

Міністерство освіти і науки України  
Державний університет «Житомирська політехніка»  
Факультет інформаційно-комп'ютерних технологій  
Кафедра інженерії програмного забезпечення

**ПОЯСНЮВАЛЬНА ЗАПИСКА**  
до кваліфікаційної магістерської роботи  
на тему: «Система 2D дизайну нігтів»

Виконала студентка 2-го курсу, групи ІПЗм-19-1  
спеціальності 121 «Інженерія програмного  
забезпечення»

  
Т.О. Бойко  
Керівник старший викладач кафедри КН,

  
Г.В. Марчук  
Рецензент старший викладач кафедри КН,

  
В.Л. Левківський

Житомир – 2020

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ Інформаційно-комп'ютерних технологій  
(назва прописними)

КАФЕДРА Інженерія програмного забезпечення  
(назва прописними)

СПЕЦІАЛЬНІСТЬ Інженерія програмного забезпечення  
(назва прописними)

ЗАТВЕРДЖУЮ

Зав. кафедри інженерії

програмного забезпечення

(назва кафедри)

А.В. Морозов

« 07 » Вересня 2020 р.

**ЗАВДАННЯ**

на випускню кваліфікаційну роботу

Студента Бойко Таїси Олегівни

Тема роботи: Система 2D дизайну нігтів

Затверджена Наказом університету від «07» Вересня 2020 р. № 243

Термін здачі студентом закінченої роботи 01 грудня 2020р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження)  
використання інструментів розробки програмного забезпечення для  
створення додатку, що полегшуватиме роботу клієнтів манікюрних салонів  
Консультанти з випускної кваліфікаційної роботи із зазначенням розділів,  
що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Марчук Г.В.	02.09.2020	02.09.2020
2	Марчук Г.В.	15.10.2020	15.10.2020
3	Марчук Г.В.	10.11.2020	10.11.2020

Керівник



(підпис)

### Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	02.09.2020	виконано
2	Пошук, огляд та аналіз аналогічних розробок	10.09.2020	виконано
3	Формулювання технічного завдання	20.09.2020	виконано
4	Опрацювання літературних джерел	29.09.2020	виконано
5	Проектування структури	07.10.2020	виконано
6	Написання програмного коду, розробка бази даних	21.11.2020	виконано
7	Тестування	30.11.2020	виконано
8	Написання пояснювальної записки	15.12.2020	виконано
9	Захист	19.12.2020	

Студент

(підпис)

Керівник

(підпис)

## РЕФЕРАТ

Кваліфікаційна робота магістра складається з серверного додатку, клієнтського додатку для пристроїв на різних платформах та пояснювальної записки. Пояснювальна записка до випускної роботи містить 81 сторінку, 28 ілюстрацій та 6 таблиць.

Метою кваліфікаційної роботи є розробка програмної системи, що полегшить вибір манікюру для клієнтів манікюрних салонів та допоможе майстрам та адміністраторам салонів управляти наявними матеріалами.

В роботі проаналізовано галузь салонів краси, сформовано основні вимоги до системи та завдання на розробку програмного забезпечення. Описано варіанти використання системи, загальна структура програмного коду, основні компоненти системи та особливості розробки і розгортання системи. Окрім цього проведено тестування серверної та клієнтської частини додатку та проаналізовано результати тестування.

**КЛЮЧОВІ СЛОВА:** МАНІКЮРНИЙ САЛОН, КОЛІР, РОЗПІЗНАВАННЯ ЗОБРАЖЕННЯ, РОЗРОБКА МУЛЬТИПЛАТФОРМЕНИХ ДОДАТКІВ.

## ABSTRACT

The master's qualification work consists of a server application, a multiplatform client application and a thesis. The final version of thesis contains 81 pages, 28 pictures and 6 tables.

The goal of the qualification work is development of a software solution that facilitates the choice of manicure for manicure salon customers. This solution should provide assistance in managing salon's products to manicure artists and other salon's staff.

The analysis of the beauty industry was conducted as a part of a thesis. Basing on the result on the analysis was created basic requirements for systems and designed software development tasks. The following activities were made:

- defining use cases of the solution;
- designing the general code architectural structure;
- defining main software components;
- software development and system deployment;
- client and server app functional testing, user interface and experience testing, bug fix.

**KEY WORDS:** MANICURE SALON, COLOR, IMAGE RECOGNITION, DEVELOPMENT OF MULTIPLATFORM APPLICATIONS.

## ЗМІСТ

<b>РЕФЕРАТ.....</b>	<b>4</b>
<b>ABSTRACT .....</b>	<b>5</b>
<b>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ .....</b>	<b>8</b>
<b>ВСТУП.....</b>	<b>9</b>
<b>РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 2D ДИЗАЙНУ НІГТІВ.....</b>	<b>11</b>
1.1. ПОСТАНОВКА ЗАДАЧІ .....	11
1.2. АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ.....	15
1.3. ВИБІР АРХІТЕКТУРИ СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ .....	18
1.4. ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	21
Висновки до першого розділу .....	26
<b>РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ.....</b>	<b>27</b>
2.1. ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ ТА ОБ’ЄКТНО- ОРІЄНТОВАНОЇ СТРУКТУРИ СИСТЕМИ.....	27
2.2. РОЗРОБКА БАЗИ ДАНИХ .....	32
2.3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ.....	35
2.4. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ.....	39
Висновки до другого розділу .....	46
<b>РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ .....</b>	<b>47</b>
3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ. ....	47
3.2. СТРУКТУРА ІНТЕРФЕЙСУ КЛІЄНТСЬКОГО ДОДАТКУ.....	50
3.3. ТЕСТУВАННЯ РОБОТИ СИСТЕМИ.....	62
Висновки до третього розділу .....	64
<b>ВИСНОВКИ .....</b>	<b>66</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>	<b>68</b>

<b>ДОДАТКИ.....</b>	<b>72</b>
<b>ДОДАТОК А.....</b>	<b>73</b>
<b>ДОДАТОК Б.....</b>	<b>74</b>

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CORS – Cross-Origin Resource Sharing

CSS – Cascading Style Sheets

CV – Computer Vision

CRUD – Create Read Update Delete

FK – Foreign Key

GCP – Google Cloud Platform

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

PK – Primary Key

REST – Representational State Transfer

RGB – Red, Green, Blue

SQL – Structured Query Language

WSGI – Web Server Gateway Interface

UI – User Interface

UX – User Experience

ЄДР - Єдиний Державний Реєстр

ОС – Операційна система

ПЗ – Програмне Забезпечення

ПП – Приватне підприємство

ТОВ – Фізична особа-підприємець

ФОП – Фізична особа-підприємець



## ВСТУП

В умовах постійного розвитку інформаційних технологій в сучасному світі, кожна галузь людської діяльності прагне досягнути максимальної автоматизації процесів та мінімізувати ймовірність виникнення помилок працівників під час виконання роботи. В сучасному суспільстві індустрія краси (beauty-індустрія) займає високу позицію серед потреб населення. В результаті такої популярності галузь краси стала стрімко розвиватись, а багато видів послуг перетворились в окремі самостійні напрямки. Наприклад перукарське мистецтво, манікюр, косметологія та інші типи послуг перетворюються в окремі напрямки діяльності, відкриваються спеціалізовані салони краси та школи вивчення певного напрямку. Кожен напрямок галузі має власну специфіку роботи, а програмного забезпечення, що забезпечувало б повну автоматизацію кожного напрямку, немає. Саме тому є необхідність в такому ПЗ.

*Актуальність теми* полягає в тому, що популярна галузь господарської діяльності не має програмного забезпечення для автоматизації багатьох робочих процесів. Створення програмної системи за проектом магістерської роботи, вирішить проблему обліку матеріалів манікюрного салону для працівників та пошуку потрібного матеріалу для клієнта.

*Метою випускної роботи* є розробка програмної системи, що полегшить вибір манікюру для клієнтів манікюрних салонів та допоможе майстрам та адміністраторам салонів управляти наявними матеріалами.

Встановлена мета обумовлює наступні завдання:

- вивчення особливості обслуговування клієнта манікюрного салону;
- проведення аналізу процесу роботи майстра манікюру;
- вивчення можливих матеріалів;
- визначення архітектури та узагальненої структури системи;
- обґрунтування та вибір засобів реалізації системи;
- проектування системи з врахуванням можливості її використання як клієнтами салону краси, так і майстрами;

– реалізація системи.

*Об'єктом дослідження* є технології роботи манікюрного салону та процес надання послуг.

*Предметом дослідження* є процес використання технологій розробки програмного забезпечення для автоматизації обраних процесів роботи манікюрного салону.

Розроблена система може бути використана для впровадження у діяльність манікюрних салонів, шкіл вивчення манікюрної справи, а також у комерційних цілях малого бізнесу типу ФОП, ТОВ чи ПП.

Система була впроваджена для роботи ФОП Романюк Лідія Костянтинівна, основним напрямком економічної діяльності якої є надання послуг перукарнями та салонами краси. Акт впровадження програмного забезпечення на виробництво наведено у Додатку А.

*Публікації.* За результатами проведеного дослідження галузі салонів краси було взято участь у Міжнародній науковій інтернет-конференції "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення" (випуск 54). На конференції було опубліковано тези на тему «Аналіз способів застосування інформаційних технологій в галузі салонів краси».

Архів публікацій конференції знаходиться за посиланням <http://www.konferenciaonline.org.ua/arhiv-konferenciy/arhiv-konferenciy10-12-2020>.

## РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 2D ДИЗАЙНУ НІГТІВ

### 1.1. Постановка задачі

Основною метою створення системи 2D дизайну нігтів є полегшення процесу роботи салону краси, або манікюрного салону, що спеціалізується у напрямку манікюрних робіт. Існує загально прийнятий процес надання та отримання манікюрних послуг, який не передбачає великої кількості комп'ютеризації та автоматизації певних дій. Під час відвідування манікюрного салону, клієнт повинен обрати колір майбутнього манікюру серед запропонованого асортименту. Часто, перелік доступних у салоні кольорів є досить великим. Як правило, асортимент розміщується на пластикових пластинах, які клієнт перебирає з метою пошуку потрібного кольору та відтінку. Ці пластини не завжди знаходяться в визначеному порядку, тому пошук бажаного відтінку може займати певний час, адже клієнтові важко орієнтуватись в великому асортименті. Така ситуація створює ряд проблем, які можна вирішити за допомогою впровадження необхідного програмного забезпечення. Розглянемо існуючі проблеми:

1. Клієнт не має доступу до структурованого переліку доступних пропозицій та їх цінкових категорій;
2. Клієнт витрачає велику кількість обмеженого часу на пошук бажаного відтінку. Окрім того, клієнт не може зробити свій вибір та порівняти можливі варіанти ще до візиту до салону;
3. Використання пластикових пластин створює середовище розповсюдження бактерій, що є особливо гострою проблемою в сучасних реаліях епідеміологічного становища;
4. Використання пластикових пластин не є екологічним підходом;
5. Майстер, або адміністратор манікюрного салону не має структурованої бази матеріалів.

Розглянувши та проаналізувавши перелічені проблеми в процесах обслуговування клієнта манікюрного салону, можна зробити висновок, що існує потреба в проектуванні та розробці програмного рішення, що допоможе оптимізувати роботу салону.

Існують такі вимоги до проектування та розробки системи 2D дизайну нігтів:

- система повинна бути доступна для клієнта, адміністратора та майстра манікюрного салону у вигляді клієнтського додатку;
- додаток повинен бути доступний для встановлення на мобільні пристрої;
- додаток повинен бути доступний для пристроїв, що мають встановлений будь-який сучасний веб-браузер та доступ до мережі Інтернет;
- додаток повинен мати сучасний дизайн, який підлаштовується під різні пристрої та адаптується під розмір екрану;
- додаток повинен мати зрозумілий користувацький інтерфейс та відповідати вимогам сучасного UI та UX дизайну.

Розробку системи 2D дизайну нігтів можна розділити на окремі завдання, виконання яких задовольнить перераховані вище вимоги:

1. Розробка сервісу обробки зображення. Сервіс, що працює в якості REST API та обробляє зображення користувача. Сервіс приймає запит на отримання основних кольорів зображення. Сервіс оброблює зображення за допомогою інструментів реалізації комп'ютерного зору та надає відповідь у вигляді переліку основних кольорів зображення.
2. Розробка бази даних. База даних, що містить дані про колекції доступних матеріалів, опис лаків та інформацію про залишок лаків у салоні.
3. Розробка сервісу пошуку матеріалів за кольором. Сервіс, що буде порівнювати матеріали з бази даних з заданим кольором та здійснювати пошук найбільш підходящих матеріалів.

4. Розробка додатку для адміністратора. Сервіс роботи адміністратора має бути доступний як мобільний додаток та веб додаток. Сервіс є розширенням звичайного клієнтського додатку, окрім того, цей сервіс має містити наступні додаткові функції роботи адміністратора:

- Управління колекціями матеріалів: додавання нової колекції/категорії/бренду
- Управління матеріалами: додавання нового матеріалу, завантаження фото матеріалу, видалення, редагування існуючих матеріалів
- Редагування доступу до адміністративного додатку. Редагування списку адміністраторів для додавання користувача з правом доступу до адміністраторського додатків.

5. Розробка додатку для клієнта. Додаток має містити такі ключові компоненти інтерфейсу:

- Головна сторінка. Сторінка на якій є такі елементи:
  - кнопка завантаження фотографії з галереї/файлової системи пристрою, текст кнопки має бути індикатором того, завантажене зображення, чи ні;
  - стандартне зображення при завантаженні екрану, яке змінюється на обране зображення користувача після його завантаження з галереї/файлової системи пристрою;
  - інструмент для вибору кольору на секції зображення;
  - список підібраних кольорів від сервісу обробки зображення; кожен колір є навігаційною кнопкою до переліку підходящих кольорів у салоні.
- Сторінка всіх матеріалів. Ця сторінка має відображати список матеріалів (лаків) з їх повною інформацією:
  - колір;
  - виробник/бренд;
  - код кольору за брендом;

- ціна;
- Сторінка обраних матеріалів. Матеріали, можуть бути показані за певною характеристикою:
  - результат пошуку підходящих за кольором матеріалів;
  - нові матеріали;
  - результат фільтрації за певною колекцією/брендом.
- Сторінка одного матеріалу. Сторінка з повною інформацією про цей матеріал та кнопка «Додати до обраних»;
- Приховане меню, що містить елементи навігації основними розділами додатку;
- Навігаційне меню в верхній частині екрану, що містить такі елементи:
  - кнопка відкриття прихованого меню (за умови, що користувач знаходиться на головній сторінці чи на сторінці матеріалів);
  - кнопка повернення до попередньої сторінки (якщо користувач знаходиться поза головною сторінкою чи сторінкою матеріалів);
  - кнопка відкриття списку обраних матеріалів (за умови, що користувач авторизований в системі);
- Список обраних матеріалів. Сторінка, що доступна для авторизованих користувачів. Сторінка містить перелік лаків, які користувач відмітив як обрані. Кожен елемент в переліку може бути видалений, а також можна переглянути детальну інформацію про обраний лак, або запустити модуль доповненої реальності.
- Сторінка авторизації користувача. Авторизація за допомогою системи Google. За умови, що користувач має права адміністратора, додаток буде відкрито в режимі адміністратора після авторизації.

## 1.2. Аналіз аналогів програмного продукту

Після ретельного ознайомлення з галуззю салонів краси та манікюрних салонів, можна переходити до дослідження існуючих програмних продуктів для вирішення визначеного завдання. Провівши пошук існуючих програмних продуктів для галузі, можна розділити всі рішення на чотири категорії:

1. Системи онлайн запису та управління фінансами салону краси;
2. Ігри-симулятори салону краси, які дозволяють створити певний дизайн під час ігрового процесу;
3. Симулятори зовнішнього вигляду нігтів. Це системи доповненої реальності з можливістю перегляду можливого результату манікюру в віртуальній формі;
4. Онлайн-магазини продажу матеріалів для манікюрних салонів.

Дослідження ринку існуючих програмних продуктів, показало, що наразі немає такого додатку, який би повністю вирішував поставлену задачу. Розглянемо декілька програмний продуктів, які частково вирішують поставлену задачу.

Додаток «Wanna Nails» - це мобільний додаток доступний для завантажування на мобільні пристрої з операційною системою Apple IOS з офіційного магазину додатків – App Store. Приклад користувацького інтерфейсу наведено на рисунку 1.1.

Цей додаток дозволяє спробувати різні кольори лаків з запропонованих колекцій за допомогою технології доповненої реальності. Перегляд доступних лаків можна здійснити в меню в нижній частині екрану. В цьому меню знаходиться перелік колекцій лаків та кольорів, доступних в обраній колекції.

Також, користувачеві пропонується посилання на інтернет-магазин для придбання обраного лаку і одразу ж надається ціна лаку. В налаштуваннях додатку можна визначити інтернет магазин, посилання на який, будуть відображатись в меню. Доступно два варіанти: Amazon та Aliexpress.

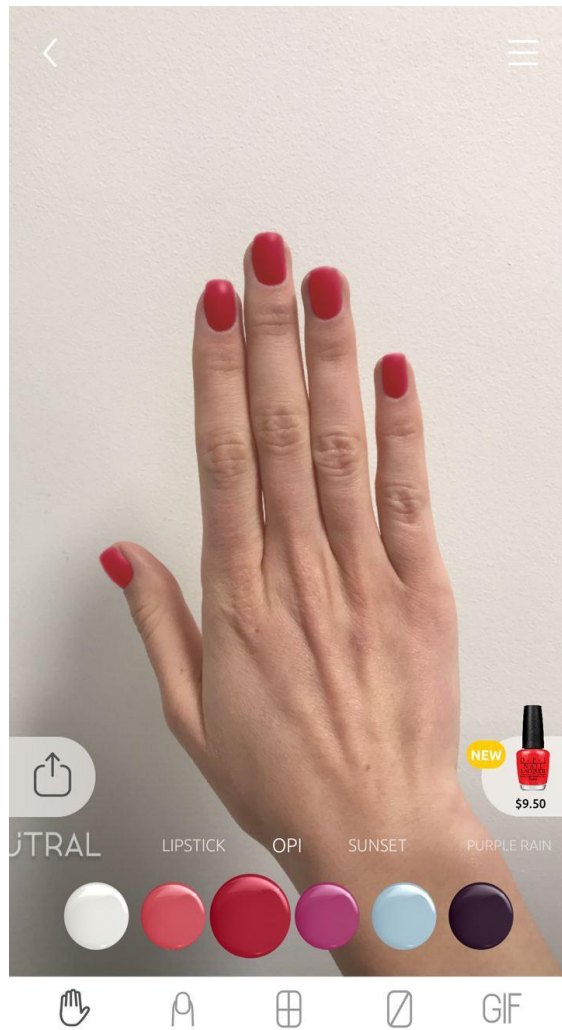


Рисунок 1.1 – Інтерфейс додатку «Wanna Nails»

Механізм роботи доповненої реальності працює чітко, контури пальців та нігтів зазвичай визначаються правильно та досить реалістично. Серед додаткових функцій додатку є наступні:

- порівняння різних кольорів лаку на руці;
- створення колажу фотографій з різними варіантами лаків;
- визначення різних кольорів для окремих нігтів.

Хоча додаток має галерею доступних матеріалів, користувачеві не надається можливість переглянути галерею всіх кольорів. Також, відсутня функція пошуку кольорів. Сама ж галерея створена на основі обраних пропозицій, а не на основі існуючих матеріалів у конкретному манікюрному



салоні. Окрім того, додаток недоступний для користувачів пристроїв на базі операційної системи Android.

В якості іншого аналогу програмного продукту розглянемо додаток GLOSS Company (рис.1.2). Цей додаток доступний як веб та мобільний додаток для пристроїв на базі операційних систем Apple IOS та Android. Мобільні додатки можна встановити з офіційних магазинів додатків App Store та Google Play. Проте у мобільному додатку є оголошення про припинення роботи мобільної версії.

Доступ до веб-додатку можна отримати за посиланням <https://gloss.company/uk/>. Основною функцією додатку є інтернет-магазин товарів для манікюрних салонів виробника Gloss. На рисунку 1.2 зображено зовнішній вигляд головної сторінки веб-додатку.

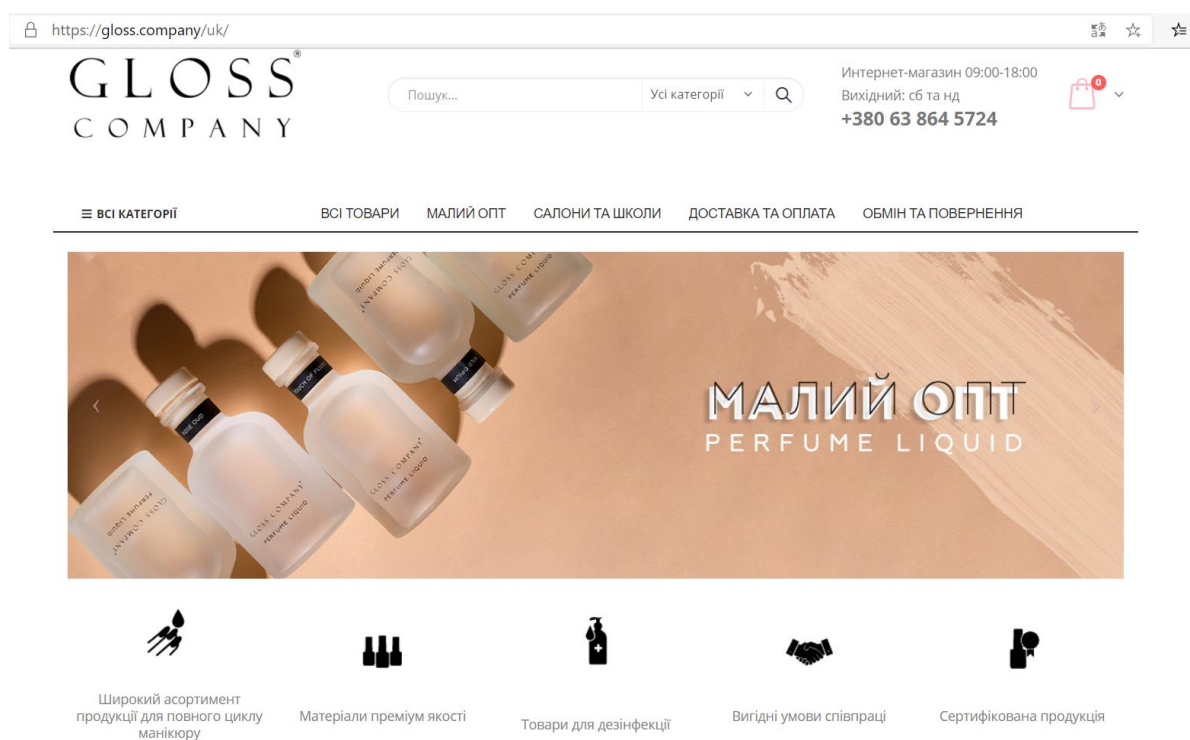


Рисунок 1.2 – Головна сторінка додатку «GLOSS

Додаток має велику галерею товарів з багатьма категоріями. GLOSS Company має зручний та зрозумілий користувацький інтерфейс та виглядає як класичний сучасний інтернет-магазин товарів: є функція зміни мови (українська, англійська, російська), кошик товарів, список обраних товарів, перелік

категорій, фільтр відображення товарів, інформація про доставку тощо. На рисунку 1.3 зображено зовнішній вигляд сторінки перегляду продукції за категорією.

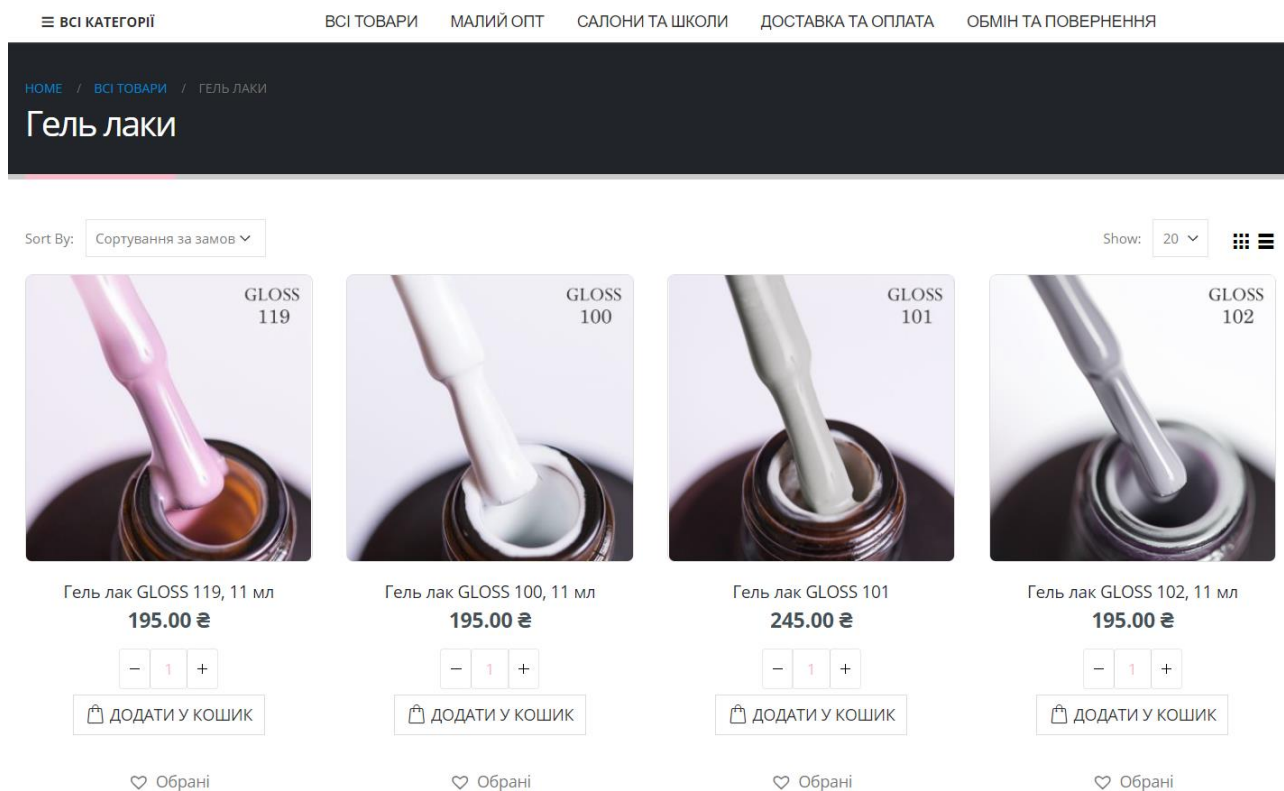


Рисунок 1.3 –Сторінка категорії товарів додатку «GLOSS Company»

Оскільки цей додаток є інтернет магазином товарів певного виробника, неможливо знайти продукції від інших виробників. Тому такий додаток не можна застосувати для роботи в манікюрному салоні, можливо лише обрати певний відтінок за бажанням. Тому цей додаток повністю не вирішує поставлене завдання.

### 1.3. Вибір архітектури системи 2D дизайну нігтів

Система 2D дизайну нігтів передбачає одночасну участь декількох користувачів і потребує централізоване зберігання даних та їх поширення між різними користувачами. Саме тому, оптимальним рішенням для розробки становить клієнт-серверна архітектура.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних додатків [39]. Така архітектура передбачає взаємодію та обмін даними між клієнтським та серверним додатком. Вона передбачає такі основні компоненти:

- сервер чи набір серверів, які зберігають та надають інформацію, або виконують іншу дії для програм які звертаються до них (клієнтів);
- клієнт чи набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

На рисунку 1.4 зображена схема архітектури клієнт-сервер, та її основні компоненти.

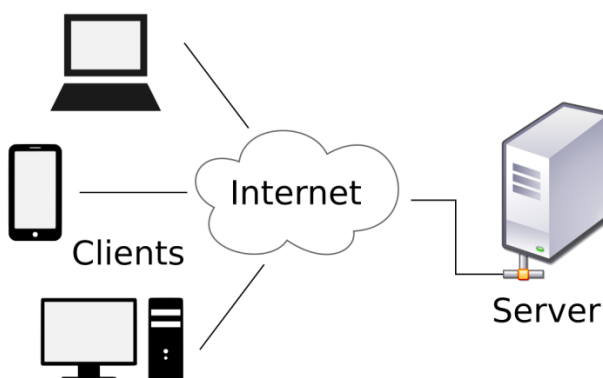


Рисунок 1.4 - Схема роботи клієнт-серверної архітектури

Розглянемо детальніше ключові елементи даної системи.

Сервер представляє собою веб-сервер, що є елементом центральної обробки даних та забезпечує комунікацію між клієнтами. Також, до сервера відноситься база даних – централізоване сховище інформації, якою оперує сервер за запитом клієнтів.

У системі передбачено один клієнтський додаток для користувачів різних пристроїв. Він є елементом системи, що надає клієнту манікюрного салону доступ до галереї матеріалів, їх пошуку та вибору. Для адміністратора салону краси, цей додаток надає можливість змінювати дані в центральній базі даних. Клієнтський додаток надає можливість користувачам входити в систему,

переглядати матеріали до використання та змінювати їх за умови наявності прав доступу до таких дій.

В якості мережі використовується мережа інтернет, що забезпечує комунікацію між усіма архітектурними елементами.

Сам клієнтський додаток є складною системою, оскільки комбінує в собі додаток для мобільних пристроїв та для веб-браузерів.

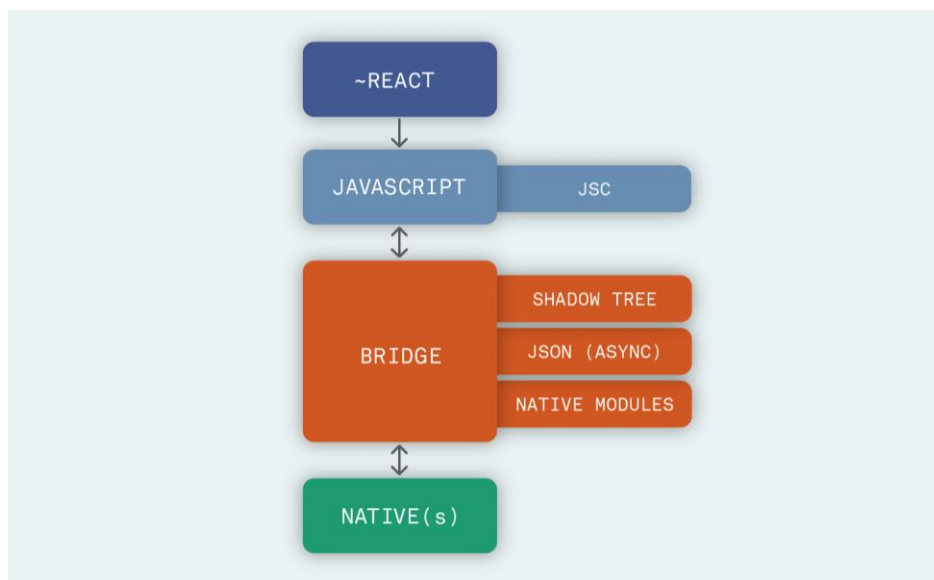


Рисунок 1.5 - Архітектура системи React Native

Підхід створення універсального додатку розроблений за допомогою використання інструменту React Native. Розглянемо загальну архітектуру системи React Native, що зображена на рисунку 1.5.

В системі є чотири основні частини:

- код на бібліотеці React – кодова база, створена розробниками для конкретних функцій готового додатку;
- код JavaScript - інтерпретований код React;
- система «міст» – зв'язок між JavaScript кодом і API середовища виконання додатку на пристрою;
- «Native» середовище виконання пристрою (OC Android, IOS або веб-браузер).

Ключовим аспектом архітектури є те, що компоненти JavaScript та Native, не мають жодної інформації один про одного. Це означає, що для спілкування вони покладаються на асинхронні повідомлення JSON, передані через міст [22].

При роботі з інструментом React Native, його архітектура залишається незмінною, незалежно від впливу розробників додатку. Проте, архітектура самого додатку може бути змінена в залежності від підходу до його розробки.

Хоча в програмному коді додатку є можливість написання класичного коду для систем Android та IOS, основний код для розробки функціональних можливостей описується у вигляді JavaScript коду на базі фреймворку React.

#### 1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Отримавши перелік вимог та завдань до проекту, можна побачити, що система має досить складну будову: необхідно розробити систему розпізнавання кольорів зображення та API для доступу до цієї системи, мобільний додаток з можливістю роботи користувачів з різними ролями та веб додаток з такими ж можливостями, а також систему доповненої реальності для моделювання нігтів.

Розглянемо вимоги до апаратного забезпечення для різних компонентів системи:

- Додаток для клієнта та адміністратора манікюрного салону має працювати на таких пристроях:
  - Мобільні пристрої на базі операційної системи Android 8.0 Oreo та вище [1]. Мінімальний об'єм оперативної пам'яті пристрою 1Гб, від 2Гб внутрішньої пам'яті пристрою чи карти пам'яті, доступ до системи Інтернет;

- Мобільні пристрої на базі операційної системи Apple IOS 12 та вище. Мінімальний об'єм оперативної пам'яті пристрою 1Гб, від 2Гб внутрішньої пам'яті пристрою, доступ до системи Інтернет;
- Комп'ютери, ноутбуки, планшети на базі процесорів Intel, AMD у стандартній комплектації та на базі будь-якої операційної системи що підтримує роботу сучасних веб-браузерів: Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge Chromium і подібні.
- Сервіс виконання розпізнавання зображення та серверна частина додатку в загальному має наступні вимоги до апаратного забезпечення:
  - встановлена операційна система Linux;
  - підтримка мови Python версії 3.7 та вище;
  - стабільний швидкісний доступ до Інтернету;
  - оперативна пам'ять – не менше 8Гбайт;
  - об'єм дискової пам'яті – не менше 128Гбайт.

Оскільки клієнтський додаток має бути доступним для різних ОС та середовищ, необхідно обрати технологію, що допоможе створювати універсальне рішення та позбавить від необхідності розробляти три додатки з однаковим функціоналом під різні операційні системи. Розглянемо можливі варіанти серед існуючих інструментів та проведемо вибір інструментів для розробки системи.

Сьогодні існує велика кількість різноманітних технологій для розробки мобільних та веб додатків. Це технології можна поділити на дві великі категорії [40]:

- Нативні додатки (native apps) – додатки, що розроблюються конкретно під потрібну платформу (ОС Android, Apple IOS, ОС Windows і т. п.). Для розробки нативних додатків можна використовувати такі технології:
  - Мови програмування Java та Kotlin для розробки під операційну систему Android;

- Мови програмування Objective-C та Swift для розробки додатків під Apple IOS;
  - Технологія Flutter. Інструмент для розробки нативних додатків під мобільні пристрої Android та IOS. Розробка на цьому інструменті відбувається на мові програмування Dart;
  - Технологія React Native. Інструмент для розробки нативних додатків під мобільні пристрої, а також веб додатків для різних браузерів. Розробка на React Native відбувається за допомогою мови програмування JavaScript та бібліотеки ReactJS.
- Гібридні додатки – додатки, що розроблюються за допомогою інструментів веб програмування HTML, CSS, JavaScript та додаткових бібліотек мови JavaScript. Для розробки гібридних додатків є ряд технологій:
- Electron;
  - Ionic;
  - PhoneGap.

Гібридні інструменти як Electron, Ionic та PhoneGap працюють за принципом пакування веб-додатку у вигляді звичайного додатку під певну операційну систему. Перевагою цих технологій є простота розробки, оскільки програмний код не сильно відрізняється від того, що використовується для створення звичайної веб сторінки, а написаний веб додаток відображується як сторінка браузера – WebView в контейнері для операційної системи. Проте така простота впливає на показники швидкості та якості роботи готового продукту в контексті потрібної операційної системи. Окрім того, серед зазначених технологій є такі, що втрачають популярність в спільноті розробників, а автори інструменту PhoneGap, компанія Adobe, взагалі припинили підтримку свого проекту [26].

На відміну від гібридних додатків, нативні додатки мають кращі показники швидкості роботи, саме тому обрано розробка цієї категорії додатків.

Серед існуючих варіантів технологій для розробки додатку найбільш оптимальним є React Native. Кодова база універсальна для мобільної версії Android, IOS та веб додатку, тому швидкість розробки такої системи буде більш високою, ніж розробка трьох окремих додатків з власною кодовою базою. Мова програмування JavaScript та бібліотека ReactJS є більш популярними, ніж мова Dart, саме тому React Native є більш оптимальним, ніж Flutter.

React Native - це технологія розробки мобільних та веб додатків з відкритим кодом, створений компанією Facebook, Inc [17]. Вона використовується для розробки програм для Android, Android TV, iOS, macOS, tvOS, веб сайтів та Windows, дозволяючи розробникам використовувати фреймворк React разом із можливостями власної платформи. React Native не використовує HTML або CSS. Натомість повідомлення з потоку роботи JavaScript використовуються для управління нативними компонентами. React Native також дозволяє писати власний код такими мовами, як Java, Objective-C або Swift, що робить інструмент ще більш гнучким.

Окрім додатку клієнта, потрібно розробити сервіс обробки зображення. Для цього обрана мова програмування Python, оскільки вона має велику кількість розроблених бібліотек для імплементації комп'ютерного зору.

Мова програмування Python - це інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна,



функціональна та аспектно-орієнтована [15]. Для серверного додатку використовується Python версії 3.

Основні бібліотеки для роботи алгоритму обробки зображення:

- Numpy – бібліотека для роботи з масивами;
- Scikit learn – бібліотека з реалізацією алгоритму k-means;
- OpenCV – бібліотека для розпізнавання зображень.

Алгоритм обробки зображення має знаходитись на Python-сервері, тому потрібно влаштувати комунікацію між клієнтським додатком та серверним. Для створення серверу використовується модуль Gunicorn.

Gunicorn (скор. англ. Green Unicorn «Зелений єдиноріг») - це HTTP-сервер інтерфейсу шлюзу веб-сервера Python. Сервер Gunicorn в сумісний з низкою веб-фреймворків, просто реалізований, легкий на ресурсах сервера і досить швидко [12].

Сервіс обробки зображення та сервісу пошуку підходящого матеріалу за кольором знаходяться на сервері gunicorn, для отримання та відправки даних з серверу необхідно влаштувати програмний інтерфейс REST.

Для реалізації програмного інтерфейсу за підходом REST використовується бібліотека Flask. Flask - це простий та легкий фреймворк веб-додатків інтерфейс шлюзу веб-сервера WSGI (Web Server Gateway Interface) [10]. Він призначений для швидкого та легкого початку роботи з можливістю масштабування до складних додатків. Flask не додає жодних залежностей до проекту. Для бібліотеки є багато розширень, які полегшують додавання нових функціональних можливостей.

В якості бази даних для проекту обрано реляційну базу даних SQLite. SQLite - це бібліотека на мові C, яка реалізує невеликий, швидкий, автономний та високонадійний механізм баз даних SQL. SQLite - це полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту SQL-92. Основний код SQLite поширюється як суспільне надбання, тобто може використовуватися без обмежень та безоплатно з будь-

якою метою. Фінансову підтримку розробників SQLite здійснює спеціально створений консорціум, до якого входять такі компанії, як Adobe, Oracle, Mozilla, Nokia, Bentley і Bloomberg [18].

#### Висновки до першого розділу

Дослідження та аналіз предметної області дозволив сформулювати уявлення про особливості діяльності манікюрних салонів та процеси, що потребують автоматизації. Після формування основної ідеї використання програмного забезпечення, було визначено та сформовано основні вимоги до програмного продукту. Аналіз аналогів програмного продукту показав, що наразі немає готових рішень, що повністю задовольняли б визначені вимоги.

Відповідно до встановленого завдання, була обрана оптимальна архітектура системи, що допоможе реалізувати потрібний програмний продукт. На основі визначених апаратних вимог до системи та її архітектури, було проаналізовано можливі засоби реалізації системи. Для клієнтського додатку було проведено порівняння можливих технологій реалізації. За результатами проведеного порівняння, були обрані інструменти та технології для проектування та розробки системи 2D дизайну нігтів.

## РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ

### 2.1. Визначення варіантів використання системи та об'єктно-орієнтованої структури системи

Для проведення проектування системи спершу необхідно визначити варіанти використання користувачами та скласти перелік вимог до системи.

#### *Бізнес вимоги:*

1. Основні цілі: проект створюється з метою розробки системи 2D дизайну нігтів.
2. Можливості: створення системи дизайну нігтів для клієнту манікюрного салону.
3. Представлення проекту: проект включає в себе клієнтський додаток для адміністратора та клієнту салону та сервер, що виконує операції розпізнавання зображень та обробки даних.

#### *Вимоги користувачів:*

1. Авторизація користувачів в системі: реєстрація нового користувача в системі та автентифікація зареєстрованого користувача;
2. Перегляд даних у системі;
3. Розширення існуючих даних системи новими матеріалами;
4. Розпізнавання кольору за вибраним зображенням;
5. Пошук матеріалів за заданим кольором;
6. Перегляд ціни манікюру з використанням обраного матеріалу.

#### *Функціональні вимоги:*

1. Доступ до бази даних та адміністрування даних системи;
2. Метод розпізнавання кольору на зображенні;
3. Симуляція обраного манікюру за допомогою технології доповненої реальності.

#### *Нефункціональні вимоги до системи:*

1. *Сприйняття.* Час необхідний для ознайомлення користувача з методом роботи з додатком має становити не більше пів години для досвідчених користувачів мобільних телефонів та веб-браузерів та не більше ніж півтори години для користувачів, які мають мінімальний досвід роботи з мобільними телефонами та ПК. Вважається, що адміністратор системи володіє високим рівнем користувацьких знань сучасних мобільних телефонів та ПК, тому час необхідний для ознайомлення користувачів цієї ролі становить не більше години, враховуючи додаткові функціональні можливості адміністратора.

2. *Надійність.* Система не повинна помилково закінчувати роботу без попередження для користувача. Помилки, створені користувачем під час його роботи, повинні бути оброблені системою валідації даних та поведінки користувача. Такі помилки мають бути забезпечені відповідним повідомленням про неправильне використання додатку, некоректний ввід даних та інші помилки.

3. *Продуктивність.* Система має підтримувати роботу користувача з додатком протягом всього часу безпосередньої взаємодії та підтримувати роботу запитів системи в фоновому режимі. Система повинна забезпечувати синхронізацію даних під час роботи різних користувачів та комунікацію адміністратора та звичайного користувача.

4. *Можливість експлуатації.* Мобільний додаток має автоматично оновлюватися на мобільному пристрої користувача за наявності дозволу зі сторони операційної системи. Веб-додаток повинен оновлюватись незалежно від системи кінцевого користувача.

Визначимо основні типи користувачів системи, так званих акторів. Під час роботи з об'єктом дослідження було виділено такі актори:

– Гість - незареєстрований користувач, який має доступ до даних системи та основних функцій;

- Клієнт – зареєстрований та автентифікований користувач, що має доступ до перегляду даних системи, основних та додаткових функцій додатку;
- Адміністратор - зареєстрований та автентифікований користувач який є адміністратором манікюрного салону і має доступ на зміну даних.

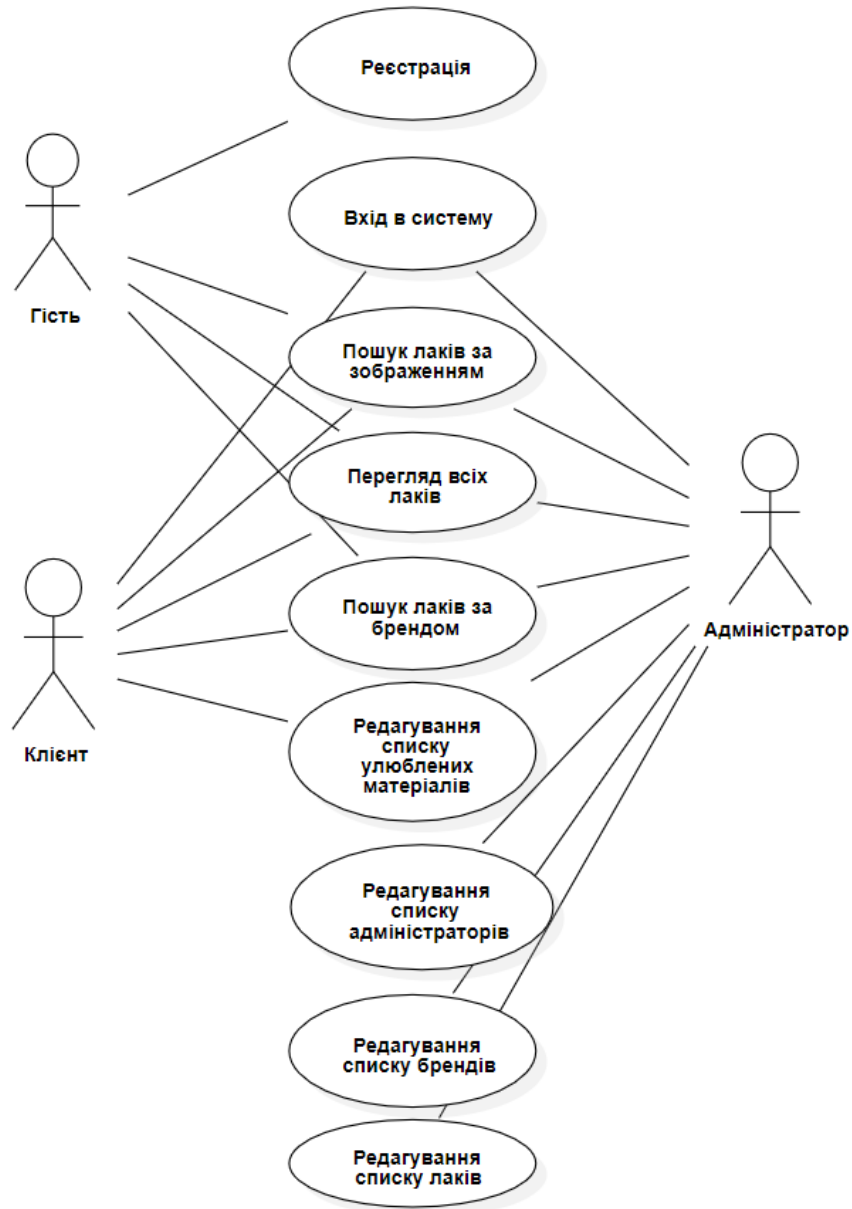


Рисунок 2.1 - Діаграма варіантів використання системи

Для опису варіантів використання застосовується діаграма варіантів використання (use case diagram) - діаграма, що відображує відносини, які існують між акторами і варіантами використання. Вона представляє собою засіб, що дає можливість замовнику, кінцевому користувачеві і розробнику спільно обговорювати функціональність та поведінку системи [28].

На рисунку 2.1 зображено діаграма варіантів використання системи, на якій схематично показані актори та основні дії, які виконують ці актори при роботі з системою.

Розглянемо опис наведених варіантів використання:

- Реєстрація – виконується лише гостем системи, який не створив облікового запису;
- Вхід в систему – авторизація існуючого користувача за вказаними даними, відбувається автоматично, за умови наявності облікового запису та історії попереднього входу;
- Пошук лаків за зображенням – використання зображення з пам'яті системи для пошуку лаків в системі за кольорами з зображення;
- Перегляд всіх лаків – перегляд всіх лаків в системі;
- Пошук лаків за брендом – перегляд лаків в системі з групуванням за назвою бренду;
- Редагування списку улюблених матеріалів – додавання та видалення лаків до переліку улюблених матеріалів;
- Редагування списку адміністраторів – додавання та видалення права доступу до дій адміністратора для користувачів;
- Редагування списку брендів – додавання та видалення брендів;
- Редагування списку лаків – додавання та видалення лаків.

Для створення клієнтського додатку необхідно створити об'єктно-орієнтовану систему, що відображатиме основні сутності та міститиме деякі функції додатку. Оскільки розробка клієнтський додатку здійснюється за використанням технології React.js, основна частина функціоналу описується у вигляді компонентів, що відповідають парадигмі функціонального програмування.

На рисунку 2.2 зображено діаграму класів клієнтського додатку.

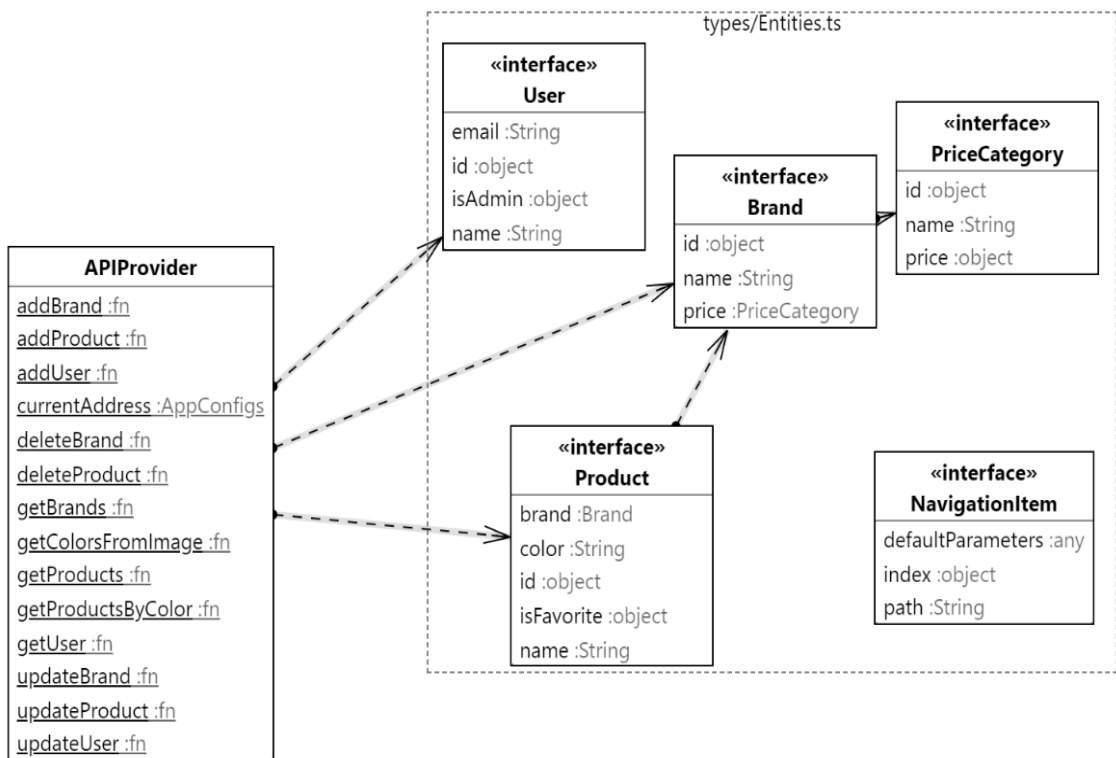


Рисунок 2.2 - Діаграма класів клієнтського додатку

Розглянемо діаграму класів клієнтського додатку:

- User – сутність користувача, містить інформацію про поштову адресу, ім'я та наявність прав адміністратора;
- Brand – виробник продукту. Сутність містить інформацію про назву та цінову категорію продукту;
- PriceCategory – цінова категорія певного бренду;
- Product – продукт (лак). Основна сутність системи, містить інформацію про назву, колір, бренд та належність до списку улюблених продуктів у поточного користувача додатку;
- NavigationItem – службова сутність для створення переліків можливих посилань в межах додатку;
- APIProvider – клас для взаємодії з серверною частиною додатку. Клас містить набір статичних методів для створення та надсилання HTTP запиту до серверу.

## 2.2. Розробка бази даних

Важливою частиною роботи системи є обробка інформації, а саме створення, оновлення, видалення та отримання даних. Система базується на класичній реляційній базі даних SQLite. Реляційна база потребує проектування сутностей, які будуть зберігатись в таблицях бази даних. Розглянемо основні сутності бази даних системи, їх перелік знаходиться в таблиці 2.1.

Таблиця 2.1

Перелік таблиць бази даних

Назва таблиці	Опис
Users	Користувачі системи
Products	Продукти (лаки) для створення манікюру
Brands	Бренди до яких належать продукти
PriceCategories	Типи цін на певні бренди
FavoriteProducts	Службові записи про відмічені улюблені лаки у користувачів

Після ознайомлення з загальною структурою бази даних, розглянемо детальніше структуру кожної таблиці з описом полів.

Таблиця 2.2

Структура таблиці Users

Назва поля	Тип даних	Ключ	Опис
Id	Integer	PK	Ідентифікатор користувача
Name	Text	-	Ім'я користувача
Email	Text	-	Електронна пошта користувача
IsAdmin	Integer(0    1)*	-	Позначка, чи є у користувача права адміністратора



Таблиця 2.3

Структура таблиці Products

Назва поля	Тип даних	Ключ	Опис
Id	Integer	PK	Ідентифікатор лаку
Name	Text	-	Назва лаку (код в системі виробника)
Color	Text	-	Колір лаку у вигляді RGB формату**
Brand	Integer	FK	Ідентифікатор бренду, до якого належить продукт

Таблиця 2.4

Структура таблиці Brands

Назва поля	Тип даних	Ключ	Опис
Id	Integer	PK	Ідентифікатор бренду
Name	Text	-	Назва бренду
PriceCategory	Integer	FK	Ідентифікатор цінової категорії, до якої належить бренд

Таблиця 2.5

Структура таблиці PriceCategories

Назва поля	Тип даних	Ключ	Опис
Id	Integer	PK	Ідентифікатор цінової категорії
Name	Text	-	Назва цінової категорії
Price	Integer	-	Ціна продукту в категорії

Структура таблиці FavoriteProducts

Назва поля	Тип даних	Ключ	Опис
Id	Integer	PK	Ідентифікатор
UserId	Integer	FK	Ідентифікатор користувача
ProductId	Integer	FK	Ідентифікатор продукту

\* - SQLite не підтримує тип даних Boolean. Для відображення таких даних використовується значення чисел 0 та 1 [6].

\*\* - Для роботи алгоритму пошуку за кольором, важливо зберігати колір у вигляді RGB значення.

Окрім детального опису таблиць та їх полів, розглянемо зв'язки між таблицями на схемі зв'язку таблиць у базі даних на рисунку 2.3.

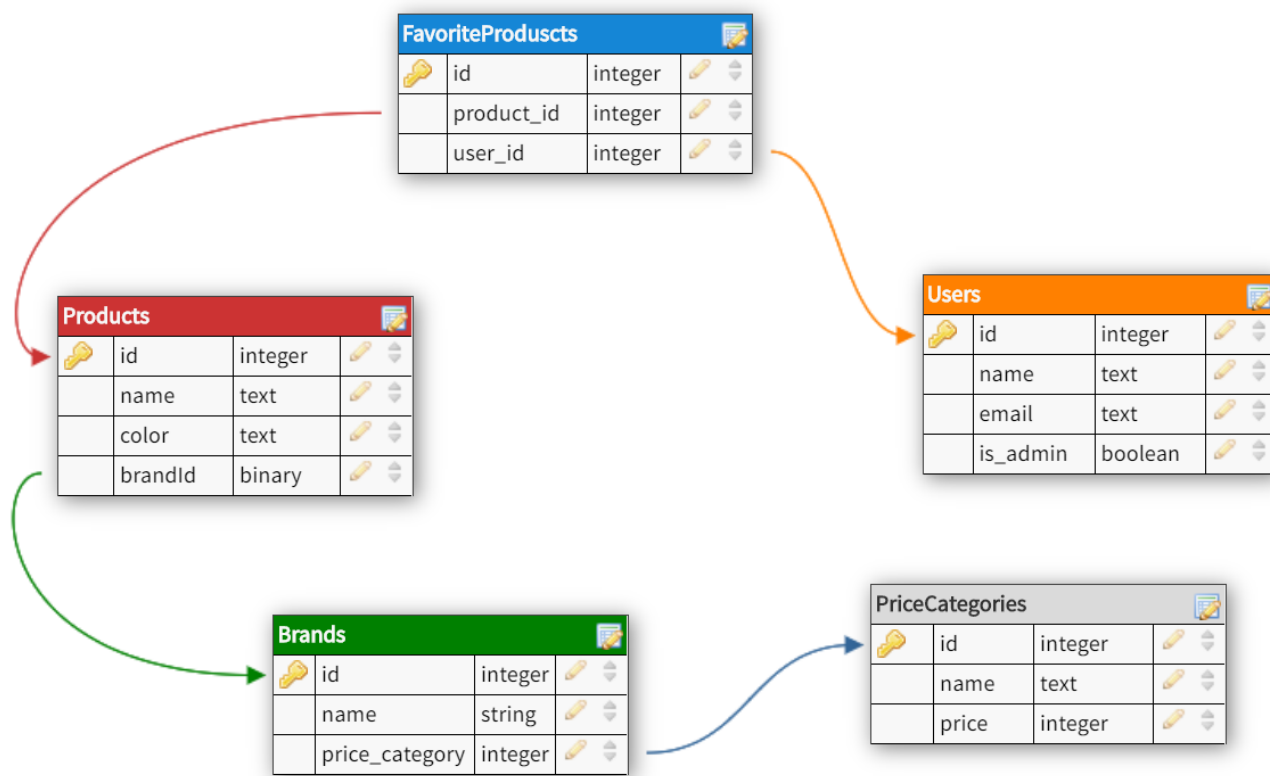


Рисунок 2.3 – Схема зв'язку таблиць бази даних

Схема зв'язку таблиць бази даних відображує основні таблиці та зовнішні ключі вказаних полів.

### 2.3. Проектування та реалізація алгоритмів роботи системи

Для проектування та реалізації алгоритмів необхідно спланувати процес роботи типового користувача у додатку.

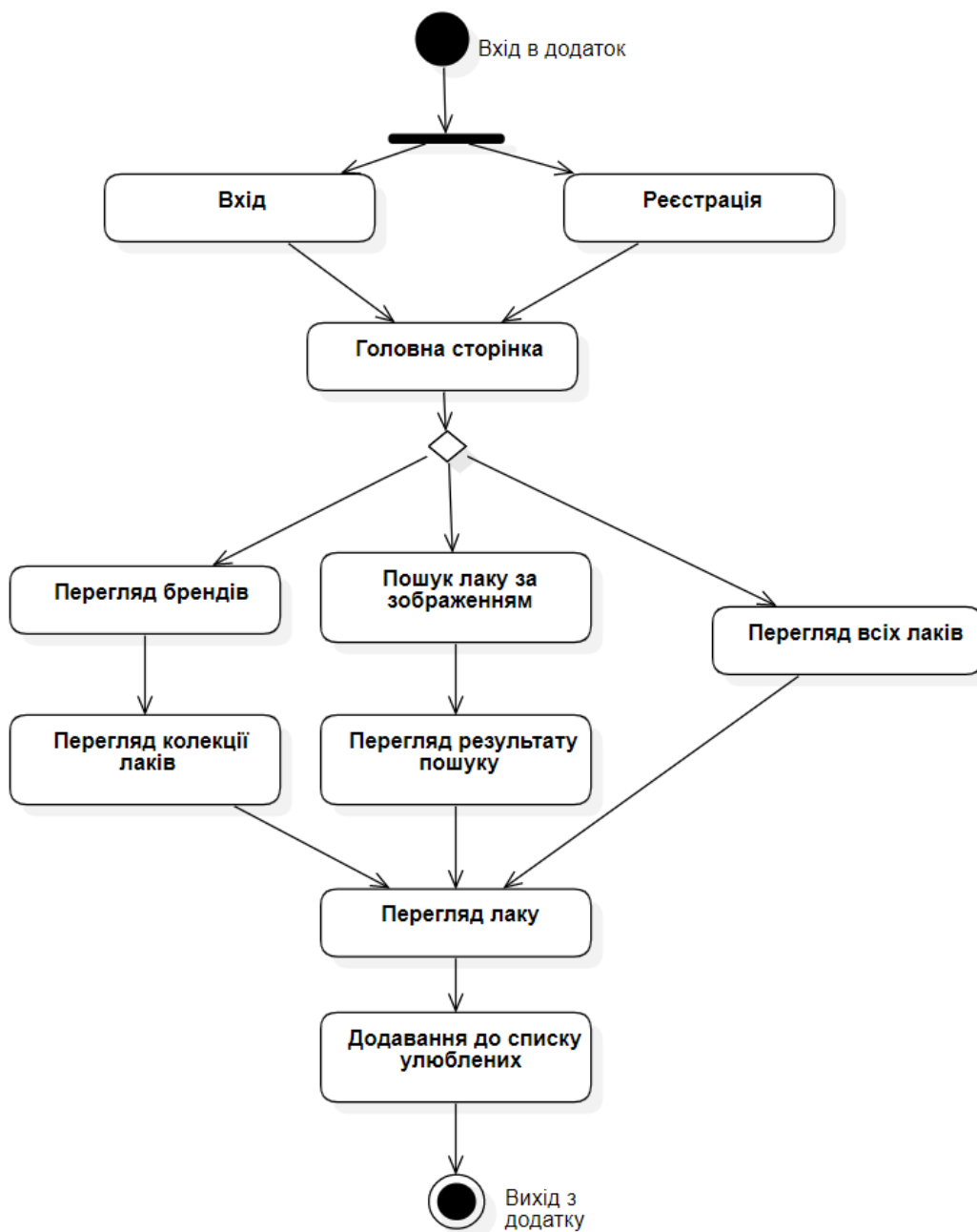


Рисунок 2.4 – Діаграма діяльності користувача

На рисунку 2.4 зображена діаграма діяльності користувача, який реєструється або входить в обліковий запис системи. Також, для спрощення

вигляду діаграми не додано можливості роботи адміністратора. Розглянемо послідовні кроки діяльності користувача за діаграмою.

- Вхід в додаток – користувач запускає додаток на обраній ОС;
- Реєстрація – створення облікового запису в системі;
- Вхід – користувач входить в існуючий обліковий запис системи;
- Головна сторінка – користувач переходить на головну сторінку додатку та отримує перелік можливих дій;
- Перегляд брендів – користувач отримує перелік брендів у базі даних;
- Пошук лаку за зображенням – користувач завантажує зображення та отримує перелік кольорів з зображення, за якими можна здійснити пошук лаків;
- Перегляд всіх лаків – користувач отримує перелік всіх лаків всіх брендів у базі даних;
- Перегляд колекції лаків – користувач отримує перелік всіх лаків за обраним брендом у базі даних;
- Перегляд результату пошуку – користувач отримує перелік лаків, що схожі за кольором з зображення;
- Перегляд лаку – користувач отримує детальну інформацію про обраний лак;
- Додавання до списку улюблених – користувач може додати лак до списку улюблених;
- Вихід з додатку – користувач припиняє роботу з додатком.

Додаток для користувача з правами адміністратора має ряд додаткових можливостей:

- Додавання, видалення та редагування брендів;
- Додавання, видалення та редагування лаків;
- Модифікація списку адміністраторів системи.

Процеси розпізнавання зображення та пошуку підходящих кольорів є нетривіальними задачами, реалізація яких потребує розробки власних алгоритмів. Розглянемо підходи до вирішення цих задач та алгоритми їхньої реалізації.

Пошук лаків за кольорами зображенням можна розділити на два процеси:

1. Розпізнавання кольорів на зображенні;
2. Порівняння лаків з заданим кольором пошуку.

Розпізнавання кольорів на зображенні задача є задачею кластерного аналізу.

Кластерний аналіз — це багатовимірна статистична процедура, яка виконує збір даних, які містять інформацію про вибірку об'єктів, а потім упорядковує дані в порівняно однорідні групи — кластери. Основна мета кластерного аналізу — знаходження груп схожих об'єктів у вибірці [28]. Узагальнена схема роботи алгоритмів кластеризації зображена на рисунку. В випадку вирішення задачі пошуку кольорів на зображенні, кластерами є кольори, які ми шукаємо.

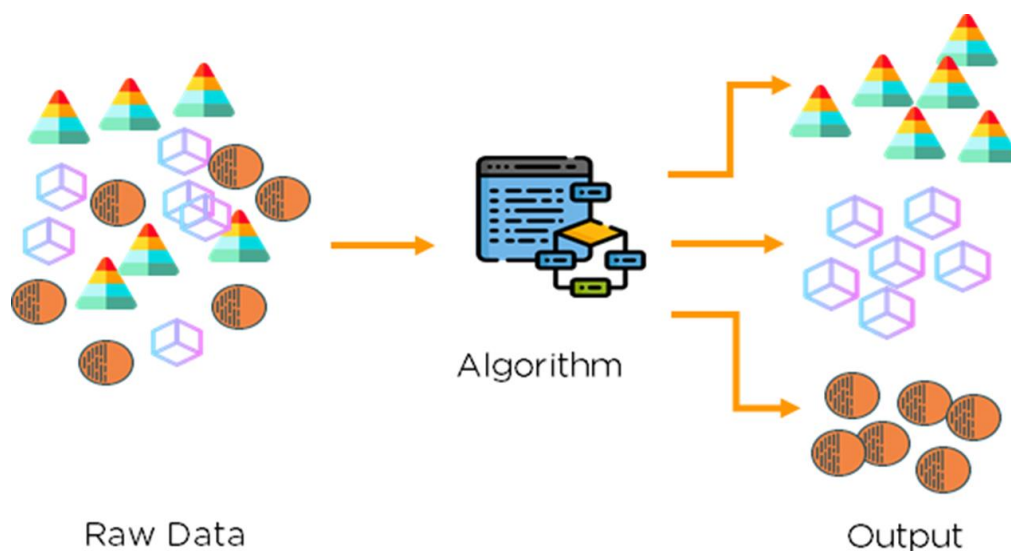


Рисунок 2.5 – Узагальнена схема роботи алгоритмів кластеризації

Задачі кластеризації вирішуються з застосуванням різних алгоритмів. Власне задача визначення кольору розв'язується за допомогою методу k-середніх. Головні переваги методу k-середніх — простота та швидкість роботи.

Метод k-середніх більш зручний для кластеризації великої кількості спостережень, ніж метод ієрархічного кластерного аналізу. Мета методу — розділити  $n$  спостережень на  $k$  кластерів, так щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням [24]. Кількість кластерів, яку необхідно утворити задається в якості вхідних даних для алгоритму.

Для створення алгоритму пошуку за кольором, необхідно розглянути формулу відстані кольорів (англ. Color difference).

Формула відстані кольорів – це математичне уявлення, яке чисельно висловлює відмінність між двома кольорами в колориметрії. Для використання цієї формули необхідно мати два кольори у моделі RGB, що має три чисельні складові: червоний, зелений, сині.

Найпростіше уявити кожен складову кольору RGB, як лінійну величину у Евклідовому просторі. В такому просторі можна знайти різницю між кожною складовою двох кольорів, використовуючи формулу віддалі між точками. В результаті отримуємо формулу:

$$distance = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2},$$

де  $R_1, G_1, B_1$  – складові компоненти першого кольору;

де  $R_2, G_2, B_2$  – складові компоненти другого кольору;

$distance$  – відстань між кольорами [5].

Чисельне значення відстані обернено пропорційне до візуальної різниці між кольорами. Це означає, що два кольори більш схожі між собою, якщо вони мають невелику чисельну відстань.

Блок-схема програмної реалізація пошуку лаку за кольором зображено на рисунку 2.6. Алгоритм пошуку досить простий, виконується лінійний перебір кожного доступного кольору, для якого знаходиться відстань з заданим кольором – критерієм пошуку. Після прорахунку відстаней всіх кольорів

відбувається сортування за значенням відстані. Перші елементи в відсортованому масиві – це найбільш схожі до пошукового запиту лаки.

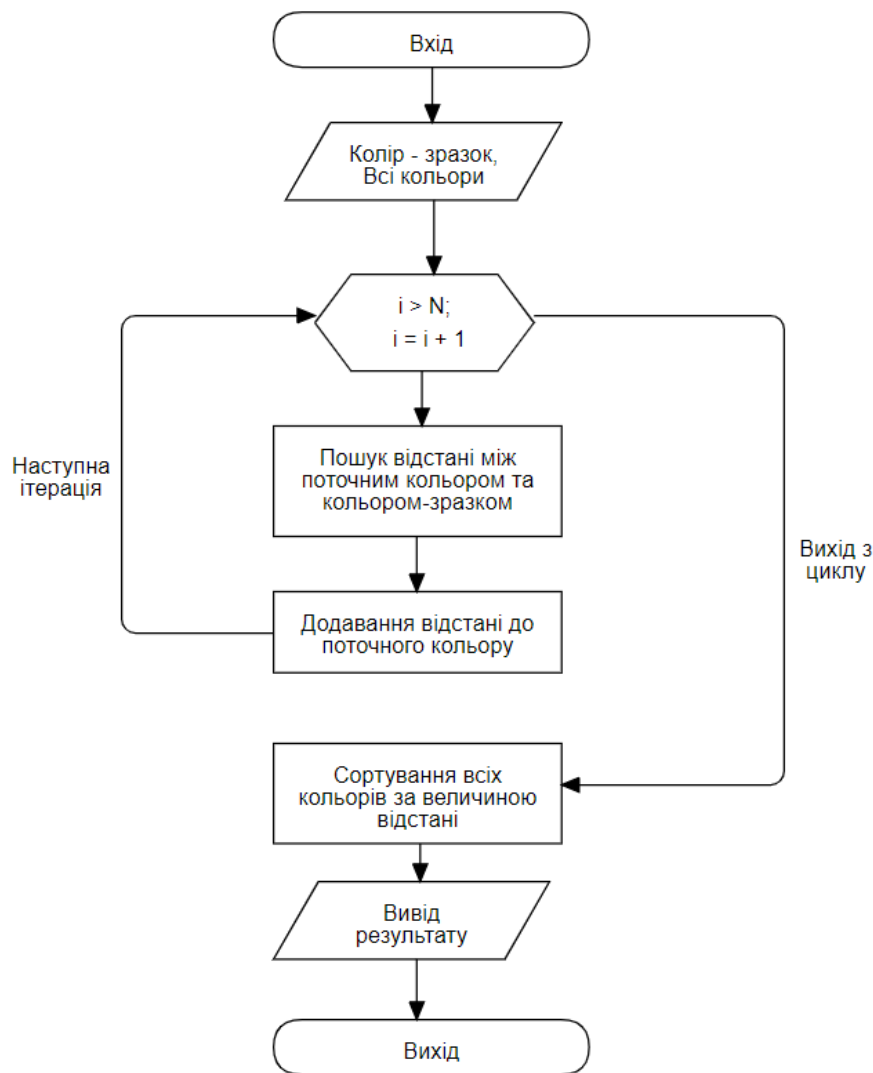


Рисунок 2.6 – Схема алгоритму пошуку кольору

#### 2.4. Реалізація функціоналу системи 2D дизайну нігтів

Реалізацію функціоналу системи необхідно розділити на дві основні частини: реалізація клієнтського додатку та реалізація серверного додатку. Розглянемо реалізацію ключових функцій серверного додатку.

Звернення до серверу відбуваються методом створення HTTP-запитів. Для обробки цих запитів ініціалізовано сервер та додаток на основі бібліотеки Flask. Точкою входу в серверний додаток є створення flask-додатку. Клієнтський додаток та серверний додаток можуть знаходитись на різних

серверах за різними адресами, тому необхідно активувати спільне використання ресурсів з різних джерел (CORS) [3].

Програмний код ініціалізації серверного додатку, активації CORS та приклад обробки HTTP запиту з отриманням брендів.

```
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/brands', methods=['GET'])
def get_brands():
    return jsonify(db_requests.get_brands())

if __name__ == "__main__":
    app.run(host="localhost", port=8080, debug=True)
```

В прикладі вище використовується команда `db_requests.get_brands()`. Це виклик функції `get_brands`, яка звертається до бази даних з відповідним SQL запитом. Функція `jsonify` перетворює дані у JSON формат для надсилання відповіді на запит клієнтові. Для роботи з базою даних створено клас `Database`, що реалізує шаблон проектування `Singleton`, який встановлює обмеження на кількість створених об'єктів класу [19]. Код реалізації класу `Database`:

```
class Database(metaclass=Singleton):
    connection = None

    def connect(self):
        if self.connection is None:
            self.connection = sqlite3.connect(DATABASE_PATH,
check_same_thread=False)
            self.cursor = self.connection.cursor()
            self.__create_tables()
        return self.cursor

    def __create_tables(self):
        self.cursor.execute(sql_create_brands_table)
        self.cursor.execute(sql_create_products_table)
        self.cursor.execute(sql_create_users_table)
        self.cursor.execute(sql_create_fav_products_table)
        self.cursor.execute(sql_create_fav_price_categories_table)
```



Розпізнавання зображення також відбувається на сервері. Нижче наведено код обробки зображення за допомогою методу кластеризації:

```
from sklearn.cluster import KMeans
import cv2
from collections import Counter

def format_RGB(color):
    return f"rgb({int(color[0])}, {int(color[1])}, {int(color[2])})"

def get_colors(image, number_of_colors):
    modified_image = cv2.resize(image, (600, 400), interpolation =
cv2.INTER_AREA)
    modified_image =
modified_image.reshape(modified_image.shape[0]*modified_image.shape[1], 3)

    clf = KMeans(n_clusters = number_of_colors)
    labels = clf.fit_predict(modified_image)
    counts = Counter(labels)

    center_colors = clf.cluster_centers_
    ordered_colors = [center_colors[i] for i in counts.keys()]
    formatted_colors = [format_RGB(ordered_colors[i]) for i in counts.keys()]

    return formatted_colors
```

Функція `get_colors` отримує зображення у вигляді об'єкту бібліотеки OpenCV, оброблює це зображення та викликає алгоритм кластеризації для пошуку кольорів. Кольори перетворюються у коректний RGB формат та повертаються з функції.

Реалізація формули знаходження відстані між кольорами та допоміжний клас для ініціалізації компонентів кольору:

```
class Color:
    def __init__(self, color):
        color_components = re.search("rgb\\((\\d{1,3}), (\\d{1,3}),
(\\d{1,3})\\)", color)
        self.red = int(color_components.group(1))
        self.green = int(color_components.group(2))
        self.blue = int(color_components.group(3))

    def get_color_difference(target_color, item):
        current_color = Color(item["color"])
        return math.sqrt(pow((target_color.red - current_color.red), 2)
+ pow((target_color.green - current_color.green), 2)
+ pow((target_color.blue - current_color.blue), 2))
```

Іншою важливою частиною системи є клієнтський додаток. Додаток реалізовано з використанням технології React Native, тому для розробки не

використовується HTML, а створення стилізації сторінок має схожі CSS властивості, проте відрізняється специфічним синтаксисом і рядом обмежень.

Для покращення структури коду та використання можливості типізації, в проєкті було додано підтримку розробки мовою TypeScript.

Розглянемо деякі ключові компоненти клієнтського додатку.

Для комунікації з сервером створено клас `APIProvider`. В цьому класі реалізуються методи створення HTTP-запитів до серверу. Програмний код запиту на отримання кольорів за зображенням:

```
static getColorsFromImage = async (image: ImagePickerResult):  
Promise<Array<string>> => {  
  let contentType: string =  
    Platform.OS === 'web' ? 'text' : 'multipart/form-data';  
  try {  
    let response: Response = await fetch(  
      `${APIProvider.currentAddress}/get-color`,  
      {  
        method: 'POST',  
        body: transformImageToFormData(image),  
        headers: { 'Content-Type': contentType },  
      }  
    );  
    let json = await response.json();  
    return json;  
  } catch (e) {  
    console.warn(e);  
    return [];  
  }  
};
```

Як і інші методи звернення до серверу, метод `getColorsFromImage` – асинхронний. Це означає, що виконання запиту та очікування на результат запиту не блокуватиме інші дії користувача. Для позначення того, що метод є асинхронним, вказано модифікатор `async` перед описом методу. Обов’язковою умовою `async` методу є те, що він повинен повертати результат у вигляді JavaScript-об’єкту `Promise`. `Promise` – це контейнерний об’єкт, тобто він містить в собі додаткові дані. В випадку з методом `getColorsFromImage` це масив рядкових значень – кольорів на зображенні.

Схожим чином формуються інші методи для HTTP-запитів до серверу.

Клієнтський додаток має різні сторінки, тому важливо сформувати спосіб навігації між цими сторінками. Компонент AppNavigator містить реалізацію підтримки переміщення між сторінками додатку:

```
const Drawer = createDrawerNavigator();

const navigatorTheme = {
  ...DefaultTheme,
  colors: {
    ...DefaultTheme.colors,
    background: 'transparent',
  },
};

export const AppNavigator = (): React.ReactElement => {
  return (
    <NavigationContainer theme={navigatorTheme}>
      <SafeAreaView style={{ flex: 1 }} edges={['top']}>
        <Drawer.Navigator
          screenOptions={{ gestureEnabled: true }}
          drawerContent={(props) => <MainDrawer {...props} />}
        >
          <Drawer.Screen name='Home' component={HomeScreen} />
          <Drawer.Screen name='Products' component={ProductsScreen} />
          <Drawer.Screen name='Brands' component={BrandsScreen} />
        </Drawer.Navigator>
      </SafeAreaView>
    </NavigationContainer>
  );
};
```

Основний функціонал клієнтського додатку – це сторінки для взаємодії з клієнтом. Кожна сторінка є React-компонентом з власним станом та функціональними можливостями. Нижче наведено програмний код ініціалізації головної сторінки та створення запиту на пошук за зображенням:

```
export default ({ navigation }): React.ReactElement => {
  const theme = useTheme();
  const [imageUrl, setImageUrl] = useState<null | string>(null);
  const [selectedColors, setSelectedColors] = useState<Array<string>>([]);

  useEffect(() => {
    imagePickerPermissionRequest();
  }, []);

  const launchImagePicker = async () => {
    let result = await ImagePicker.launchImageLibraryAsync();
    if (!result.cancelled) {
      setImageUrl(result.uri);
      setSelectedColors([]);
      let colors = await APIProvider.getColorsFromImage(result);
      setSelectedColors(colors);
    }
  };
};
```

```

    }
  };

  return (
    ... // об'єкт розмітки компоненту
  );
};

```

Створення React компоненту – це процес створення функції, що відповідатиме вимогам системи React. Компонент містить опис його ініціалізації, обробку користувацьких подій та опис зовнішнього вигляду компоненту. В наведеному програмному коді є використання функції `useEffect`. Ця функція є подією життєвого циклу компоненту (hook), яка відпрацьовує в момент ініціалізації компоненту та/або зміни його стану [33]. Для компоненту головної сторінки додатку `useEffect` викликається одразу після ініціалізації компоненту та виконує запит на отримання доступу до файлової системи пристрою.

Для відображення розмітки, компонент повинен повертати об'єкт типу `ReactElement`. React надає можливість створювати шаблони компоненту з використанням логічних операторів та фрагментів коду. Це дає можливість створювати гнучкі інтерфейси, що перебудовуються залежно від стану компоненту. Розглянемо фрагмент коду для створення розмітки шаблону компоненту головної сторінки додатку:

```

<Layout style={{ flex: 1 }}>
  <TopNavigationMain navigation={navigation} />
  <Layout style={styles.iconContainer}>
    <Layout>
      {imageUrl ? (
        <Image style={styles.image} source={{ uri: imageUrl }} />
      ) : (
        <PlusCircleIcon
          style={styles.icon}
          fill={theme['color-primary-400']}
        />
      )}
    <Button
      status='success'
      appearance='outline'
      style={styles.button}
      onPress={launchImagePicker}
    >
      {imageUrl ? 'Change image' : 'Select image'}
    </Button>
  </Layout>
</Layout>

```

```

</Layout>
<Layout style={styles.buttonContainer}>
  {selectedColors.map((color, index) => (
    <TouchableOpacity
      key={index}
      style={{ ...styles.colorBlock, backgroundColor: color }}
      onPress={...}
    ></TouchableOpacity>
  ))}
</Layout>
</Layout>
</Layout>

```

В цьому фрагменті коду використовуються тернарні оператори для визначення певних елементів. Наприклад, якщо користувач ще не завантажив зображення, на сторінці відображається іконка додавання зображення, інакше додається обране зображення. Аналогічна логіка описана зміни для підпису кнопки.

Додавання стилів відбувається в шаблоні компоненту. Для цього існує спеціальний атрибут `style`, який приймає об'єкт з описом правил стилізації. Ці правила можна описувати безпосередньо в атрибуті елементу, а можна створювати таблиці стилів наступним чином:

```

const styles = StyleService.create({
  iconContainer: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  icon: {
    width: 160,
    height: 160,
  },
  image: {
    width: 300,
    height: 300,
    resizeMode: 'contain',
    marginBottom: 10,
  },
  button: {
    marginTop: 8,
    marginBottom: 5,
  }
})

```

На відміну від класичного CSS, в правилах стилізації відсутні одиниці вимірювання. Деякі властивості співпадають з таблицями CSS (наприклад

width, height, justifyContent) проте є ряд специфічних властивостей, таких як resizeMode.

Інші сторінки створені схожим чином та змінюються в залежності від наявності різних функцій.

#### Висновки до другого розділу

Система 2D дизайну нігтів реалізована як поєднання клієнтського та серверного додатків для створення цілісної платформи. Встановлені бізнес вимоги, вимоги користувачів, функціональні та нефункціональні вимоги стали основою для проектування системи. Визначена структура сутностей була використана для розробки об'єктно-орієнтованої структури клієнтського додатку та проектування бази даних.

В другому розділі кваліфікаційної магістерської роботи було проведено формування варіантів використання системи, визначено основні актори та їх можливості. Проведено проектування та розробка бази даних.

Також, було розроблено діаграми діяльності користувача, визначено та описано алгоритми роботи основних функцій системи. На основі описаних алгоритмів, було розроблено основний функціонал як серверного, так і клієнтського додатку.

Також, було наведено приклади програмного коду з поясненнями його основних особливостей.

## РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

### 3.1. Порядок встановлення та налаштування системи.

Система 2D дизайну нігтів складається з клієнтського та серверного додатку. Кожен з цих компонентів має свій процес встановлення та налаштування. Клієнтський додаток розроблений з використанням технології React Native, тому він може бути доступним на різних платформах: ОС Android, IOS, веб-браузер. Серверний додаток включає в себе систему Firebase автентифікації, REST API та процес бази даних.

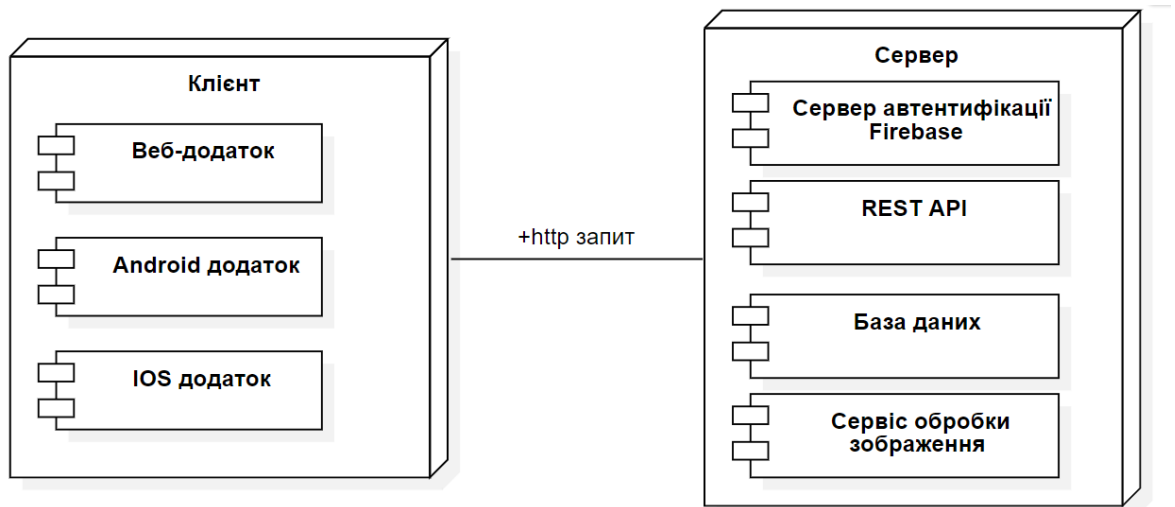


Рисунок 3.1 – Діаграма розгортання системи

На рисунку 3.1 зображена діаграма розгортання системи. На цій діаграмі зображено основні компоненти клієнтської та серверної частин додатку та їх взаємозв'язок. Розглянемо процеси встановлення та налаштування кожного з компонентів системи.

Для налаштування серверної частини додатку необхідно створити проект в системі Firebase та налаштувати параметри клієнтських додатків. В консолі управління проектами Firebase <https://console.firebase.google.com/> необхідно створити клієнтські додатки. На рисунку 3.2 зображено інтерфейс налаштування проекту Firebase.

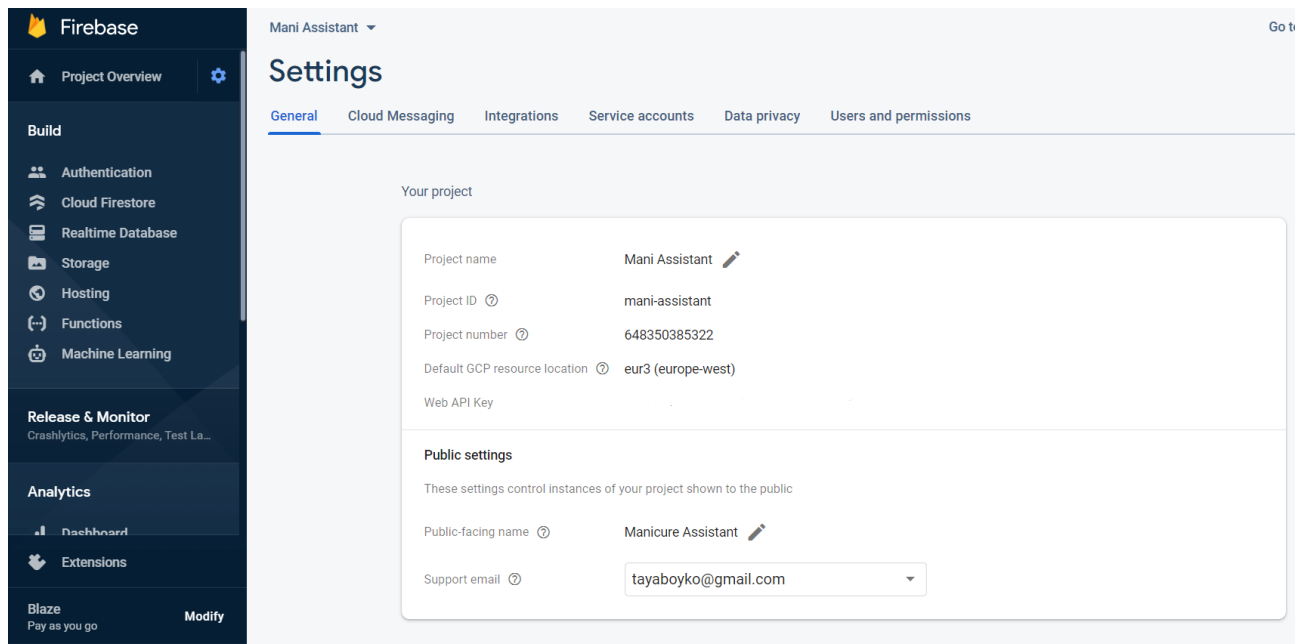


Рисунок 3.2 – Інтерфейс консолі налаштування проекту у Firebase

Після додавання клієнтських додатків та їх конфігурації, необхідно активувати систему автентифікації. Для цього потрібно перейти за посиланням у пункті меню «Authentication» та активувати обраний спосіб.

Для встановлення основних компонентів серверної частини використовується система контейнеризації Docker. Docker – це програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації. Ця система дозволяє скласти додаток з усім його системним оточенням і залежностями в контейнер, який може бути виконаний на віртуальній Linux-системі [7].

Для створення та конфігурації Docker-контейнеру використовується файл налаштування Dockerfile. Програмний код файлу налаштування наведено нижче:

```
FROM python:3.9-buster

RUN pip install --upgrade pip
RUN pip install numpy Flask flask_cors gunicorn KMeans opencv-python-headless scikit-learn

WORKDIR /app
COPY . /app
ENV PORT 8080
CMD exec gunicorn --bind 0.0.0.0:$PORT --workers 1 --threads 8 --timeout 0
app:app
```



Завдяки описаній конфігурації у Dockerfile, для контейнеру встановлюються необхідні залежності, оголошуються системні зміни та описується запуски додатку. Розгортання Docker-контейнеру може відбуватись локально (на серверній машині), або віддалено на хмарному оточені.

Для встановлення стабільного доступу до серверного додатку використовується хмарне оточення. В якості такого оточення використовується платформа Google Cloud Platform [11]. GCP – це набір сервісів хмарного обчислення на базі інфраструктури від компанії Google. Для розгортання серверного додатку на платформі Google Cloud Run використовується пакет команд gcloud.

Програмний код для розгортання контейнеру в хмарному оточені наведено нижче:

```
PROJECT_ID=$(gcloud config get-value project)
DOCKER_IMG="gcr.io/$PROJECT_ID/app"
cd src
gcloud builds submit --tag $DOCKER_IMG --timeout=10000
REGION="europe-west1"
gcloud run deploy app --image $DOCKER_IMG --platform managed --region
$REGION --allow-unauthenticated
```

Для розгортання контейнеру на хмарному оточені необхідно задати ряд параметрів:

- Назва проекту;
- Назва контейнеру;
- Регіон розгортання контейнеру;
- Правило дозволу на доступ до контейнеру.

Цей код запускається на машині розробника у директорії, що містить програмний код серверного додатку. Після виконання цього коду Google Cloud Run надає посилання для доступу до розгорнутого контейнеру у хмарі.

Порядок встановлення клієнтського додатку є простішим. Для встановлення веб-додатку необхідно розгорнути програмний код на статичному сервері.

Для встановлення Android додатку на пристрої, необхідно встановити програмний файл з розширенням .apk у систему та надати йому відповідні доступи.

Для встановлення IOS додатку на пристрої, необхідно встановити програмний файл з розширенням .app у систему. Для встановлення такого файлу не з офіційного магазину додатків AppStore, необхідно надати дозвіл на встановлення файлів від невідомих постачальників.

### 3.2. Структура інтерфейсу клієнтського додатку

Розглянемо основні сторінки клієнтського додатку на прикладі Android додатку на емуляторі Google Pixel з встановленою версією оперативної системи Android 11.

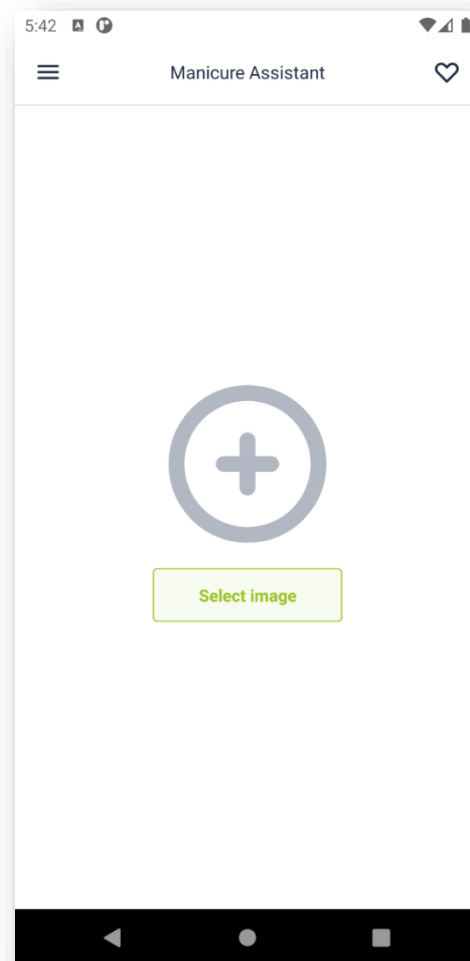


Рисунок 3.3 – Інтерфейс головної сторінки додатку

Основною функцією клієнтського додатку є пошук за зображенням, тому ця функція відображується на основному екрані. Для пошуку лаків за зображенням, користувач має завантажити бажане зображення з пам'яті пристрою.

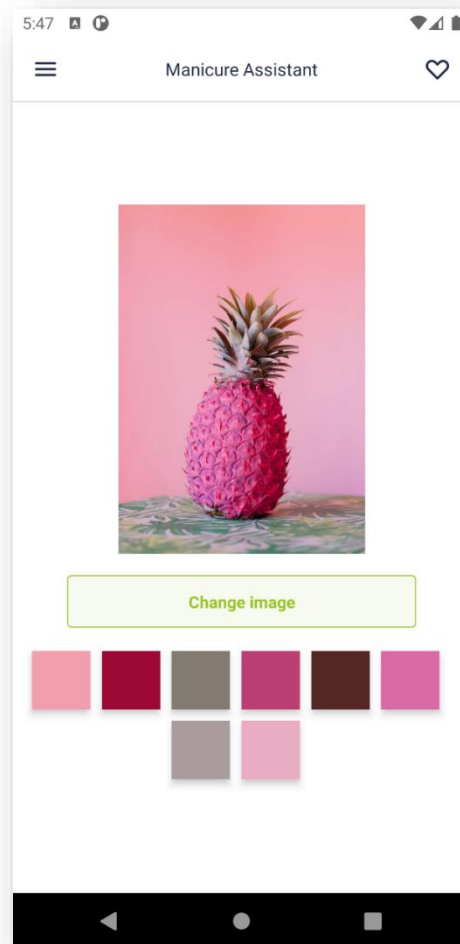


Рисунок 3.4 – Інтерфейс головної сторінки  
з завантаженим зображенням

Як тільки користувач додає зображення для пошуку, автоматично відбувається запит на сервер для отримання кольорів з зображення. Результатом розпізнавання зображення є перелік кольорів, які відображаються під доданим зображенням. Інтерфейс головної сторінки з завантаженим зображенням для пошуку зображено на рисунку 3.4.

Для того щоб перейти до результатів пошуку за обраним кольором, необхідно натиснути на обраний колір серед наявних в переліку.

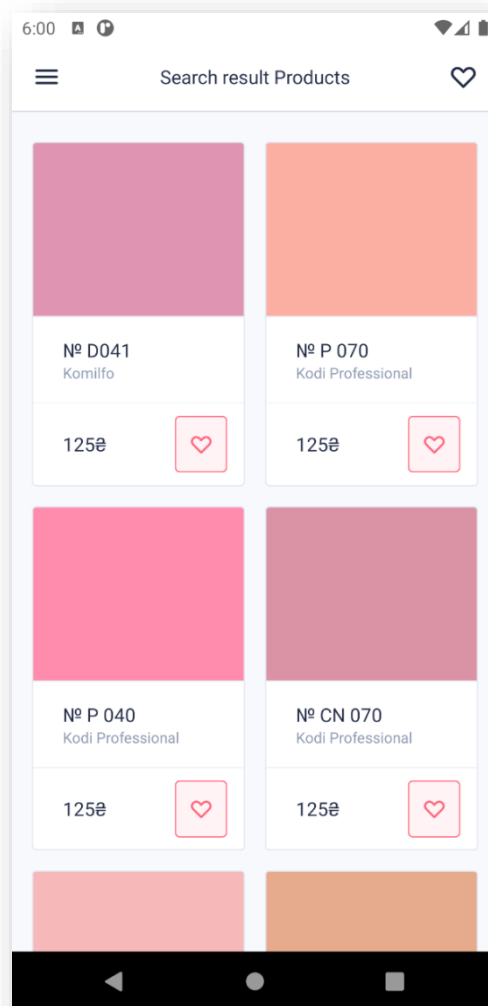


Рисунок 3.5 – Результат пошуку матеріалів за обраним кольором

Натиснувши на обраний колір, користувач переходить на сторінку результату пошуку. На цій сторінці зображено перелік найбільш підходящих матеріалів за кольором. Інтерфейс сторінки результатів пошуку зображено на рисунку 3.5.

Кожен матеріал має свою назву (код за виробником), бренд та ціну в залежності від цінової категорії, що встановлена для певного бренду. Ця інформація зображена у переліку матеріалів для надання повного опису користувачеві.

В випадку, якщо користувач авторизований у додатку, йому доступна функція додавання певного матеріалу до переліку улюблених. Цю дію можна виконати, натиснувши на кнопку з зображенням серця.

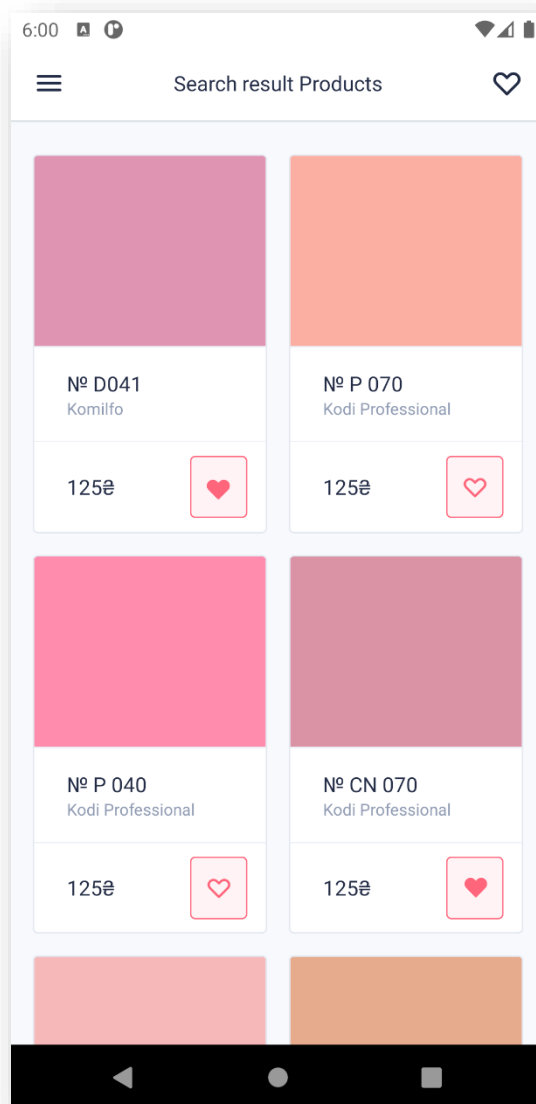


Рисунок 3.6 – Додавання матеріалів до списку улюблених

Наприклад, користувач обрав перший та третій колір та додав його до переліку улюблених, в такому разі інтерфейс кнопки зміниться, на екрані з'явиться індикатор того, які продукти додані до улюблених. Результат цієї дії зображено на рисунку 3.6.

На головних сторінках додатку є верхнє меню. В цьому меню знаходиться кнопка відкриття навігаційної панелі, назва поточної сторінки та кнопка переходу до списку улюблених матеріалів (за умови авторизації в системі).

Навігаційна панель відкривається з анімацією та займає всю площу екрану. Основна частина навігаційної панелі містить такі елементи:

- Назва додатку та його зображення;
- Посилання на головну сторінку;
- Посилання на перелік брендів;
- Посилання на перелік матеріалів;
- Кнопка авторизації.

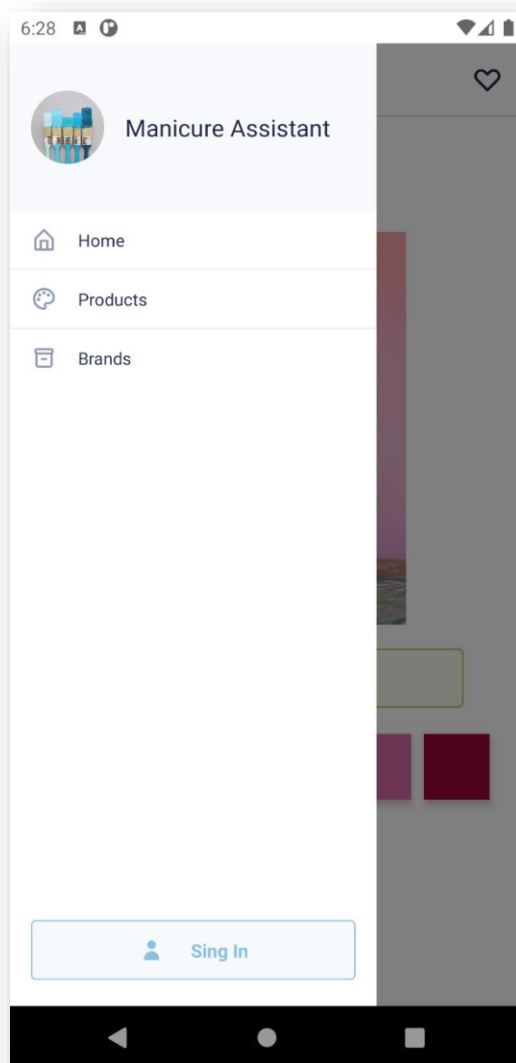


Рисунок 3.7 – Інтерфейс навігаційної панелі додатку

Кнопка авторизації має два стани, в залежності від того авторизований користувач, чи ні. Якщо користувач не авторизований, кнопка має підпис

«Увійти в обліковий запис», інакше, підпис змінюється на «Вийти з облікового запису».

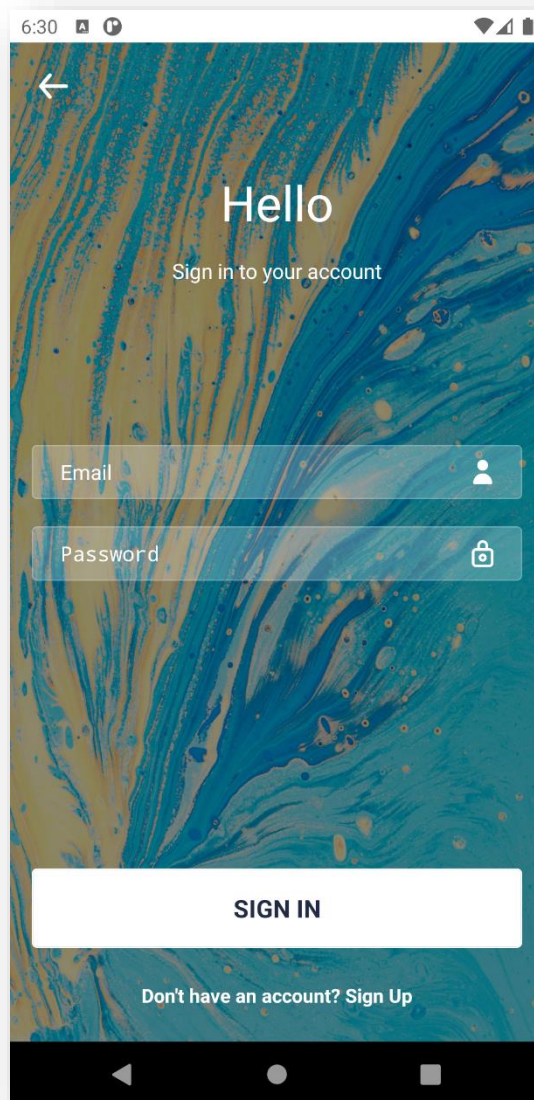


Рисунок 3.8 – Інтерфейс сторінки авторизації

Сторінка авторизації зображена на рисунку 3.8. На цій сторінці знаходиться привітання до користувача, поле вводу електронної адреси облікового запису, поле вводу паролю, кнопка «Ввійти в обліковий запис» та посилання на сторінку створення облікового запису, за умови, що користувач не має створеного облікового запису. Для входу в існуючий обліковий запис необхідно ввести електронну пошту та пароль. У випадку, якщо користувач

успішно ввів ці данні, система авторизує користувача та сторінка входу в обліковий запис змінюється на попередню робочу сторінку.

У випадку, якщо користувач бажає створити новий обліковий запис, він переходить на сторінку його створення, інтерфейс якої зображено на рисунку 3.9.

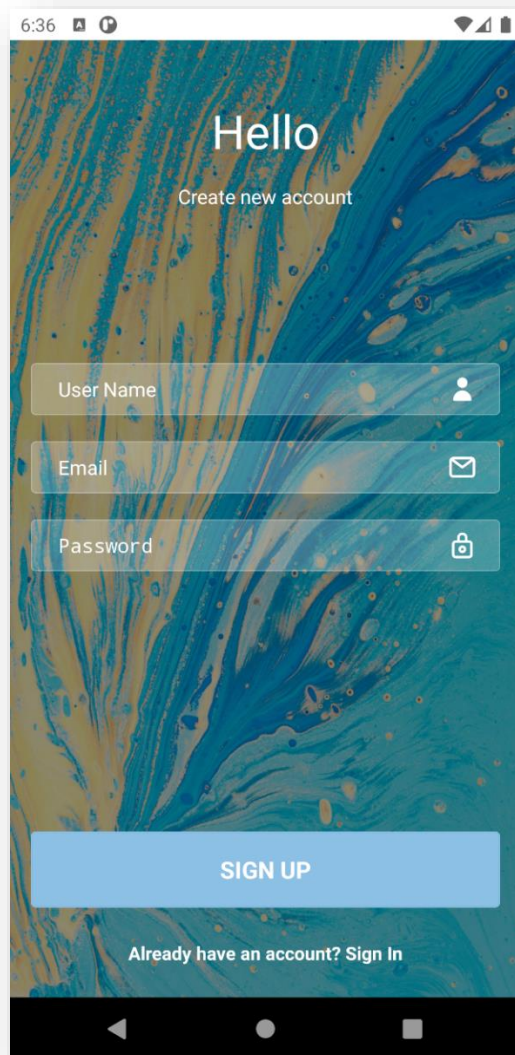


Рисунок 3.9 – Інтерфейс сторінки створення облікового запису

На цій сторінці знаходиться привітання до користувача, поле вводу імені, поле вводу електронної адреси, поле вводу пароллю, кнопка «Створити обліковий запис» та посилання на сторінку входу в існуючий обліковий запис. Для створення облікового запису необхідно ввести ім'я, електронну пошту та пароль. У випадку, якщо користувач успішно ввів всі данні, система створить



нового користувача та сторінка створення в облікового запису зміниться на головну сторінку.

Окрім авторизації та пошуку матеріалів, користувач може переглядати всі доступні бренди та продукти в манікюрному салоні.

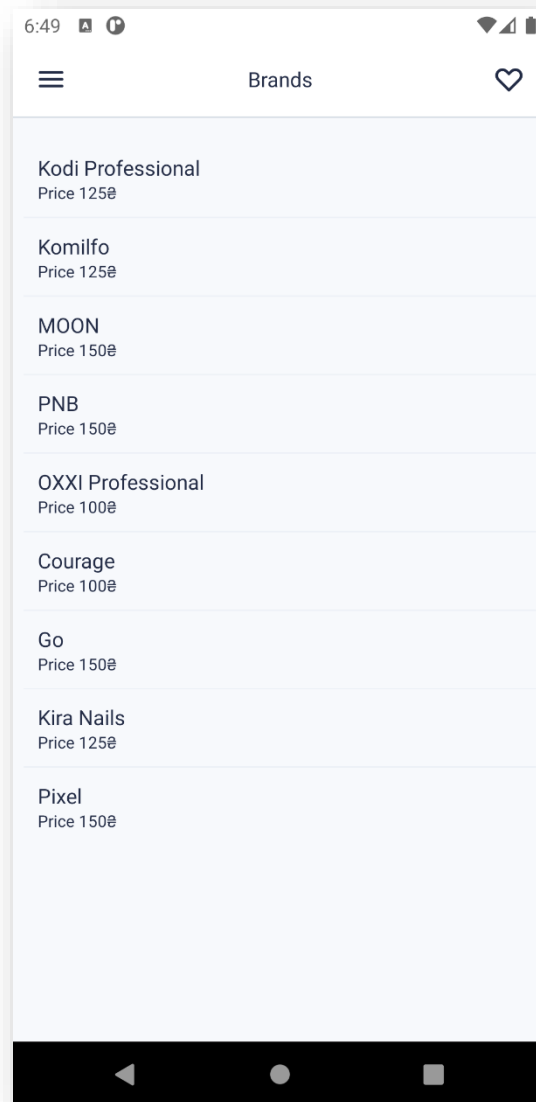


Рисунок 3.10 – Сторінка переліку брендів

На рисунку 3.10 зображено інтерфейс сторінки переліку брендів. Сторінка відображує всі наявні бренди та їх цінові категорії. Кожен бренд є посиланням на сторінку відображення матеріалів цього бренду. Натиснувши на обраний бренд, додаток переходить на відповідний перелік матеріалів.

Наприклад, користувач обрав переглянути всі продукти бренду «Komilfo». В результаті буде отримано перелік всіх доступних лаків цього бренду. Зовнішній вигляд такого переліку зображено на рисунку 3.11.

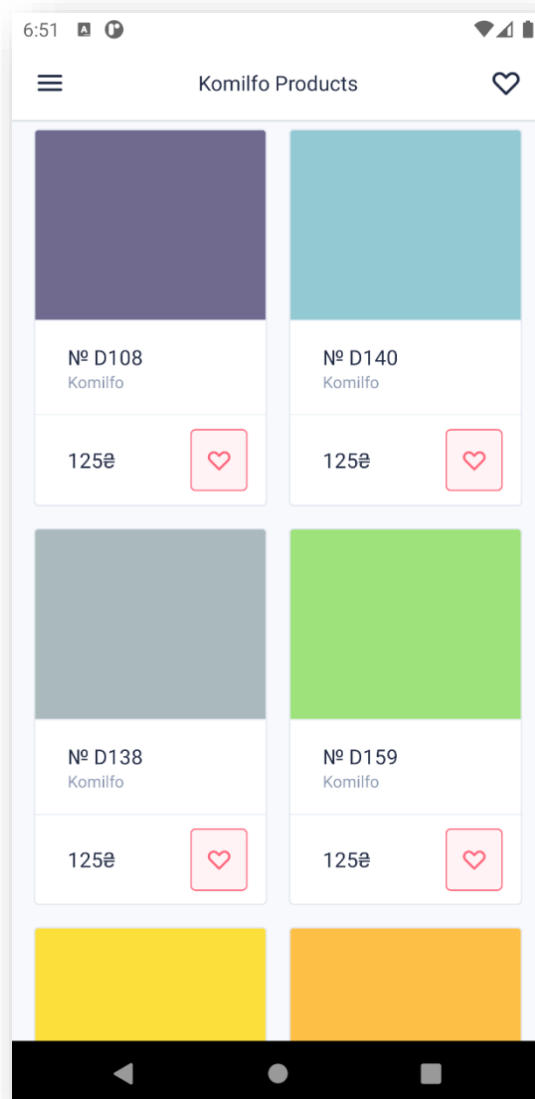


Рисунок 3.11 – Сторінка перегляду матеріалів за брендом «Komilfo»

Для користувачів з правом доступу до ролі адміністратора, інтерфейс є більш розширеним. Основні функції роботи додатку зберігаються, проте адміністратор має можливість виконувати додаткові дії.

Серед цих дій є можливості виконання операцій додавання, редагування та видалення брендів та продуктів. Розглянемо деякі сторінки інтерфейсу для користувача з роллю адміністратора.

Сторінка переліку брендів для користувача-адміністратора має більш складний інтерфейс. На цій сторінці з'являється кнопка додавання нового бренду. Окрім цього, кожен бренд у переліку має дві додаткові кнопки: редагування та видалення. Інтерфейс сторінки переліку брендів для адміністратора зображено на рисунку 3.12.

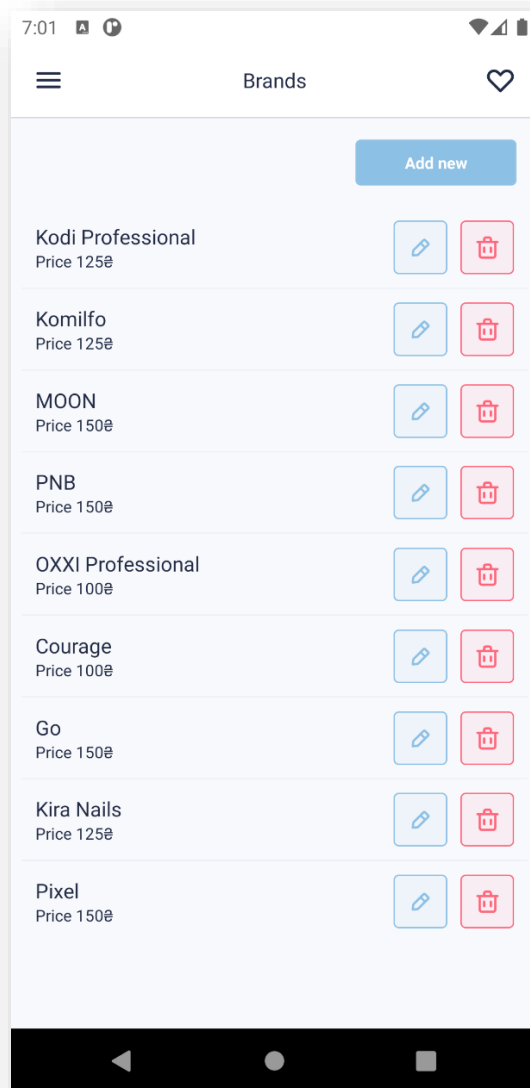


Рисунок 3.12 – Інтерфейс сторінки переліку брендів для адміністратора

Для зручності користувача, зовнішній вигляд кнопок відповідає загальноприйнятим підходам дизайну інтерфейсу. Кнопка редагування бренду – кнопка блакитного кольору з зображенням олівця, кнопка видалення бренду – кнопка червоного кольору з зображенням сміттєвої корзини. Натиснувши на кнопку редагування бренду, користувач переходить на сторінку редагування.

Інтерфейс сторінки редагування зображено на рисунку 3.13. На цій сторінці є поле вводу назви бренду та перелік доступних цінових категорій. Поле вводу назви бренду за замовчуванням містить поточну назву бренду, а цінова категорія за замовчуванням обрана поточна. Поле вибору цінової категорії є списком, що розгортається. Користувач може обрати потрібний варіант серед списку.

В нижній частині сторінки знаходиться кнопка збереження, натискання на яку відправить нову інформацію до серверу для зміни бази даних.

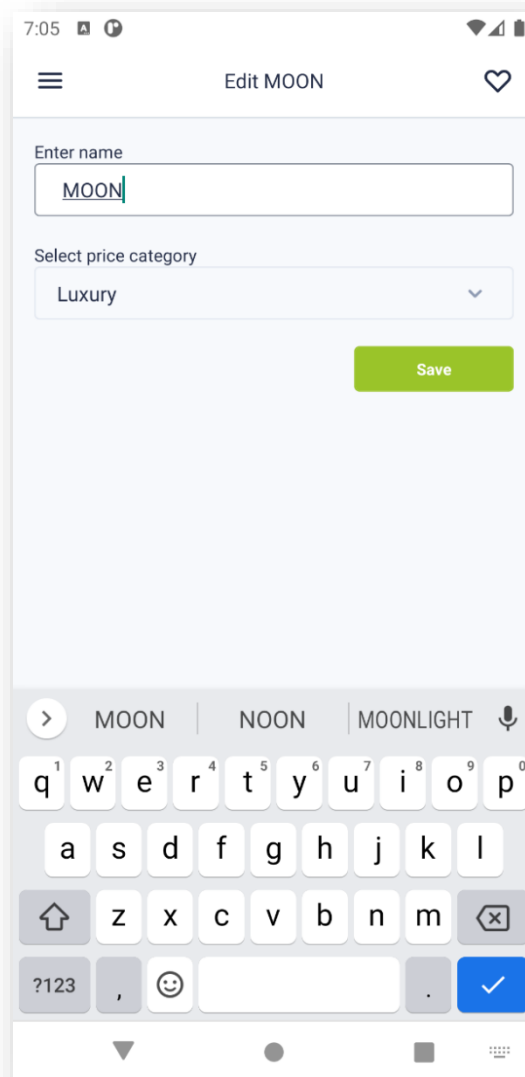


Рисунок 3.13 – Інтерфейс сторінки редагування бренду

Інтерфейс сторінки додавання нового бренду виглядає аналогічно до інтерфейсу сторінки редагування бренду, але не має заповнених значень за замовчуванням.

Сторінка перегляду переліку матеріалів також має додаткові можливості для користувача-адміністратора, інтерфейс цієї сторінки зображено на рисунку 3.14. Кожен елемент списку матеріалів має дві додаткові кнопки: кнопку редагування продукту з зображенням олівця та кнопку видалення продукту з зображенням сміттєвої корзини. Натискання на кнопку редагування відкриває сторінку зміни інформації про продукт. Відповідно, натискання на кнопку видалення матеріалу, надсилає на сервер запит про видалення матеріалу.

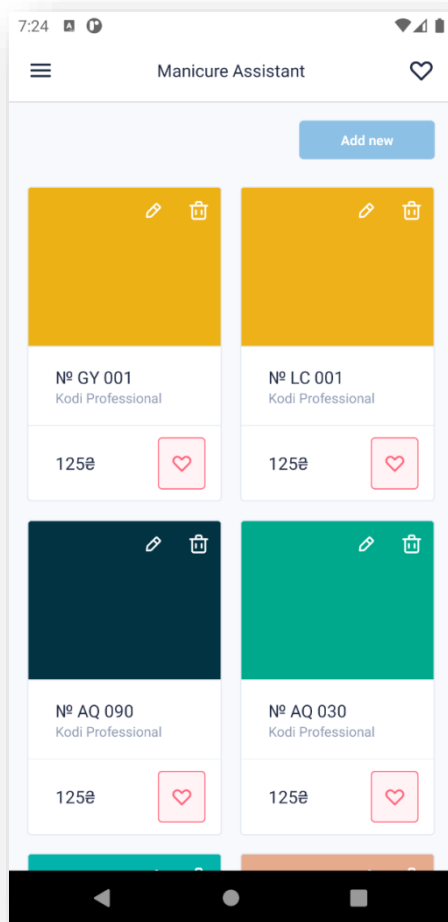


Рисунок 3.14 – Інтерфейс сторінки переліку матеріалів для адміністратора

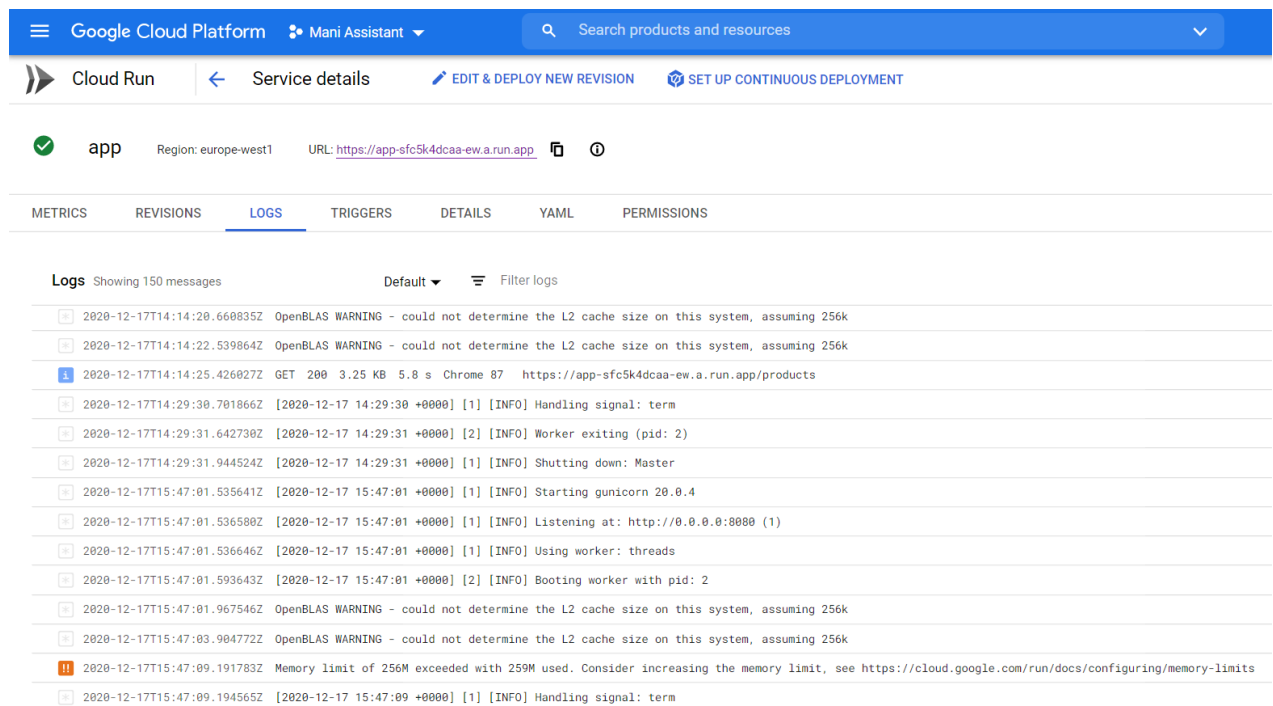
Також, на сторінці переліку продуктів адміністратору доступна кнопка додавання нового продукту.

### 3.3. Тестування роботи системи

Важливим етапом процесу розробки програмного забезпечення є його тестування. Тестування ПЗ допомагає знайти помилки в роботі системи з метою їх виправлення.

Для перевірки працездатності системи періодично проводились різні типи тестування. Під час розробки функціональності, або зміни програмного коду проводилось поверхневе тестування. Після розробки певної функціональної особливості проводилось функціональне тестування, тестування зовнішнього вигляду користувацького інтерфейсу та його зручності.

Для перевірки серверної частини та відслідковування помилок на сервері використовувався модуль аналітики хмарного сервісу Google Cloud Run. Цей модуль надає інформацію про навантаження на сервер, кількість запитів до нього та хронологію цих запитів. Окрім того, під час відслідковування помилкових ситуацій, використовувався сервіс журналу повідомлень (рисунок 3.15).



Google Cloud Platform									
Search products and resources									
Cloud Run Service details									
app Region: europe-west1 URL: https://app-sfc5k4dcaa-ew.a.run.app									
METRICS REVISIONS LOGS TRIGGERS DETAILS YAML PERMISSIONS									
Logs Showing 150 messages Default Filter logs									
2020-12-17T14:14:20.660835Z OpenBLAS WARNING - could not determine the L2 cache size on this system, assuming 256k									
2020-12-17T14:14:22.539864Z OpenBLAS WARNING - could not determine the L2 cache size on this system, assuming 256k									
2020-12-17T14:14:25.426027Z GET 200 3.25 KB 5.8 s Chrome 87 https://app-sfc5k4dcaa-ew.a.run.app/products									
2020-12-17T14:29:30.701866Z [2020-12-17 14:29:30 +0000] [1] [INFO] Handling signal: term									
2020-12-17T14:29:31.642730Z [2020-12-17 14:29:31 +0000] [2] [INFO] Worker exiting (pid: 2)									
2020-12-17T14:29:31.944524Z [2020-12-17 14:29:31 +0000] [1] [INFO] Shutting down: Master									
2020-12-17T15:47:01.535641Z [2020-12-17 15:47:01 +0000] [1] [INFO] Starting gunicorn 20.0.4									
2020-12-17T15:47:01.536580Z [2020-12-17 15:47:01 +0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)									
2020-12-17T15:47:01.536646Z [2020-12-17 15:47:01 +0000] [1] [INFO] Using worker: threads									
2020-12-17T15:47:01.593643Z [2020-12-17 15:47:01 +0000] [2] [INFO] Booting worker with pid: 2									
2020-12-17T15:47:01.967546Z OpenBLAS WARNING - could not determine the L2 cache size on this system, assuming 256k									
2020-12-17T15:47:03.904772Z OpenBLAS WARNING - could not determine the L2 cache size on this system, assuming 256k									
2020-12-17T15:47:09.191783Z Memory limit of 256M exceeded with 259M used. Consider increasing the memory limit, see https://cloud.google.com/run/docs/configuring/memory-limits									
2020-12-17T15:47:09.194565Z [2020-12-17 15:47:09 +0000] [1] [INFO] Handling signal: term									

Рисунок 3.15 – Журнал повідомлень сервісу Goggle cloud run

Для тестування компоненту REST API серверу використовувався інструмент Postman. Postman – це інструмент для створення запитів до API, який дозволяє протестувати роботу API, виключаючи можливі помилки створення запиту на клієнті, чи обробки результату цього запиту [20].

Тестування клієнтського додатку відбувалось в мануальному режимі. Під час тестування було виявлено та виправлено ряд помилок, а також розроблено обробку помилкових дій користувача.

Наприклад, сторінка створення в облікового запису містить поле вводу електронної пошти. Користувач може помилково ввести пошту, що вже асоційована з іншим користувачем, або намагатиметься повторно створити обліковий запис. В такому випадку, система реагує на помилкові дані та надсилає користувачеві попередження про те, що така адреса вже використовується. Приклад попередження користувачеві зображено на рисунку 3.16.

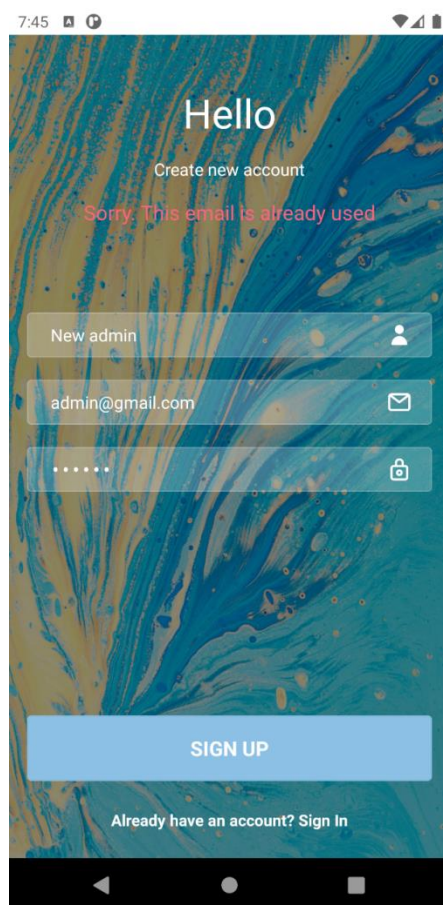


Рисунок 3.16 – Попередження про некоректну електронну адресу

Аналогічним чином оброблюються і інші помилки вводу даних користувачем.

Оскільки клієнтський додаток може використовуватись на різних пристроях, важливо провести функціональне тестування та тестування користувацького інтерфейсу на різних пристроях. Наприклад, сторінка відображення матеріалів підлаштовується під розмір екрану. Відображення списку на великих екранах відбувається за іншим інтерфейсом. На рисунку 3.17 зображено інтерфейс сторінки переліку матеріалів у веб-браузері Google Chrome на персональному комп'ютері на базі ОС Windows з розширенням екрану 1366 пікселів.

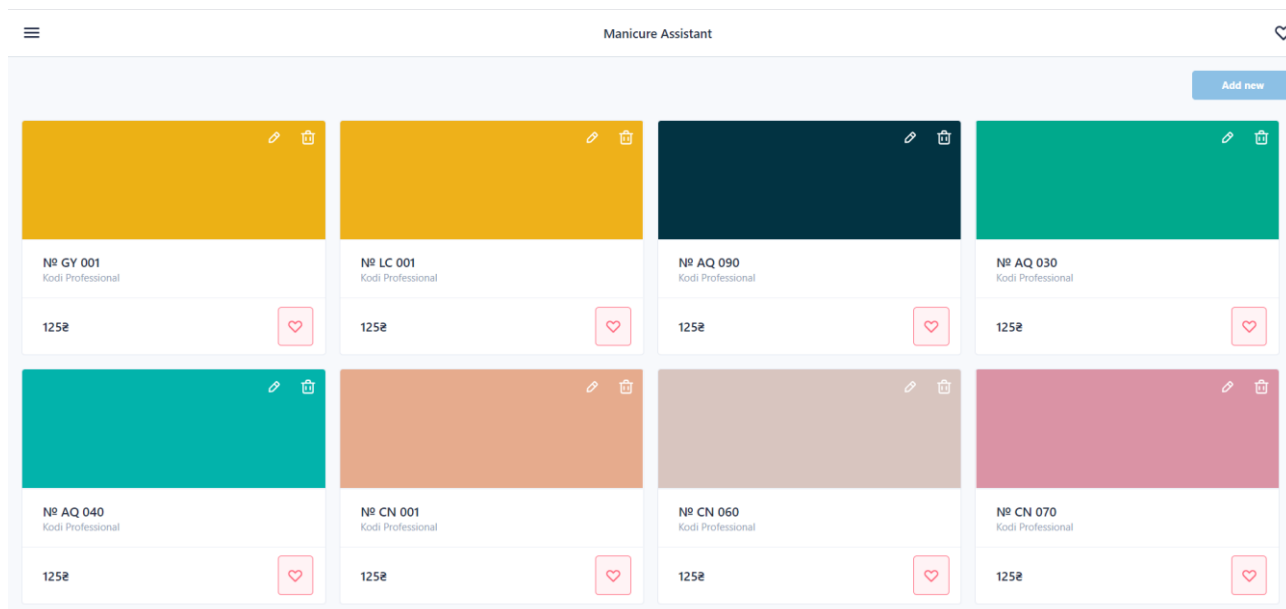


Рисунок 3.17 – Інтерфейс сторінки переліку матеріалів на екрані розміром 1366 пікселів

### Висновки до третього розділу

У третьому розділі кваліфікаційної магістерської роботи розглянуто порядок розгортання та налаштування системи 2D дизайну нігтів. Описано інструменти, що використовувались для розгортання системи та передумови, необхідні для її налаштування.

Важливою частиною системи є користувацький додаток, інтерфейс якого було детально описано у цьому розділі. Наведено опис сторінок клієнтського



додатку та реалізації його інтерфейсу. До детального опису додано зображення клієнтського додатку, для повного розуміння зовнішнього вигляду системи.

Окрім цього, було проведено тестування клієнтської та серверної частини додатку. В результаті тестування виявлено та виправлено ряд помилок в роботі системи, сформовано підходи до обробки некоректної інформації користувача та зміни інтерфейсу в залежності від середовища виконання додатку.

## ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було спроектовано та розроблено систему 2D дизайну нігтів, що складається з клієнтського додатку для мобільних пристроїв та веб-браузерів та серверного додатку.

В контексті аналізу напрямків використання інформаційних технологій для 2D дизайну нігтів було проведено огляд галузі манікюрних салонів. На базі інформації про процеси діяльності манікюрних салонів було сформовано основні вимоги до системи та встановлено задачу на проектування та розробку додатку.

Аналіз аналогів програмного продукту виявив, що наразі немає програмних засобів, що повністю вирішували б задачі, встановлені до магістерської роботи.

Враховуючи вимоги до системи було обрано клієнт-серверну архітектуру. Система на базі цієї архітектури дозволяє вирішувати такі завдання як надання зручного інтерфейсу користувачам, підтримка роботи додатку з багатьма користувачами, збереження загальних даних для манікюрного салону та інші. Більшість сучасних користувачів надають перевагу мобільним телефонам, а переважна більшість клієнтів манікюрних салонів мають мобільні телефони. Проте, адміністратор салону краси може використовувати робочий комп'ютер з доступом в інтернет. Тому важливо було створити додаток, що працюватиме на різних пристроях як звичайний додаток для встановлення та як веб-додаток для відкриття в браузерах. Після аналізу можливих інструментів та технологій було обрано платформу React Native.

Під час визначення варіантів використання системи було встановлено наявність користувачів з трьома рівнями доступу до додатку та перелік функціональних можливостей клієнтського додатку для різних користувачів.

Розроблено структура бази даних, сформовано перелік сутностей системи, таблиць та зв'язків між ними.

Для задачі розпізнавання зображення та пошуку матеріалів за кольором було визначено алгоритми роботи та описано ключові особливості. Маючи визначені алгоритми роботи системи, було реалізовано функціональні можливості клієнтського та серверного додатків. Наведено фрагменти коду для ключових частин системи.

Для опису інтерфейсу та порядку роботи з системою було перераховано інструменти, що використовувались для розгортання системи та передумови, необхідні для її налаштування. Важливою частиною системи є користувацький додаток, інтерфейс якого детально описано у пояснювальній записці. Наведено опис сторінок клієнтського додатку та реалізації його інтерфейсу. До детального опису додано зображення клієнтського додатку, для повного розуміння зовнішнього вигляду системи.

Окрім цього, було проведено тестування клієнтської та серверної частини додатку. В результаті тестування виявлено та виправлено ряд помилок в роботі системи, сформовано підходи до обробки некоректної інформації користувача та зміни інтерфейсу в залежності від середовища виконання додатку.

Реалізована система готова до використання та може бути впроваджена для роботи манікюрних салонів та покращення якості обслуговування клієнтів цих салонів.

В майбутньому, заплановано ряд нових функціональних можливостей та способів покращення системи.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android version history [Електронний ресурс] – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/Android\\_version\\_history](https://en.wikipedia.org/wiki/Android_version_history) [Accessed 5 Nov. 2020]
2. BASH Programming - Introduction HOW-TO: Variables [Електронний ресурс] – Режим доступу до ресурсу: <https://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-5.html> [Accessed 25 Nov. 2020]
3. CORS Enabled [Електронний ресурс] – Режим доступу до ресурсу: [https://www.w3.org/wiki/CORS\\_Enabled](https://www.w3.org/wiki/CORS_Enabled) [Accessed 15 Nov. 2020]
4. Cloud Run documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://cloud.google.com/run/docs> [Accessed 2 Dec. 2020]
5. Color difference [Електронний ресурс] – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/Color\\_difference](https://en.wikipedia.org/wiki/Color_difference) [Accessed 23 Oct. 2020]
6. Datatypes In SQLite Version 3 [Електронний ресурс] – Режим доступу до ресурсу: <https://sqlite.org/datatype3.html> [Accessed 15 Nov. 2020]
7. Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Docker> [Accessed 2 Dec. 2020]
8. Fetch [Електронний ресурс] – Режим доступу до ресурсу: <https://javascript.info/fetch> [Accessed 15 Sep. 2020]
9. Firebase CLI reference [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/cli>. [Accessed 1 Dec. 2020]
10. Flask [Електронний ресурс] – Режим доступу до ресурсу: <https://palletsprojects.com/p/flask/> [Accessed 16 Nov. 2020]
11. Get started with Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: <https://cloud.google.com/docs> [Accessed 2 Dec. 2020]
12. Gunicorn - Python WSGI HTTP Server for UNIX [Електронний ресурс] – Режим доступу до ресурсу: <https://gunicorn.org/> [Accessed 16 Nov. 2020]

13. Introduction – React Native [Электронный ресурс] – Режим доступа до ресурсу: <https://reactnative.dev/docs/getting-started> [Accessed 15 Oct. 2020]
14. Promise [Электронный ресурс] – Режим доступа до ресурсу: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [Accessed 15 Sep. 2020]
15. Python [Электронный ресурс] – Режим доступа до ресурсу: <https://uk.wikipedia.org/wiki/Python> [Accessed 11 Nov. 2020]
16. REST [Электронный ресурс] – Режим доступа до ресурсу: <https://uk.wikipedia.org/wiki/REST> [Accessed 16 Nov. 2020]
17. React Native [Электронный ресурс] – Режим доступа до ресурсу: [https://en.wikipedia.org/wiki/React\\_Native](https://en.wikipedia.org/wiki/React_Native) [Accessed 14 Dec. 2020]
18. SQLite [Электронный ресурс] – Режим доступа до ресурсу: <https://en.wikipedia.org/wiki/SQLite> [Accessed 11 Nov. 2020]
19. Singleton in Python [Электронный ресурс] – Режим доступа до ресурсу: <https://refactoring.guru/design-patterns/singleton/python/example> [Accessed 3 Dec. 2020]
20. The Collaboration Platform for API Development [Электронный ресурс] – Режим доступа до ресурсу: <https://www.postman.com/> [Accessed 13 Dec. 2020]
21. The Great Promo Video for the mobile app WANNA Nails [Электронный ресурс] – Режим доступа до ресурсу: [https://medium.com/slонmedia/the-great-promo-video-for-the-mobile-app-wanna-nails-5b8b647fb546](https://medium.com/slونmedia/the-great-promo-video-for-the-mobile-app-wanna-nails-5b8b647fb546) [Accessed 16 Nov. 2020]
22. The New React Native Architecture Explained [Электронный ресурс] – Режим доступа до ресурсу: <https://formidable.com/blog/2019/react-codegen-part-1/> [Accessed 18 Nov. 2020]
23. TypeScript [Электронный ресурс] – Режим доступа до ресурсу: <https://uk.wikipedia.org/wiki/TypeScript> [Accessed 22 Sep. 2020]

24. Understanding K-means Clustering in Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1> [Accessed 11 Oct. 2020]
25. Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Unified\\_Modeling\\_Language](https://uk.wikipedia.org/wiki/Unified_Modeling_Language) [Accessed 14 Sep. 2020]
26. Update for Customers Using PhoneGap and PhoneGap Build [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.phonegap.com/update-for-customers-using-phonegap-and-phonegap-build-cc701c77502c> [Accessed 2 Dec. 2020]
27. Uploading Files [Електронний ресурс] – Режим доступу до ресурсу: <https://flask.palletsprojects.com/en/1.1.x/patterns/fileuploads/> [Accessed 24 Nov. 2020]
28. Use case diagram [Електронний ресурс] – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/Use\\_case\\_diagram](https://en.wikipedia.org/wiki/Use_case_diagram) [Accessed 14 Sep. 2020]
29. What Is SQLite? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sqlite.org/index.html> [Accessed 2 Nov. 2020]
30. What Is Software Testing | Everything You Should Know [Електронний ресурс] – Режим доступу до ресурсу: <https://www.softwaretestingmaterial.com/software-testing/> [Accessed 2 Dec. 2020]
31. What is a Container? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.docker.com/resources/what-container> [Accessed 2 Dec. 2020]
32. Yclients [Електронний ресурс] – Режим доступу до ресурсу: <https://www.yclients.com/en> [Accessed 16 Nov. 2020]
33. API-довідник хуків [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.reactjs.org/docs/hooks-reference.html> [Accessed 19 Oct. 2020]

34. Віддаль між двома точками [Електронний ресурс] – Режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Віддаль\\_між\\_двома\\_точками](https://uk.wikipedia.org/wiki/Віддаль_між_двома_точками) [Accessed 14 Nov. 2020]
35. Діаграма класів [Електронний ресурс] – Режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Діаграма\\_класів](https://uk.wikipedia.org/wiki/Діаграма_класів) [Accessed 14 Sep. 2020]
36. Евклідов простір [Електронний ресурс] – Режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Евклідов\\_простір](https://uk.wikipedia.org/wiki/Евклідов_простір) [Accessed 14 Nov. 2020]
37. Загальна характеристика задач кластерного аналізу [Електронний ресурс] – Режим доступу до ресурсу: <https://sites.google.com/site/ne4itkalogika/necitka-klasterizacia/zadaci-klasternogo-analizu> [Accessed 11 Oct. 2020]
38. Кластерний аналіз [Електронний ресурс] – Режим доступу до ресурсу: [https://pidru4niki.com/11800912/ekonomika/klasterniy\\_analiz](https://pidru4niki.com/11800912/ekonomika/klasterniy_analiz) [Accessed 11 Oct. 2020]
39. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Клієнт-серверна\\_архітектура](https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура) [Accessed 8 Nov. 2020]
40. Типи мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: <https://smile-ukraine.com/ua/mobile-apps/mobile-apps-types> [Accessed 29 Nov. 2020]

# ДОДАТКИ



Акт впровадження результатів кваліфікаційної магістерської роботи.

«ЗАТВЕРДЖУЮ»

Директор Романюк Лідія  
Костянтинівна

АКТ

Про впровадження результатів кваліфікаційної магістерської роботи студента групи ІПЗм-19-1 Державного університету «Житомирська політехніка» Бойко Таїси Олегівни.

ФОП Романюк Лідія Костянтинівна склала дійсний акт про те, що результати кваліфікаційної магістерської роботи студентки Бойко Таїси Олегівни зокрема:

- система бази даних лаків у салоні;
- мобільний додаток для управління базою даних;
- мобільний додаток для пошуку лаку за зображенням;

були використані для щоденної підприємницької діяльності з надання манікюрних послуг клієнтам.

Впровадження результатів роботи дозволило зменшити втрати часу на формування фізичної картотеки доступних матеріалів а також зробить більш ефективним процес пошуку бажаного лаку для клієнта.

ФОП Романюк Лідія Костянтинівна

15.12.2020 р.  
Затверджую *[підпис]*

Витяг з ЄДР юридичних осіб, фізичних осіб-підприємців та громадських формувань про діяльність ФОП Романюк Лідії Костянтинівни.



**ВИТЯГ**  
**з Єдиного державного реєстру юридичних осіб, фізичних осіб-підприємців та громадських формувань**

Відповідно до статті 11 Закону України "Про державну реєстрацію юридичних осіб, фізичних осіб-підприємців та громадських формувань" на запит: **БОЙКО ТАІСА ОЛЕГІВНА** від **15.12.2020** за кодом **346680836862** станом на **15.12.2020 11:37:11** відповідно до наступних критеріїв пошуку:

**Прізвище, ім'я та по батькові фізичної особи-підприємця:** РОМАНЮК ЛІДІЯ КОСТЯНТИНІВНА

**До документу внести:**

Країна громадянства

Місцезнаходження (місце проживання або інша адреса, за якою здійснюється зв'язок з фізичною особою – підприємцем)

Види діяльності

Дата та номер запису в Єдиному державному реєстрі

Інформація для здійснення зв'язку з фізичною особою – підприємцем

Відомості про осіб, які можуть вчиняти дії від імені фізичної особи – підприємця, у тому числі підписувати договори, тощо

надається інформація з Єдиного державного реєстру юридичних осіб, фізичних осіб-підприємців та громадських формувань (ЄДР) у кількості **1** записів:

**Запис 1**

**Прізвище, ім'я, по батькові фізичної особи-підприємця:**

РОМАНЮК ЛІДІЯ КОСТЯНТИНІВНА

**Актуальний стан на фактичну дату та час формування:**

zareestrovano

**Країна громадянства фізичної особи-підприємця:**

Україна

**Місцезнаходження фізичної особи-підприємця:**

Україна, 12001, Житомирська обл., Пулинський р-н, селище міського типу Пулини, ВУЛИЦЯ ПОКРОВСЬКА, будинок 43

***Види економічної діяльності:***

96.02 Надання послуг перукарнями та салонами краси (основний)

***Дата та номер запису в Єдиному державному реєстрі юридичних осіб, фізичних осіб-підприємців та громадських формувань:***

Дата запису: 19.08.2019 Номер запису: 23020000000001427

***Прізвища, імена, по батькові осіб, які мають право вчиняти юридичні дії від імені фізичної особи - підприємця без довіреності, у тому числі підписувати договори, дані про наявність обмежень щодо представництва від імені фізичної особи-підприємця:***

Відомості відсутні

***Інформація для здійснення зв'язку:***

+380967727144, lidija9686@icloud.com

***Дата та час формування витягу:***

15.12.2020 11:40:24

Єдиний державний реєстр юридичних осіб, фізичних осіб-підприємців та громадських формувань знаходиться у стані формування. Інформація про юридичних осіб, фізичних осіб-підприємців та громадських формувань та зареєстрованих до 01.07.2004 та не включених до Єдиного державного реєстру юридичних осіб, фізичних осіб-підприємців та громадських формувань отримується в органі виконавчої влади, в якому проводилась державна реєстрація.

## Лістинг обраних класів та скриптів клієнтського додатку

## App.tsx

```

import React from 'react';
import * as eva from '@eva-design/eva';
import { ApplicationProvider, IconRegistry } from '@ui-kitten/components';
import { SafeAreaProvider } from 'react-native-safe-area-context';
import { StatusBar } from 'expo-status-bar';
import { EvaIconsPack } from '@ui-kitten/eva-icons';
import { AppNavigator } from '../src/app.navigator';
import { initFirebaseApp } from '../src/services/FirebaseApp';

import { default as theme } from '../theme.json';

export default () => {
  initFirebaseApp();
  return (
    <>
      <IconRegistry icons={EvaIconsPack} />
      <ApplicationProvider {...eva} theme={{ ...eva.light, ...theme }}>
        <SafeAreaProvider>
          <StatusBar />
          <AppNavigator />
        </SafeAreaProvider>
      </ApplicationProvider>
    </>
  );
};

```

## app.navigator.tsx

```

import React from 'react';
import { createDrawerNavigator } from '@react-navigation/drawer';
import { DefaultTheme, NavigationContainer } from '@react-
navigation/native';
import { SafeAreaView } from 'react-native-safe-area-context';
import MainDrawer from '../menus/drawer-main.component';
import HomeScreen from '../screens/home';
import ProductsScreen from '../screens/product/list';
import BrandsScreen from '../screens/brands/list';
import EditBrandScreen from '../screens/brands/edit';
import SignInScreen from '../screens/sign-in';
import SignUpScreen from '../screens/sign-up';

const Drawer = createDrawerNavigator();

/*
 * Navigation theming: https://reactnavigation.org/docs/en/next/themes.html
 */
const navigatorTheme = {

```

```

    ...DefaultTheme,
    colors: {
      ...DefaultTheme.colors,
      background: 'transparent',
    },
  };

export const AppNavigator = (): React.ReactElement => {
  return (
    <NavigationContainer theme={navigatorTheme}>
      <SafeAreaView style={{ flex: 1 }} edges={['top']}>
        <Drawer.Navigator
          screenOptions={{ gestureEnabled: true }}
          drawerContent={(props) => <MainDrawer {...props} />}
        >
          <Drawer.Screen name='Home' component={HomeScreen} />
          <Drawer.Screen name='Products' component={ProductsScreen} />
          <Drawer.Screen name='Brands' component={BrandsScreen} />
          <Drawer.Screen name='SignIn' component={SignInScreen} />
          <Drawer.Screen name='SignUp' component={SignUpScreen} />
          <Drawer.Screen name='EditBrand' component={EditBrandScreen} />
        </Drawer.Navigator>
      </SafeAreaView>
    </NavigationContainer>
  );
};

```

### brands/list.tsx

```

import React, { useState, useEffect } from 'react';
import { TouchableOpacity, View } from 'react-native';
import {
  Button,
  Divider,
  Layout,
  List,
  Text,
  StyleService,
  useStyleSheet,
} from '@ui-kitten/components';
import { Brand } from '../../types/Entities';
import TopNavigationMain from '../../menus/top-menu-main.component';
import { APIProvider } from '../../services/APIProvider';
import { AuthService } from '../../services/AuthService';
import { EditIcon, DeleteIcon } from '../../components/icons';
import { User } from '../../types/Entities';

export default ({ navigation, title, data }): React.ReactElement => {
  const styles = useStyleSheet(themedStyles);
  const [dataState, setDataState] = useState(data);
  const [currentUser, setCurrentUser] =
    useState<User>(AuthService.currentUser);

  useEffect(() => {
    (async function () {

```

```

        if (!dataState) {
            setDataState(await APIProvider.getBrands());
        }
        AuthService.subscribe(() => {
            setCurrentUser(AuthService.currentUser);
        });
    })();
}, []);

const renderListItem = ({ item }: { item: Brand }): React.ReactElement =>
(
    <TouchableOpacity
        onPress={async () => {
            navigation.navigate('Products', {
                data: await APIProvider.getProducts(item.id),
                title: item.name,
            });
        }}
        style={styles.listItem}
    >
        <View>
            <Text category='s1'>{item.name}</Text>
            <Text category='c2'>Price {item.price}</Text>
        </View>
        { currentUser?.isAdmin &&
            <View style={styles.flexRow}>
                <Button
                    status='info'
                    appearance='outline'
                    accessoryLeft={EditIcon}
                    onPress={() => navigation.navigate('EditBrand', {brand: item})}
                    style={styles.actionButton}
                ></Button>
                <Button
                    status='danger'
                    appearance='outline'
                    accessoryLeft={DeleteIcon}
                    style={styles.actionButton}
                ></Button>
            </View>
        }
    </TouchableOpacity>
);

return (
    <Layout style={{ flex: 1 }}>
        <TopNavigationMain navigation={navigation} title='Brands' />
        <Layout style={styles.container}>
            { currentUser.isAdmin &&
                <Layout style={styles.flexEnd}>
                    <Button style={styles.addButton} size='small' status='info'>Add
new</Button>
                </Layout>
            }
        <List

```

```

        contentContainerStyle={styles.list}
        data={dataState}
        renderItem={renderListItem}
        ItemSeparatorComponent={Divider}
      />
    </Layout>
  </Layout>
);
};

const themedStyles = StyleService.create({
  container: {
    flex: 1,
    backgroundColor: 'background-basic-color-2',
  },
  list: {
    paddingHorizontal: 8,
    paddingVertical: 16,
  },
  flexRow: {
    flexDirection: 'row',
  },
  listItem: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    padding: 10,
  },
  actionButton: {
    width: 40,
    height: 40,
    marginLeft: 10,
  },
  addButton: {
    marginTop: 16,
    width: 120,
  },
  flexEnd: {
    display: 'flex',
    flexDirection: 'row',
    justifyContent: 'flex-end',
    marginRight: 16,
    backgroundColor: 'transparent',
  }
});

```

Лістинг обраних класів та скриптів клієнтського додатку

color\_compare.py

```

from db import db_requests
import numpy as np
from .Color import Color, get_color_difference
from utils.color_sort import sort_by_color_dif

```

```

def find_closest_color(target_color):
    data = db_requests.get_products()
    # target_product = db_requests.get_product_by_id(target_id)

    add_color_dif(data, Color(target_color))
    sort_by_color_dif(data, 0, len(data)-1)
    return data[:6]

def add_color_dif(data, target_color):
    for item in data:
        item['dif'] = get_color_difference(target_color, item)

```

### Color.py

```

import re
import math

class Color:
    def __init__(self, color):
        color_components = re.search("rgb\\((\\d{1,3}), (\\d{1,3}), (\\d{1,3})\\)", color)
        self.red = int(color_components.group(1))
        self.green = int(color_components.group(2))
        self.blue = int(color_components.group(3))

    def get_color_difference(target_color, item):
        current_color = Color(item["color"])
        return math.sqrt(pow((target_color.red - current_color.red), 2)
            + pow((target_color.green - current_color.green), 2)
            + pow((target_color.blue - current_color.blue), 2))

```

### Database.py

```

import sqlite3
from .scripts import *

DATABASE_PATH = r"/app/db/database.db"

class Singleton(type):
    _instances = {}
    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args,
**kwargs)
        return cls._instances[cls]

class Database(metaclass=Singleton):
    connection = None

    def connect(self):
        if self.connection is None:
            self.connection = sqlite3.connect(DATABASE_PATH,
check_same_thread=False)
            self.cursor = self.connection.cursor()
            self.__create_tables()

```



```

        return self.cursor

    def __create_tables(self):
        self.cursor.execute(sql_create_brands_table)
        self.cursor.execute(sql_create_products_table)
        self.cursor.execute(sql_create_users_table)
        self.cursor.execute(sql_create_fav_products_table)
        self.cursor.execute(sql_create_fav_price_categories_table)

```

## populate.py

```

def populate_brands(records):
    brands = list()
    for row in records:
        brands.append({"id": row[0], "name": row[1], "price": row[2]})
    return brands

def populate_products(records):
    products = list()
    for row in records:
        products.append(populate_product(row))
    return products

def populate_product(row):
    return {"id": row[0], "name": row[1], "color": row[2], "brand": row[3],
"brandId": row[4], "price": row[5]}

```