Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка підсистеми оплати міського транспорту на основі біометричних даних
(тема)

Виконав:
студент 2 курсу, групи СШІм-19-1
Еккер А.В.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)
Керівник доц. Золотухін О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Еккеру Артему Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка підсистеми оплати міського транспорту на основі біометричних даних

затверджена наказом університету від __ _____ 20 __ р. № _____

2. Термін подання студентом роботи до екзаменаційної комісії _____ _____ 20 ___ р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових проекті, Java documentation

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної галузі, алгоритми роботи серверної та клієнтської частини додатку, аналіз алгоритмів реконструкції зображень відбитків пальців, визначення архітектури додатку, вибір, розробка та обґрунтування технічного забезпечення підсистеми, обґрунтування засобів захисту інформації від несанкціонованого доступу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Блок-схема алгоритму роботи клієнтського додатку, Рисунок 2 – Use-Case діаграма додатку, Рисунок 3 – Контекстна діаграма оплати проїзду в транспорті на основі біометричних даних, Рисунок 4 – Діаграма послідовності оплати проїзду в транспорті на основі біометричних даних, Рисунок 5 – Архітектура клієнтської частини підсистеми, Рисунок 6 – Декомпозиція процесу отримання біометричних даних

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Розділи додаткової частини	доц. Золотухін О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційне проектування	02.11.2020	виконано
2	Аналіз завдання та пошук літератури з теми	08.11.2020	виконано
3	Опрацювання літератури	09.11.2020	виконано
4	Вибір програмних засобів для розробки системи	11.11.2020	виконано
5	Розробка програмного засобу	13.11.2020	виконано
6	Аналіз отриманих результатів	04.12.2020	виконано
7	Оформлення пояснювальної записки	05.12.2020	виконано
8	Оформлення презентаційних матеріалів	07.12.2020	виконано
9	Представлення на рецензування	08.12.2020	виконано
10	Попередній захист2	09.12.2020	виконано
11	Захист перед ЕК	15.12.2020	

Дата видачі завдання 2 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 118 с., 8 табл., 50 рис., 4 дод., 18 джерел.

АЛГОРИТМИ, АРХІТЕКТУРА, ДОДАТОК, ЕКОНОМІКО-МАТЕМАТИЧНА МОДЕЛЬ, МАШИННЕ НАВЧАННЯ, МІСЬКИЙ ТРАНСПОРТ, ANDROID, JAVA, PYTHON.

Об’єкт дослідження – оплата проїзду на основі біометричних даних.

Метою атестаційної роботи є розробка компонентів підсистеми оплати проїзду в міському транспорті на основі біометричних даних для надання клієнтам зручності користування послугами міського транспорту.

Методи дослідження – методи визначення алгоритмічної моделі служби та проектування роботи додатку і моделювання алгоритмів роботи програмного забезпечення. Дослідження проводилося на IBM-сумісному персональному комп'ютері (тактова частота процесора 2.4 ГЦ, обсяг оперативної пам'яті – 8 ГБ) за допомогою пакетів програм Intellij Idea, Intellij PyCharm та Android Studio та на мобільному пристрої Honor під керуванням операційної системи «Android».

Результати атестаційної роботи – додаток, що дозволяє сплачувати проїзд у міському транспорті на основі біометричних даних.

Область застосування – розроблений додаток можна широко використовувати серед великої кількості користувачів для сплати міського транспорту.

РЕФЕРАТ

Пояснительная записка: 118 л., 8 табл., 50 рис., 4 доп., 18 источников.

АЛГОРИТМЫ, АРХИТЕКТУРА, МАШИННОЕ ОБУЧЕНИЕ, ПРИЛОЖЕНИЕ, ЭКОНОМИКО-МАТЕМАТИЧЕСКАЯ МОДЕЛЬ, ОБЩЕСТВЕННЫЙ ТРАНСПОРТ, ANDROID, JAVA, PYTHON

Объект исследования – оплата проезда на основе биометрический данных.

Целью аттестационной работы является разработка компонентов подсистемы оплаты проезда в общественном транспорте на основе биометрических данных для предоставления клиентам удобства пользования услугами общественного транспорта.

Методы исследования – методы определения алгоритмической модели сервиса, проектирование работы приложения и моделирование алгоритмов работы программного обеспечения. Исследование проводилось на IBM-совместимом персональном компьютере (тактовая частота процессора 2.4 ГЦ, объем оперативной памяти – 8 ГБ) с помощью пакетов программ Intellij Idea, Intellij PyCharm и Android Studio и на мобильном устройстве Honor под управлением операционной системы «Android».

Результаты аттестационной работы – приложение, позволяющее оплачивать проезд в общественном транспорте на основе биометрических данных.

Область применения – разработанное приложение можно широко использовать среди большого количества пользователей для оплаты транспорта.

ABSTRACT

Explanatory note: 118 p., 8 tables, 50 fig., 4 app., 18 sources.

ALGORITHMS, ANDROID, APPLICATION, ARCHITECTURE, JAVA, MACHINE LEARNING, MATHEMATICAL MODEL, PYTHON, UBRAN TRANSPORT

Object of research – payment subsystem for urban transport based on biometric data.

The aim of the thesis is to develop payment subsystem for urban transport based on biometric data program components to increase a convenience level of use of the urban transport.

Research methods – methods of building of algorithmic model of the service, application design and software implemented algorithms modelling. The study was conducted on an IBM-compatible personal computer (CPU clock speed 2.4 HZ, RAM – 8 GB) using Intellij Idea, Intellij PyCharm and Android Studio IDE and also using a Honor smartphone running under “Android” OS.

The results of the thesis – an application that allows a client to pay for urban transport on the basis of biometric data.

Scope – created client application can be used among a huge number of potential clients for the purpose of payment for urban transport.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

БД – база даних;

СУБД – система управління базами даних;

CLI – Command-line Interface – різновид текстового інтерфейсу користувача й комп'ютера, в якому інструкції комп'ютеру можна дати тільки введенням із клавіатури текстових рядків (команд);

DI – Впровадження залежності (Dependency injection, DI) — шаблон проектування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію управління» для отримання залежностей;

Inception – згорткова архітектура, розроблена у Google;

MVC – Model-View-Controller (архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення);

VGG – згорткова мережа для виділення ознак зображень.

ВСТУП

Світ не стоїть на місці: він постійно розвивається. Особливо це стосується світу автоматизації повсякденних дій. Це відбувається тому, що електронні пристрої надають людству великий об’єм можливостей і користі. Винайдений порівняно недавно, він вже щільно інтегрувався в наше життя. Це одна з причин, чому індустрія електронних додатків розвивається семимильними кроками: за прогнозами, в 2021 обсяг додатків, що автоматизують повсякденні дії користувачів буде складати 17% з усіх, за даними Google.

Світ стає мобільним і електронні додатки вже давно міцно інтегрувалися в життя людства. Багато компаній зрозуміли можливі переваги, які може принести розробка програми для бізнесу, ще на самому початку розвитку інформаційного світу. В даний час розробка власного бізнес додатка стало загальним трендом.

Використання біометричних даних у додатках стає все більшим трендом через водночас ненадійність інших методів, наприклад паролів, та надійність біометричних методів. Не останньою у списку переваг використання біометричних даний йде й простота використання для кінцевого користувача, адже набагато зручніше просто прикласти палець до сканеру, чи показати обличчя на камеру-сканер аніж тримати у голові безліч логінів та паролів.

Це стає дуже гарним трендом у світі технологій. Наприклад, компанія Mastercard анонсувала, що з квітня 2019 року її користувачі матимуть можливість використовувати біометричні дані, такі як відбитки пальців чи розпізнавання обличчя, для ідентифікації під час покупок і оплати.

Про підвищений інтерес до біометричних рішень свідчить усе більша доступність відповідного функціоналу на планшетах і смартфонах, підтримка

подібних рішень споживачами та нові нормативні вимоги ЄС щодо надійної ідентифікації. Переважна більшість споживачів – 93% – обирає біометрію замість паролів для підтвердження платежів.

Нещодавнє дослідження Оксфордського університету, проведене спільно з Mastercard, виявило, що 92% професіоналів у банківському секторі планують запроваджувати біометричні рішення. Банки також зазначають, що з використанням біометричної автентифікації їхні клієнти значно охочіше завершуватимуть процес купівлі товарів. Частота відмов може скоротитись на 70%, порівняно з іншими методами автентифікації, такими як одноразовий пароль, надісланий через SMS. В результаті це значно поліпшить взаємодію банків з користувачами. Це ще раз підкреслює перевагу використання такого методу автентифікації клієнтів для бізнесу, де автентифікація необхідна.

Держави також розуміють перевагу такого методу автентифікації для громадян, ще з кінця 20 сторіччя на сході будували плани щодо біометричних паспортів, а 6 грудня 2012 року набрав чинності закон «Про Єдиний державний демографічний реєстр», який передбачає впровадження електронних паспортів для громадян України. Із появою біометричних паспортів у світі також значно спростилася процедура перетину кордону. На пунктах пропуску працівникам Прикордонної служби не доводиться вручну вводити дані особи. Тепер за допомогою оснащеного приладу таку інформацію зчитують із чіпа, що знаходиться в документі. При цьому людський фактор при перетині кордону мінімальний, що робить таку систему більш надійною.

Додаток, на основі біометричних даних для сплати міського транспорту буде не лише корисним та прозорим для громадян, але й дасть змогу покращити інфраструктуру міста на основі зібраних статистичних даних. Також перевагами такого методу сплати є:

– зручність – користувачам такої системи не потрібно буде поповнювати картки проїзду чи розраховуватися готівкою,

– швидкість системи – такий процес сплати прискорить посадку пасажирів на зупинках, що вплине також і на зручність перевезень,

– прозорість – така система буде прозорою, а прибуток неможливо буде приховати, що добре позначиться на бюджеті країни

– автоматизація – система дозволить не відволікатися водію, який найчастіше є й кондуктором, від свого головного обов’язку – керування транспортним засобом, що позитивно вплине на безпечність міського транспорту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

З ростом обчислювальних потужностей в все більше повсякденних потреб автоматизуються, робляться біль зручними для користування та більш надійними для користувачів. Міським транспортом користуються або користувалися усі, але чомусь загальна автоматизація майже не торкнулася такої повсякденної та повторюваної речі як оплата транспорту. Картки для проїду це непогана спроба, але вони майже не відрізняються від сплати готівкою, адже усе одно потрібно знімати готівку та класти її на окрему картку, тип паче для окремого виду транспорту. Саме тому сплата проїзду завдяки біометричним даним може стати гарним рішенням, адже користувачам стане набагато простіше користуватися таким сервісом завдяки банальній простоті використання та економії часу.

Система розпізнавання біометричних даних для сплати громадського транспорту буде займатися автоматизацією комерційної діяльності пасажирських перевезень. Дана система буде зчитувати дані з сенсора, розташованого на вході в транспорт, після буде отримувати біометричну інформацію і проводити транзакцію по оплаті проїзду, на виході видаючи або надрукований квиток на проїзд, або повідомлення про помилку, будь то помилка при розпізнаванні або недостача коштів на банківській картці людини, що намагається зробити оплату проїзду.

Щодо вибору біометричних даних, за якими буде проходити автентифікація користувачів даної системи – відбиток пальця припускає невеликий відсоток помилкових відмов в автентифікації, але гарантує стовідсоткову відсутність помилкового доступу, що є вирішальним фактором вибору саме відбитку пальців як основного методу автентифікації.

Компанія «Sandria» проводила дослідження ефективності найпопулярніших методів біометричної автентифікації на комерційних приладах. Дані про надійність різних методів розпізнавання наведено в таблиці 1.1.

Таблиця 1.1 – Надійність біометричних методів автентифікації

Техніка	Відсоток помилкового доступу
Розпізнавання голосу (метод Alpha)	3%
Розпізнавання голосу (метод ECCO)	2%
Динаміка факсимільного підпису	2%
Сканування сітківки	0,4%
Геометрія руки	0,1%
Відбитки пальців	9% помилкових відмов, 0% помилкових доступів

1.2 Огляд аналогів

Біометрична автентифікація стає все більш популярною в банківському і фінансовому секторі. З серпня 2005 японським банкам доводилося відшкодовувати збитки клієнтів в результаті неналежного зняття готівки за законом, якщо винна поведінка не може бути доведено щодо клієнта. Насправді, більше 40 японських банків використовують розпізнавання пальмових вен в більш ніж 19 000 банкоматів. Понад 600 000 клієнтів Tokyo Mitsubishi Bank UFJ використовують біометричну перевірку в банкоматах. Інший блок фінансових компаній, які впровадили технологію, яка використовує малюнок вен на кінчиках пальців людини: Sumitomo Mitsui Banking Corp., Mizuho Bank і Japan Post. Банк Columbian Bancafe впроваджує сканування відбитків пальців по всій своїй мережі банкоматів, розроблене корпорацією NCR (NYSE: NCR). У 2007 році

бразильський банк Bradesco запровадить таку ж систему з 25 000 банкоматів.

Також є кілька проектів з біометричними платіжними системами в сфері продажу, наприклад, у Великобританії, Німеччині і в США, Thrift-way і Kroger, як супермаркети, так і продовольчі магазини, використовують технологію оплати за відбитками пальців. Інша система використовується в Великобританії, де біометрична автентифікація поєднується або з номером телефону, або з датою народження. Постачальником цієї технології в Сан-Франциско є компанія Pay by Touch. Компанія була профінансована за рахунок венчурного капіталу в розмірі 130 мільйонів доларів США двома хедж-фондами. У Німеччині на різних споживчих ринках EDEKA і декількох шкільних кафетеріях запроваджено платіжна система, яка називається digiPROOF. Система, яка використовується в шкільній їдальні, привела до суперечностей в Німеччині, оскільки вона може вплинути на закони про конфіденційність даних.

Pay By Touch представила онлайн-сервіс біометричної автентифікації, де відбиток пальця користувача об'єднується і шифрується з унікальним ідентифікатором пристрою перед його відправкою на сервер автентифікації під назвою TrueMe. Ця система може використовуватися для онлайн-платежів, онлайн-банкінгу, онлайн-брокерів і інших захищених паролем веб-сайтів і сервісів.

Навіть банківські гіганти, такі як MasterCard планують впровадження використання біометричних даних. За оцінками MasterCard, додавання біометричної автентифікації на основі смарт-карт до оплати за допомогою кредитної картки POS зменшить шахрайство на 80%. Комбінація зі смарт-картами підтримує захист конфіденційності і підвищує безпеку і довіру. Умовне порівняння засобів сплати проїзду представлено в таблиці 1.2.

З таблиці 1.2 видно що сплата відбитком є найпростішим та найшвидшим способом сплати, хоча і потребує попередньої реєстрації.

Вона виграє у гібридних засобів в швидкості та зручності, а у більш звичних способів сплати – у простості та також швидкості сплати.

Таблиця 1.2 – Умовне порівняння засобів сплати проїзду

	Готівка	Банківська карта	Відбиток пальця	Розпізнавання обличчя	Гібрид (біометрична інформація та дата народження чи номер телефону)
Мінуси при процесі сплати	Очікування решти, необхідність носити готівку з собою	Введення пін-коду, необхідність носити картку з собою	–	–	Необхідність підтвердження біометричних даних
Мінуси, що сповільнюють процес сплати	Пошук коштів, очікування решти	Введення пін-коду	–	Довгий процес розпізнавання та пошуку відповідностей	Необхідність підтвердження біометричних даних датою народження чи номером телефону
Потрібність підготовки даних	–	–	Потреба в реєстрації відбитку в системі	Потреба реєстрації обличчя в системі	Потреба реєстрації біометричних та додаткових даних в системі

1.3 Постановка задачі

В ході роботи планується розробити клієнт-серверний додаток для забезпечення автоматизації оплати проїзду користувачами завдяки біометричним даним. Система буде складатися з двох частин: клієнтська частина буде містити в собі програмне забезпечення для зчитування відбитку пальця, безпосередньо сам сканер і пристрій для видачі квитків. Передбачається що початкова версія буде використовувати оптичні сканери, через їх дешевизну, що б з самого початку не було потрібно занадто великих фінансових вкладів для тестування системи.

По-перше потрібно проаналізувати вимоги до програмного продукту, а саме провести 3 види діяльності: виявлення вимог – задача комунікації з користувачами системи для виявлення вимог до системи, аналіз вимог – виявлення недоліків вимог (неповноти, неоднозначностей, неточностей чи суперечностей) і їх виправлення, запис вимог – документація вимог специфікаціями процесу. Після треба сформували загальний план специфікації вимог до програмного забезпечення, а саме: функції продукту, характеристики користувачів, загальні обмеження, припущення й залежності. А також конкретні вимоги: інтерфейс користувача, програмний інтерфейс, обмеження пам’яті, операції та функції продукту.

Після того як мета і специфікація програми описані, можна переходити до проектування програмного забезпечення, що включає в себе: проектування архітектури програмного забезпечення, проектування окремих компонентів програмного забезпечення та їх зв’язок та проектування користувацьких інтерфейсів.

Окремим кроком можна описати проектування бази даних, адже перша її версія будується саме на припущеннях щодо роботи системи та її архітектури, тому доцільною буде побудова діаграм(IDEF0, IDEF3) для більш наочного відображення усіх процесів системи.

Після всіх етапів проектування, наведених вище потрібно переходити до саме написання програмного продукту згідно всіх проектних вимог та тестування програмного забезпечення для перевірки відповідності усім пунктам специфікації програмного продукту.

Після того як всі компоненти системи написані та протестовані, слід переходити до системної інтеграції – поєднання компонентів системи в єдину та забезпечення роботи окремих підсистем як єдиної системи.

Останнім кроком необхідно впровадити програмне забезпечення та організувати процес супроводження програмного забезпечення, а саме покращення, оптимізації та виправлення дефектів у програмному забезпеченні після його вводу в експлуатацію. Цей процес містить в собі: коригування, вдосконалення продукту для усунення виявлених помилок або нереалізованих задач; адаптацію, підлаштування продукту до умов експлуатації, що змінилися, або в новому середовищі виконання; поліпшення, еволюційну зміна продукту для підвищення продуктивності або рівня супроводу та перевірку ПЗ, пошук і виправлення помилок при експлуатації системи.

Підключення до інтернету є обов'язковим фактором так як робота з інформацією відбуватиметься на віддалених серверах через досить велике навантаження при пошуку відповідників з відбитками, робити це на компактних пристроях, які будуть встановлені в транспорт недоцільно. Дані установки будуть використані для зчитування даних, їх передачі на сервер і для отримання даних про транзакції, або про її скасування з певної причини. Дистанційні сервера будуть обробляти інформацію, отриману з пристроїв зчитування, знаходити відповідності в базі і посилати транзакцію на обробку, після обробки транзакції відповідь буде передано пристрою зчитування і в разі успіху транзакції – сигнал передається на чековий апарат для друку квитка, в разі ж невдачі буде виведено повідомлення користувачу про причини невдачі транзакції. Основною перевагою такого виду спілкування є невеликий обсяг даних, які потрібно

передати по мережі, а також захищеність передачі: навіть в разі якщо дані будуть перехоплені, то зловмисники зможуть отримати або зашифрований відбиток людини, розшифровка якого недоцільна по витраченому часу, або повідомлення про успіх або невдачі транзакції, ніякі банківські дані, а тим більше дані безпосередньо про транзакції не передаються, що робить систему безпечною. А якщо будуть використані ультразвукові датчики зчитування відбитків, то буде і захист від муляжів, які можна прикласти до сканера, завдяки тому, що даний метод зчитування біометричної інформації може зчитувати не тільки відбиток, але й інші біометричні дані, наприклад, пульс.

Для більш довговічного використання можливо використовувати не оптичний сканер, як на сучасних смартфонах, а, наприклад, ультразвуковий, він набагато менш схильний до зносу, так як не вимагає безпосереднього контакту з пальцем для зняття відбитка, а також він отримує зображення в десятки разів краще, ніж всі представлені на даний момент біометричні сканери, його ж поки єдиним мінусом є його досить висока ціна. Якщо ж цю систему планується ставити в місцях з тропічним або ж субтропічним кліматом – найбільш доцільно буде використовувати термосканери для зняття відбитків пальців, так як вони дешевше ультразвукових і працюють краще за інші методи (крім ультразвукового), але в нашому кліматі їх використання недоцільне так як у них є проблеми в роботі при температурах нижче -10 градусів за Цельсієм. Ще одним варіантом можуть стати радіочастотні сканери. Вони чудово виконують свою роботу оскільки аналізують фізіологічні властивості шкіри і тому ймовірність обману даного сканера прямує до нуля, але їх мінусом є нестійка робота при поганому контакті пальця. Якщо ж будуть потрібні сканери з низькою собівартістю і хорошою надійністю (наприклад, для тестування системи перші кілька місяців і оцінки доцільності подальшого фінансування), то найкращим варіантом будуть ємнісні сканери, їх робота основана на напівпровідниковій матриці і працює досить надійно, мінусом

ж даних сканерів є повна відсутність захисту від муляжів. Отже необхідно визначити вимоги до програмного продукту, розробити діаграми, що наочно демонструють функції та залежності системи, врахувати всі потреби системи, після чого реалізувати серверну та клієнтську частину додатку. Обидві частини додатку повинні бути протестовані та вдосконалені після проходження тестів за потреби. Також розробити інтерфейс користувача для клієнтського додатку.

1.4 Обґрунтування мети вирішення поставленої задачі

Мета цієї інновації – спростити та зробити більш прозорою оплату за проїзд в автобусах, маршрутних таксі, тролейбусах, трамваях і метро.

Перевагою цієї системи для пасажирів є передовсім зручність та комфорт. Такий спосіб оплати транспорту звільняє пасажирів від потреби передавати гроші за проїзд та чекати на решту. Також це можливість уникнення пільгової дискримінації.

Також дана система підвищить рівень безпеки на дорозі, адже водій тепер зможе спокійно керувати транспортним засобом, а не виконувати паралельно функції кондуктора.

Для міст одною з найбільших переваг буде статистика. Облік реального пасажиропотоку на кожному маршруті, що допоможе оптимізувати інфраструктуру пасажирських перевезень у місті та зробити маршрути оптимальними, виходячи з точної статистики. Також це вирішить проблеми компенсацій за пільговий проїзд громадян компаніям перевізникам.

Найбільша перевага цієї системи для міста – це боротьба з корупцією, адже у системи буде статистика, що унеможливить переховування податків через заниження показників перевезень у звітах.

2 МЕТОДИ І АЛГОРИТМИ РОЗПІЗНАВАННЯ ТА ВИЗНАЧЕННЯ ІДЕНТИЧНОСТІ ВІДБИТКІВ

2.1 Обґрунтування вибору математичної схеми і розробка математичного опису розроблюваної задачі

Усі алгоритми розпізнавання чи зчитування відбитків дуже схожі і відрізняються тільки методами зчитування, що залежить від типу сканера, що використовується та від методу порівняння відбитку, що було зчитано з відбитками, що є у базі. Тому буде розглянуто типовий алгоритм, що й буде використовуватись надалі у роботі.

Принцип розпізнавання пальців базується на наявності в відбитку особливих точок – закінчень ліній і розгалужень, або. Кожна особлива точка умовно має свій напрямок і тип (закінчення або розгалуження). Приклад наведено на рисунку 2.1, де закінчення ліній виділені квадратами, а розгалуження кружечками а на рисунку 2.2 представлено зіставлення таких точок на відбитках.

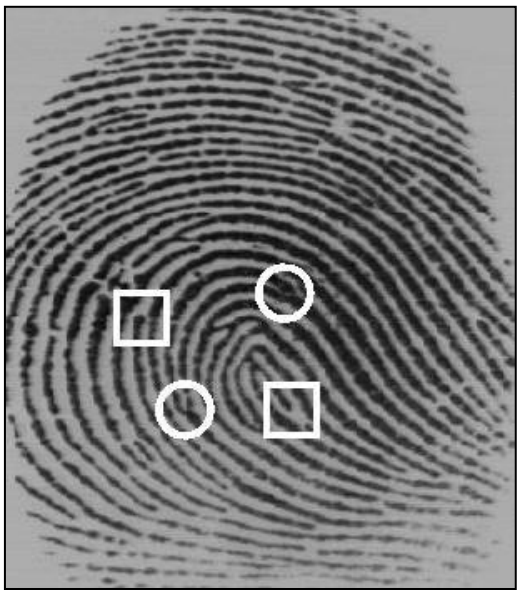

Рисунок 2.1 – Приклад виділення особливих точок на відбитку

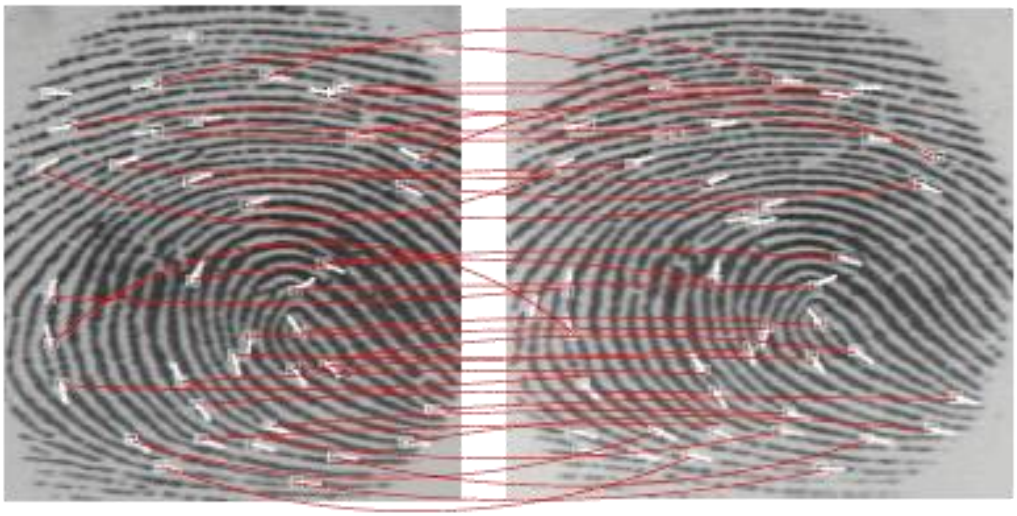

Рисунок 2.2 – Зіставлення точок на відбитках

Маючи відомий набір особливих точок (x, y, θ, type), де x та y – координати точок, θ – кут відносно осі y, type – тип (закінчення або розгалуження) витягнутий при реєстрації, і тестовий, алгоритм оцінює схожість точкових зразків і видає результат – «Пізнаний» або «Непізнаний». Приклад наведено на рисунку 2.3.

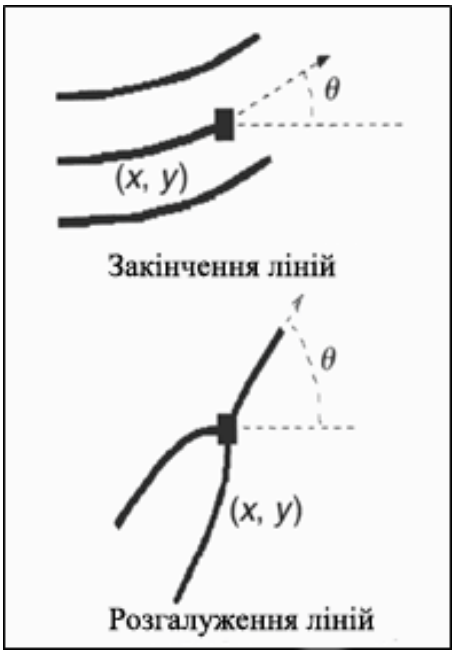

Рисунок 2.3 – Визначення особливих точок

Схема роботи алгоритму візуально представлена на рисунках 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10 та 2.11.

Рисунок 2.4 – Вхідне зображення

Рисунок 2.5 – Адаптивна фільтрація

Рисунок 2.6 – Бінаризація, виділення однорідних областей

Рисунок 2.7 – Морфологічна обробка

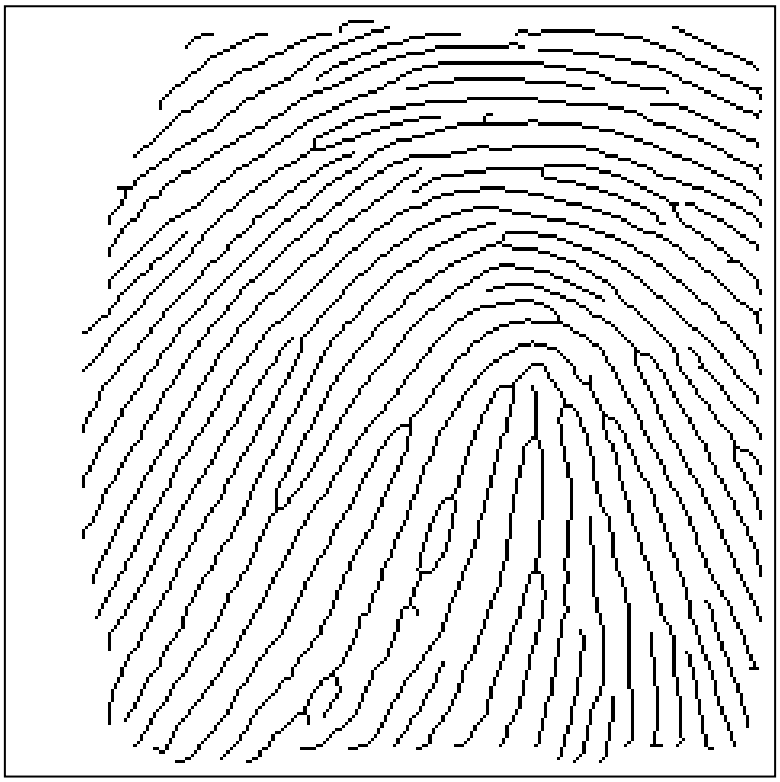

Рис 2.8 – Звуження ліній

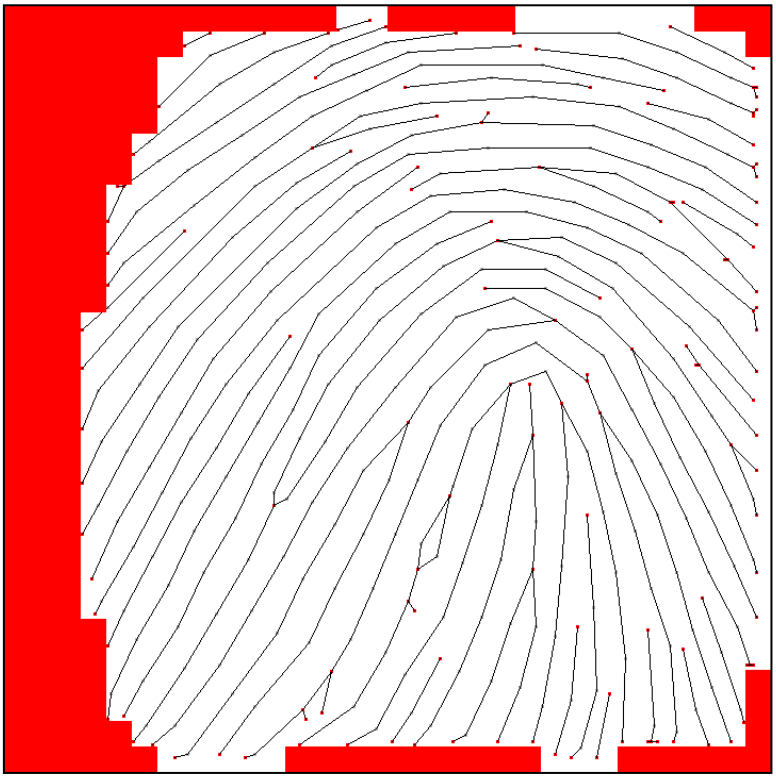

Рисунок 2.9 – Векторизація ліній

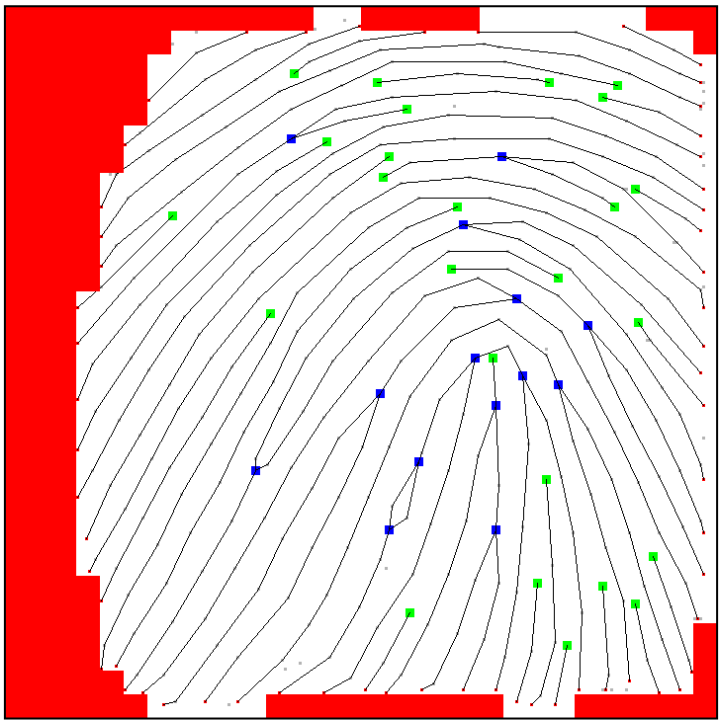

Рисунок 2.10 – Векторна постобробка ліній

Рисунок 2.11 – Порівняння двох наборів особливих точок

Адаптивна фільтрація спрямована на усунення шумів вхідного зображення (бруд, пил на датчику) та усунення дрібних дефектів (затягування невеликих розривів, пор, згладжування ліній пальця). Вона реалізована на основі обробки фільтрами з апертурою 9x9 з урахуванням локальної спрямованості ліній. Область застосування фільтрів – 8-бітові сірі зображення, що використовують весь динамічний діапазон значень, тобто мають плавні переходи значень від лінії пальця до западини. Приклад роботи алгоритму представлено на рисунку 2.12.

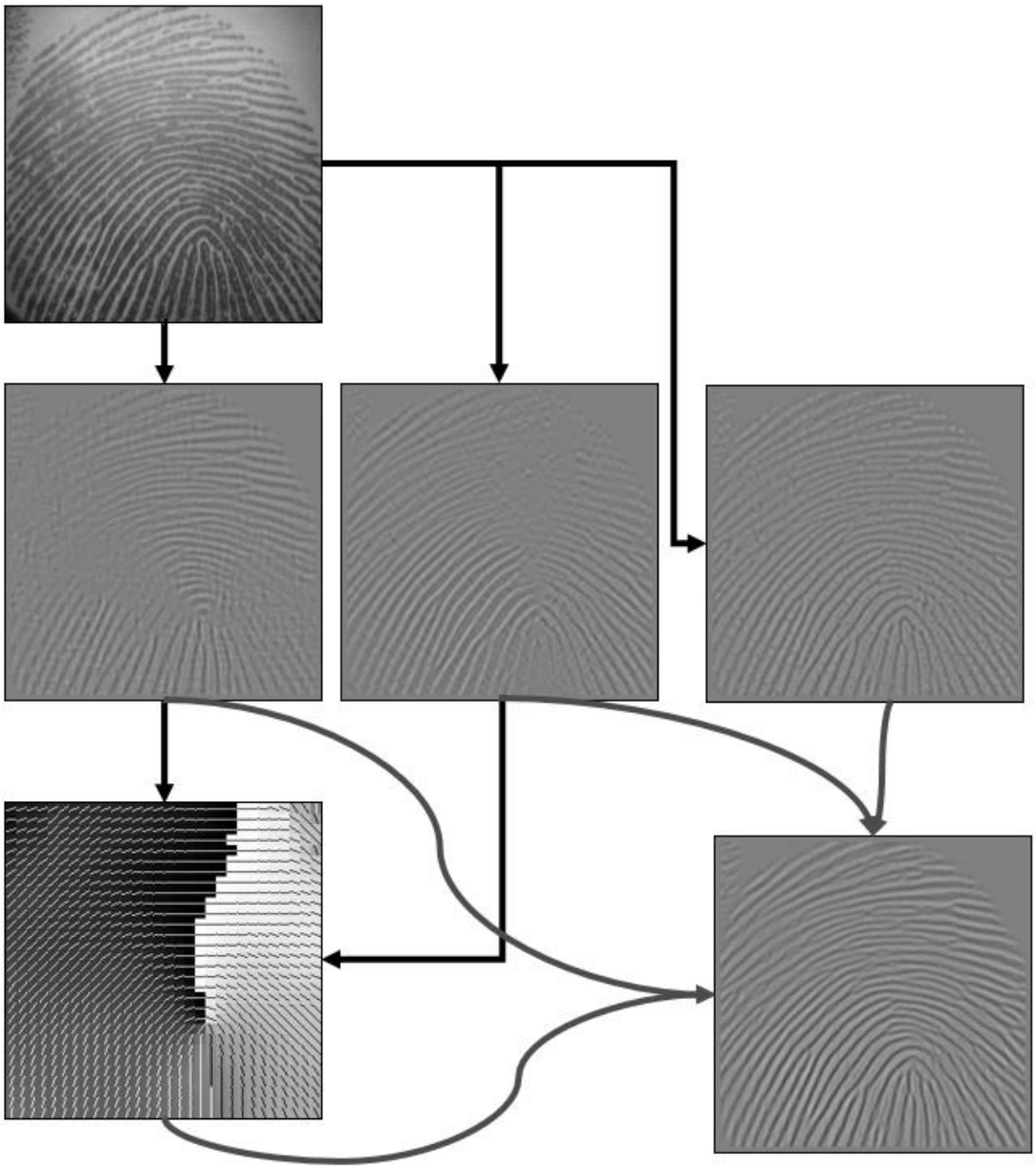

Рисунок 2.12 – Приклад роботи адаптивної фільтрації

Після бінаризации відфільтрованого зображення проводиться виділення однорідних областей і подальша морфологічна обробка зображення. Вона включає в себе видалення так званого шуму «Сіль і перець», тобто видалення дрібних областей білого і чорного кольорів. Це усуває такі помилкові структури, як «озера» в товстих лініях і дрібні лінії, що відсіває частина помилкових особливих точок на цій стадії. Приклади представлені на рисунках 2.6 та 2.7 , де на рисунку 2.6 представлено відбиток до морфологічної обробки, а на рисунку 2.7 – після неї.

Морфологічна операція звуження призводить бінарне зображення до його скелету, в якому товщина всіх ліній – 1 піксель. Операція стягує лінії в центр, не роблячи при цьому розривів. Реалізовано на основі ітеративної нелінійної фільтрації з апертурою фільтра 3x3. Процес звуження представлено на рисунку 2.13.

Рисунок 2.13 – Процес звуження ліній відбитку

Процедура векторизації перетворює скелет зображення в частково-лінійне уявлення, відкидаючи надмірну інформацію про лінії пальця. Точність апроксимації може регулюватися за допомогою параметрів алгоритму. Векторизація залишає особливі точки без змін, викидаючи лише проміжні. Після роботи алгоритму компактний шаблон відбитка з повною інформацією про пальці може бути збережений в окремий файл.

Векторна постобробка спрямована на відсіювання помилкових структур в частково-лінійному уявленні відбитка. Деякі конфігурації ліній свідомо не властиві лініях відбитку пальців, і якщо вони все-таки зустрічаються, то вони напевно є результатом недопрацювання попередніх етапів.

Деякі з вироблених операцій: видалення «мостів», видалення «паличок», з'єднання невеликих розривів, видалення «вусиків». Приклади векторної постобробки представлені на рисунку 2.14.

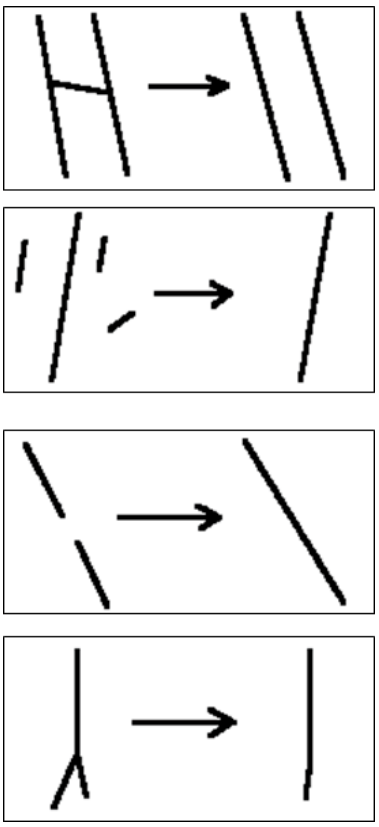

Рисунок 2.14 – Приклад роботи векторної постобробки

На етапі порівняння особливих точок алгоритм за певними критеріями намагається знайти таке поєднання двох наборів, при якому значна частина точок знайшла б свою пару в іншому наборі. При поєднанні з деякими вагами враховуються як напрямки точок, так і їх тип. Не всі точки сполучаються зі своїми парами в іншому наборі – причиною цьому служить як розбіжність зон прикладання пальця, так і похибки алгоритмів екстракції особливих точок. Стосовно збігання пар до загальної кількості точок з урахуванням абсолютного значення кількості точок вважається коефіцієнт суміщення, який і служить результатом верифікації. Приклад порівняння особливих точок представлено на рисунку 2.15.

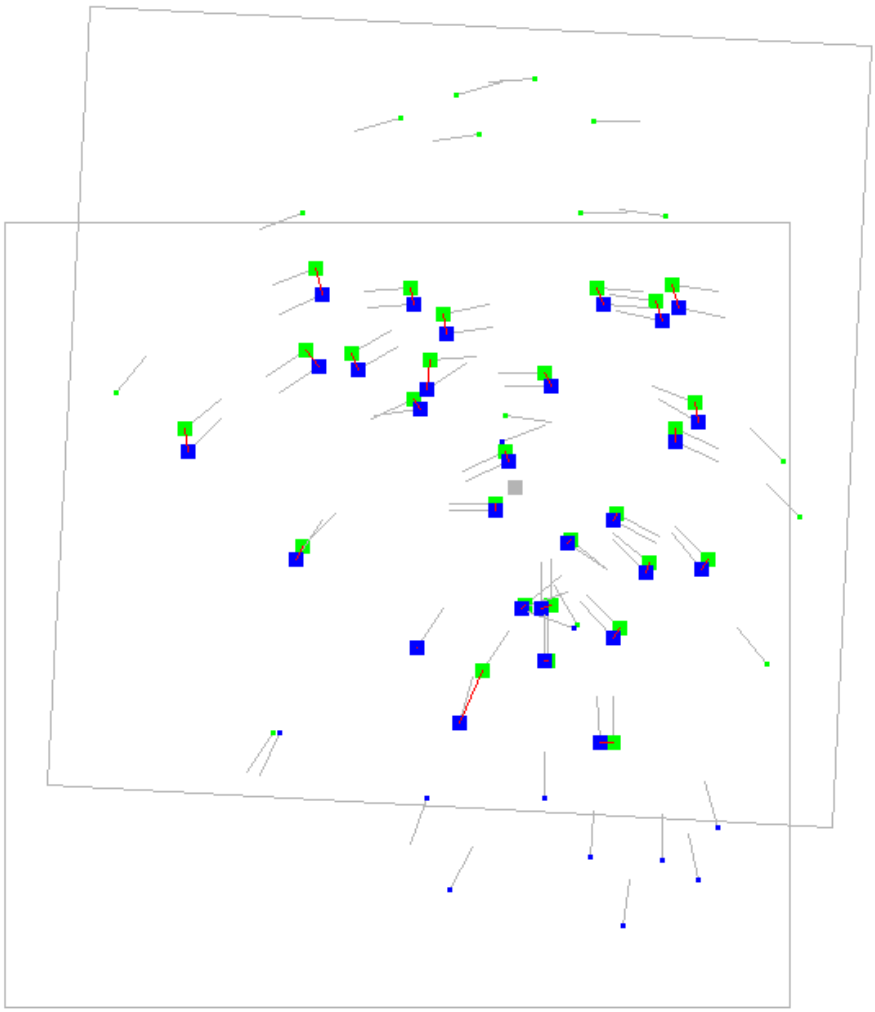

Рисунок 2.15 – Порівняння особливих точок відбитків

2.2 Алгоритм роботи клієнтського додатку з відбитками пальців

Для реалізації роботи з відбитками був використаний Fingerprint API від Google. Він з’явився в версії android 6.0 та використовується для того щоб користувач мобільного пристрою міг замість паролів використовувати відбиток пальця, що значно прискорює вхід всюди, куди потрібно вводити паролі. Блок-схема алгоритму представлена на рисунку 2.16.

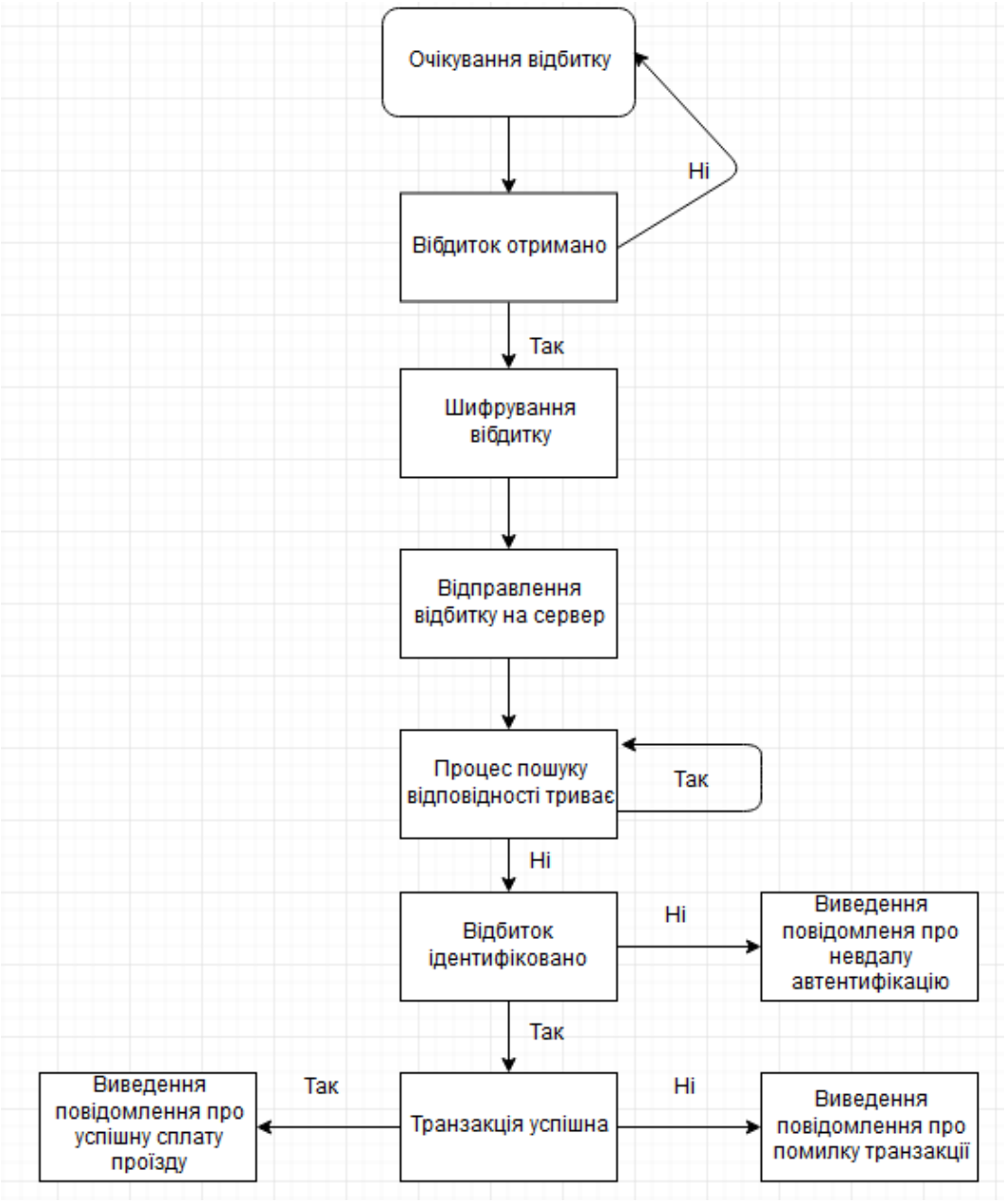

Рисунок 2.16 – Блок-схема алгоритму роботи клієнтського додатку

2.3 Алгоритми машинного навчання для реконструкції зображень відбитків пальців

Згорткові мережі підходять до реалізації поставленої задачі, а саме збільшення якості відбитків для подальшого розпізнавання, адже відбиток дуже легко представити як чорно-біле зображення. А зображення, зазвичай представляють як масив векторів чисел, якщо зображення чорно-біле, то це просто масив інтенсивностей, а якщо кольорове, то масив векторів з трьох чисел, що позначають інтенсивності трьох основних кольорів (червоного, зеленого і чорного в стандартному RGB, синього, зеленого і червоного в трьох типах колбочок в людському оці і т. д.). Таке знання структури входів мережі може істотно допомогти в навчанні нейронної мережі.

Основна ідея згорткової мережі полягає в тому, що обробка ділянки зображення має відбуватися незалежно від конкретного розташування цієї ділянки. Тому згорткова мережа просто робить це припущення в явному вигляді: наприклад покриває вхід невеликими вікнами (наприклад, 5 × 5 пікселів) і буде виділяти ознаки в кожному такому вікні невеликою нейронною мережею.

Причому – і тут ключове міркування – ознаки будемо виділяти в кожному вікні одні й ті ж, тому маленька нейронна мережа буде всього одна, входів у неї буде всього 5 × 5 = 25, а з кожної картинки для неї може вийти дуже багато різних входів.

Потім результати цієї нейронної мережі знову можна буде подати ву вигляді «картинки», замінюючи вікна 5 × 5 на їх центральні пікселі, і на ній можна буде застосувати другий згортковий шар, з уже іншою маленькою нейронною мережею, і так далі. Через декілька таких кроків можна побачити, що в кожному згортковому шарі буде зовсім небагато вільних параметрів, особливо в порівнянні з повнозв'язними аналогами.

Загально, згортка це лінійне перетворення вхідних даних особливого виду. Якщо x^l – карта ознак в шарі під номером l, то результат двовимірної

згортки з ядром розміру 2d + 1 і матрицею ваг W розміру (2d + 1) × (2d + 1) на наступному шарі буде таким:

$$y_{i,j}^{l} = \sum_{-d \le a,b \le d} W_{a,b} x_{i,j+b}^{l}, \tag{2.1}$$

де $y_{i,j}^{l}$ – результат згортки у шарі l;

$x_{i,j}^{l}$ – її вхід, тобто вхід усього попереднього шару.

Інакше кажучи, щоб отримати компоненту (i; j) наступного рівня, застосовується лінійне перетворення до квадратного вікна попереднього рівня, тобто треба скалярно помножити пікселі з вікна на вектор згортки.

Приклад проілюстровано на рисунку 2.17: застосовується згортка з матрицею ваг W розміру 3 × 3 до матриці X розміру 5 × 5. Зверніть увагу, що множення підматриці вихідної матриці X, відповідної вікну, і матриці ваг W – це не множення матриць, а просто скалярний добуток відповідних векторів. А всього вікно вміщується в матриці X дев'ять разів.

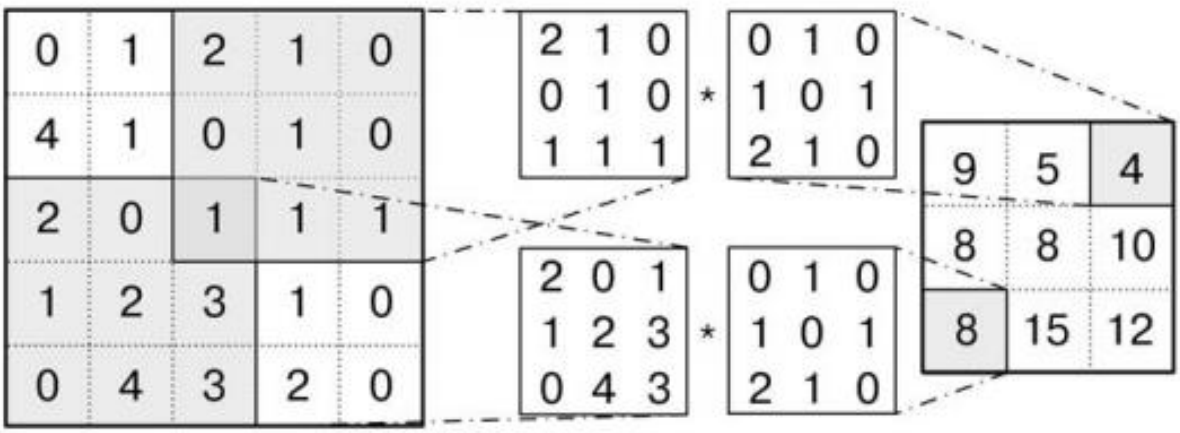

Рисунок 2.17 – Приклад розрахунку результату згортки: два приклади підматриці та спільний результат

Це перетворення характеризується дуже важливими для вирішення поставленої задачі властивостями: згортка зберігає структуру входу (порядок в нашому одномірному випадку), так як застосовується до кожної ділянки вхідних даних окремо; операція згортки має властивість розрідженості, так як значення кожного нейрона чергового шару залежить тільки від невеликої частки вхідних нейронів (а, наприклад, в повнозв’язній нейронної мережі кожен нейрон залежав би від усіх нейронів попереднього шару); згортка багаторазово використовує одні і ті ж ваги, так як вони повторно застосовуються до різних ділянок входу.

Однією з найпопулярніших глибоких згорткових архітектур є модель, яку прийнято називати VGG. Назва походить від того, що ця модель була розроблена в Оксфордському університеті в групі візуальної геометрії (Visual Geometry Group), і їх моделі, представлені на ряд конкурсів з комп'ютерного зору, виступали там під кодовою назвою VGG.

VGG – це відразу дві конфігурації згорткових мереж, на 16 і 19 шарів. Основним нововведенням стала ідея використовувати фільтрів розміром 3 × 3 з одиничним кроком згортки замість використання, як в кращих моделях попередніх років, згорток з фільтрами 7 × 7 з кроком 2 і 11 × 11 з кроком 4. Причому це не просто твердження, а добре аргументована пропозиція.

По-перше, рецептивне поле трьох поспіль згорткових шарів розміром 3 × 3 має розмір 7 × 7, в той час як ваг у них буде всього 27, проти 49 в фільтрі 7 × 7.

Аналогічно і з фільтрами 11 × 11. Це означає, що VGG може стати більш глибокою, тобто містити більше шарів, при цьому одночасно зменшуючи загальне число ваг.

По-друге, наявність додаткової нелінійності між шарами дозволяє збільшити «роздільну здатність» в порівнянні з єдиним шаром з більшою згорткою. Цей же аргумент можна використовувати як мотивацію для того, щоб ввести в мережу згортки розміром 1 × 1; такі шари теж дозволяють

додати додаткову нелінійність в мережу, не змінюючи розмір рецептивного поля. Схему мережі VGG-16 представлено на рисунку 2.18.

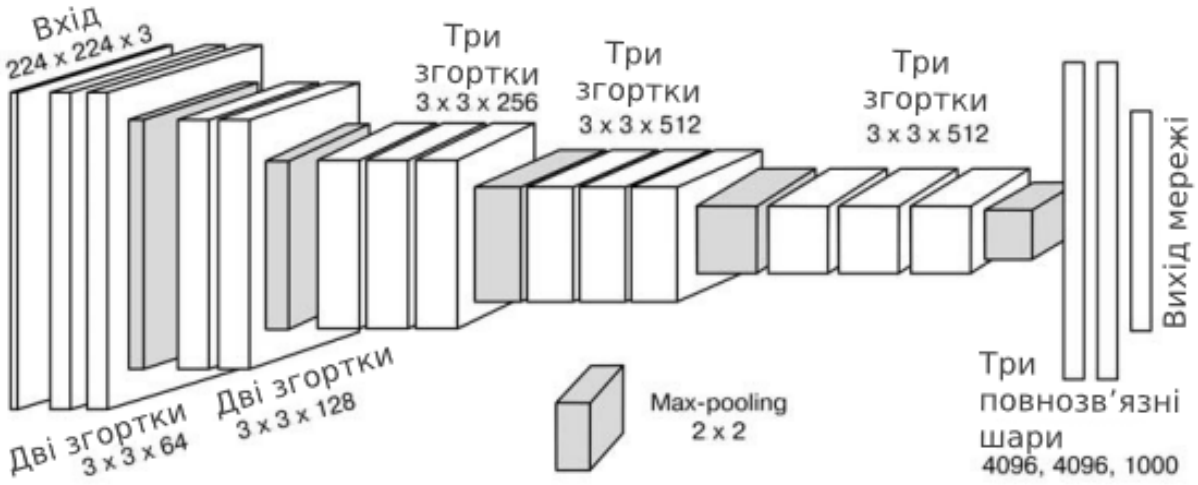

Рисунок 2.18 – Схема мережі VGG-16

Також можна визначити архітектуру Inception. Вона була розроблена в Google і з'явилася практично одночасно з VGG. Ця робота містить кілька дуже цікавих і важливих ідей, завдяки яким, зокрема, не дивлячись на більшу заявлену глибину – 22 шару без урахування субдискретизації – у Inception насправді менше параметрів, ніжу VGG.

Основою Inception є модулі, що комбінують згортки розміром 1 × 1, 3 × 3 і 5 × 5, а також max-pooling субдискретизацію. Одним з ключових нововведень є використання згорткових шарів 1 × 1 не стільки в якості додаткової нелінійності, скільки для зниження розмірності між шарами.

Згортки 3 × 3 і тим більше 5 × 5 між шарами з великим числом каналів (а в Inception-модулях каналів може бути аж до 1024), виявляються вкрай ресурсоємними, незважаючи на малі розміри окремо взятих фільтрів. А фільтри 1 × 1 можуть допомогти скоротити число каналів, перш ніж подавати їх на фільтри більшого розміру.

Ця ідея відображена на рисунку 2.19, а, на якому показана структура одного блоку. А на рисунку 2.20 представлена дуже загальна і

високорівнева схема всієї мережі: вона починається з двох «звичайних» згорткових шарів, а потім йдуть 11 Inception-модулів, двічі перемежовуючихся субдискретизацією, яка знижує розмірність; після цього мережу завершується традиційними повнозв'язними шарами, що дають вихід класифікатора.

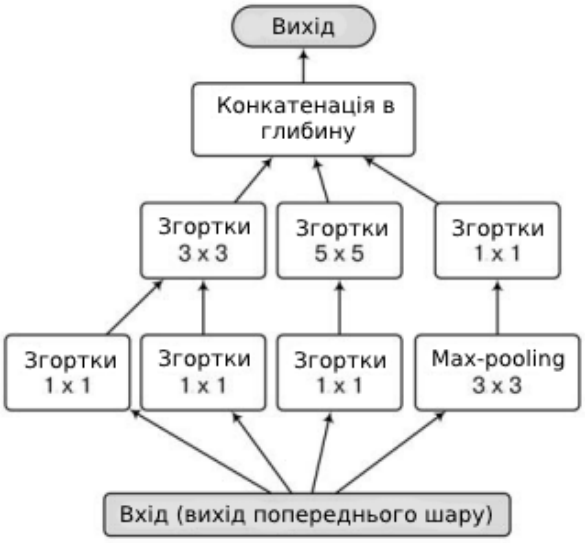

Рисунок 2.19 – Схема одного Inception модуля

Проте, як показала практика, глибокі архітектури стало можливо навчати ефективно, однак ті рішення, до яких сходилися нейронні мережі великої глибини, часто виявлялися гіршими, ніж у менш глибоких моделей. І ця деградація не була пов'язана з перенавчанням, як можна було б припустити. Виявилося що це більш фундаментальна проблема: з додаванням нових шарів помилка зростає не тільки на тестовій, а й на тренувальній множині.

Для вирішення проблеми деградації команда з Microsoft Research розробила нову ідею: глибоке залишкове навчання. Базовий шар нейронної мережі на рисунку 2.19 перетворюється в залишковий блок з обхідним шляхом на рисунку 2.20.

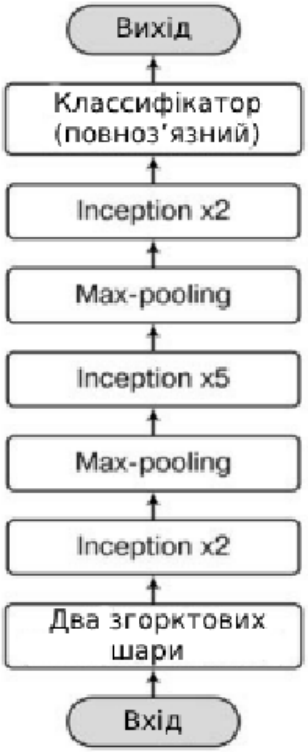

Рисунок 2.20 – Загальна схема мережі GoogLeNet

Математично відбувається все доволі просто: коли два шляхи, «складний» і «обхідний», зливаються назад, їх результати просто складаються один з одним. І залишковий блок висловлює таку функцію:

$$y^{(k)} = F\left(x^{(k)}\right) + x^{k}\ , \tag{2.2}$$

де (x^k) – вхідний вектор шару k;

$F(x)$ – функція, яку обчислює шар нейронів;

$y^{(k)}$ – вихід звлишкового блоку, який потім стане входом наступного шару (x^{k+1}).

Головна ж причина полягає в тому, що градієнт під час зворотного поширення може проходити через цей блок безперешкодно, градієнти не будуть затухати, адже завжди є можливість пропустити градієнт безпосередньо:

$$\frac{\partial y^{(k)}}{\partial x^{(k)}} = \frac{\partial F(x^{(k)})}{\partial x^{(k)}} \ . \tag{2.3}$$

Це означає, що навіть насичений і повністю навчений шар F, похідні якого близькі до нуля, не завадить навчанню.

3 ПРОЕКТУВАННЯ І РОЗРОБКА ДОДАТКУ

3.1 Визначення основних варіантів використання клієнтського додатку

За допомогою функціональної схеми в процесі реалізації програмного продукту одразу можна визначити ролі, що буде потрібно реалізувати, функції та методи для кожної з ролей.

Кожна функція у Use-Case діаграмі повинна бути атомарною у тих випадках, де це можливо. Атомарність функцій у системі дає можливість краще спроектувати додаток та розпаралелити реалізацію цих функцій завдяки їх незалежності одна від одної.

У якості функціональної основи для проектованого додатка доцільно виділити наступні надаються клієнту служби можливості:

– сплата міського транспорту на основі біометричних даних;
– подача заявок на реєстрацію перевізників;
– встановлення ціни проїзду;
– встановлення маршруту проїзду;
– авторизація перевізників;
– прийняття та відхилення заявок перевізників;
– можливість перегляду інформації з бази даних.

Відповідно до запланованого функціоналу, можна виділити п’ять ролей, які можуть використовувати клієнтський додаток. Такими є:

– незареєстрований користувач;
– зареєстрований користувач;
– незареєстрований перевізник;
– зареєстрований перевізник;
– адміністратор.

Відповідна до заданого аналізу функціоналу додатку та ролей системи Use-Case діаграма представлена на рисунку 3.1.

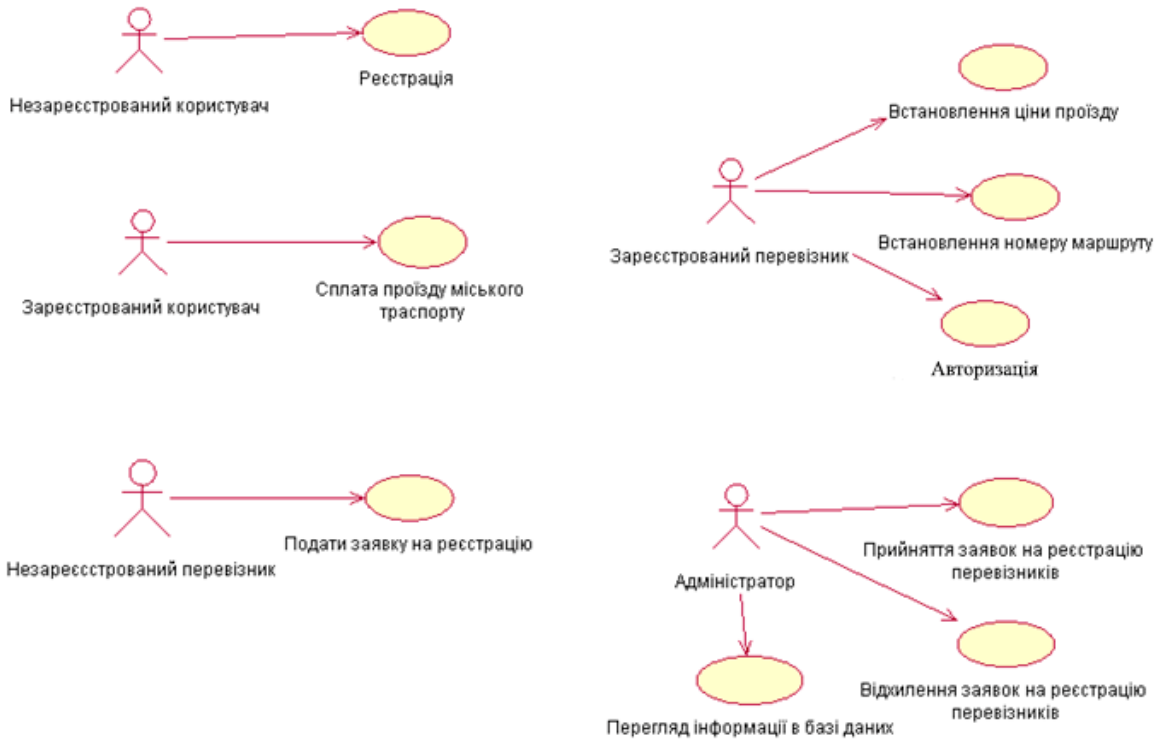

Рисунок 3.1 – Use-Case діаграма додатку

3.2 Реконструкція зображень відбитків пальців за допомогою глибинного навчання

Для тренування та перевірки моделі буде використано набір даних відбитків пальців FVC2002 – це набір даних для перевірки відбитків пальців, який був організований ще в 2000 році, а потім знову в 2002 році. Цей набір даних складається з чотирьох різних відбитків пальців, а саме недорогих оптичних датчиків, недорогих ємнісних датчиків, оптичних датчиків та синтетично згенерованих, кожен сенсор має різні розміри зображення. Набір даних містить 3200 зображень у наборі A, 800 зображень для кожного виду датчиків. На рисунках 3.2 та 3.3 зображено приклади відбитків після їх зчитування.

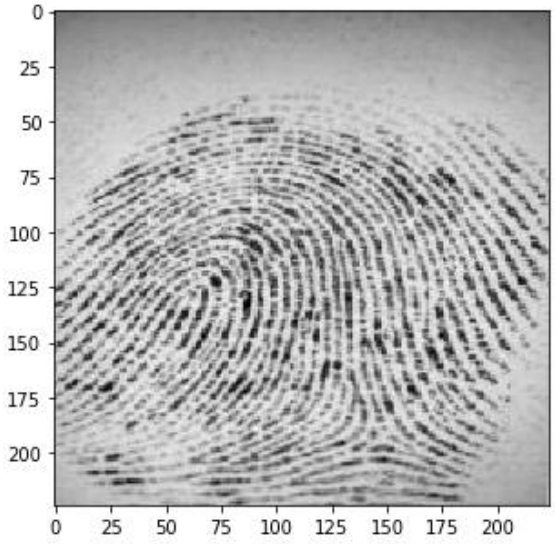

Рисунок 3.2 – Приклад відображення відбитку після зчитування

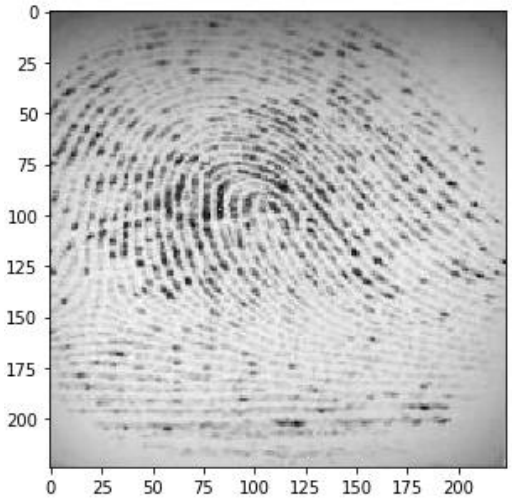

Рисунок 3.3 – Приклад відображення відбитку після зчитування

Результат двох вищенаведених графіків наведено із набору даних. Як можна побачити, відбитки пальців не дуже чіткі, автоенкодеру потрібно буде вивчити особливості та правильно реконструювати ці зображення.

Зображення з набору даних насправді є зображеннями у відтінках сірого із значеннями пікселів від 0 до 255 із розміром 224 x 224, тому перед тим, як подавати дані в модель, дуже важливо їх попередньо обробити. Спочатку потрібно перетворити кожне зображення набору даних 224 x 224

у матрицю розміром 224 x 224 x 1, яку потім треба бути передати у мережу.

Далі потрібно перевірити тип даних масиву NumPy; він повинен бути у форматі float32, якщо ні, то потрібно перетворити його в цей формат, адже доведеться масштабувати значення пікселів у діапазоні 0-1 включно. Також потрібно перевірити максимальне та мінімальне значення пікселів, воно потрібно бути 0,0 та 1,0. Після всього цього важливо розділити дані. Для того, щоб модель добре узагальнила знання, потрібно розділити дані на дві частини: набір для навчання та набір для перевірок. Модель буде навчатися на 80% даних і перевіряти себе на 20% з решти навчальних даних. Це також допоможе зменшити шанси «перенавчання», оскільки модель будете перевіряти себе на даних, яких вона не бачила на етапі навчання. Для правильного та довільного розподілу даних було використано функцію train_test_split з модулю sklearn.

Зображення мають розмір 224 x 224 x 1 або представляють собою 50176-мірний вектор. Матриця зображень була перетворена у масив, масштабована між 0 і 1 та переформована так, щоб вона мала розмір 224 x 224 x 1, результат цих трансформацій і буде передано на вхід до мережі. Тренування мережу буде проходити протягом 50 епох.

Сам же автоенкодер поділяються на 2 частини: енкодер та декодер. Енкодер буде включати в себе 3 шари (layers). Перший шар матиме 32 фільтри розміром 3 x 3, а потім шар зменшення вибірки (downsampling, max-pooling), другий шар матиме 64 фільтри розміром 3 x 3, а потім ще один шар зменшення вибірки, кінцевий шар енкодера матиме 128 фільтрів розміром 3 x 3. Декодер також буде включати в себе 3 шари. Перший шар матиме 128 фільтрів розміром 3 x 3, а потім шар збільшення вибірки (upsampling), другий шар матиме 64 фільтри розміром 3 x 3, а потім інший шар збільшення вибірки, кінцевий шар декодера матиме один фільтр розміром 3 x 3. Шар максимального об’єднання (max-pooling layer) буде зменшувати вибірку вводу в два рази кожного разу, коли буде

використовуватися, тоді як шар збільшення вибірки буде збільшувати вибірку вводу в два рази кожного разу, коли він використовується. Кількість фільтрів, розмір фільтра, кількість шарів, кількість епох, в процесі тренування моделі, – це всі гіперпараметри, які можуть бути довільно змінені.

Також потрібно вказати тип втрати через аргумент втрати у моделі. У цьому випадку була вибрана середня квадратична помилка, оскільки втрати після кожної партії будуть обчислюватися між партією прогнозованого виходу та основною істиною, використовуючи середньоквадратичну помилку піксель за пікселем. Після тренування моделі можна побудувати графік втрат між тренуванням та даними перевірки, щоб візуалізувати ефективність моделі. Графік представлено на рисунку 3.4.

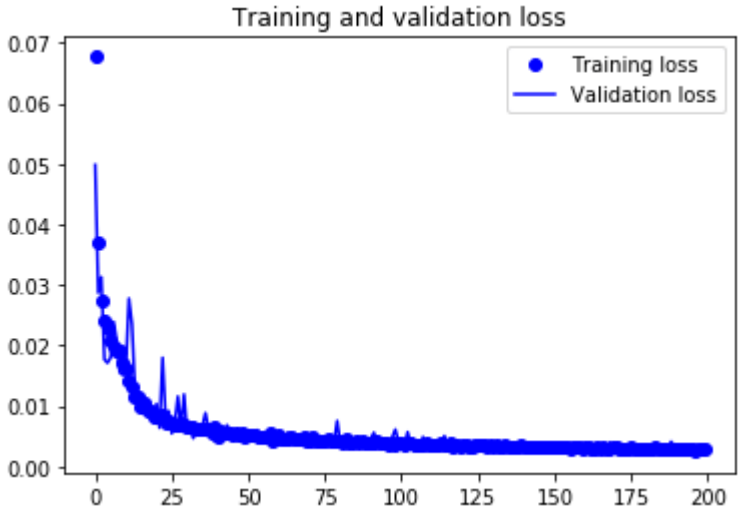

Рисунок 3.4 – Графік втрат між тренуванням та даними перевірки

Можна побачити, що втрата перевірки та втрата тренувань синхронізуються. Це показує, що модель не «перенавчена»: втрати при валідації зменшуються і рідко буває розрив між навчанням і втратою перевірки, особливо, після 40-ї епохи навчання. Отже, можна сказати, що можливості узагальнення інформації моделі є хорошими. Робота моделі з

реконструювання відбитків була перевірена на зображеннях, які були відмічені як зображення для валідації, тобто модель не тренувалася на цих зображеннях, що робить тест більш прозорим. На рисунках 3.5 та 3.6 зображено графіки відбитків до реконструкції та після відповідно.

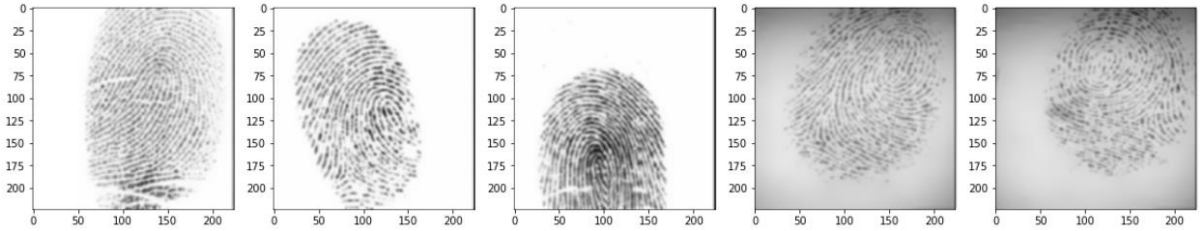

Рисунок 3.5 – Тестові зображення

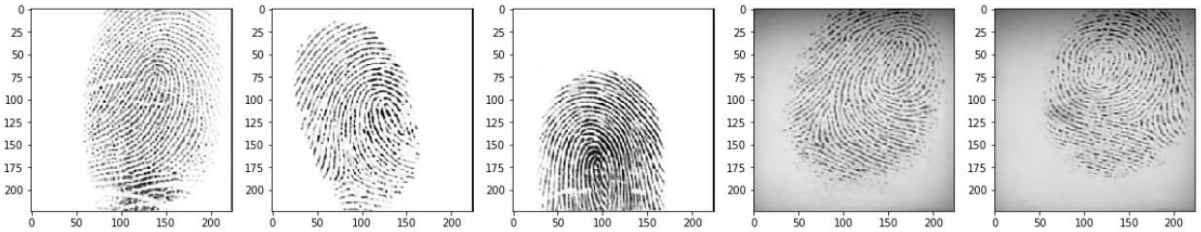

Рисунок 3.6 – Зображення після реконструкції

3.3 Процесна модель клієнтського додатку

IDEF0 – методологія функціонального моделювання і графічного описання процесів, призначена для формалізації і опису бізнес-процесів. Особливістю IDEF0 є її акцент на ієрархічне представлення об'єктів, що значно полегшує розуміння предметної області. В IDEF0 розглядаються логічні зв'язки між роботами, а не послідовність їх виконання в часі.

Методологія IDEF0 передбачає побудову ієрархічної системи діаграм – одиничних описів фрагментів системи. Спочатку проводиться опис системи в цілому і її взаємодії з навколишнім світом (контекстна діаграма), після чого проводиться функціональна декомпозиція – система

розбивається на підсистеми і кожна підсистема описується окремо (діаграми декомпозиції). Потім кожна підсистема розбивається на дрібніші і так далі до досягнення потрібної міри деталізації.

Функціональна модель IDEF0 являє собою набір блоків, кожен з яких представляє собою «чорний ящик» з входами і виходами, управлінням та механізмами, які деталізуються (декомпозуються) до необхідного рівня. Найбільш важлива функція розташована у верхньому лівому кутку. А з'єднуються функції між собою за допомогою стрілок і описів функціональних блоків. При цьому кожен вид стрілки або активності має власне значення. Дана модель дозволяє описати всі основні види процесів, як адміністративні, так і організаційні.

Стрілки можуть бути:

– вхідні – вступні, які ставлять певне завдання;

– вихідні – виводять результат діяльності;

– керуючі (зверху вниз) – механізми управління (положення, інструкції тощо);

– механізми (від низу до верху) – що використовується для того, щоб зробити необхідну роботу.

Вхідні та вихідні стрілки точніше було б називати такими, що вводять і виводять, так як по-англійськи вони називаються Input і Output відповідно. Але особливості перекладу і звичні назви виглядають вже так, як склалося. І все ж для правильного розуміння термінів важливо пам'ятати їх значення в даному випадку. Це підтверджується ще і тим, що дана нотація створена насамперед для розробки ПО, і терміни переводити правильніше в цій точки зору.

Стрілки підписуються за допомогою іменників (досвід, план, правила), а блоки – за допомогою дієслів, тобто в них описуються дії, які виробляються (створити товар, укласти договір, провести відвантаження).

IDEF0 – це дуже простий і одночасно наочний мова опису бізнес-процесів. За допомогою цього стандарту можлива передача інформації між

розробниками, консультантами та користувачами. Стандарт дуже ретельно розроблявся, він зручний для проектування, універсальний.

Побудована для даної предметної області IDEF0-схема представлена на рисунку 3.7.

Перший рівень декомпозиції контекстної діаграми процесу оплати проїзду в міському транспорті на основі біометричних даних представлений на рисунку 3.8.

Декомпозиція процесу отримання біометричних даних представлена на рисунку 3.9.

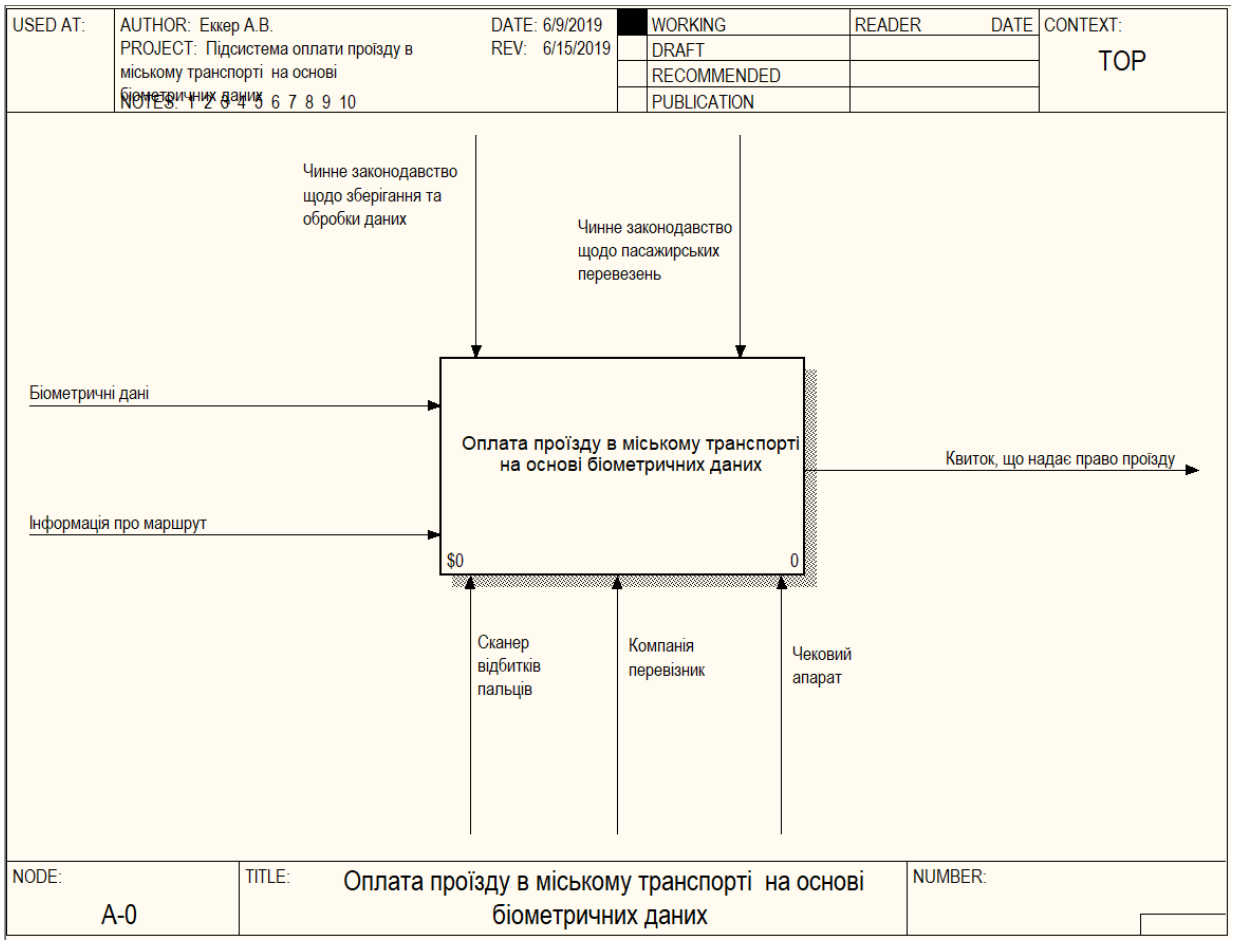

Рисунок 3.7 – Контекстна діаграма процесу оплати проїзду в міському транспорті на основі біометричних даних

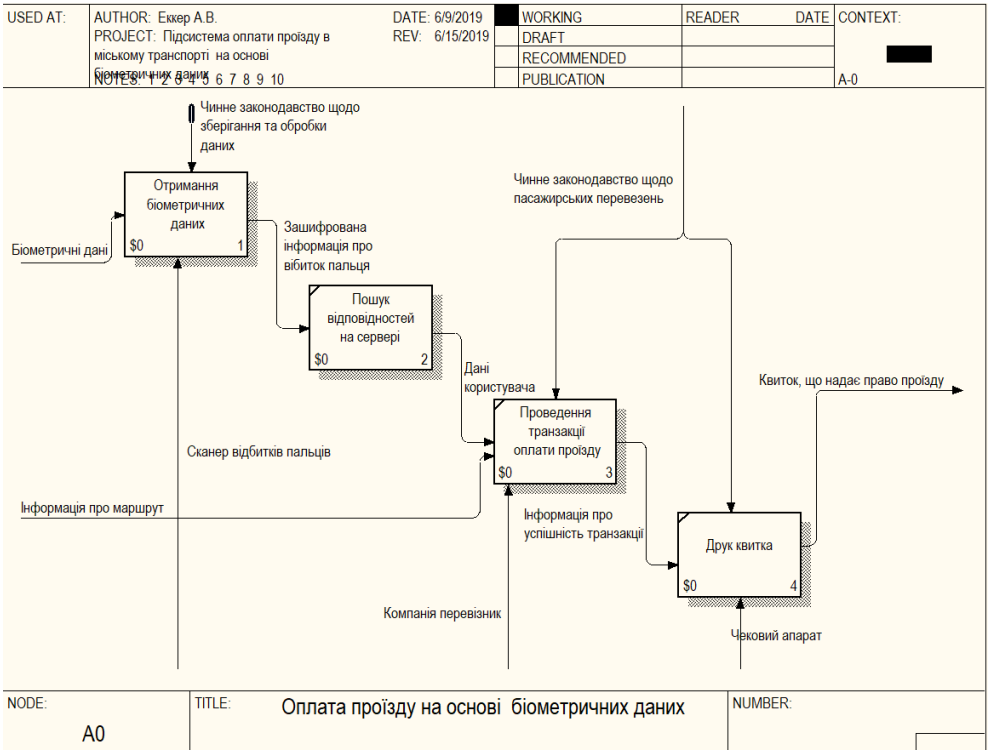

Рисунок 3.8 – Результат декомпозиції першого рівня процесу оплати проїзду в міському транспорті на основі біометричних даних

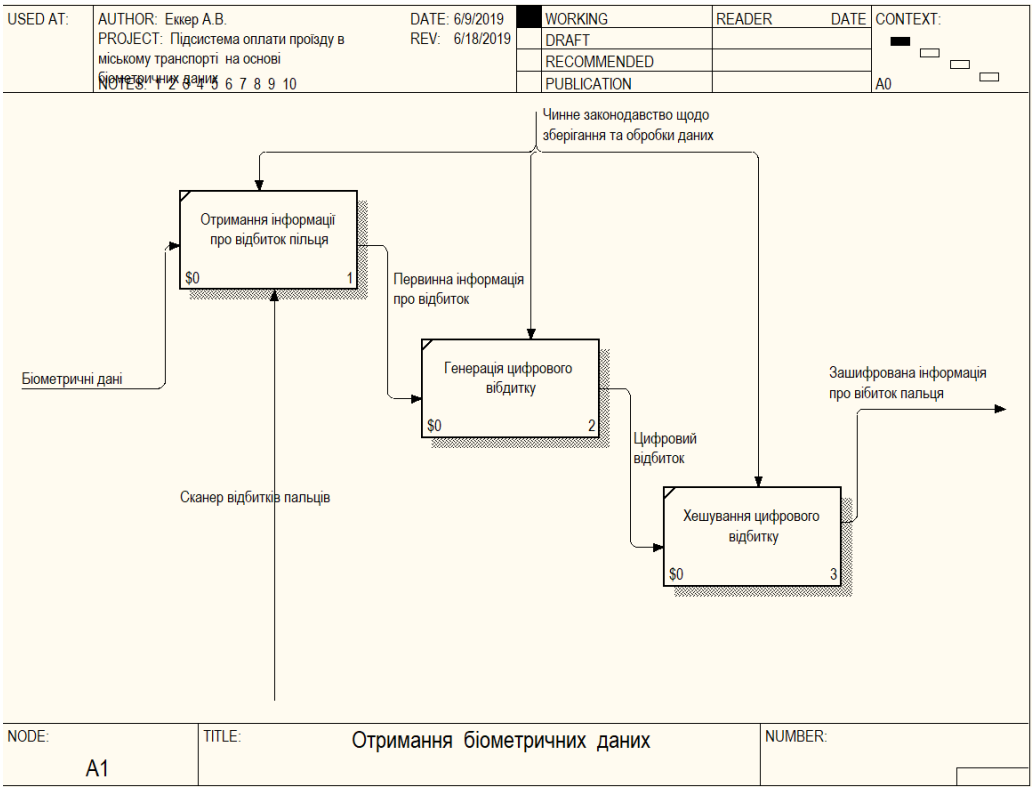

Рисунок 3.9 – Декомпозиція процесу «Отримання біометричних даних»

Також була побудована діаграма послідовності основного бізнес процесу – сплати проїзду у міському транспорті на основі біометричних даних. Діаграма послідовності – різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень.

На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення. Визначені стандартом UML 2.0 діаграми послідовностей мають ті ж можливості що і визначені стандартом UML 1.x, та підтримують додаткові можливості зміни стандартного порядку повідомлень. На рисунку 3.10 представлена діаграма послідовності сплати проїзду у міському транспорті на основі біометричних даних.

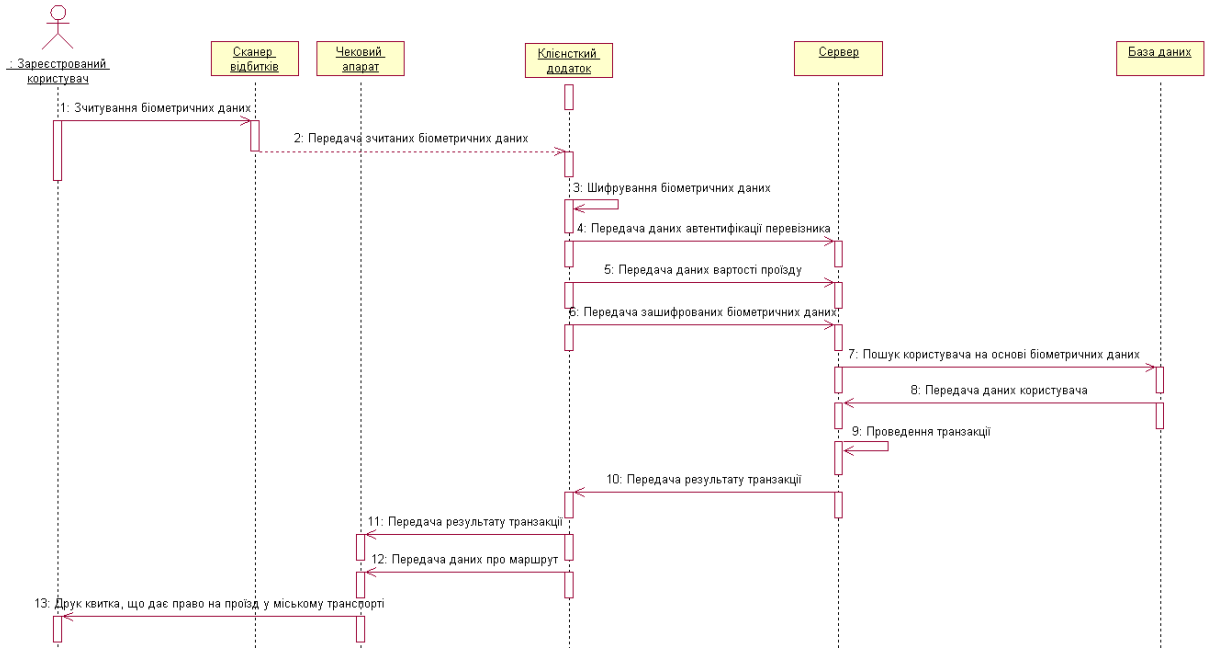

Рисунок 3.10 – Діаграма послідовності сплати проїзду у міському транспорті на основі біометричних даних

3.4 Визначення архітектури додатку

Серверна частина побудована на архітектурі REST. Перевагами REST є продуктивність, масштабованість, узагальненість, простота та змінність. REST э кращим вибором для розподілених систем. Так як система буде розширюватись – REST буде кращим вибором для високонавантаженої системи с багатьма функціями. Наприклад, якщо за допомогою системи можна буде не тільки сплачувати проїзд, а й переглядами маршрути, та бронювати місця на дальні рейси. Приклад такої не REST системи представлений на рисунку 3.11.

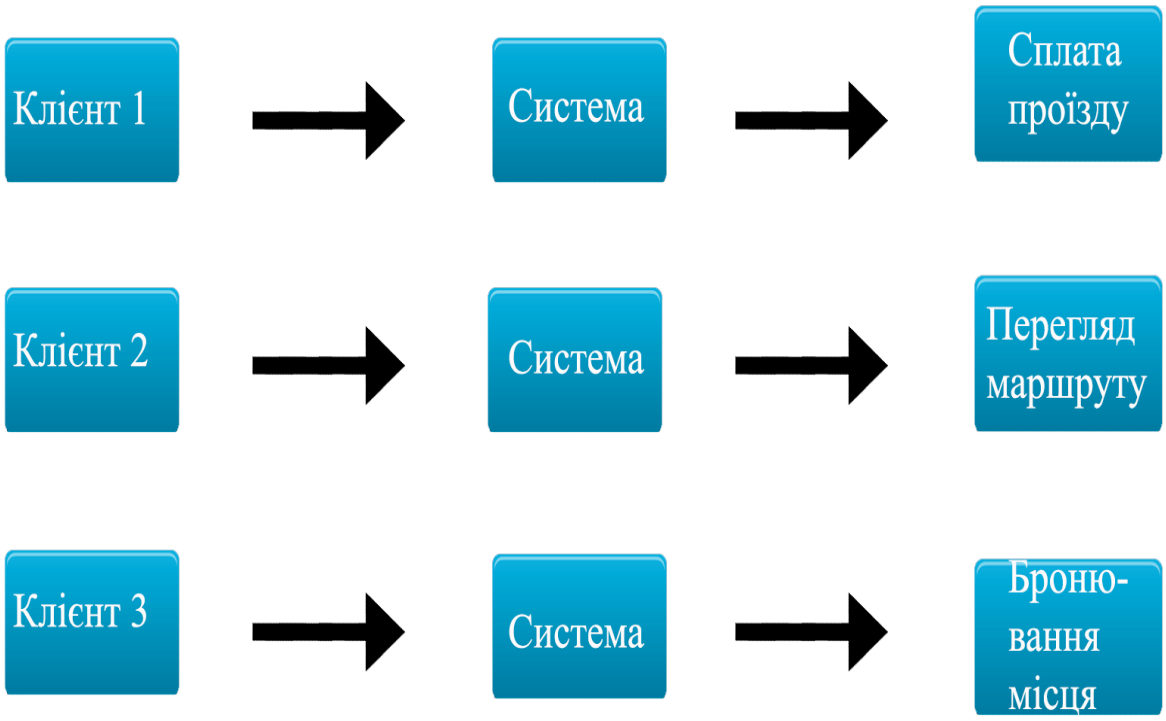

Рисунок 3.11 – Приклад не REST системи

Як видно на рисунку 3.11, у системі є «вузьке місце», через яке будуть проходити усі запити до неї, це не є проблемою для звичайних

систем і майже не впливає на швидкість їх роботи, але коли треба побудувати високонавантажену систему такий підхід є хибним, а деколи навіть фатальним. На рисунку 3.12 приведена система з такими ж функціями як і на рисунку 3.6, проте побудована на архітектурі REST.

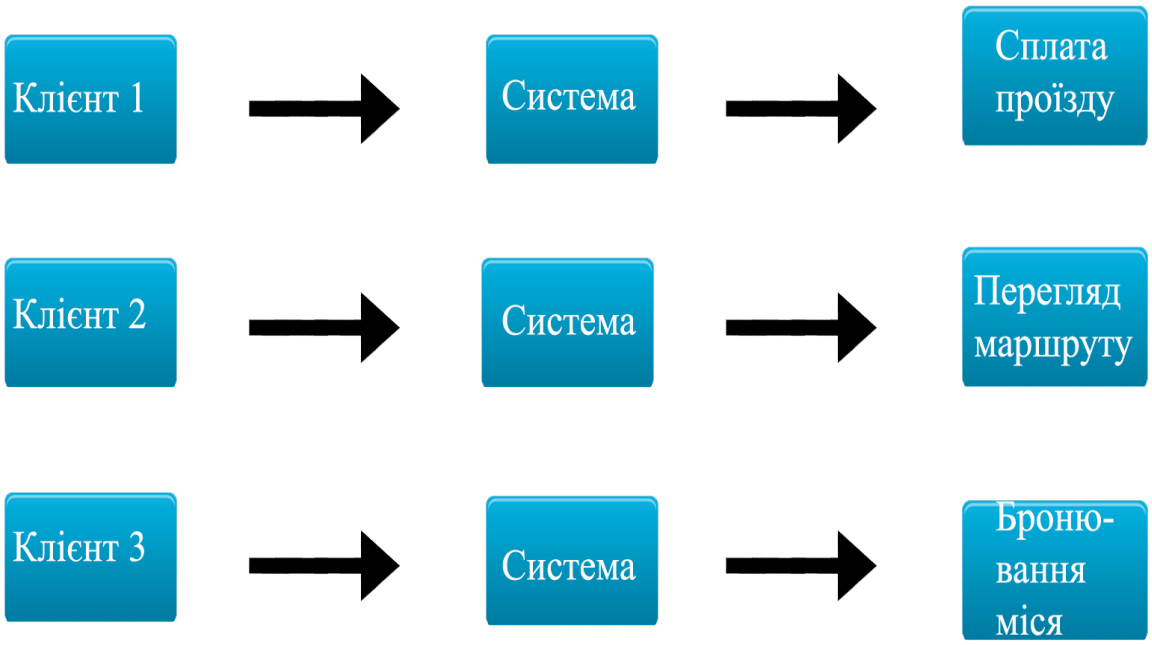

Рисунок 3.12 – Приклад REST системи

Головними плюсами REST систем є: зрозумілий принцип дії сервісу (URL говорить сам за себе), відсутність «вузького місця» у веб-сервісі, легка розстанова пріоритетів – для різних задач різні сервери з різною продуктивністю, виявлення URI пошуковими системами (легкість пошуку в пошукових системах).

Також плюсами REST э: короткі однозначні запроси (URI як ідентифікатор ресурсу, URI має строгий формат), використання HTTP методів як ідентифікаторів CRUD операцій (create – put, read – get, update – post, delete – delete), запроси не акумулюють жодної інформації щодо стану

сервісу, запроси кешуються. У таблиці 3 наведено порівняння SOAP та REST архітектур.

Таблиця 3.1. Порівняння REST та SOAP архітектур

	REST	SOAP
Застосування	Управління ресурсами	Реалізація бізнес–логіки
Формат даних	XML, JSON, тощо	XML
Кешування запитів	Так	Ні
Словник	Заснований на словнику HTTP	Потребує реалізації
Документація	Коротка	Об’ємна

Архітектура клієнтської частини представлена на рисунку 3.13.

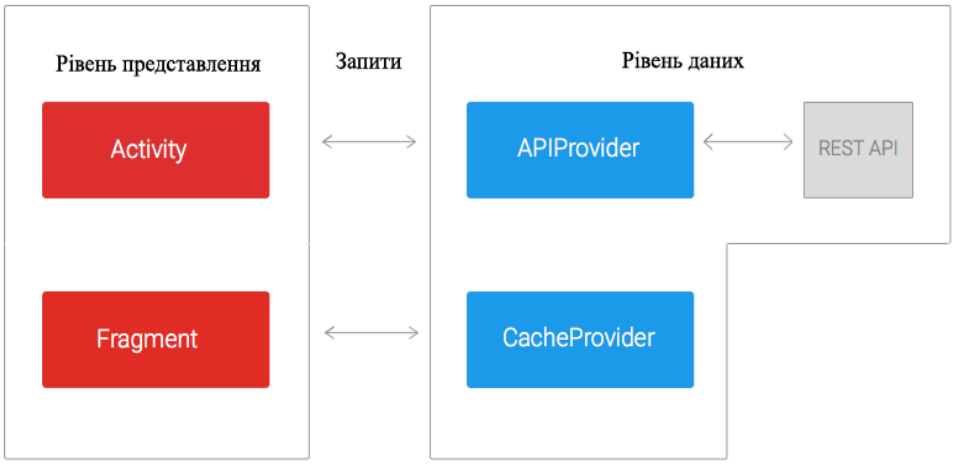

Рисунок 3.13 – Архітектура клієнтської частини підсистеми

APIProvider надає методи, що дозволяють Activity і фрагментам взаємодіяти з REST API. Ці методи використовують URLConnection і

AsyncTask, щоб виконати запит у фоновому потоці, а потім доставити результати в Activity через функції зворотного виклику. Аналогічно працює і CacheProvider: є методи, які дістають дані з SharedPreferences і є функції зворотного виклику, які повертають результати.

3.5 Вибір платформи, мови програмування та СУБД

Вибір платформи розробки визначається рядом різних чинників. Також, кожна платформа, що надає інструменти для розробки мобільних додатків, дотримується не одного мови, а декількох. Це обумовлюється різними вимогами до додатків, і різні мови програмування та додані до них додаткові інструменти дозволяють вирішити завдання різних рівнів.

Для серверної частини була обрана мова програмування Java. По-перше бо Java працює на JVM(Java Virtual Machine). Одна із найсильніших сторін віртуальної машини Java, завжди була її здатність з легкістю «жонглювати» декількома потоками. JVM оптимізована для великих багатоядерних машин, і вона без проблем може керувати сотнями потоків. Завдяки цій здатності, на JVM з'явилися і інші мови – створюються крос-компілятори і емулятори, що працюють поверх JVM.

Ці можливості використовуються багатьма веб-сайтами та веб-серверами з високою відвідуваністю.

Ruby є одним із сучасних конкурентів Java. У нього більш чистий, що нагадує справжню англійську мову, синтаксис. Але все ж, коли любителям Ruby потрібна висока продуктивність, вони використовують JRuby. Це версія Ruby, яка біжить поверх JVM, забезпечуючи набагато більш високу продуктивність при великих навантаженнях з безліччю потоків. Проте у Ruby немає такої кількості бібліотек, у порівнянні з Java, тому у виборі між Java та Ruby все ж більш цікавим варіантом є Java. Проте така логіка розповсюджується далеко не на весь перелік можливих

задач, у деяких, наприклад у веб-розробці – Ruby фаворит з фреймворком RubyOnRails.

При використанні мови, яка запускається на JVM інженери з Sun та Oracle роблять свою частину роботи зі створення крос-платформної середовища, а всі інші користуються цим. Інженери Sun та Oracle зачісують платформу і турбуються про сумісність.

Microsoft запозичила цю ідею (а також багато іншого), створивши C# і свій підхід до створення компіляторів для мов, які працюють на C# VM (CLR). C# програмісти кажуть, що можуть писати на різних мовах – правда, тільки на VM під Windows, що не додає гнучкості у розробці та розгортці додатків, тому C# також не був би кращим вибором, та навряд зміг би конкурувати з Java у контексті гнучкості.

Також слід взяти до уваги популярність мови Java. Популярність мови впливає на кількість готових рішень для проблем, що зустрічаються при розробці додатків, будь то веб чи десктопні додатки. Це значно пришвидшує розробку, адже не потрібно писати та тестувати деякі методи, або навіть модулі додатків, можна взяти готове рішення та імпортувати його у свій проект. На рисунку 3.14 представлений графік популярності мов програмування в Україні та близькому зарубіжжі за 2021 рік по даним dou.ua.

Ще можна відзначити мову Python. Python є високорівневою мовою широкого застосування. Автор мови (Ґвідо ван Россум) зробив неабиякий акцент на читабельності коду та синтаксисі. Мова дозволяє програмісту реалізувати задачі із значно меншою кількістю рядків коду, ніж потрібно у інших мовах таких як C++ чи Java. Мова підтримує популярні серед інших мов програмування парадигми: ООП, імперативне та функціональне програмування. Python є динамічно типізованим. Включає у себе автоматичне управління пам’яттю та надає масу додаткових бібліотек. Але ця мова є інтерпритованою, тобто вона не компілюється у машинний код,

саме тому Python, здебільшого, програє за швидкістю компільованим мовам програмування, однією з яких є Java.

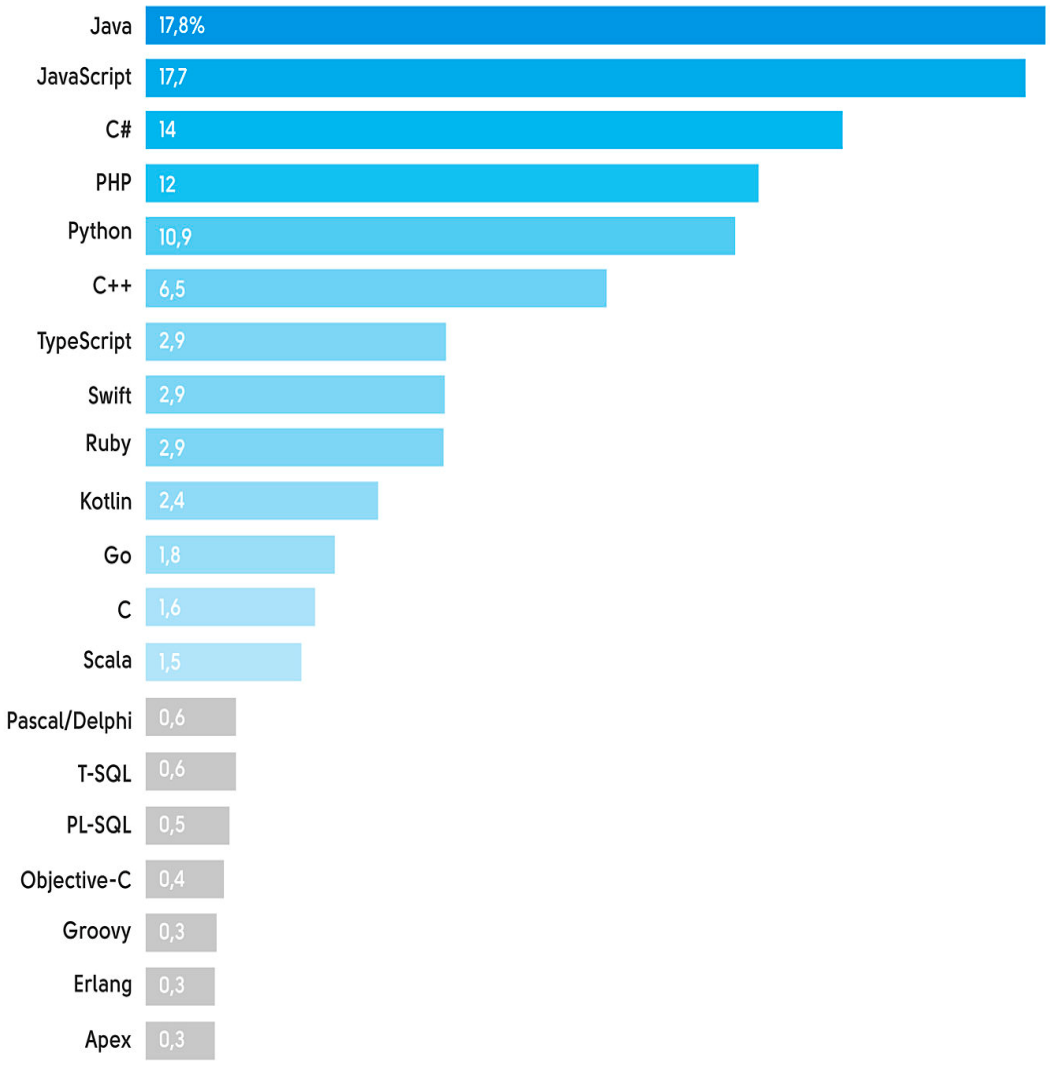

Рисунок 3.14 – Популярність мов програмування

Ці слова також підтверджує кількість артефактів, у репозиторії Maven їх знаходиться понад чотирнадцять з половиною мільйонів. Авжеж не всі їх можна використати з Java, але така кількість готових рішень для майже будь-яких задач вражає.

Також для написання серверної частини був використаний фреймворк Spring. Основна перевага Spring – можливість розробки програми як набору слабо зв’язаних компонентів. Чим менше компоненти програми знають один про одного, тим простіше розробляти новий і підтримувати існуючий функціонал додатку. Класичний приклад – управління транзакціями. Spring дозволяє керувати транзакціями абсолютно незалежно від основної логіки взаємодії з БД. Зміна цієї логіки не порушить транзакційність, так само як зміна логіки управління транзакціями не зламає логіку програми. Також Spring заохочує до модульності. Компоненти можна додавати і видаляти (майже) незалежно один від одного. Також Spring помітно спрощує модульне тестування : в компонент, розроблений для роботи в IoC(Inversion of Control) контейнері дуже легко інжектувати несправжні залежності і перевірити роботу тільки цього компонента. Ну, і як приємне доповнення, Spring сильно полегшує ініціалізацію і настройку компонентів додатка, дозволяючи гнучко налаштовувати додаток без істотних змін Java-коду.

Як СУБД був використаний PostgreSQL. PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає йому деякі переваги над іншими SQL базами даних з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird. Фундаментальна характеристика об'єктно-реляційної бази даних – це підтримка об'єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить PostgreSQL неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних.

Існує великий список типів даних, які підтримує PostgreSQL. Крім числових, з плаваючою точкою, текстових, булевих і інших очікуваних типів даних (а також безлічі їх варіацій), PostgreSQL може похвалитися підтримкою uuid, грошового, що перераховується, геометричного, бінарного типів, мережевих адрес, бітових рядків, текстового пошуку, xml,

json , масивів, композитних типів і діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів і розташування балок.

PostgreSQL забезпечує зберігання різних типів мережевих адрес. Тип даних CIDR (безкласова маршрутизація інтернет домену, Classless Internet Domain Routing) слідує угоді для мережевих адрес IPv4 і IPv6. Ось кілька прикладів: 192.168.100.128/20, 10.1.2.3/34, 99.1.2.3/99, ::ffff:1.2.1.0/128, 2001:4f8:3:ba:2e0:81ff:qe22:z1f1/128. Також для зберігання мережевих адрес доступний тип даних INET, який використовується для IPv4 і IPv6 хостів, де підмережі є необов'язковими. Тип даних MACADDR може використовуватися для зберігання MAC-адрес для ідентифікації обладнання, таких як 08-00-2b-01-02-03.

Оскільки PostgreSQL – це об'єктно-реляційна база даних, масиви значень можуть зберігатися для більшості існуючих типів даних. Зробити це можна шляхом додавання квадратних дужок до специфікації типу даних для стовпця або за допомогою виразу ARRAY. Розмір масиву може бути заданий, але це необов'язково. MySQL, MariaDB, і Firebird так не вміють. Щоб зберігати такі масиви значень в традиційних реляційних базах даних, доведеться використовувати обхідний шлях і створювати окрему таблицю з рядками для кожного із значень масиву.

Геодані швидко стають основною вимогою для багатьох додатків. PostgreSQL вже давно підтримує безліч геометричних типів даних, таких як точки, лінії, кола і багатокутники. Один з цих типів – PATH, він складається з безлічі послідовно розташованих точок і може бути відкритим (початкова та кінцева точки не пов'язані) або закритим (початкова та кінцева точки пов'язані). У MySQL 5.7.8 і в MariaDB, починаючи з версії 5.3.3, були додані розширення типів даних для підтримки стандарту географічної інформації OpenGIS. Ця версія MySQL і наступні версії MariaDB пропонують зберігання типів даних, аналогічне штатним геоданим PostgreSQL. Проте, в MySQL і MariaDB значення даних спочатку повинні бути конвертовані в геометричний формат простими

командами перед тим, як будуть вставлені в таблицю. Firebird на даний момент не підтримує геометричні типи даних.

Підтримка JSON в PostgreSQL дозволяє вам перейти до зберігання schema-less даних в SQL базі даних. Це може бути корисно, коли структура даних вимагає певної гнучкості: наприклад, якщо в процесі розробки структура все ще змінюється або невідомо, які поля буде містити об'єкт даних. Тип даних JSON забезпечує перевірку коректності JSON, який дозволяє використовувати спеціалізовані JSON оператори і функції, вбудовані в PostgreSQL для виконання запитів і маніпулювання даними.

Також доступний тип JSONB – двійковий різновид формату JSON, у якому прогалини видаляються, сортування об'єктів не зберігається, натомість вони зберігаються найбільш оптимальним чином, і зберігається тільки останнє значення для ключів-дублікатів. JSONB зазвичай є кращим форматом, оскільки вимагає менше місця для об'єктів, може бути проіндексований і обробляється швидше, так як не вимагає повторного синтаксичного аналізу. В MySQL 5.7.8 і MariaDB 10.0.1 була додана підтримка вбудованих об'єктів JSON. Але, хоча існує безліч функцій і операторів для JSON, які тепер доступні в цих базах даних, вони не індексуються так, як JSONB в PostgreSQL. Firebird поки що не долучився до тренду і підтримує об'єкти JSON тільки у вигляді тексту.

Якщо раптом так трапиться, що великого списку типів даних PostgreSQL виявиться недостатньо, можна використовувати команду CREATE TYPE, щоб створити нові типи даних. Оскільки вони не є об'єктно-реляційними, MySQL, MariaDB і Firebird не пропонують таку потужну функціональність.

PostgreSQL може обробляти багато даних. Поточні опубліковані обмеження наведені у таблиці 3.2.

Щодо цілісності даних – PostgreSQL прагне відповідати стандарту ANSI-SQL: 2008, відповідає вимогам ACID (атомарність, узгодженість, ізольованість і надійність) і відомий своєю посилальної і трансакційною

цілісністю. Первинні ключі, обмежуючі і каскадні зовнішні ключі, унікальні обмеження, обмеження NOT NULL, перевірочні обмеження та інші функції забезпечення цілісності даних дають впевненість, що тільки коректні дані будуть збережені.

Таблиця 3.2 – Обмеження PostgreSQL

Максимальний розмір бази даних	Необмежений
Максимальний розмір таблиці	32 TB
Максимальний розмір строки	1.6 TB
Максимальний розмір поля	1 GB
Максимальна кількість рядків в таблиці	Необмежено
Максимальна кількість стовпців в таблиці	250-1600 залежно від типу стовпця
Максимальна кількість індексів в таблиці	Необмежено

MySQL і MariaDB більше працюють на те, щоб відповідати стандарту SQL з двигунами таблиць InnoDB / XtraDB. Тепер вони пропонують опцію STRICT з використанням режимів SQL, яка встановлює перевірки коректності використовуваних даних.

Незважаючи на це, в залежності від того, який режим використовувати, недостовірні і навіть урізані без вашого відома дані можуть бути вставлені або створені при оновленні. Жодна з цих баз даних зараз не підтримує CHECK обмеження. Крім того, у них існує безліч особливостей щодо обмежень посилальної цілісності по зовнішнім ключам. Також цілісність даних може істотно постраждати в залежності

від обраного двигуна зберігання. MySQL (і fork MariaDB) не роблять секрету з того, що проміняли цілісність і відповідність стандартам на швидкість і ефективність.

Також для міграції бази даних був використаний Flyway. Flyway це інструментарій для супроводу баз даних і синхронізації їхньої структури з пов'язаним із нею програмним забезпеченням. Flyway можна розглядати як аналог системи контролю версій для БД, який виконує завдання автоматизації відображення змін у структурі бази даних для відповідності версії БД і версії програмного забезпечення, що працює з цією БД.

Іншими словами, Flyway дозволяє прив'язати стан структури БД до версії застосунку і змінювати цю структуру в залежності від обраної версії програми. Наприклад, при переході на нову версію програми Flyway дозволяє на всіх серверах привести схему зберігання даних до нової версії. Flyway також дає можливість швидко дізнатися якій версії ПЗ відповідає наявна БД, перевірити цілісність схеми і в разі порушення структури (наприклад, ручного додавання / видалення поля) виправити схему.

Клієнтська частина була написана на Java Android. Java Android був вибраний майже с тих самих причин що і Java, хоча має доволі «сильного» конкурента – Kotlin. Java Android додаток збирається швидше за допомогою Gradle ніж Kotlin, та має дуже велику екосистему готових рішень, але вирішальним стало те, що Java додатки компактніше в порівнянні з додатками, написаними на Kotlin а також потребують менше ресурсів пристрою, на якому запускаються.

Так як клієнтський додаток планується запускати на пристроях з Android GO, що не можуть похизуватися великим обсягом пам’яті та обчислювальними здібностями вибір пав на Java Android.

3.6 Змістовний опис і аналіз використовуваних інформаційних технологій

Як було зазначено вище – операційна система першої версії клієнтської частини – android. Вона була обрана через присутність в ній деяких реалізацій для зчитування відбитку пальців, які проявили себе як одні з найнадійніших.

Серверна ж частина буде реалізована на мові Java через її кросплатформенність та припускати запуск серверного програмного забезпечення в контейнерах Docker так як у Docker нього досить багато плюсів для даного рішення: більш просте викладання і розгортання – заснована на контейнерах Docker платформа дозволять легко контролювати корисне навантаження. Docker контейнери можуть працювати на локальній машині, як реальної так і на віртуальній машині в дата центрі, так і в хмарі, що сприятливо позначиться на впровадженні, так як серверна частина може бути розгорнути майже на будь-яких, вже наявних машинах і не потребуватиме додаткового фінансування. Легкий процес переносу програмного забезпечення Docker дозволяє легко динамічно управляти навантаженням. Можливо використовувати Docker, щоб розгорнути або погасити додаток або сервіси. Швидкість контейнерів Docker дозволяє робити це майже в режимі реального часу, що сприятливо позначиться на процесі розгортання програми.

Docker надає стійку, рентабельну альтернативу віртуальним машинам на основі гіпервізора. Він особливо корисний в умовах високих навантажень, наприклад, при створення власного хмари або платформи-як-сервісу (platform-as-service). Але він так само корисний для маленьких і середніх додатків, коли вам хочеться отримувати більше з наявних ресурсів. Це одні з найважливіших чинників вибору саме контейнерів Docker, так як вони чудово зможуть показати свою продуктивності і на етапі тестування програми, коли навантаження буде не настільки велике, і

на етапі безпосереднього впровадження програмного забезпечення в експлуатацію, коли система буде під високим навантаженням.

Docker використовує архітектуру клієнт-сервер. Docker клієнт спілкується з демоном Docker, який бере на себе тягар створення, запуску, розподілу контейнерів. Обидва, клієнт і сервер можуть працювати на одній системі, ви можете підключити клієнт до віддаленого демона Docker. Клієнт і сервер спілкуються через сокет або через RESTful API. Останній важливий фактор використання Docker – він призначений саме для архітектури клієнт-сервер а так само підтримує запуск декількох контейнерів на одній машині, якщо її ресурси дозволяють.

Ідеальним рішенням буде використання docker cotainer і вивантаження даних з основної бази з твердотільного накопичувача в in-memory database H2, яка працює значно швидше, так як зберігається в оперативній пам'яті, але це спричинить великі витрати на обладнання і не розглядається як основне рішення через своєю надмірною вартості, хоча і дає прибавку в швидкості відгуку в кілька разів.

Як і було зазначено вище, система складається з двох частин – клієнтськоїта серверної.

Серверна частина реалізована на мові Java з використанням фреймворку Spring. Також були використані PostgreSQL як база даних та flyway для міграції бази даних. Flyway дозволяє прив'язати стан структури БД до версії застосунку і змінювати цю структуру в залежності від обраної версії програми. Наприклад, при переході на нову версію програми Flyway дозволяє на всіх серверах привести схему зберігання даних до нової версії. Flyway також дає можливість швидко дізнатися якій версії ПЗ відповідає наявна БД, перевірити цілісність схеми і в разі порушення структури (наприклад, ручного додавання / видалення поля) виправити схему. Серверна частина задокументована та має dockerfile для простішого її розгортання у Docker контейнері. Серверна частина використовує Spring

Authentication для автентифікації користувачів та Spring Boot для розгортання серверної частини.

Клієнтська частина реалізована на мові Java, та використовує android sdk. Клієнтська частина зчитує відбиток користувача, відправляє його на сервер, де проходить пошук по відбитку, потім сервер проводить транзакцію сплати проїзду та пересилає клієнтській частині відповідь, яку та обробляє та виводить на екран.

Реалізована система може бути легко розширена через свою структуровану архітектуру, а база даних легко підлягає змінам через використання flyway. Для подальшого розвитку системи пропонується додати до реалізації детальну інформацію про маршрути, розклад та водіїв.

Також для серверної частини був використаний Spring Framework. Spring це універсальний фреймворк з відкритим кодом для платформи Java. Spring Framework став популярним як заміна моделі Enterprise JavaBeans. Spring може будети розглянутий як колекція менших фреймворків або фреймворків у фреймворку. Більшість цих фреймворків може працювати незалежно один від одного, проте вони забезпечують більшу функціональність при спільному їх використанні. Ці фреймворки діляться на структурні елементи типових комплексних програм. Фреймворки, що були використані у проекті наведені у таблиці 3.3.

Таблиця 3.3 – Використані Spring фреймворки

Фреймворк	Опис фреймворку
Inversion of Control-контейнер	Конфігурація компонентів додатків і управління життєвим циклом Java-об'єктів
Inversion of Control-контейнер	Конфігурація компонентів додатків і управління життєвим циклом Java-об'єктів

Продовження таблиці 3.3

Фреймворк	Опис фреймворку
Фреймворк доступу до даних	Працює з системами керування базами даних на Java-платформі, використовуючи JDBC- і ORM-засоби і забезпечуючи вирішення завдань, які повторюються в великому числі Java-based environments
Фреймворк управління транзакціями	Координація різних API керування транзакціями і інструментарій настроюваного управління транзакціями для об'єктів Java
Фреймворк MVC	Каркас, заснований на HTTP і сервлетах, що надає безліч можливостей для розширення та налаштування (customization)
Фреймворк аутентифікації і авторизації	Конфігурується інструментарій процесів аутентифікації і авторизації, що підтримує багато популярних і стали індустріальними стандартами протоколів, інструментів, практик через дочірній проект Spring Security (раніше відомий як Acegi)
Тестування	Каркас, що підтримує класи для написання модульних і інтеграційних тестів

Для тестування серверної частини також був використаний JUnit. JUnit це бібліотека для модульного тестування програмного забезпечення на мові Java. JUnit був використаний для тестування методів, описаних у серверній частині програмного забезпечення для сплати проїзду у міському транспорті на основі біометричних даних.

3.7 Вибір, розробка й обґрунтування технічного забезпечення підсистеми

Серверна частина системи повинна бути розгорнута на окремому веб-сервері, спочатку (для стадії) тестування системи не потрібно складної системи, як наприклад кластера, тому для тестового запуску можна розгорнути її майже на будь-якому веб-хостингу, проте кращим виходом будуть веб-сервери від Amazon, які, у разі потреби, можна розширювати майже безкінечно.

Наприклад для розгортання веб-серверу краще за все буде використати C5 instances сервери від Amazon, вони оптимізовані під процес обчислення, можуть бути налаштовані на використання обмеженої кількості ресурсів, що звільнить від переплат та мають невелику ціну за співвідношенням обчислення/ціна. Так же за необхідністю можливо буде переключитися с EBS, який працює на звичайних SSD зі швидкістю приблизно 600мб/с на NVMe SSD приблизно 3гб/с. Характеристики різних типів C5 веб-серверів наведено у таблиці 3.4.

Для розгортання бази даних пропонується використати I3 веб-сервери Amazon, які спроектовані під зберігання великих баз даних. Це сімейство веб-серверів забезпечує зберігання даних, що використовує Memory Express (NVMe) SSD, оптимізованого для низьких затримок, дуже високої випадкової продуктивності вводу та виводу та високої послідовної пропускної здатності читання.

Таблиця 3.4 – Характеристики веб-серверів C5 від Amazon

Тип (назва) сервера	Кількість віртуальних процесорів	Оперативна пам’ять (GB)	Жорсткий диск	Швидкість хмарного сховища EBS (Mbps)	Пропускна здатність мережі (Gbps)
c5.large	2	4	EBS-Only	Up to 3,500	Up to 10
c5.xlarge	4	8	EBS-Only	Up to 3,500	Up to 10
c5.2xlarge	8	16	EBS-Only	Up to 3,500	Up to 10
c5.4xlarge	16	32	EBS-Only	3,500	Up to 10
c5.9xlarge	36	72	EBS-Only	7,000	10
c5.18xlarge	72	144	EBS-Only	14,000	25
c5d.large	2	4	50 NVMe SSD	Up to 3,500	Up to 10
c5d.xlarge	4	8	100 NVMe SSD	Up to 3,500	Up to 10
c5d.2xlarge	8	16	200 NVMe SSD	Up to 3,500	Up to 10
c5d.4xlarge	16	32	400 NVMe SSD	3,500	Up to 10
c5d.9xlarge	36	72	900 NVMe SSD	7,000	10
c5d.18xlarge	72	144	900 x 2 NVMe SSD	14,000	25

Характеристики різних типів I3 веб-серверів наведено у таблиці 3.5.

Таблиця 3.5 – Характеристики різних типів I3 веб-серверів від Amazon

Тип (назва) сервера	Кількість віртуальних процесорів	Оперативна пам’ять (GB)	Локальне сховище (GB)	Пропускна здатність мережі (Gbps)
i3.large	2	15.25	1 x 475 NVMe SSD	Up to 10
i3.xlarge	4	30.5	1 x 950 NVMe SSD	Up to 10
i3.2xlarge	8	61	1 x 1,900 NVMe SSD	Up to 10
i3.4xlarge	16	122	2 x 1,900 NVMe SSD	Up to 10
i3.8xlarge	32	244	4 x 1,900 NVMe SSD	10
i3.16xlarge	64	488	8 x 1,900 NVMe SSD	25

Для тестування системи та перших її запусків, коли система буде працювати тільки на вибіркових маршрутах досить буде комбінації з c5.large та i3.large, які з часом можна буде змінювати на більш продуктивні.

Для клієнтської частини досить буде використати мобільні пристрої на операційній системі Android GO. Система android була вибране через масовість її використання та масовість виробництва мобільних пристроїв з даною операційною системою. Масовість виробництва дуже впливає на ціну пристроїв, тому це дозволить заощадити як на самому пристрої, так і на розробці програмного забезпечення для нього, бо всі засоби розробки під android безкоштовні.

А саме пристрої на Android GO дозволяють заощадити в багатьох областях. Насамперед це стосується, звичайно ж, вартості. Усі смартфони на Android Go мають цінник нижче $100. Така політика Google. Цінник хоч

і приємний, але це далеко не основна частина економії. Ми також говоримо про економію внутрішнього простору та ресурсів смартфона.

Усі застосунки, які оптимізовані для цієї операційної системи, займають у два та більше разів менше місця в пам’яті пристрою. До того ж споживають значно менше оперативної пам’яті, та й висока процесорна потужність їм не потрібна. Висновок напрошується сам собою – застосунки з позначкою «Go» працюватимуть у рази спритніше, ніж аналогічні «повноцінні» застосунки на старіших версіях Android.

У таблиці 6 наведено приклади мобільних пристроїв на операційній системі Android GO.

Таблиця 3.6 – Характеристики мобільних пристроїв на базі операційної системи Android GO

Назва	Процесор	Оперативна пам’ять (GB)	Пам’ять (GB)	Вага (грам)	Ціна ($)
Lava Z50	1.1GHz quad-core MediaTek MT6737M	1	8	140	65
Micromax Bharat Go	1.1GHz quad-core MediaTek MT6737M	1	8	130	65
Huawei Y3 2018	1.1GHz quad-core MediaTek MT6737M	1	8	175	75
Alcatel 1X	1.3GHz quad-core MediaTek MT6739	1	16	151	99

3.8 Методичні рекомендації щодо використання підсистеми

По-перше користувачу потрібно зареєструватись у системі, після реєстрації його дані а також відбиток пальцю буде зберігатись у hash-подібному рядку. Приклад формату зберігання відбитку у hash-подібному рядку представлено на рисунку 3.15.

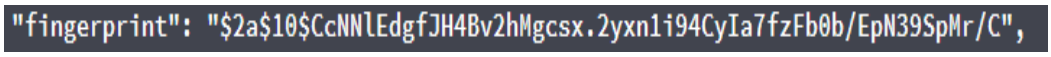

Рисунок 3.15 – Приклад формату зберігання відбитку на сервері

Також компанія перевізник або водій транспорту при першому запуску клієнтського додатку повинен ввести номер маршруту, ціну проїзду а також логін та пароль, без яких клієнтський додаток не зможе підключитися до сервера. На рисунках 3.17 та 3.18 представлено екран входу до клієнтського додатку.

Після цього зареєстровані користувачі можуть проводити оплату відбитком пальця, приклавши його до сканеру відбитків. Дані про ціну маршруту зчитуються з даних, введених перевізником при першому запуску клієнтського додатку. Тестові дані користувачів представлені на рисунку 3.16.

Рисунок 3.16 – Тестові дані користувача

Клієнтський додаток також повідомляє про успішність транзакції. Він виводить інформацію про успішність авторизації користувача та успішність проведення транзакції, або виводить повідомлення про помилку. На рисунку 3.19 представлено вид клієнтського додатку після вводу первинних даних перевізником, а приклад успішної та невдалої сплати проїзду представлено на рисунках 3.20 та 3.21 відповідно.

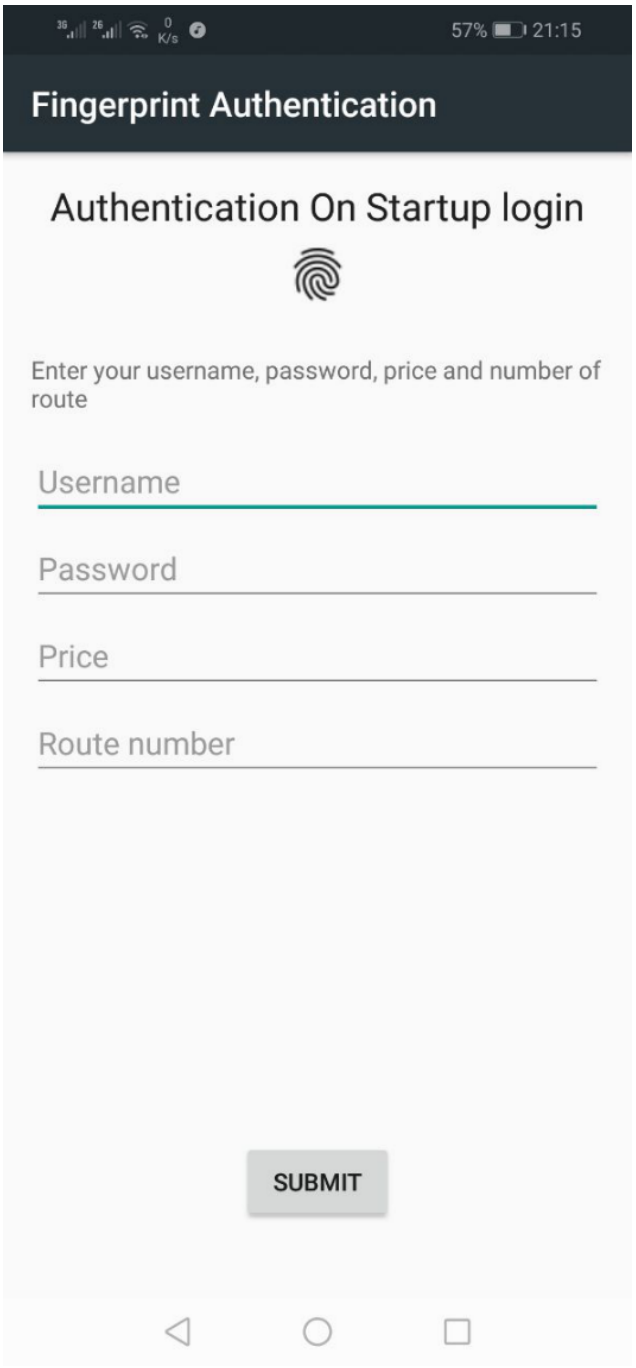

Рисунок 3.17 – Вид клієнтського додатку після першого входу

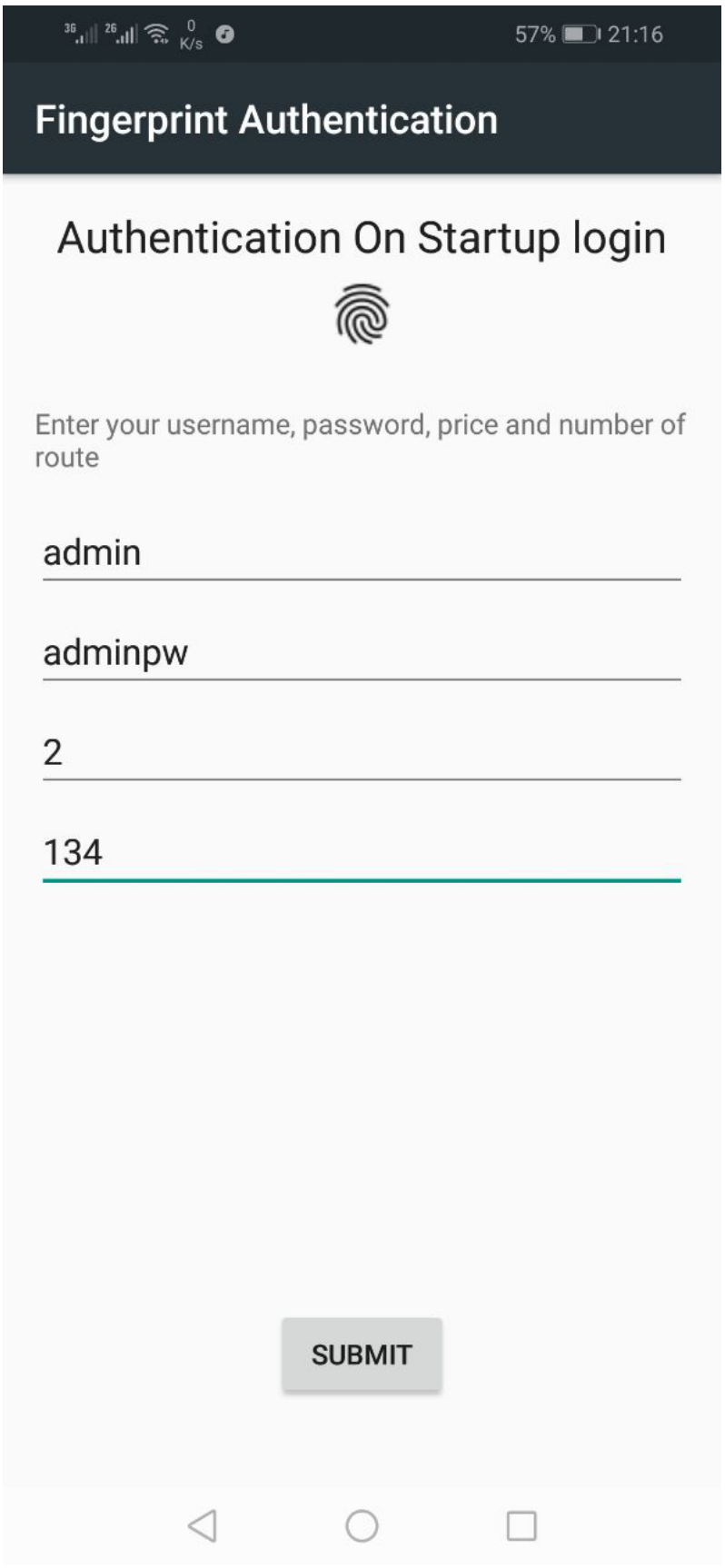

Рисунок 3.18 – Приклад введення даних після першого входу до клієнтського додатку

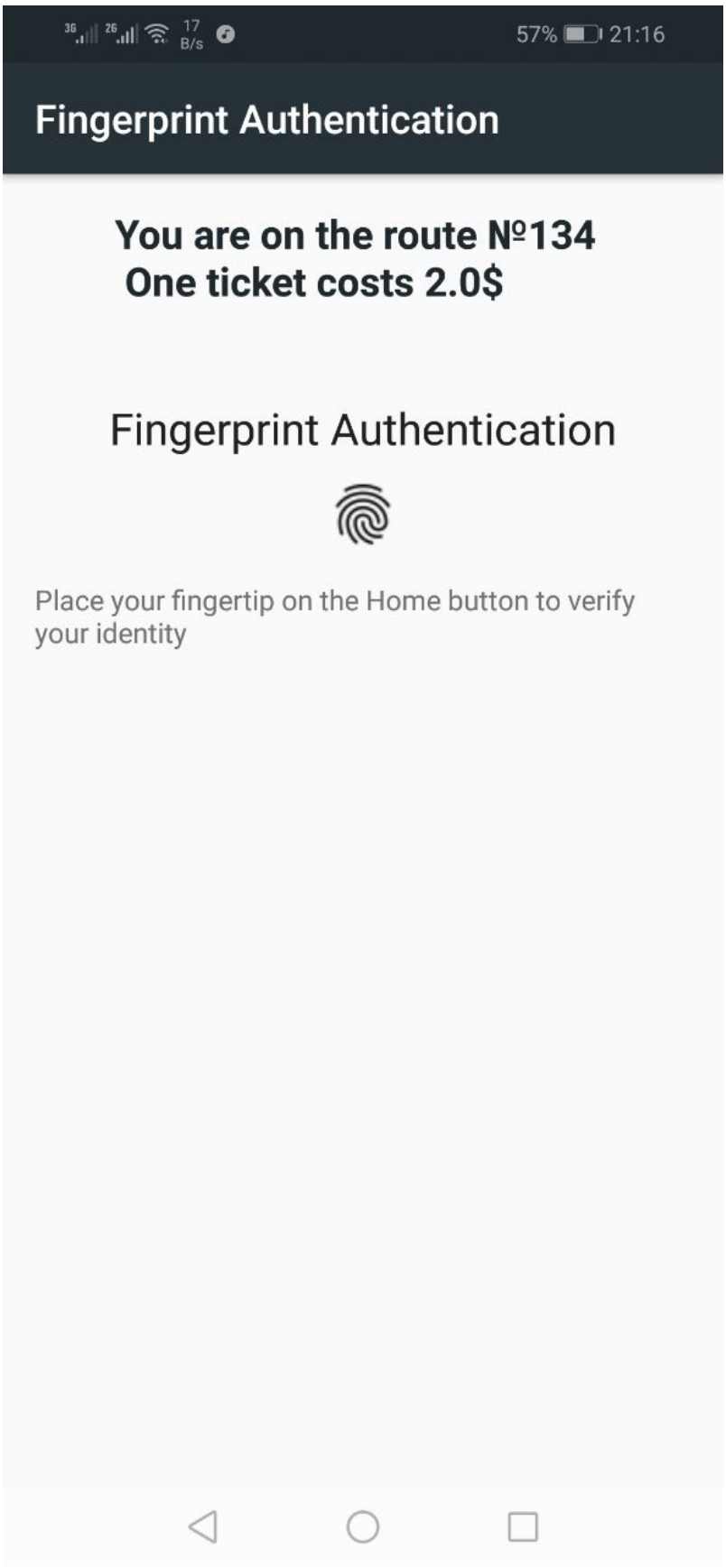

Рисунок 3.19 – Клієнтський додаток після вводу первинних даних перевізником

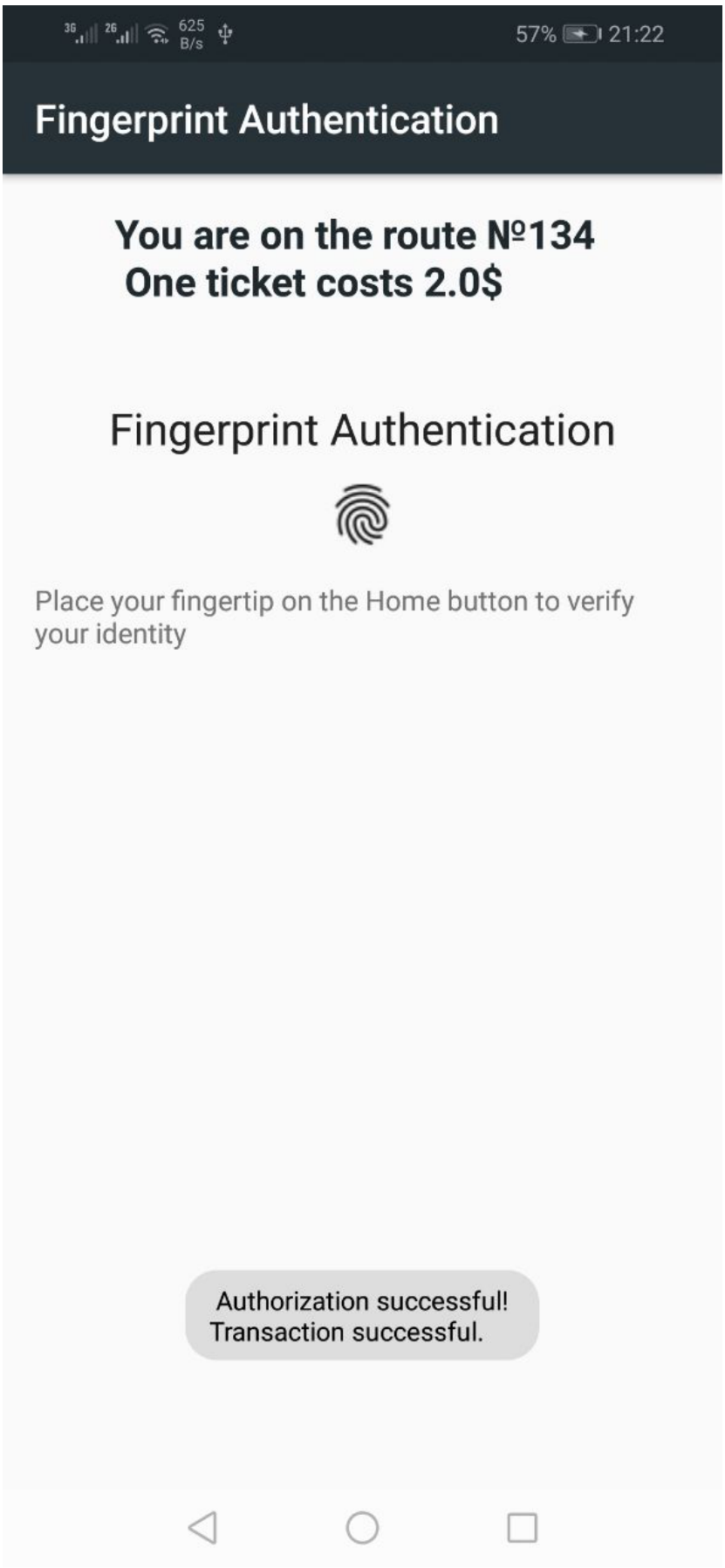

Рисунок 3.20 – Приклад повідомлення про успішну сплату проїзду

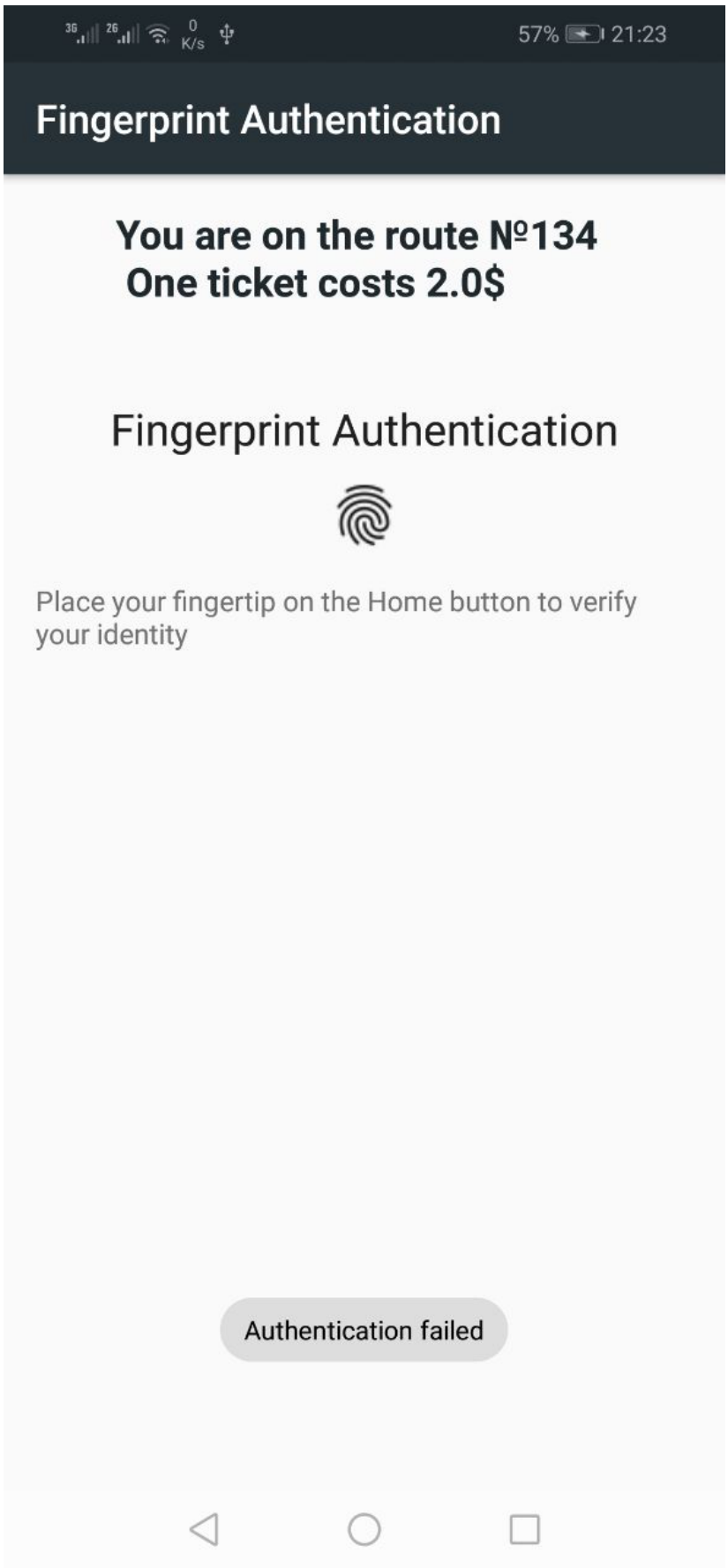

Рисунок 3.21 – Приклад повідомлення про невдалу сплату проїзду

Приклад отримання інформації про користувача до сплати та після сплати проїзду представлено на рисунках 3.22 та 3.23. Для тестування була обрана ціна 2, на стільки ж повинно змінитись поле «amount», яке наразі емулює стан рахунку користувача.

Рисунок 3.22 – Результат отримання інформації про користувача до сплати проїзду

Рисунок 3.23 – Результат отримання інформації про користувача після сплати проїзду

3.9 Тестування та оцінка надійності функціонування програмних рішень

Для тестування були використані JUnit та MockMvc. Тестами були покриті всі бізнес функції додатку та всі види запитів до серверу. Для тестування бізнес функцій був використаний JUnit, а для тестування запитів до серверу – MockMvc. Для тестування також була використана база даних H2, яка зберігає дані у пам’яті, що пришвидшує проходження тестів та не забиває сервер тестовими даними при тестування, наприклад, додаванні користувача у базу.

У ході тестування були виявлені деякі неточності с HTTP кодами відповідей сервера, які були виправлені.

Також була протестована автентифікація, яка потрібна щоб не дозволити зчитувати дані про буд-що у системі, не маючи на це дозволу. Як і задумувалось, якщо звернутися до сервера без даних, потрібних для автентифікації – сервер поверне HTTP код 401(неуповноважений), а якщо у користувача є дані для доступу до серверу, але їх недостатньо для проведення операції, що була запрошена – сервер поверне HTTP код 403(заборонено). Інформація про проходження деяких тестів автентифікації представлено на рисунку 3.24.

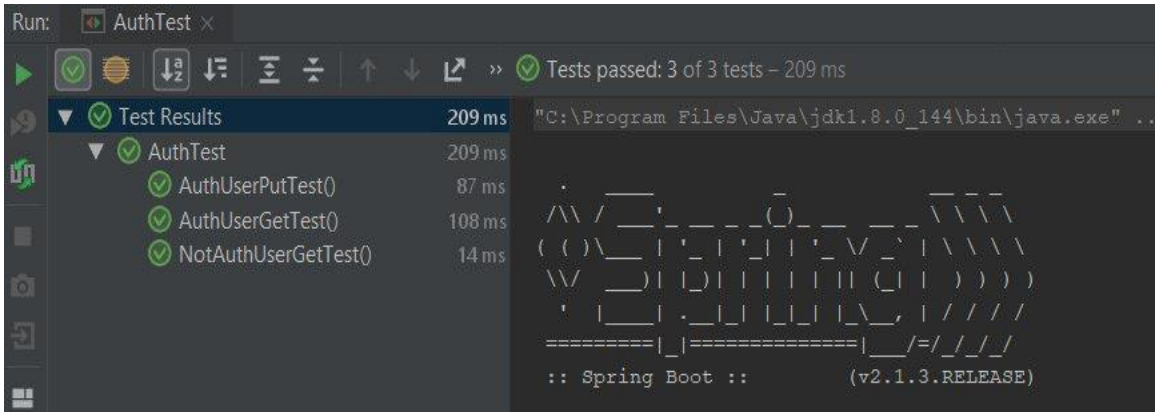

Рисунок 3.24 – Інформація про проходження тестів автентифікації

На рисунках 3.25 та 3.26 представлено MockMvc та JUnit тести відповідно. Представлений MockMvc тест перевіряє відповіді від сервера, в залежності від запитів та типу користувача. Представлений JUnit тест перевіряє бізнес-логіку, пов’язану з роботою з користувачами.

Рисунок 3.25 – Приклад MockMvc тестів

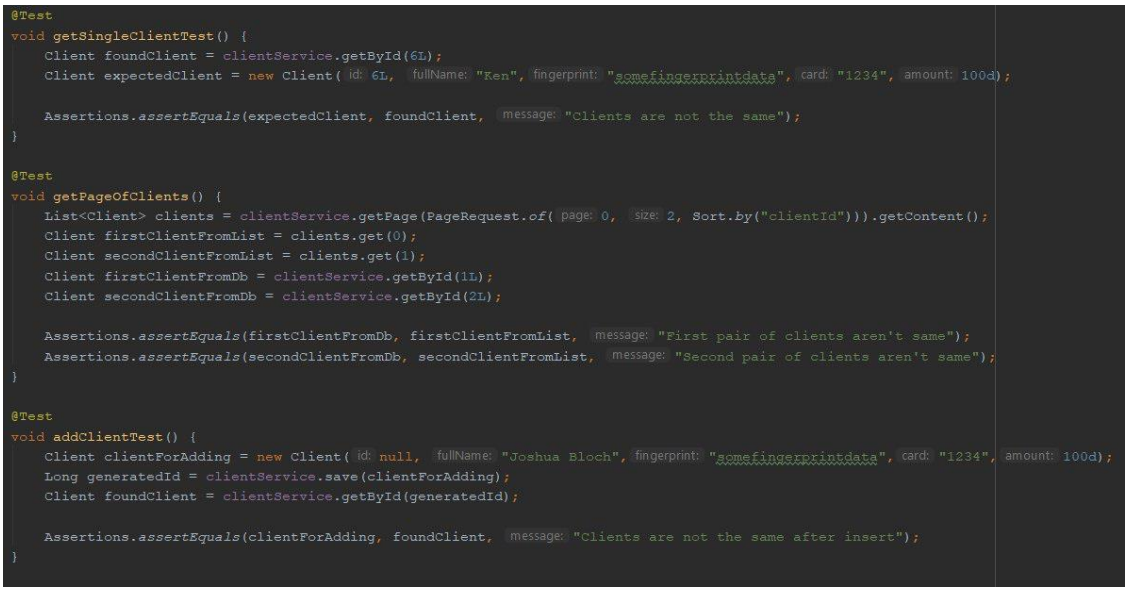

Рисунок 3.26 – Приклад JUnit тестів

3.10 Обґрунтування засобів захисту інформації від несанкціонованого доступу

Для доступу до веб-серверу використовується Spring Security. Spring Security це Java / JavaEE фреймворк, що представляє механізми побудови систем автентифікації та авторизації, а також інші можливості забезпечення безпеки для корпоративних додатків, створених за допомогою Spring Framework.

Основними частинами Spring Security є: ExceptionTranslationFilter, що використовується для забезпечення попереджень про виняткових ситуаціях; AuthenticationEntryPoint, що відповідає за перевірку введених користувачем даних та AuthentificationManager, що відповідає за підбір налаштувань для конкретного користувача.

До переваг Spring Security можна віднести: «легковажність», можливість зберігання зв’язків користувач-пароль-дані в будь-яких можливих форматах (бази даних, колекції, прості масиви, рядки і т.п.). До недоліків: реалізація фреймворка накладає обмеження на рішення складних завдань. Якщо порівнювати EJB і Spring Security то можна побачити наступну статистику: в великих корпоративних системах і банках, де є фінансові інвестиції в апаратне забезпечення використовують EJB, але більшість все ж переходить на Spring Security, який менш вимогливий до ресурсів системи. На сьогоднішній момент розроблено дуже багато засобів, які дозволяють реалізувати той же функціонал (Shiro, CORBA Auth), але жоден з цих засобів не має можливості роботи з великими корпоративними системами.

Для демонстрації роботи Spring Security буде використаний Swagger UI. На рисунку 3.27 представлено запит до серверу, що має повернути клієнта з id 2.

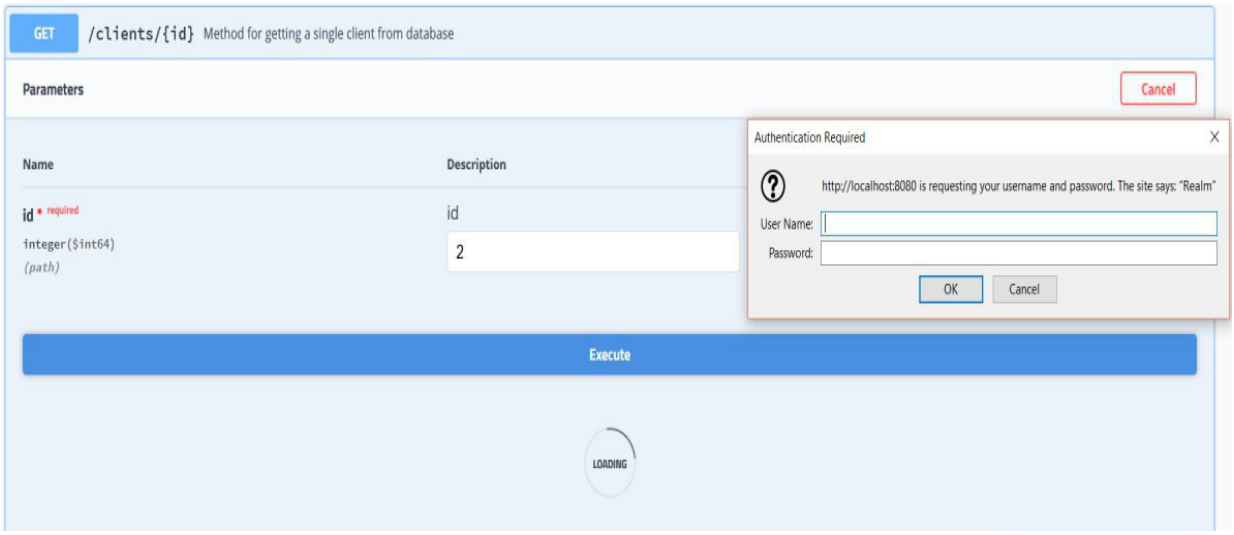

Рисунок 3.27 – Запит до сервера через Swagger UI

Як ми бачимо на рисунку 3.27 у браузері відкривається вікно для авторизації, без якої виконання такого запиту неможливе. Якщо ввести правильні дані у вікно, представлене на рисукнку 3.28 – запит виконається і сервер поверне результат, що представлений на рисунку 3.29.

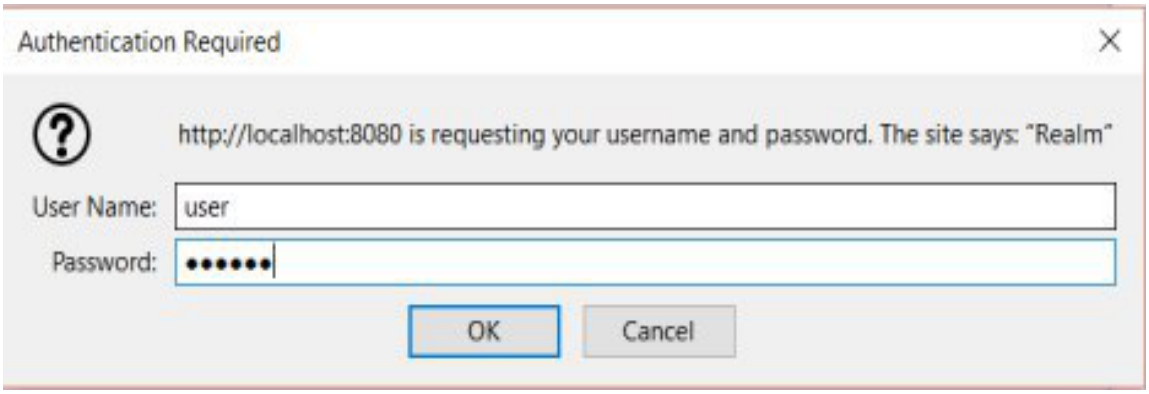

Рисунок 3.28 – Вікно вводу даних автентифікації

Рисунок 3.29 – Відповідь від сервера

Якщо ввести невірні дані то знов відкриється вікно для автентифікації, що надає змогу повторити ввід даних. Якщо відмінити ввід даних, у нашому випадку натиснути на кнопку «Cancel», то сервер відповість на такий запит HTTP кодом 401(неуповноважений). Приклад такої відповіді від сервера наведено на рисунку 3.30.

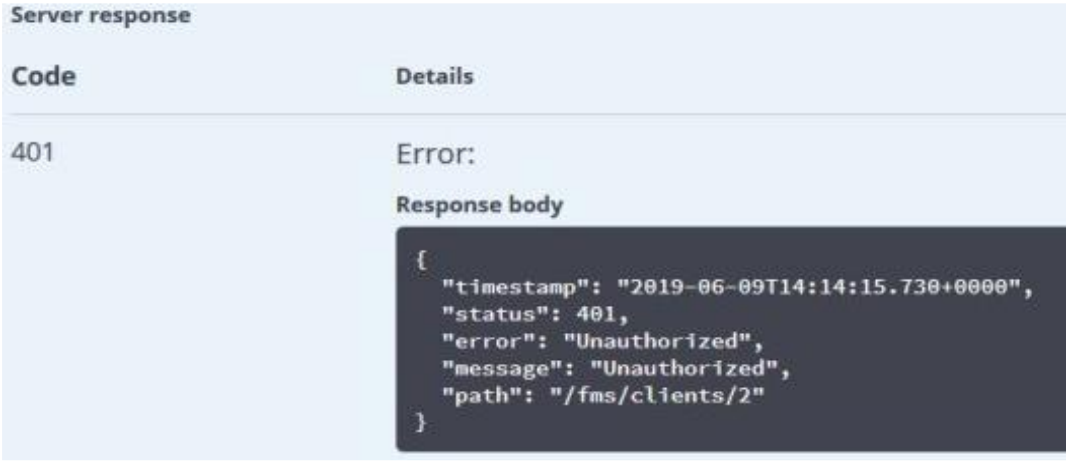

Рисунок 3.30 – Приклад відповіді від сервера при спробі запиту без даних для автентифікації

ВИСНОВКИ

Атестаційна робота була присвячена розробці Розробка підсистеми оплати проїзду в міському транспорті на основі біометричних даних. Актуальність розробки такого засобу зумовлена поєднанням найбільш затребуваних функціональних можливостей, що можуть надати додатки для сплати міського транспорту.

У ході виконання роботи проект не був обмежений тільки створенням мобільного додатка. Одним з найважливіших аспектів є вирішення ряду інженерних завдань, а також створення алгоритмічної моделі функціонування системи.

Була сформована чітка модель додатку. Також були визначені основні вимоги до додатку і здійснений докладний аналіз існуючих платформ та мов розробки, що у сукупності надають змогу створювати додатки необхідної складності з рядом потреб.

Були розглянуті і вирішені проблеми, пов'язані з: визначенням загальної структури програмного продукту, визначенням складу необхідних і додаткових ефективних компонентів системи, проектуванням і розробкою структури, алгоритмів взаємодії і бізнес-логіки компонентів системи, методами проектування сервісів і структури програмного забезпечення, проектуванням і реалізацією інформаційного забезпечення системи, вибору і узгодження інтерфейсів взаємодії сервісів і системи, а також визначення та створення алгоритмів діалогів користувачів з сервісами.

Для реалізації мобільного додатку було обрано мову програмування Java. Архітектура клієнтського додатку була побудована за шаблоном Model-View-Presenter. Для реалізації серверної частини додатку було обрано мову програмування Java, архітектура серверної частини додатку була побудована як REST сервіс й підтримує принцип інверсії контролю, зокрема за допомогою ін’єкції залежностей.

Були вирішені проблеми зі зберіганням та передачею даних користувачів. Для шифрування використовується шифр BCrypt, дані користувачів не передаються та не зберігаються у відкритому вигляді. Так як розроблений програмний засіб відповідає вимогам, що пред'явлені в постановці завдання, то можна зробити висновок, що атестаційна робота виконана в повному обсязі.

Також для збільшення якості зчитуємих відбитків було використано глибинне навчання, а саме був натренований згортковий автоенкодер.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wikipedia Minimum viable product. URL: https://en.wikipedia.org/wiki/Minimum_viable_product (дата звернення 8.11.2020)

2. Oracle.com SSL/TLS String Encryption: An Introduction. URL: https://docs.oracle.com/cd/B14099_19/web.1012/q20206/ssl/ssl_intro.html (дата звернення 12.11.2020)

3. Oracle.com Overview of Secure Sockets Layer (SSL) in Oracle Application Server. URL: https://docs.oracle.com/cd/B14099_19/core.1012/b13995/ssl_intro.htm (дата звернення 9.11.2020)

4. Skywell Какую платформу выбрать для мобильного приложения. URL: http://www.skywell.com.ua/blog/kakuyu-platformu-vybrat-dlya-mobilnogo-prilozheniya/ (дата звернення 9.11.2020)

5. Брюс Экель. Философия Java, 4-е издание. [Текст] // Брюс Экель — М.: «President, MindView, Inc», 2006. — 9с. Bruce Eckel, Thinking in Java, 4th edition – Introduction – 9c.

6. Adtmag Java 10 Released, First in the Faster Cadence. URL: https://adtmag.com/articles/2018/03/21/java-10.aspx (дата звернення 14.11.2020)

7. Wikipedia Software Architecture – Електронні данні. URL: https://en.wikipedia.org/wiki/Software_architecture (дата звернення 15.11.2020)

8. Wikipedia Separation of Concerns. URL: https://en.wikipedia.org/wiki/Separation_of_concerns (дата звернення 19.11.2020)

9. Oracle.com The Java Language Environment – Електронні данні. – URL: https://oracle.com/technetwork/java/intro-141325.html (дата звернення 11.11.2020)

10. Wikipedia Data binding – Електронні данні. URL: https://en.wikipedia.org/wiki/Data_binding (дата звернення 21.11.2020)

11. MockObjects.com All about Mock Objects, a technique from improving the design of code with Test-Driver Development. – URL: www.mockobjects.com (дата звернення 18.11.2020)

12. MartinFowler.com Inversion of Control Containers and Dependency Injection pattern. URL: https://www.martinfowler.com/articles/injection.html (дата звернення 12.11.2020)

13. Android official documentation Kotlin and Android. URL: https://developer.android.com/kotlin/ (дата звернення 15.11.2020)

14. OverFeat: Intergrated Recognition, Localization using Convolutional Networks, 2013.

15. Кнудсен E.I. Fundamental Components of Attention // Annual Review of Neuroscience, 2007.

16. Румельхарт Д.Е., Хінтон Г.Е., Вильямс Р.Д. Learning Internal Representations by Error Propagation / Cambridge, MA, USA: MIT Press, 1986.

17. Румельхарт Д.Е., Хінтон Г.Е., Вильямс Р.Д. Learning Representations by Back-propagating Errors / Cambridge, MA, USA: MIT Press, 1988.

18. Зейлер М.Д. Visualizing and Understanding Convolutional Networks. URL: http://arxiv.org/abs/1311.2901 (дата звернення 14.11.2020)