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 ІПЗ,

\_\_\_\_\_\_\_\_\_\_\_\_ 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва кафедри)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_ р.

**ЗАВДАННЯ**

на випускну 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 \_\_\_\_\_\_\_. Пояснювальна записка до випускної роботи містить 00 сторінок, 00 ілюстрацій та 00 таблиць.

Метою кваліфікаційної роботи

Ключові слова: .

ABSTRACT

Master's graduate work

KEYWORDS:

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7](#_Toc59044032)

[ВСТУП 8](#_Toc59044033)

[Розділ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 2D ДИЗАЙНУ НІГТІВ 9](#_Toc59044034)

[1.1. Постановка задачі 9](#_Toc59044035)

[1.2. Аналіз аналогів програмного продукту 13](#_Toc59044036)

[1.3. Вибір архітектури системи 2D дизайну нігтів 16](#_Toc59044037)

[1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 19](#_Toc59044038)

[Висновки до першого розділу 23](#_Toc59044039)

[Розділ 2. ПРОЕКТУВАННЯ СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ 25](#_Toc59044040)

[2.1. Визначення варіантів використання системи та об’єктно-орієнтованої структури системи 25](#_Toc59044045)

[2.2. Розробка бази даних 30](#_Toc59044046)

[2.3. Проектування та реалізація алгоритмів роботи системи 33](#_Toc59044047)

[2.4. Реалізація функціоналу системи 2D дизайну нігтів 37](#_Toc59044048)

[Висновки до другого розділу 44](#_Toc59044049)

[Розділ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ 45](#_Toc59044050)

[3.1. Порядок встановлення та налаштування системи. 45](#_Toc59044051)

[Висновки до третього розділу 45](#_Toc59044052)

[ВИСНОВКИ 46](#_Toc59044053)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 47](#_Toc59044054)

[ДОДАТКИ 49](#_Toc59044055)

[ДОДАТОК А 50](#_Toc59044056)

[ДОДАТОК Б 51](#_Toc59044057)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

ОС – Операційна система

JVM – Java Virtual Machine

SDK – Software Development Kit

JSON – JavaScript Object Notation

IDE – Integrated development environment

DAL – Data Access Layer

UI – User Interface

MVVM – Model View Viewmodel

DOM – Document Object Model

CRUD – Create Read Update Delete

ВСТУП

*Актуальність теми.*

*Метою*

*Об’єктом дослідження*

*Предметом дослідження*

Тезы

# АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 2D ДИЗАЙНУ НІГТІВ

## 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 цінових 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.

1. 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 завантажені 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.

## 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 реалістичне. Серед додаткових функцій додатку є наступні:

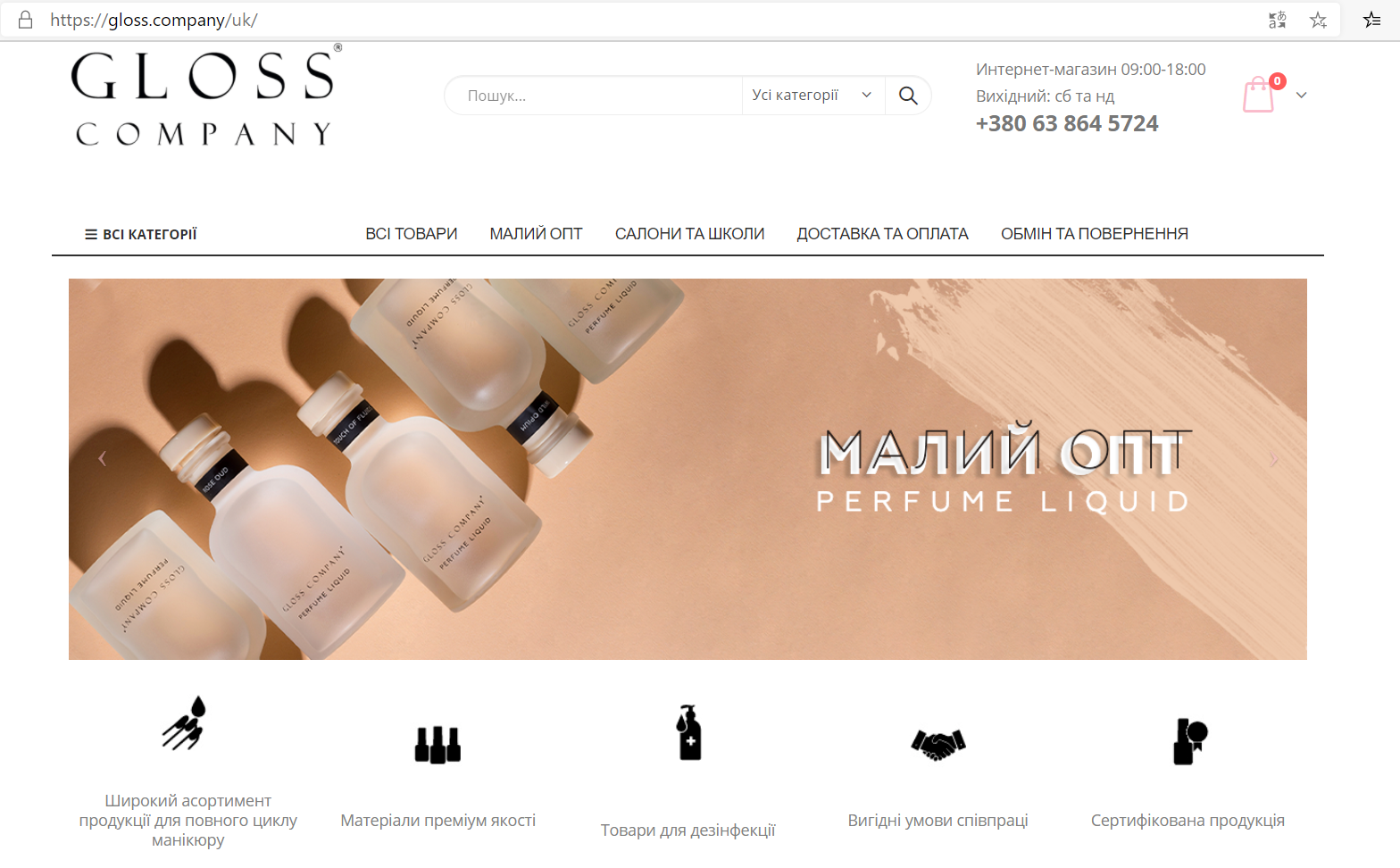
* порівняння різних кольорів лаку на руці;
* створення колажу фотографій з різними варіантами лаків;
* визначення різних кольорів для окремих нігтів.

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 Company»



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.

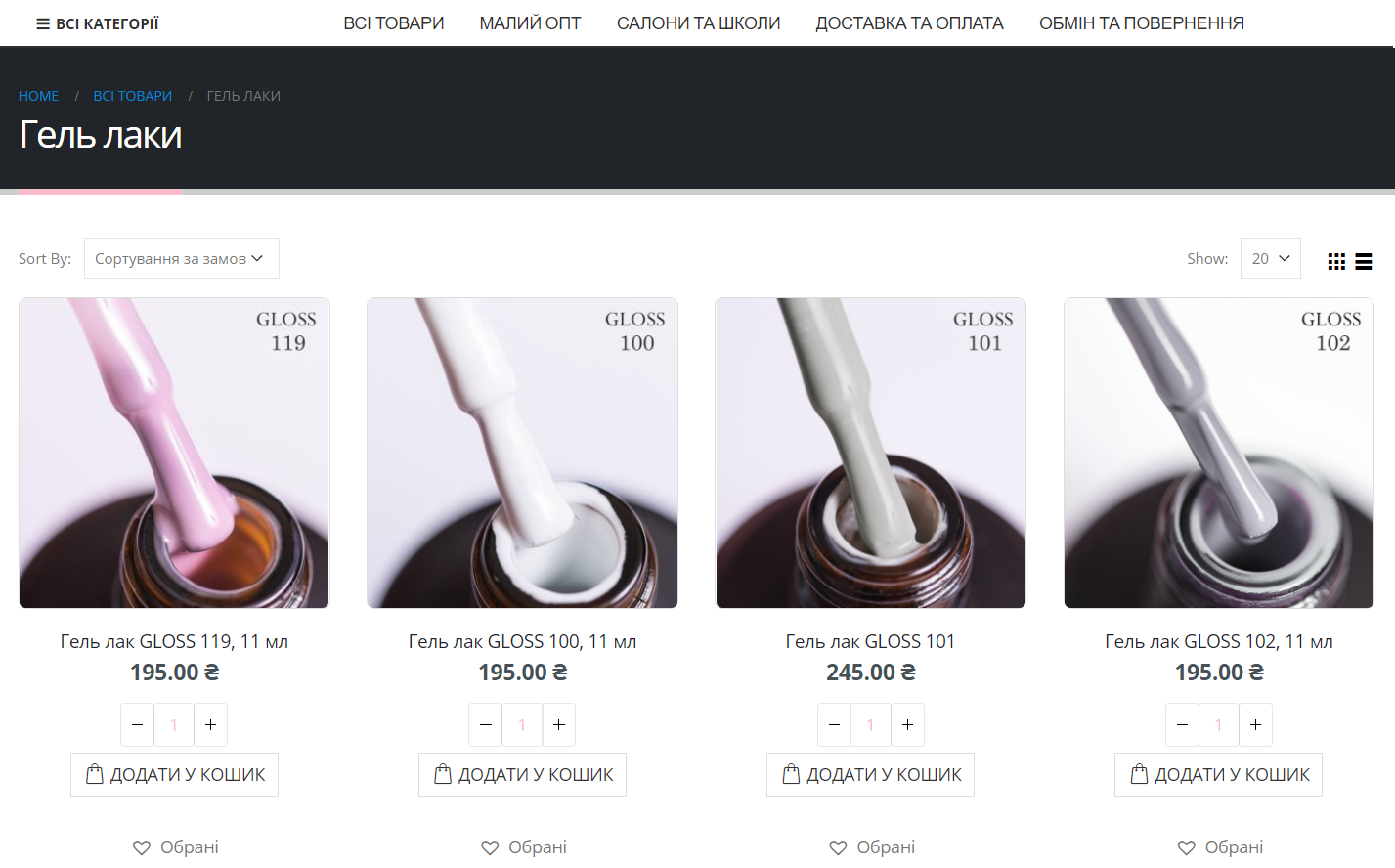


Рисунок 1.3 –Сторінка категорії товарів додатку «GLOSS Company»

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.

## 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 [LINK]. 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.

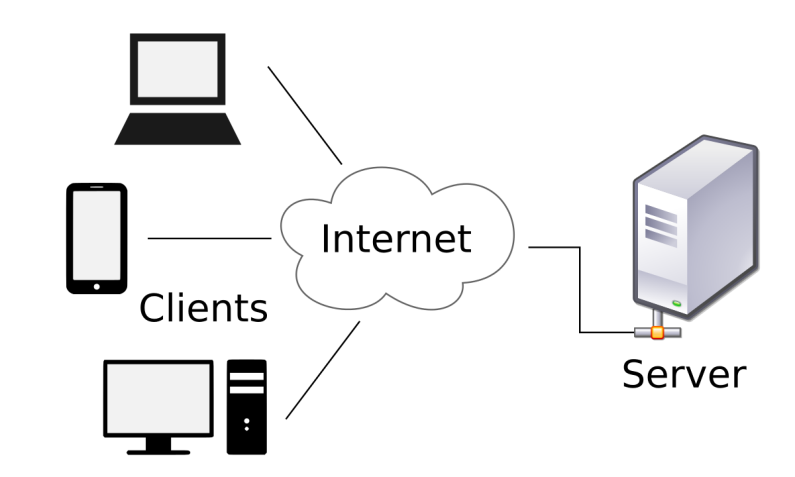


Рисунок 1.4 - Схема роботи клієнт-серверної архітектури

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.

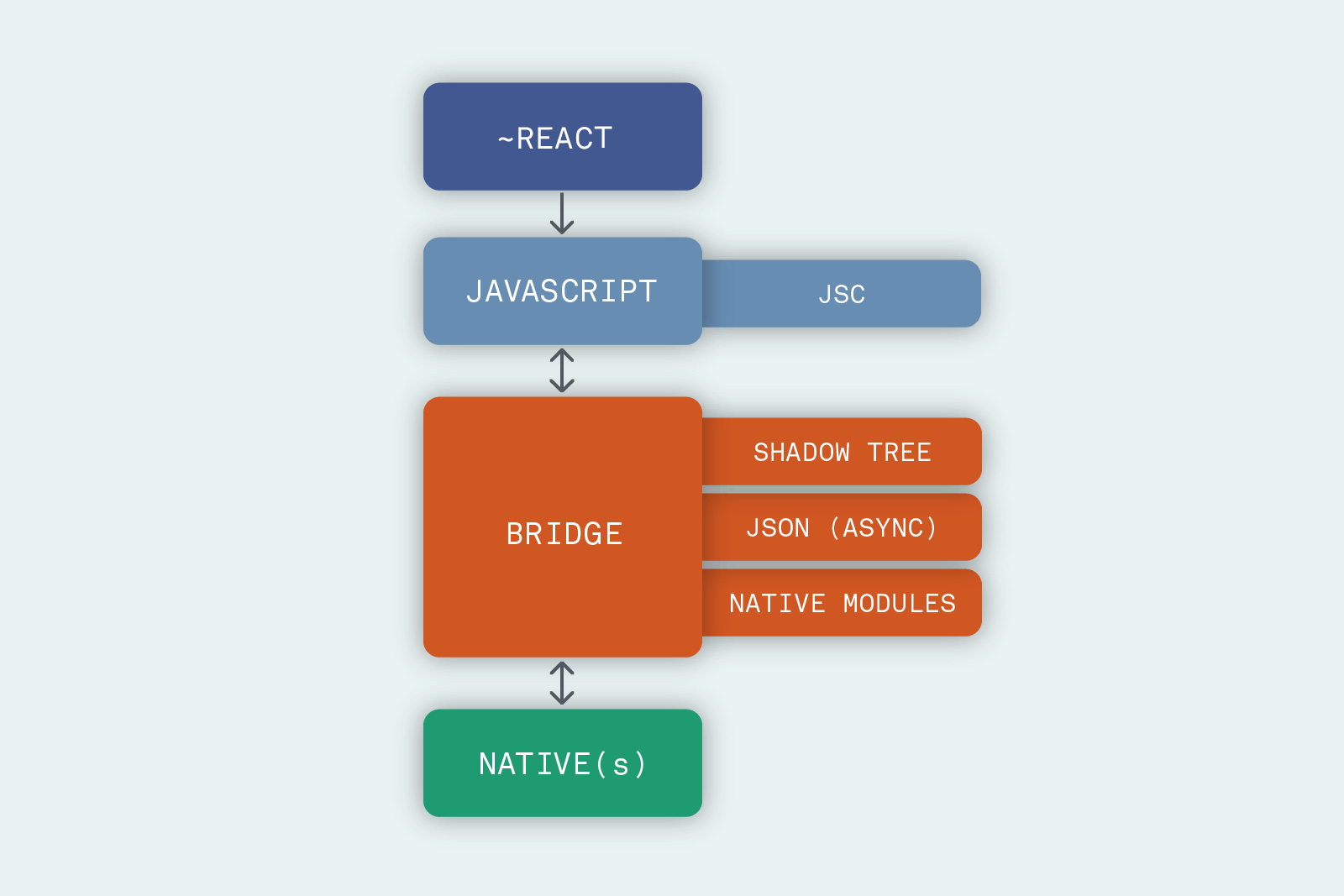


Рисунок 1.5 - Архітектура системи React Native

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 (ОС 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 [LINK].

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.

## 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 можливостю 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. 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, Mozila 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 [LINK]:

* 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 [LINK].

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

Серед існуючи 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 [LINK]. 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 [LINK]. 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 [LINK].

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) [LINK]. 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 [LINK].

## Висновки до першого розділу

Дослідження та аналіз предметної області дозволив сформувати 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 вимоги 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.

# [ПРОЕКТУВАННЯ](file:///C:\Users\Anna\AppData\Roaming\Skype\бакалаврат\fragment_bakalavr.docx#_Toc212894718) СИСТЕМИ 2D ДИЗАЙНУ НІГТІВ



## [Визначення](file:///C:\Users\Anna\AppData\Roaming\Skype\бакалаврат\fragment_bakalavr.docx#_Toc212894719) 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.

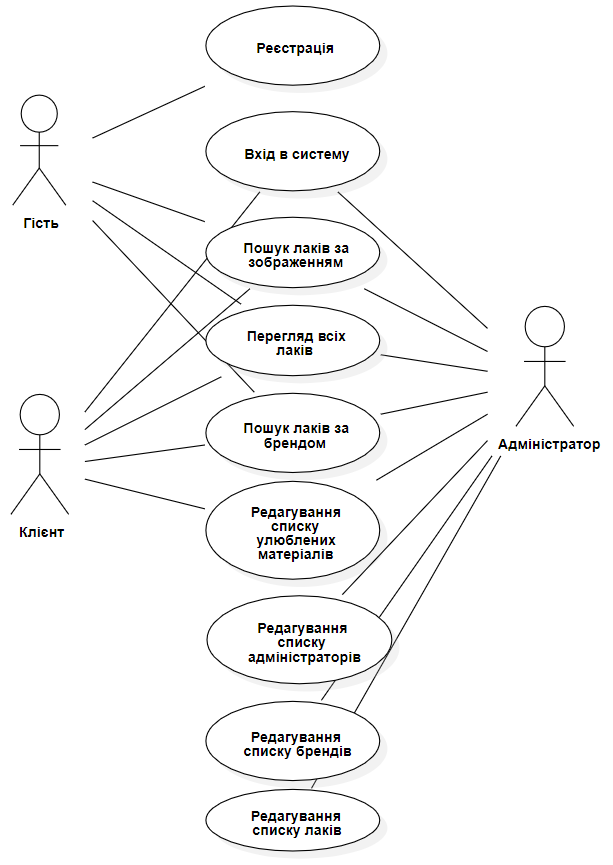


Рисунок 2.1 - Діаграма варіантів використання системи

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.

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.

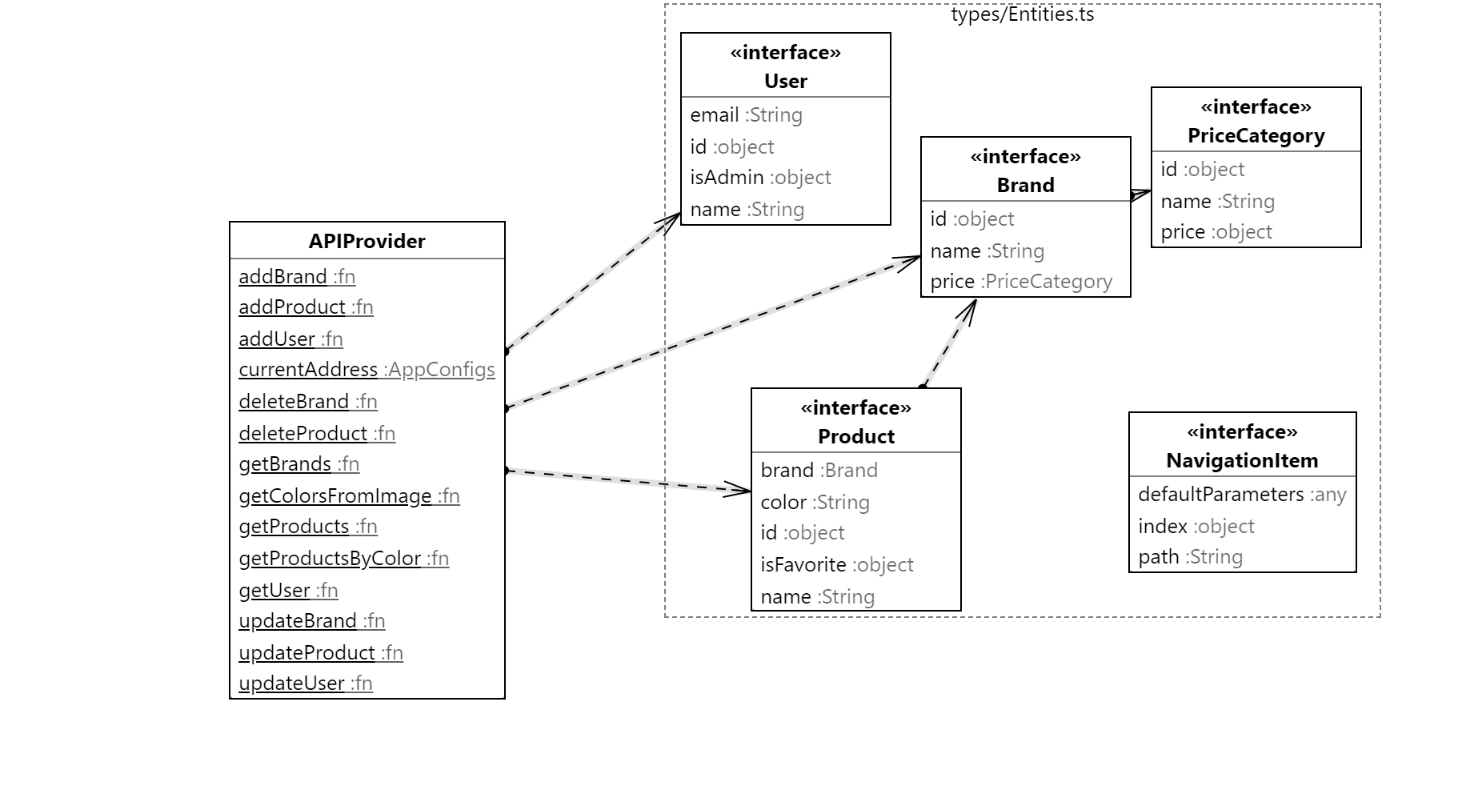


Рисунок 2.2 - Діаграма класів клієнтського додатку

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.

## [Розробка бази](file:///C:\Users\Anna\AppData\Roaming\Skype\бакалаврат\fragment_bakalavr.docx#_Toc212894723) 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 |
| FavoriteProduscts | 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 | - | Електрона 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 |

Таблиця 2.6

Структура таблиці FavoriteProduscts

|  |  |  |  |
| --- | --- | --- | --- |
| 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.

\*\* - 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.

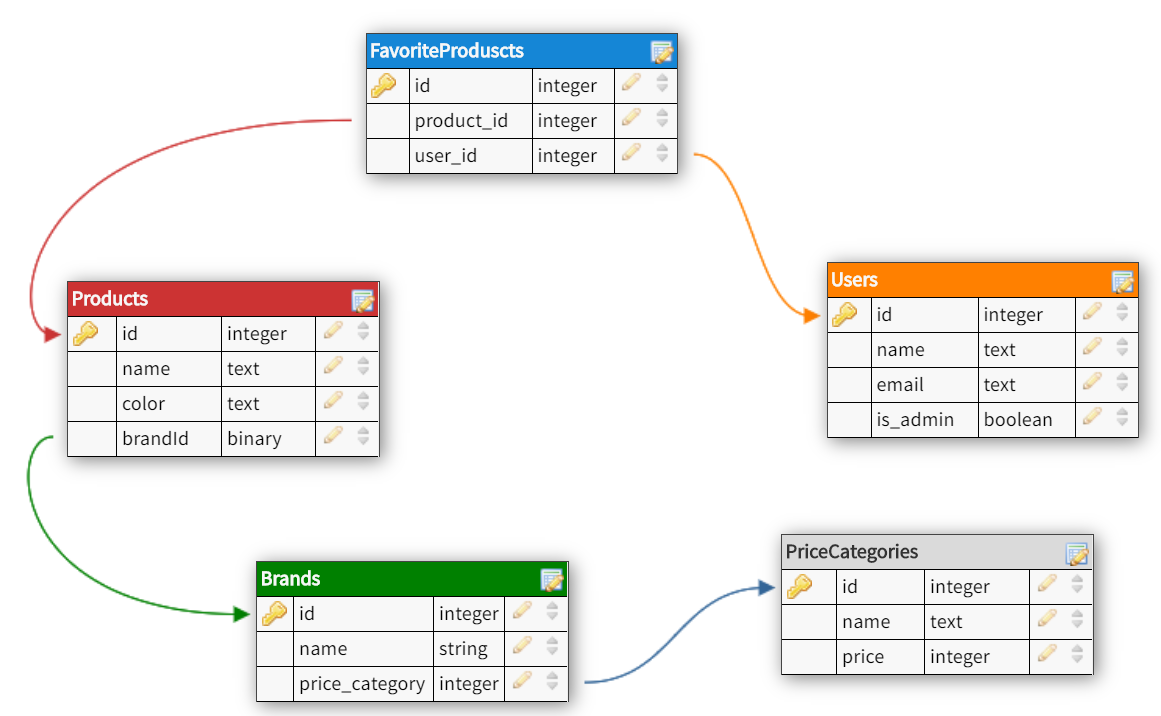


Рисунок 2.3 – Схема зв’язку таблиць бази даних

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

## Проектування та реалізація алгоритмів роботи системи

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

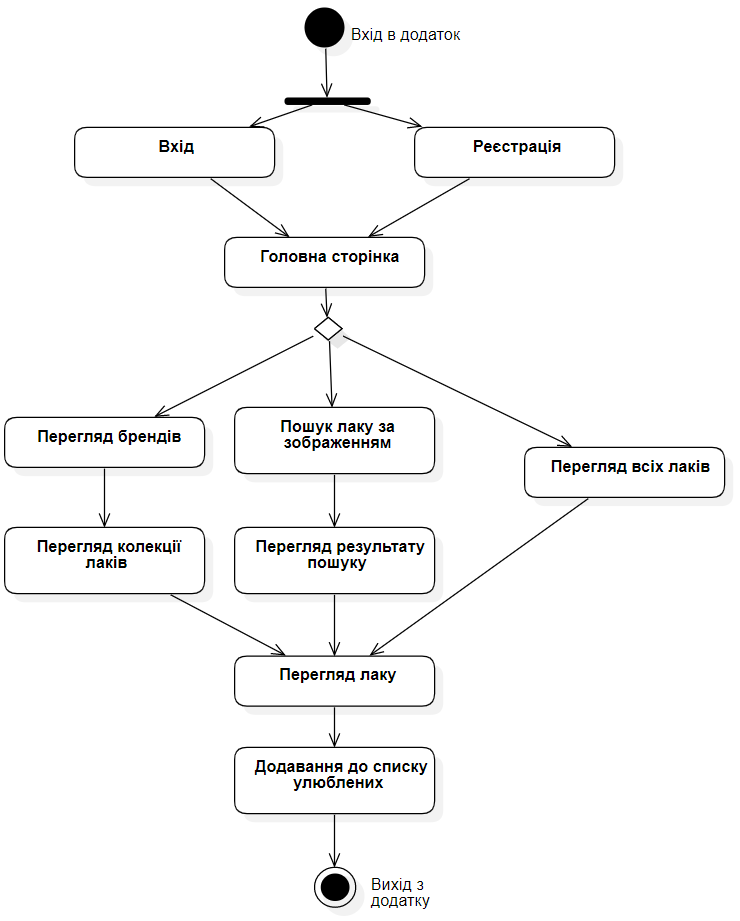


Рисунок 2.4 – Діаграма діяльності користувача

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 [LINK]. Узагальнена схема роботи алгоритмів кластеризації зображена на рисунку. В випадку вирішення задачі пошуку кольорів на зображені, кластерами є кольори, які ми шукаємо.

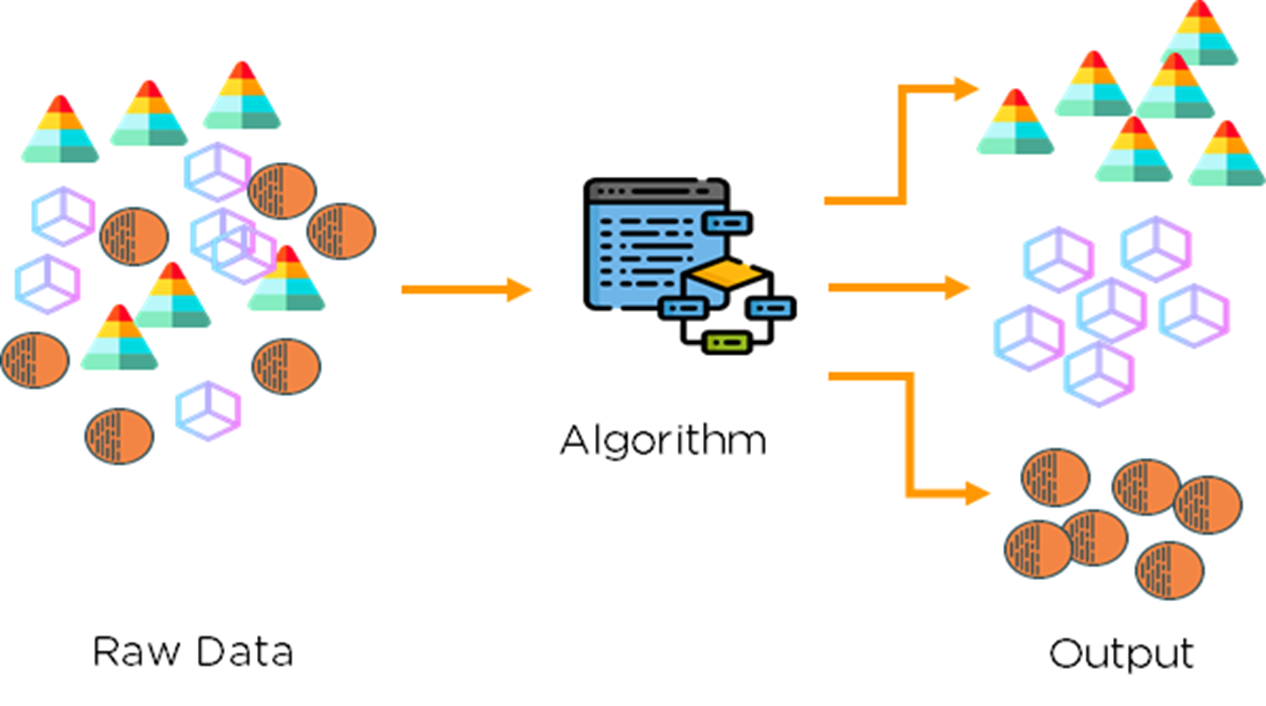


Рисунок 2.5 – Узагальнена схема роботи алгоритмів кластеризації

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 [LINK]. 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, синій [LINK].

Найпростіше уявити кожну 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:

*,*

де – складові компоненти першого кольору;

де – складові компоненти другого кольору;

– відстань між кольорами.

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.

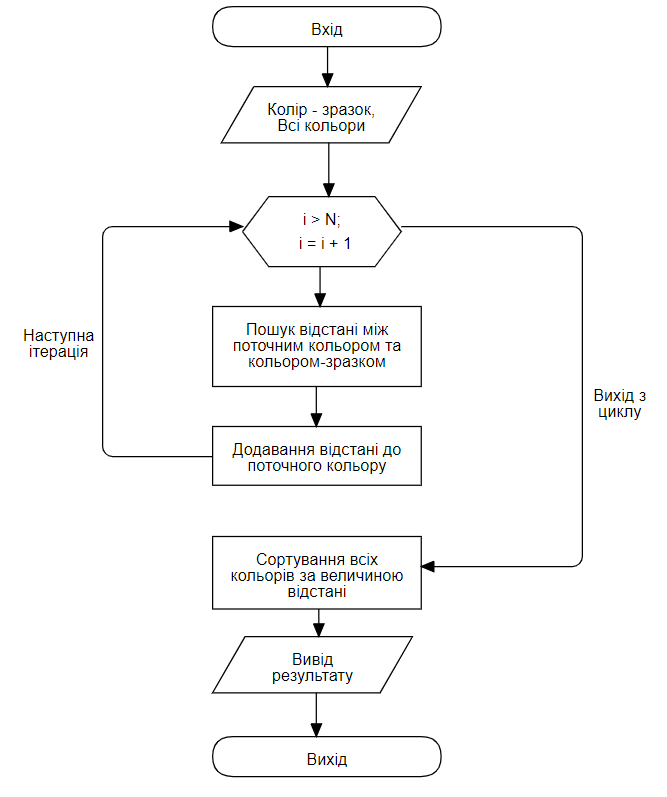


Рисунок 2.6 – Схема алгоритму пошуку кольору

## 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).

Програмний код ініціалізації серверного додатку, активації CORS та приклад обробки HTTP запиту з отриманню 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. 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 проекті 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 зображені.

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. 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 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.

# ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

## [Порядок](file:///C:\планшет\Магістерська%20робота1.docx#_Toc423399390) встановлення та налаштування системи.

## Висновки до третього розділу

У третьому розділі кваліфікаційної

# ВИСНОВКИ

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler, Martin. Patterns of enterprise application architecture. – 2003, Addison-Wesley.
2. Гамма Еріх, Хелм Річард, Джонсон Роберт, Вліссідес Джон. Прийоми об'єктно-орієнтованого проектування. – 2013, Пітер.
3. Android Projects overview [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/studio/projects>
4. Develop Android apps with Kotlin [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/kotlin>.
5. Firebase CLI Reference [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/cli>
6. Firebase Cloud Storage [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/storage>.
7. Firebase Realtime Database [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/database>.
8. Frameworks comparison [Електронний ресурс] – Режим доступу: https://vuejs.org/v2/guide/comparison.html#React
9. Getting Started [Електронний ресурс] – Режим доступу до ресурсу: <https://012.vuejs.org/guide>
10. Material Design Introduction Started [Електронний ресурс] – Режим доступу до ресурсу: <https://material.io/design/introduction/#principles>
11. Mobile Operating System Market Share Ukraine [Електронний ресурс] – Режим доступу до ресурсу: <http://gs.statcounter.com/os-market-share/mobile/ukraine/#monthly-201804-201904-bar>.
12. Service Workers [Електронний ресурс] – Режим доступу до ресурсу: <https://w3c.github.io/ServiceWorker/>.
13. Software Design Patterns [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/software-design-patterns.
14. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/TypeScript>.
15. Unified Modeling Language [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/Unified_Modeling_Language>.
16. What is Vuex? [Електронний ресурс] – Режим доступу: <https://vuex.vuejs.org/>.
17. Антипатерн [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/Антипатерн>.
18. Діаграма класів [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма\_класів.
19. Клієнт серверна архітектура [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Клієнт-серверна\_архітектура.
20. Реактивні потоки [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/Реактивні_потоки>.
21. https://uk.wikipedia.org/wiki/Python

# ДОДАТКИ

# ДОДАТОК А

Лістинг деяких класів Android додатку

NewVisitActivity.java

package com.boiko.taisa.salon.ui.activity

class NewVisitActivity : AppCompatActivity(), NewVisit.View

# ДОДАТОК Б

Лістинг деяких класів та скриптів веб додатку

firebase-messaging-sw.js

import \* as firebase from 'firebase/app';

import 'firebase/storage';