Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ З ПІДБОРУ СТИЛЮ ОДЯГУ
SOFTWARE IMPLEMENTATION OF A RECOMMENDER SYSTEM FOR SELECTING A CLOTHING STYLE

Виконав: студент(ка) 4 курсу, групи КНТ-117
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Марічев Д.О.
(прізвище та ініціали)
Керівник Колпакова Т.О.
(прізвище та ініціали)
Рецензент Голуб Т.В.
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
“____” _____ 2021року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Марічева Дениса Олександровича
(прізвище, ім’я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація рекомендаційної системи з підбору стилю одягу. Software Implementation of a Recommender System for Selecting a Clothing Style

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “ 30 ” березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) 17 травня 2021 року
3. Вихідні дані до проєкту (роботи) інформація про сучасні підходи до розробки вебзастосунків, перелік існуючих систем керування базами даних, мов програмування та їх вебфреймворків

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір засобів розробки програмного продукту. 3. Розробка серверної частини. 4. Розробка клієнтської частини. 5. Тестування розробленої системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Колпакова Т.О., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “ 15 ” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Вибір програмних засобів для розробки програми	4 тиждень	Розділ 2
4	Проєктування та розробка програми	5-6 тижні	Розділи 3-4
5	Тестування програми	7 тиждень	Розділ 5
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка) _____ Марічев Д.О.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Колпакова Т.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 109 с., 13 табл., 34 рис., 3 дод., 20 джерел.

ОДЯГ, СТИЛЬ, ВЕБСАЙТ, PYTHON, ПІДБІР ОБРАЗІВ, FULLSTACK, FLASK, HTML, CSS, JAVASCRIPT, SVELTE, JSON, POSTGRESQL, REDIS, SINGLE PAGE APPLICATION, БАКАЛАВР

Об’єкт дослідження – процес підбору стилю одягу.

Предмет дослідження – вебсайти для підбору готових образів (фотокарток моделей) на основі обраних елементів одягу.

Мета роботи – розроблення та розгортання вебзастосунку для полегшення перегляду та пошуку готових образів (фотокарток моделей) на основі обраних елементів одягу.

Матеріали, методи та технічні засоби: об’єктно-орієнтоване програмування, технологія Single Page Application, мови програмування Python, JavaScript, фреймворки Flask, Svelte, персональний комп’ютер з процесором AMD Ryzen 5 2600 під управлінням операційної системи Linux Ubuntu 20.10.

Результати. Створено вебзастосунок, який надає можливість користувачам легко знаходити та переглядати готові стилі одягу, створювати аккаунт та зберігати вподобані образи.

Висновки. Розроблено та розгорнуто вебзастосунок, який допомагає користувачам легко знаходити та переглядати готові стилі одягу, створювати аккаунт та зберігати вподобані образи. Для підтримки актуальності образів необхідно періодично додавати нові образи до системи.

Галузь використання – надання інформаційних послуг користувачам, які слідкують за тенденціями моди.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 109 pages, 13 tables, 34 figures, 3 appendixes, 20 sources.

CLOTHES, STYLE, WEBSITE, PYTHON, SELECTION OF IMAGES, FULLSTACK, FLASK, HTML, CSS, JAVASCRIPT, SVELTE, JSON, POSTGRESQL, REDIS, SINGLE PAGE APPLICATION, BACHELOR

The object of research is the process of choosing a style of clothing.

The subject of research are websites for the selection of ready-made images (photos of models) based on selected items of clothing.

The purpose of the work is to develop and deploy a web application to facilitate viewing and searching for ready-made images (photos of models) based on selected items of clothing.

There are such materials, methods and technical means as: object-oriented programming, Single Page Application technology, Python and JavaScript programming languages, Flask and Svelte frameworks, personal computer with AMD Ryzen 5 2600 processor running Linux Ubuntu 20.10.

The results. Developed and deployed a web application that allows users to easily find and view ready-made clothing styles, create an account and save their favorite images.

The conclusions. A web application has been developed and deployed that helps users easily find and view ready-made clothing styles, create an account, and save their favorite images. To maintain the relevance of the images, it is necessary to periodically add new images to the system.

The field of use is the provision of information services to users who follow fashion trends.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

JWT – JSON Web Token;

MPA – Multi Page Application;

SPA – Simple Page Application;

БД – база даних;

СКБД – система керування базами даних.

ВСТУП

На сьогоднішній день актуально слідкувати за індустрією моди. Людям необхідно відповідати певним соціальним статусам, і саме мода допомагає правильно орієнтуватися в створенні свого стилю.

У перекладі з латини мода – тимчасове, непостійне переважання смаків. В самому перекладі закладена суть цього терміну – постійна зміна.

Мода приходить і уходить, а стиль – це те, що людина обирає сама для себе. Стиль визначається елементами одягу, які обираються у відповідності один до одного. Існує безліч різних стилів одягу, наприклад, класичний, романтичний, спортивний тощо.

Мода і стиль – поняття хоча і різні, але завжди йдуть разом. Не можна носити все тільки ультрамодне, але й не можна носити лише одну класику, адже тоді ваші образи будуть занадто нудними. Саме тому з кожним роком стає все більше і більше стилістів, які допомагають людям виглядати модно і стильно [1].

Останнім часом, а саме з розвитком мережі Інтернет, почали з'являтися різні онлайн сервіси, які допомагають у різних сферах життя. Хотілося би, щоб знаходити стиль одягу можна було би не виходячи з дому, просто відкривши сайт, після чого за декілька кліків підібрати необхідний стиль одягу.

За допомогою подібних сервісів користувачі зможуть відзначати вподобані образи, які будуть включати підібрані один до одного елементи одягу, завдяки цьому буде формуватися список найпопулярніших образів.

Таким чином, користувачі зможуть знаходити та переглядати найпопулярніші образи для того, щоб в подальшому використовувати їх у реальному житті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мета, актуальність дипломної роботи

Дипломна робота присвячена розв’язанню актуальної задачі, а саме створенню зручного способу підбору модного стилю одягу за допомогою вебсайту, адже питання моди та стилю завжди були і будуть актуальними у нашому житті.

Метою даної роботи є дослідження різноманітних підходів до створення вебзастосунків, розроблення вебсайту, який надає можливість користувачам легко знаходити та переглядати готові стилі одягу, створювати аккаунт та зберігати вподобані образи.

Для досягнення зазначеної мети були виділені наступні задачі:

- проаналізувати існуючі вебсайти, які вирішують подібні завдання;
- проаналізувати основні методи веброзробки;
- спроєктувати структуру розроблюваного продукту;
- спроєктувати базу даних;
- розробити інтерфейс;
- розробити необхідні програмні компоненти;
- провести тестування розробленого вебсайту.

1.2 Огляд літератури

Для проєктування і розробки дипломного проєкту було розглянуто наступні технології: клієнт-серверна архітектура, API, SPA, різні мови програмування (Python, PHP, Java, C#, JavaScript, Ruby, Go), фреймворки (Flask, Django, Svelte, Angular, React), SQL та NoSQL бази даних тощо.

Додаткова література, що була розглянута:

- статті, що описують принципи API;
- статті про переваги та недоліки SPA;
- офіційна документація засобів розробки.

1.3 Аналіз існуючих програмних продуктів, що вирішують подібні завдання

Для того, щоб визначити список вимог до розроблюваного проєкту було проаналізовано вже існуючі рішення. Пошук аналогів був проведений за допомогою пошуку в мережі Інтернет.

lookastic.ru – сайт, який надає можливість користувачам переглядати різні образи, виконувати пошук по конкретним елементам одягу, додавати образи до списки вподобаних та надає можливість користувачу купляти вподобані елементи одягу за декілька кліків.

Інтерфейс головної сторінки сайту зображений на рисунку 1.1.

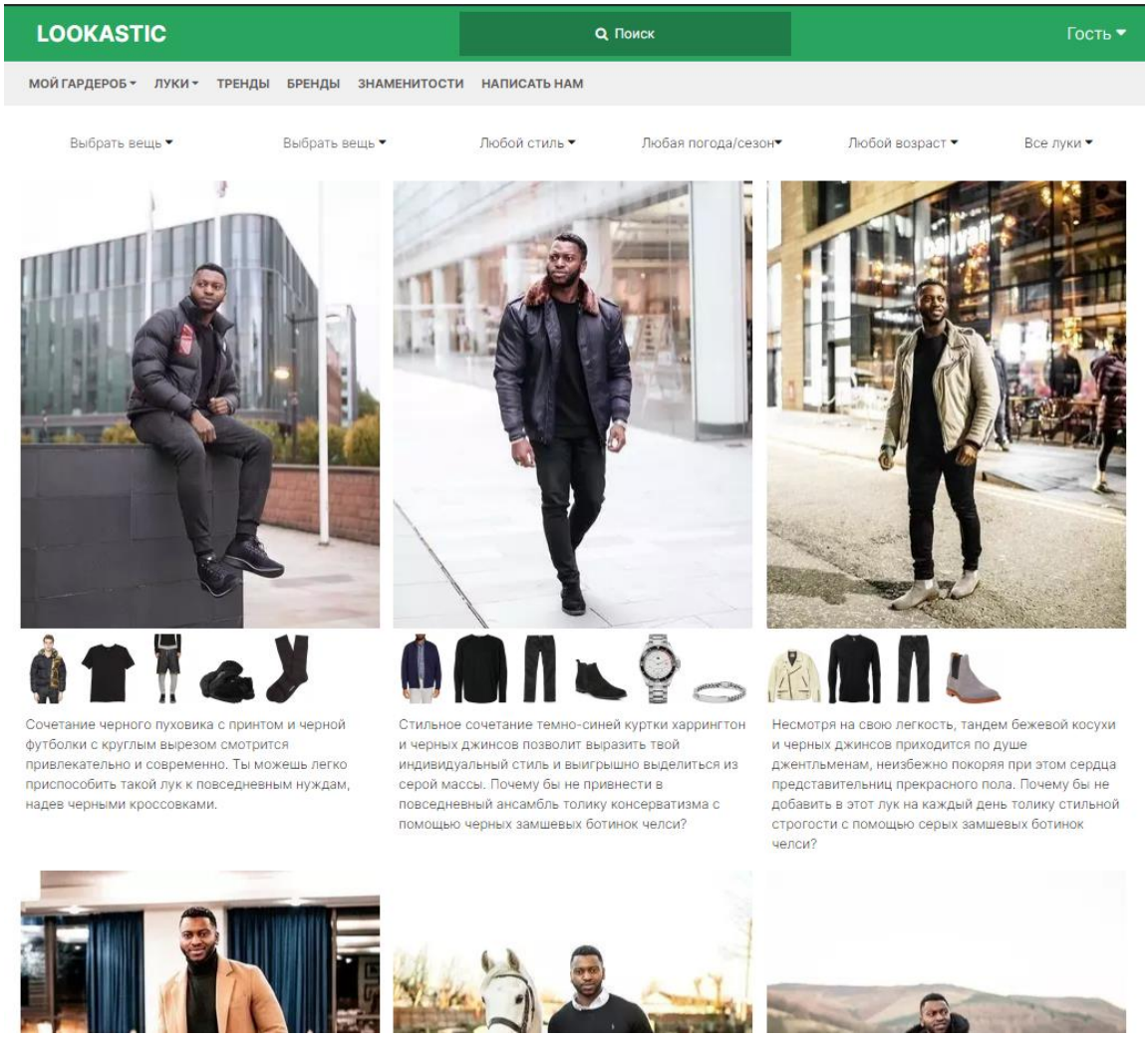


Рисунок 1.1 – Інтерфейс сайту lookastic.ru

nadevai.by – вебсайт, що дозволяє користувачам переглядати образи за встановленими фільтрами, наприклад, відобрази лише ті образи, в яких присутні сорочка ти балетки (рис. 1.2).

Сайт також надає можливість слідкувати за новинами індустрії моди.

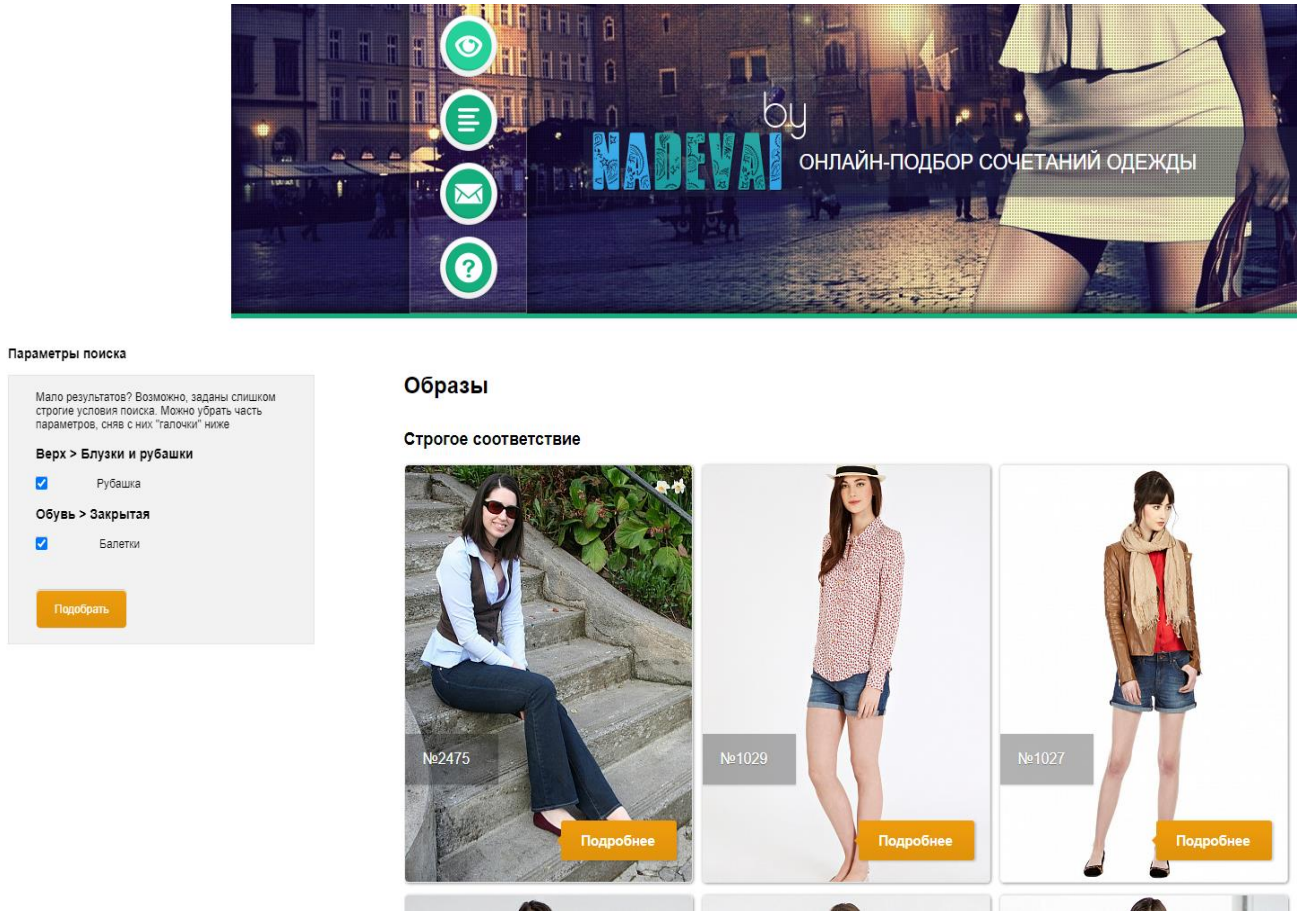


Рисунок 1.2 – Результати пошуку образів на сайті nadevai.by

lookbuck.com – інтернет магазин одягу, який публікую образи з тими елементами одягу, які присутні «на полицях» цього магазину.

При бажанні користувачі можуть зареєструватися та додавати вподобані образи до списку обраних.

Також дозволяє сортувати образи за сезонами року.

Інтерфейс сайту зображений на рисунку 1.3.

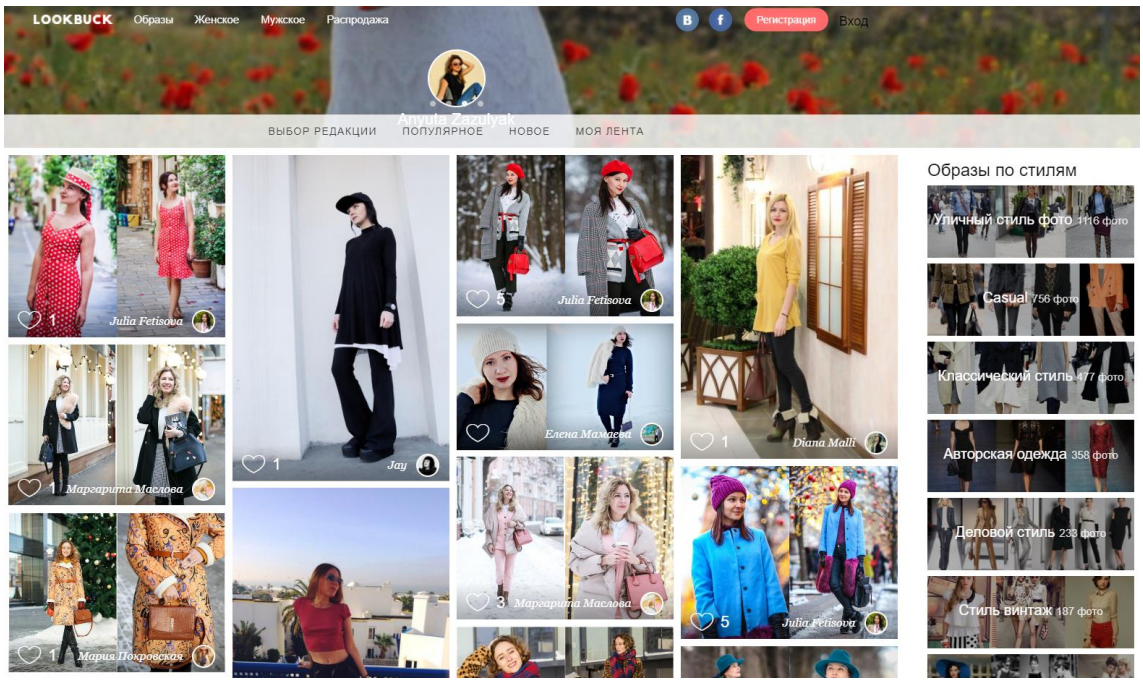


Рисунок 1.3 – Інтерфейс сайту lookbuck.com

styleclub.com.ua – вебсайт, за допомогою якого користувачі мають можливість комбінувати різні елементи одягу у свої власні образи (рис 1.4).

Рисунок 1.4 – Інтерфейс сторінки створення образу на сайті styleclub.com.ua

Після реєстрації з’являється можливість зберігати створені образи для подальшого перегляду.

Також присутнє сортування елементів одягу за брендом та кольором.

За результатами проведеного аналізу було сформовано порівняльну таблицю розглянутих сайтів за різними критеріями оцінки (табл. 1.1).

Таблиця 1.1 – Порівняння розглянутих вебсайтів

Назва критерію	Назва сайту			
	lookastic.ru	nadevai.by	lookbuck.com	styleclub.com.ua
Можливість пошуку образів за обраними елементами одягу	+	+	-	-
Можливість реєстрації акаунту	+	-	+	+
Можливість збереження вподобаних образів	+	-	+	-
Інтуїтивно зрозумілий інтерфейс сайту	±	+	±	±
Сучасність дизайну сайту	+	±	+	-
Адаптивний інтерфейс на мобільних пристроях	+	±	+	-
Швидкість завантаження сторінок	+	+	±	-

1.4 Висновки за розділом 1

Зважаючи на результати порівняльного аналізу, можна визначити основні вимоги до розроблюваного сайту. Перш за все, сайт повинен надавати можливість виконувати пошук образів за обраними елементами одягу та зберігати образи до списку вподобаних.

Для можливості зберегти вподобані образи, користувачу буде необхідно зареєструватися.

Сайт повинен бути інтуїтивно зрозумілим та мати дизайн, який відповідає сучасним нормам. Інтерфейс сайту повинен бути адаптивним задля коректного відображення на мобільних пристроях.

Для зручності користуванням сайту, він повинен швидко завантажуватися та відкриватися.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Принципи роботи сучасних вебзастосунків

Розроблюваний вебзастосунок буде працювати за принципами клієнт-серверної архітектури. В основі такої архітектури лежать два компоненти: клієнт і сервер.

Клієнт (frontend) – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій. В нашому випадку клієнтом буде виступати веббраузер користувача.

Сервер (backend) – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитами клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше [2].

Принцип клієнт-серверної архітектуру зображено на рисунку 2.1.

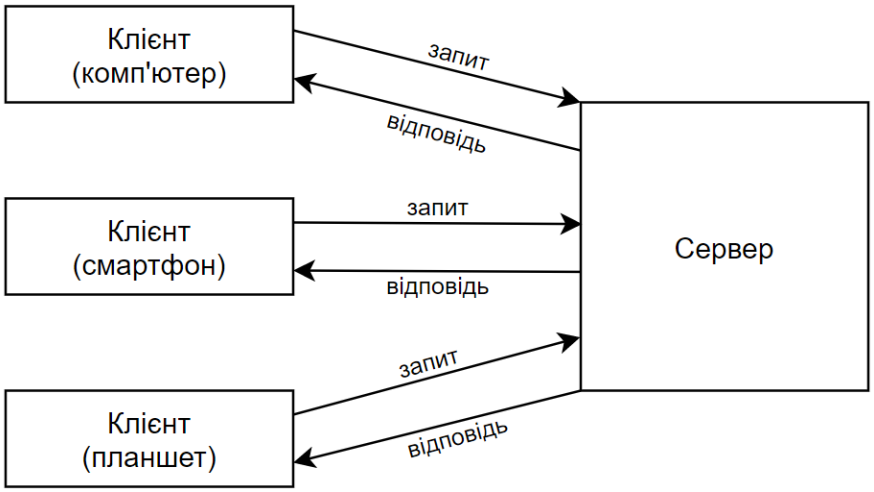


Рисунок 2.1 – Принцип клієнт-серверної архітектури

Сьогодні існує два загальних підходи до побудови вебдодатків: традиційні вебзастосунки, які виконують більшу частину логіки програми на сервері (Multi Page Application), та односторінкові програми (Single Page Application), які виконують більшу частину логіки інтерфейсу користувача у веббраузері, спілкуючись з сервером, в основному, використовуючи Application Programming Interface (API).

Однією з причин повільної роботи традиційних сайтів полягає в тому, що популярні серверні фреймворки відразу передають клієнтській частині готовий html файл. Наприклад, коли ми натискаємо на посилання на традиційному сайті, що демонструє слайд-шоу, екран блимає, і протягом декількох секунд відбувається перезавантаження всієї сторінки повністю: елементів навігації, рекламних банерів, верхнього колонтитула і тексту, хоча змінилася тільки поточна фотографія і, можливо, пояснювальний текст. Гірше того, немає ніякого індикатора, що показує, що якийсь елемент на сторінці знову готовий до взаємодії з користувачем. Таке повільне, непослідовне і неефективне функціонування стає неприйнятним для все більш вимогливих користувачів Інтернет [3].

Single Page Application можуть підтримувати розширену функціональність на стороні клієнта, яка не вимагає перезавантаження сторінки, коли користувачі виконують дії або переходять між областями вебсайту.

SPA можуть завантажуватися швидше, отримуючи дані у фоновому режимі, а на окремі дії користувача реагують ефективніше, оскільки повне перезавантаження сторінок трапляється рідко. Такі системи надають можливість підтримувати поступові оновлення, зберігаючи частково заповнені форми або документи, без того, щоб користувач повинен був натискати кнопку для подання форми.

Часто SPA повинні реалізовувати функції, вбудовані в традиційні вебпрограми, наприклад, змінення поточної URL адреси в адресному рядку, що відображає поточну операцію (дозволяючи користувачам робити

закладки або посилання на цю URL адресу, щоб повернутися на цю сторінку пізніше) [4].

Порівняння багатосторінкового та односторінкового принципів зображено на рисунку 2.2.

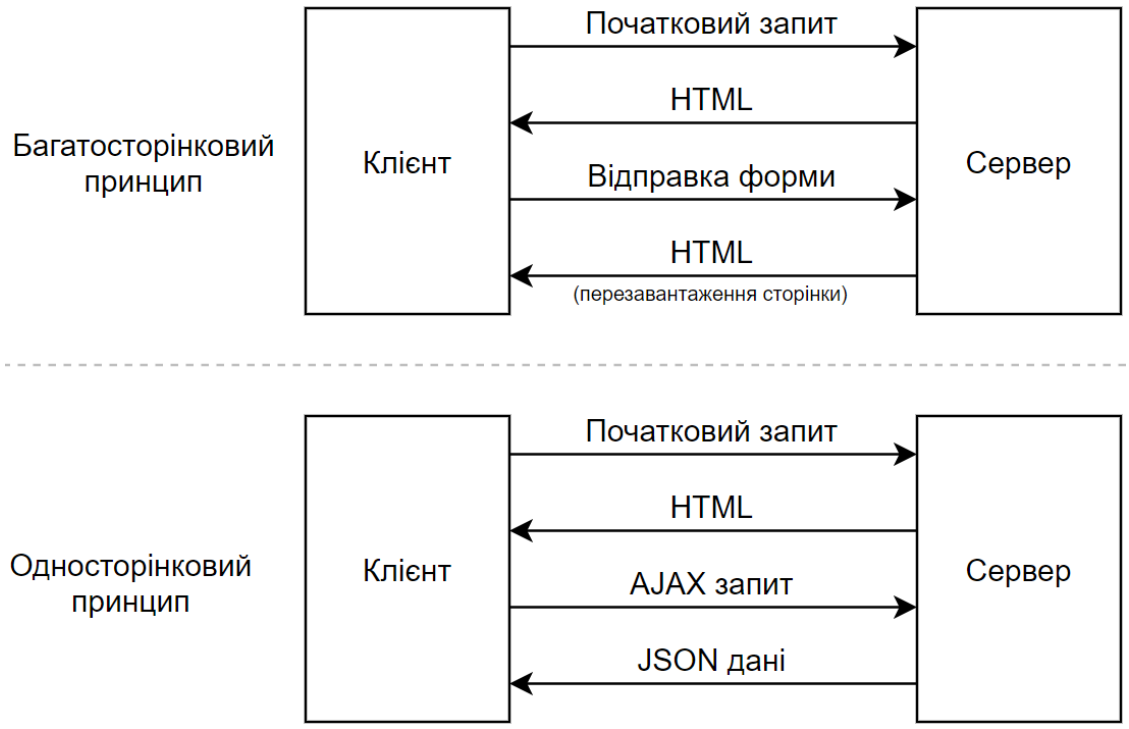


Рисунок 2.2 – Багатосторінковий та односторінковий принципи

2.2 Вибір інструментів для розробки серверної частини

2.2.1 Вибір мови програмування

Для розробки серверної частини серверу існує багато мов програмування. Розглянемо найпопулярніші з них, а саме Python, PHP, Java, C#, JavaScript, Ruby, Go.

Python – дуже популярна мова програмування, яка використовується як для веброзробки, так і для створення настільних додатків. У мережі Інтернет можна знайти величезну кількість навчальних сайтів, навчальних посібників і керівництв по Python, що робить цю мову доступною для новачків.

Крім того, синтаксис Python простий і легкий для розуміння в порівнянні з іншими мовами. Python підтримує об'єктно-орієнтоване, функціональне і аспектно-орієнтоване програмування, а також це динамічно типізована мова з відкритим вихідним кодом [5].

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата стандартна бібліотека (як вихідні тексти, так і бінарні дистрибутиви для всіх основних операційних систем) можуть бути отримані з сайту Python www.python.org, і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручний як мова розширення для прикладних програм, що потребують подальшого налагодження [6].

PHP використовують близько 78% всіх сайтів. Мова з'явилася в 1995 році, коли було не так багато можливостей для створення динамічних вебсторінок. PHP динамічно типізований, і один і той же фрагмент коду може вести себе по-різному в залежності від контексту, що робить програми на PHP складними для масштабування і іноді повільними.

PHP – відмінна мова для початківців по ряду причин:

- пробачає помилки: ви можете запустити програму, і вона буде виконуватися, поки не досягне ділянки з проблемним кодом;

- у мови велике співтовариство, а для новачків є багато навчальних матеріалів. Мова постійно оновлюється, тому переконайтеся, що вивчаєте останню версію;

- встановити і налаштувати PHP досить легко в порівнянні, наприклад, з Ruby.

Java – одна з найпопулярніших мов програмування. Вона універсальна і використовується вже більше 20 років.

Універсальність забезпечується віртуальною машиною Java (Java Virtual Machine). У багатьох мовах під час компіляції програма переводиться в код, який може працювати по різному на різних пристроях або платформах. В Java цієї проблеми немає. Віртуальна машина грає роль проміжного рівня – з програми на Java вона робить код, який може виконуватися на будь-якому комп'ютері незалежно від того, де код було скомпільовано.

C# – високорівнева мова програмування. Це означає, що розробники можуть писати на ньому програми, незалежні від архітектури процесора конкретного комп'ютера.

C# популярна серед розробників, тому що вона володіє деякими перевагами C++, але на ній простіше писати код і уникати при цьому грубих помилок.

JavaScript (JS) – гнучка мова, яку можна використовувати як для фронтенду, так і для бекенду.

Це гарна мова для початківців, оскільки в ній мало налаштувань, і можна почати писати код прямо в браузері.

Гнучкість JavaScript часом обходиться дорого: скрипти працюють повільно, їх складно підтримувати і масштабувати, як і в більшості динамічно типізованих мов. У JavaScript велика спільнота, і для її вивчення є багато корисних матеріалів.

Значно розширює можливості JavaScript програмна платформа Node.js. З її допомогою код, написаний на JavaScript, можна запускати без браузера на серверній частині, а наявність величезної кількості готових рішень в пакетній екосистемі npm дозволяє розробнику не витрачати час на створення більшості типових рішень.

Ruby – досить лаконічна мова і не вимагає багато коду для бекенду, що дозволяє розробникам швидко створювати і запускати прототипи (цим Ruby схожий на Python). Популярність Ruby зросла на початку 2000 років, але з тих пір помітно знизилася. Ця мова з відкритим вихідним кодом, а значить його можна модифікувати і доповнювати.

Go – компільована мова програмування, розроблена в середині компанії Google. Мова Go розроблялася для створення високоефективних програм, що працюють на сучасних розподілених системах і багатоядерних процесорах. Це мова з суворою статичною типізацією, вона має досить лаконічний і простий синтаксис, заснована на C, але істотно доопрацьована, з великою кількістю синтаксичного цукру [5].

За результатами аналізу мов програмування було створено порівняльну таблицю (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Характеристика	Мова програмування						
	Python	PHP	Java	C#	JS	Ruby	Go
Простота	10	8	5	5	8	2	6
Читабельність	10	7	6	6	7	3	8
Швидкість розробки	9	8	7	6	9	5	7
Популярність	9	7	6	6	8	4	5
Кількість готових бібліотек	9	3	6	6	6	7	4
Швидкість виконання коду	5	4	8	7	7	6	9
Всього балів	52	37	38	36	45	27	39

Зважаючи на результати порівняльного аналізу, мовою програмування серверної частини було обрано Python. Таким чином, обравши цю мову програмування ми отримуємо простоту синтаксису, велику читабельність коду та велику кількість готових бібліотек, через що ми і отримуємо високу швидкість розробки.

2.2.2 Вибір фреймворку

При прийнятті рішення, який фреймворк використовувати, необхідно в першу чергу звернути увагу на розмір і складність проєкту. Якщо необхідно розробляти велику систему з безліччю функцій, то правильним вибором може бути фреймворк повного стека (Fullstack Framework). Якщо ж вебдодаток досить простий з обмеженим набором функцій, то варто звернути увагу на мікрофреймворки.

Так само варто проаналізувати чи буде додаток в майбутньому масштабуватися, які ресурси будуть необхідні для його роботи. Досить одного сервера або потрібно буде кілька серверів, які обсяги трафіку і даних будуть проходити і який обсяг нового функціоналу може бути доданим.

Фреймворки в своєму розвитку дотримуються визначених принципів, що може стати певним обмеженням в застосуванні тих чи інших технологій. Завжди залишиться можливість обійти обмеження і реалізувати свій функціонал, але часто це призводить до великих часових витратах, ніж просто написати все на чистому Python.

Найпопулярнішими фреймворками, які використовують мову Python, являються Django та Flask.

Django – фреймворк повного стека. Він включає в себе багато часто використовуваних функцій, замість того, щоб доповнюватися ними у вигляді окремих бібліотек. Він включає в себе аутентифікацію, URL роутинг, генерацію шаблонів відображення, ORM для взаємодії з БД і ряд інших функцій. Ці функції і роблять Django легко масштабованим, неймовірно швидким і надзвичайно універсальним.

З «коробки» Django легко працює з такими БД як PostgreSQL, MySQL, SQLite і Oracle. Взаємодія з іншими БД можуть бути реалізовані сторонніми додатками. З Django можна створити будь-який вебдодаток від невеликих проєктів до складних вебсайтів.

Популярність Django як фреймворка для розробки вебдодатків продовжує зростати [7].

Flask – це, по суті, мікрофреймворк, і тому більша увага в ньому приділена простоті роботи, а не функціональності. Для налаштування і встановлення потрібно набагато менше часу, ніж з іншими фреймворками, а над HTTP-функціями буде менше рівнів абстракції.

Flask був натхненний фреймворком Sinatra Ruby. Основна ідея Flask полягає в тому, щоб допомогти створити міцну основу вебдодатків. Він має ряд корисних функцій: вбудований сервер розробки і швидкий відладчик, Jinja2 шаблонізатор, підтримка безпечних файлів cookie, можливість підключати будь-яку ORM [8].

Зважаючи на те, що потрібен мінімалістичний фреймворк було вирішено обрати Flask в якості фреймворку для серверної частини вебзастосунку.

2.2.3 Вибір системи керування базою даних

Для збереження даних можна використовувати звичайні текстові файли, але це буде дуже незручно та малоефективно. Для зберігання даних прийнято використовувати бази даних (БД). БД, в кінцевому підсумку, теж зберігає дані у файлах, але цим процесом керує так звана система керування базами даних (СКБД). Є декілька найпопулярніших СКБД, а саме MySQL, PostgreSQL, MongoDB та Redis.

MySQL. Проста в установці, працює без особливих налаштувань. При належному підході може гнучко налаштовуватися під різні потреби, але є і підводні камені, в деяких випадках вона буде повільно працювати, як би її не конфігурували.

Мінуси MySQL:

- низька швидкість роботи;

- зміна структури даних може бути досить трудомістким процесом, особливо при великій кількості зав’язків між даними в різних таблицях і навіть при простому додаванні полів;

- чутливість до нестабільності сервера. Якщо неправильно завершити MySQL, можна настільки пошкодити таблиці і бази даних, що відновити можна буде тільки з повного резервного відновлення, звичайно, якщо його регулярно робити.

PostgreSQL – свого роду мастодонт, дуже стара і стабільна СКБД. Вона майже як MySQL, тільки краще. Але треба вміти її налаштовувати. Ця СКБД є дуже стабільною, її практично неможливо зламати, пошкодити таблиці як в MySQL. Зараз PostgreSQL розвивається групою інтернет-розробників, приблизно тим же способом, що і інше Open Source програмне забезпечення. Користувачі мають доступ до вихідного коду, вони виправляють помилки, займаються удосконаленням продукту, пропонують введення нових можливостей.

Мінуси PostgreSQL:

- необхідність досвіду роботи з цією СКБД, щоб добре її налаштувати для великих проєктів;

- система авторизації за замовчуванням може викликати труднощі при використанні або налаштуванні.

MongoDB – документно-орієнтована СКБД, яка не потребує опису схеми таблиць. В цій СКБД дані зберігаються у форматі JSON. В деяких випадках MongoDB справляється із завданням набагато краще, ніж MySQL або PostgreSQL.

Мінуси MongoDB:

- немає простих транзакцій, принаймні в тому класичному вигляді, якими вони є в MySQL та PostgreSQL. При додаванні великої кількості даних, які залежать один від одного, можуть бути певні труднощі, які доведеться вирішувати самостійно на рівні коду;

- практично відсутня зв’язаність даних;

- потрібно перебудовувати мислення саме під NoSQL.

Redis. Найчастіше цю СКБД використовують в якості кешу для роботи з даними з іншої, більш повільної СКБД. Нечасто, але все ж може використовуватися в якості самостійно БД для даних, при цьому Redis вміє працювати з різними типами даних, в тому числі списками, чергами, а ще дуже просто працювати з часом життя даних. Redis працює в оперативній пам'яті, через це вона дуже швидка. Також вміє зберігати дані на диску.

Мінуси:

- обсяг даних не повинен перевищувати обсяг вільного ОЗУ на сервері;

- бажано зберігати не дуже важливі дані, бо деякі з них можуть бути втраченими після рестарту сервера;

- відсутність, в класичному розумінні, транзакцій [9].

Проаналізувавши усі властивості, переваги і недоліки різних СКБД, було вирішено використовувати PostgreSQL, як основну БД, а Redis – для кешування даних, а саме для пришвидшення роботи основної БД.

2.3 Вибір інструментів для розробки клієнтської частини

Клієнтську частину можна реалізувати, використовуючи лише стандартні можливості мови JavaScript, але це займе набагато більше часу, ніж у випадку використання JavaScript фреймворків. Розглянемо найпопулярніші з них.

React – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження

сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим.

React обробляє тільки користувацький інтерфейс у застосунках. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити [10].

Vue – JavaScript-фреймворк, що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних за допомогою реактивного зв'язування даних.

Vue використовує синтаксис шаблонів на основі HTML, що дозволяє декларативно зв'язувати рендеринг DOM з основними екземплярами даних в Vue. Всередині Vue компілює шаблони в рендерингові функції віртуального DOM. В поєднанні з реактивною системою, Vue здатний розумно обчислити кількість компонентів для ре-рендерингу та застосувати мінімальну кількість маніпуляцій з DOM, коли стан застосунку зміниться [11].

Svelte – це компонентний фреймворк, схожий на React або Vue, але з важливою відмінністю. Традиційні фреймворки дозволяють писати декларативний state-driven код, але є й мінуси: браузер повинен виконати додаткову роботу для перетворення цих декларативних структур в маніпуляції з DOM, використовуючи техніки, такі як virtual DOM diffing, які проїдають наявний бюджет кадрів відтворення і додають обов'язків складальника сміття.

Замість цього Svelte працює під час збирання (компіляції), перетворюючи ваші компоненти в високоефективний імперативний код, який

з хірургічною точністю оновлює DOM. В результаті можна писати амбітні програми з відмінними характеристиками по продуктивності.

Перша версія Svelte була присвячена перевірці гіпотези, що спеціально створений компілятор може генерувати надійний код і забезпечувати відмінний користувацький досвід. Svelte 3 – це вже суттєвий перегляд концепції. На ньому можна писати компоненти з кількістю шаблонного коду, значно меншою, ніж деінде [12].

Результати огляду фронтенд фреймворків було об’єднано у порівняльну таблицю (табл. 2.2).

Таблиця 2.2 – Порівняння фронтенд фреймворків

Характеристика	Назва фреймворку		
	React	Vue	Svelte
Простота	+	+	+
Документація	-	+	+
Наявність віртуального DOM	+	+	-
Швидкість роботи	+-	+-	+
Малий розмір готового js файлу	-	-	+
Справжня реактивність	-	-	+

2.4 Висновки за розділом 2

В розділі було проведено аналіз мов програмування, технологій збереження даних, та сучасних фреймворків.

За результатами аналізу були обрані мови програмування, а саме Python для серверної частини та JavaScript для клієнтської частини. В якості вебфреймворків було обрано Flask та Svelte. В якості системи керування базами даних було обрано реляційну БД PostgreSQL та допоміжну NoSQL БД – Redis.

3 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

3.1 Проєктування бази даних

База даних відіграє важливу роль у роботі вебзастосунку, адже саме в ній зберігаються усі оброблені в процесі роботи дані. В базі даних можуть зберігатися великі обсяги різних даних, через це необхідно правильно спроєктувати структуру БД. Задля цього було вирішено використати алгоритми нормалізації.

Нормалізація схеми бази даних – покроковий процес розбиття одного відношення (таблиці) відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей.

Нормальна форма – властивість відношення в реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно може призвести до помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

Перша нормальна форма (1НФ, 1NF) утворює базу для структурованої схеми бази даних:

- кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які ідентифікують запис;

- уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи неключові атрибути;

- атомарність, кожен атрибут повинен мати лише одне значення, а не множину значень.

Друга нормальна форма (2НФ, 2NF) вимагає, аби дані, що зберігаються в таблицях із композитним ключем, не залежали лише від частини ключа:

- схема бази даних повинна відповідати вимогам першої нормальної форми;

- дані, що повторно з'являються в декількох рядках, виносяться в окремі таблиці.

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

- схема бази даних повинна відповідати всім вимогам другої нормальної форми;

- будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю [13].

Для реалізації дипломного проєкту було вирішено привести БД до третьої нормальної форми.

Систему баз даних можна розділити на два головні компоненти: сервер і набір клієнтів (або зовнішніх інтерфейсів). Сервер – це і є СКБД. Клієнтами є різні додатки, написані прикладними програмістами, або вбудовані додатки, що поставляються разом з СКБД. Один сервер може одночасно обслуговувати багато клієнтів.

Сучасні СУБД включають в себе словник даних. Це частина бази даних, яка описує самі дані, що зберігаються в ній. Словник даних допомагає СУБД виконувати свої функції [14].

В розроблюваному проєкті необхідно зберігати наступні дані: інформацію про користувачів, елементі одягу, образи, інформацію про те, які саме елементи одягу входять до конкретного образу, інформацію про те, які саме образи користувач додавав до вподобаних.

Проаналізувавши необхідні дані для збереження в БД, було виділено наступні сутності: «Користувач», «Елемент одягу» та «Образ».

Дані про елементи одягу мають деревовидну структуру, приклад наведено на рисунку 3.1.

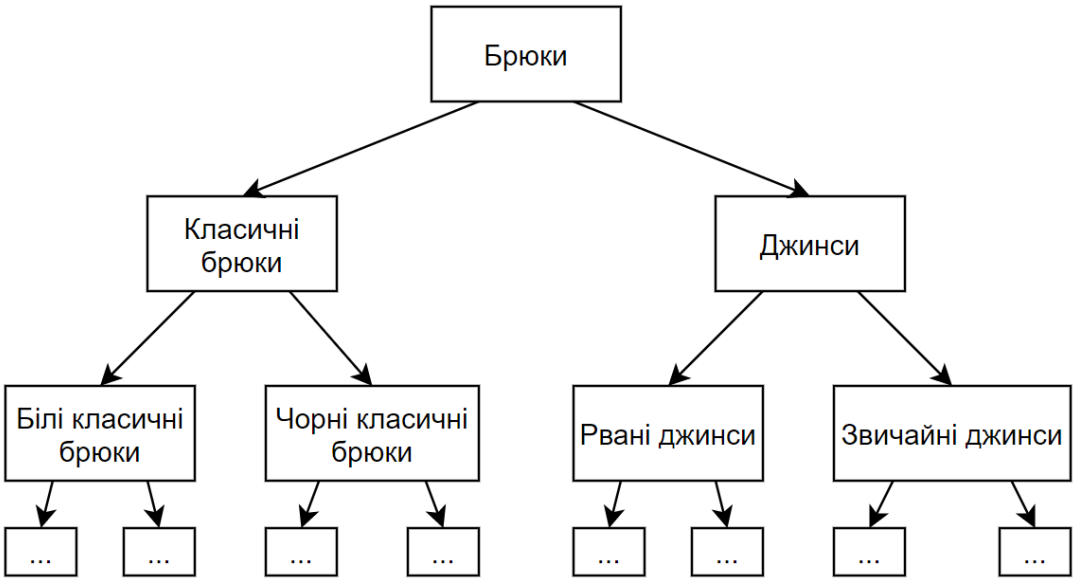


Рисунок 3.1 – Приклад деревовидної структури елементів одягу

Для збереження даних у деревовидній структурі можна зробити так, щоб кожний елемент посилався на свого предка. Структура таблиці «all_clothes» наведена у таблиці 3.1.

Таблиця 3.1 – Структура полів у таблиці «all_clothes»

Назва	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор запису
name	VARCHAR(100)	Назва елементу одягу
image_path	VARCHAR(250)	Шлях до файлу зображення
parent_id	BIGINT	Посилання на батьківський елемент одягу
parent_path	LTREE	Шлях до цього елементу з урахуванням усіх предків
display_priority	INTEGER	Пріоритет відображення

«users» – таблиця, яка зберігає інформацію про користувачів. Структура полів продемонстрована у таблиці 3.2.

Таблиця 3.2 – Структура полів таблиці «users»

Назва	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор запису
username	VARCHAR(40)	Ім’я
password_hash	VARCHAR(70)	Хеш паролю
registered_at	TIMESTAMP	Дата реєстрації

«characters» – таблиця, яка зберігає інформацію про образи. Структура полів продемонстрована у таблиці 3.3.

Таблиця 3.3 – Структура полів таблиці «characters»

Назва	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор запису
author_id	BIGINT	Посилання на автора
image_path	VARCHAR(250)	Шлях до файлу зображення
description	TEXT	Опис образу
posted_at	TIMESTAMP	Дата створення образу

«clothes_on_characters» – таблиця, яка використовується для реалізації зв’язку «багато-до-багатьох» між таблицею «characters» та «all_clothes». Описую те, які саме елементи одягу присутні в образі. Структура полів продемонстрована у таблиці 3.4.

Таблиця 3.4 – Структура полів таблиці «clothes_on_characters»

Назва	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор запису
character_id	BIGINT	Посилання на образ
clothes_id	BIGINT	Посилання на елемент одягу

«favorite_characters_of_users» – таблиця, яка використовується для реалізації зв’язку «багато-до-багатьох» між таблицею «characters» та «users». Описую те, які образи користувач додав до вподобаних. Структура полів продемонстрована у таблиці 3.5.

Таблиця 3.5 – Структура полів таблиці «favorite_characters_of_users»

Назва	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор запису
user_id	BIGINT	Посилання на користувача
character_id	BIGINT	Посилання на образ

На рисунку 3.2 зображена ER-діаграма бази даних.

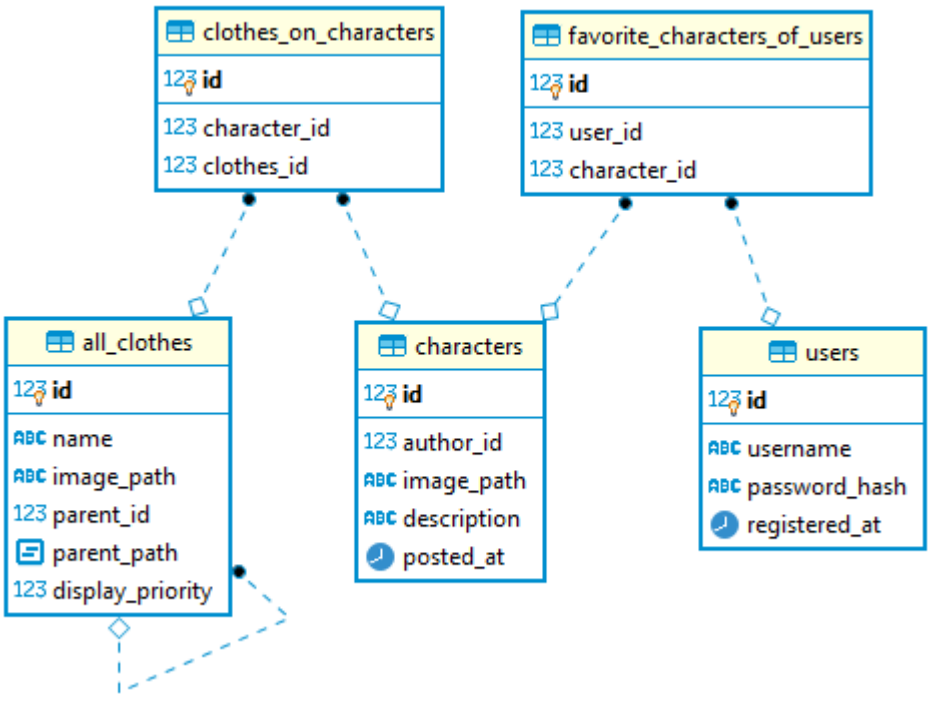


Рисунок 3.2 – ER-діаграма бази даних

3.2 Розробка вебсерверу

Перш за все необхідно створити класи (моделі), в яких будуть зберігатися дані після діставання їх з БД: Clothes, Character та User. Структуру цих класів (моделей) наведено в таблицях 3.6-3.8 відповідно.

Таблиця 3.6 – Структура класу Clothes

Назва поля	Тип поля	Опис
id	int	Унікальний ідентифікатор запису
name	str	Назва елементу одягу
image_path	str	Шлях до файлу зображення
parent_id	int	Посилання на батьківський елемент одягу
parent_path	str	Шлях до цього елементу з урахуванням усіх предків

Таблиця 3.7 – Структура класу Character

Назва поля	Тип поля	Опис
id	int	Унікальний ідентифікатор запису
author_id	int	Посилання на автора
image_path	str	Шлях до файлу зображення
description	str	Опис образу
posted_at	datetime	Дата створення образу
likes	int	Кількість вподобань
clothes	list	Список елементів одягу у складі цього образу
is_favorite	bool	Чи є цей образ у списку вподобаних образів користувача

Таблиця 3.8 – Структура класу User

Назва поля	Тип поля	Опис
id	int	Унікальний ідентифікатор запису
username	str	Ім’я
password_hash	str	Хеш паролю
registered_at	datetime	Дата реєстрації

Авторизація та аутентифікація користувачів буде здійснюватися за допомогою JSON Web Token (JWT).

JWT – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON. Ця інформація є перевіреною і надійною, тому що вона має цифровий підпис.

JWT можуть бути підписані з використанням секретного (за допомогою алгоритму HMAC) або пари відкритого/секретного ключів з використанням RSA або ECDSA.

JSON Web Token використовується для передачі інформації, що стосується особистості і характеристик клієнта. Цей «контейнер» підписується сервером, щоб клієнт не втручався в нього і не міг змінити, наприклад, ідентифікаційні дані або будь-які характеристики (наприклад, змінити роль з простого користувача на адміністратора або змінити логін клієнта).

Цей токен створюється в разі успішної аутентифікації і перевіряється сервером перед початком виконання кожного клієнтського запиту. Токен використовується додатком як "посвідчення особи" клієнта (контейнер з усією інформацією про нього). Сервер ж має можливість перевіряти дійсність і цілісність токена безпечним способом. Це дозволяє додатку бути stateless (stateless програма не зберігає дані клієнта, згенеровані за один сеанс для використання в наступному сеансі з цим клієнтом (кожен сеанс виконується незалежно)), а процесу аутентифікації незалежним від використовуваних

сервісів (в тому сенсі, що технології клієнта і сервера можуть різнитися, включаючи навіть транспортний канал, хоча найбільш часто використовується HTTP) [15].

JWT – складається з трьох частин:

- заголовок – частина, в якій зберігається інформація службова інформація, наприклад, тип шифрування токену;

- корисне навантаження – частина, в які зберігаються дані, наприклад, ім’я користувача чи його унікальний ідентифікатор;

- підпис – частина, яка утворюються методом об’єднання заголовку та корисного навантаження з подальшим шифруванням.

Приклад JWT зображений на рисунку 3.3.

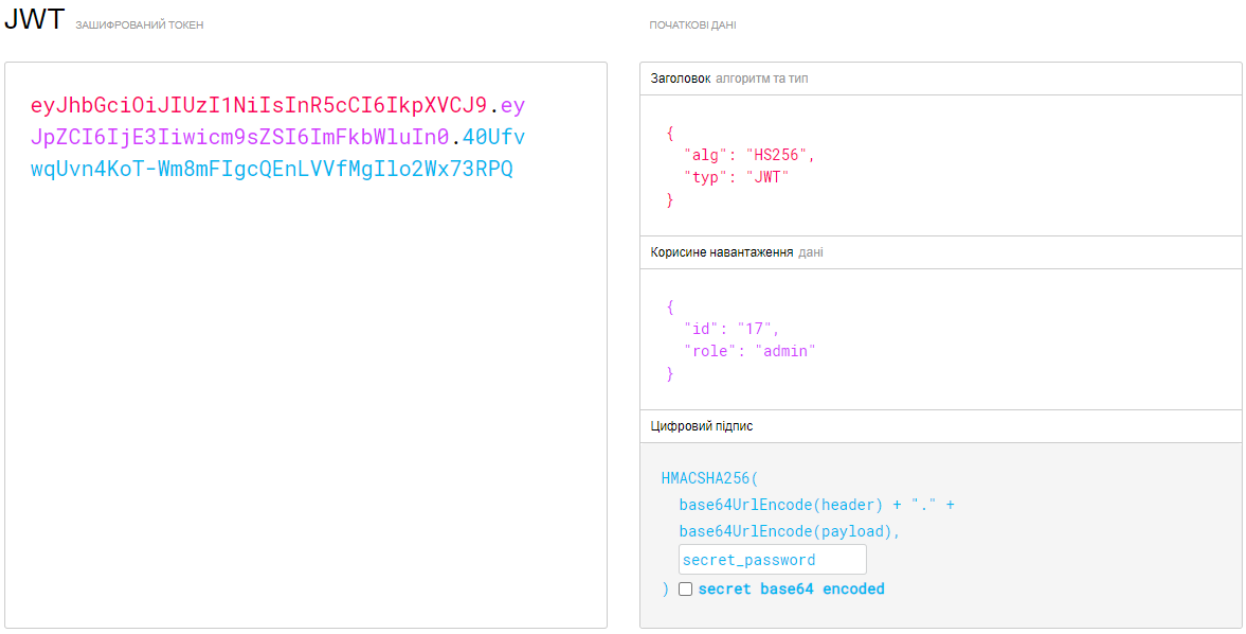


Рисунок 3.3 – Приклад JWT

Після першого логіна клієнту повертається згенерований сервером JWT. При кожному наступному запиті, клієнт повинен передавати JWT встановленим API способом (наприклад, через заголовок або як параметр запиту). Сервер декодує заголовок і корисне навантаження і перевіряє зарезервовані поля. Якщо все у порядку, за вказаним в заголовку алгоритмом

складається підпис. Якщо отриманий підпис збігається з переданим, користувача авторизують [16].

Після того, як користувач завершує сеанс роботи на сайті, його токен стає недійсним і видаляється на стороні клієнта (у браузері).

При спробі отримати доступ до захищених ресурсів, а саме тих, для роботи з якими необхідно бути авторизованим, сервер поверне помилку з кодом 401 Unauthorized.

Для керування JWT був розроблений спеціальний клас, його код наведено нижче:

```
class JWTAuthorization:
    def __init__(
        self, secret_key,
        hash_method='sha256',
        hash_iterations=10000,
        jwt_algorithm='HS256'
    ):
        self.secret_key = secret_key
        self.hash_method = hash_method
        self.hash_iterations = hash_iterations
        self.jwt_algorithm = jwt_algorithm

    def generate_password_hash(self, password: str) -> str:
        """Creates a hashed password from a raw password"""
        return pbkdf2_hmac(
            hash_name=self.hash_method,
            password=password.encode('utf-8'),
            salt=self.secret_key.encode('utf-8'),
            iterations=self.hash_iterations
        ).hex()

    def is_correct_password(self, password: str, hash: str) -> bool:
        """Compares the raw password with a hashed password"""
        return self.generate_password_hash(password) == hash

    def create_token(self, data: dict, exp: Optional[int] = None) -> str:
        """Creates and returns a JWT with the required data"""
        if exp:
            data['exp'] = exp

        return jwt.encode(
```

```
            payload=data,
            key=self.secret_key,
            algorithm=self.jwt_algorithm
        )

    def get_acces_token_from_headers(self, headers: dict) -> str:
        """Retrieves and returns a token from the request headers"""
        auth_data = headers.get('Authorization')

        if not auth_data:
            raise AuthorizationException(
                'You should add the \'Authorization\' header to your request!'
            )

        if not auth_data.startswith('Bearer '):
            raise AuthorizationException(
                'The \'Authorization\' header must start with \'Bearer \'!'
            )

        return auth_data[7:]

    def get_user_id_from_acces_token(self, token: str) -> int:
        """Retrieves and returns user_id from token"""
        try:
            data = jwt.decode(
                jwt=token,
                key=self.secret_key,
                algorithms=self.jwt_algorithm
            )
            return data['user_id']
        except KeyError:
            raise AuthorizationException(
                'Can\'t extract \'user_id\' from token!'
            )
        except jwt.exceptions.ExpiredSignatureError:
            raise AuthorizationException(
                'You are using an expired token!'
            )
        except jwt.exceptions.DecodeError:
            raise AuthorizationException(
                'Token cannot be decoded because it failed validation!'
            )
        except jwt.exceptions.InvalidSignatureError:
            raise AuthorizationException(
                ('Token’s signature doesn’t match the one provided as part of '
```

```
            'the token!')
        )
    except jwt.exceptions.InvalidTokenError:
        raise AuthorizationException(
            'Token cannot be decoded!'
        )
```

Для доступу клієнтської частини до серверної необхідно розробити інтерфейс комунікації, а саме URL адреси, на які будуть здійснюватися HTTP запити з клієнтської частини. Перелік усіх URL адрес та їх опис наведено у таблиці 3.9.

Таблиця 3.9 – Перелік усіх доступних URL адрес

URL	Метод	Опис
1	2	3
/api/v1/users/register	POST	Реєструє користувача у системі
/api/v1/users/login	POST	Авторизує користувача та повертає йому JWT
/api/v1/clothes	GET	Повертає перелік усіх елементів одягу
/api/v1/characters/find	GET	Знаходить та повертає перелік образів, в яких присутні передані елементи одягу
/api/v1/characters/<id>/add-to-favorites	POST	Додає образ до вподобаних
/api/v1/characters/<id>/remove-from-favorites	POST	Видаляє образ із вподобаних
/api/v1/clothes/primary	GET	Повертає перелік тільки основних елементів одягу
/api/v1/clothes/by-parent-id/<id>	GET	Повертає елементи одягу, які мають конкретного предка

Продовження таблиці 3.9

1	2	3
/api/v1/characters	GET	Повертає перелік усіх образів
/api/v1/characters/<id>	GET	Повертає конкретний образ
/api/v1/characters/favorites	GET	Повертає вподобані образи користувача
/api/v1/characters/filter	GET	Повертає відфільтрований перелік образів

Для розробки серверної частини було використано наступні бібліотеки:

- flask – основний вебфреймворк;
- psycopg2 – бібліотека для роботи з СКБД PostgreSQL;
- redis – бібліотека для роботи з СКБД Redis;
- pyjwt – бібліотека для полегшення роботи з JWT;
- pydantic – бібліотека для валідації даних;
- python-dotenv – бібліотека для роботи зі змінними середовища.

3.3 Висновки за розділом 3

Було спроєктовано та розроблено серверну частину проєкту. Спроєктовано структуру БД та розроблено систему авторизації користувачів за допомогою JWT. Для доступу клієнтської частини до серверної було розроблено інтерфейс комунікації, а саме URL адреси, на які будуть здійснюватися HTTP запити з клієнтської частини.

4 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

4.1 Проєктування та розробка інтерфейсу

Перед тим, як почати створювати інтерфейс сторінок за допомогою HTML та CSS, необхідно продумати розташування основних елементів на розроблюваних сторінках. Після цього створюються так звані «мокапи», на яких відображаються основні елементи сторінок у вигляді простих примітивів.

На рисунку 4.1 зображений «мокап» головної сторінки, сторінки з результатами пошуку та сторінки вподобаних образів користувача.

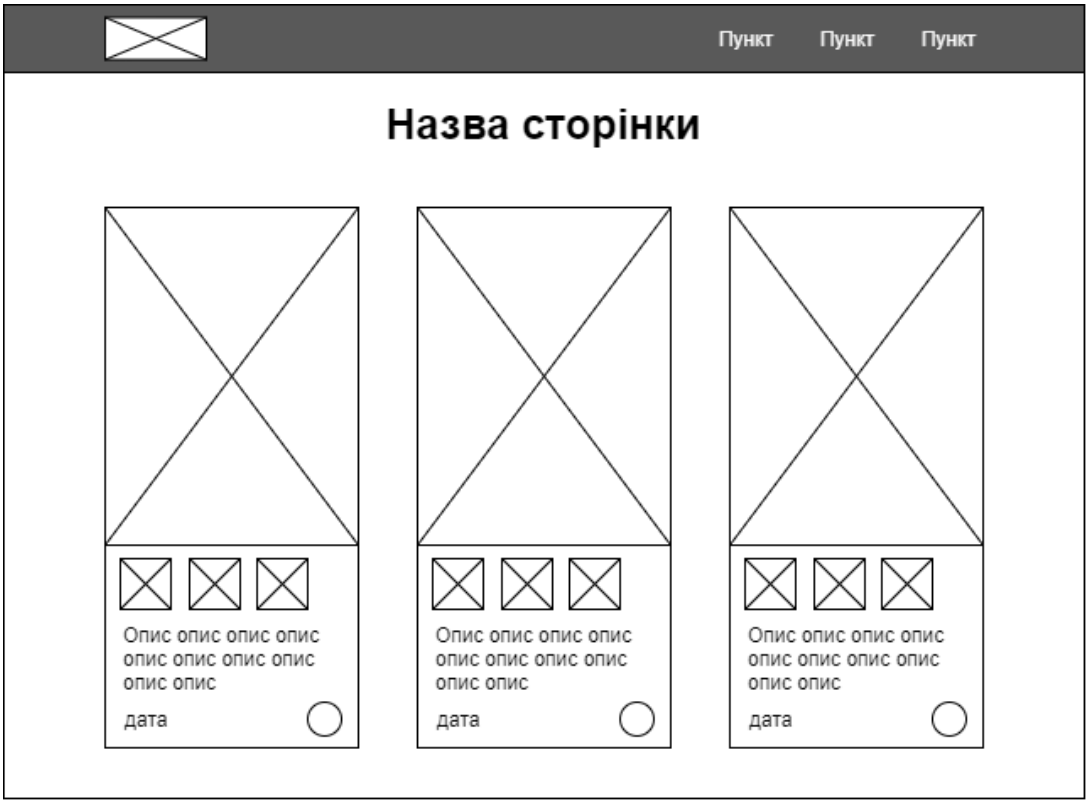


Рисунок 4.1 – «Мокап» головної сторінки, сторінки з результатами пошуку та сторінки вподобаних образів користувача

На рисунку 4.2 зображений «мокап» сторінок реєстрації та входу до свого облікового запису.

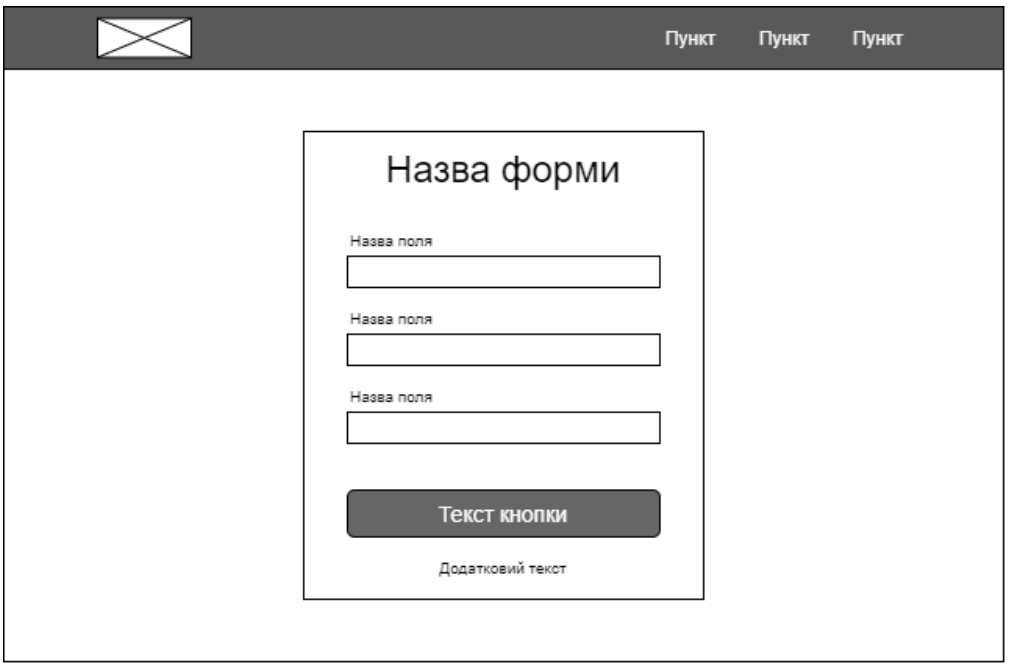


Рисунок 4.2 – «Мокап» сторінок реєстрації та входу до свого облікового запису

На рисунку 4.3 зображено «мокап» сторінки пошуку.

Рисунок 4.3 – «Мокап» сторінки пошуку

Так як при розробці клієнтської частити за допомогою фреймворку Svelte є можливість використовувати компоненти, було вирішено виокремити наступні сторінки вебсайту та обернути їх у компоненти (рис. 4.4):

- Index.svelte – компонент головної сторінки сайту;
- Register.svelte – компонент сторінки реєстрації користувачів;
- Login.svelte – компонент сторінки авторизації користувачів;
- Search.svelte – компонент сторінки пошуку образів;
- SeacrhResult.svelte – компонент сторінки з результатами пошуку образів;
- FavoriteCharacters.svelte – компонент сторінки обраних образів користувача;
- NotFound404.svelte – компонент сторінки при помилці 404;
- Oops.svelte – компонент сторінки при інших помилках.

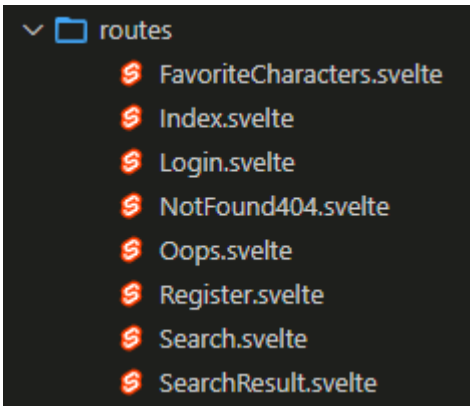


Рисунок 4.4 – Компоненти вебсторінок

Дублювання – головний ворог добре спроєктованої системи. Наслідки дублювання коду – зайва робота, зайвий ризик і зайва надлишкова складність. Дублювання проявляється у багатьох формах. Звичайно, точний збіг рядків коду свідчить про дублювання. Схожі рядки часто вдається переписати так, щоб схожість стала ще більш очевидним, це спростить рефакторинг коду. Крім того, дублювання може існувати і в інших формах – таких, як дублювання реалізації [17].

Ті елементи, які найчастіше повторюються на сторінках було виокремлено у окремі допоміжні компоненти (рис. 4.5):

- Header.svelte – компонент верхньої планки сторінок сайту;
- ClothesBox.svelte – компонент блоку з елементами одягу;
- CharactersCard.svelte – компонент з блоками образів.

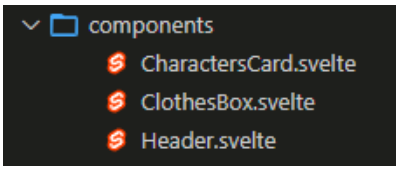


Рисунок 4.5 – Допоміжні компоненти

Остаточний вигляд інтерфейсу головної сторінки зображений на рисунку 4.6.

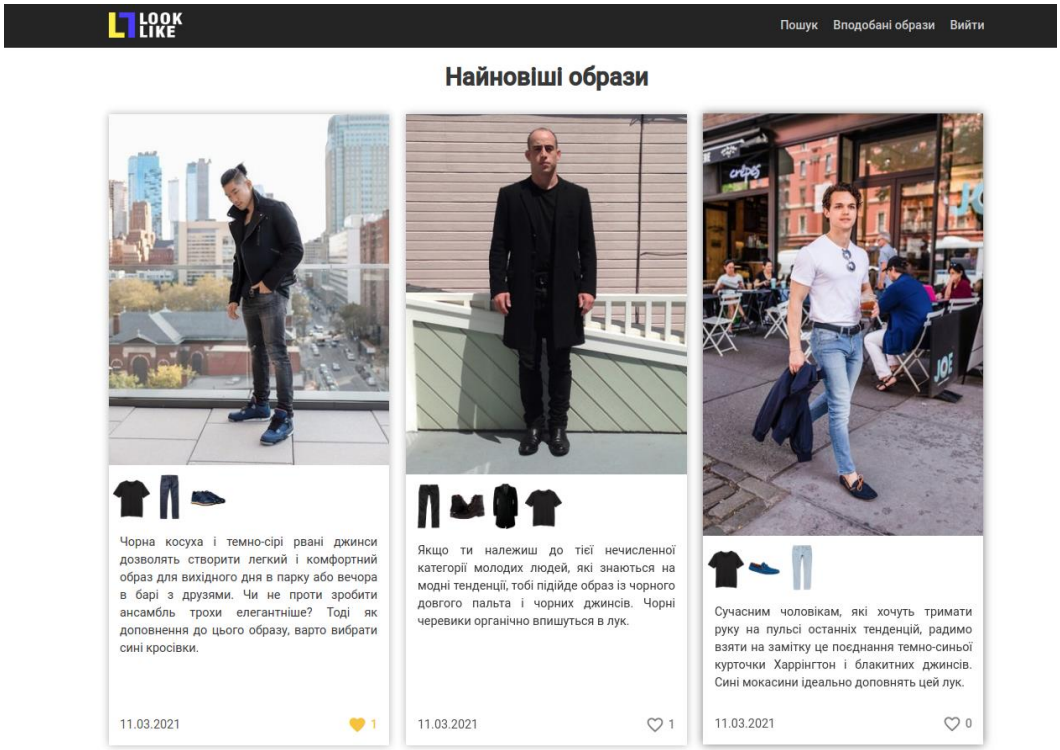


Рисунок 4.6 – Інтерфейс головної сторінки сайту

Остаточний вигляд інтерфейсу сторінки пошуку образів зображений на рисунку 4.7.

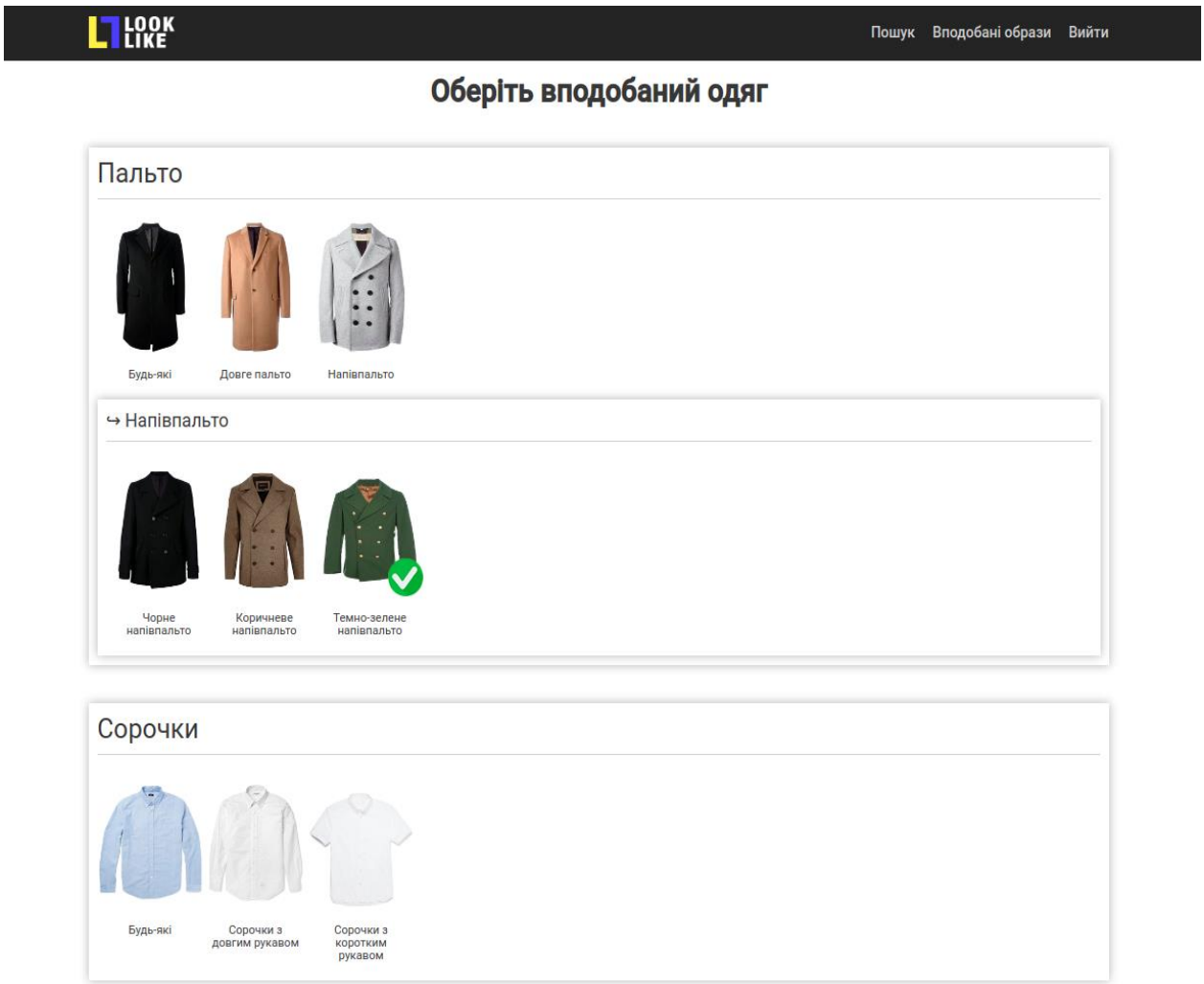


Рисунок 4.7 – Інтерфейс сторінки пошуку образів

Остаточний вигляд інтерфейсу сторінки вподобаних образів зображений на рисунку 4.8.

Остаточний вигляд інтерфейсу сторінки реєстрації зображений на рисунку 4.9.

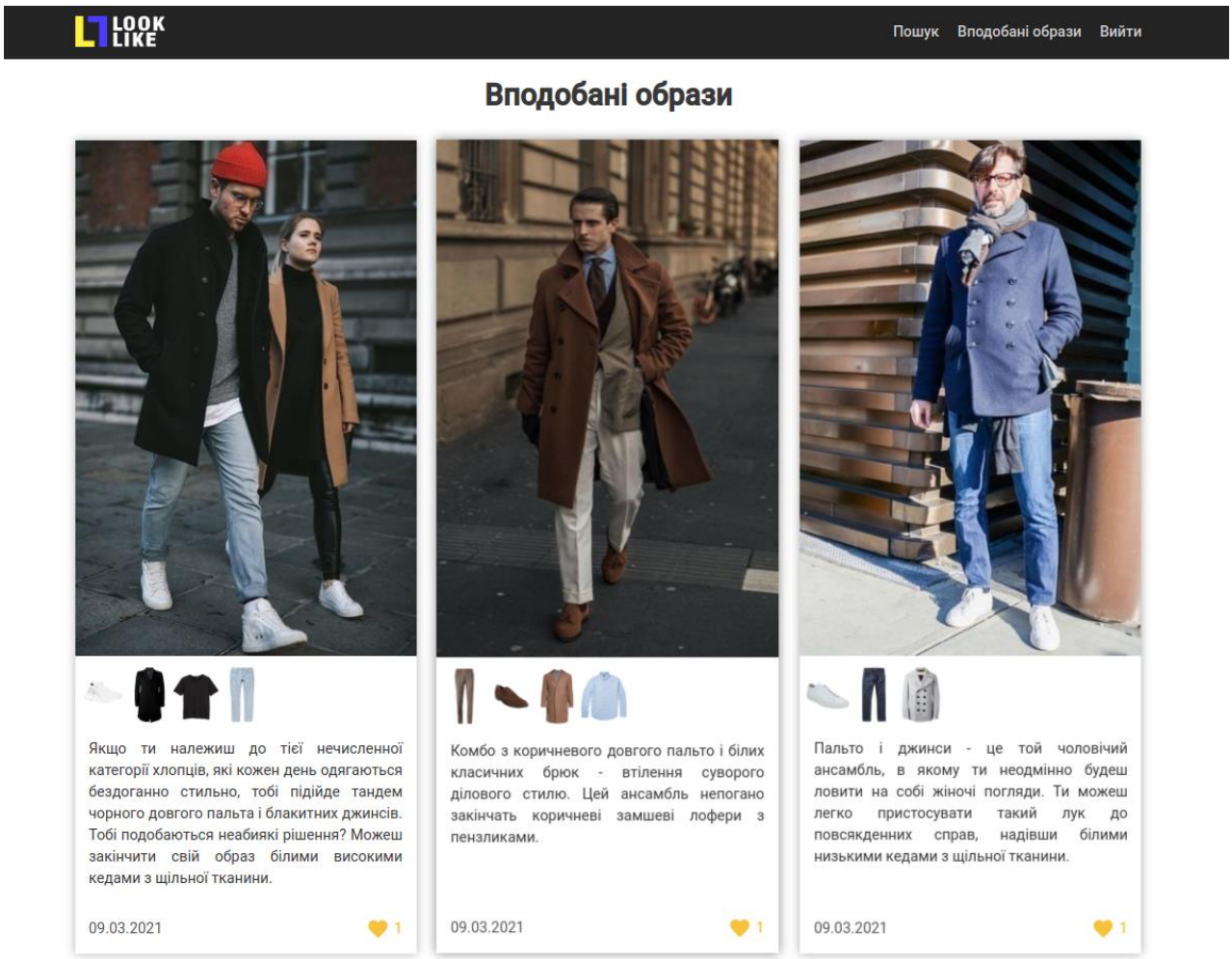


Рисунок 4.8 – Інтерфейс сторінки вподобаних образів користувача

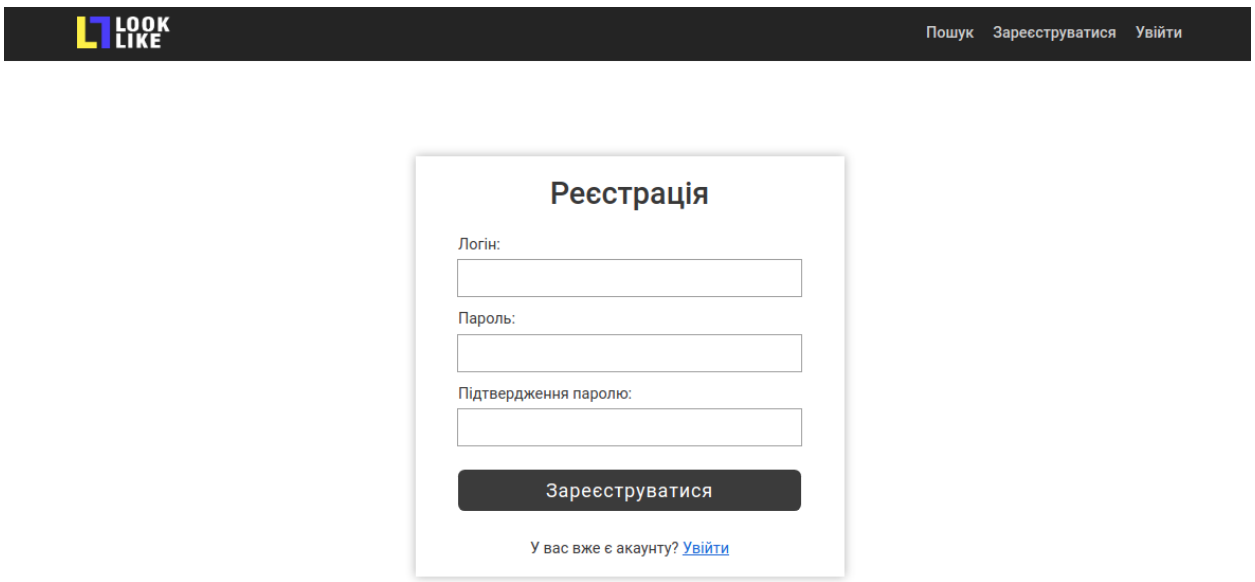


Рисунок 4.9 – Інтерфейс сторінки реєстрації користувачів

Остаточний вигляд інтерфейсу сторінки авторизації користувача зображений на рисунку 4.10.

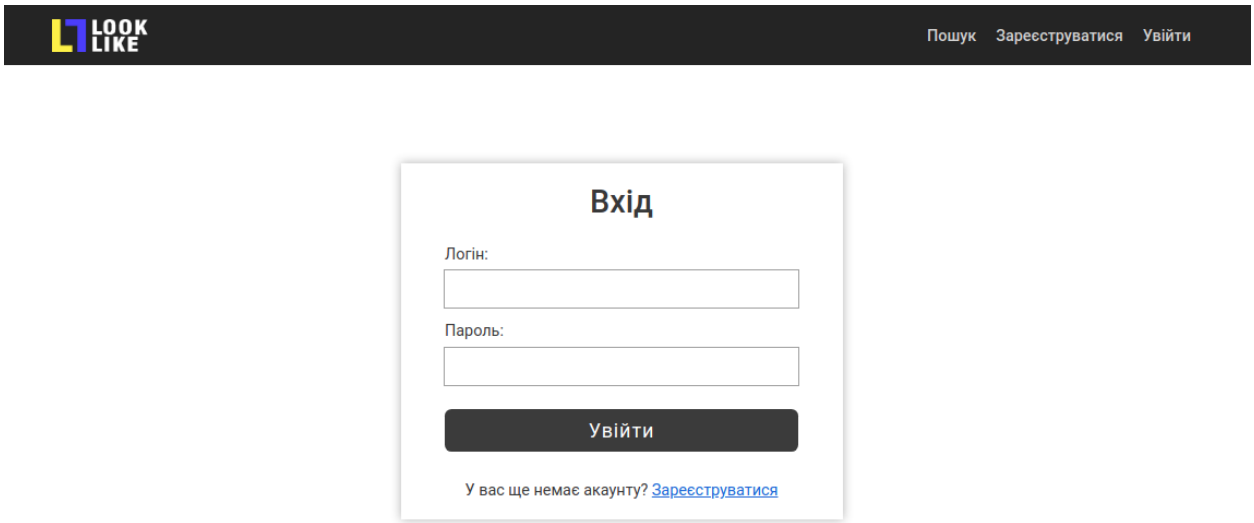


Рисунок 4.10 – Інтерфейс сторінки авторизації користувачів

4.2 Налаштування взаємодії з серверною частиною

Для взаємодії клієнтської частини з серверної необхідно відправляти HTTP запити.

Fetch API надає інтерфейс JavaScript для роботи із запитами та відповідями HTTP. Він також надає глобальний метод fetch(), який дозволяє легко і логічно отримувати ресурси по мережі асинхронно.

Подібна функціональність раніше досягалася за допомогою XMLHttpRequest. Fetch є кращою альтернативою, яка може бути легко використана іншими технологіями [18].

Дані будуть передаватися у форматі JSON – це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення в написанні вебпрограм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти [19].

Приклад коду запиту до серверної частини, який повертає список елементів одягу написаний мовою JavaScript наведений нижче:

```
const response = await fetch('/api/v1/clothes', {
   method: 'GET',
   headers: {
      'Accept': 'application/json'
   }
});
let data = await response.json();
```

4.3 Висновки за розділом 4

Було спроєктовано та розроблено клієнтську частину проєкту. Був спроєктований клієнтський інтерфейс, який є інтуїтивно зрозумілим та відповідає сучасним нормам розробки дизайну сайтів. Налагоджена взаємодія між клієнтською та серверною частинами за допомогою Fetch API, який керує відправкою HTTP запитів та отриманням на них відповідей.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

5.1 Тестування серверної частини

Для проведення тестування серверної частини, а саме правильної роботи HTTP запитів та відповідей на них, було вирішено обрати спеціальну програму – Postman.

Postman – це сучасний та зручний HTTP-клієнт для тестування вебсайтів. Postman надає можливість складати і редагувати HTTP-запити різної складності. Усі виконані запити автоматично зберігаються у пам’яті програми для повторного застосування. Відповіді на запити також, при потребі, можна зберігати як файли на жорсткому диску.

Для того, щоб бути впевненим у тому, що серверна частина працює як слід, необхідно протестувати усі її URL адреси.

На рисунку 5.1 зображений приклад тестування URL адреси api/v1/clothes. Після виконання HTTP-запиту з методом GET було отримано перелік усіх елементів одягу у БД у форматі JSON. Отримані дані цілком збігаються з існуючими даними у БД.

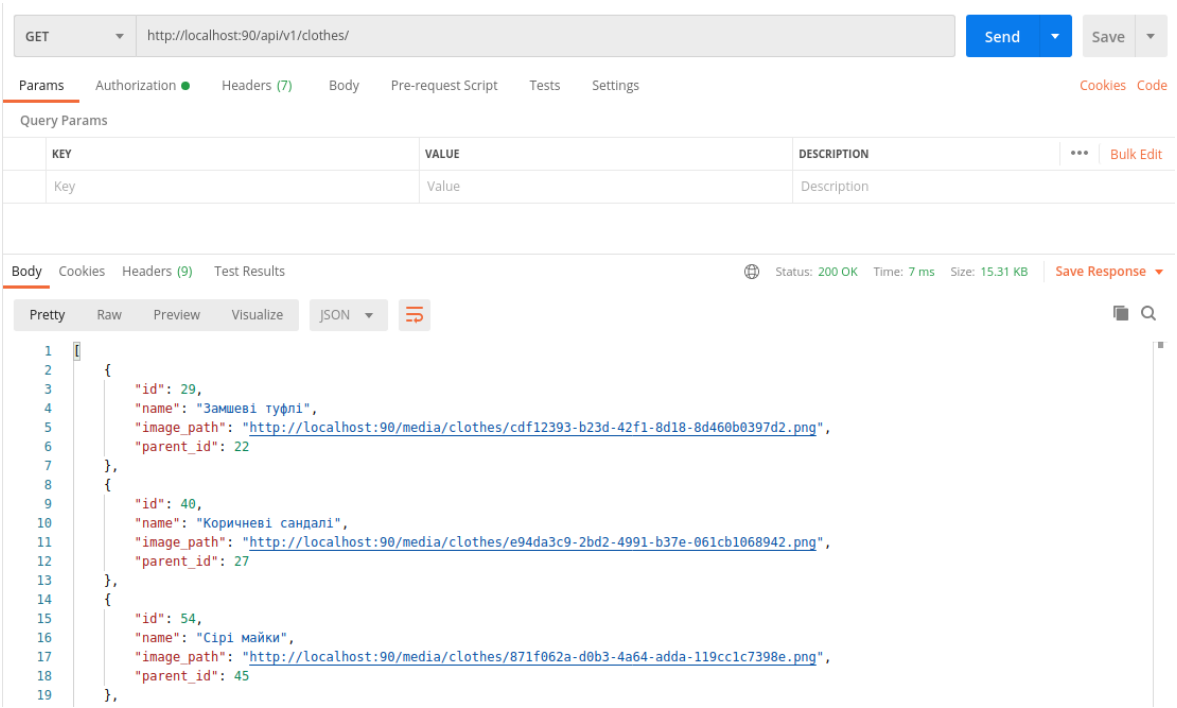


Рисунок 5.1 – Результат тестування HTTP-запиту

Після проведення тестування усіх URL адрес серверної частини можна зробити вивід, що серверна частина працює стабільно та саме так, як було задумано.

5.2 Тестування клієнтської частини

Для тестування клієнтської частини було вирішено використовувати вбудований у браузер Google Chrome інструмент розробки та тестування Google Lighthouse.

Google Lighthouse – це інструмент аудиту з відкритим вихідним кодом, який допомагає розробникам підвищити продуктивність і доступність своїх вебпроєктів. Будь-який бажаючий може використовувати його безкоштовно, щоб побачити, як його вебсайт відповідає високим стандартам Google в веб розробці [20].

Результати після запуску та проходження тестування розробленого сайту наведено на рисунку 5.2 та у таблиці 5.1.

Таблиця 5.1 – Результати тестування розробленого сайту за допомогою Google Lighthouse

Характеристика	Отриманий бал	Максимальний бал
Продуктивність	100	100
Спеціальні можливості	91	100
Рекомендації	93	100
Пошукова оптимізація	100	100

Виходячи з отриманих оцінок можна зробити висновок, що клієнтська частина повністю відповідає сучасним критеріям веброзробки.

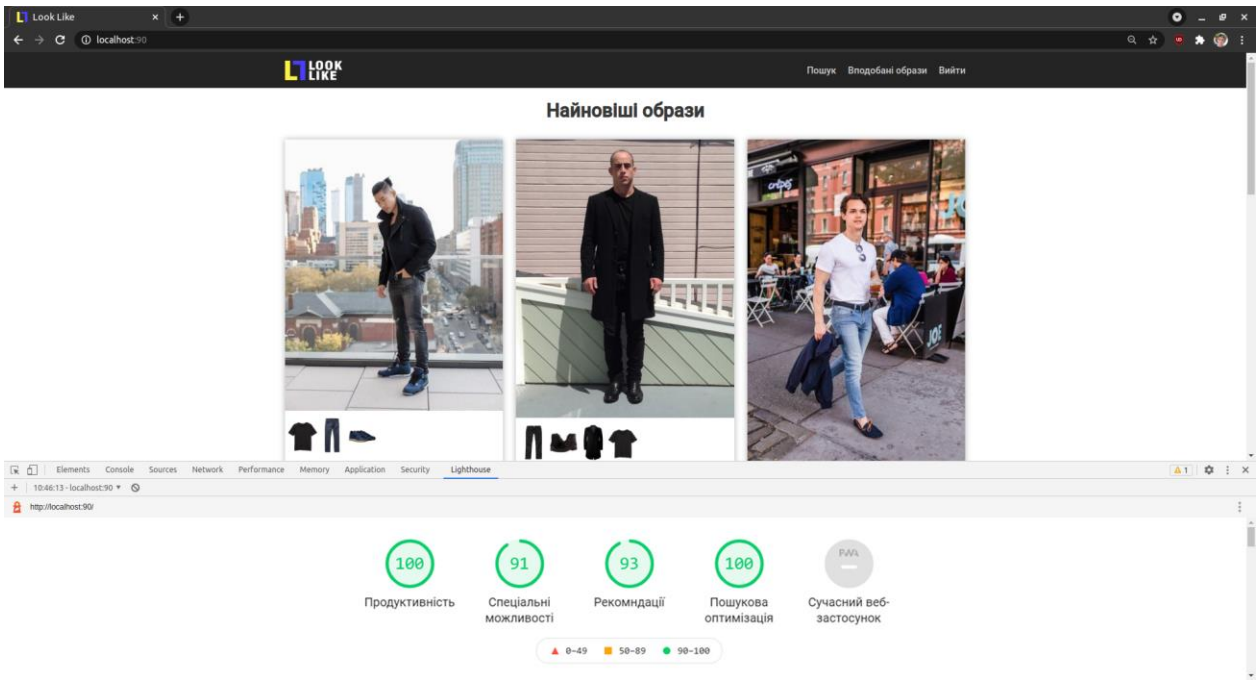


Рисунок 5.2 – Результати тестування розробленого сайту за допомогою Google Lighthouse

5.3 Висновки за розділом 5

Було проведено тестування серверної та клієнтської частин, використовуючи програми Postman та Google Lighthouse, на наявність помилок чи інших можливих збоїв.

За результатами тестування розроблений вебзастосунок працює стабільно та придатний для використання користувачами.

ВИСНОВКИ

Під час виконання цієї дипломної роботи було спроєктовано вебзастосунок, який надає можливість переглядати та виконувати пошук готових образів (фотокарток моделей) на основі обраних користувачем елементів одягу.

Після визначення задач було знайдено та проаналізовано існуючі аналоги у мережі Інтернет, визначено їх переваги та недоліки. Далі був проведений аналіз існуючих вебтехнологій та методів створення вебдодатків. Було вирішено розробляти вебзастосунок використовуючи технологію SPA, тим самим більш явно розділивши систему на серверну та клієнтську частини, таким чином було отримано ряд переваг.

В результаті проведеного аналізу були обрані мови програмування, а саме Python для серверної частини та JavaScript для клієнтської частини. В якості вебфреймворків було обрано Flask та Svelte. В якості системи керування базами даних було обрано реляційну БД PostgreSQL та допоміжну NoSQL БД – Redis. Наступним кроком було проєктування архітектури серверної та клієнтської частин та подальша їх реалізація за допомогою обраних технологій та інструментів.

Після реалізації системи було проведено тестування серверної та клієнтської частин на наявність помилок чи інших можливих збоїв. За результатами тестування розроблений вебзастосунок працює стабільно та придатний для використання користувачами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мода и стиль [Электрон. ресурс]. – Режим доступа: https://image66.ru/blog/sovety_stilistov/moda-i-stil.html.

2. Клієнт-серверна архітектура [Електрон. ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture.

3. Миковски, М.С. Разработка одностраничных веб-приложений / М.С. Миковски, Д.К. Пауэлл; пер. с англ. А. Слинкина. – М.: ДМК Пресс, 2014. – 512 с.

4. Choose Between Traditional Web Apps and Single Page Apps [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps.

5. 8 основных языков для бэкенда [Электрон. ресурс]. – Режим доступа: https://tproger.ru/translations/7-basic-languages-for-backend-development.

6. Вікіпедія: Python [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Python.

7. Какой Python-фреймворк учить в 2020 году? [Электрон. ресурс]. – Режим доступа: https://igorosa.com/what-python-web-framework-to-learn-in-2020.

8. ТОП 9 фреймворков для Python-разработчиков [Электрон. ресурс]. – Режим доступа: https://web-academy.com.ua/stati/340-9-python.

9. Рассуждение на тему, какую базу данных выбирать [Электрон. ресурс]. – Режим доступа: https://habr.com/ru/post/348220.

10. Вікіпедія: React [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/React.

11. Вікіпедія: Vue [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Vue.js.

12. Svelte 3: Переосмысление реактивности [Электрон. ресурс]. – Режим доступа: https://habr.com/ru/post/449450.

13. Вікіпедія: Нормалізація баз даних [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Нормалізація_баз_даних.

14. Моргунов, Е.П. PostgreSQL. Основы языка SQL / Е.П. Моргунов. – СПБ.: БХВ-Петербург, 2018. – 336 с.

15. Шпаргалки по безопасности: JWT [Электрон. ресурс]. – Режим доступа: https://habr.com/ru/company/acribia/blog/457090.

16. Аутентификация с помощью JSON Web Token [Электрон. ресурс]. – Режим доступа: https://codex.so/jwt.

17. Мартин, Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин. – СПБ.: Питер, 2019. – 464 с.

18. Использование Fetch [Электрон. ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API/Using_Fetch.

19. Вікіпедія: JSON [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/JSON.

20. Использование Google Lighthouse для аудита веб-приложений [Электрон. ресурс]. – Режим доступа: https://webdevblog.ru/ispolzovanie-google-lighthouse-dlya-audita-veb-prilozhenij.

ДОДАТОК А

Код серверної частини

А.1 Код файлу looklike/blueprints/characters_routes.py

```
from typing import Optional
from flask import Blueprint, request, json as flask_json
from looklike.configs import config
from looklike.db_helper import CharactersDBHelper
from looklike.exceptions import (
    ObjectNotFoundException,
    CharacterAlreadyInFavorites
)
from looklike.serializers import CharactersWithClothesSerializer
from looklike.redis_client import redis_client, format_cache_key
from looklike.routes_decorators import authorized_required, authorized_optional
from looklike.utils import get_ids_from_string, jsoning

url_prefix = '/api/v1/characters/'
characters_bp = Blueprint('characters_bp', __name__, url_prefix=url_prefix)

@characters_bp.route('', methods=['GET'])
@authorized_optional
def get_characters(user_id: Optional[int] = None):
    # URL example:  /api/v1/characters
    all_characters = CharactersDBHelper.get_all_characters(
        user_id=user_id,
        with_clothes=True
    )
    return jsoning(CharactersWithClothesSerializer.serialize(all_characters))

@characters_bp.route('<int:character_id>', methods=['GET'])
def get_character(character_id):
    # URL example:  /api/v1/characters/15
    try:
        character = CharactersDBHelper.get_character_by_id(
            character_id=character_id,
            with_clothes=True
        )
    except ObjectNotFoundException as e:
        return jsoning({'message': str(e)}), 404

    return jsoning(CharactersWithClothesSerializer.serialize(character))

@characters_bp.route('favorites', methods=['GET'])
@authorized_required
def get_favorite_characters(user_id: int):
    # URL example:  /api/v1/characters/favorites
    try:
        characters = CharactersDBHelper.get_favorites(
            user_id=user_id,
            with_clothes=True
        )
    except ObjectNotFoundException as e:
        return jsoning({'message': str(e)}), 404
    return jsoning(CharactersWithClothesSerializer.serialize(characters))

@characters_bp.route('filter', methods=['GET'])
@authorized_optional
def get_filtered_characters(user_id: Optional[int] = None):
    # URL example:  /api/v1/characters/filter?type=newness&limit=10
    filter_type = request.args.get('type')
    filter_limit = request.args.get('limit')
```

```
    if filter_type == 'newness':
        if filter_limit:
            try:
                filter_limit = int(filter_limit)
            except ValueError:
                return jsoning(
                    {'message': 'Argument \'filter_limit\' must be integer!'}
                ), 400

            filtered_characters = CharactersDBHelper.get_newest_characters(
                user_id=user_id,
                limit=filter_limit,
                with_clothes=True
            )
        else:
            filtered_characters = CharactersDBHelper.get_newest_characters(
                user_id=user_id,
                with_clothes=True
            )

        return jsoning(
            CharactersWithClothesSerializer.serialize(filtered_characters)
        )
    return jsoning({'message': 'Filter type not allowed!'})

@characters_bp.route('find', methods=['GET'])
@authorized_optional
def find_characters_by_clothes(user_id: Optional[int] = None):
    # URL example:  /api/v1/characters/find?clothes=[2,9]
    clothes_list = request.args.get('clothes')

    if clothes_list:
        try:
            clothes_ids = get_ids_from_string(clothes_list)
        except ValueError:
            return jsoning({
                'message':
                    'Argument \'clothes\' must be array of integers!'
            }), 400

        try:
            if config.REDIS_CACHING:
                cache_key = format_cache_key(clothes_ids)
                cache_value = redis_client.get(cache_key)
                if cache_value:
                    response_text = cache_value.decode('utf-8')
                    return response_text, 200, {
                        'Content-Type': 'application/json'
                    }

            characters = CharactersDBHelper.get_characters_by_clothes(
                clothes_ids=clothes_ids,
                user_id=user_id,
                with_clothes=True
            )
        except ObjectNotFoundException as e:
            return jsoning({'message': str(e)}), 404
        serialized = CharactersWithClothesSerializer.serialize(characters)

        if config.REDIS_CACHING:
            new_cache_value = flask_json.dumps(serialized)
            redis_client.set(
                name=cache_key,
```

```
            value=new_cache_value,
            ex=config.REDIS_CACHE_EXPIRE
        )
    return jsoning(serialized)

  return jsoning({
    'message':
      'You must add query parameter \'clothes\' with array of '
      'clothes IDs!'
  })

@characters_bp.route('<int:character_id>/add-to-favorites', methods=['POST'])
@authorized_required
def add_character_to_favorites(character_id: int, user_id: int):
  # URL example:  /api/v1/characters/17/add-to-favorites
  try:
    CharactersDBHelper.add_to_favorites(
      character_id=character_id,
      user_id=user_id
    )
  except ObjectNotFoundException as e:
    return jsoning({'message': str(e)}), 404
  except CharacterAlreadyInFavorites as e:
    return jsoning({'message': str(e)}), 400
  return jsoning({'message': 'Character added to favorites'})

@characters_bp.route(
  '<int:character_id>/remove-from-favorites',
  methods=['POST']
)
@authorized_required
def remove_character_from_favorites(character_id: int, user_id: int):
  # URL example:  /api/v1/characters/17/remove-from-favorites
  try:
    CharactersDBHelper.remove_from_favorites(
      character_id=character_id,
      user_id=user_id
    )
  except ObjectNotFoundException as e:
    return jsoning({'message': str(e)}), 404
  return jsoning({'message': 'Character removed from favorites'})
```

A.2 Код файлу looklike/blueprints/clothes_routes.py

```
from flask import Blueprint
from looklike.db_helper import ClothesDBHelper
from looklike.serializers import ClothesSerializer
from looklike.utils import jsoning

url_prefix = '/api/v1/clothes/'
clothes_bp = Blueprint('clothes_bp', __name__, url_prefix=url_prefix)

@clothes_bp.route('', methods=['GET'])
def get_clothes():
  # URL example:  /api/v1/clothes
  all_clothes = ClothesDBHelper.get_all_clothes()
  return jsoning(ClothesSerializer.serialize(all_clothes))

@clothes_bp.route('primary', methods=['GET'])
def get_primary_clothes_only():
```

```
    # URL example:  /api/v1/clothes/primary
    primary_clothes = ClothesDBHelper.get_primary_clothes()
    result = ClothesSerializer.serialize_primary_only(primary_clothes)
    return jsoning(result)

@clothes_bp.route('by-parent-id/<int:parent_id>', methods=['GET'])
def get_clothes_by_parent_id(parent_id: int):
    # URL example:  /api/v1/clothes/by-parent-id/10
    clothes_by_parent = ClothesDBHelper.get_clothes_by_parent_id(parent_id)
    return jsoning(ClothesSerializer.serialize(clothes_by_parent))
```

А.3 Код файлу looklike/blueprints/users_routes.py

```
from flask import Blueprint, request
from looklike.authorizations import JWTAuthorization
from looklike.configs import config
from looklike.db_helper import UsersDBHelper
from looklike.exceptions import (
    ObjectNotFoundException,
    UserAlreadyExistsException
)
from looklike.models import UserRegistration, UserLogin
from looklike.utils import jsoning

url_prefix = '/api/v1/users/'
users_bp = Blueprint('users_bp', __name__, url_prefix=url_prefix)

@users_bp.route('register', methods=['POST'])
def register():
    """Registers a user and returns it"""
    data = UserRegistration.parse_raw(request.data)

    try:
        user = UsersDBHelper.create_user(data)
    except UserAlreadyExistsException as e:
        return jsoning({'message': str(e)}), 409

    return jsoning(user.json(exclude={'password_hash'})), 201
@users_bp.route('login', methods=['POST'])
def login():
    """Authorizes the user and returns access token"""
    data = UserLogin.parse_raw(request.data)

    try:
        user = UsersDBHelper.get_user_by_username(data.username)
    except ObjectNotFoundException as e:
        return jsoning({"message": str(e)}), 404

    auth = JWTAuthorization(secret_key=config.SECRET_KEY)

    if not auth.is_correct_password(data.password, user.password_hash):
        return jsoning({'message': 'Invalid password!'}), 401

    token = auth.create_token(data={'user_id': user.id})
    return jsoning({'access_token': token})
```

А.4 Код файлу looklike/__init__.py

```
from flask import Flask
from pydantic import ValidationError

from looklike.configs import config
from looklike.blueprints.clothes_routes import clothes_bp
from looklike.blueprints.characters_routes import characters_bp
from looklike.blueprints.users_routes import users_bp
from looklike.utils import jsoning

def create_app() -> Flask:
    app = Flask(__name__)
    app.config.from_object(config)

    app.register_blueprint(clothes_bp)
    app.register_blueprint(characters_bp)
    app.register_blueprint(users_bp)

    @app.errorhandler(ValidationError)
    def handle_validation_error(e):
        return jsoning(e.errors()), 400

    return app
```

А.5 Код файлу looklike/authorizations.py

```
from typing import Optional
from hashlib import pbkdf2_hmac
import jwt
from looklike.exceptions import AuthorizationException

class JWTAuthorization:
    def __init__(
        self, secret_key,
        hash_method='sha256',
        hash_iterations=10000,
        jwt_algorithm='HS256'
    ):
        self.secret_key = secret_key
        self.hash_method = hash_method
        self.hash_iterations = hash_iterations
        self.jwt_algorithm = jwt_algorithm

    def generate_password_hash(self, password: str) -> str:
        """Creates a hashed password from a raw password"""
        return pbkdf2_hmac(
            hash_name=self.hash_method,
            password=password.encode('utf-8'),
            salt=self.secret_key.encode('utf-8'),
            iterations=self.hash_iterations
        ).hex()

    def is_correct_password(self, password: str, hash: str) -> bool:
        """Compares the raw password with a hashed password"""
        return self.generate_password_hash(password) == hash

    def create_token(self, data: dict, exp: Optional[int] = None) -> str:
        """Creates and returns a JWT with the required data"""
```

```
        if exp:
            data['exp'] = exp

        return jwt.encode(
            payload=data,
            key=self.secret_key,
            algorithm=self.jwt_algorithm
        )

    def get_acces_token_from_headers(self, headers: dict) -> str:
        """Retrieves and returns a token from the request headers"""
        auth_data = headers.get('Authorization')

        if not auth_data:
            raise AuthorizationException(
                'You should add the \'Authorization\' header to your request!'
            )

        if not auth_data.startswith('Bearer '):
            raise AuthorizationException(
                'The \'Authorization\' header must start with \'Bearer \'!'
            )

        return auth_data[7:]
    def get_user_id_from_acces_token(self, token: str) -> int:
        """Retrieves and returns user_id from token"""
        try:
            data = jwt.decode(
                jwt=token,
                key=self.secret_key,
                algorithms=self.jwt_algorithm
            )
            return data['user_id']
        except KeyError:
            raise AuthorizationException(
                'Can\'t extract \'user_id\' from token!'
            )
        except jwt.exceptions.ExpiredSignatureError:
            raise AuthorizationException(
                'You are using an expired token!'
            )
        except jwt.exceptions.DecodeError:
            raise AuthorizationException(
                'Token cannot be decoded because it failed validation!'
            )
        except jwt.exceptions.InvalidSignatureError:
            raise AuthorizationException(
                ('Token’s signature doesn’t match the one provided as part of '
                 'the token!')
            )
        except jwt.exceptions.InvalidTokenError:
            raise AuthorizationException(
                'Token cannot be decoded!'
            )
```

A.6 Код файлу looklike/configs.py

```
import os
import sys
from dotenv import load_dotenv
```

```
# Load all values from .env file
load_dotenv()

class BaseConfig:
    SECRET_KEY = os.getenv('SECRET_KEY')
    POSTGRESQL_URL = os.getenv('POSTGRESQL_URL')
    REDIS_URL = os.getenv('REDIS_URL')
    REDIS_CACHING = True
    REDIS_CACHE_EXPIRE = 10 * 60
    AVAILABLE_HOSTS = '*'
    JSON_SORT_KEYS = False
    MEDIA_URL = os.getenv('MEDIA_URL')

class DevConfig(BaseConfig):
    DEBUG = True

class WithOutCachingDevConfig(DevConfig):
    REDIS_CACHING = False

class ProdConfig(BaseConfig):
    DEBUG = False

config_class_name = os.getenv('APP_CONFIG_CLASS')
config = getattr(sys.modules[__name__], config_class_name)
```

A.7 Код файлу looklike/database.py

```
from contextlib import contextmanager
from urllib.parse import urlparse
from psycopg2.pool import ThreadedConnectionPool
from psycopg2.extras import RealDictCursor
from looklike.configs import config

url = urlparse(config.POSTGRESQL_URL)

connection_pool = ThreadedConnectionPool(
    minconn=1,
    maxconn=50,
    database=url.path[1:],
    user=url.username,
    password=url.password,
    host=url.hostname,
    port=url.port
)

@contextmanager
def get_db_connection():
    try:
        connection = connection_pool.getconn()
        yield connection
    finally:
        connection_pool.putconn(connection)

@contextmanager
def get_db_cursor(commit: bool = False):
    with get_db_connection() as connection:
        cursor = connection.cursor(cursor_factory=RealDictCursor)
        try:
            yield cursor
```

```
            if commit:
                connection.commit()
        finally:
            cursor.close()
```

A.8 Код файлу looklike/db_helper.py

```
from typing import Optional
from looklike.authorizations import JWTAuthorization
from looklike.configs import config
from looklike.database import get_db_cursor
from looklike.exceptions import (
    ObjectNotFoundException,
    UserAlreadyExistsException,
    CharacterAlreadyInFavorites
)
from looklike.models import (
    Clothes, Character,
    User, UserRegistration
)

class ClothesDBHelper:
    @staticmethod
    def get_all_clothes() -> list[Clothes]:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT id, name, image_path, parent_id, parent_path FROM '
                'all_clothes;'
            )
            data = cursor.fetchall()

        all_clothes = [Clothes(**item) for item in data]
        return all_clothes

    @staticmethod
    def get_clothes_by_parent_id(parent_id: int) -> list[Clothes]:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT id, name, image_path, parent_id, parent_path FROM '
                'all_clothes WHERE parent_id = %s ORDER BY '
                'display_priority;',
                (parent_id,)
            )
            data = cursor.fetchall()

        clothes_by_parent = [Clothes(**item) for item in data]
        return clothes_by_parent

    @staticmethod
    def get_primary_clothes() -> list[Clothes]:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT id, name, image_path, parent_id, parent_path FROM '
                'all_clothes WHERE parent_path ~ \'*{,2}\' ORDER BY '
                'display_priority;'
            )
            data = cursor.fetchall()

        primary_clothes = [Clothes(**item) for item in data]
        return primary_clothes
```

```
    @staticmethod
    def get_clothes_parent_paths(clothes_ids: list[int]) -> list[str]:
        with get_db_cursor() as cursor:
            query = 'SELECT parent_path FROM all_clothes WHERE id = ANY(%s)'
            cursor.execute(query, (clothes_ids,))
            data = cursor.fetchall()

        parent_paths = [item['parent_path'] for item in data]
        return parent_paths

    @staticmethod
    def get_character_clothes(character: Character) -> list[Clothes]:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT id, name, image_path, parent_id, parent_path FROM '
                'all_clothes WHERE id IN(SELECT clothes_id FROM '
                'clothes_on_characters WHERE character_id = %s) ORDER BY '
                'display_priority',
                (character.id,)
            )
            data = cursor.fetchall()

        clothes = [Clothes(**item) for item in data]
        return clothes

class CharactersDBHelper:
    @staticmethod
    def get_favorites(
        user_id: int,
        with_clothes: bool = False
    ) -> list[Character]:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT c.id, c.author_id, c.image_path, c.description, '
                'c.posted_at, COUNT(fcou.character_id) AS likes FROM '
                'characters c LEFT JOIN favorite_characters_of_users fcou ON '
                'fcou.character_id = c.id WHERE c.id IN(SELECT character_id '
                'FROM favorite_characters_of_users WHERE user_id = %s) GROUP '
                'BY c.id;',
                (user_id,)
            )
            data = cursor.fetchall()

        favorite_characters = [Character(**item) for item in data]

        if with_clothes:
            CharactersDBHelper._inject_clothes_to_characters(
                favorite_characters
            )

        if user_id:
            CharactersDBHelper._inject_is_favorite_field(
                user_id=user_id,
                characters=favorite_characters
            )

        return favorite_characters

    @staticmethod
    def add_to_favorites(character_id: int, user_id: int) -> None:
        with get_db_cursor() as cursor:
            cursor.execute(
```

```
            'SELECT EXISTS (SELECT 1 FROM characters WHERE id = %s);',
            (character_id,)
        )
        data = cursor.fetchone()

    if not data.get('exists'):
        raise ObjectNotFoundException(
            f'Character with id={character_id} not found!'
        )

    if CharactersDBHelper.is_favorite_character(user_id, character_id):
        raise CharacterAlreadyInFavorites(
            'The character already exists in favorites'
        )

    with get_db_cursor(commit=True) as cursor:
        cursor.execute(
            'INSERT INTO favorite_characters_of_users (character_id, '
            'user_id) VALUES (%s, %s);',
            (character_id, user_id)
        )

@staticmethod
def remove_from_favorites(character_id: int, user_id: int) -> None:
    if not CharactersDBHelper.is_favorite_character(user_id, character_id):
        raise ObjectNotFoundException(
            'This character was not found in favorites!'
        )

    with get_db_cursor(commit=True) as cursor:
        cursor.execute(
            'DELETE FROM favorite_characters_of_users WHERE character_id '
            '= %s AND user_id = %s;',
            (character_id, user_id)
        )

@staticmethod
def get_all_characters(
    user_id: Optional[int] = None,
    with_clothes: bool = False
) -> list[Character]:
    with get_db_cursor() as cursor:
        cursor.execute(
            'SELECT c.id, c.author_id, c.image_path, c.description, '
            'c.posted_at, COUNT(fcou.character_id) AS likes FROM '
            'characters c LEFT JOIN favorite_characters_of_users fcou ON '
            'fcou.character_id = c.id GROUP BY c.id;'
        )
        data = cursor.fetchall()

    all_characters = [Character(**item) for item in data]

    if with_clothes:
        CharactersDBHelper._inject_clothes_to_characters(all_characters)

    if user_id:
        CharactersDBHelper._inject_is_favorite_field(
            user_id=user_id,
            characters=all_characters
        )

    return all_characters
```

```
@staticmethod
def get_character_by_id(
    character_id: int,
    with_clothes: bool = False
) -> Character:
    with get_db_cursor() as cursor:
        cursor.execute(
            'SELECT c.id, c.author_id, c.image_path, c.description, '
            'c.posted_at, COUNT(fcou.character_id) AS likes FROM '
            'characters c LEFT JOIN favorite_characters_of_users fcou ON '
            'fcou.character_id = c.id WHERE c.id = %s GROUP BY c.id;',
            (character_id,)
        )
        data = cursor.fetchone()

    if not data:
        raise ObjectNotFoundException(
            f'Character with id={character_id} not found!'
        )

    character = Character(**data)

    if with_clothes:
        CharactersDBHelper._inject_clothes_to_characters([character])

    return character

@staticmethod
def get_newest_characters(
    user_id: Optional[int] = None,
    limit: int = 10,
    with_clothes: bool = False
) -> list[Character]:
    with get_db_cursor() as cursor:
        cursor.execute(
            'SELECT c.id, c.author_id, c.image_path, c.description, '
            'c.posted_at, COUNT(fcou.character_id) AS likes FROM '
            'characters c LEFT JOIN favorite_characters_of_users fcou ON '
            'fcou.character_id = c.id GROUP BY c.id ORDER BY c.posted_at '
            'DESC LIMIT %s;',
            (limit,)
        )
        data = cursor.fetchall()

    newest_characters = [Character(**item) for item in data]

    if with_clothes:
        CharactersDBHelper._inject_clothes_to_characters(newest_characters)

    if user_id:
        CharactersDBHelper._inject_is_favorite_field(
            user_id=user_id,
            characters=newest_characters
        )

    return newest_characters

@staticmethod
def get_characters_by_clothes(
    clothes_ids: list[int],
    user_id: Optional[int] = None,
    with_clothes: bool = False
) -> list[Character]:
```

```
    with get_db_cursor() as cursor:
        parent_paths = ClothesDBHelper.get_clothes_parent_paths(
            clothes_ids
        )

        if not parent_paths:
            raise ObjectNotFoundException(
                'One or more Clothes from the list were not found!'
            )

        query = CharactersDBHelper._format_specific_query(parent_paths)
        cursor.execute(query, parent_paths)
        data = cursor.fetchall()

    characters = [Character(**item) for item in data]

    if with_clothes:
        CharactersDBHelper._inject_clothes_to_characters(characters)

    if user_id:
        CharactersDBHelper._inject_is_favorite_field(
            user_id=user_id,
            characters=characters
        )

    return characters

@staticmethod
def is_favorite_character(user_id: int, character_id: int) -> bool:
    with get_db_cursor() as cursor:
        cursor.execute(
            'SELECT EXISTS (SELECT 1 FROM favorite_characters_of_users '
            'WHERE character_id = %s AND user_id = %s);',
            (character_id, user_id)
        )
        data = cursor.fetchone()

    return data.get('exists')

@staticmethod
def _inject_clothes_to_characters(characters: list[Character]):
    for character in characters:
        character.clothes = ClothesDBHelper.get_character_clothes(
            character
        )

@staticmethod
def _inject_is_favorite_field(user_id: int, characters: list[Character]):
    for character in characters:
        character.is_favorite = CharactersDBHelper.is_favorite_character(
            user_id=user_id,
            character_id=character.id
        )

@staticmethod
def _format_specific_query(parent_paths: list[str]) -> str:
    base_query = (
        'SELECT c.id, c.author_id, c.image_path, c.description, '
        'c.posted_at, COUNT(fcou.character_id) AS likes FROM characters c '
        'LEFT JOIN favorite_characters_of_users fcou ON fcou.character_id '
        '= c.id WHERE c.id IN(SELECT coc.character_id FROM '
        'clothes_on_characters coc WHERE coc.clothes_id IN(SELECT ac.id '
        'FROM all_clothes ac WHERE ac.parent_path <@ %s)) GROUP '
```

```
            'BY c.id'
        )

        for i, _ in enumerate(parent_paths):
            if i == 0:
                query = f'{base_query}'
            else:
                query = f'{query} INTERSECT {base_query}'

        return query


class UsersDBHelper:
    @staticmethod
    def get_user_by_username(username: str) -> User:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT id, username, password_hash, registered_at FROM '
                'users WHERE username = %s;',
                (username,)
            )
            data = cursor.fetchone()

        if not data:
            raise ObjectNotFoundException(
                'User with this name was not found!'
            )

        return User(**data)

    @staticmethod
    def is_user_exists(username: str) -> bool:
        with get_db_cursor() as cursor:
            cursor.execute(
                'SELECT EXISTS (SELECT 1 FROM users WHERE username = %s);',
                (username,)
            )
            data = cursor.fetchone()

        return data.get('exists')

    @staticmethod
    def create_user(user: UserRegistration) -> User:
        if UsersDBHelper.is_user_exists(user.username):
            raise UserAlreadyExistsException(
                'User with this name already exists!'
            )

        auth = JWTAuthorization(secret_key=config.SECRET_KEY)
        password_hash = auth.generate_password_hash(user.password)

        with get_db_cursor(commit=True) as cursor:
            cursor.execute(
                'INSERT INTO users (username, password_hash) VALUES (%s, %s) '
                'RETURNING id, registered_at',
                (user.username, password_hash)
            )
            data = cursor.fetchone()

        return User(
            id=data['id'],
            username=user.username,
            password_hash=password_hash,
```

```
            registered_at=data['registered_at']
        )
```

A.9 Код файлу looklike/exceptions.py

```
class ObjectNotFoundException(Exception):
    pass

class AuthorizationException(Exception):
    pass

class UserAlreadyExistsException(Exception):
    pass

class CharacterAlreadyInFavorites(Exception):
    pass
```

A.10 Код файлу looklike/models.py

```
from dataclasses import dataclass, field
from datetime import datetime
from typing import Optional
from pydantic import BaseModel, validator
from looklike.utils import datetime_to_timestamp

@dataclass
class Clothes:
    id: int
    name: str
    image_path: str
    parent_id: int
    parent_path: str

    def __str__(self):
        return f'<Clothes id={self.id} Name={self.name}>'

    def __repr__(self):
        return f'<Clothes id={self.id} Name={self.name}>'

@dataclass
class Character:
    id: int
    author_id: int
    image_path: str
    description: str
    posted_at: datetime
    likes: int
    clothes: list = field(default_factory=list)
    is_favorite: Optional[bool] = None

    def __str__(self):
        return f'<Character id={self.id} Description={self.description}>'

    def __repr__(self):
        return f'<Character id={self.id} Description={self.description}>'

class User(BaseModel):
    id: int
```

```
    username: str
    password_hash: str
    registered_at: datetime

    class Config:
        json_encoders = {
            datetime: datetime_to_timestamp
        }

class UserLogin(BaseModel):
    username: str
    password: str

class UserRegistration(BaseModel):
    username: str
    password: str

    @validator('username')
    def username_validator(cls, v):
        if not v.isalnum():
            raise ValueError('must be alphanumeric')

        if len(v) > 30 or len(v) < 3:
            raise ValueError('must be more than 3 but less than 30 characters')

        return v

    @validator('password')
    def password_validator(cls, v):
        if len(v) < 8:
            raise ValueError('must contain at least 8 characters')

        number_of_capitals = sum(character.isupper() for character in v)

        if number_of_capitals < 2:
            raise ValueError('must contain at least 2 capital letters')

        number_of_digits = sum(character.isdigit() for character in v)

        if number_of_digits < 3:
            raise ValueError('must contain at least 3 digits')

        return v
```

A.11 Код файлу looklike/redis_client.py

```
import redis
from looklike.configs import config

redis_client = redis.Redis.from_url(config.REDIS_URL)

def format_cache_key(clothes_ids: list[int]) -> str:
    cache_key = 'search_character_'
    for i, clothes_id in enumerate(clothes_ids):
        cache_key += str(clothes_id)
        if (i < len(clothes_ids) - 1):
            cache_key += ','
    return cache_key
```

A.12 Код файлу looklike/routes_decorators.py

```
from functools import wraps
from flask import request
from looklike.authorizations import JWTAuthorization
from looklike.configs import config
from looklike.exceptions import AuthorizationException
from looklike.utils import jsoning

def authorized_required(func):
    """Available to authorized users only"""
    @wraps(func)
    def decorated_function(*args, **kwargs):
        try:
            auth = JWTAuthorization(secret_key=config.SECRET_KEY)
            token = auth.get_acces_token_from_headers(request.headers)
            user_id = auth.get_user_id_from_acces_token(token)
        except AuthorizationException as e:
            return jsoning({'message': str(e)}), 401
        return func(*args, **kwargs, user_id=user_id)

    return decorated_function

def authorized_optional(func):
    """Available to all users"""
    @wraps(func)
    def decorated_function(*args, **kwargs):
        try:
            auth = JWTAuthorization(secret_key=config.SECRET_KEY)
            token = auth.get_acces_token_from_headers(request.headers)
            user_id = auth.get_user_id_from_acces_token(token)
        except AuthorizationException:
            return func(*args, **kwargs)
        return func(*args, **kwargs, user_id=user_id)

    return decorated_function
```

A.13 Код файлу looklike/serializers.py

```
from typing import Union
from looklike.configs import config
from looklike.models import Clothes, Character

class ClothesSerializer():
    @classmethod
    def serialize(cls, clothes) -> Union[dict, list]:
        if isinstance(clothes, list) or isinstance(clothes, tuple):
            return [cls.serialize_one(cl) for cl in clothes]
        else:
            return cls.serialize_one(clothes)

    @classmethod
    def serialize_one(cls, clothes: Clothes) -> dict:
        image_url = f'{config.MEDIA_URL}{clothes.image_path}'

        return {
            'id': clothes.id,
            'name': clothes.name,
            'image_path': image_url,
```

```
            'parent_id': clothes.parent_id
        }

    @classmethod
    def serialize_primary_only(
        cls,
        primary_clothes: list[Clothes]
    ) -> list[dict]:
        result = []
        for clothes in primary_clothes:
            if clothes.parent_id:
                continue

            serialized = cls.serialize_one(clothes)
            children = cls._get_children(clothes, primary_clothes)

            serialized['children'] = [
                cls.serialize_one(child) for child in children
            ]

            result.append(serialized)

        return result

    @classmethod
    def _get_children(
        cls,
        clothes: Clothes,
        clothing_list: list[Clothes]
    ) -> list[Clothes]:
        children = list(
            filter(lambda x: x.parent_id == clothes.id, clothing_list)
        )

        return children

class CharactersSerializer():
    @classmethod
    def serialize(cls, characters) -> Union[dict, list]:
        if isinstance(characters, list) or isinstance(characters, tuple):
            return [cls.serialize_one(character) for character in characters]
        else:
            return cls.serialize_one(characters)

    @classmethod
    def serialize_one(cls, character: Character) -> dict:
        image_url = f'{config.MEDIA_URL}{character.image_path}'

        output_data = {
            'id': character.id,
            'author_id': character.author_id,
            'image_path': image_url,
            'description': character.description,
            'likes': character.likes,
            'posted_at': {
                'date': character.posted_at.strftime('%d.%m.%Y'),
                'time': character.posted_at.strftime('%H:%M:%S'),
                'full': character.posted_at.strftime('%d.%m.%Y %H:%M:%S')
            }
        }

        if character.is_favorite is not None:
            output_data['is_favorite'] = character.is_favorite
```

```
        return output_data

class CharactersWithClothesSerializer():
    @classmethod
    def serialize_one(cls, character: Character) -> dict:
        serialized_character = CharactersSerializer.serialize_one(character)
        serialized_clothes = ClothesSerializer.serialize(character.clothes)
        serialized_character['clothes'] = serialized_clothes

        return serialized_character

    @classmethod
    def serialize(cls, characters) -> Union[dict, list]:
        if (isinstance(characters, list) or
                isinstance(characters, tuple)):
            return [
                cls.serialize_one(character) for character in characters
            ]
        else:
            return cls.serialize_one(characters)
```

A.14 Код файлу looklike/utils.py

```
from flask import jsonify, current_app

def get_ids_from_string(string: str) -> list[int]:
    return [abs(int(id)) for id in string.strip('[]').split(',')]

def datetime_to_timestamp(value):
    return value.strftime('%s')

def jsoning(data):
    if isinstance(data, str):
        return current_app.response_class(
            data,
            mimetype=current_app.config["JSONIFY_MIMETYPE"],
        )

    return jsonify(data)
```

A.15 Код файлу runserver.py

```
from looklike import create_app

if __name__ == "__main__":
    app = create_app()
    app.run()
```

ДОДАТОК Б

Код клієнтської частини

Б.1 Код файлу components/CharactersCard.svelte

```
<script>
  import { navigate } from "svelte-routing";
  import { isAuthorized, formatAuthorizationHeader } from '../auth.js';
  import { apiUrl } from '../settings.js';
  import { formatBigNumber } from '../utils.js';

  export let character;

  async function likeButtonClickHandler(event) {
    if (!isAuthorized()) {
      navigate('/login');
      return;
    }

    if ($character.is_favorite) {
      await removeFromFavorites($character.id);
      $character.is_favorite = false;
      $character.likes -= 1;
    } else {
      await addToFavorites($character.id);
      $character.is_favorite = true;
      $character.likes += 1;
    }
  }

  async function removeFromFavorites(character_id) {
    try {
      const response = await fetch(`${apiUrl}/characters/${character_id}/remove-from-favorites`, {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Authorization': formatAuthorizationHeader()
        }
      });

      if (response.status === 401) {
        navigate('/login');
        return;
      }

      if (response.status === 404) {
        navigate('/oops');
        return;
      }
    }
    catch (e) {
      navigate('/oops');
      return;
    }
  }

  async function addToFavorites(character_id) {
    try {
      const response = await fetch(`${apiUrl}/characters/${character_id}/add-to-favorites`, {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Authorization': formatAuthorizationHeader()
        }
```

```
            });

            if (response.status === 401) {
              navigate('/login');
              return;
            }

            if (response.status === 404 || response.status === 400) {
              navigate('/oops');
              return;
            }
          }
          catch (e) {
            navigate('/oops');
            return;
          }
        }
      </script>

      <div class="character-item">
        <img src={$character.image_path} class="character-item__image" alt="Образ">
        <div class="character-item__clothes-grid" >
          {#each $character.clothes as clothes}
            <div class="character-item__clothes-wrapper">
              <div tooltip={clothes.name}>
                <img src={clothes.image_path} class="character-item__clothes-image" alt={clothes.name}>
              </div>
            </div>
          {/each}
        </div>
        <p>{$character.description}</p>
        <div class="character-item__additional-wrapper">
          <span                                                    class="character-item__date"
tooltip={$character.posted_at.time}>{$character.posted_at.date}</span>
          <button class="like-button" on:click={likeButtonClickHandler}>
            {#if $character.is_favorite}
              <i class="material-icons like-icon active">favorite</i>
              <span                                                class="character-item__likes-count
active">{formatBigNumber($character.likes)}</span>
            {:else}
              <i class="material-icons like-icon disactive">favorite_border</i>
              <span                                                class="character-item__likes-count
disactive">{formatBigNumber($character.likes)}</span>
            {/if}
          </button>
        </div>
      </div>

      <style>
        .character-item {
          position: relative;
          transition: 0.4s all;
          margin-top: 30px;
          box-shadow: 0 0 10px rgba(0,0,0,0.3);
        }
        .character-item:hover {
          transform: translate(-3px, -3px);
          transition: 0.2s all;
          box-shadow: 0 0 15px rgba(0,0,0,0.9);
        }
        .character-item__image {
          width: 100%;
          height: 560px;
```

```
}
.character-item__image:hover {
    cursor: pointer;
}
.character-item__clothes-grid {
    display: flex;
    flex-wrap: wrap;
    padding: 0 5px;
    width: 100%;
}
.character-item__clothes-wrapper {
    width: 14%;
    position: relative;
}
.character-item__clothes-image {
    width: 100%;
    height: 60px;
    margin-top: 10px;
    cursor: pointer;
}
.character-item p {
    padding: 15px 15px 70px 15px;
    text-align: justify;
    font-size: 1.6rem;
    line-height: 1.5em;
}
.character-item__additional-wrapper {
    position: absolute;
    bottom: 0;
    width: 100%;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 0 15px 15px 15px;
}
.character-item__date {
    display: block;
    position: relative;
    font-size: 1.6rem;
    color: rgb(82, 82, 82);
}
.like-button {
    display: flex;
    align-items: center;
    background-color: transparent;
    cursor: pointer;
    transition: 0.2s all;
}
.like-button:hover {
    background-color: rgba(255, 255, 255, 0.1);
    transition: 0.2s all;
}
.character-item__likes-count {
    margin-left: 5px;
    font-size: 1.6rem;
    font-weight: 500;
}
.like-icon {
    font-size: 25px !important;
    transition: 0.2s all;
}
.disactive {
    color: rgb(128, 128, 128);
```

```
    }
    .active  {
        color: #f7c121;
    }

    @media (max-width: 768px) {
        .character-item:hover {
            /* Removes original hover effect on mobile devices */
            transform: none;
            box-shadow: 0 0 10px rgba(0,0,0,0.3);
        }
    }
</style>
```

Б.2 Код файлу components/ClothesBox.svelte

```
<script>
    import { apiUrl } from '../settings.js';

    export let clothes = [];

    function clothesItemClickHandler(event) {
        let item = event.currentTarget;
        let boxContainer = item.parentElement;
        let clothesBox = boxContainer.parentElement;
        const itemId = item.dataset.id;
        const itemName = item.querySelector('span').textContent;
        const isItemSelected = item.classList.contains('selected');

        let majorClothesBox = findMajorClothesBox(item);
        if (majorClothesBox) {
            removeAllSelectedItemFromClothesBox(majorClothesBox);
            majorClothesBox.dataset.selectedClothesId = itemId;
        }

        if (isItemSelected) {
            delete majorClothesBox.dataset.selectedClothesId;
            removeAllAdjacentClothesBox(item);
            return;
        }

        item.classList.add('selected');
        removeAllAdjacentClothesBox(item);

        if (item.dataset.isMajorItem) return;

        loadClothesChildren(itemId)
            .then(function(clothesChildren) {
                if (clothesChildren && clothesChildren.length) {
                    clothesBox.appendChild(createClothesBox(clothesChildren, itemName));
                }
            });
    };

    function createClothesBox(clothesList, boxTitle) {
        let box = document.createElement('div');
        box.className = 'clothes-box';

        let hr = document.createElement('hr');
        let h2 = document.createElement('h2');
```

```
    h2.textContent = `> ${boxTitle}`;

    let boxContainer =  document.createElement('div');
    boxContainer.className = 'clothes-box__container';

    for (const clothesObject of clothesList) {
        let newItem = document.createElement('div');
        newItem.className = 'clothes-box__container__item';
        newItem.dataset.id = clothesObject.id;
        newItem.addEventListener('click', clothesItemClickHandler);

        let imgWrapper = document.createElement('div');
        imgWrapper.className = 'img-wrapper';

        let img = document.createElement('img');
        img.src = clothesObject.image_path;
        img.alt = clothesObject.name;

        let span = document.createElement('span');
        span.textContent = clothesObject.name;

        imgWrapper.appendChild(img);
        newItem.appendChild(imgWrapper);
        newItem.appendChild(span);

        boxContainer.appendChild(newItem);
    }

    box.appendChild(h2);
    box.appendChild(hr);
    box.appendChild(boxContainer);

    return box;
}

function removeAllAdjacentClothesBox(item) {
    let boxContainer = item.parentElement;
    let box = boxContainer.parentElement;
    let subClothesBoxes = box.querySelectorAll('.clothes-box');

    if (subClothesBoxes) {
        for (let subBox of subClothesBoxes) {
            if (subBox.parentElement === box) box.removeChild(subBox);
        }
    }
}

function findMajorClothesBox(item) {
    let lastFoundClothesBox;
    let boxContainer = item.parentElement;
    if (!boxContainer) return undefined;
    let clothesBox = boxContainer.parentElement;

    if (!clothesBox) {
        return undefined;
    } else {
        lastFoundClothesBox = clothesBox;
    }

    while (lastFoundClothesBox) {
        let tmpClothesBox = lastFoundClothesBox.parentElement;

        if (tmpClothesBox && tmpClothesBox.classList.contains('clothes-box')) {
```

```
                lastFoundClothesBox = tmpClothesBox;
            } else {
                break;
            }
        }

        return lastFoundClothesBox;
    }

    function removeAllSelectedItemFromClothesBox(clothesBox) {
        let allSelectedItem = clothesBox.querySelectorAll('.selected');

        for (let item of allSelectedItem) {
            item.classList.remove('selected');
        }
    }

    async function loadClothesChildren(parent_id) {
        const url = `${apiUrl}/clothes/by-parent-id/${parent_id}`;
        const response = await fetch(url, {
            method: 'GET',
            headers: {
                'Accept': 'application/json'
            }
        });

        if (!response.ok) {
            alert(response.status + ' ' + response.statusText);
            return;
        }

        return await response.json();
    };
</script>

<div class='clothes-box'>
    <h1>{clothes.name}</h1>
    <hr>
    <div class='clothes-box__container'>
        <div on:click={clothesItemClickHandler} class='clothes-box__container__item' data-id={clothes.id} data-is-major-item="true">
            <div class='img-wrapper'>
                <img src={clothes.image_path} alt={clothes.name}>
            </div>
            <span>Будь-які</span>
        </div>

        {#each clothes.children as children}
            <div on:click={clothesItemClickHandler} class='clothes-box__container__item' data-id={children.id}>
                <div class='img-wrapper'>
                    <img src={children.image_path} alt={children.name}>
                </div>
                <span>{children.name}</span>
            </div>
        {/each}
    </div>
</div>

<style>
    :global(.clothes-box) {
        margin-top: 40px;
        padding: 10px;
```

```
    box-shadow: 0 0 10px rgba(0,0,0,0.3);
}
:global(.clothes-box .clothes-box) {
    margin-top: 10px;
}
:global(.clothes-box hr) {
    border: none;
    color: rgb(212, 212, 212);
    background-color: rgb(212, 212, 212);
    height: 1px;
    margin-top: 10px;
}
:global(.clothes-box h1) {
    font-weight: 400;
    font-size: 2.88rem;
}
:global(.clothes-box h2) {
    font-weight: 400;
    font-size: 2rem;
}
:global(.clothes-box__container) {
    display: flex;
    flex-wrap: wrap;
}
:global(.clothes-box__container__item) {
    width: 120px;
    margin-top: 25px;
    cursor: pointer;
}
:global(.clothes-box__container__item .img-wrapper img) {
    width: 100%;
    height: 135px;
    transition: 0.1s all;
}
:global(.clothes-box__container__item span) {
    display: block;
    text-align: center;
    font-size: 1.28rem;
    margin-top: 5px;
    padding: 10px 8px;
    transition: 0.3s all;
    line-height: 1.44rem;
}
:global(.clothes-box__container__item:hover .img-wrapper img) {
    transform: scale(1.07);
    transition: 0.1s all;
}
:global(.clothes-box__container__item:hover span) {
    transition: 0.3s all;
    background-color: rgb(241, 241, 241);
}
:global(.clothes-box__container__item.selected .img-wrapper) {
    position: relative;
}
:global(.clothes-box__container__item.selected .img-wrapper::after) {
    content: '';
    position: absolute;
    bottom: 0px;
    right: 0px;
    width: 40px;
    height: 40px;
    background: center no-repeat url('../images/selected.png');
    background-size: 40px 40px;
```

```
  }
</style>
```

Б.3 Код файлу components/Header.svelte

```
<script>
  import { onDestroy, onMount } from 'svelte';
  import { link } from 'svelte-routing';
  import { isAuthorized, logOut } from '../auth.js'

  let startPoint;

  onMount(() => {
    document.addEventListener('touchstart', function(event) {
      startPoint = event.changedTouches[0];
    });

    document.addEventListener('touchmove', function(event) {
      const currentPoint = event.changedTouches[0];

      const xDifference = currentPoint.clientX - startPoint.clientX;
      const yDifference = currentPoint.clientY - startPoint.clientY;

      if(Math.abs(xDifference) > 100 && Math.abs(yDifference) < 50) {
        if(xDifference < 0) {
          burgerMenuClose();
        }

        if(xDifference > 0) {
          burgerMenuOpen();
        }
      }
    });
  });

  onDestroy(() => {
    document.body.parentElement.classList.remove('scroll-lock');
  });

  function burgerClickHandler(event) {
    const burger = event.currentTarget;
    const menu = burger.nextElementSibling;

    burger.classList.toggle('active');
    menu.classList.toggle('active');
    document.body.parentElement.classList.toggle('scroll-lock')
  }

  function burgerMenuOpen() {
    const burger = document.querySelector('.header__burger');
    const menu = burger.nextElementSibling;

    burger.classList.add('active');
    menu.classList.add('active');
    document.body.parentElement.classList.add('scroll-lock')
  }

  function burgerMenuClose() {
    const burger = document.querySelector('.header__burger');
    const menu = burger.nextElementSibling;
```

```
    burger.classList.remove('active');
    menu.classList.remove('active');
    document.body.parentElement.classList.remove('scroll-lock')
  }

  function logOutLinkClickHandler() {
    logOut();
  }
</script>

<header class="header">
  <div class="container">
    <div class="header__body">
      <a href="/" class="header__logo" use:link>
        <img src="images/logo.png" alt="Логотип">
      </a>

      <div class="header__burger" on:click={burgerClickHandler}>
        <span></span>
      </div>

      <nav class="header__menu">
        <ul class="header__list">
          <li>
            <a href="/search" use:link>Пошук</a>
          </li>

          {#if isAuthorized()}
            <li>
              <a href="/favorites" use:link>Вподобані образи</a>
            </li>
            <li>
              <a href="/" class="logout-link" on:click={logOutLinkClickHandler}>Вийти</a>
            </li>
          {:else}
            <li>
              <a href="/register" use:link>Зареєструватися</a>
            </li>
            <li>
              <a href="/login" use:link>Увійти</a>
            </li>
          {/if}
        </ul>
      </nav>
    </div>
  </div>
</header>
<div id="fake-header-area"></div>

<style>
  #fake-header-area {
    width: 100%;
    height: 60px;
  }
  .header {
    width: 100%;
    position: fixed;
    top: 0;
    left: 0;
    z-index: 9999;
  }
  .header::before {
```

```
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgb(36, 36, 36);
    z-index: 2;
}
.header__body {
    position: relative;
    display: flex;
    justify-content: space-between;
    height: 60px;
    align-items: center;
}
.header__logo {
    flex: 0 0 100px;
    z-index: 3;
}
.header__logo img {
    display: block;
    max-width: 100%;
}
.header__burger {
    display: none;
}
:global(.header__burger.active::before) {
    transform: rotate(45deg);
    top: 9px !important;
}
:global(.header__burger.active::after) {
    transform: rotate(-45deg);
    bottom: 9px !important;
}
:global(.header__burger.active span) {
    transform: scale(0);
}
.header__list {
    display: flex;
    position: relative;
    z-index: 2;
}
.header__list li {
    position: relative;
    margin: 0 0 0 20px;
}
.header__list li a {
    font-size: 1.6rem;
    font-weight: 500;
    transition: 0.15s all;
    color: rgb(209, 209, 209);
}
.header__list li a:hover {
    transition: 0.15s all;
    color: white
}

@media (max-width: 768px) {
    :global(html.scroll-lock) {
        overflow: hidden;
        position: relative;
        height: 100%;
```

```
}
#fake-header-area {
  height: 50px;
}

.header__body {
  height: 50px;
}
.header__logo {
  flex: 0 0 80px;
}
.header__burger {
  display: block;
  position: relative;
  width: 30px;
  height: 20px;
  z-index: 3;
}
.header__burger span {
  background-color: white;
  border-radius: 15px;
  position: absolute;
  width: 100%;
  height: 2px;
  left: 0;
  top: 9px;
  transition: 0.3s all;
}
.header__burger::before,
.header__burger::after {
  content: '';
  background-color: white;
  border-radius: 15px;
  position: absolute;
  width: 100%;
  height: 2px;
  left: 0;
  transition: 0.3s all;
}
.header__burger::before {
  top: 0;
}
.header__burger::after {
  bottom: 0;
}
.header__menu {
  position: fixed;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background-color: rgb(94, 94, 94);
  padding: 70px 20px 30px 20px;
  overflow: auto;
  transition: 0.3s all;
}
:global(.header__menu.active) {
  left: 0 !important;
}
.header__list {
  display: block;
}
.header__list li {
```

```
            margin: 0 0 20px 0;
        }
        .header__list li a {
            font-size: 2.5rem;
        }
    }
</style>
```

Б.4 Код файлу routes/FavoriteCharacters.svelte

```
<script>
    import { onMount } from 'svelte';
    import { writable } from 'svelte/store';
    import { navigate } from 'svelte-routing';
    import Header from '../components/Header.svelte';
    import Footer from '../components/Footer.svelte';
    import CharactersCard from '../components/CharactersCard.svelte';
    import { isAuthorized, formatAuthorizationHeader } from '../auth.js'
    import { apiUrl, siteTitle } from '../settings.js';

    let favoriteCharacters = [];

    onMount(async () => {
        if (isAuthorized()) {
            const result = await loadFavoriteCharacters();

            for (let item of result) {
                favoriteCharacters.push(writable(item));
            }

            favoriteCharacters = favoriteCharacters;
        } else {
            navigate('/login');
        }
    });

    async function loadFavoriteCharacters() {
        const response = await fetch(`${apiUrl}/characters/favorites`, {
            method: 'GET',
            headers: {
                'Accept': 'application/json',
                'Authorization': formatAuthorizationHeader()
            }
        });

        let responseData = await response.json();

        if (!response.ok) {
            navigate('/oops');
        }

        return responseData;
    };
</script>

<svelte:head>
    <title>Вподобані образи | {siteTitle}</title>
</svelte:head>

<Header/>
```

```
<div class="container">
  <h1>Вподобані образи</h1>
  <div class="characters-grid">
    {#if favoriteCharacters.length}
      {#each favoriteCharacters as character}
        <CharactersCard character={character}/>
      {/each}
    {/if}
  </div>
</div>
<Footer/>

<style>
  h1 {
    text-align: center;
    margin-top: 20px;
    color: #383838;
  }
</style>
```

Б.5 Код файлу routes/Index.svelte

```
<script>
  import { onMount } from 'svelte';
  import { writable } from 'svelte/store';
  import Header from '../components/Header.svelte';
  import Footer from '../components/Footer.svelte';
  import CharactersCard from '../components/CharactersCard.svelte';
  import { isAuthorized, formatAuthorizationHeader } from '../auth.js';
  import { apiUrl, siteTitle } from '../settings.js';

  let newessCharacters = [];

  onMount(async () => {
    const result = await loadNewessCharacters();

    for (let item of result) {
      newessCharacters.push(writable(item));
    }

    newessCharacters = newessCharacters;
  });

  async function loadNewessCharacters() {
    let request_headers = {
      'Accept': 'application/json'
    };

    if (isAuthorized()) {
      request_headers['Authorization'] = formatAuthorizationHeader();
    }

    const response = await fetch(`${apiUrl}/characters/filter?type=newness&limit=3`, {
      method: 'GET',
      headers: request_headers
    });

    let responseData = await response.json();

    if (!response.ok) {
```

```
            navigate('/oops');
        }

        return responseData;
    };
</script>

<svelte:head>
    <title>{siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
    <h1>Найновіші образи</h1>
    <div class="characters-grid">
        {#each newessCharacters as character}
            <CharactersCard character={character}/>
        {/each}
    </div>

    <h1>Найпопулярніші образи</h1>
    <div class="characters-grid">
        {#each newessCharacters as character}
            <CharactersCard character={character}/>
        {/each}
    </div>
</div>

<style>
    h1 {
        text-align: center;
        margin-top: 20px;
        color: #383838;
    }
</style>
<Footer/>
```

Б.6 Код файлу routes/Login.svelte

```
<script>
    import { link, navigate } from 'svelte-routing';
    import Header from '../components/Header.svelte';
    import Footer from '../components/Footer.svelte';
    import { setToken } from '../auth.js'
    import { apiUrl, siteTitle } from '../settings.js';
    import { LoginFormValidator } from '../form-validators.js';
    import { whooshAnimation } from '../utils.js';

    let errors = [];

    async function submitLoginFormHandler(event) {
        event.preventDefault();

        const validator = new LoginFormValidator(event.target);
        const validationErrors = validator.validate();

        if (validationErrors.length) {
            errors = validationErrors;
            return;
        }
```

```
    const tokenObject = await tryLogin(validator.username, validator.password);

    if (tokenObject) {
      setToken(tokenObject.access_token);
      navigate('/');
    }
  }

  async function tryLogin(username, password) {
    try {
      const response = await fetch(`${apiUrl}/users/login`, {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({username, password})
      });

      if (response.status === 404) {
        errors = ['Не знайдено користувача з таким ім\'ям!']
        return null;
      }

      if (response.status === 401) {
        errors = ['Надано невірний пароль!']
        return null;
      }

      return await response.json();
    }
    catch (e) {
      navigate('/oops');
    }
  }
</script>

<svelte:head>
  <title>Вхід | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
  {#if errors.length }
    <div in:whooshAnimation class="block-status b-errors">
      <span>Помилки:</span>
      <ul>
        {#each errors as error}
          <li>{error}</li>
        {/each}
      </ul>
    </div>
  {/if}

  <div class="lr-card" class:min-margin-if-errors = "{errors.length}">
    <div class="lr-card-title">Вхід</div>
    <div class="lr-card-main">
      <form on:submit={submitLoginFormHandler} method="POST">
        <div class="lr-card-form-container">
          <div class="lr-card-fields-wrapper">
            <label>Логін: <input type="text" name="username" autocomplete="username"></label>
```

```
                        <label>Пароль: <input type="password" name="password" autocomplete="current-password"></label>
                    </div>
                    <button type="submit" class="black-button lr-card-button">Увійти</button>
                </div>
            </form>
        </div>
        <div class="lr-card-additionally">У вас ще немає акаунту? <a href="/register" class="lr-card-link" use:link>Зареєструватися</a></div>
    </div>
</div>
<Footer/>
```

Б.7 Код файлу routes/NotFound404.svelte

```
<script>
    import Header from '../components/Header.svelte';
    import Footer from '../components/Footer.svelte';
    import { siteTitle } from '../settings.js';
</script>

<svelte:head>
    <title>Сторінка не знайдена | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
    <div id="notfound">
        <div class="notfound">
            <div class="notfound-404">
                <h1>4<span></span>4</h1>
            </div>
            <h2>Упс! Сторінку не знайдено</h2>
            <p>Невірно набрана адреса, або такої сторінки на сайті більше не існує</p>
            <a class="black-button" href="/">Повернутися на головну</a>
        </div>
    </div>
</div>
<Footer/>

<style>
    #notfound {
        position: relative;
        height: 80vh;
    }
    #notfound .notfound {
        position: absolute;
        left: 50%;
        top: 50%;
        -webkit-transform: translate(-50%, -50%);
        -ms-transform: translate(-50%, -50%);
        transform: translate(-50%, -50%);
    }
    .notfound {
        max-width: 520px;
        width: 100%;
        text-align: center;
        line-height: 1.4;
    }
    .notfound .notfound-404 {
```

```
    height: 190px;
  }
  .notfound .notfound-404 h1 {
    font-size: 146px;
    font-weight: 700;
    margin: 0px;
    color: #383838;
  }
  .notfound .notfound-404 h1>span {
    display: inline-block;
    width: 120px;
    height: 120px;
    background-image: url('../images/emoji.png');
    background-size: cover;
    z-index: -1;
  }
  .notfound h2 {
    font-size: 22px;
    font-weight: 700;
    margin: 0;
    color: #383838;
  }
  .notfound p {
    color: #787878;
    font-size: 1.6rem;
    font-weight: 300;
    margin: 1.6rem 0 3.2rem 0;
  }
</style>
```

Б.8 Код файлу routes/Oops.svelte

```
<script>
  import Header from '../components/Header.svelte';
  import Footer from '../components/Footer.svelte';
  import { siteTitle } from '../settings.js';
</script>

<svelte:head>
  <title>Помилка | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
  <div id="some-error">
    <div class="some-error">
      <div>
        <span></span>
      </div>
      <h2>Упс! Щось пішло не так</h2>
      <p>Невірно оброблений запит, або сервер наразі недоступний</p>
      <a class="black-button" href="/">Повернутися на головну</a>
    </div>
  </div>
</div>
<Footer/>

<style>
  #some-error {
    position: relative;
```

```
    height: 80vh;
  }
  #some-error .some-error {
    position: absolute;
    left: 50%;
    top: 50%;
    -webkit-transform: translate(-50%, -50%);
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
  }
  .some-error {
    max-width: 520px;
    width: 100%;
    text-align: center;
    line-height: 1.4;
  }
  .some-error div {
    height: 190px;
  }
  .some-error span {
    display: inline-block;
    width: 120px;
    height: 120px;
    background-image: url('../images/emoji.png');
    background-size: cover;
    z-index: -1;
  }
  .some-error h2 {
    font-size: 22px;
    font-weight: 700;
    margin: 0;
    color: #383838;
  }
  .some-error p {
    color: #787878;
    font-size: 1.6rem;;
    font-weight: 300;
    margin: 1.6rem 0 3.2rem 0;
  }
</style>
```

Б.9 Код файлу routes/Register.svelte

```
<script>
  import { link, navigate } from 'svelte-routing';
  import Header from '../components/Header.svelte';
  import Footer from '../components/Footer.svelte';
  import { setToken } from '../auth.js'
  import { apiUrl, siteTitle } from '../settings.js';
  import { RegisterFormValidator } from '../form-validators.js';
  import { whooshAnimation } from '../utils.js';

  let errors = [];

  async function submitRegisterFormHandler(event) {
    event.preventDefault();

    const validator = new RegisterFormValidator(event.target);
    const validationErrors = validator.validate();
```

```
        if (validationErrors.length) {
            errors = validationErrors;
            return;
        }

        const tokenObject = await tryRegister(validator.username, validator.password);

        if (tokenObject) {
            setToken(tokenObject.access_token);
            navigate('/');
        }
    }

    async function tryRegister(username, password) {
        try {
            const RegisterResponse = await fetch(`${apiUrl}/users/register`, {
                method: 'POST',
                headers: {
                    'Accept': 'application/json',
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({username, password})
            });

            if (RegisterResponse.status === 409) {
                errors = ['Користувач з таким ім\'ям вже існує!']
                return null;
            }

            const LoginResponse = await fetch(`${apiUrl}/users/login`, {
                method: 'POST',
                headers: {
                    'Accept': 'application/json',
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({username, password})
            });

            return await LoginResponse.json();
        }
        catch (e) {
            console.log(e)
            navigate('/oops');
        }
    }
</script>

<svelte:head>
    <title>Реєстрація | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
    {#if errors.length }
        <div in:whooshAnimation class="block-status b-errors">
            <span>Помилки:</span>
            <ul>
                {#each errors as error}
                    <li>{error}</li>
                {/each}
            </ul>
        </div>
    {/if}
```

```
        <div class="lr-card" class:min-margin-if-errors = "{errors.length}">
          <div class="lr-card-title">Реєстрація</div>
          <div class="lr-card-main">
            <form on:submit={submitRegisterFormHandler} method="POST">
              <div class="lr-card-form-container">
                <div class="lr-card-fields-wrapper">
                  <label>Логін: <input type="text" name="username" autocomplete="username"></label>
                  <label>Пароль: <input type="password" name="password" autocomplete="new-
password"></label>
                  <label>Підтвердження паролю: <input type="password" name="passwordConfirmation"
autocomplete="new-password"></label>
                </div>
                <button type="submit" class="black-button lr-card-button">Зареєструватися</button>
              </div>
            </form>
          </div>
          <div class="lr-card-additionally">У вас вже є акаунту? <a href="/login" class="lr-card-link"
use:link>Увійти</a></div>
        </div>
      </div>
      <Footer/>
```

Б.10 Код файлу routes/Search.svelte

```
<script>
  import { onMount } from 'svelte';
  import { navigate } from 'svelte-routing';
  import Header from '../components/Header.svelte';
  import Footer from '../components/Footer.svelte';
  import ClothesBox from './../components/ClothesBox.svelte';
  import { apiUrl, siteTitle } from '../settings.js';
  import { whooshAnimation } from '../utils.js';

  let clothesList = [];
  let error;

  onMount(async () => {
    const result = await loadPrimaryClothes();
    clothesList = result;
  });

  function searchButtonClickHandler(event) {
    const allSelectedItemIds = collectAllSelectedItemIds();

    if (!allSelectedItemIds.length) {
      error = 'Небхідно обрати хоча б один елемент одягу!';
      return;
    }

    const formatedSelectedItemIdsString = formatSelectedItemIds(allSelectedItemIds);
    const url = `/search-result?${formatedSelectedItemIdsString}`

    navigate(url);
  }

  function collectAllSelectedItemIds() {
    let selectedItemIds = [];
    let allMajorClothesBoxes = document.querySelectorAll('.clothes-box');
```

```
        for (let majorClothesBox of allMajorClothesBoxes) {
            const selectedItemId = majorClothesBox.dataset.selectedClothesId;
            if (selectedItemId) selectedItemIds.push(selectedItemId);
        }

        return selectedItemIds;
    }

    function formatSelectedItemIds(allSelectedItemIds) {
        let str = 'clothes[]=';

        for (let i = 0; i < allSelectedItemIds.length; i++) {
            str += allSelectedItemIds[i];
            if (i !== allSelectedItemIds.length - 1) str += ',';
        }

        return str;
    }

    async function loadPrimaryClothes() {
        const response = await fetch(`${apiUrl}/clothes/primary`, {
            method: 'GET',
            headers: {
                'Accept': 'application/json'
            }
        });
        return await response.json();
    }
</script>

<svelte:head>
    <title>Пошук | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
    <h1>Оберіть вподобаний одяг</h1>
    {#each clothesList as clothes}
        <ClothesBox clothes={clothes}/>
    {/each}

    {#if error }
        <div in:whooshAnimation class="block-status b-errors">
            <span>Помилки:</span>
            <ul>
                <li>{error}</li>
            </ul>
        </div>
    {/if}

    <div class="black-button-wrapper">
        <button class="black-button" on:click={searchButtonClickHandler}>Підібрати образи</button>
    </div>
</div>
<Footer/>

<style>
    h1 {
        text-align: center;
        margin-top: 20px;
        color: #383838;
    }
    .black-button-wrapper {
```

```
        display: flex;
        justify-content: center;
        margin-top: 30px;
    }
</style>
```

Б.11 Код файлу routes/SearchResult.svelte

```
<script>
    import { onMount } from 'svelte';
    import { writable } from 'svelte/store';
    import { navigate } from 'svelte-routing';
    import CharactersCard from '../components/CharactersCard.svelte';
    import Header from '../components/Header.svelte';
    import Footer from '../components/Footer.svelte';
    import { isAuthorized, formatAuthorizationHeader } from '../auth.js'
    import { apiUrl, siteTitle } from '../settings.js';
    import { getAllUrlGetParams, whooshAnimation } from '../utils.js';

    let error;
    let searchResult = [];

    onMount(async () => {
        const urlPart = formatUrlPart(window.location.href);
        const result = await loadSearchResult(urlPart);

        if (result && Array.isArray(result)){
            if (!result.length) {
                error = 'На жаль, образів з такими речами не знайдено';
            } else {
                for (let item of result) {
                    searchResult.push(writable(item));
                }

                searchResult = searchResult;
            }
        }
    });

    function formatUrlPart(url) {
        const allUrlGetParams = getAllUrlGetParams(url);
        const clothesIds = allUrlGetParams.clothes[0];
        let ouputString = '?clothes=';

        if (!clothesIds) return ouputString;

        return ouputString + clothesIds;
    }

    async function loadSearchResult(urlPart) {
        let request_headers = {
            'Accept': 'application/json'
        };

        if (isAuthorized()) {
            request_headers['Authorization'] = formatAuthorizationHeader();
        }

        const response = await fetch(`${apiUrl}/characters/find${urlPart}`, {
            method: 'GET',
```

```
            headers: request_headers
        });

        let responseData = await response.json();

        if (!response.ok) {
            if (responseData.message === 'One or more Clothes from the list were not found!') {
                error = 'Не знайдено один або декілька елементів одягу зі списку';
            } else {
                navigate('/oops');
            }
        }

        return responseData;
    }
</script>

<svelte:head>
    <title>Результати пошуку | {siteTitle}</title>
</svelte:head>

<Header/>
<div class="container">
    {#if searchResult.length}
        <h1>Знайдені образи</h1>
        <div class="characters-grid">
            {#each searchResult as character}
                <CharactersCard character={character}/>
            {/each}
        </div>
    {/if}

    {#if error}
        <div in:whooshAnimation class="block-status b-errors">
            <span>Помилки:</span>
            <ul>
                <li>{error}</li>
            </ul>
        </div>
    {/if}

    <div class="black-button-wrapper">
        <a class="black-button" href="/search">Обрати інший одяг</a>
    </div>
</div>
<Footer/>

<style>
    h1 {
        text-align: center;
        margin-top: 20px;
        color: #383838;
    }
    .black-button-wrapper {
        margin-top: 30px;
        display: flex;
        justify-content: center;
    }
</style>
```

Б.12 Код файлу App.svelte

```
<script>
   import { Router, Route } from 'svelte-routing';

   import Index from './routes/Index.svelte';
   import Register from './routes/Register.svelte';
   import Login from './routes/Login.svelte';
   import Search from './routes/Search.svelte';
   import SearchResult from './routes/SearchResult.svelte';
   import Oops from './routes/Oops.svelte';
   import FavoriteCharacters from './routes/FavoriteCharacters.svelte';
   import NotFound404 from './routes/NotFound404.svelte';
</script>

<Router>
   <Route path='/' component={Index}/>
   <Route path='/register/' component={Register}/>
   <Route path='/login/' component={Login}/>
   <Route path='/search/' component={Search}/>
   <Route path='/search-result/' component={SearchResult}/>
   <Route path='/favorites/' component={FavoriteCharacters}/>

   <!-- Register Oops page -->
   <Route path='/oops/' component={Oops}/>

   <!-- Register 404 page route -->
   <Route component={NotFound404}/>
</Router>
```

Б.13 Код файлу auth.js

```
export function getToken() {
   return localStorage.getItem('LOOKLIKE_JWT_TOKEN');
}

export function setToken(token) {
   localStorage.setItem('LOOKLIKE_JWT_TOKEN', token);
}

export function isAuthorized() {
   const token = getToken();
   return token && token !== 'undefined';
}

export function formatAuthorizationHeader() {
   return `Bearer ${getToken()}`
}

export function logOut() {
   setToken('');
}
```

Б.14 Код файлу form-validators.js

```
export class RegisterFormValidator {
  constructor(form) {
    this.form = form;
    this.validationErrors = [];
  }

  validate() {
    const formData = new FormData(this.form);

    this.username = formData.get('username');
    this.password = formData.get('password');
    this.passwordConfirmation = formData.get('passwordConfirmation');

    this.validateUsername();
    this.validatePassword();
    this.validatePasswordConfirmation();
    this.comparePasswords();

    return this.validationErrors;
  }

  validateUsername() {
    if (!this.username) {
      this.validationErrors.push('Поле логіну є обов\'язковим для заповнення');
      return;
    }

    const isOnlyHasAllowedCharacters = /^[a-zA-Z1-9]+$/.test(this.username);

    if (!isOnlyHasAllowedCharacters) {
      this.validationErrors.push('Логін повинен містити лише латинські літери та цифри!');
    }

    if (this.username.length < 3 || this.username.length > 30) {
      this.validationErrors.push('Логін повинен містити від трьох до тридцяти символів!');
    }
  }

  validatePassword() {
    if (!this.password) {
      this.validationErrors.push('Поле паролю є обов\'язковим для заповнення');
      return;
    }

    const reCapital = this.password.match(/[A-Z]/g);
    const reDigits = this.password.match(/[0-9]/g);

    const numberOfCapitalLetters = reCapital ? reCapital.length : 0;
    const numberOfDigits = reDigits ? reDigits.length: 0;

    if (this.password.length < 8) {
      this.validationErrors.push('Пароль повинен містити мінімум шість символів!');
    }

    if (numberOfCapitalLetters < 2) {
      this.validationErrors.push('Пароль повинен містити мінімум дві великі літери!');
    }

    if (numberOfDigits < 3) {
```

```
            this.validationErrors.push('Пароль повинен містити мінімум три цифри!');
        }
    }

    validatePasswordConfirmation() {
        if (!this.passwordConfirmation) {
            this.validationErrors.push('Поле підтвердження паролю є обов\'язковим для заповнення');
        }
    }

    comparePasswords() {
        if (this.password !== this.passwordConfirmation) {
            this.validationErrors.push('Паролі не співпадають!');
        }
    }
}


export class LoginFormValidator {
    constructor(form) {
        this.form = form;
        this.validationErrors = [];
    }

    validate() {
        const formData = new FormData(this.form);

        this.username = formData.get('username');
        this.password = formData.get('password');

        this.validateUsername();
        this.validatePassword();

        return this.validationErrors;
    }

    validateUsername() {
        if (!this.username) {
            this.validationErrors.push('Поле логіну є обов\'язковим для заповнення');
        }
    }

    validatePassword() {
        if (!this.password) {
            this.validationErrors.push('Поле паролю є обов\'язковим для заповнення');
        }
    }
}
```

Б.15 Код файлу main.js

```
import App from './App.svelte';

const app = new App({
    target: document.body
});

export default app;
```

Б.16 Код файлу settings.js

```
export const siteTitle = 'Look Like'
export const apiUrl = '/api/v1';
```

Б.17 Код файлу utils.js

```
import { elasticOut } from 'svelte/easing';

export function getAllUrlGetParams(url) {
  // get query string from url (optional) or window
  var queryString = url ? url.split('?')[1] : window.location.search.slice(1);
  // we'll store the parameters here
  var obj = {};
  // if query string exists
  if (queryString) {
    // stuff after # is not part of query string, so get rid of it
    queryString = queryString.split('#')[0];

    // split our query string into its component parts
    var arr = queryString.split('&');

    for (var i = 0; i < arr.length; i++) {
      // separate the keys and the values
      var a = arr[i].split('=');

      // set parameter name and value (use 'true' if empty)
      var paramName = a[0];
      var paramValue = typeof (a[1]) === 'undefined' ? true : a[1];

      // (optional) keep case consistent
      paramName = paramName.toLowerCase();
      if (typeof paramValue === 'string') paramValue = paramValue.toLowerCase();

      // if the paramName ends with square brackets, e.g. colors[] or colors[2]
      if (paramName.match(/\[(\d+)?\]$/)) {
        // create key if it doesn't exist
        var key = paramName.replace(/\[(\d+)?\]/, '');
        if (!obj[key]) obj[key] = [];

        // if it's an indexed array e.g. colors[2]
        if (paramName.match(/\[\d+\]$/)) {
          // get the index value and add the entry at the appropriate position
          var index = /\[(\d+)\]/.exec(paramName)[1];
          obj[key][index] = paramValue;
        } else {
          // otherwise add the value to the end of the array
          obj[key].push(paramValue);
        }
      } else {
        // we're dealing with a string
        if (!obj[paramName]) {
          // if it doesn't exist, create property
          obj[paramName] = paramValue;
        } else if (obj[paramName] && typeof obj[paramName] === 'string'){
          // if property does exist and it's a string, convert it to an array
          obj[paramName] = [obj[paramName]];
          obj[paramName].push(paramValue);
        } else {
```

```
                // otherwise add the property
                obj[paramName].push(paramValue);
            }
        }
    }
}

return obj;
}

export function formatBigNumber(number) {
    if (number > 1_000_000) {
        return `${(number / 1_000_000).toFixed(1)}M`
    } else if (number > 1_000) {
        return `${(number / 1_000).toFixed(1)}K`
    } else {
        return `${number}`
    }
}

export function whooshAnimation(node, params) {
    const existingTransform = getComputedStyle(node).transform.replace('none', '');

    return {
        delay: params.delay || 0,
        duration: params.duration || 500,
        easing: params.easing || elasticOut,
        css: (t, u) => `transform: ${existingTransform} scale(${t})`
    };
}
```

ДОДАТОК В

Слайди презентації

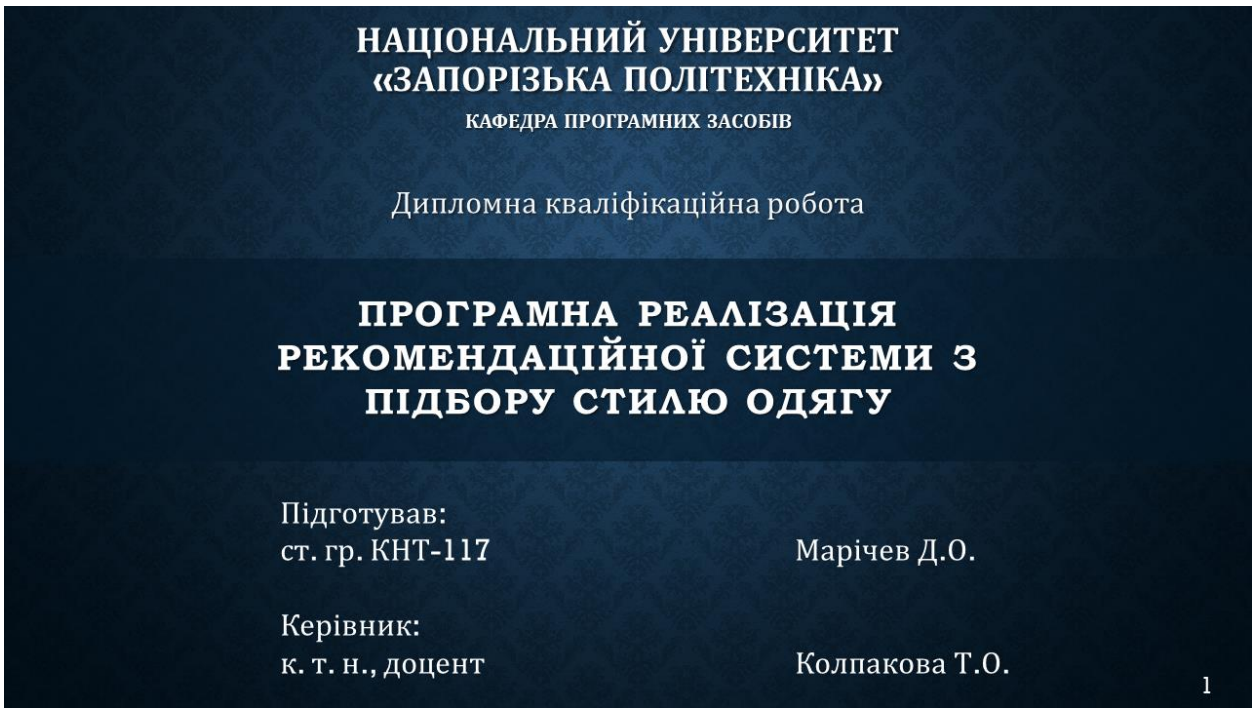


Рисунок В.1 – Слайд №1

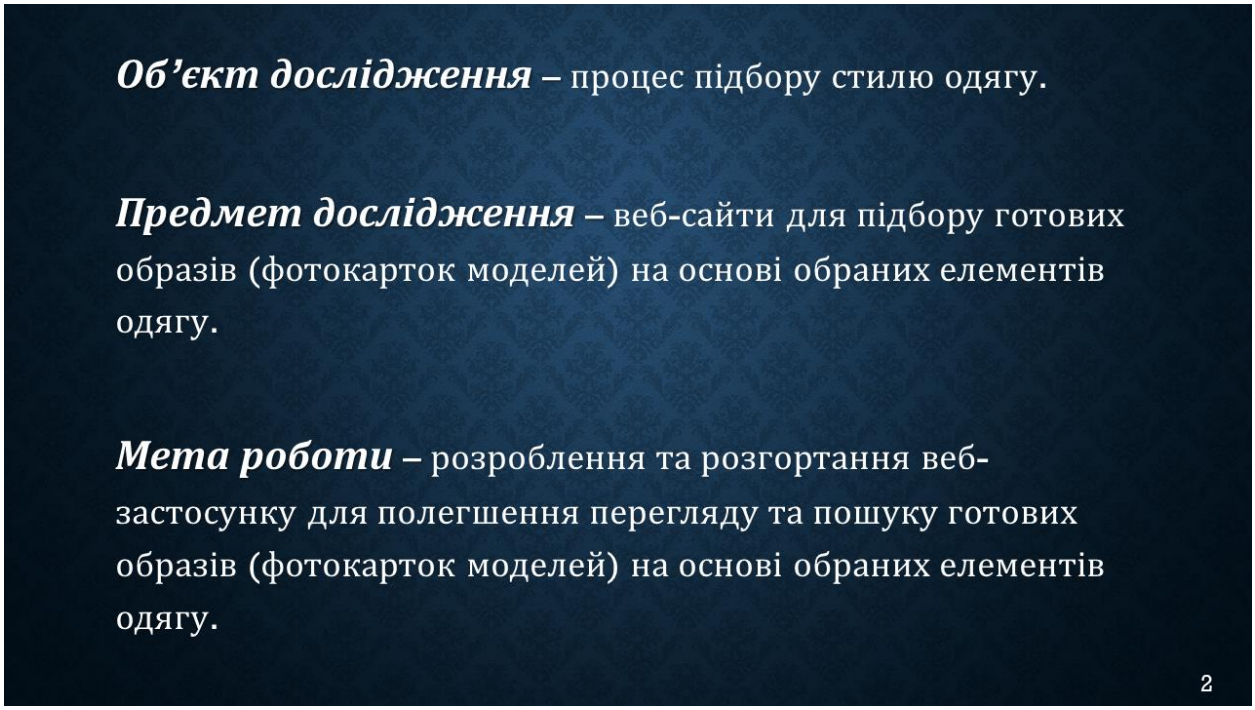


Рисунок В.2 – Слайд №2

Рисунок В.3 – Слайд №3

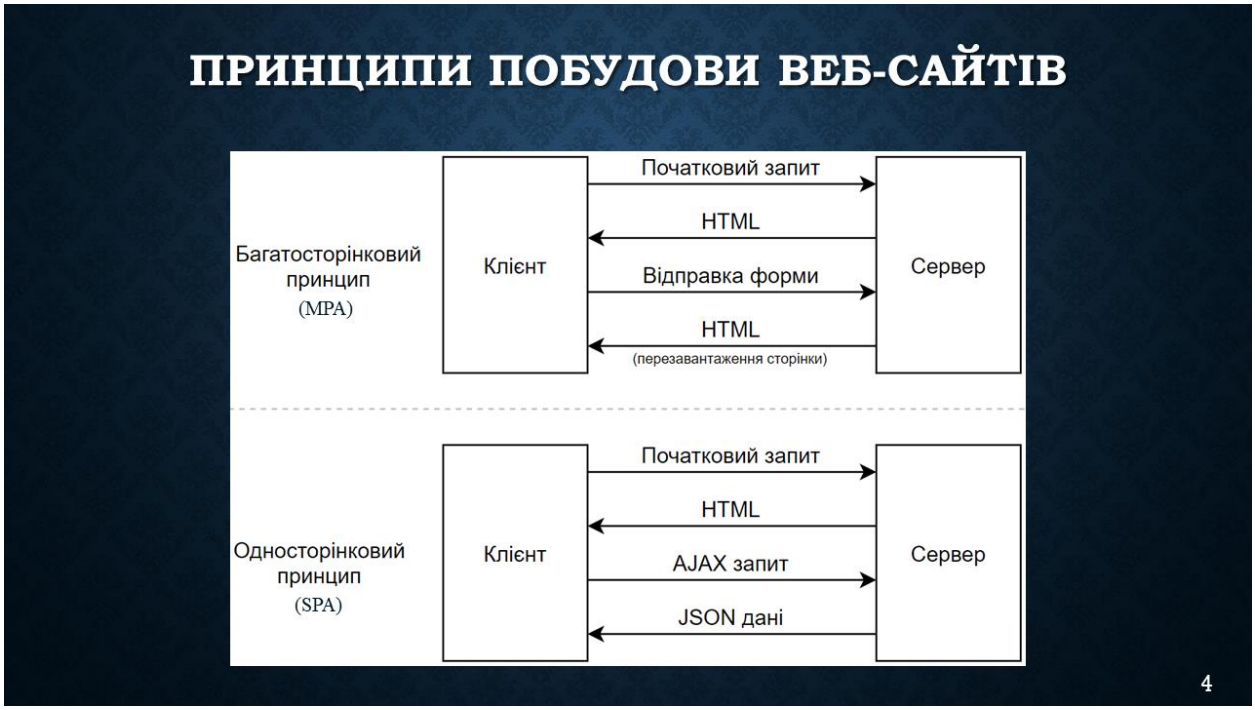


Рисунок В.4 – Слайд №4

ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ФРЕЙМВОРКІВ

Характеристика	Мова програмування						
	Python	PHP	Java	C#	JS	Ruby	Go
Простота	10	8	5	5	8	2	6
Читабельність	10	7	6	6	7	3	8
Швидкість розробки	9	8	7	6	9	5	7
Популярність	9	7	6	6	8	4	5
Кількість готових бібліотек	9	3	6	6	6	7	4
Швидкість виконання коду	5	4	8	7	7	6	9
Всього балів	52	37	38	36	45	27	39

Обрано:
Python для реалізації серверної частини
JavaScript (JS) для реалізації клієнтської частини

Обрано:
Flask для серверної частини
Svelte для клієнтської частини

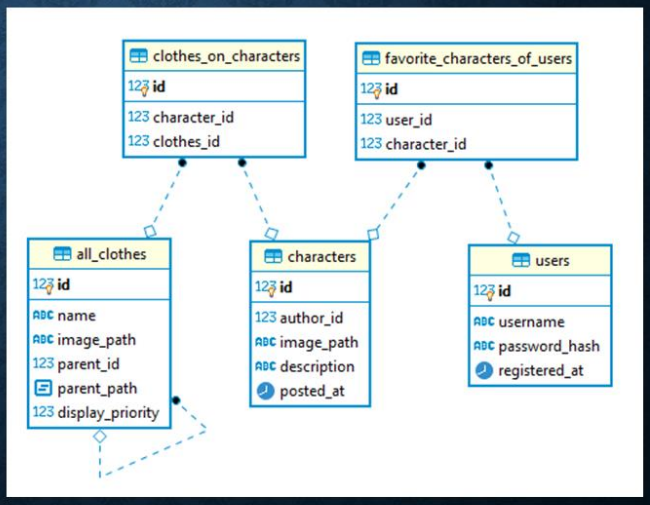
Характеристика	Назва фреймворку		
	React	Vue	Svelte
Простота	+	+	+
Документація	-	+	+
Наявність віртуального DOM	+	+	-
Швидкість роботи	+-	+-	+
Малий розмір готового js файлу	-	-	+
Справжня реактивність	-	-	+

5

Рисунок В.5 – Слайд №5

БАЗИ ДАНИХ

Обрано:
PostgreSQL як основну базу даних
Redis як допоміжну базу даних



6

Рисунок В.6 – Слайд №6

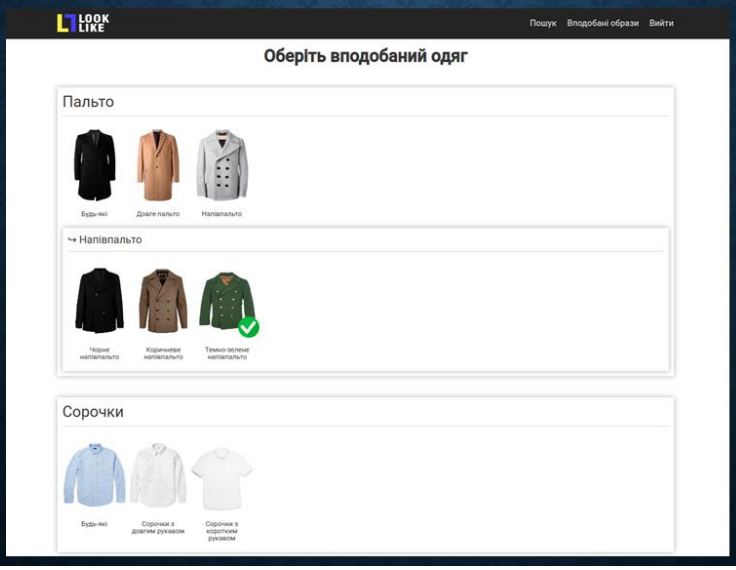
РОЗРОБКА API

URL	Метод	Опис
/api/v1/users/register	POST	Реєструє користувача у системі
/api/v1/users/login	POST	Авторизує користувача та повертає йому JWT
/api/v1/clothes	GET	Повертає перелік усіх елементів одягу
/api/v1/characters/find	GET	Знаходить та повертає перелік образів, в яких присутні передані елементи одягу
/api/v1/characters/<id>/add-to-favorites	POST	Додає образ до вподобаних
/api/v1/characters/<id>/remove-from-favorites	POST	Видаляє образ із вподобаних
/api/v1/clothes/primary	GET	Повертає перелік тільки основних елементів одягу
/api/v1/clothes/by-parent-id/<id>	GET	Повертає елементи одягу, які мають конкретного предка
/api/v1/characters	GET	Повертає перелік усіх образів
/api/v1/characters/<id>	GET	Повертає конкретний образ
/api/v1/characters/favorites	GET	Повертає вподобані образи користувача
/api/v1/characters/filter	GET	Повертає відфільтрований перелік образів

7

Рисунок В.7 – Слайд №7

ІНТЕРФЕЙС СТОРІНКИ ПОШУКУ ОБРАЗІВ



8

Рисунок В.8 – Слайд №8

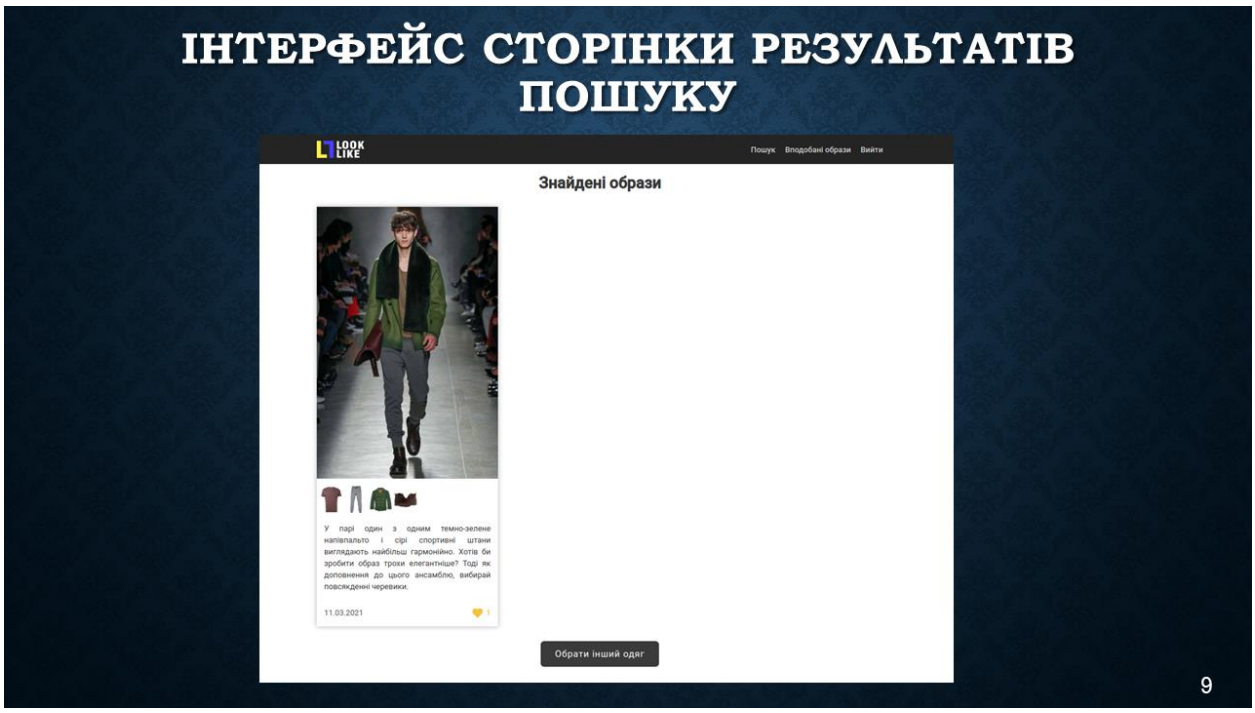


Рисунок В.9 – Слайд №9

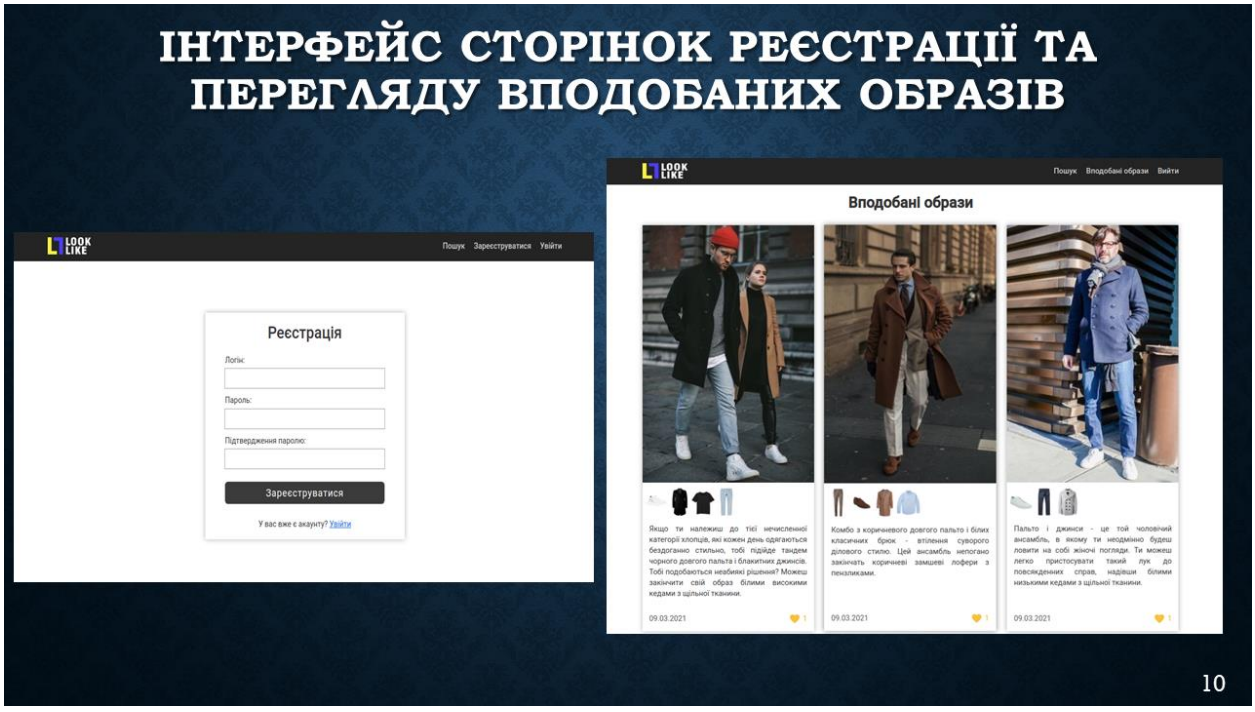


Рисунок В.10 – Слайд №10

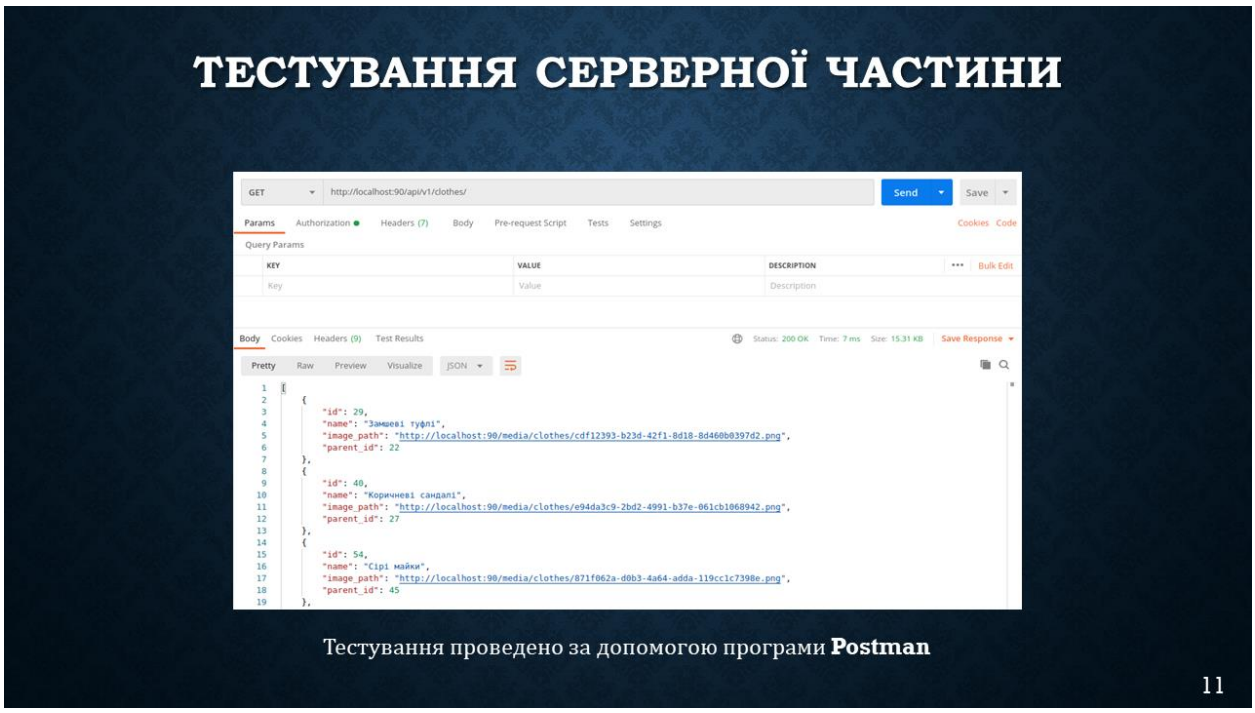


Рисунок В.11 – Слайд №11

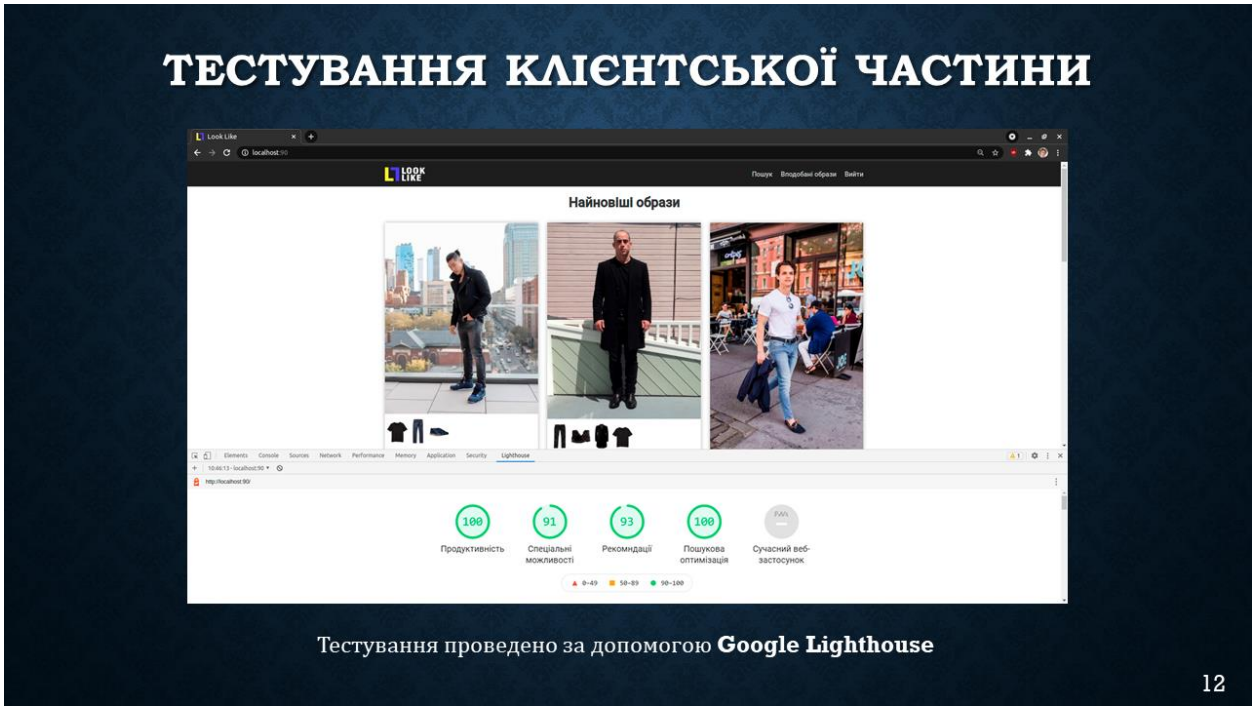


Рисунок В.12 – Слайд №12

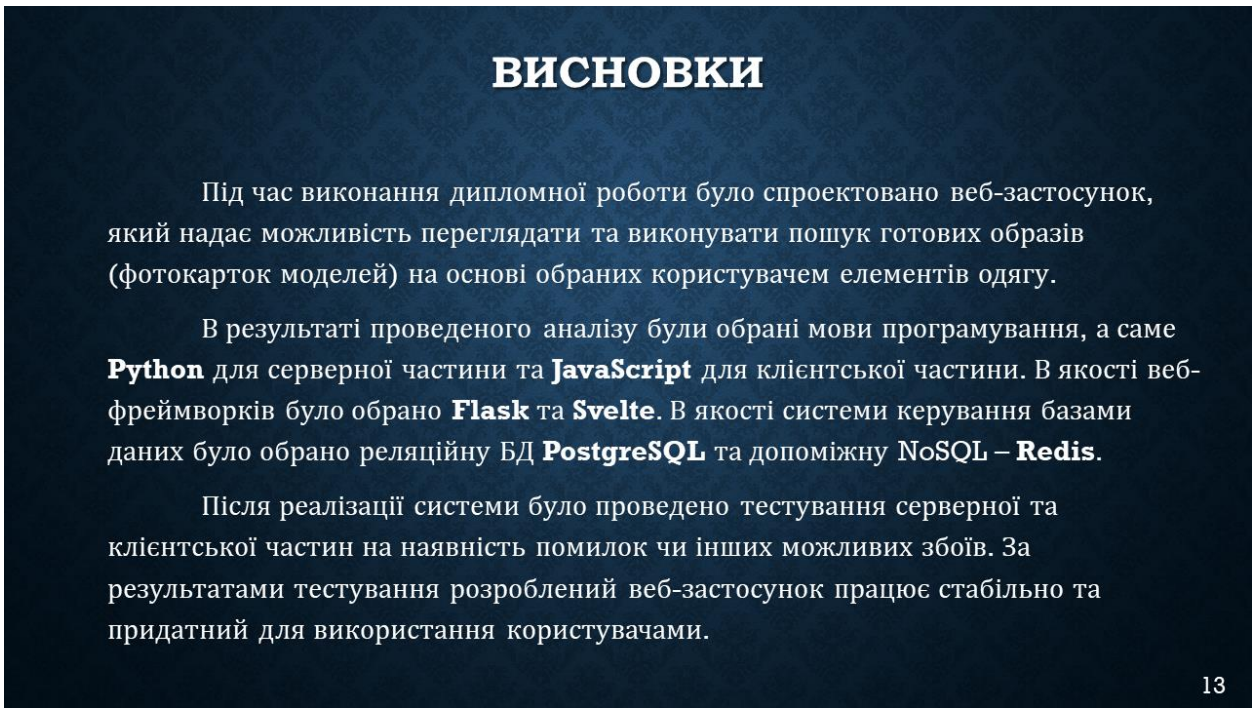


Рисунок В.13 – Слайд №13