Люди все більше і більше схильні до того, щоб робити буденні речі не виходячи з дому. Наприклад, сьогодні майже зникла необхідність лишати домівку для того, щоб придбати необхідний товар. Ще двадцять років тому таке уявити було неможливо.

У зв'язку з цим стався поштовх розвитку технологій та засобів доставки товарів. Насьогодні існує багато варіантів того, як ви можете отримати те, що було замовлено. Саме тому проблема передачі речей між двома людьми є вкрай актуальною.

В даний момент існує дуже велика кількість програмних продуктів як схожих за своїм призначенням, так і кардинально різних. Але всі вони мають одну мету – бути корисними користувачеві. Також при розробці програмного забезпечення завжди слід враховувати той фактор, що успішним та популярним стає той продукт, яким зручно і легко користуватись. Саме тому більшість розробників сучасних програм мають на меті створити таку програму, якою якомога легше можна користуватись. Також важливим фактором є наскільки швидко користувач може виконати ту чи іншу дію в системі.

Темою даної роботи є реалізація системи, що автоматизує передачу товарів між двома людьми, роблячи цей процес більш гнучким та зручним. Також дана система матиме мінімізувати контакт між людьми, що вкрай актуально в часи пандемії.

Аутентифікація – один з невід'ємних етапів взаємодії людини з програмою. В деяких випадках аутентифікуватись доводиться досить часто, тому цей етап доцільно пришвидшити та спростити наскільки це можливо.

Саме тому наразі доволі популярним методом аутентифікації є розпізнавання обличчя. В цьому виді аутентифікації користувачеві непотрібно робити нічого, окрім наведення камери на своє лице. Даний метод є доволі надійним, оскільки не

має необхідності зберігати чи запам'ятовувати пароль. Саме тому було прийнято рішення використати цей метод у даній програмній системі.

Метою даної роботи є проектування та реалізація програмно-апаратної системи керування автоматичною камерою схову (сейфом) з веб додатку, дослідження та реалізація автентифікації користувача за фотографією обличчя.

Метою дослідження є проектування оптимальної архітектури системи опираючись на вимоги до неї, також дослідження методів розпізнавання облич та визначення оптимального.

Методи дослідження – USE-CASE моделювання, моделювання за допомогою діаграми компонент системи, ER-моделювання, нейро-мережеві методи.

В результаті роботи було розроблено програмну систему, що дозволяє зберігати, видаляти, змінювати інформацію про наявні в системі камери схову, бронювати їх та керувати доступом до них з можливістю автентифікуватись за фотографією обличчя.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

В наші дні в інтернеті можна купити майже все що завгодно – від мобільного телефона або пляшки молока до вечірньої сукні на святковий вечір. Тому вкрай актуальним є питання як куплений товар доставити до споживача, що його придбав.

Також актуальною проблемою є передача бідь-яких речей між двома людьми. Наприклад, ваша подруга залишила у вас вдома капелюх, але в вас найближчим часом зовсім не має можливості зустрітись та передати його їй. Скоріш за все в такій ситуації ви скористаєтесь послугами однієї з служб доставки.

Є декілька добре відомих нам засобів:

- доставка поштою;
- доставка кур’єром;
- самовивіз з однієї с торгових точок магазину (якщо мова йде про купівлю).

Але всі вони мають свої переваги та недоліки.

Найчастіше споживач подібних послуг обирає тип доставки залежно від своїх вподобань.

Чого потребує кінцевий користувач:

- своєчасну доставку;
- надійність (ніхто не бажає отримати зіпсований товар);
- витратити якомога менше зусиль та часу на те, щоб товар віддати/отримати;
- витратити якомога менше коштів.

Як ми можемо бачити, у даній предметній галузі фігурують такі основні поняття:

- кур’єр;

- замовлення;
- товар;
- споживач;
- відправник;
- отримувач;
- місцезнаходження;
- час передачі/доставки;
- речі, що доставляють.

На рисунку 1.1 зображені етапи, через які проходить кожен споживач послуг пошти, а саме:

- дістатися поштового відділення;
- оформити посилку та передати її поштарям;
- транспортування посилки на розподільчий склад;
- транспортування посилки до відділення отримувача;
- дістатися відділення отримання;
- оформлення отримання посилки.

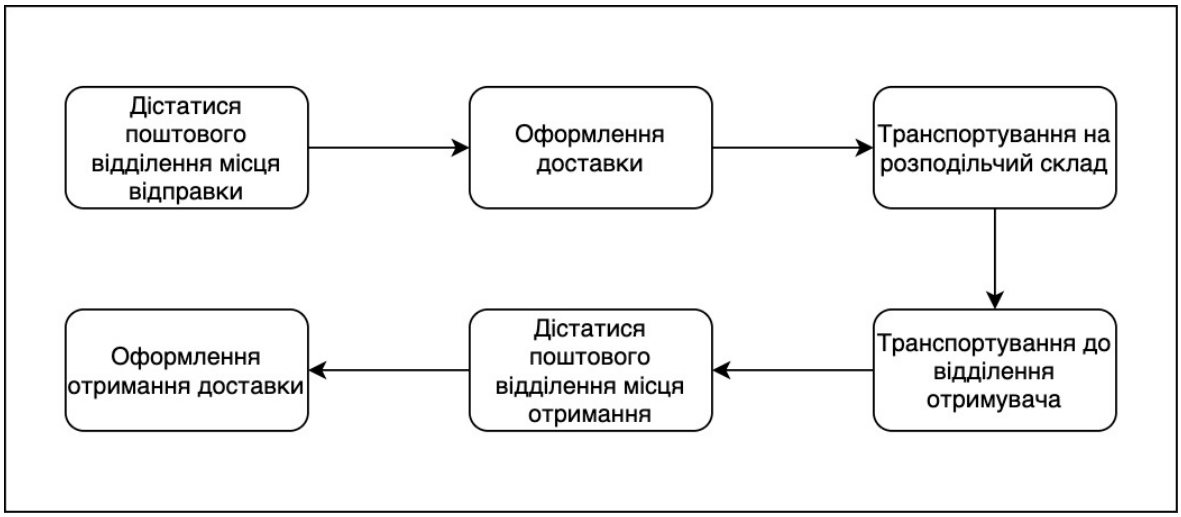


Рисунок 1.1 Етапи доставки товару поштою

На рисунку 1.2 зображені етапи, через які проходить кожен споживач послуг кур’єра, а саме:

- пошук кур'єра або кур'єрської служби;

- оформлення доставки;
- передача посилки кур'єру;
- транспортування посилки кур'єром;
- оформлення отримання посилки;
- передача посилки отримувачу.

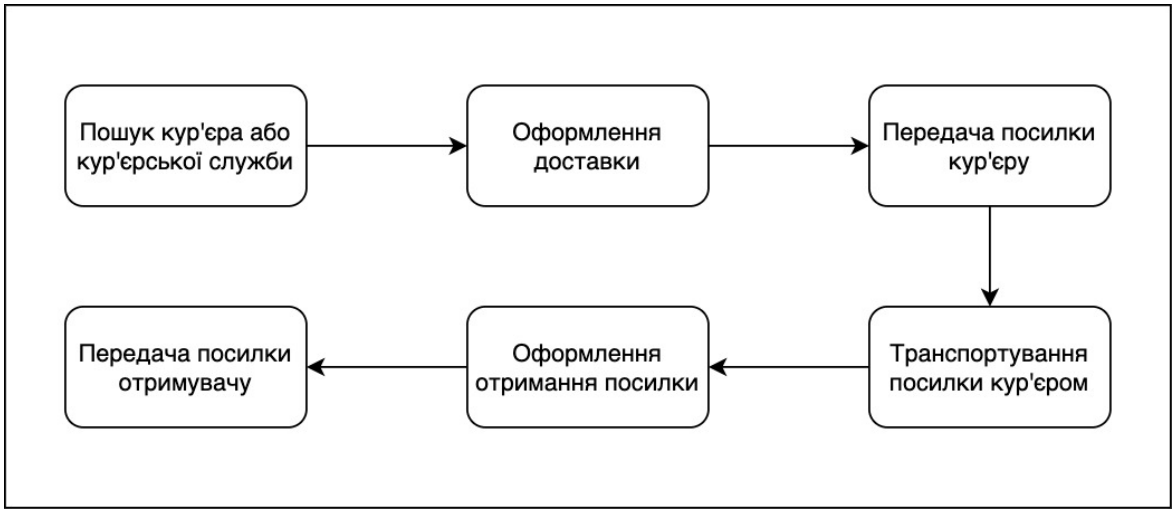


Рисунок 1.2 Етапи доставки товару кур’єром

На рисунку 1.3 зображені етапи, через які проходить кожен споживач товару, якщо обирає послугу самовивозу с торгового пункту, а саме:

- оформлення самовивозу;
- дістатися магазину представництва;
- оформлення отримання.

Рисунок 1.3 Отримання купленого товару самовивозом

Наразі в Україні сфера доставки досить розвинена, є чимало суттєвих гравців на цьому ринку.

Наприклад, існує близько десяти відомих поштових служб, чимало кур’єрських, у деяких торгових марок є навіть свою власні кур’єри. У багатьох торгових брендів є достатньо представництв, де споживачі мають змогу самостійно забрати куплений товар, проте є й недоліки в засобах, описаних вище.

1.2 Виявлення проблем та актуалізація рішень

Кожен з наявних видів доставки має свої недоліки.

Доставка поштою займає чимало часу, зокрема якщо мова йде про доставку товару в межах міста, або навіть одного району, вам все одно потрібно витратити зусилля та час на те, щоб віднести товар у відділення, вистояти чергу, яка час від часу буває чималою, оформити замовлення.

Тільки після цього товар потрапляє на склад, на якому відбувається його розподілення між перевізниками.

Наступним етапом є перевезення товару у зазначене відправником відділення.

Слід мати на увазі, що отримувач також має витратити свій час на те, щоб дістатися відділення, відстояти чергу і тільки потім забрати свою річ.

При отриманні посилки часто необхідно мати при собі документи, що підтверджують особистість.

Таким чином відправлення дістається отримувача в середньому за одну або дві доби.

З власного досвіду можу зазначити, що деколи доводиться витратити близько 15 хвилин на чергу до касира почтового відділення.

При тому близько 5 хвилин витрачається на оформлення доставки.

Наприклад, найближче почтове відділення від мого місця проживання знаходиться в 5 хвилинах пішки.

В сумі виходить 25 хвилин на відправку посилки, завелика цифра як для настільки простої дії.

Сьогодні в Україні є декілька основних гравців на ринку доставки поштою. Це «Нова пошта», «Укрпошта» та «Інтайм».

За відгуками користувачів [5], сервіс «Нова пошта», є досить швидким, має широку мережу відділень, зручну автоматизацію та інтеграцію з різними сервісами.

Але майже половина споживачів послуг даної компанії зазначили, що цей сервіс має порівняно високу вартість доставки.

«Укрпошта» навпаки має невисоку ціну, проте чверть респондентів поскаржилися на поганий сервіс, повільну доставку та незручній онлайн-кабінет.

Компанія «Інтайм», за словами тих, що пройшли опитування, пропонує послуги за невисоку ціну, але вантаж часто доставляють необережно та відділення розташовані не досить зручно.

Доставка кур'єром частіше займає менше часу, але й ціна від того вища.

Таким способом зникає одразу декілька етапів, що присутні у доставці поштою – а саме її оформлення, пересування до відділення, транспортування посилки до розподільчого складу.

Але необхідно враховувати те, що кур'єр доставить посилку в той час, коли дістанеться місцезнаходження отримувача, не завжди це буде зручно для людини, якщо вона мала на цей час свої плани.

Доставка кур'єром вразлива до ситуації на дорогах міста, якщо в місті затори, або кур'єр потрапляє у халепу, наприклад, у ДТП – то отримання замовлення відстрочується.

Даний варіант розвитку подій є ймовірним, оскільки за статистикою, що була оприлюднена інформаційним каналом ТСН [1], в період з січня по серпень 2019 року в Україні сталося 86,7 тисячі аварій, тобто в середньому щодня - 409, або ж одна ДТП кожні 3-4 хвилини.

Самовивіз з однієї з торгівельних точок магазину частіше за все є безкоштовною послугою, але не завжди в торгівельної марки є представництво у зручному для вас місці.

Таким чином, якщо ви живете у середньому за розміром населеному пункті, як, наприклад, Харків, то дорога з одного кінця міста в інше (до потенційно можливого розташування магазину) може сягати близько однієї години.

Також не можна забувати, що таким способом ви витратите час на те, щоб відстояти чергу на отримання замовлення, та на саме оформлення отримання.

В деяких поштових компаній є послуга відправки посилки через поштомат. Поштомат – електронний програмно-технічний комплекс з великою кількістю комірок, терміналом самообслуговування, що призначений для автоматичних операцій прийому і видачі посилок: кур'єр доставляє посилку до поштомату, а отримувач її забирає у зручний для себе час. Дуже зручна функція, але наразі в Україні їх недостатньо, тобто це значить, що від багатьох районів міста добиратися цих поштоматів буде незручно.

Також недолік даної реалізації такий, шо всі ці поштомати обслуговують одну тільки службу доставки, тобто неможливо придбати їх, наприклад, супермаркету, та використовувати у своїх цілях.

Також неможливо використати їх без доставки, а тільки як камеру схову (згадаємо приклад с подругою, що лишила у вас вдома капелюх).

Слід мати на увазі питання безпеки товару. Не можна допустити того, щоб товар дійшов до споживача пошкодженим, або зовсім зник.

Бували випадки, коли під час доставки поштую речі губились через недосконалість процесів або помилки співробітників, або їх цупили.

Наприклад, як якщо вірити відгуку однієї з працівниць пошти [2], то зникнення речей з посилки досить частий випадок.

Більш за це, процеси транспортування далекі від ідеалу, над цілісністю товару всередині коробки ніхто не піклується, намагаються якомога швидше завершити транспортування, це значить, що вашу посилку можуть просто кидати з рук у руки.

Більш принциповим це питання стає, коли всередині боксу щось крихке, наприклад – кришталева ваза.

Всі описані способи доставки, окрім самовивозу, частіше за все потребують додаткової сплати. Буває так, що ціна зростає суттєво.

Поштові відділення, кур'єри, точки видачі торгівельних представництв та навіть поштомати – всі вони працюють у заздалегідь встановлений час.

Наприклад, якщо вам зручно забрати куплений товар вночі – ви цього зробити не зможете, оскільки всі ці установи працюють тільки в денний час.

Також важливим є той факт, що доставка поштою, кур'єром або самовивіз вимагають безпосереднього контакту з людьми. Під час епідемій контакт стає небезпечним, оскільки як раз це сприяє поширенню різноманітних вірусів.

Наприклад в ситуації пандемії COVID-19, коли за статистикою, опублікованою ресурсом «WORLDOMETER» [3], станом на 21 травня 2021 року виявлено 167 857 345 випадків захворювання ця проблема стає вкрай актуальною.

За даними Міністерства охорони здоров'я України [4] поширення нового вірусу відбувається повітряно-крапельним шляхом. Це означає, що необхідно триматися від людини, що є потенційним переносником вірусу на відстані не менше одного метра – це забагато для того, щоб передати посилку та оформити отримання. Також це небезпечно для самих кур'єрів, оскільки їм необхідно кожного дня зустрічати величезну кількість людей та контактувати з ними.

Наразі відомо, що COVID-19 також живе на поверхнях близько 3 годин. Це означає, що є ймовірність інфікування через передачу різноманітних предметів, в тому числі і посилок.

Тим самим можна зазначити, що кожен з наявних видів доставки має свої переваги та недоліки.

1.3 Постановка задачі

Рішенням проблеми є розробка програмно-апаратної системи «Інтелектуальний сейф».

Дана система дозволить автоматизувати обмін та видачу товарів між споживачами через камери схову с керованим доступом до них, тим самим надати альтернативу існуючим засобам доставки.

Задачею цієї дипломної роботи є реалізація системи, що матиме такі переваги над іншими конкурентами:

- зробити відсутньою необхідність стояти у черзі. Надати можливість користувачам переглядати вільні камери схову та бронювати їх на конкретний час;
- оформлення доставки через веб додаток самими користувачам, що знаходиться у вільному доступі через мережу Інтернет, що дозволить економити час на реєстрації відправлення;
- виключення необхідності присутності найманого робітника при користуванні системою;
- можливість назначати або змінювати користувача впродовж існування замовлення, який забере товар (отримувача);
- авторизація тільки в межах системи, не потрібно верифікувати свою особистість за допомогою офіційних документів;
- доступ до функціоналу системи не залежно від часу доби;
- масштабованність системи для можливості розширення присутності в якомога більшій кількості точок з групою камер по місту;
- виключити або звести до мінімуму необхідність обслуговування даної системи людьми, зменшивши тим самим кінцеву вартість послуг;
- можливість інтегрувати в систему інші установи;
- можливість використання системи іншими установами;
- зручний веб ресурс з приємним дизайном;
- гарантування безпеки та цілісності товарів, що зберігається в середині камери схову;

- можливість отримання посилки відразу після того, як вона була покладена у камеру схову. Також необхідно надати можливість користуватися системою та забирати товар у будь-який час впродовж доби;
- виключити можливість камери схову бути відкритою помилково, не тією людиною.

Таким чином, маємо систему, що складається з двох частин – апаратної (камери схову) та програмної оболонки для користувачів, через яку вони будуть бронювати ці камери, відкривати та закривати в залежності від обраного сценарію.

Апаратна частина повинна керуватися з мікрокомп’ютера, щоб зменшити кінцеву вартість продукту на виході.

Необхідно реалізувати процес першого запуску камери схову якомога простішим, зробити його таким, щоб його могли виконувати й представники інших підприємств.

Надати змогу мати в системі декілька ролей з різними повноваженнями:

- звичайні споживачі, що можуть покласти/забрати товар з камери схову;
- адміністратор, що має можливість відкрити/закрити камеру;
- адміністратор, що дозволяє боксам, що з’явились у системі розпочати виконувати замовлення, редагує інформацію про них якщо потрібно, видаляє бокси з системи.

Відправник повинен мати змогу переглянути які камери схову вільні на даний час.

Також необхідно надати змогу обирати хто саме забере товар, покладений у камеру. Можна обрати себе як отримувача.

Необхідно надати змогу обирати хто саме сплатить за надані системою послуги – отримувач чи відправник.

Кінцева вартість повинна залежати від тривалості зберігання.

Необхідно реалізувати зручних механізм відкриття/закриття камери схову, також надійний механізм авторизації, щоб зловмисники не змогли викрасти покладену всередину річ.

Таким чином була сформована задача, яку потрібно розв’язати в ході даної дипломної роботи.

РОЗДІЛ 2
АНАЛІЗ ТА ВИБІР МЕТОДІВ АУТЕНТИФІКАЦІЇ КОРИСТУВАЧА ПО ЙОГО ЗОБРАЖЕННЮ

2.1 Використання технології в системі

Багато людей все більше знайомі з технологією розпізнавання обличчя завдяки функціям розблокування в нових телефонах. У цьому випадку розпізнавання обличчя використовується, щоб визначити, що особа є власником пристрою, і дозволити доступ до телефону.

Даний спосіб аутентифікації надзвичайно простий та швидкий. Тому було прийнято рішення реалізувати даний функціонал.

2.2 Загальні відомості про технологію

Розпізнавання обличчя – це технологія, яка може зіставити людське обличчя з цифрового зображення або відео з базою даних збережень облич. Розпізнавання обличчя зазвичай використовує біометричні дані, щоб допомогти визначити риси обличчя. Цей тип ідентифікації корисний для різних комерційних та правоохоронних програм. Коли справа доходить до цифрової аутентифікації, розпізнавання обличчя відноситься до категорії біометричних.

Основна мета розпізнавання облич — ідентифікувати особу, як окремо, так і серед групи людей. Кількість помилкових спрацьовувань може змінюватися в залежності від технології, яка використовується для розпізнавання обличчя. Найкращий алгоритм ідентифікації обличчя має коефіцієнт помилок 0,08%. Системи розпізнавання облич, які працюють із визначенням живості, мають вищі показники точності.

Існують різні переваги розпізнавання обличчя залежно від галузі та застосування. Це може бути зручним, безпечним і безпроблемним методом ідентифікації людини на відстані без будь-якого фізичного контакту.

Цю технологію використовують багато компаній та організацій. Ось кілька прикладів використання технології розпізнавання облич:

- контроль доступу. Контроль доступу до персональних комп'ютерів, будинків, автомобілів, офісів та інших приміщень є одним з найбільш очевидних методів використання розпізнавання облич. А iPhone X від Apple є ідеальним прикладом використання FRT для розблокування смартфона.
- покупки в Інтернеті. Alibaba, відома китайська компанія електронної комерції, планує використовувати платформу Alipay, щоб дозволити користувачам робити покупки через Інтернет. І як перший крок, Alipay вже запустила систему розпізнавання обличчя «Smile to Pay» у KFC у Ханчжоу. Система розпізнає обличчя протягом двох секунд, а потім перевіряє сканування, надсилаючи сповіщення на мобільний телефон. «Smile to Pay» також може ідентифікувати людей, які носять макіяж або перуки як маскування.
- допомога азартним гравцям. Система розпізнавання облич лише порівнює обличчя людей, які грають в ігрові автомати, з самопроголошеними проблемними гравцями в казино. Він попереджає команду безпеки, коли пристрій виявляє таких осіб, потім охорона непомітно підходить до гравців і супроводжує їх з приміщення.
- пошук злочинців. Розпізнавання облич — це технологія боротьби зі злочинністю, яка використовується для розпізнавання правопорушників правоохоронними та розвідувальними органами. Наприклад, офіцеру достатньо лише зробити знімок та завантажити його у MORIS (Mobile Offender Identification and Information System) – портативного біометричного пристрою, приєднаного до смартфона.

- упорядкування фотографій. Найпоширенішим способом використання цієї технології є компанії Apple, Google і навіть Facebook. Вони відрізняють портрет від пейзажу, знаходять користувача в кадрі та сортують фотографії за категоріями за допомогою власних систем розпізнавання облич. І ми всі надаємо величезну підтримку алгоритму розпізнавання обличчя кожного разу, коли завантажуємо зображення та відзначаємо на ньому своїх друзів.

Розпізнавання обличчя при правильному використанні призвело до підвищення функцій безпеки, зменшення випадків злочинності, пришвидшення обробки та більшої зручності для громадськості. Однак є занепокоєння щодо впровадження упередженості в системи, які покладаються на розпізнавання обличчя. Нещодавно Facebook оголосив, що припиняє використання розпізнавання облич для розпізнавання людей на своїй платформі.

2.3 Різниця між розпізнаванням обличчя та аутентифікацією за фотографією обличчя

«Автентифікація обличчям» зазвичай мається на увазі, коли потрібен збіг 1:1, наприклад, коли людина розблокує свій телефон або додаток за допомогою свого обличчя камерою мобільного телефону. Мета полягає в тому, щоб переконатися, що присутній є та сама особа, яка раніше була зареєстрована в процесі підтвердження особи.

"Розпізнавання облич" мається на увазі, коли в типі відповідності 1:n, коли система потребує пошуку в базі даних для збігу одного конкретного обличчя серед багатьох інших облич. Розпізнавання облич уже широко використовується, наприклад, поліцією в багатьох країнах, щоб шукати підозрюваних на відеозйомках з місця злочину та шукати відповідність у базі даних підозрюваних.

2.4 Опис підходів до розпізнавання облич

Існують три різні підходи до розпізнавання облич на основі нерухомих:

- Холістичний підхід. У цілісному підході вся область обличчя береться як вхідна грань. Такі методи, як аналіз основних компонентів (PCA), власні грані, грані Фішера, машина опорних векторів (SVM), аналіз незалежних компонентів (ICA) тощо, можна розглядати як цілісні підходи.
- Функціональний підхід. У цьому підході зосереджуються на певних рисах обличчя, таких як ніс, очі тощо, які розглядаються як особливості та витягуються як унікальні риси обличчя та зберігаються в базі даних. Такі методи, як чиста геометрія, архітектура динамічних посилань і прихована марковська модель, підпадають під цей підхід.
- Гібридний підхід. Цей підхід в основному є комбінаційним. В ньому разом з основними методами поєднуються новітні методи, такі як системи, подібні до штучної нейронної мережі (ANN).

2.5 Методи розпізнавання облич

Аналіз принципових компонентів (PCA). Аналіз принципових компонентів є широко використовуваним алгоритмом розпізнавання обличчя. Він в основному використовується для зменшення розмірності. Математично основними компонентами розподілу граней є власні грані. Їх також називають власними векторами, оскільки вони є коваріаційною матрицею зображень обличчя. Основною метою використання PCA є отримання власних векторів, відомих як власні грані. Кожну з граней у наборі зображень можна розглядати як лінійну комбінацію власних граней. Кожна з матриць M*N зображення перетворюється в матрицю стовпців M*1. Береться середнє значення кожної матриці стовпців і створюється нормована матриця шляхом віднімання середнього значення з кожної матриці стовпців. Обчислюється коваріація матриці, з якої знаходять власні

вектори. Найкращі грані Eigen визначають простір обличчя. Повторне вираження даних є основною метою аналізу принципових компонентів. PCA можна ефективно використовувати для зменшення шуму та надмірності[18].

Базові вектори PCA беруться з набору навчальних зображень I. Після знаходження середнього і віднімання його з навчального набору створюються вибірки даних, формула 2.1:

$$i_1 , i_2, \dots, i_n \in I - \bar{I} \tag{2.1}$$

Ці вибірки даних упорядковано в матрицю, формула 2.2:

$$\mathrm{X} = \begin{bmatrix} \begin{bmatrix} \vdots \\ i_1 \\ \vdots \end{bmatrix} & \dots & \begin{bmatrix} \vdots \\ i_n \\ \vdots \end{bmatrix} \end{bmatrix} \tag{2.2}$$

Коваріаційна матриця буде XX^T для навчальних зображень, і основні компоненти цієї коваріаційної матриці можна розрахувати за допомогою наступного рівняння(формула 2.3):

$$R^T(XX^T) = A \tag{2.3}$$

Тут A — діагональна матриця власних значень, а R — матриця ортонормованих власних векторів. Для простору граней розглядаються лише власні вектори, пов’язані з більшими власними значеннями.

Лінійний дискримінантний аналіз (LDA). Цей вид роспізнавання був розроблений Р. А. Фішером у 1930 році[19]. Лінійний дискримінантний аналіз (LDA) використовується для пошуку набору базових зображень, які допомагають максимізувати відношення міжкласового розкиду до розкиду всередині - розкид класів. Проблема з алгоритмами полягає в тому, що матриця розсіювання завжди має тенденцію бути єдиною в межах класу, і причиною може бути те, що кількість

пікселів у зображенні набагато більше, ніж кількість зображень у наборі даних. Згідно з попередніми дослідженнями, LDA може ефективно використовуватися в умовах зміни освітленості. Основна мета LDA — збільшити відстань між класами, і ця функція дуже допомагає, коли справа доходить до розпізнавання облич.

Незалежний компонентний аналіз (ICA). Незалежний компонентний аналіз – це нещодавно розроблений статистичний метод, який в деяких аспектах можна вважати кращою версією PCA. Здебільшого використовується для сліпого поділу джерел і сліпої згортки. Існує два принципово різних способи застосування ICA для розпізнавання обличчя - архітектура I і II. Архітектура I розглядає вхідні зображення як лінійну комбінацію статистично незалежних базових зображень, об'єднаних невідомою матрицею[20]. Отримані коефіцієнти не є статистично незалежними, а архітектура II має статистично незалежні коефіцієнти. Базові образи архітектури II відображають глобальні властивості.

Машина опорних векторів (SVM). Машина опорних векторів (SVM) формально визначається окремою гіперплощиною як дискримінаційний класифікатор. Коли надаються позначені навчальні дані, оптимальна гіперплощина створюється за допомогою алгоритму, який класифікує нові приклади. Машина опорних векторів (SVM) виконує класифікацію шляхом створення N-вимірної гіперплощини, яка оптимальним чином розділяє дані на дві категорії. SVM дуже тісно пов'язаний з нейронними мережами. Основною метою аналізу даних SVM є знайти ідеальну гіперплощину, яка розділяє векторні кластери таким чином, що екземпляри з однією класифікацією зазначеної змінної знаходяться з одного боку площини, а екземпляри з іншою категорією – з іншого боку площини. Опорні вектори — це вектори поблизу гіперплощини. На малюнку показана функція класифікатора SVM. На малюнку 2.1 ми бачимо, що різні класи розділяються за допомогою ліній. У другій частині малюнка ми бачимо, як класифікатор обчислив оптимальну гіперплощину та максимальний запас.

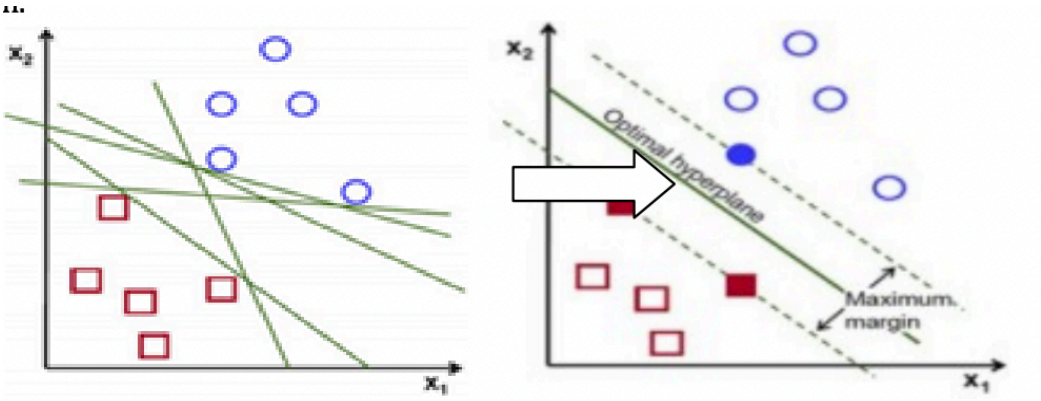


Рисунок 2.1 SVM класифікація

Даний метод можна ефективно використовувати в системі розпізнавання облич.

2.6 Опис алгоритму аутентифікації по обличчю – метод машинного навчання

Алгоритм розпізнавання облич складається з чотирьох основних етапів:

- виявлення облич - Face Detection
- вирівнювання обличчя - Face Alignment
- витяг функції - Feature Extraction
- відповідність функцій - Feature Matching

На рисунку 2.2 зображено схему алгоритму розпізнавання обличчя.

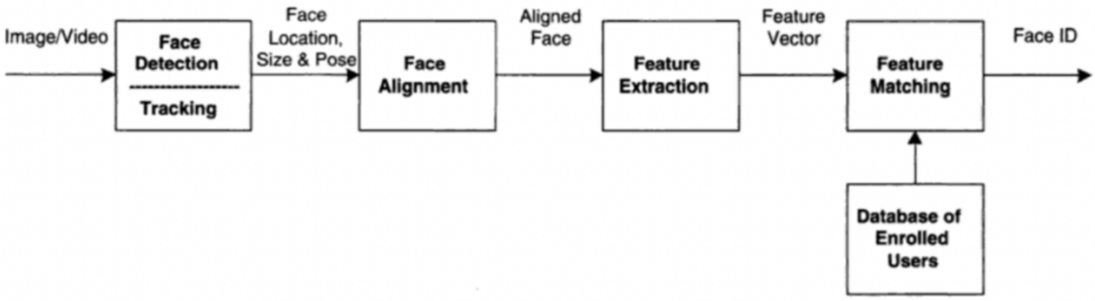


Рисунок 2.2 Схема алгоритму розпізнавання обличчя

Face Detection. Метод виявлення облич використовується для пошуку облич, присутніх на даному зображенні, вилучення облич, якщо вони існують, і обрізання

обличчя лише для створення стислого файлу для подальшого вилучення об’єктів. Існує кілька варіантів алгоритму для виконання цього завдання в системі виявлення/розпізнавання обличчя.

Методи, що використовуються в виявленні облич:

- каскадний детектор обличчя Haar: детектор обличчя на основі каскаду Haar був найсучаснішим у сфері виявлення облич протягом багатьох років з 2001 року, коли його представили Віола і Джонс у своїй статті «Швидке виявлення об’єктів з використанням посиленого каскаду простих функцій». За останні роки було багато покращень. Цей метод має просту архітектуру, яка працює майже в режимі реального часу на ЦП. Крім того, він може виявляти зображення в різних масштабах. Але основним недоліком є те, що він дає хибні результати, а також не працює на нефронтальних зображеннях.
- dlib (HOG) Face Detection: Це широко використовувана модель виявлення обличчя, заснована на функціях HoG і SVM, опублікована в 2005 році в статті «Гістограми орієнтованих градієнтів для виявлення людини». HOG, або Гістограма орієнтованих градієнтів, — це дескриптор ознак, який часто використовується для вилучення елементів із даних зображення. Це найшвидший метод на процесорі, який може працювати з фронтальними зображеннями і злегка не фронтальними. Але він не здатний виявляти невеликі зображення та обробляти оклюзії. Крім того, під час виявлення часто виключаються деякі частини підборіддя та чола.
- розпізнавання облич Dlib (CNN): Цей метод, вперше представлений у статті 2016 року «Ефективна техніка розпізнавання обличчя на основі CNN за допомогою Dlib», використовує детектор об’єктів з максимальною маржею (MMOD) з функціями на основі CNN. Процес навчання дуже простий, не має необхідності в великому обсязі даних для навчання спеціального детектора об’єктів. Він дуже швидко працює на графічному процесорі і може працювати з різними орієнтаціями обличчя на

зображеннях. Він також може впоратися з оклюзіями. Але основним недоліком є те, що він навчається на мінімальному розмірі обличчя 80*80, тому він не може виявляти маленькі обличчя на зображеннях. Він також дуже повільний на ЦП.

- MTCNN Face Detection: Multi-task Cascaded Convolutional Networks (MTCNN) — це платформа, розроблена як рішення як для виявлення обличчя, так і для вирівнювання обличчя. Цей метод вперше був представлений у статті під назвою «Визначення та вирівнювання суглобів за допомогою багатозадачних каскадних згорткових мереж» у 2016 році. Цей метод дає найточніші результати з усіх чотирьох методів. Він працює для облич, які мають різну орієнтацію на зображеннях, і може виявляти обличчя в різних масштабах. Він навіть може впоратися з оклюзіями. Він не має жодного серйозного недоліку як такого, але є порівняно повільним, ніж каскадний метод HOG і Haar.

Face Alignment. Вирівнювання обличчя є ранньою фазою сучасного процесу розпізнавання обличчя. Google повідомила, що вирівнювання обличчя підвищує точність моделі розпізнавання обличчя FaceNet з 98,87% до 99,63%. Це збільшення точності майже на 1 відсоток. Ми можемо легко застосувати 2D вирівнювання обличчя всередині OpenCV в Python. З конфігураціями каскаду haar в OpenCV є модулі для виявлення обличчя та очей. Вилучення місць для очей дуже важливо для вирівнювання обличчя. OpenCV знаходить розташування очей за допомогою звичайного каскадного методу. Отримавши розташування очей виявленого обличчя, ви можете повернути зображення на 1 градус, поки обидва ока не стануть горизонтальними. Це збільшить складність рішення, щоб ви могли вирівняти обличчя на основі кутів між двома очима, використовуючи правило косинуса. MTCNN також знаходить деякі орієнтири на обличчі, такі як розташування очей, носа та рота. Якщо ми використовуємо MTCNN у конвеєрі розпізнавання обличчя, він автоматично вирівняє виявлене обличчя.

Feature Extraction. Вилучення ознак є основним і найважливішим кроком ініціалізації для розпізнавання обличчя. Він витягує біологічні компоненти вашого обличчя. Ці біологічні компоненти є рисами вашого обличчя, які відрізняються від людини до людини. Існують різні методи, які виділяють різні комбінації ознак, відомі як вузлові точки. Жодна людина не може мати всі вузлові точки, схожі один на одного, за винятком однояйцевих близнюків.

Feature Matching. Останнім етапом технології розпізнавання обличчя є прийняття рішення, чи збігаються риси обличчя нового зразка з рисами з бази даних осіб чи ні. Ці класифікації на основі шаблонів можливі за допомогою різних статистичних підходів. Зазвичай це займає всього секунди.

2.7 Розпізнавання облич методом машинного навчання

FaceNet — це назва системи розпізнавання облич, яку запропонували дослідники Google у 2015 році у статті під назвою FaceNet: уніфіковане вбудовування для розпізнавання та кластеризації облич. Він досяг найсучасніших результатів у багатьох еталонних наборах даних розпізнавання облич, таких як Labeled Faces in the Wild (LFW) і Youtube Face Database.

Вони запропонували підхід, у якому він створює високоякісне відображення обличчя із зображень за допомогою архітектур глибокого навчання, таких як ZF-Net і Inception. Потім він використовував метод, який називається триплетною втратою, як функцію втрат для навчання цієї архітектури. Розглянемо архітектуру більш детально.

Архітектура даної нейронної мережі зображена на малюнку 2.3:

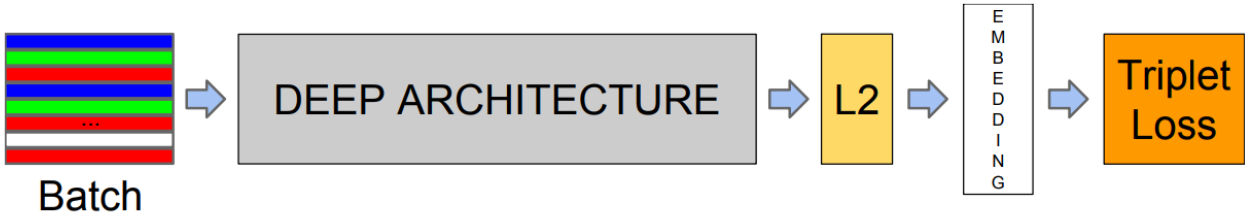


Рисунок 2.3 Архітектура FaceNet мережі

У своїй архітектурі FaceNet використовує наскрізне навчання. Він використовує ZF-Net або Inception як свою базову архітектуру. Він також додає кілька згорток 1*1, щоб зменшити кількість параметрів. Ці моделі глибокого навчання виводять вбудовування зображення $f(x)$ з нормалізацією L2. Потім ці вкладення передаються у функцію втрат для обчислення втрат. Мета цієї функції втрат полягає в тому, щоб зробити квадрат відстані між двома вбудованими зображеннями незалежним від стану зображення, а поза однакової ідентичності мала, тоді як квадрат відстані між двома зображеннями різної ідентичності великий. Тому використовується нова функція втрати, яка називається триплетною втратою. Ідея використання триплетної втрати в цій архітектурі полягає в тому, що вона змушує модель посилювати межі між гранями різних ідентичностей.

Потрійна функція втрат. Вбудовування зображення представлено $f(x)$, наприклад $x \in R$. Це вбудовування у вигляді вектора розміром 128 і нормується таким чином, що:

$$\left\| f(x) \right\|_2^2 = 1 \tag{2.4}$$

$$\| f(x_i^\alpha) - f\left(x_i^\rho\right) \|_2^2 + \alpha < \| f(x_i^\alpha) - \| f(x_i^n) \|_2^2 \tag{2.5}$$

$$\forall (f(x_i^\alpha), f\left(x_i^\rho\right), f(x_i^n)) \in \mathrm{T} \tag{2.6}$$

де α– це поле, яке дотримується для розрізнення позитивних і негативних пар, а T– простір зображення.

Тому функція втрат визначається так(формула 2.7):

$$L = \sum_i^N \left[\| f(x_i^\alpha) - f\left(x_i^\rho\right) \|_2^2 - \| f(x_i^\alpha) - f(x) \|_2^2 + \alpha \right] \tag{2.7}$$

Якщо потрійна функція втрат легко задовольняється властивістю, зазначеною вище, то це не допоможе навчанню, то важливо мати ці триплети, які порушують вищенаведене рівняння.

Вибір триплету. Щоб забезпечити швидше навчання, нам потрібно взяти триплети, які порушують наведене вище рівняння. Це означає, що для заданого x_i^α нам потрібно вибрати триплети, такі, що $\left\|f(x_i^\alpha) - f\left(x_i^\rho\right)\right\|2$ є максимальним і $\|f(x_i^\alpha) - f(x_i^n)\|2$ є мінімальним. Генерувати трійки на основі всієї навчальної множини є витратним з точки зору обчислень. Існує два способи створення триплетів:

- Створення триплетів на кожному кроці на основі попередніх контрольних точок і обчислення мінімуму та максимуму для підмножини даних.
- Вибір жорсткого позитивного x_i^ρ і жорсткого негативного x_i^ρ за допомогою мінімального та максимального значення для міні-пакету.

Навчання. Ця модель навчається за допомогою стохастичного градієнтного спуску (SGD) із зворотним поширенням та AdaGrad. Ця модель навчається на кластері ЦП протягом 1-2 тис. годин. Стійке зниження втрат (і збільшення точності) спостерігалося після 500 годин тренування. Ця модель навчається за допомогою двох мереж:

- ZF-Net. Візуалізація нижче на рисунку 2.4 вказує на різні рівні ZF-Net, які використовуються в цій архітектурі, з вимогами до пам’яті:

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Рисунок 2.4 Співвідношення рівнів ZF-Net та вимог до пам’яті

Як можемо бачити, в архітектурі є 140 мільйонів параметрів, а для навчання цієї моделі потрібно 1,6 мільярда FLOPS пам’яті.

- Inception. Візуалізація, зображена на рисунку 2.5 вказує на різні рівні початкової моделі, що використовуються в цій архітектурі, з вимогами до пам’яті:

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

Рисунок 2.5 Співвідношення різних рівнів початкової моделі та вимог до пам’яті

Як можемо бачити, в архітектурі всього 7,5 мільйонів параметрів, але для навчання цієї моделі потрібно 1,6 мільярда FLOPS пам’яті (схоже на ZF-Net).

Результати. Ця модель використовує 4 різні типи архітектури для набору даних Labeled Faces in the wild та Youtube Face DB.

Labeled Faces in the Wild набір даних. Ця архітектура використовує стандартний протокол без обмежень для набору даних LFW. По-перше, ця модель використовує 9 навчальних розділів, щоб встановити порогове значення відстані L2, а потім на десятому розподілі вона класифікує два зображення як однакові або різні.

Ця модель досягає точності класифікації 98,87% з 0,15% стандартної помилки. Це зменшує частоту помилок, про які повідомляє DeepFace, більш ніж у 7 разів, а інші найсучасніші DeepId – на 30%.

Датасет Youtube Face. У наборі даних Youtube Face повідомляється про точність 95,12% зі стандартною помилкою 0,39, використовуючи перші 100 кадрів. Це краще, ніж 91,4% точність, запропонована DeepFace, і 93,5% від DeepId на 100 кадрах.

Таким чином як результат була отримана модель, що є незмінною щодо оклюзії, пози, освітлення та навіть віку тощо.

РОЗДІЛ 3
АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Формування вимог до програмної системи

В системі, що розробляється, фігурують три основні групи користувачі, що мають різну роль та права доступу:

- кінцеві споживачі (звичайні користувачі системи);
- адміністратори системи;
- адміністратори, що керують локаціями.

У додатку А зображено Use-case діаграму актора-адміністратора, що керує локацією локацією.

Основні функції, що йому доступні:

- реєстрація;
- авторизація;
- перегляд списку камер схову, що закріплені за його локацією;
- перегляд стану камер схову (її поточного статусу);
- обрати камеру схову зі списку щоб видалити або відкрити;
- додати нову камеру схову до власної локації;
- мати доступ до статистики найбільш завантажених локацій.

Прикладом адміністратора локації може бути охоронець в супермаркеті, що вирішив поставити розумні камери схову замість старих, аналогових, до яких ми звикли.

На рисунку у додатку Б зображено Use-case діаграму звичайного користувача системи (споживача).

Основні функції, що йому доступні:

- реєстрація;
- авторизація;

- перегляд карти з вільними камерами схову;
- обрати камеру;
- оформити збереження;
- обрати платника послуг (сплатити самому чи сплатить отримувач);
- обрати отримувача, доступ до камери можуть отримати відразу декілька користувачів;
- виставити час, на яких товар буде покладено у камеру;
- продовження зберігання товару;
- перегляд списку замовлень ;
- фільтрація замовлень за їх статусом;
- обрати замовлення;
- змінити отримувача;
- відкрити камеру;
- продовжити зберігання.

У додатку В зображено Use-case діаграму адміністратора системи.

Це найсильніша в системі людина за повноваженнями.

Цей тип користувачів може переглянути абсолютно всі камери схову, що наявні в системі, подивитися їх стан, видалити з системи або відредагувати інформацію про них.

Також саме адміністратор займається тим, що підтверджує нові камери схову та дозволяє їм функціонувати повноцінно, виконувати замовлення споживачів.

Саме адміністратору також доступний список усіх замовлень, та саме він може відкрити будь-яку камеру схову.

Завдяки адміністратору звичайних зареєстрований користувач системи може стати адміністратором, закріпленим за локацією.

Адміністратор локацій завжди має бути закріплений хоча б за однією локацією. Для цього потрібно обрати одну з наявних локацій. Таким чином даний користувач матиме змогу адмініструвати обраною локацією.

Можна закріпити відразу декілька локацій за одним адміністратором локацій.

Основні функції, що йому доступні:

- авторизація [14];
- перегляд списку всіх камер схову;
- перегляд стану камери схову;
- обрати камеру схову щоб видалити її з системи або відредагувати інформацію про неї;
- підтвердити нову камеру схову;
- перегляд списку замовлень;
- обрати замовлення, щоб мати змогу відкрити камеру схову, що його виконує;
- додати адміністратора локації;
- переглянути список всіх адміністраторів, що закріплені за локацією;
- обрати одного адміністратора, щоб переглянути список усіх його локацій, видалити локацію, або додати нову.

Отже, таким чином було сформовано основні функціональні вимоги до системи, на основі яких було спроектовано користувацький інтерфейс.

3.2 UML проектування ПЗ

У додатку Г зображено основні класи веб-додатку системи.

Таким чином було виділено 6 основних сутностей та зв’язки між ними.

Клас «Користувач» (User) має такі атрибути:

- id – унікальний ідентифікатор;
- email – електронна адреса користувача, є унікальною у межах системи;
- phone_number – телефон користувача;
- first_name – ім’я користувача;
- last_name – прізвище користувача.

Клас «Користувач» має такі методи:

- admin? – предикат, що свідчить про те, є користувач адміністратор, чи ні;

- location_admin? – предикат, що свідчить про те, є користувач адміністратором локації, чи ні.

Клас «Камера схову» (Box) має такі атрибути:

- id – унікальний ідентифікатор;
- secret_key – секретний ключ, за яким здійснюється доступ до камери схову;
- payment_per_hour – сума, яку користувач повинен сплати за одну годину користування камерою схову;
- location – посилання на клас «Локація» (Location), що зберігає інформацію про місцезнаходження камери схову;
- locked – булеве значення, що говорить про те, закрита камера схову в даний момент чи відкрита.

Клас «Локація» (Location) має такі атрибути:

- id – унікальний ідентифікатор
- location_name – назва локації, наприклад, ТЦ «Караван»
- lng – широта локації (складова частина координат на місцевості)
- ltd – довгота локації (складова частина координат на місцевості)

Клас «Локація» (Location) безпосередньо зв’язаний з класом «Камера схову» (Box), оскільки без локації існування камери схови не є можливим.

Клас «Замовлення» (Order) має такі атрибути:

- id – унікальний ідентифікатор;
- started_at – час, коли замовлення почало діяти;
- ends_at – час, коли замовлення перестає діяти;
- recipients – користувачі, що є отримувачами замовлення, тобто це користувачі, що мають змогу відкрити камеру схову після того, як замовлення почало діяти;
- payment_type – тип оплати, тобто хто сплачую замовлення, відправник, чи отримувач;
- payer – платник замовлення, тобто користувач, що його повинен сплатити;

- closed_at – час, коли замовлення було закрито, тобто виконано.

Клас «Транцакція» (Transaction) має такі атрибути:

- id – унікальний ідентифікатор;
- order – посилання на клас «Замовлення» (Order), за яке було сплачено;
- payer – посилання на клас «Користувач» (User). Це поле має значення платника замовлення, тобто користувача, що його сплатив.;
- paid_at – час, коли замовлення було сплачено, тобто час, коли транзакція була створена.

Клас «Транцакція» (Transaction) безпосередньо зв’язаний з класом «Замовлення» (Order), оскільки він створюється тільки тоді, коли відбувається сплата за замовлення.

Клас «Дія» (Action) має такі атрибути:

- id – унікальний ідентифікатор;
- order – посилання на клас «Замовлення» (Order), над яким відбувалась будь-яка дія з тих, що спостерігається;
- user - посилання на клас «Користувач» (User), який справив дію;
- action_type – тип дії, що була зроблена.

Клас «Дія» (Action) безпосередньо зв’язаний з класом «Замовлення» (Order), оскільки він створюється тільки тоді, коли відбувається будь-яка дія з тих, що спостерігається.

Однією з частин системи є застосування, через яке програмні пристрої (камерами схову) взаємодіють з іншими компонентами системи. Іншими словами, тільки через цей додаток камери схову можуть надати інформацію про свій стан чи отримати команду від основного серверу. Основні класи, з яких складається дане застосування зображені у додатку Ґ.

Клас «Environment» видає доступ до даних, які змінюються в залежності від середи, в якій запущена програма.

Наприклад, URL, за яким дана програма звертається до серверу системи, змінюється залежно від того, в якому режимі запущена система – в режимі розробки чи ні.

Метод класу «Environment» – get_initial_uri – повертає URL серверу, що обслуговує дане застосування.

Клас «SystemDataLoader» відповідає за системні дані.

Він вивантажує та повертає унікальний ідентифікатор камери схову, номер піну, який подає напругу на замок та секретний ключ, за яким сервер ідентифікує девайс.

Методи класу «SystemDataLoader»:

- get_secret_key – повертає секретний ключ, за яким сервер ідентифікує девайс;
- get_pin_num – повертає номер піну, який подає напругу на замок;
- get_box_id – повертає унікальний ідентифікатор камери схову;

Виклик класу «Init» необхідний при першому завантаженні камери схову.

Методи класу «Init»:

- post_form – відправляє дані про нову камеру схову на сервер, що керує пристроями;
- store_box_id – зберігає унікальний ідентифікатор камери схову, який було отримано у відповіді сервера на запит ініціалізації.

Клас «Init» включає в себе можливості класів «Environment» та «SystemDataLoader».

Клас «StatesService» відправляє запит на сервер щодо поточного стану даної камери схову та повертає результат запиту.

Методи класу «StatesService»:

- get_state – відправляє запит на сервер та повертає його відповідь;
- parse_state – дістає з відповіді сервера стан даного девайсу.

Клас «StatesService» включає в себе можливості класів «Environment» та «SystemDataLoader».

Клас «GpioController» контролює напругу, що подається на пін gpio мікрокомп’ютера RaspberryPI [15].

Клас «GpioController» має єдиний атрибут – gpio_pin_num, що зберігає інформацію про пін gpio, від напруги якого залежить стан замку камери схову.

Методи класу «GpioController»:

- get_state – відправляє запит на сервер та повертає його відповідь;
- set_numbering – виставляє нумерацію розташування gpio пінів мікрокомп’ютера RaspberryPI;
- setup_signal – встановлює вихідний сигнал gpio піна;
- set_high – наказує мікрокомп’ютеру подати високу напругу на gpio пін;
- set_low - наказує мікрокомп’ютеру припинити подачу високої напруги на gpio пін;
- clean_up – завершує сеанс роботи с gpio піном, викликаєтсья при завершенні роботи.

Клас «Safe» - клас, що контролює роботу всього сейфу.

Клас «Safe» має єдиний атрибут – gpio_controller, що екземпляром класу «GpioController». Необхідний для того, щоб мати змогу контролювати стан замку камери схову.

Методи класу «Safe»:

- run – розпочинає роботу сейфу;
- set_state – подає команду на «GpioController» яку напругу подати на пін.

3.3 Проектування архітектури ПЗ

На рисунку 3.1 зображено діаграму розгортання системи.

Таким чином було вирішено, що система буде складатися с 9 компонентів:

- основного серверу, що обслуговує веб-клієнт;
- основної бази даних, де зберігається службова информація;
- серверу, що обслуговує камери схову;

- база даних, що зберігає поточний стан камер схову;
- веб-клієнту, через який споживачі та адміністратори взаємодіють з системою;
- клієнту, що керує пристроями (камерами схову);
- сервісу, що відповідає за авторизацію та аутентифікацію;
- бази даних сервісу аутентифікації та авторизації;
- нейронної мережі, що ідентифікує користувача по зображенню.

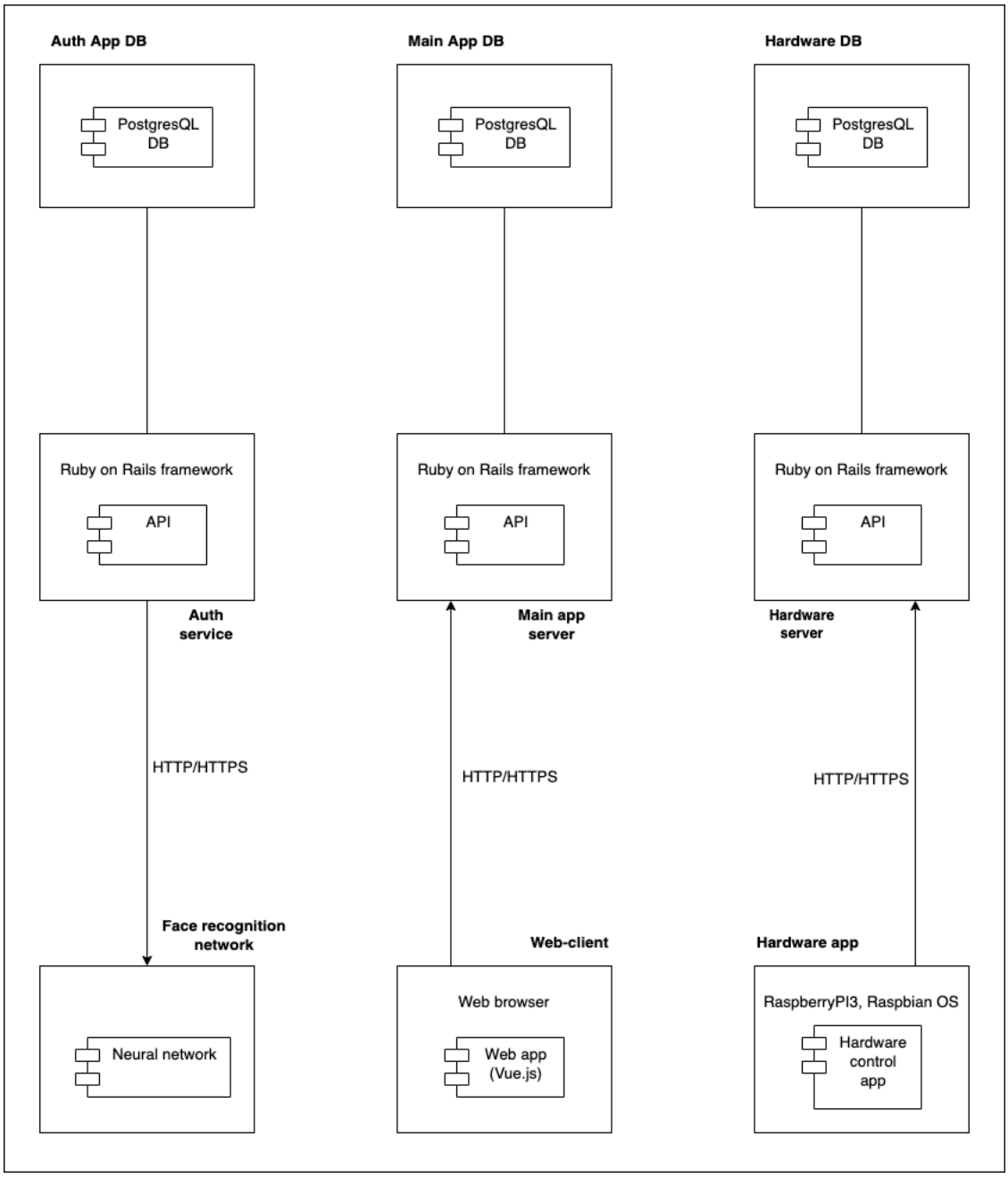


Рисунок 3.1 Діаграма розгортання системи

Основний сервіс системи(Main App Server) інкапсулює бізнес логіку системи та всі основні процеси, що в ній відбуваються. Наприклад, саме цей сервіс має інтерфейс оформлення замовлення на камеру схову, також він відповідає за створення та перегляд транзакцій окремого користувача. Як наслідок Main App DB, з якою взаємодіє основний сервер, зберігає всю необхідну для цього інформацію.

Hardware server відповідає за надання актуальної інформації щодо стану окремої камери схову, також цей сервіс має інтерфейс оновлення її поточного стану.

Дана реалізація мотивована тим, що в системі передбачена велика кількість камер схову, які мають відсилати запити з суттєвою частотою [7]. Для розподілення навантаження було вирішено розділити серверну частину системи за призначенням [6].

Поточний стан камери схову зберігається у відповідній базі даних(Hardware App DB).

Auth Service відповідає за аутентифікацію та авторизацію. Необхідні дані для цього зберігаються у відповідній базі даних Auth App DB.

Face Recognition Network – нейронна мережа, що використовує Auth Service для аутентифікації користувача за зображенням.

На рисунку 3.2 зображено діаграму, що ілюструє взаємодію між собою всіх 9 програмних компонентів [8] системи.

Веб додаток (Web app), через який споживачі та адміністратори взаємодіють з системою має зв’язок тільки з головним сервером (Main App server).

Головний сервер, та сервер, що обслуговує камери схову, мають окремі бази даних(Hardware DB та Main App DB). Таким чином ці два сервіси є максимально незалежними один від одного. Також внесення змін в одну логічну частину не вимагає внесення змін в іншу. Але в разі змінення стану камери схову(відкриття або закриття замку) вимагає додаткового запиту на Hardware server.

Камери схову взаємодіють зі своїм окремим сервером (Hardware server) через клієнтський додаток Hardware App, що розгорнуто на мікрокомп’ютері Raspberry Pi 3.

З Auth Service взаємодіє тільки Main App server заради аутентифікування користувача та перевірку його прав на проведення відповідної операції в рамках системи.

З Auth App DB взаємодіє тільки Auth Service для читання та запису даних, що необхідні для аутентифікації окремого користувача та перевірки його повноважень.

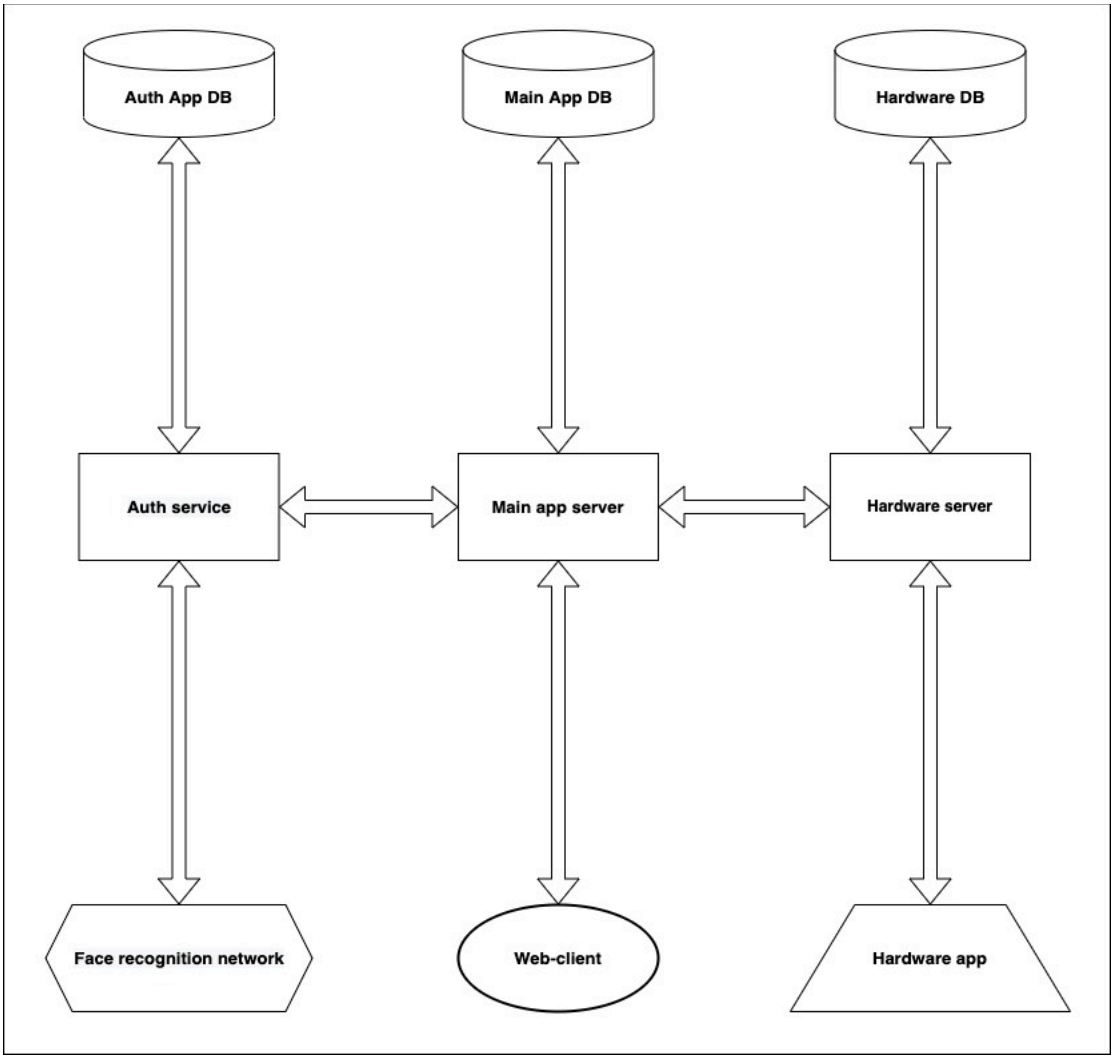


Рисунок 3.2 Діаграма, що ілюструє взаємодію програмних компонентів системи

Face Recognition Network приймає запити на перевірку відповідності користувача за його фотографією та надає відповідь на них сервісу аутентифікації.

3.4 Проектування структури зберігання даних

База даних є невід'ємною складовою майже будь-якої прикладної системи, та програмно-апаратна система "Інтелектуальний сейф" не є винятком.

Процес проектування бази даних [9] складається з визначення інформації, яка потрібна бути присутньою в системі.

Виходячи з функціональних вимог до системи, маємо такі основні сутності:

- користувач;
- камера схову;
- локація;
- замовлення;
- транзакція;
- дія.

Є декілька вимог та обмежень до поведінки описаних вище сутностей:

- користувач може мати права адміністратора, що надають йому більше повноважень в межі системи;
- користувач може займати роль адміністратора локацій, що надають йому права адміністрування камерами схову в межі певної локації;
- камера схову закріплена тільки за однією локацією;
- замовлення може бути створене тільки на одну камеру схову;
- замовлення зберігає тільки одного користувача, що його створив;
- замовлення може сплатити тільки один користувач;
- замовлення може мати декілька отримувачів;
- користувач може бути отримувачем декількох замовлень;
- над замовленнями може відбуватись багато дій;
- користувач може бути ініціатором багатьох дій;

- до замовлення може відноситись багато транзакцій;
- користувач може створювати багато транзакцій над замовленнями;
- користувач може мати декілька завантажених фотографій себе для подальшої автентифікації по обличчю.

Оскільки система поділена на декілька окремих сервісів, що мають кардинально різне логічне призначення, то і сховища даних, що необхідні для їх функціонування, також мають бути окремими серверами бази даних.

Таким чином маємо три бази даних:

- Auth App DB;
- Hardware App DB;
- Main App DB;

На рисунку 3.3 наведено схему бази даних сервісу автентифікації.

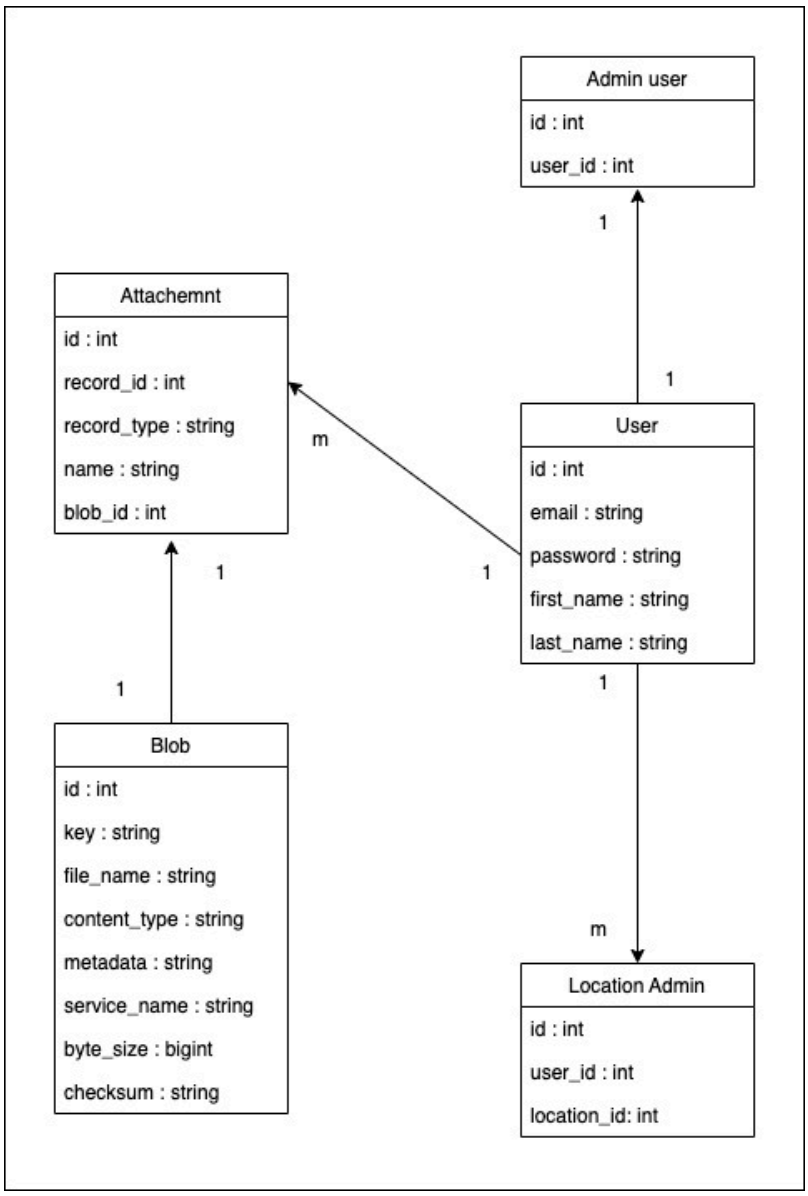


Рисунок 3.3 Схема бази даних сервісу автентифікації

Сутність «адміністратор-користувач» необхідна для того, щоб зберігати інформацію про те, хто з користувачів системи має права адміністратора.

Сутність «адміністратор» локацій необхідна для того, щоб зберігати інформацію про те, який користувач та на якій саме локації має права адміністратора.

Сутність «користувач» необхідна для того, щоб зберігати інформацію про користувача: його повне ім’я, електронну адресу та пароль.

Сутність «вкладення» та залежна від неї сутність відповідають за збереження пов'язаною з завантаженими фотографіями користувача інформації.

Шляхом проектування бази даних було виявлено наступні зв’язки між сутностями:

- користувач – адміністратор-користувач (Admin user – User) – один до одгоно;
- користувач – адміністратор локацій (User – Location Admin) – один до багатьох;
- користувач – вкладення (User – Attachment) – один до багатьох;
- вкладення – блоб(Attachment – Blob) – один до одного.

На рисунку 3.4 наведено схему бази даних сервісу, що обслуговує камери схову.

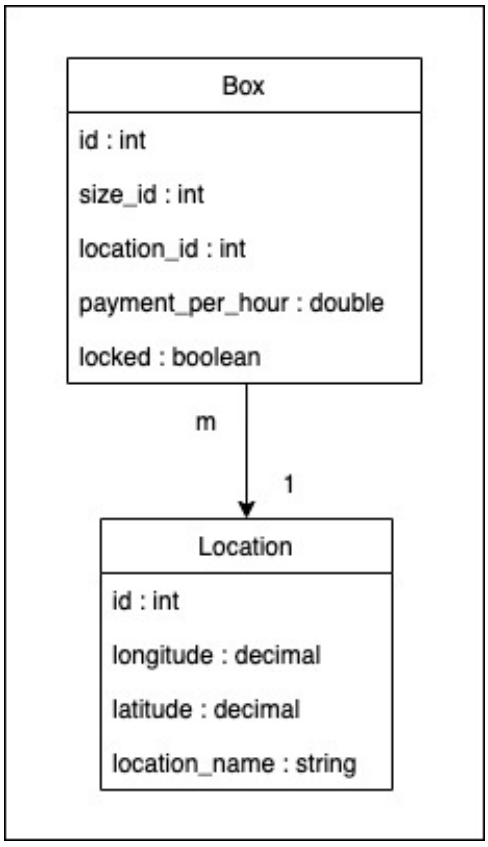


Рисунок 3.4 Схема бази даних сервісу, що обслуговує камери схову

Сутність «камера схову» необхідна для збереження інформації щодо поточного стану сейфу, його розмірів та ціни за його використання.

Сутність «локація» необхідна для того, щоб зберігати інформацію щодо локації, де розташована група камер схову. До цієї інформації належить її місцезнаходження(широта та довгота) та безпосередньо назва цієї локації.

Шляхом проектування бази даних було виявлено наступний зв’язок між сутностями:

- локація – камера схову(Location – Box) – один до багатьох.

На рисунку 3.5 наведено схему бази даних основного сервісу.

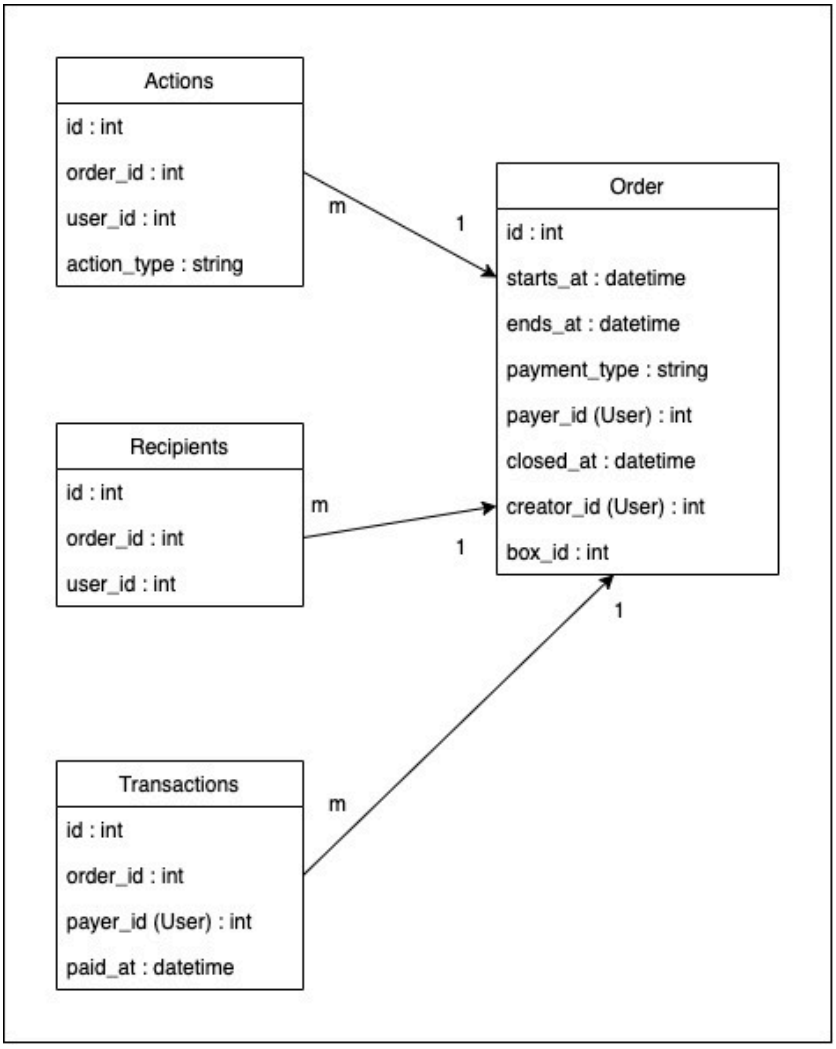


Рисунок 3.5 Схема бази даних основного сервісу

Сутність «замовлення» необхідна для збереження інформації щодо бронювання камери схову, а саме коли замовлення вступає в силу, коли очікується його завершення, як саме послугу буде сплачено, хто її сплачує або сплатить, хто його ініціював та на яку саме камеру схову це замовлення було оформлено.

Сутність «дія» має призначення зберігати історію дій, що були здійснені над замовленням та ким саме вони були здійснені.

Сутність «транзакція» зберігає інформацію про платежі за замовлення та самого платника.

Сутність «отримувач» зберігає інформацію про те, хто саме має отримувати замовлення.

Шляхом проектування бази даних було виявлено наступні зв’язки між сутностями:

- замовлення – дія(Order – Actions) – один до багатьох;
- замовлення – отримувач(Order – Recipient) – один до багатьох;
- замовлення – транзакція(Order – Transactions) – один до багатьох;

Як СУБД було вирішено використовувати PostgresQL як яскраву представницю сімейства реляційних СУБД [17], що зарекомендувала себе як надійне open-source рішення, що добре маштабується.

РОЗДІЛ 4
ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ ТА ТЕСТУВАННЯ

4.1 Серверна частина

Для реалізації серверної частини застосунку було обрано фреймворк Ruby on Rails разом з мовою програмування Ruby.

Дані технології зарекомендували себе як надійне рішення для написання масштабованих додатків-API для клієнтів, що зручно підтримувати та розширювати з часом.

Ruby – скриптова об'єктно орієнтована мова програмування с динамічною типізацією, що найчастіше використовується для розробки серверних рішень у зв’язку с фреймворком Ruby on Rails [13].

Переваги Ruby on Rails:

- висока швидкість розробки. Проекти на Rails розробляються дійсно швидше аналогів на PHP, Python або Java, це підтверджено досвідом багатьох розробників та є загальноприйнятим фактом. Обумовлено це як технічними особливостями архітектури фреймворку, так і інструментами для розробки (консольні утиліти і генератори, готові бібліотеки, розширення і модулі);
- складна бізнес-логіка реалізується простіше та прозоріше. Це обумовлено тим, що конвенції написання програмного коду на базі Rails дозволяють писати дійсно зрозумілий програмний код, який згодом простіше супроводжувати та модифікувати в адекватні терміни. Дотримання закладених у фреймворк угод і стандартів кодування робить програмний код пригодним до супроводжування не тільки тим, хто розробляв його спочатку, а й будь-якими іншими фахівцями, що приходять на проект пізніше;

- висока надійність та подальша підтримка рішень. У розробці на Rails найчастіше використовують TDD-підхід, це обумовлено прекрасними вбудованими у сам фреймворк засобами, що робить створені рішення більш стабільними. Функціональність самого фреймворку також покрита тестами, що робить його використання дійсно надійним. Для бізнес-систем ця складова вкрай важлива – від стабільності роботи додатку часто залежить ефективність роботи бізнесу в цілому.
- потужність та високі навантаження. Фреймворк пристосований для розробки рішень, що мають високі вимоги до їх продуктивності.

Ruby on Rails реалізує патерн MVC, було прийняте рішенні дотримуватися цього шаблону під час реалізації додатку, оскільки таким чином кількість залежностей між програмними компонентами знижується. Складові частини за цим шаблоном є такими:

- model – модель. Бізнес модель або об'єкт предметної області;
- view – подання. Елементи користувальницького інтерфейсу;
- controller – контролеру. Відстежує події користувальницького інтерфейсу.

На рисунку 4.1 зображено як компоненти за шаблоном MVC взаємодіють між собою.

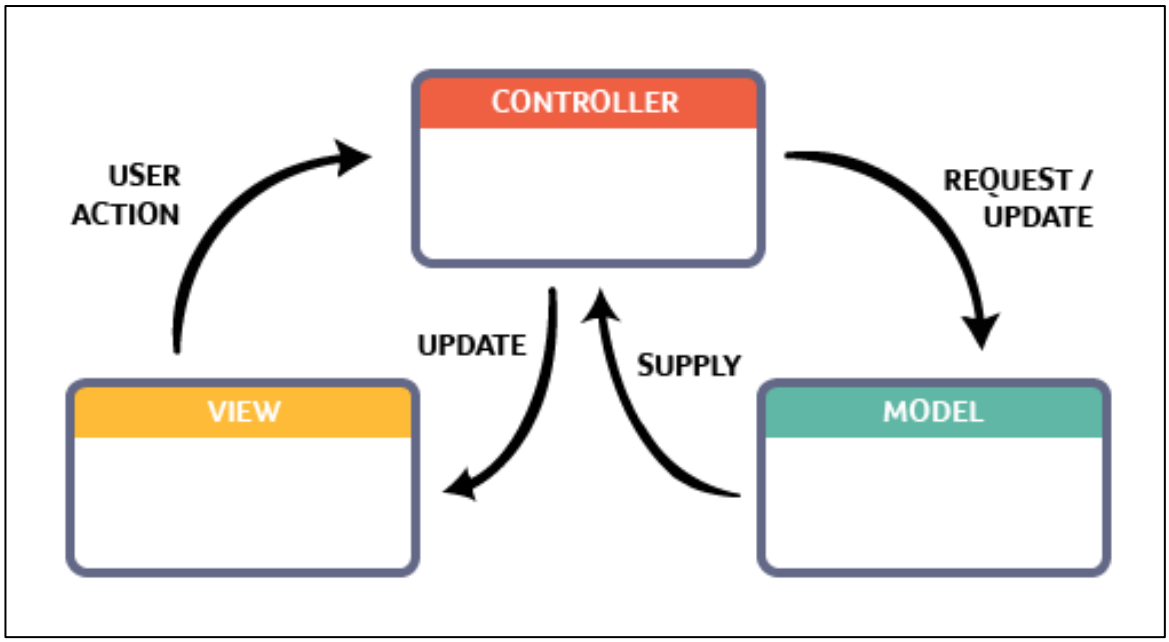


Рисунок 4.1 Взаємодія компонентів між собою за шаблоном MVC

Як модель було прийнято рішення використовувати ORM ActiveRecord.

ActiveRecord є вбудованим у фреймоврк рішенням, що відповідає за взаємодію з базою даних. Дана бібліотека полегшує роботу з сутностями, надаючи інтерфейс взаємодії, надає інструменти для версіонування структури БД, надає готвоі методі для валідації полів та багато іншого.

На рисунку 4.2 зображено приклад коду, що ілюструє створення міграції схеми бази даних.

```
class DeviseTokenAuthCreateUsers < ActiveRecord::Migration[5.2]
  def change
    create_table(:users) do |t|
      t.string :provider, :null => false, :default => "email"
      t.string :uid, :null => false, :default => ""
      t.string :encrypted_password, :null => false, :default => ""
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      t.boolean  :allow_password_change, :default => false
      t.datetime :remember_created_at
      t.string   :confirmation_token
      t.datetime :confirmed_at
      t.datetime :confirmation_sent_at
      t.string   :unconfirmed_email
      t.string :name
      t.string :nickname
      t.string :image
      t.string :email
      t.json :tokens

      t.timestamps
    end

    add_index :users, :email,                unique: true
    add_index :users, [:uid, :provider],     unique: true
    add_index :users, :reset_password_token, unique: true
    add_index :users, :confirmation_token,   unique: true
  end
end
```

Рисунок 4.2 Приклад міграції структури бази даних

Цей фрагмент відповідає за створення моделі «користувач» зі всіма необхідними полями для регістрації/авторизації.

Типи данних, що містить ця модель – строка, дата, JSON, логічний тип даних.

Також за допомогою міграцій можна задати індекси полям моделі. Наприклад, було проіндексовано поле «email» для більш ефективного пошуку за даним критерієм, оскільки така виборка будет здійснюватись часто.

Одним із потужних інструментів ActiveRecord є валідації полів моделі. Перевірка валідності здійснюється на рівні додатку, тим самим перешкоджаючи потрапляння помилкових даних до рівня СУБД. На рисунку 4.3 зображено приклад валідації в класі моделі « Box». Таким чином здійснюється перевірка на обов'язкову присутність значень у полей «secret_key» та «pin_num».

```
class Box < ApplicationRecord
  validates :secret_key, :pin_num, presence: true
end
```

Рисунок 4.3 Приклад коду, що реалізує валідації полів

Контроллером Rails виступають нащадки класу «ApplicationController».

Було прийнято рішення використовувати саме їх для реалізації REST API [12] системи.

Приклад такого контролеру зображено на расунку 4.4.

```
class BoxesController < ApplicationController
  def show
    @box = Box.find(params[:id])

    render json: @box, status: :ok
  end

  def update
    @box = Box.find(params[:id])

    @box.update!(locked: params[:locked])

    render json: @box, status: :ok
  end
end
```

Рисунок 4.4 Приклад коду, що реалізує контролер

Методи «show» та «update» відповідають за окремі методи HTTP, що належать ресурсу «boxes».

Наприклад метод «show» контролеру «Boxes» відповідає за ендпоінт «GET /boxes/:id» з параметром «id», а метод «update» - за ендпоінт «PATCH /boxes/:id» з параметром тим самим параметром.

Метод «params» повертає параметри, що прийшли в тілі запиту.

Метод «render» повертає у подання відповідь серверу на запит.

Оскільки серверне рішення являє собою API системи, то в якості представлення у моделі MVC виступає JSON, що віддає сервер.

До Rails можна під'єднати бібліотеку «ActiveModel Serializer», що формує відповідь серверу у вигляді JSON об'єкту запису, що повертається у методі контролеру.

Приклад коду, що ілюструє роботу даної бібліотеки зображено на рисунку 4.5.

```
class BoxSerializer < ActiveModel::Serializer
  attributes :id,
             :address,
             :ltd,
             :lng,
             :locked
end
```

Рисунок 4.5 Приклад реалізації перетворення екземпляру класу «Box» у JSON за допомогою бібліотеки «ActiveModel Serializer»

Таким чином задаються поля перетворюваного об'єкту, що слід повернути у відповіді серверу.

4.2 Клієнтський додаток

Для реалізації клієнтського додатку було обрано технологію Vue.js 2.0, що використовують для створення односторінкових веб рішень [16].

Даний фреймворк ідеально підходить для створення середніх за розміром додатків, він є реактивним, володіє чіткою конвенцією щодо створення компонент, має вбудовані інструменти для створення маршрутів, постійно розвивається та вдосконалюється.

За допомогою даного фреймворку значно полегшується робота з DOM-документом. Все це робить додатки на Vue.js зручними для подальшої підтримки та масштабування.

Також слід відзначити, що Vue.js є найпродуктивнішим рішенням серед аналогів. В порівнянні з React, Vue.js є більш потужним та надійним.

Vue.js може бути інтегрованим з Bootstrap для створення інтерфейсу. Bootstrap – набір шаблонів для створення інтерфейсу.

Іншими словами, Bootstrap - це бібліотека, що складається з готових елементів інтерфейсу на класами стилів CSS для його кастомізації. Елементи інтерфейсу у Bootstrap мають приємний вигляд. Також вони дуже різноманітні та сама їх кількість велика.

Через переваги цих двох технологій було вирішено використовувати їх у парі. Таким чином, в результаті було досягнуто того, що користувацький інтерфейс має приємний вигляд, працює надійно та швидко.

Компонента Vue.js складається з трьох основних структурних елементів:

- template – html розмітка компоненту;
- script –JavaScript код компоненти;
- style – стилі, що використовую html код.

Було прийнято рішення використовувати саме цю структуру, оскільки саме її рекомендують розробники фреймворку.

Приклад такої компоненти зображено на рисунку 4.6.

```
<template>
    <div class="container container-margin">
        <div class="row">
            <div class="col-3">
                <div class="box-shadow">
                    <form style="margin-bottom: 0px;">
                        <div class="form-group text-dark">
                            <b-form-group>
                                <b-form-radio size="lg" v-
model="selected_order_status" name="some-radios" value="all">All</b-form-radio>
...
                            </b-form-group>
                        </div>
                    </form>
                </div>
            </div>
...
        </div>
    </div>
</template>

<script>
    export default {
        name: "Orders",
        data: function () {
            return {
                selected_order_status: 'all',
                orders: [ ... ]
            }
        },
    }
</script>

<style scoped>
    .container-margin {
        margin-top: 96px;
        margin-bottom: 30px;
    }
...
</style>
```

Рисунок 4.6 Компонента «Orders»

У тег «template» вставляється звичайний html код, єдина вимога – кореневим елементом повинен бути лише один тег. Таке обмеження зроблене для того, щоб розмітку компонентів можна булу поєднувати між собою.Тег «script» зберігає в собі JavaScript код, до якого можна звертатися с html розмітки компоненту. Це можуть бути змінні, які прив’язуються до елементів інтерфейсу, об’єкти чи функції, що приймають аргумент та повертають оброблений результат. Тег «style»

містить в собі набор стилів, що використовуються html шаблоном цього самого компоненту. Таким чином ми отримали інтерфейс, зображений на рисунку 4.7.

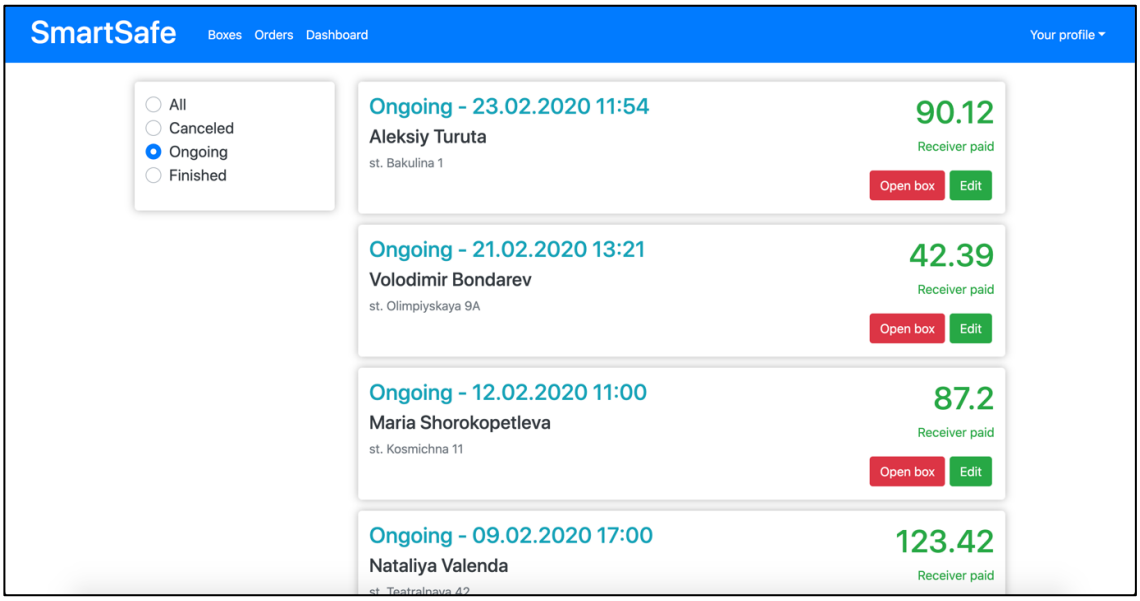


Рисунок 4.7 Сторінка замовлень

Отже отриманий приклад елементу інтерфейсу складається з меню навігації, у шапці додатку, панелі фільтрів збоку та самого списку замовлень. Демонстрація роботи вбудованої компоненти Google Maps зображена на рисунку 4.8.

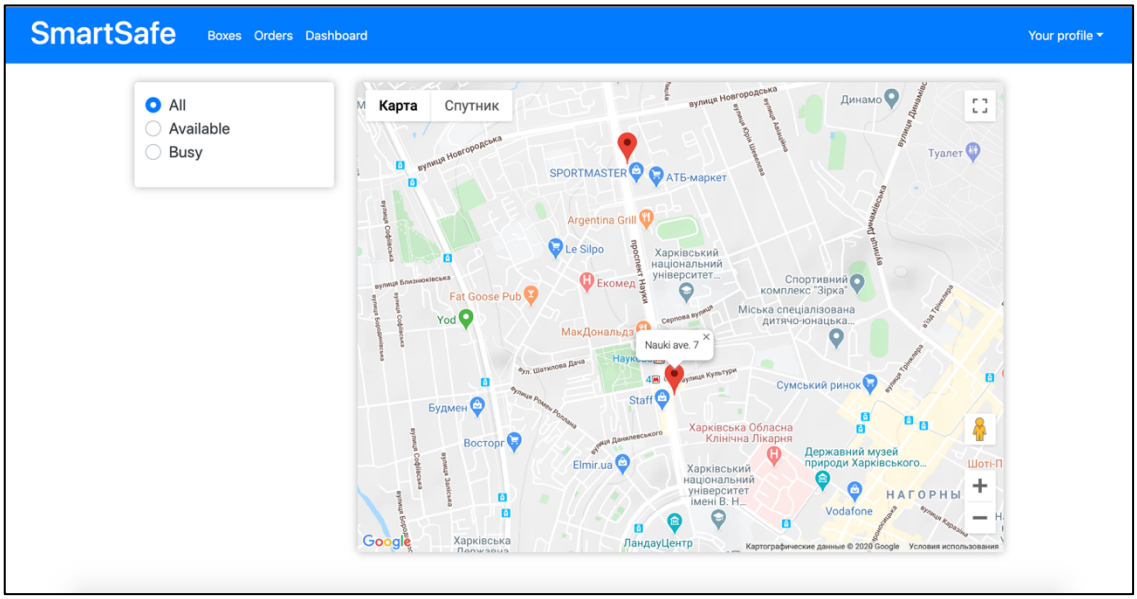


Рисунок 4.8 Демонстрація роботи вбудованої компоненти Google Maps

Таке легке підключення можливе завдяки модульній конструкції фреймворку, що є його вагомою перевагою.

4.3 Апаратна частина

Апаратною частиною є самі камери схову, які складаються з металевого каркасу (коробки) та електроніки. Для того, щоб камера відкривалась та закривалась по сигналу, було прийнято рішення використовувати звичайний електронний замок, котрих на ринку зараз дуже багато.

Принцип дії дуже простий, низька напруга – замок у закритому стані, напруга висока – замок відкривається.

Як уже було сказано раніше, для подання напруги на замок було використано мікрокомп’ютер RaspberryPI, оскільки це рішення є найбільш універсальним, простим у використанні, розгортанні, у подальшій єксплуатації. На рисунку 4.9 зображено зібраний прототип, що було отримано в ході виконання дипломної роботи.

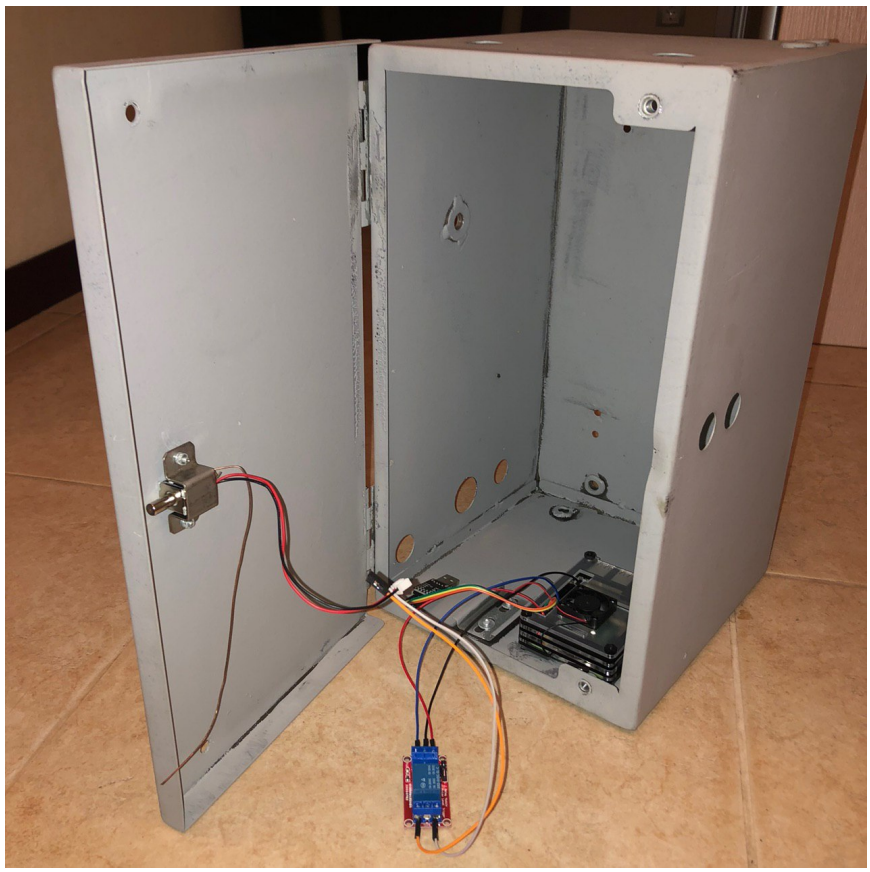


Рисунок 4.9 Прототип сейфу у стадії MVP

Отже як можемо бачити, замок прикріплений до двері сейфу, а самим замком керує мікрокомп’ютер.

4.4 Тестування розробленого програмного забезпечення

Було прийняте рішення тестувати API системи за допомогою автоматичних юніт-тестів.

Юніт-тест – це тест, за допомогою якого тестується невелика частина програми, частіше за все це один конкретний клас або маленька частини коду. Такий підхід тестування доцільний використанні автоматичних тестів, оскільки вразі негативного результату тесту буде просто знайти помилку.

Наявність автоматичних тестів надзвичайно важлива для довготривалої підтримки та розширення функціоналу системи.

Найпоширенішим інструментом для створення автоматичних тестів в екосистемі Ruby on Rails є бібліотека Rspec. На рисунку 4.10 зображено приклад автоматичних тестів API системи.

```
require 'rails_helper'

describe BoxesController, type: :controller do
  describe 'GET /boxes/:id' do
    let(:box) { create :box, :locked }

    it 'returns status of the box' do
      get :show, params: { id: box.id }

      expect_status(200)
      expect_json(id: box.id, locked: true)
    end
  end

  describe 'PATCH /boxes/:id' do
    let(:box) { create :box, :locked }

    it 'updates state if the box of the box' do
      patch :update, params: { id: box.id, locked: false }

      expect_status(200)
      expect(box.reload.locked).to eq(false)
      expect_json(id: box.id, locked: false)
    end
  end
end
```

Рисунок 4.10 Тестування API за допомогою бібліотеки Rspec

Як можемо бачити, тестується кожен ендпоінт API, приймаючи параметр на та перевіряється резутьтат, що він повертає. Існують також тесті на моделі системи, сервіси, на окремі класи та модулі. Їх також доцільно використовувати залежно від ситуації.

Але тестування автоматичними тестами API не гарантує коректної роботи системи в цілому. Необхідно також впровадити тестування інтеграції серверною частини с клієнтською.

Для цього було використано метод мануального тестування моделюючи поведінку користувача системи.

Мануальне тестування – тестування програмного забезпечення без використання автоматичних засобів.

Такий вид тестування є найпоширенішим. Звісно що швидкість такого методу є меншою, ніж у автоматичного тестування, але це самий надійний засіб та найбільш наближений до того, як системою користуються в реальному житті.

Також під час мануального тестування системи, а особливо під час мануального регресійного тестування, найпростіше за все помітити недоліки, помилки та невраховані кейси після розробки, оскільки тестування відбувається не окремих програмних компонент чи невеликою сукупності компонент, а того, як ці сукупності компонент поводять себе в системі.

Оскільки розроблена система не є великою, то регресійне тестування її функціоналу не займає значну кількість часу.

Також через невеликий розмір системи не є доцільним документувати всі можливі кейси, вони є інтуїтивно зрозумілими та логічними.

ВИСНОВКИ

В результаті дипломної роботи була спроектована архітектура програмно-апаратної системи, що представляє з себе розумні камері схову, що керуються з веб-додатку, які доцільно використовувати для обміну товарами між споживачами.

Дана система дозволяє автоматизувати обмін та видачу товарів, а також надати інтерфейс для адміністрування власникам подібних точок видачі, що складаються з даних камер схову.

Також було проаналізовано доцільність надання можливості аутетифікації за обличчям. В роботі були порівняні класичні методи розпізнавання обличчя та методи з використанням глибоких нейронних мереж. Результати досліджень вкахують, що глибокі нейронні мережі дають кращі результати по цільовим метрикам розпізнавання обличчя. Таким чином для програмної системи «Інтелектуальний сейф» була використана нейронна мережа архітектури FaceNet.

На наборі даних Labeled Faces in the Wild модель досягає точності класифікації 98,87% з 0,15% стандартної помилки. Це зменшує частоту помилок, про які повідомляє DeepFace, більш ніж у 7 разів, а інші найсучасніші DeepId – на 30%. При наборі даних Youtube Face повідомляється про точність 95,12% зі стандартною помилкою 0,39, використовуючи перші 100 кадрів. Це краще, ніж 91,4% точність, запропонована DeepFace, і 93,5% від DeepId на 100 кадрах.

Розробка серверної частини системи була здійснена за допомогою фреймворку Ruby on Rails, доступом до сейфу керує мікрокомп’ютер RapberryPI, як найбільш універсальний з доступних на ринку. Обрана СУБД – PostgersQL. Vue.js було обрано для реалізації для веб-частини системи.

Був проведений аналіз предметної області та на його основі розроблені бізнес вимоги. У якості специфікації програмного продукту було складено концептуальний документ. За функціональними вимогами було складено моделі поводження користувачів у вигляді діаграм UseCase. Також була спроектована та

описана архітектура майбутньої системи, частини якої мають можливість взаємодіяти за допомогою архітектурного шаблону REST [12].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. У СЕРЕДНЬОМУ КОЖНІ ТРИ ГОДИНИ В УКРАЇНІ ГИНЕ ЛЮДИНА В ДТП. СТАТИСТИКА АВАРІЙ – ТСН. Служба новин – [Електронный ресурс] – Режим доступу: https://tsn.ua/ukrayina/u-serednomu-kozhni-tri-godini-v-ukrayini-gine-lyudina-v-dtp-statistika-avariy-1392594.html
2. Ваши посылки тоже швыряют, смиритесь – Newslab. Служба новин – [Электронный ресурс] – Режим доступу: https://newslab.ru/article/837080
3. COVID-19 CORONAVIRUS PANDEMIC – WORLDOMETER. Сайт статистики – [Електронный ресурс] – Режим доступу: https://www.worldometers.info/coronavirus/
4. Коронавірус в Україні – Кабінет Міністрів України. Інформаційний сайт – [Электронный ресурс] – Режим доступу: https://covid19.gov.ua/
5. Огляд українських служб доставки – Хорошоп. Інформаційний портал – [Электронный ресурс] – Режим доступу: https://horoshop.ua/ua/blog/obzor-ukrainskikh-sluzhb-dostavki/
6. Паттерсон А., Хеннесси Л. Компьютерная архитектура. Количественный подход. ТЕХНОСФЕРА Москва – 2016. – 936 с.
7. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ СТУДЕНТОВ КАФЕДРЫ АСОИУ – Сетевые технологии – [Електронный ресурс] – Режим доступу: http://www.4stud.info/networking/lecture5.html
8. Википедия. Діаграма компонентів – Wikipedia – [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_компонентів/
9. Побудова додатків баз даних в архітектурі «клієнт-сервер» – buklib – [Електронний ресурс]– Режим доступу: https://buklib.net/books/23148/
10. HTML 5 book: CSS and CSS3 – html5book – [Електронний ресурс] – Режим доступу: https://html5book.ru/css-css3/
11. Википедия. Діаграма компонентів – Wikipedia – [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_компонентів/

12. REST Architectural Constraints – REST – [Електронний ресурс] – Режим доступу: https://restfulapi.net/rest-architectural-constraints/

13. Мацумото Ю., Флэнаган Д. Язык программирования Ruby. Питер, 2011. – 496с.

14. Википедия. Авторизація – Wikipedia – [Електронний ресурс] – Режим доступа: www/URL:https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D

15. Raspberry Pi OS - RaspberryPI ORG – [Електронний ресурс] – Режим доступу: https://www.raspberrypi.org/downloads/

16. Єрохін А.Л, Турута А.П., Оценки эффективности работы компонент web – Харків. // Національний технічний університет «Харківський політехнічний інститут», 2007 – 4с.

17. Черепанова Ю.Ю., Широкопетлєва М.С., Деякі аспекти автоматизованого тестування знань мови SQL. – Варшава. // Diamond trading tour, 2018. 4с.

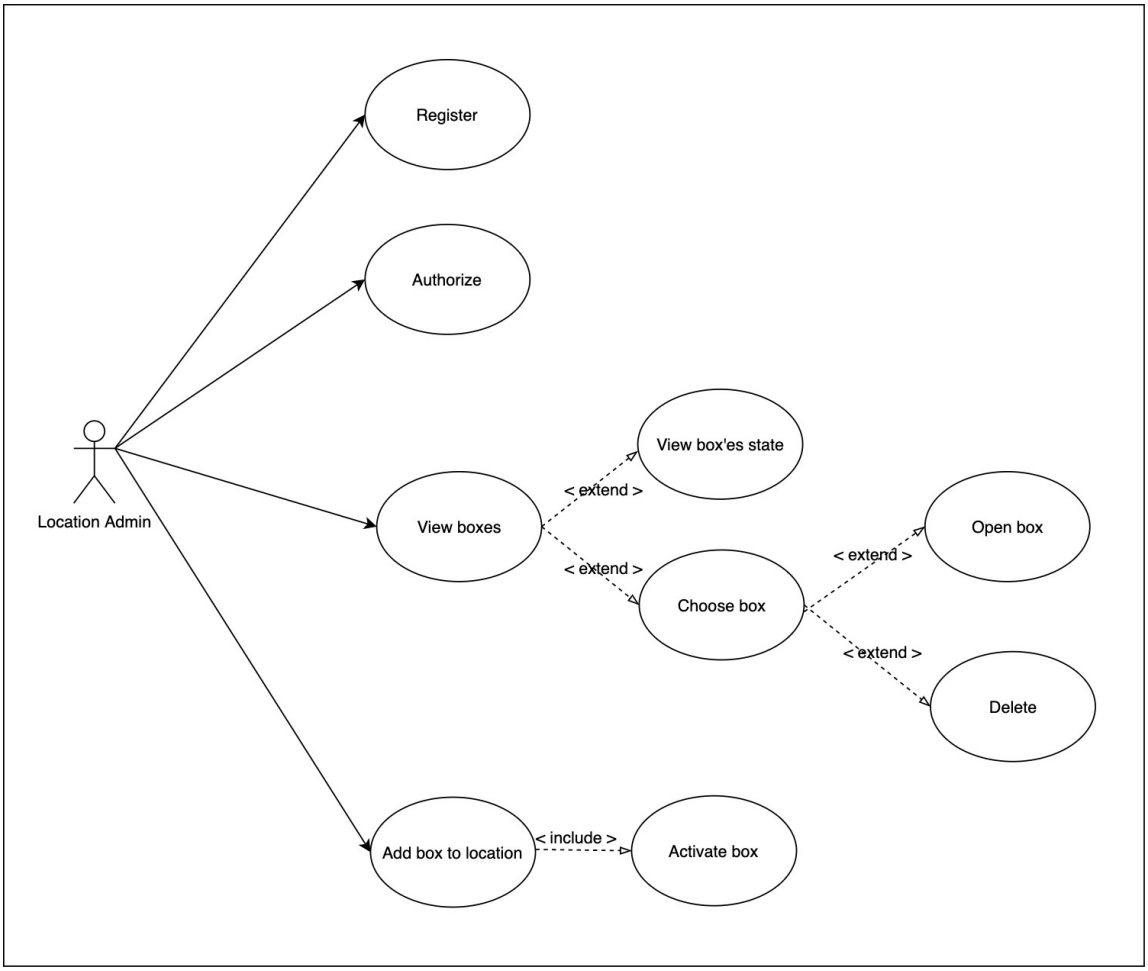
18. Джебара Т., Оцінка 3D-пози та нормалізація для розпізнавання обличчя // Центр інтелектуальних машин, Університет Макгілла – 1996. 68с.

19. Горпаде, Сантаджі, Джейшрі Горпад, Шамла Мантрі. Розпізнавання букв за допомогою нейронних мереж // Міжнародний журнал комп'ютерних наук і технологій (IJCST) - 2010, с.92-98.

20. Кюнгім Б.. Дрейпер Б. А, PCA проти ICA: порівняння набору даних FERET // Матеріали 6-ї спільної конференції з інформаційних наук (JCIS) - 2002, с. 824- 827.

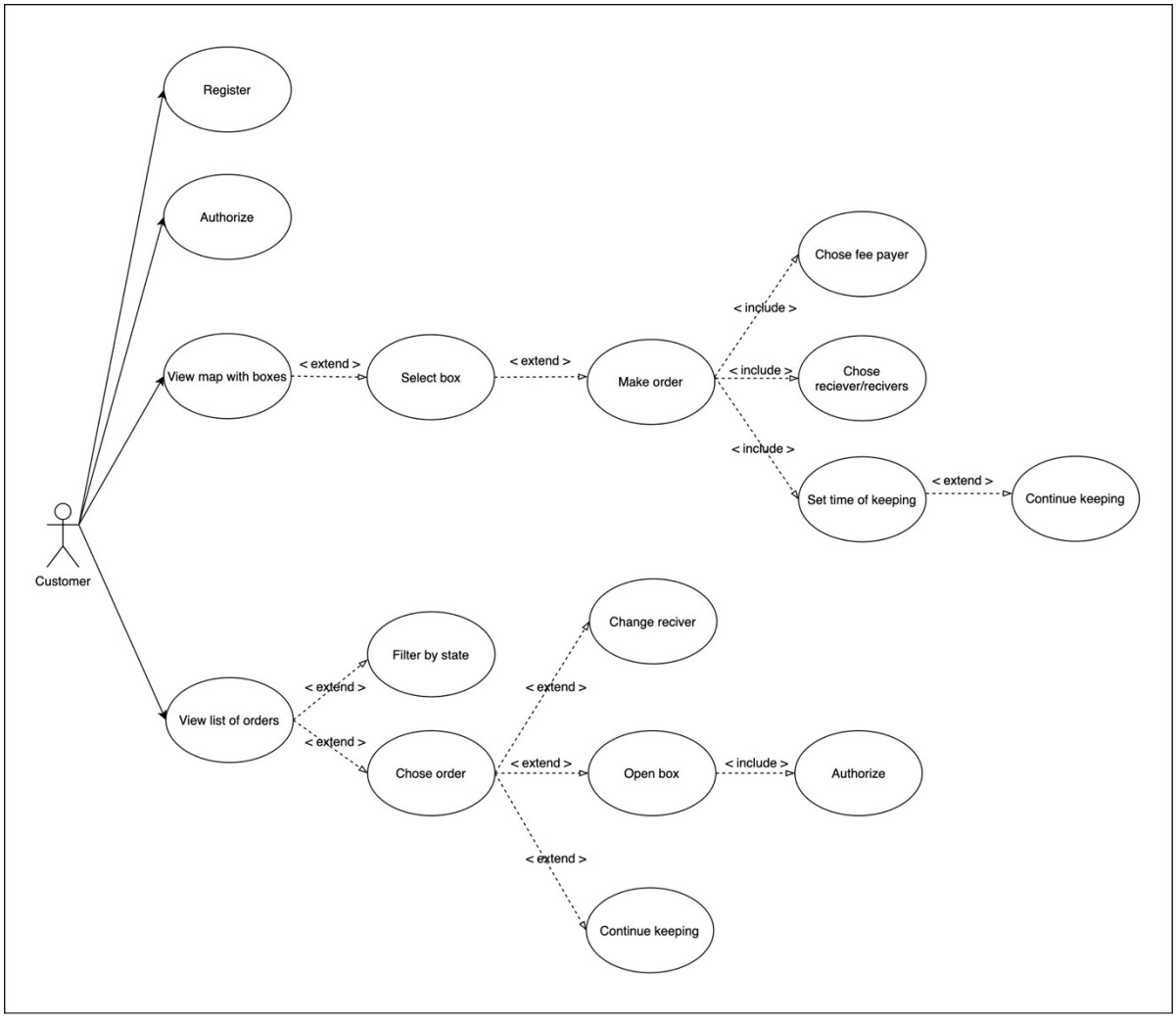
ДОДАТОК А

Use-case діаграма актора-адміністратора, що керує локацією



ДОДАТОК Б

Use-case діаграма звичайного користувача системи (споживача).

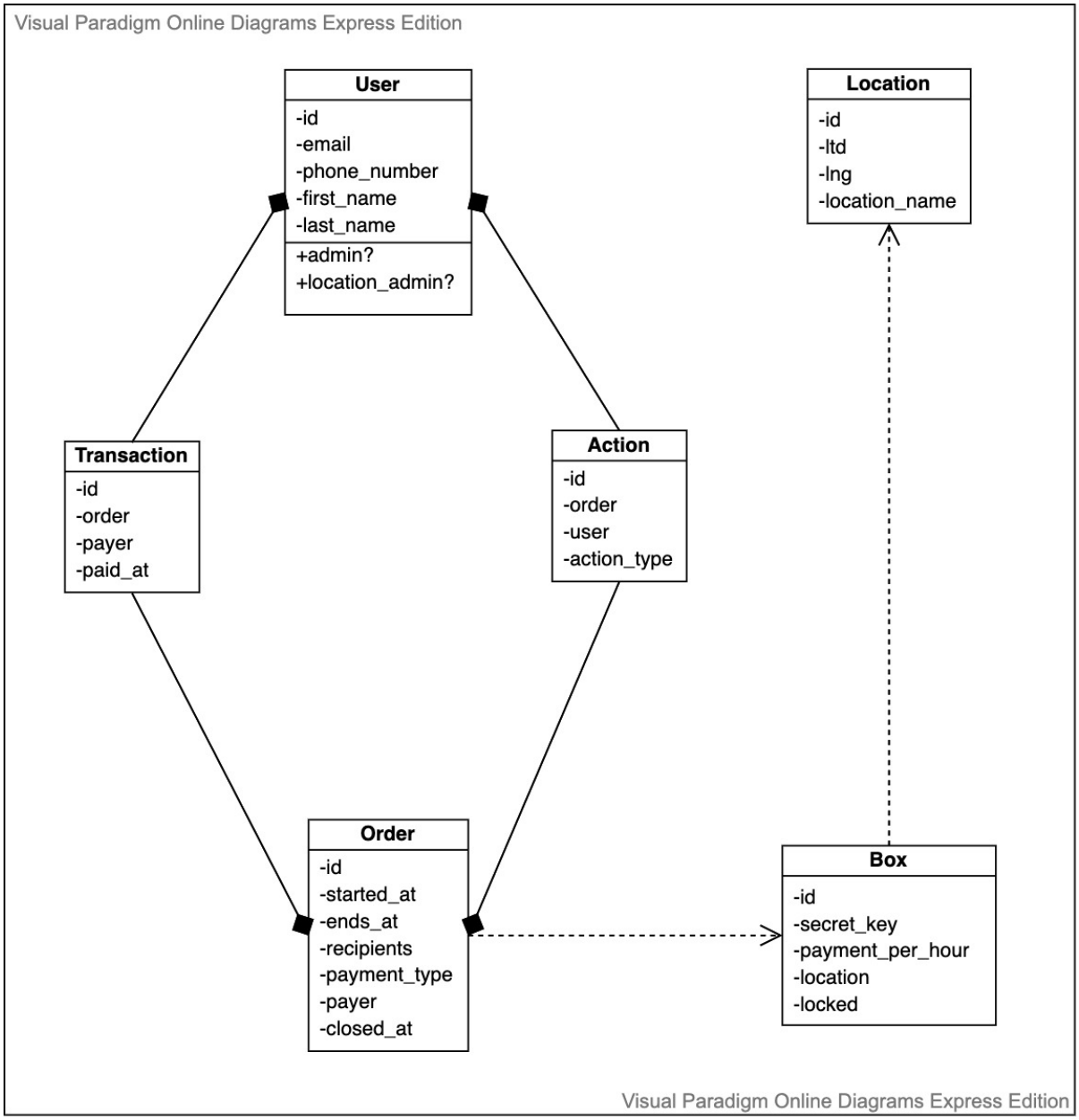


ДОДАТОК В

Use-case діаграма адміністратора системи.

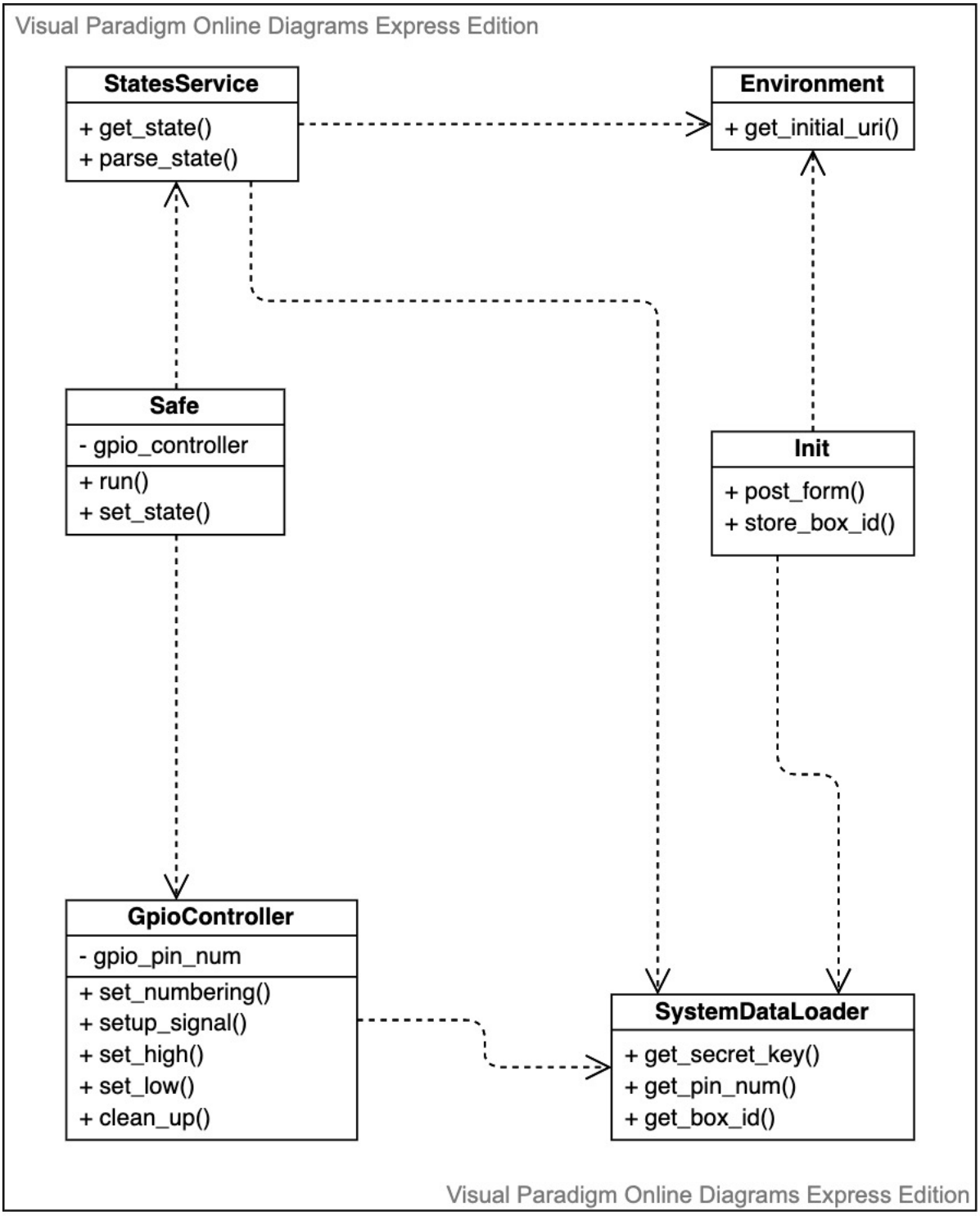
ДОДАТОК Г

Діаграма основних класів веб-додатку



ДОДАТОК Ґ

Діаграма класів застосування, що керує пристроями (камерами схову)



ДОДАТОК Д

Приклади коду програм

```
require 'rails_helper'

describe BoxesController, type: :controller do
  describe 'GET /boxes/:id' do
    let(:box) { create :box, :locked }

    it 'returns status of the box' do
      get :show, params: { id: box.id }

      expect_status(200)
      expect_json(id: box.id, locked: true)
    end
  end

  describe 'PATCH /boxes/:id' do
    let(:box) { create :box, :locked }

    it 'updates state if the box of the box' do
      patch :update, params: { id: box.id, locked: false }

      expect_status(200)
      expect(box.reload.locked).to eq(false)
      expect_json(id: box.id, locked: false)
    end
  end
end
class BoxesController < ApplicationController
  def show
    @box = Box.find(params[:id])

    render json: @box, status: :ok
  end

  def update
    @box = Box.find(params[:id])

    @box.update!(locked: params[:locked])

    render json: @box, status: :ok
  end
end
create_table "users", force: :cascade do |t|
  t.string "provider", default: "email", null: false
  t.string "uid", default: "", null: false
  t.string "encrypted_password", default: "", null: false
  t.string "reset_password_token"
  t.datetime "reset_password_sent_at"
  t.boolean "allow_password_change", default: false
  t.datetime "remember_created_at"
  t.string "confirmation_token"
  t.datetime "confirmed_at"
  t.datetime "confirmation_sent_at"
  t.string "unconfirmed_email"
  t.string "name"
```

```
  t.string "nickname"
  t.string "image"
  t.string "email"
  t.json "tokens"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["confirmation_token"], name: "index_users_on_confirmation_token",
unique: true
  t.index ["email"], name: "index_users_on_email", unique: true
  t.index ["reset_password_token"], name: "index_users_on_reset_password_token",
unique: true
  t.index ["uid", "provider"], name: "index_users_on_uid_and_provider", unique:
true
end
<template>
    <div class="container container-margin">
        <div class="row">
            <div class="col-3">
                <div class="box-shadow">
                    <form style="margin-bottom: 0px;">
                        <div class="form-group text-dark">
                            <b-form-group>
                                <b-form-radio size="lg" v-
model="selected_box_status" name="some-radios" value="all">All</b-form-radio>
                                <b-form-radio size="lg" v-
model="selected_box_status" name="some-radios" value="available">Available</b-
form-radio>
                                <b-form-radio size="lg" v-
model="selected_box_status" name="some-radios" value="busy">Busy</b-form-radio>
                            </b-form-group>
                        </div>
                    </form>
                </div>
            </div>
            <div class="col-9">
                <div class="box-shadow" style="padding: 0px">
                    <GmapMap
                            :center="startLocation"
                            :zoom="15"
                            map-type-id="terrain"
                            style="width: 100%; height: 600px; border-radius:
2rem;"
                    >
                        <GmapInfoWindow :options="infoOptions"
:position="infoPosition" :opened="infoOpened" @closeclick="infoOpened=false">
                            {{infoContent}}
                        </GmapInfoWindow>
                        <GmapMarker
                                v-for="(item, key) in coordinates"
                                :key="key"
                                :position="getPosition(item)"
                                :clickable="true"
                                @click="toggleInfo(item, key)"
                                :draggable="true"
                        />
                    </GmapMap>
                </div>
            </div>
        </div>
    </div>
</template>
```

```
<script>
    export default {
        name: "Boxes",
        data: function () {
            return {
                name: 'map',
                selected_box_status: 'all',
                filter_canceled: false,
                filter_ongoing: false,
                filter_finished: false,
                startLocation: { lat: 50.014377, lng: 36.228190 },
                coordinates: {
                    0: {
                        address: 'Nauki ave. 21',
                        lat: '50.019743',
                        lng: '36.225261'
                    },
                    1: {
                        address: 'Nauki ave. 7',
                        lat: '50.011611',
                        lng: '36.227864'
                    }
                },
                infoPosition: null,
                infoOpened: false,
                infoContent: null,
                infoOptions: {
                    pixelOffset: {
                        width: 0,
                        height: -35
                    }
                }
            }
        },
        methods: {
            getPosition: function(marker) {
                return {
                    lat: parseFloat(marker.lat),
                    lng: parseFloat(marker.lng)
                }
            },
            toggleInfo: function(marker, key) {
                this.infoPosition = this.getPosition(marker)
                this.infoContent = marker.address
                if (this.infoCurrentKey == key) {
                    this.infoOpened = !this.infoOpened
                } else {
                    this.infoOpened = true
                    this.infoCurrentKey = key
                }
            }
        },
        created: function() {
            // let map = document.createElement('script');
            //
map.setAttribute('src',"https://maps.googleapis.com/maps/api/js?key=AIzaSyB5lBcIrf
BNfy2doF617jmBi18WWWemeVM");
            // document.head.appendChild(map);
        },
        components: {}
    }
</script>
```

```
<style scoped>
    .container-margin {
        margin-top: 96px;
        margin-bottom: 30px;
    }

    .box-shadow {
        border-color: white !important;
        box-shadow: 0 0 10px #b2b2b2; /* Параметры тени */
        padding: 14px;
        border-radius: 0.3rem;
        margin-bottom: 14px;
    }

    table {
        border-color: white;
    }

    .full-sized {
        width: 100%;
    }
</style>
```

ДОДАТОК Е

KNU

Інформаційно-апаратна система “Smart Safe”
Software Architecture Document (SAD)

CONTENT OWNER: Radchenko Mykyta

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1.0	• 1.0	• 21.04.2022

Table of Contents

1. Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 ("Document Management and Configuration Control Information") explains revision history. This tells you if you're looking at the correct version of the SAD.
- Section 1.2 ("Purpose and Scope of the SAD") explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.
- Section 1.3 ("How the SAD Is Organized") explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 ("Stakeholder Representation") explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 ("Viewpoint Definitions") explains the viewpoints (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 ("Views"). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.

- Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1. Document Management and Configuration Control Information

Revision Number: <<1.0 >>

Revision Release Date: <<1.0 >>

Purpose of Revision: << 21.04.2022>>

Scope of Revision: basic information about system

1.2. Purpose and Scope of the SAD

This SAD specifies the software architecture for **SmartSafe** system**.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means

.

that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work

assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 2.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3. Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471]. As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A view is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system. The remainder of Section 1.5 defines

the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoints that apply to that class of stakeholders concerns
Radchenko Mykyta	Student and engineer

1.3.1. "Student and engineer" Viewpoint Definition

1.3.1.1. Abstract

This viewpoint summarizes the views of a student developing this project and their concerns.

1.3.1.2. Stakeholders and Their Concerns Addressed

The main concern of a student is to get the highest grade possible

1.3.1.3. Language(s) to Model/Represent Conforming Views

Technologies that were used: Ruby, Ruby on Rails, Javascript, Vue.js, HTML&CSS, PostgresQL, RaspberryPI3

2. Architecture Background

2.1. Problem Background

Nowadays, you can buy almost anything on the Internet - from a mobile phone or a bottle of milk to an evening dress for a holiday party. Therefore, the question of how to deliver the purchased product to the consumer who purchased it is extremely important.

Also an urgent problem is the transfer of some things between two people. For example, your friend left a hat at home, but in the near future you do not have the opportunity to meet and pass it to her. Most likely in such a situation you will use the services of one of the delivery services.

There are several well-known tools:

- delivery by mail;

- courier delivery

- self-pickup from one of the outlets of the store (if it is a purchase).

But they all have their advantages and disadvantages.

Most often, the consumer of such services chooses the type of delivery depending on their preferences.

What the end user needs:

- timely delivery;

- reliability (no one wants to get spoiled goods);

- spend as little effort and time as possible to give / receive the goods;

- spend as little money as possible.

As we can see, the following basic concepts appear in this subject area:

- courier

- order;

- cargo;

- consumer;

- sender;

- recipient;

- location;

- time of transfer / delivery;

- things that deliver.

Stages that every consumer of mail services goes through, namely:

- get to the post office;

- draw up a parcel and hand it over to the postmen;

- transportation of the parcel to the distribution warehouse;

- transportation of the parcel to the recipient's office;

- get to the receiving office;

- registration of receipt of the parcel.

Stages that every consumer of courier services goes through, namely:

- search for a courier or courier service;

- registration of delivery;

- transfer of the parcel to the courier;

- parcel transportation by courier;

- registration of receipt of the parcel;

- transfer of the parcel to the recipient.

Stages that each consumer of goods goes through if he chooses the service of self-pickup from the point of sale, namely:

- registration of self-removal;

- get to the store store;

- receipt receipt.

Currently in Ukraine the sphere of delivery is quite developed, there are many significant players in this market.

For example, there are about ten well-known postal services, many courier, and some brands even have their own couriers. Many brands have enough offices where consumers can pick up the purchased goods, but there are drawbacks in the tools described above.

Each of the available types of delivery has its drawbacks.

Delivery by mail takes a lot of time, in particular when it comes to delivery of goods within the city, or even one area, you still need to spend effort and time to take the goods to the branch, to stand in line, which from time to time is considerable, to issue order.

Only after that the goods get to the warehouse where it is distributed among carriers.

The next stage is the transportation of goods to the office specified by the sender.

It should be borne in mind that the recipient must also spend his time to get to the office, stand in line and only then pick up your thing.

Upon receipt of the parcel, it is often necessary to have identity documents.

Thus, the shipment reaches the recipient in an average of one or two days.

From my own experience, I can say that sometimes you have to spend about 15 minutes waiting in line at the post office cashier.

At the same time about 5 minutes are spent on registration of delivery.

For example, the nearest post office from my residence is a 5-minute walk away.

In total, it takes 25 minutes to send a parcel, too large a figure for such a simple action.

Today in Ukraine there are several major players in the mail delivery market. These are Nova Poshta, Ukrposhta and Intime.

According to user reviews [2], the service "New Mail" is quite fast, has a wide network of branches, convenient automation and integration with various services.

But almost half of the consumers of the company's services said that this service has a relatively high cost of delivery.

Ukrposhta, on the other hand, has a low price, but a quarter of respondents complained about poor service, slow delivery and an inconvenient online office.

Intime, according to those surveyed, offers services at a low price, but cargo is often delivered carelessly and offices are not conveniently located.

Delivery by courier often takes less time, but the price is higher.

In this way, several stages that are present in the delivery by mail disappear at once - namely, its registration, movement to the branch, transportation of the parcel to the distribution warehouse.

But keep in mind that the courier will deliver the parcel when the recipient's location arrives, it will not always be convenient for the person if he had his plans at that time.

Delivery by courier is vulnerable to the situation on the city roads, if the city is congested, or the courier gets into trouble, for example, in an accident - the receipt of the order is delayed.

This scenario is likely, because according to statistics published by the information channel TSN [3], in the period from January to August 2019 in Ukraine there were 86.7 thousand accidents, ie an average of 409 daily, or one accident each 3-4 minutes.

Self-pickup from one of the store's outlets is often a free service, but not always the brand has a representative office in a convenient place for you.

Thus, if you live in an average-sized settlement, such as Kharkiv, the road from one end of the city to the other (to the potential location of the store) can take about one hour.

Also, do not forget that in this way you will spend time defending the queue to receive the order, and the actual receipt.

Some postal companies have a service to send a parcel through a post office. ATM is an electronic software and hardware complex with a large number of cells, a self-service terminal, designed for automatic operations of receiving and issuing parcels: the courier delivers the parcel to the ATM, and the recipient picks it up at a convenient time. It is a very convenient function, but currently there are not enough of them in Ukraine, which means that it will be inconvenient to get these post offices from many districts of the city.

Also, the disadvantage of this implementation is that all these ATMs serve only one delivery service, ie it is impossible to buy them, for example, a supermarket, and use them for their own purposes.

It is also impossible to use them without delivery, but only as a storage room (remember the example of a friend who left your hat at home).

The issue of product safety should be borne in mind. It is impossible to allow the goods to reach the consumer damaged, or disappear completely.

There were cases when during the delivery of mail things were lost due to imperfect processes or errors of employees, or they were stolen.

For example, if you believe the response of one of the employees of the post office [4], the disappearance of things from the parcel is quite common.

Moreover, the transportation processes are far from ideal, no one cares about the integrity of the goods inside the box, try to complete the transportation as soon as possible, which means that your parcel can simply be thrown from hand to hand.

This question becomes more fundamental when there is something fragile inside the box, for example - a crystal vase.

All the described methods of delivery, except for self-pickup, often require additional payment. It happens that the price rises significantly.

Post offices, couriers, sales outlets and even post offices all work at a pre-arranged time.

For example, if it is convenient for you to pick up the purchased goods at night - you will not be able to do it, because all these institutions work only during the day.

Also important is the fact that delivery by mail, courier or pickup requires direct contact with people. During epidemics, contact becomes dangerous because it contributes to the spread of various viruses.

For example, in the situation of the COVID-19 pandemic, when according to statistics published by the resource "WORLDOMETER" [5], as of May 21, 2021, 167,857,345 cases of the disease were detected, this problem becomes extremely relevant.

According to the Ministry of Health of Ukraine [6], the new virus is spread by airborne droplets. This means that you need to stay away from a person who is a potential carrier of the virus at a distance of at least one meter - this is too much to send a parcel and issue a receipt. It is also dangerous for the couriers themselves, as they need to meet and communicate with a huge number of people every day.

It is now known that COVID-19 also lives on surfaces for about 3 hours. This means that there is a possibility of infection through the transfer of various items, including parcels.

Thus, it can be noted that each of the available types of delivery has its drawbacks.

2.1.1. System Overview

The solution - a software and hardware system "Smart Safe".

This system will automate the exchange and delivery of goods between consumers through storage rooms with controlled access to them, thus providing an alternative to existing means of delivery.

2.1.2. Goals and Context

The task of this thesis is to implement a system that will have the following advantages over other competitors:

- to eliminate the need to stand in line. Enable users to view free storage rooms and book them for a specific time;

- delivery through the web application by users themselves, which is freely available via the Internet, which will save time on registration of the shipment;

- eliminating the need for the presence of an employee when using the system;

- the ability to assign or change the user during the existence of the order, which will pick up the goods (recipient);

- authorization only within the system, you do not need to verify your identity with official documents;

- access to the functionality of the system regardless of the time of day;

- scalability of the system for the possibility of expanding the presence in as many points with a group of cameras in the city;

- eliminate or minimize the need to service this system by people, thereby reducing the final cost of services;

- the ability to integrate other institutions into the system;

- the possibility of using the system by other institutions;

- a convenient web resource with a nice design;

- guaranteeing the safety and integrity of goods stored in the middle of the storage room;

- the possibility of receiving the parcel immediately after it was placed in the storage room. It is also necessary to provide the opportunity to use the system and pick up the goods at any time during the day;

- to exclude the possibility of the storage room to be opened by mistake, not by that person.

Thus, we have a system consisting of two parts - hardware (storage chamber) and software shell for users, through which they will book these cameras, open and close depending on the selected scenario.

The hardware must be controlled from a microcomputer to reduce the final cost of the output.

It is necessary to implement the process of the first launch of the storage room as simple as possible, to make it so that it can be performed by representatives of other companies.

To enable to have several roles in the system with different powers:

- ordinary consumers who can put / pick up the goods from the storage room;

- administrator who has the ability to open / close the camera;

- The administrator, who allows the boxes that have appeared in the system to start fulfilling orders, edits information about them if necessary, removes the boxes from the system.

The sender should be able to see which storage rooms are currently available.

You also need to be able to choose who will pick up the goods placed in the camera. You can choose yourself as the recipient.

You need to be able to choose who will pay for the services provided by the system - the recipient or the sender.

The final cost should depend on the duration of storage.

It is necessary to implement a convenient mechanism for opening / closing the storage chamber, as well as a reliable authorization mechanism so that attackers can not steal the thing inside.

Thus, a problem was formed that needs to be solved in the course of this thesis.

2.1.3. Significant Driving Requirements

The system being developed includes three main groups of users with different roles and access rights:

- end users (ordinary users of the system);

- system administrators;

- Administrators who manage locations.

The main functions available to the actor-administrator:

- registration;

- authorization;

- view the list of storage cameras attached to its location;

- view the status of storage rooms (its current status);

- select a storage camera from the list to delete or open;

- add a new storage camera to your own location;

- have access to statistics of the most loaded locations.

An example of a location administrator is a security guard in a supermarket who decided to put smart storage cameras instead of the old, analog ones we are used to.

The main functions available to the average user of the system (consumer):

- registration;

- authorization;

- view the map with free storage cameras;

- select the camera;

- issue a deposit;

- choose a service payer (pay yourself or pay the recipient);

- select a recipient, multiple users can access the camera at once;

- set the time at which the product will be placed in the camera;

- continuation of storage of goods;

- view the list of orders;

- filtering orders by their status;

- choose an order;

- change the recipient;

- open the camera;

- continue storage.

The system administrator is the strongest person in the system in terms of authority.

This type of user can view absolutely all storage cameras available in the system, view their status, remove from the system or edit information about them.

It is also the administrator who confirms the new storage rooms and allows them to function fully, to fulfill consumer orders.

The administrator also has access to a list of all orders, and he can open any storage room.

With a regular administrator, a registered system user can become an administrator assigned to a location.

The location administrator must always be assigned to at least one location. To do this, select one of the available locations. This way, this user will be able to administer the selected location.

You can assign multiple locations to a single location administrator.

The main functions available to him:

- authorization [7];

- view a list of all storage cameras;

- view the status of the storage chamber;

- select a storage camera to remove it from the system or edit information about it;

- confirm the new storage room;

- view the list of orders;

- select an order to be able to open the storage room that performs it;

- add a location administrator;

- view the list of all administrators assigned to the location;

- Select one administrator to view a list of all their locations, delete a location, or add a new one.

Thus, the basic functional requirements for the system were formed, on the basis of which the user interface was designed.

2.2. Solution Background

2.2.1. Architectural Approaches

Figure 1 shows a system deployment diagram.

Thus, it was decided that the system will consist of 5 components:

- the main server that serves the web client;

- server that serves storage cameras;

- databases common to both servers;

- web client, through which consumers and administrators interact with the system;

- the client who controls the devices (storage cameras).

Figure 2 shows a diagram illustrating the interaction between all 5 software components [8] of the system.

A web application through which consumers and administrators interact with the system interacts only with the main app server.

Storage cameras interact with their separate server (Hardware server).

This implementation is motivated by the fact that the system provides a large number of storage cameras that should send requests with a significant frequency [9].

To distribute the load, it was decided to divide the server part of the system by purpose [10].

The master server and the server that serves the storage cameras have a common database.

Thus, there is no need to interact between them directly if the state of the storage chamber changes.

2.2.2. Analisys Results

Based on similar projects and general project background, upper mentioned architecture is deemed as an optimal solution and fit for this purpose.

2.2.3. Requirements Coverage

Following this architectural plan, we'll cover most of our needs for now.

3. Referenced Materials

Bass 2003	Bass, Clements, Kazman, Software Architecture in Practice, second edition, Addison Wesley Longman, 2003.
Хорошоп	Огляд українських служб доставки – www/ URL:

	https://horoshop.ua/ua/blog/obzor-ukrainskikh-sluzhb-dostavki/ – 09.05.2020
ТСН. Служба новин	У СЕРЕДНЬОМУ КОЖНІ ТРИ ГОДИНИ В УКРАЇНІ ГИНЕ ЛЮДИНА В ДТП. СТАТИСТИКА АВАРІЙ – www/ URL: https://tsn.ua/ukrayina/u-serednomu-kozhni-tri-godini-v-ukrayini-gine-lyudina-v-dtp-statistika-avariy-1392594.html – 09.05.2020
Newslab	Ваши посылки тоже швыряют, смиритесь - www/ URL: https://newslab.ru/article/837080 – 09.05.2020
WORLDOMETER	COVID-19 CORONAVIRUS PANDEMIC – www/ URL: https://www.worldometers.info/coronavirus/ – 09.05.2020
Кабінет Міністрів України. Інформаційний сайт	Коронавірус в Україні – www/ URL: https://covid19.gov.ua/ – 09.05.2020
Википедия. Авторизація	www/URL:https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D – 09.05.2020
Побудова додатків баз даних в архітектурі «клієнт-сервер»	Побудова додатків баз даних в архітектурі «клієнт-сервер» www/URL: https://buklib.net/books/23148/ – 09.05.2020
УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ СТУДЕНТОВ КАФЕДРЫ АСОИУ	Сетевые технологии – www/ URL: http://www.4stud.info/networking/lecture5.html – 09.05.2020
Компьютерная архитектура. Количественный подход	[Text] / А. Паттерсон, Л. Хеннесси // ТЕХНОСФЕРА – 2008. – Vol. 5, Issue 1 - P. 936

4. Directory

   4.1. Index of Document

   4.2. Glossary

   4.3. Acronym List

5. Sample Figures&Tables

Figure 1 - System deployment diagram

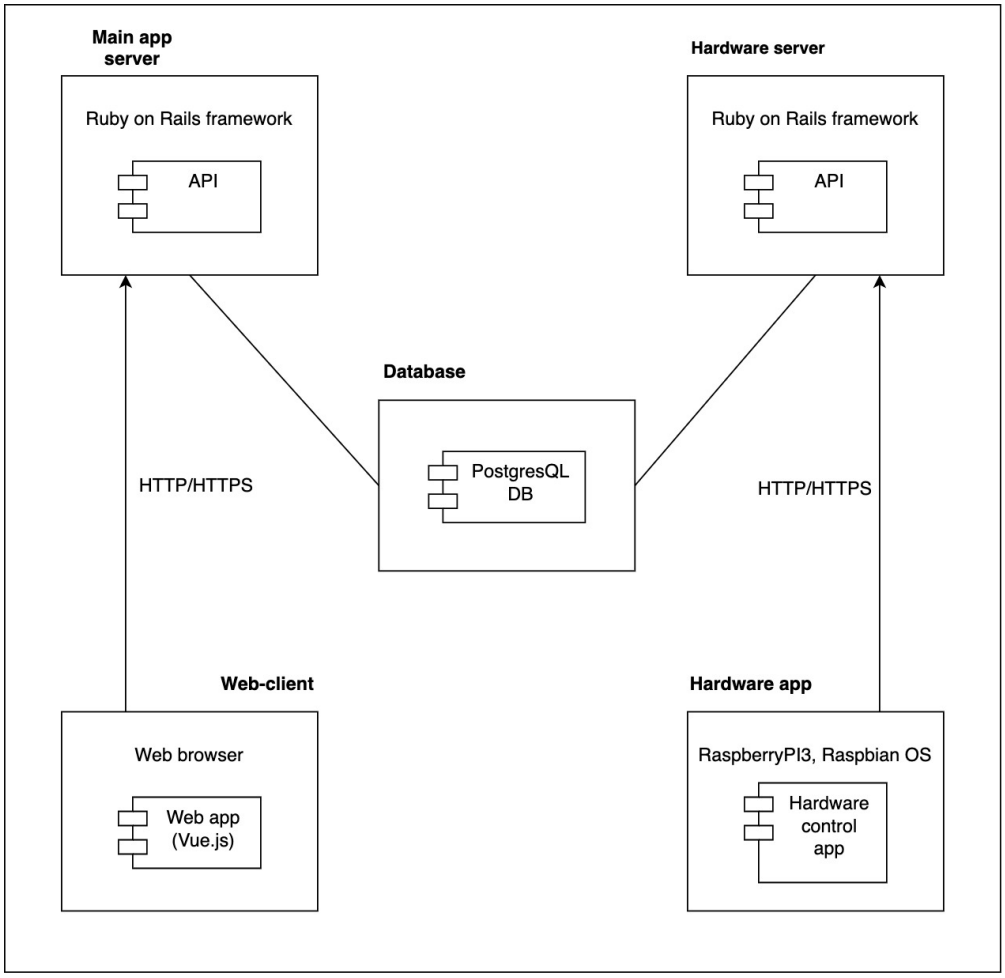
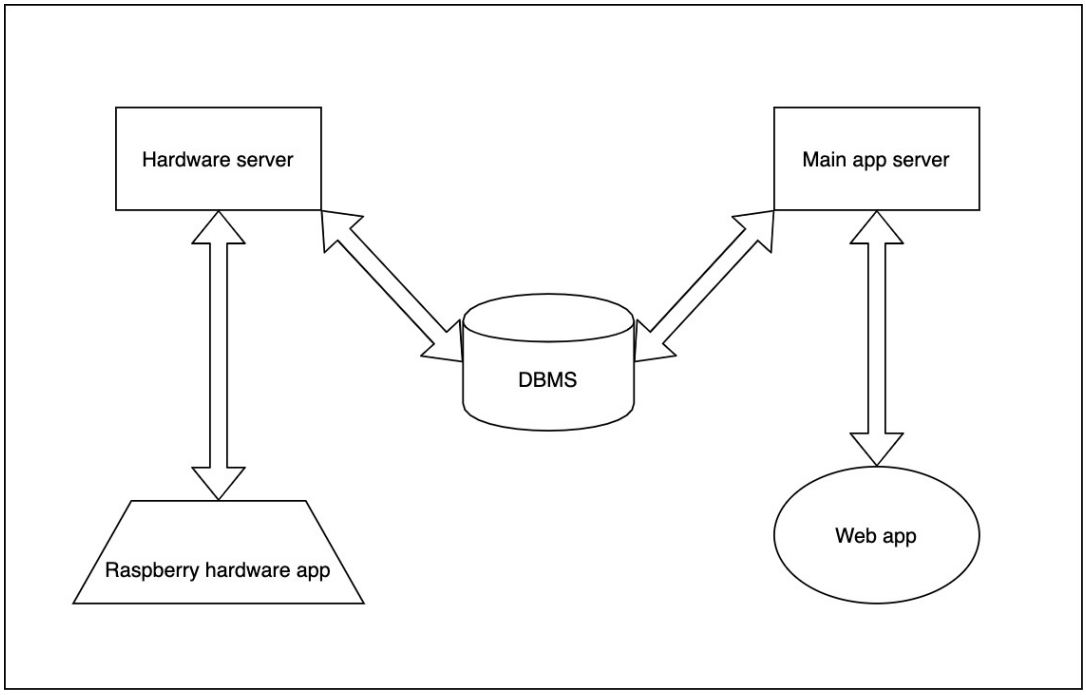


Figure 2 - Diagram illustrating the interaction of software components of the system


Hardware server
Main app server
DBMS
Raspberry hardware app
Web app