Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Мобільний додаток для управління гардеробом.Back-end
(тема)

Виконав:
здобувач ___4___ року навчання
групи ПЗПІ-21-6

Анатолій ЗАМКОВИЙ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Наталя КРАВЕЦЬ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Замковому Анатолію Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Мобільний додаток для управління гардеробом. Back-end

Затвердже нанаказом по університету від 19.05.2025р. № 397 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 06.06.2025

3. Вихідні дані до роботи Розробити Back-end мобільного застосунку для управління гардеробом, що дозволяє користувачам зберігати, редагувати та переглядати свій одяг, отримувати рекомендації щодо вбрання на основі погоди, кольорової гами та подій. Застосунок реалізується мовою програмування Python з використанням FastAPI як веб-фреймворку.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	27.05.2025	*виконано*
3	Проектування ПЗ	30.05.2025	*виконано*
4	Розробка ПЗ	02.06.2025	*виконано*
5	Тестування ПЗ	04.06.2025	*виконано*
6	Оформлення пояснювальної записки	06.06.2025	*виконано*
7	Підготовка презентації та доповіді	08.06.2025	*виконано*
8	Попередній захист	11.06.2025	*виконано*
9	Нормоконтроль, рецензування	11.06.2025	*виконано*
10	Здача роботи у електронний архів	11.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри		*виконано*

Дата видачі завдання «17 » « квітня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Наталя КРАВЕЦЬ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 122 стор., 24 рис., 16 табл., 13 джерел.

СЕРВЕРНА ЧАСТИНА, МОБІЛЬНИЙ ДОДАТОК, РЕКОМЕНДАЦІЯ ОДЯГУ, ЗОБРАЖЕННЯ, PYTHON, FASTAPI, REST API, MYSQL

Об’єкт розробки – Back-End мобільного додатка для системи керування гардеробом.

Мета розробки – створення програмного застосунку, який матиме в собі функціональність обробки користувацьких запитів для системи керування гардербом.

Метод рішення – середовище розробки Vs Code, мови програмування Python та SQL, фреймворк FASTApi, база даних MySQL.

У результаті розробки створенопрограмний застосунок, котрий має в собі функціанальність керування власним гардеробом.

ABSTRACT

SERVER SOFTWARE, MOBILE APPLICATION, CLOTHING RECOMMENDATION, IMAGES, PYTHON, FASTAPI, REST API, MYSQL

Object of development – a server-side application for a wardrobe management system.

Purpose of development – to create a software application that handles user requests for the wardrobe management system.

Method of implementation – the development environment used was VS Code, with Python and SQL as programming languages, the FastAPI framework, and the MySQL database system.

As a result of development, a software application was created that provides functionality for managing a personal wardrobe.

ЗМІСТ

ВСТУП

Темою кваліфікаційної роботи є серверна частина мобільного додатку для управління гардеробом. Основне завдання додатку – надання користувачам можливості ефективно організовувати свій одяг, створювати образи та отримувати рекомендації щодо поєднання елементів гардеробу.

На сьогоднішній день системи керування гардеробом здобувають все більшої популярності, особливо у контексті цифровізації повсякденного життя. Багато стартапів та компаній, такі як Stylebook, Cladwell та Smart Closet, пропонують мобільні додатки, які дозволяють користувачам організовувати одяг, створювати образи, аналізувати частоту використання речей та отримувати рекомендації. Проте більшість з них мають обмежену функціональність, не враховують особистих уподобань користувача в повному обсязі або не забезпечують гнучкості інтеграції з іншими системами.

З огляду на зростання обсягів особистих речей, збереження порядку та ефективне використання наявного гардеробу стають нагальними завданнями для багатьох користувачів. Існуючі рішення не завжди враховують індивідуальні потреби користувача. Тому створення адаптивної, зручної та розширюваної системи керування гардеробом є актуальним завданням, яке сприятиме підвищенню комфорту користувачів та раціоналізації їх повсякденного життя.

Метою роботи є розробка серверної частини програмної системи, яка забезпечує роботу мобільного додатку.

Завданням роботи є створити серверну частину програмної системи для мобільного додатку. Необхідно реалізувати серверІ, спроєктувати базу даних, розробити механізми генерації рекомендацій одягу, забезпечити документування API та провести тестування основних функцій.

Засоби реалізації включають сучасні інструменти та технології: мову програмування Python, фреймворк FastAPI для створення REST API, систему керування базами даних MySQL, а також засоби для тестування та

документування API. Відповідно до технологій використовується середовище розробки VS Code.

Галузь використання результатів роботи охоплює мобільні програмні рішення для персоналізованого керування гардеробом, що можуть бути інтегровані в модні застосунки, сервіси рекомендацій стилю або персональні цифрові помічники.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

З моменту появи цифрових систем ми спостерігали, як класичні методи вирішення побутових проблем отримували програмні аналоги. Спочатку це були прості додатки для розрахунків, зберігання даних, їх найпростішої обробки, а зараз у центрі уваги – великі розумні системи.

За останні роки так з’явилося багато систем керування гардеробом, оскільки все більше користувачів прагнуть впорядкувати свій одяг та спростити вибір щоденних образів. Завдяки інтеграції з прогнозами погоди, сучасні гардеробні асистенти не лише зберігають інформацію про речі, а й рекомендують ідеальні поєднання для різних подій.

Як випливає з концепції, система керування гардеробом – це зручний цифровий інструмент, що дозволяє користувачам швидко організовувати свої речі, створювати образи та отримувати персоналізовані рекомендації. Наприклад, додаток ClothesAdvisor може аналізувати ваш гардероб і пропонувати найкращі комбінації на основі вашого стилю та погоди.

1.2 Виявлення та вирішення проблем

Під час аналізу існуючих додатків було виявлено кілька ключових проблем, які потребують вирішення.

Перша проблема – це обмежена функціональність у безкоштовних версіях. Багато додатків блокують основні функції, такі як створення комплектів або детальна статистика, і пропонують їх тільки в платній версії. Це обмежує доступність для широкого кола користувачів.

Друга проблема – це відсутність якісного офлайн-режиму. Більшість додатків вимагають постійного підключення до інтернету для роботи з основним функціоналом, що робить їх малопридатними для використання, наприклад, у подорожах або місцях зі слабким зв'язком.

Третя проблема – це примітивні алгоритми підбору одягу. Існуючі додатки часто пропонують комплекти, які не враховують реальні уподобання користувача або поточні умови. Наприклад, вони можуть рекомендувати легкий светр у спекотний день або не поєднувати речі, які насправді гармоніюють між собою за стилем.

Для вирішення цих проблем у серверному додатку було запропоновано:

- онлайн-режим до більшості функціоналу із серверним зберіганням даних;

- розширені алгоритми підбору одягу, які враховують не тільки колір і категорію, але й погоду, призначення одягу (спорт, прогулянка та інші), стиль користувача;

- безкоштовний доступ до всіх основних функцій без обмежень.

1.3 Постановка задачі

Для вирішення потреб користувачів, які вони відчувають при керуванні одягом, потрібно розробити програмний застосунок для керування гардеробом.

Програмний застосунок буде мати в собі підсистеми авторизації, зберігання та редагування одягу, рекомендацій на основі погоди, подій, кольорової гами.

1.3.1 Цільова аудиторія

Мобільний додаток керування гардеробом може бути корисною для широкого кола користувачів, кожен із яких має свої унікальні потреби та мотивацію для її використання. Основними категоріями цільової аудиторії є:

- люди, які прагнуть впорядкованості – користувачі, які хочуть мати структурований гардероб і швидко знаходити необхідні речі;
- мандрівники – ті, хто часто подорожує і потребує зручного способу планування одягу для поїздок;
- любителі моди – особи, які експериментують зі стилем, стежать за трендами та хочуть мати інструмент для створення нових образів.

Завдяки гнучкості та широкому функціоналу система може бути корисною як для особистого використання, так і для професійної діяльності у сфері моди та бізнесу.

1.4 Виявлення проблем

Сьогодні на ринку мобільних додатків існує чимало рішень для управління гардеробом, але більшість з них мають схожі проблеми та обмеження. Популярні додатки, такі як "Stylebook" (рис. 2.1), "Closet Space" (рис. 2.2) та "Smart Closet" (рис. 2.3), пропонують базовий функціонал: додавання фото одягу, створення комплектів і простий аналіз гардеробу. Однак вони часто не враховують індивідуальні потреби користувачів, працюють повільно або вимагають постійного підключення до інтернету.

Деякі додатки, наприклад "Pureple", намагаються запропонувати інтелектуальні рекомендації, але їх алгоритми обмежені простим підбором за кольором або категорією, без урахування таких важливих факторів, як поточна погода, стиль користувача або особливості матеріалів. Крім того, багато рішень не мають української локалізації, що ускладнює їх використання для місцевих користувачів.

Аналіз відгуків у Google Play показує, що користувачі часто скаржаться на складний інтерфейс, відсутність офлайн-режиму та обмежені можливості персоналізації. Це підтверджує необхідність створення нового додатку, який би усунув ці недоліки та запропонував більш гнучке та зручне рішення для управління гардеробом.

Також було проведено більш детальний аналіз вище перелічених існуючих аналогічних рішень, який дозволив виявити їх сильні та слабкі сторони, що стало основою для формування унікальних характеристик нашого додатка. Дослідження охопило популярні програми, доступні у Google Play, а також деякі спеціалізовані рішення. Нижче наведено порівняльний аналіз основних функцій та характеристик аналогів.

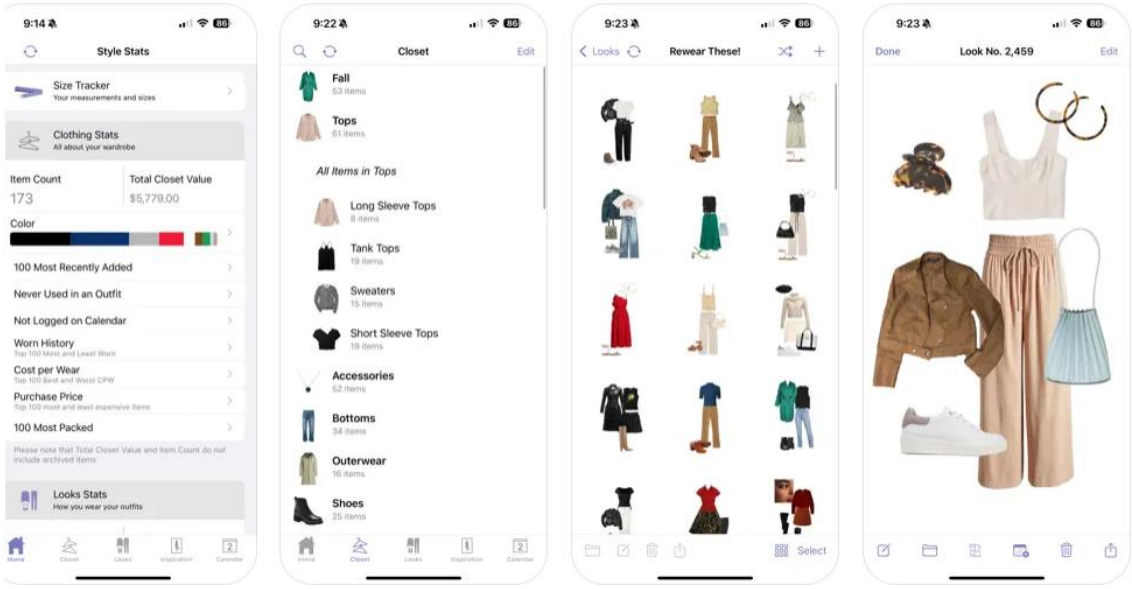


Рисунок 1.1 – Знімки екрану інтерфейсу додатку "Stylebook"(джерело [1])

Stylebook (рис. 1.1):

– основні функції: каталогізація одягу, створення образів, планування гардеробу;
– переваги: високий рівень деталізації характеристик одягу, можливість створення календаря носіння;
– недоліки: відсутність автоматичної генерації образів, обмежена безкоштовна версія.

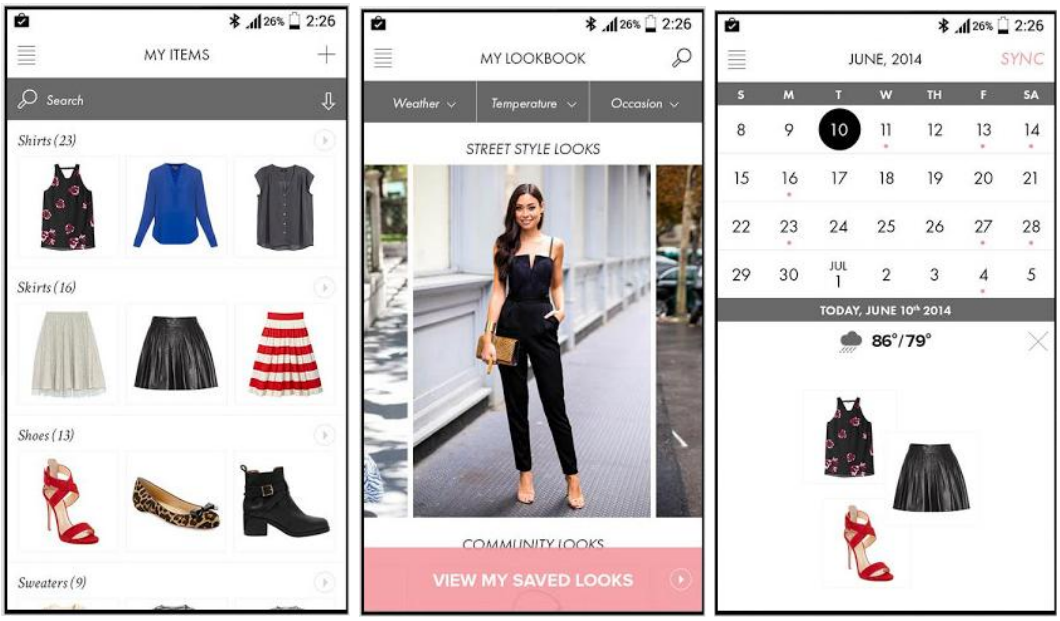


Рисунок 1.2 – Знімки екрану інтерфейсу додатку "Closet Space" (джерело [2])

ClosetSpace (рис. 1.2):

– основні функції: управління гардеробом, створення комбінацій, статистика;
– переваги: простий інтерфейс, можливість синхронізації між пристроями;
– недоліки: відсутність інтеграції з погодними сервісами, обмежена підтримка мов.

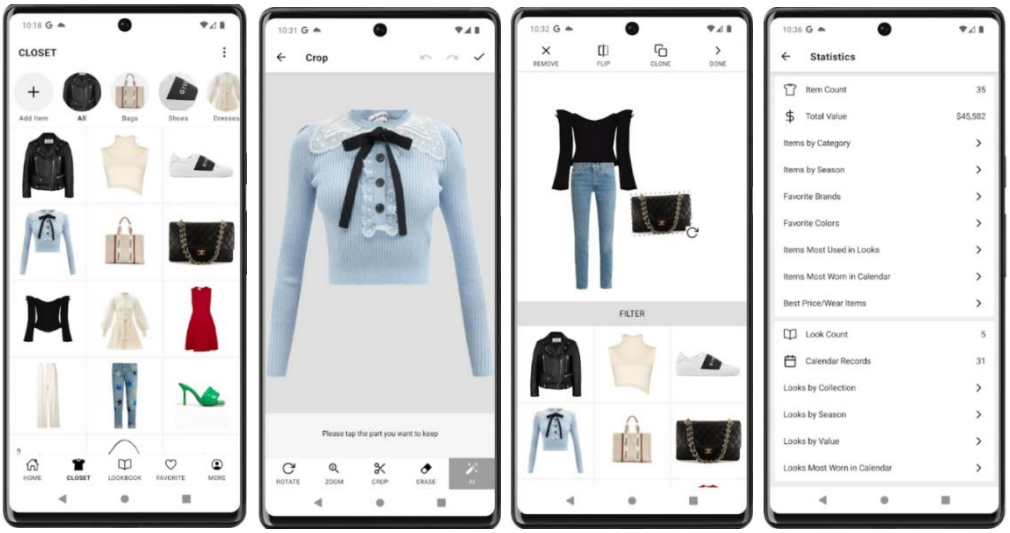


Рисунок 1.3 – Знімки екрану інтерфейсу додатку " Smart Closet" (джерело [3])

Smart Closet (рис. 1.3):

– основні функції: управління гардеробом, автоматичні рекомендації, статистика;
– переваги: використання штучного інтелекту для генерації образів, детальна статистика використання одягу;
– недоліки: висока вартість преміум-версії, вибагливість до апаратних ресурсів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги до серверної частини

Серверна частина мобільного додатка має реалізовувати наступний функціонал:

1) Авторизація та реєстрація користувачів:

1) Реєстрація за електронною адресою для збереження даних у обліковому записі.

2) Вхід у систему у за електроною поштою та паролем.

2) Редагування облікових даних, наприклад зміна електронної пошти та паролю.

3) Управління гардеробом:

1) Синхронізації даних між локальнми пристроями та сервером, де користувач буде вибирати, за якими даними (локальними або серверними) буде відбуватись синхронізація.

2) Додавання елементів одягу зі збереженням зображення на сервері.

3) Редагування характеристик одягу: назва, категорія (головний убір, верхній одяг, взуття тощо), сезонність, основний колір (формат RGB), матеріал , додаткові параметри: бренд, дата придбання, вартість (за бажанням).

4) Автоматична підбірка кольору для одягу на основі завантаженої фотографії.

5) Можливість позначання улюблених елементів одягу для покращення результату генерації комбінацій одягу.

6) Видалення фону на зображенні за бажанням користувача.

7) Видалення елементів гардеробу.

4)Генерація комбінацій одягу:

1) Ручне створення наборів (комбінацій) з можливістю вибору елементів гардеробу.

2) Автоматична генерація комбінацій на основі: стилю та кольорової гами, погодних умов та призначення (робота, відпочинок тощо), переваг користувача (кольорова палітра та інше).

5) Статистика:

1)Аналіз гардеробу за категоріями (кількість, середній вік елементів).

2) Сезонна статистика (відсоток речей для поточного сезону).

3) Розподіл за матеріалами.

4) Статистика використання комбінацій одягу.

2.2 Нефункціональні вимоги до серверної частини

- Всі дані користувачів (включаючи фотографії одягу та особисту інформацію) повинні передаватися через захищене з'єднання HTTPS.

- Авторизація користувачів має здійснюватися через JWT для безпечного доступу до даних.

- Підтримка щонайменше 100 запитів за хвилину.

- Всі критичні операції (входи в систему, зміни даних) мають бути записані в лог-файл із зазначенням часу.

- Файли користувачів (наприклад, фотографії одягу) повинні зберігатися на виділеному сервері.

- Паролі користувачів мають зберігатися у хешованому вигляді з використанням bcrypt.

- Час відповіді API для типових запитів (авторизація, отримання списку одягу, створення комбінації) не повинен перевищувати 500 мс у звичайних умовах.

- Серверна частина повинна бути реалізована з використанням сучасних засобів реалізації для забезпечення високої продуктивності та асинхронної обробки запитів.

2.3 Взаємодія серверної частини з мобільним додатком

- Сервер приймає запити на реєстрацію нових користувачів, обробляє їх та створює облікові записи в базі даних.

- Після авторизації користувача, сервер видає користувачу тимчасовий токен для використання при запитах до даних клієнта.

- Сервер обробляє запити на зміну облікових даних користувача (зокрема, електронної пошти та пароля) та забезпечує їх збереження з дотриманням вимог безпеки.

- Сервер отримує та зберігає оновлену інформацію про гардероб користувача, включаючи елементи одягу та комбінації.

- Сервер обробляє зображення одягу, завантажені користувачем, з функцією видалення заднього фону.

- Сервер генерує автоматичні комбінації одягу на основі даних гардеробу користувача.

ЗАРХІТЕКТУРА ТА ПРОЄКТУВАННЯ

3.1 UML проєктування ПЗ

Перед початком розробки серверного застосунку були визначені основні функції зі сторони користувача. Після детального аналізу була створена Use-case діаграма (див. рис. 3.1).

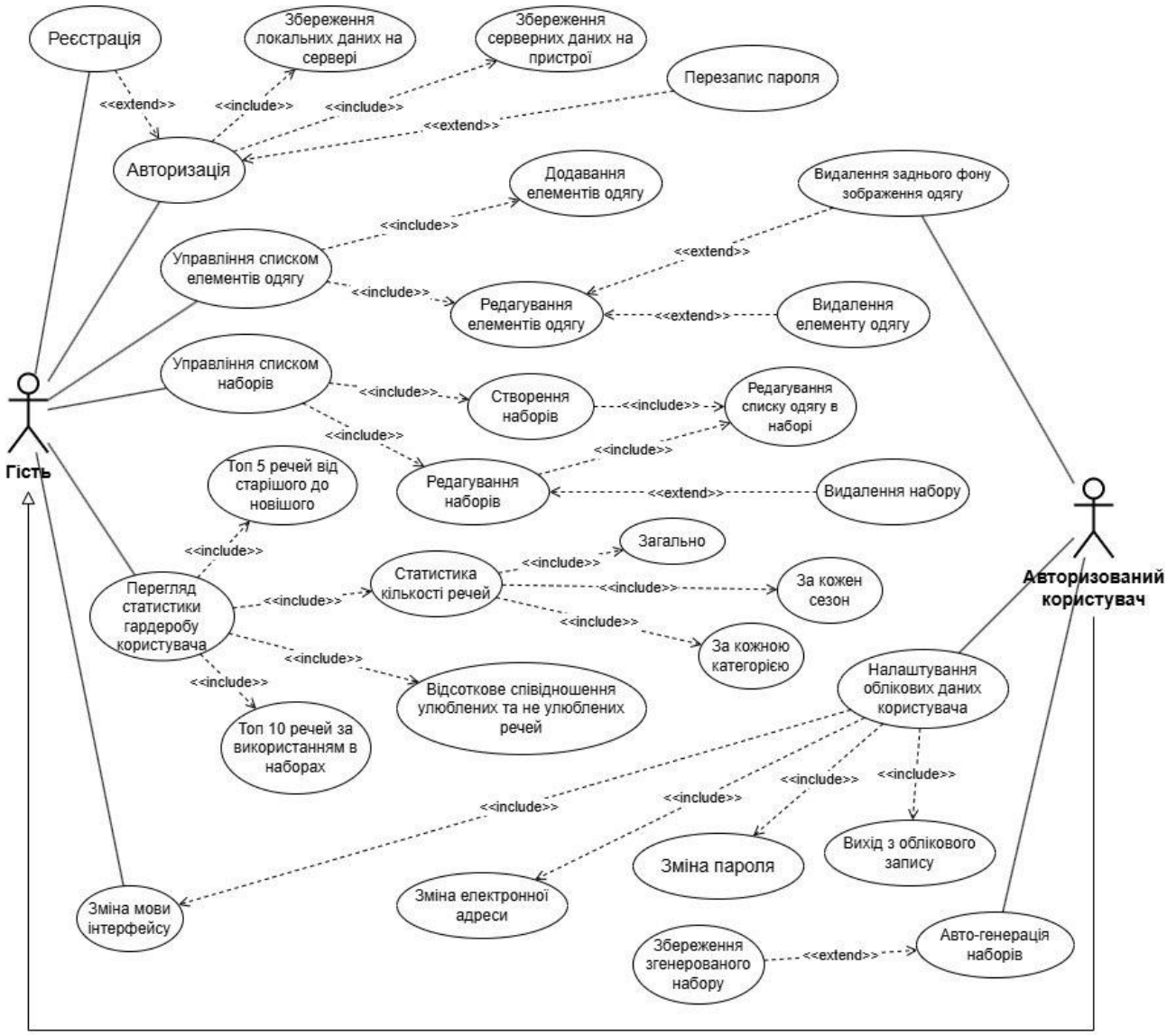


Рисунок 3.1 – Use-case діаграма системи керування гардеробом (рисунок виконаний самостійно)

Для розуміння всіх можливих дій користувача побудуєм діаграму діяльності (див. рис. 3.2 - 3.6):

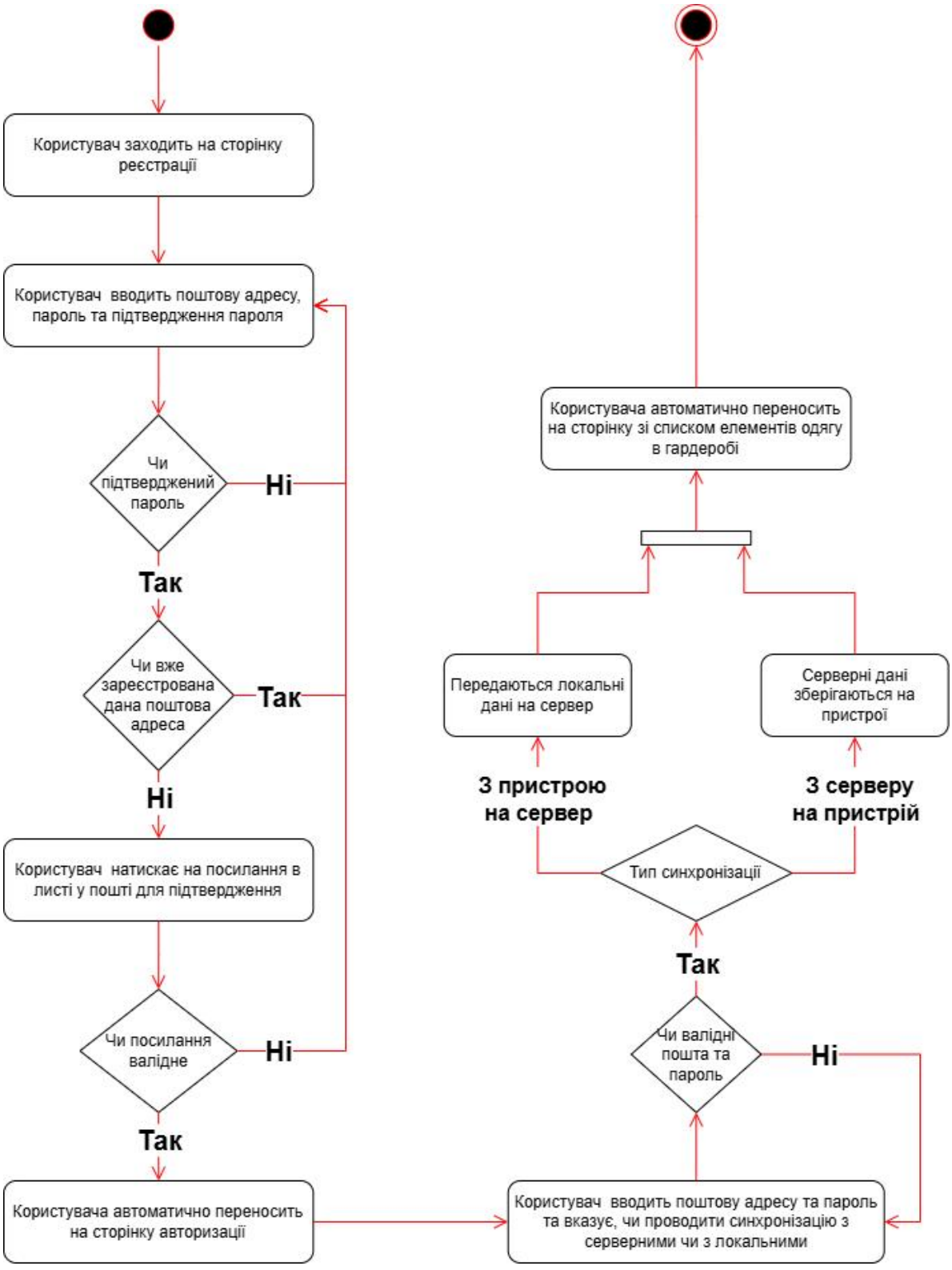


Рисунок 3.2 – Activity Diagram сторінки входу(рисунок виконаний самостійно)

Малюнок 3.2 представляє діаграму потоків користувача в системі керування гардеробом. Вона описує два основних процеси: реєстрацію користувача та синхронізацію даних гардеробу після авторизації.

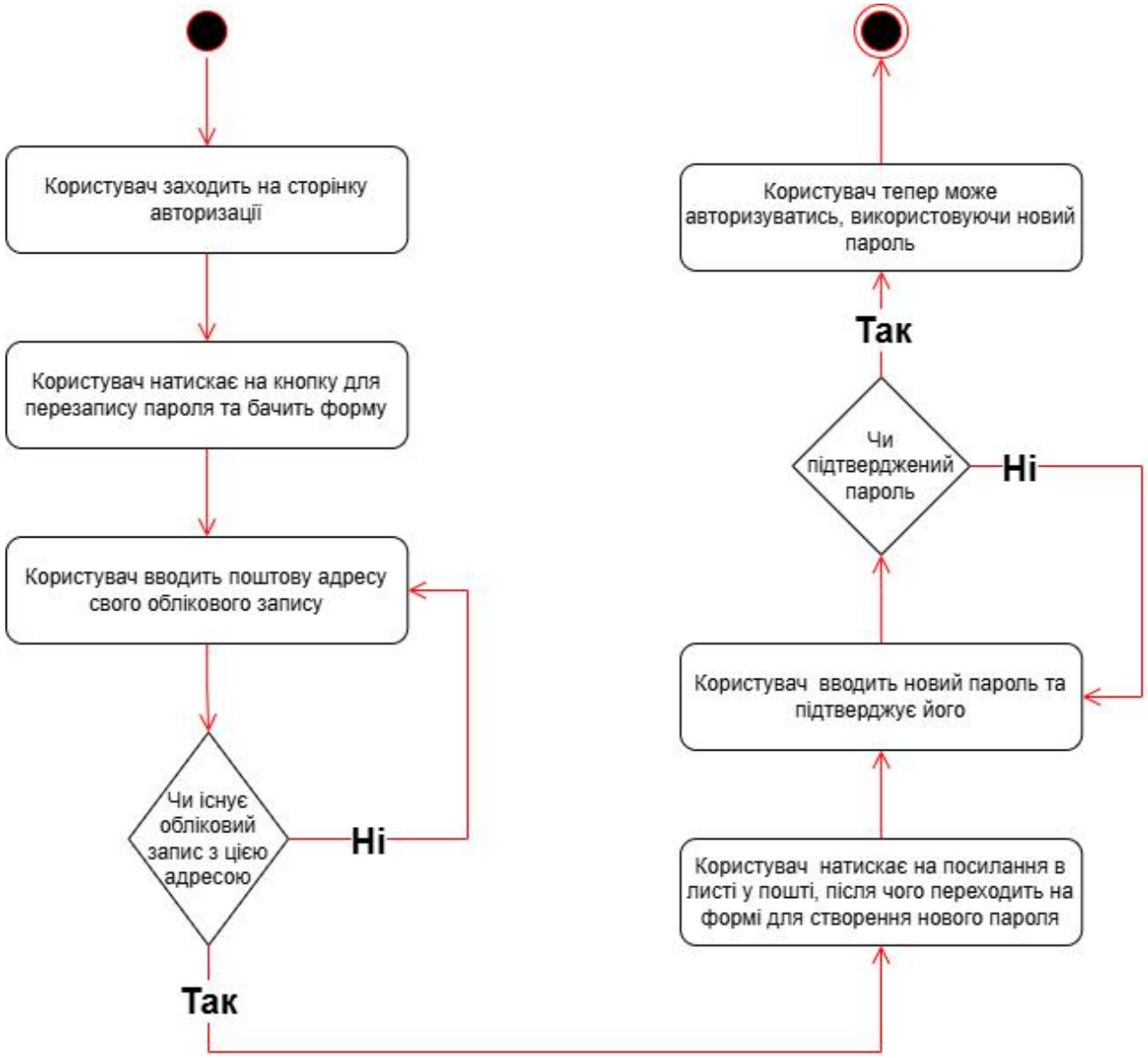


Рисунок 3.3 – Activity Diagram сторінки зміни паролю (рисунок виконаний самостійно)

Ця діаграма зображає процес відновлення пароля в системі. Користувач заходить на сторінку авторизації, натискає кнопку для скидання пароля й вводить свою електронну пошту. Якщо обліковий запис існує, користувачу

надсилається лист із посиланням. Після переходу за ним він вводить новий пароль і підтверджує його. Якщо пароль підтверджено правильно, користувач може авторизуватися з новим паролем.

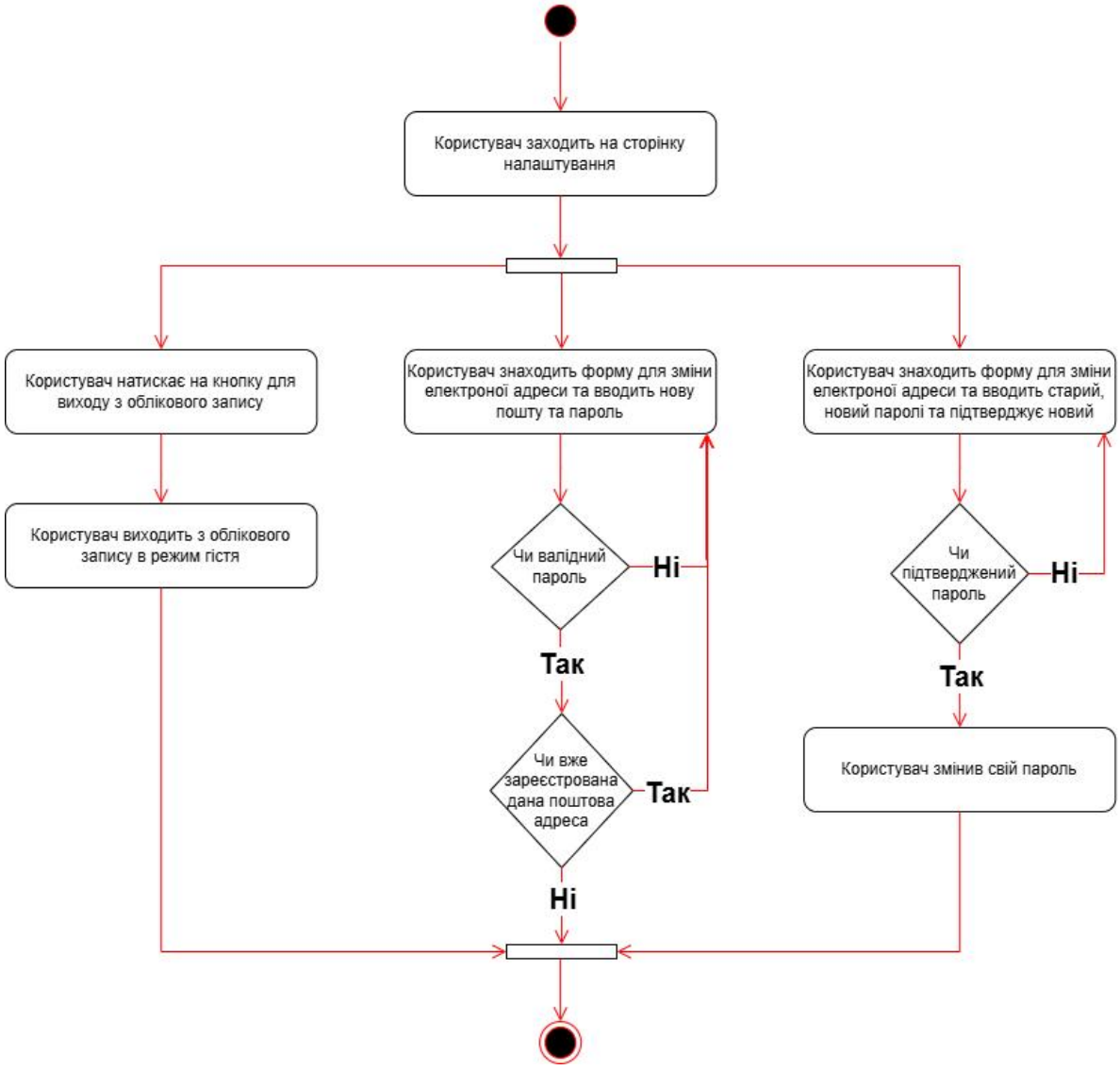


Рисунок 3.4 – Activity Diagram сторінки налаштувань (рисунок виконаний самостійно)

Ця діаграма описує процес взаємодії користувача зі сторінкою налаштувань. Користувач заходить на сторінку та має можливість змінити персональні дані, оновити пароль, електронну пошту, налаштувати параметри синхронізації або видалити обліковий запис. Після внесення змін система перевіряє коректність введених даних та зберігає їх на сервері. У разі успіху

користувач отримує повідомлення про успішне оновлення. У разі помилки – виводиться відповідне попередження, і користувач може повторити дію.

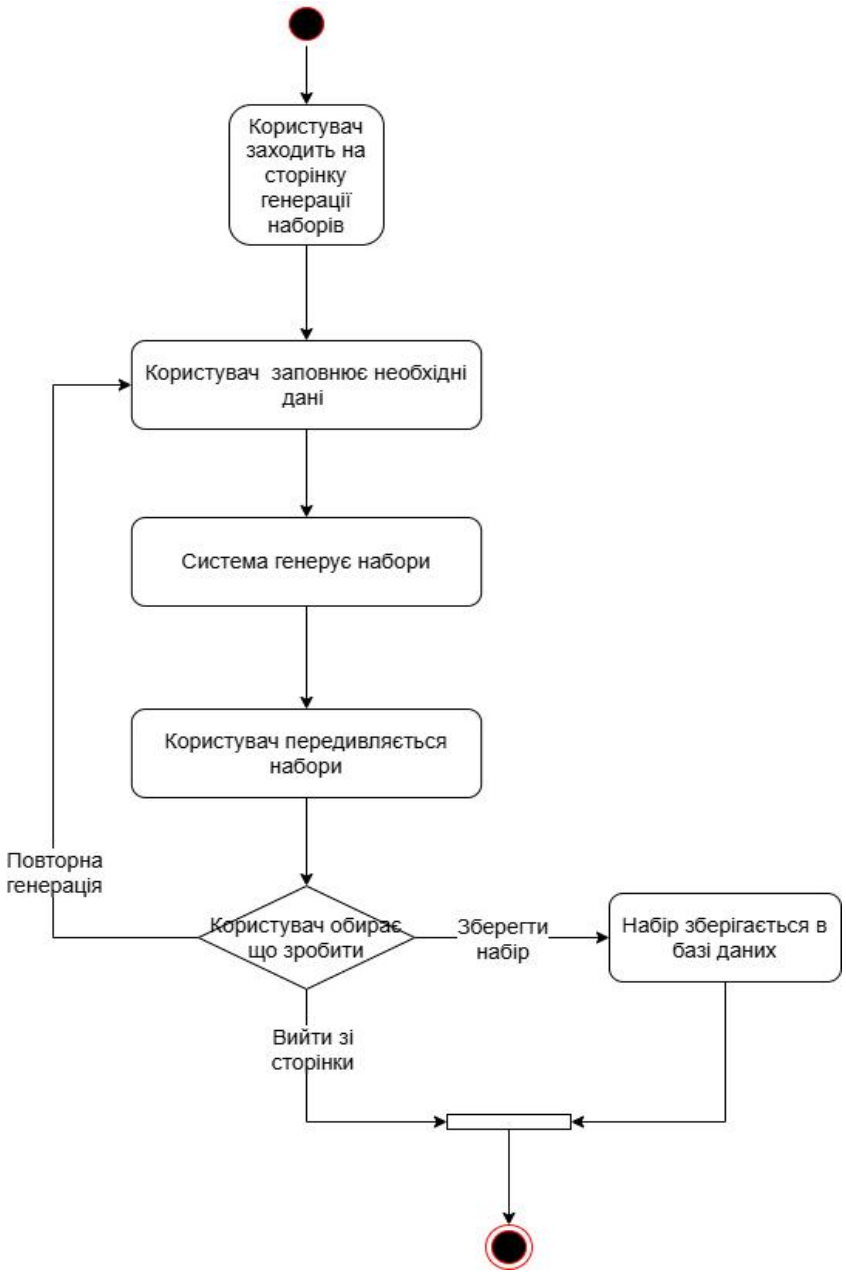


Рисунок 3.5 – Activity Diagram сторінки генерації наборів (рисунок виконаний самостійно)

Діаграма показує послідовність дій користувача під час формування набору одягу. Користувач відкриває сторінку генерації, після чого обирає параметри: погоду, подію, сезон або бажану колірну гаму. Система аналізує наявні елементи гардеробу та формує один або кілька рекомендованих наборів. Користувач може зберегти набір або згенерувати інший варіант. Після підтвердження збережений набір стає доступним у профілі користувача.

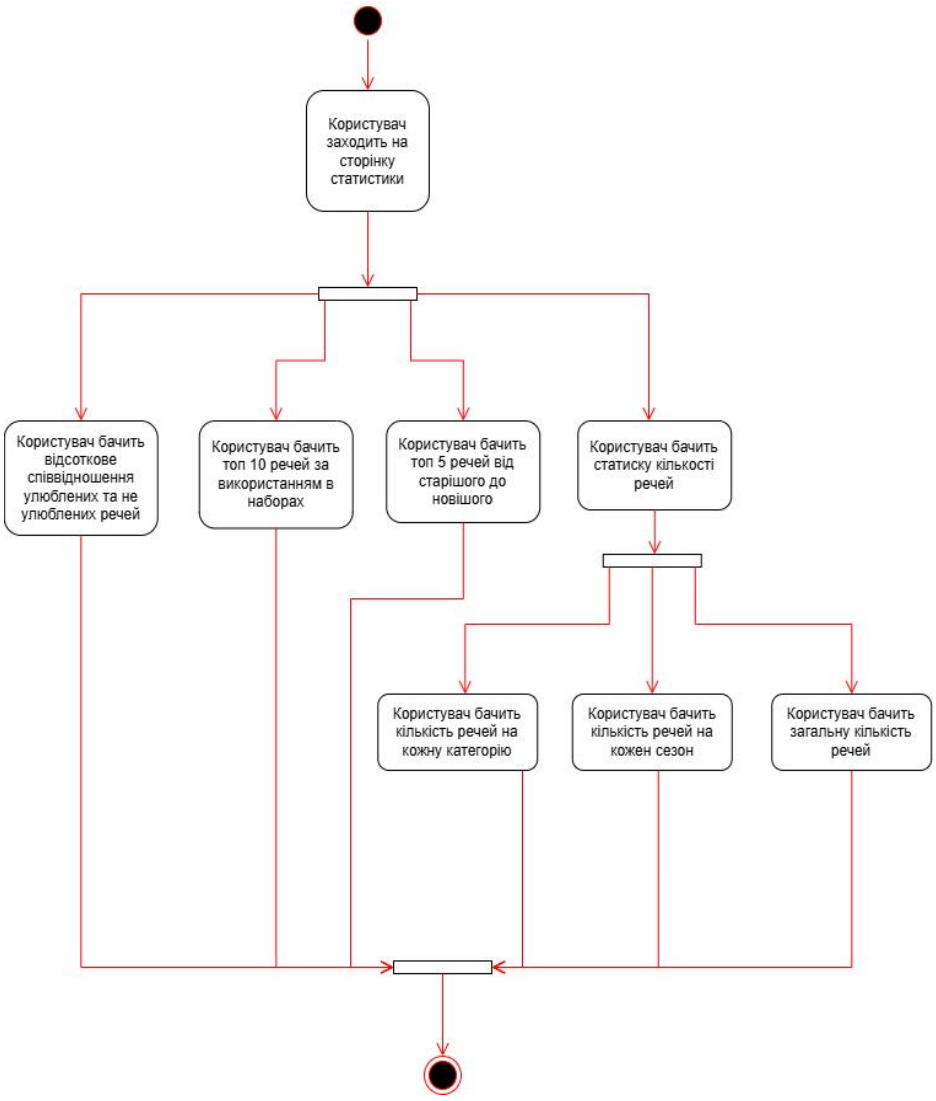


Рисунок 3.6 – Activity Diagram сторінки статистики (рисунок виконаний самостійно)

Діаграма описує взаємодію користувача зі сторінкою статистики. Після відкриття сторінки система автоматично отримує дані з облікового запису:

кількість використань одягу, найчастіше обрані елементи, сезонну активність тощо.

3.2. Проєктування архітектури ПЗ

Архітектура системи побудована за принципом клієнт-сервер. Клієнтом є Android-додаток , який взаємодіє з сервером через REST API. Для обрання технології для написання коду було проаналізовано вимоги до северної частини і було обрано мову програмування Python, так як вона містить багато високорівневих бібліотек для роботи з базами даних, обробки зображенньь.
Для реалізації системи рекомендацій було реалізовано паатерн проєктування Strategy. Відповідно до обраної мови програмування було обрано фреймворк FastAPI, так як він є найбільш вдалим рішенням для побудови REST API серверу. У порівнянні з Django, який має модуль Django REST Framework, FastAPI не обмежує структуру проєкту і не створює надмірної складності. Flask, хоча й легкий, не має вбудованих механізмів валідації і документації, що ускладнює масштабування. На сервері також розміщена реляційна база даних MySQL, яка використовується для зберігання основної інформації (користувачі, речі, тощо). Доступ до неї здійснюється лише з боку серверного застосунку. Зображення зберігаються у файловій системі сервера в окремому каталозі (/server/uploads/) і передаються клієнту через FastAPI у вигляді URL або байтового потоку.

Для розуміння взаємодії між різними частинами системи була розроблена діаграма розгортання:

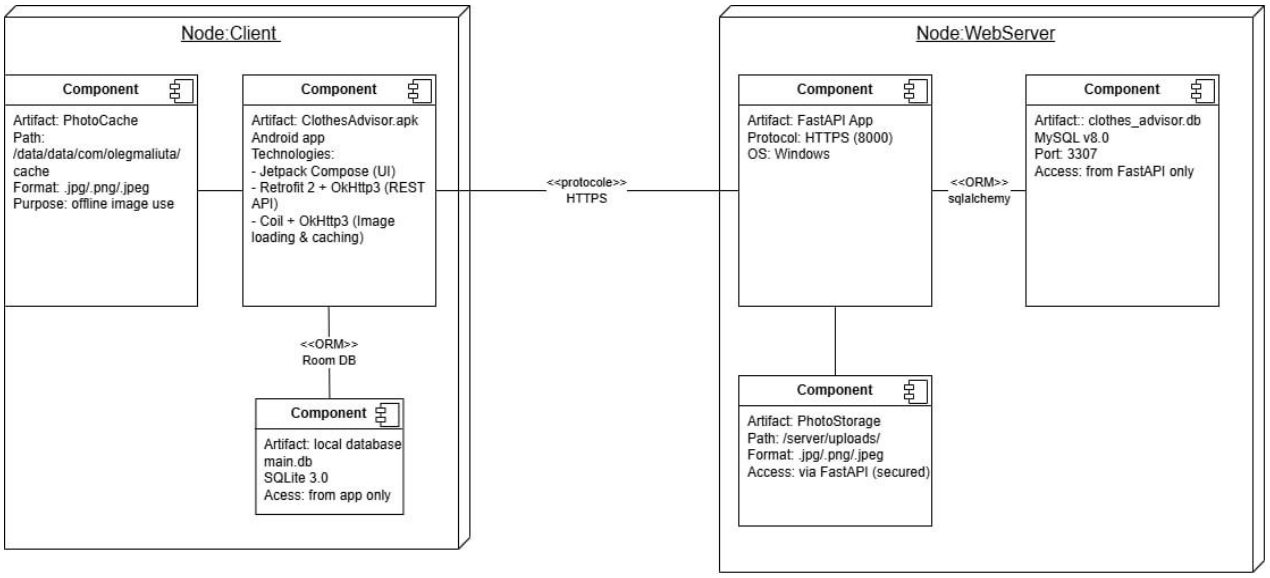


Рисунок 3.7 – Deployment Diagram системи керування гардеробом (рисунок виконаний самостійно)

Серверний додаток побудований за допомогою багатошарової архітектури. Сервер поділений на наступні шари:

- database, який відповідає за під’єдання до бази даних;

- user_manager, який відповідає за всі дії користувача та синхронізацію даних;

- close_manager, який відповідає за редагування елементів одягу та наборів;

- stats_manager, який відповідає за вивід статистики;

- recommendation_manager, який відповідає за систему рекомендації наборів.

У розробленій системі реалізовано модуль формування рекомендацій для користувача. Виходячи з того, що користувач може задавати різні параметри при запиті на рекомендацію — такі як бажаний колір, місцеположення та дата(для отримання погоди), тип події (прогулянка, спортивне тренування тощо) — застосування одного універсального алгоритму було б неефективним. Замість цього реалізовано патерн Strategy(рис. 3.10), який дозволяє легко додавати нові варіанти алгоритмів під різні умови.

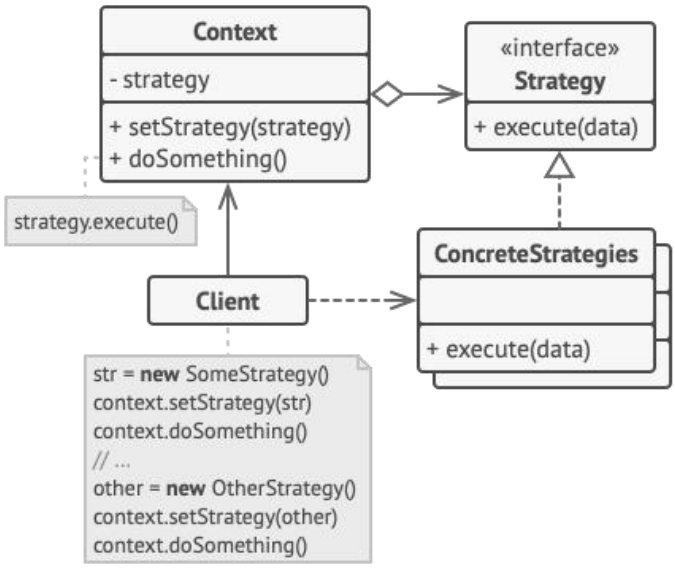


Рисунок 3.8 – Структура паттерну Strategy(рисунок взято з сайту, джерело [7])

Для цього було створено абстрактний клас за допомогою вбудованого Python модуля abc з єдиним абстрактним методом get_recommendation(user_data):

```
#  Strategy interface

class RecommendationStrategy(ABC):

    @abstractmethod

    def get_recommendation(self, user_data: Dict[str, Any]) ->
List[Dict[str, Any]]:

        Pass
```

Відповідно маємо класи, які будуть реалізовувати цей метод для всіх можливих варіантів даних, введених користувачем:

```
class ColorPreferenceStrategy(RecommendationStrategy):

    def get_recommendation(self, user_data: Dict[str, Any]) ->
List[Dict[str, Any]]:
```

```
            # Filter clothes based on the user's preferred color
            pass
    class LocationWeatherStrategy(RecommendationStrategy):
        def get_recommendation(self, user_data: Dict[str, Any]) ->
List[Dict[str, Any]]:
            # Select clothes based on current weather in the user's
location
            pass
    # Other possible strategies …
    #  Strategy context class
    class RecommendationContext:
        def __init__(self, strategy: RecommendationStrategy):
            self.strategy = strategy

        def set_strategy(self, strategy: RecommendationStrategy):
            self.strategy = strategy
        def recommend(self, user_data: Dict[str, Any]) -> List[Dict[str,
Any]]:
            return self.strategy.get_recommendation(user_data)
```

3.3. Проєктування структури зберігання даних

У даному проєкті було прийнято рішення використовувати локальну базу даних на мобільному пристрої та віддалене сховище на сервері. Як віддалене сховище використовується реляційна база даних MySQL, яка забезпечує зберігання та обробку структурованої інформації про користувачів, елементи гардеробу, набори одягу тощо. Основною причиною такого вибору стала чітко

структурована природа даних, які обробляє система. Усі ключові сутності, такі як користувачі, елементи гардеробу, набори мають визначені атрибути і чіткі зв’язки між собою. Наприклад, кожен користувач може мати багато одиниць одягу. Усе це логічно відображається у вигляді таблиць із первинними та зовнішніми ключами, що природно реалізується саме в реляційній моделі.

Основні сутності бази даних:

Користувач(таблиця users):

id - число, унікальний ідентифікатор користувача

email - електронна пошта,

password - пароль,

created_at - час створення

synchronized_at - дата останньої синхронізації даних

is_email_verified - чи підтверджений пароль.

Елемент одягу (таблиця clothing_items):

id – ціле число, унікальний ідентифікатор речі;

filename – рядок, шлях до зображення одягу (ім’я файлу), є унікальним і обов’язковим;

name – рядок, назва одягу, обов’язкове поле;

category – перелік (enum), категорія одягу (наприклад взуття, аксесуари);

season – перелік (enum), сезон, для якого призначено одяг (зима, літо, весна, осінь);

red / green / blue – цілі числа (від 0 до 255), компоненти кольору RGB;

material – рядок, матеріал, з якого виготовлено одяг (наприклад, бавовна, шкіра);

brand – рядок, бренд одягу (необов’язкове поле);

purchase_date – дата, коли було придбано річ (опційно);

price – десяткове число, вартість одягу в умовних одиницях;

is_favorite – булеве значення, вказує, чи є річ улюбленою для користувача.

Комбінація одягу (таблиця clothing_combinations):

id – ціле число, унікальний ідентифікатор комбінації;

name – рядок, назва комбінації (наприклад, "Зимова прогулянка", "Офіс");

owner_id – зовнішній ключ, який вказує на користувача (user), якому належить комбінація;

owner – зв’язок із таблицею users, що дозволяє отримати всі комбінації для певного користувача.

Для реалізації зв’язку "багато-до-багатьох" між одягом та комбінаціями використовується проміжна таблиця clothing_combination_items. Вона відповідає таблиці зі стовпцями:

combination_id – зовнішній ключ до таблиці clothing_combinations;

item_id – зовнішній ключ до таблиці clothing_items.

Це дозволяє одній речі входити в декілька комбінацій і навпаки — до однієї комбінації входити кільком одиницям одягу.

ORM-реалізація

У проєкті використовується ORM-бібліотека SQLAlchemy, яка забезпечує відповідність між Python-класами (моделями) та таблицями бази даних. Це дозволяє працювати з БД, використовуючи об'єктно-орієнтований підхід, а не безпосередньо SQL-запити.

Відповідно маємо схему БД (схему наведено в Додатку Г) та діаграму БД(див. рис. 4.9).

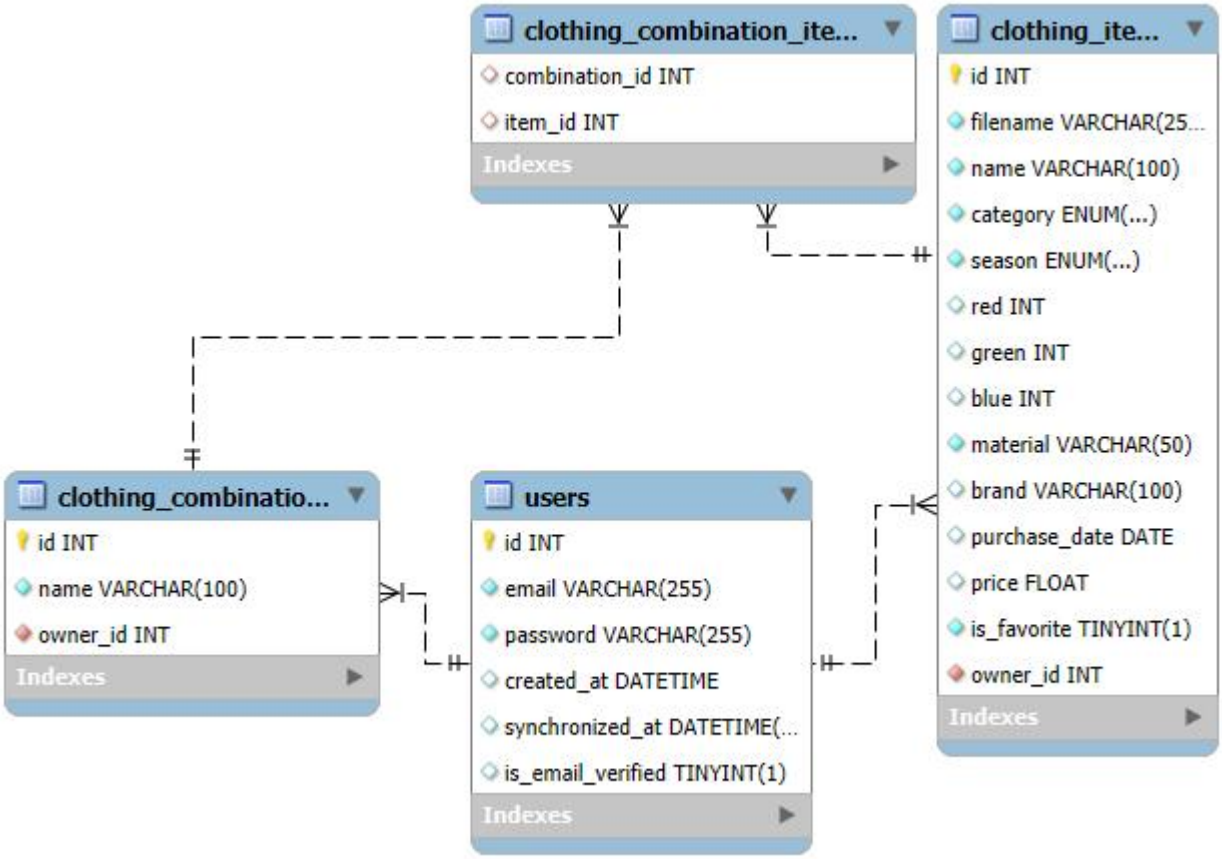


Рисунок 3.9 – ER діаграма бази даних (Рисунок створений самостійно)

Зберігання зображень

Фотографії елементів одягу не зберігаються у базі даних у вигляді BLOB-даних. Натомість, зображення зберігаються на сервері у файловій системі, а в БД зберігається лише шлях до відповідного зображення (filename). Це дозволяє зменшити розмір бази даних і спростити обробку файлів.

4ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір необхідних бібліотек

Для реалізації серверної частини мобільного додатку з управління гардеробом було обрано набір сучасних та перевірених бібліотек, які забезпечують стабільну роботу, зручну розробку та масштабованість проєкту. Завантаження сторонніх бібліотек є невід'ємною частиною сучасної розробки серверних додатків. Власноручне написання всього функціоналу — від обробки HTTP-запитів до роботи з базами даних чи реалізації безпеки — є надзвичайно трудомістким, схильним до помилок і малоефективним. Обрані бібліотеки охоплюють такі аспекти, як створення REST API, робота з базою даних, забезпечення аутентифікації та авторизації, обробка кольорів, а також логування та тестування.

4.1.1 Sqlalchemy

Для реалізації взаємодії з базою даних у бекенд-застосунку було обрано бібліотеку SQLAlchemy — один із найпопулярніших ORM-фреймворків для мови Python.

Причини вибору SQLAlchemy:

- Зручність роботи з базою даних: SQLAlchemy дозволяє працювати з таблицями бази даних як з об'єктами Python. Це спрощує читання й підтримку коду, особливо в складних структурах даних.

- Гнучкість: SQLAlchemy поєднує ORM і можливість написання «чистих» SQL-запитів при потребі, що дозволяє розробнику обирати між абстракцією та повним контролем над запитами.

- Інтеграція з FastAPI та MySQL: SQLAlchemy добре інтегрується з FastAPI — фреймворком, на якому базується бекенд, що дозволяє легко використовувати залежності, асинхронну обробку та автоматичну генерацію схем та обраною СУБД MySQL.

4.1.2 PyJWT

Для реалізації механізму автентифікації та авторизації у бекенд-частині застосунку було обрано бібліотеку PyJWT, яка забезпечує створення, підпис і перевірку JSON Web Token (JWT). Причини вибору PyJWT:

- PyJWT повністю реалізує офіційний стандарт JWT (RFC 7519), який є поширеним способом передачі безпечних токенів між клієнтом і сервером у RESTful-архітектурі.

- Простота використання: бібліотека має зрозумілий API для створення і перевірки токенів. Створення JWT займає лише кілька рядків коду, що робить її зручною у використанні під час автентифікації користувачів.

- PyJWT підтримує різні алгоритми підпису (наприклад, HS256, RS256), що дозволяє захистити дані від підробки й забезпечити довіру між клієнтом і сервером.

- PyJWT легко інтегрується з FastAPI через систему залежностей та middleware, що дозволяє створювати власні механізми авторизації з перевіркою токена при кожному запиті.

4.1.3 Bcrypt

Для безпечного зберігання паролів користувачів у базі даних було обрано бібліотеку bcrypt. Вона реалізує надійний алгоритм хешування, спеціально розроблений для захисту паролів. Bcrypt автоматично додає salt та є стійким до атак типу brute-force завдяки своїй повільності, що ускладнює масовий перебір хешів. Завдяки своїй надійності та простоті використання, bcrypt став стандартом для захищеного зберігання паролів.

4.1.4 Python-dotenv.

Для зберігання і управління конфіденційними даними, такими як API-ключі, налаштування бази даних, порти і інші змінні оточення, у проєкті було використано бібліотеку python-dotenv.

Вибір саме цієї бібліотеки обумовлений її простотою і зручністю інтеграції з Python-проєктами. Вона дозволяє зберігати всі змінні середовища у файлі .env, який не включається у систему контролю версій, що забезпечує

безпеку і гнучкість при роботі з конфігураціями у різних середовищах (локальна розробка, тестування, продакшн).

Завдяки python-dotenv:

- Конфігураційні дані централізовані в одному файлі, що спрощує налаштування програми.

- Змінні оточення автоматично завантажуються при запуску додатку, що зменшує ризик помилок через неправильні або відсутні налаштування.

- Забезпечується відокремлення коду від конфіденційних параметрів, що відповідає принципам безпеки і кращим практикам розробки.

- Забезпечується ізоляція проектних залежностей від системних пакетів та запобігання конфліктам версій бібліотек між різними проектами.

- Є можливість легкої передачі проекту з усіма залежностями (через файл requirements.txt).

Забезпечення відтворюваності робочого середовища на різних машинах.

4.1.5 Pydantic

Для перевірки та серіалізації даних у застосунку було обрано бібліотеку Pydantic. Вона забезпечує зручний спосіб визначення моделей даних на основі анотацій типів Python, автоматично перевіряє вхідні дані на відповідність типам і формує зрозумілі повідомлення про помилки. Завдяки цьому спрощується обробка запитів та відповідей у FastAPI, підвищується надійність коду та зменшується кількість ручної валідації.

4.1.6 FastAPI-Mail

У проєкті було використано бібліотеку FastAPI-Mail для реалізації функціональності надсилання електронних листів (наприклад, для підтвердження реєстрації або відновлення паролю). Вона інтегрується з FastAPI, підтримує асинхронну відправку листів, шаблони повідомлень і конфігурацію SMTP-серверів. Завдяки цьому бібліотека дозволяє легко реалізувати поштові функції з мінімальними витратами коду та часу.

4.1.7 ColorThief

У проєкті бібліотека ColorThief використовується для автоматичного визначення домінантного кольору зображення одягу, яке завантажує користувач. Це дозволяє здійснювати колірну класифікацію одягу без необхідності ручного введення кольору. Бібліотека проста у використанні, ефективно працює з зображеннями та надає готову функціональність для аналізу кольорової палітри, що значно полегшує процес обробки візуального контенту в бекенді.

4.1.8 Rembg

Бібліотека Rembg була обрана для автоматичного видалення фону із зображень одягу, що завантажуються користувачами. Це необхідно для покращення якості аналізу зображень — зокрема, для точнішого визначення кольору та кращої візуалізації елементів гардеробу без зайвих візуальних шумів.

Rembg базується на сучасних алгоритмах комп’ютерного зору, працює офлайн, легко інтегрується в бекенд і забезпечує достатню точність обробки, що робить її оптимальним вибором для мобільного застосунку з обробкою зображень.

4.1.9 Pytest

Бібліотека pytest була обрана як основний інструмент для тестування функціональності бекенду застосунку. Вона забезпечує простий, зрозумілий синтаксис для написання тестів, а також потужні можливості для організації, виконання та налагодження тестових сценаріїв.

Основні причини вибору саме pytest:

- Гнучкість: підтримка фікстур, параметризації та плагінів дозволяє масштабувати систему тестування відповідно до зростання проєкту.

- Сумісність з FastAPI: pytest добре інтегрується з FastAPI, зокрема через використання тестового клієнта TestClient, що дозволяє перевіряти API-ендпоінти у відокремленому середовищі.

4.2 Вибір та реалізація API для отримання погодних даних

Для реалізації персоналізованих рекомендацій одягу залежно від погодних умов необхідно було отримувати актуальну або прогнозовану інформацію про температуру, тип опадів, вітер тощо. Одним із найважливіших програмних рішень стало підключення до стороннього погодного API.

У рамках даного проєкту було обрано OpenWeatherMap API, оскільки він надає:

- детальні погодні дані в розрізі координат (широта, довгота),

- підтримку як поточних, так і прогнозованих погодних умов,

- можливість безкоштовного використання для базових потреб (з обмеженим обсягом запитів),

- простий та добре задокументований інтерфейс REST API.

Відповідно до обраного сервісу реалізовуєм функцію збору погодних даних на основі місцеположення та бажаної дати:

```
    def get_weather_at_time_by_coords(lat: float, lon: float, target_time: str, api_key: str = "9eb8fb241802a2c7631250c97cbe31cd"):
        url = "https://api.openweathermap.org/data/2.5/forecast"
        params = {
            "appid": api_key,
            "lat": lat,
            "lon": lon,
            "units": "metric"
        }

        response = requests.get(url, params=params)
        data = response.json()

        if response.status_code != 200 or "list" not in data:
            return f"❌ Failed to retrieve data for coordinates: ({lat}, {lon})."

        forecasts = data["list"]
        target_time_dt = datetime.strptime(target_time, "%Y-%m-%d %H:%M:%S")
```

```
        for forecast in forecasts:
            forecast_time = datetime.strptime(
                forecast["dt_txt"], "%Y-%m-%d %H:%M:%S")
            if forecast_time == target_time_dt:
                temp = forecast["main"]["temp"]
                weather = forecast["weather"][0]["description"]
                icon = forecast["weather"][0]["icon"] if "icon" in
forecast["weather"][0] else "None"
                return temp, weather, icon

        closest_forecast = min(
            forecasts,
            key=lambda x: abs(datetime.strptime(
                x["dt_txt"], "%Y-%m-%d %H:%M:%S") - target_time_dt)
        )
        logging.info(
            f"Closest forecast found for {closest_forecast['dt_txt']}
with temperature {closest_forecast['main']['temp']}°C.")
        temp = closest_forecast["main"]["temp"]
        weather = closest_forecast["weather"][0]["description"]
        icon = closest_forecast["weather"][0]["icon"] if "icon" in
closest_forecast["weather"][0] else "None"
        return temp, weather, icon
```

4.3 Створення та активація віртуального середовища venv

Для ізоляції робочого середовища проекту та забезпечення керованості залежностей у розробці було використано віртуальне середовище Python — venv (джерело [6]). Використання віртуального середовища дозволяє створити окремий простір із власними бібліотеками і пакетами, не впливаючи на глобальні налаштування системи.

Для створення віртуального середовища у каталозі проекту виконується команда:

`python -m venv venv`, де:

- python — виклик інтерпретатора Python;

- m venv — модуль для створення віртуального середовища;

- venv — ім’я каталогу, у якому буде створене середовище.

Для активації створеного віртуального середовища необхідно виконати відповідну команду залежно від операційної системи:

- **`venv\Scripts\activate`** для Windows,
- **`source venv/bin/activate`** для Linux.

Відповідно до обраного фреймворку FastAPI, файлу залежностей requirements.txt та створеного віртуального середовища напишемо скрипти інсталювання залежностей та запуску сервера. Для ОС Windows:

```
@echo off
echo Activating virtual environment...
call venv\Scripts\activate

echo Installing dependencies...
pip install -r requirements.txt
echo Installation completed.

Pause

@echo off
echo Activating virtual environment...
call venv\Scripts\activate

echo Starting FastAPI server...
python -m app.seeding_manager.seed

echo Server running at http://127.0.0.1:8000/docs
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

Pause
```

Для ОС Linux:

```
#!/bin/bash

echo "Activating virtual environment..."
source venv/bin/activate

echo "Installing dependencies..."
pip install -r requirements.txt
```

```
echo "Installation completed."

read -p "Press any key to continue..."

#!/bin/bash

echo "Activating virtual environment..."
source venv/bin/activate

echo "Starting FastAPI seeding..."
python -m app.seeding_manager.seed

echo "Server running at http://127.0.0.1:8000/docs"
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

read -p "Press any key to exit..."
```

Для забезпечення створення початкових даних було виконано команду `python -m app.seeding_manager.seed`

Код seeding_manager.seed наведено в Додатку Д

4.4 Налаштування підключення до бази даних та управління моделями

Для забезпечення єдиного доступу до бази даних, відображення класів моделі в БД через ORM Sqlalchemy(джерело [7]) було створено файл database.py:

```
from sqlalchemy import create_engine, inspect
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from dotenv import load_dotenv
import os
# Load environment variables from the .env file
load_dotenv()
# Get DATABASE_URL from environment variables
DATABASE_URL = os.getenv("DATABASE_URL")
# Create the engine for connecting to the database
engine = create_engine(DATABASE_URL, pool_size=10, max_overflow=20)
# Create SessionLocal for database sessions
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
# Base class for creating tables
Base = declarative_base()
# Import models
from app.model import *
Base.metadata.create_all(bind=engine)
# Dependency for getting the database session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Завдяки створеній функції get_db() в будь-якій точці коду можна отримати сесію до бази даних, а завдяки об’єкту Base можна створити таблиці бази даних на основі моделі за допомогою ORM. Для цього клас необхідно наслідувати від Base:

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String(255), unique=True, nullable=False)
    password = Column(String(255), unique=False, nullable=False)
    created_at = Column(DateTime, default=datetime.now(timezone.utc))  # Date and time of creation
    synchronized_at = Column(DATETIME(fsp=6), default=datetime.now(timezone.utc))  # Date and time of creation first
    is_email_verified = Column(Boolean, default=False)
class ClothingItem(Base):
    __tablename__ = "clothing_items"
    #other fields
class ClothingCombination(Base):
    __tablename__ = "clothing_combinations"
    #other fields
```

4.5 Забезпечення реєстрації та авторизації.

Відповідно до обраної технології PyJWT (джерело [8]) та алгоритму хешування bcrypt напишемо функції хешування паролю, перевірки паролю, створення токену доступу:

```
# Hash
def hash_password(password: str) -> str:
    salt = bcrypt.gensalt()
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)
    return hashed_password.decode('utf-8')

# Verify
def verify_password(plain_password: str, hashed_password: str) -> bool:
    return bcrypt.checkpw(plain_password.encode('utf-8'), hashed_password.encode('utf-8'))

# Generate JWT token
def create_access_token(data: dict, expires_delta: timedelta | None = None):
    to_encode = data.copy()
    expire = datetime.now(
        timezone.utc) + (expires_delta or timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})  # Додаємо час дії токену
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Після забезпечення хешування паролю та створення токену доступу можна написати функції реєстрації та авторизації (див. Додаток Е), синхронізації даних.

Для підтвердження електронної пошти напишемо код відправки та перевірки посилання підтвердження за допомогою fastapi_mail (див. Додаток Ж).

4.6 Обробка зображенньь елементу одягу.

Для підвищення якості взаємодії користувача з системою та можливості формування візуально привабливих комбінацій одягу, реалізовано функціонал обробки зображень елементів гардеробу. Зокрема, при завантаженні

зображення здійснюється перевірка його розміру, збереження на сервері, а також видалення фону — з метою стандартизації зовнішнього вигляду та забезпечення прозорості зображення.

Обробка реалізована у вигляді двох основних функцій:

- збереження зображення на сервері:

```
# Function for saving a file on the server
def save_file(file: UploadFile):
    # Generate a unique filename for the file
    file_extension = file.filename.split('.')[-1]
    unique_filename = f"{uuid.uuid4()}.{file_extension}"
    file_path = os.path.join(UPLOAD_DIR, unique_filename)

    # Create the directory if it doesn't exist
    os.makedirs(UPLOAD_DIR, exist_ok=True)

    # Read the entire content of the file
    content = file.file.read()

    # Check file size
    if len(content) > MAX_FILE_SIZE_BYTES:
        return JSONResponse(
            status_code=400,
            content={"detail": f"File size more than {MAX_FILE_SIZE_MB} MB."}
        )

    # Write the file to the disk
    with open(file_path, "wb") as f:
        f.write(content)

    return unique_filename
```

- видалення фону:

```
from rembg import remove
def remove_background_preview(filename: str) -> tuple[str, BytesIO]:
    input_path = os.path.join(UPLOAD_DIR, filename)
```

```
    with open(input_path, 'rb') as input_file:
        input_data = input_file.read()
        output_data = remove(input_data)

    output_io = BytesIO(output_data)


    base_name = os.path.splitext(filename)[0]
    output_filename = f"{base_name}_bg_removed.png"

    return output_filename, output_io
```

У процесі видалення фону використовується бібліотека rembg(джерело [9]), яка дозволяє позбутись зайвих елементів заднього плану, залишаючи лише об'єкт одягу.

4.7 Створення системи рекомендацій.

Система рекомендацій реалізована як модуль, що працює на основі погодних умов, подій, кольорів та класифікацій одягу. Вона складається з низки компонентів, кожен із яких виконує специфічну функцію для формування персоналізованих рекомендацій:

- clothing_grouping.json містить структуру класифікації типів одягу: верхній одяг, взуття, аксесуари, тощо. Використовується для логічної групізації елементів гардеробу та формування повних ансамблів:

```
{
  "outerwear": [
    "jacket",
    "coat",
    "hoodie",
    "sweater"
  ],
  "tops": [
    "tshirt",
    "blouse",
    "tank_top"
  ]
...
```

```
}
```

- event_recommendations.json містить структуру відповідності між одягом і типами подій (наприклад, ділова зустріч, спорт, побачення). Визначає, які елементи гардеробу доречні для тієї чи іншої ситуації:

```
{
  "tshirt": {
    "event": {
      "casual_walk": 1.0,
      "picnic": 0.9,
      "date": 0.8,
      "work_meeting": 0.3,
      "hiking": 0.7,
      "beach_party": 1.0,
      "gym": 0.95,
      "formal_event": 0.2,
      "home_relax": 1.0
    }
  },
  "pants": {
  …
}
…
}
```

- weather_recommendations.json містить температурні діапазони та погодні умови (сонце, дощ, вітер), що використовуються для формування погодних рекомендацій. Кожному типу одягу та сезону приписано коефіцієнт відповідності для конкретної погоди:

```
{
  "winter": {
    "temperature_range": "-21 to 0",
    "weather": {
      "sunny": 0.2,
      "scattered clouds": 0.3,
      "comfortable": 0.2,
      "mild": 0.1,
      "light rain": 0.3,
      "moderate rain": 0.4,
```

```
        "cloudy": 0.6,
        "warm": 0.05,
        "breezy": 0.5,
        "clear sky": 0.3,
        "broken clouds": 0.4,
        "overcast clouds": 0.6,
        "heavy snow": 1.0,
        "snow": 1.0,
        "icy winds": 0.9,
        "frosty mornings": 0.95,
        "mist": 0.7,
        "drizzle": 0.4,
        "cold": 1.0,
        "thunderstorm": 0.3,
        "fog": 0.8
      }
    },
    "spring": {
     …
    },
    "summer": {
      …
    },
    "autumn": {
     …
    },
    "tshirt": {
    …
    },
    "pants": {
  …
     }
 …
}
```

- color_controller.py - модуль для перевірки кольорової сумісності. Підтримує базові правила поєднання кольорів (наприклад, контрастні, аналогічні, нейтральні), а також дозволяє виключити негармонійні комбінації.

- recommendation_strategies.py - основна логіка рекомендацій: цей файл реалізує стратегії підбору одягу на основі погодних умов, подій і кольорів. Тут використовується багатокритеріальна оцінка відповідності, яка дозволяє сформувати набір одягу з найвищим рейтингом відповідності.

Для реалізації визначення погодних умов для одягу на основі категорії та сезону напишемо метод, який буде об’єднувати температурний діапазон та викорситовувати більше значення для погодних умов:

```
def merge_temperature_ranges(range1: str, range2: str) -> str:
            def parse_range(r):
                parts = r.replace(" ", "").split("to")
                return int(parts[0]), int(parts[1])

            min1, max1 = parse_range(range1)
            min2, max2 = parse_range(range2)
            return f"{min(min1, min2)} to {max(max1, max2)}"
```

Для реалізації оцінки того, що елемент одягу підохдить під певні параметри всередині класу ClothingItem було реалізовано необхідні методи:

```
class ClothingItem(Base):
    __tablename__ = "clothing_items"
    ...
    def evaluate_color_match(self, other_color: tuple[int, int, int], palette_type: str):
        from app.recommendation_manager.color_controller import color_match_score
        color_rgb = (self.red, self.green, self.blue)
        color_score = color_match_score(color_rgb, other_color, palette_type)
        return f"Color match score for {self.category.value} with '{other_color}': {color_score}"


    def evaluate_event_match(self, event: str):
        from app.recommendation_manager.recommendation_strategies import get_nested_value
        path = f"{self.category.value}.event.{event}"
```

```
            event_match = get_nested_value("event_recommendations.json",
path)
            return f"Event match score for {self.category.value} for
'{event}': {event_match}"

        def evaluate_weather_match(self, temp:float,weather,
temperature_mismatch: float):
                ...
```

Таким чином можна реалізувати паттерн Strategy за допомогою функцій класу ClothingItem (див. Додаток З).

5ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Модульне тестування

Після завершення ключових етапів проєктування та розробки, тестування стає обов’язковою і надзвичайно важливою частиною життєвого циклу програмного продукту. Його головна мета — ретельна перевірка створеної системи з метою виявлення можливих помилок, логічних недоліків, невідповідностей функціональним вимогам, а також оцінки загального рівня якості та зручності використання.

Основним способом перевірки була обрана перевірка через модульні тести.

Для модульного тестування розробленого серверного застосунку було використано бібліотеку Pytest (джерело [12]) . Перед проведенням тестування необхідно забезпечити використання тестової бази даних:

```
if os.getenv("TESTING") == "1":
    DATABASE_URL = os.getenv("TEST_DATABASE_URL")
else:
    DATABASE_URL = os.getenv("DATABASE_URL")
```

Відповідно перед виконанням тестування потрібно встановити TESTING рівним 1. Для OS Windows:

```
@echo off
call venv\Scripts\activate
set TESTING=1
pytest -n 0 --setup-only
pytest -n 8
Pause
```

Для OS Linux:

```
#!/bin/bash

source venv/bin/activate
```

```
export TESTING=1
pytest -n 0 --setup-only
pytest -n 8
```

Функціональне тестування було зосереджене на перевірці того, чи відповідає робота кожної окремої функції системи вимогам, сформульованим на етапах аналізу та проєктування. Для полегшення цього процесу було створено набір тест-кейсів, у яких описуються конкретні сценарії використання, послідовність дій та очікувані результати (див. Додаток И).

Таким чином можна впевнитися, що всі тест-кейси виконані:

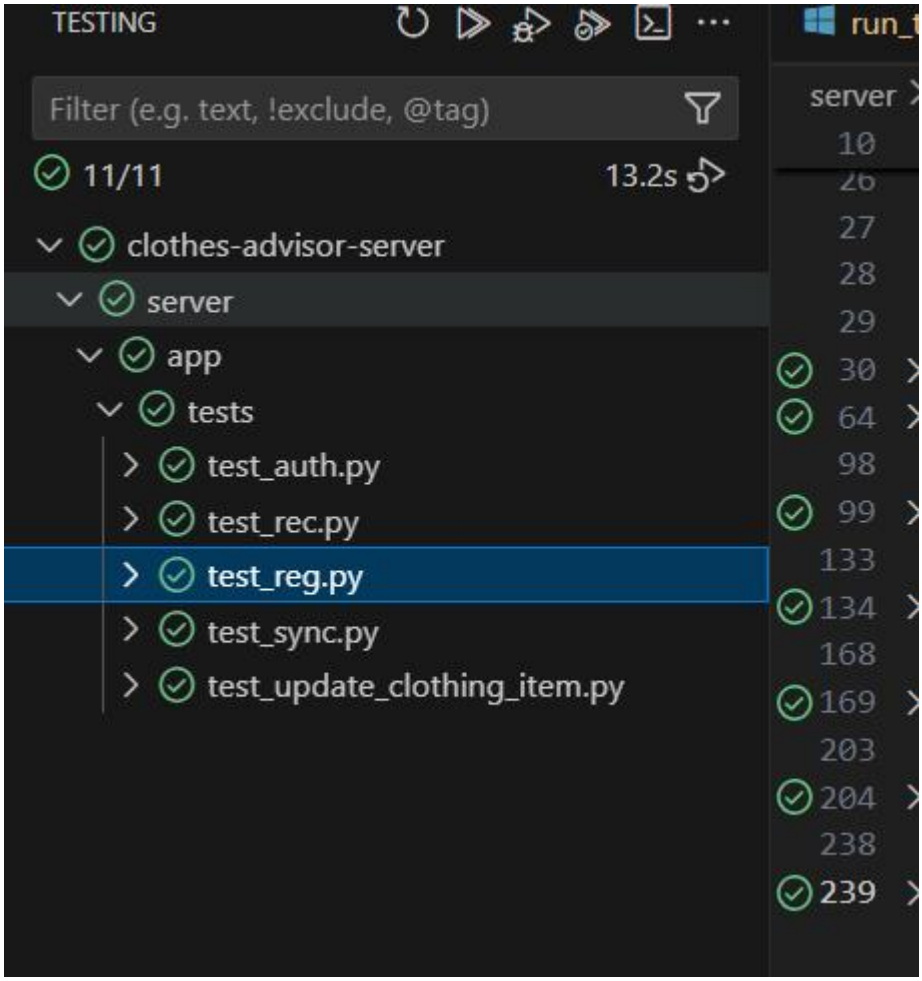


Рисунок 5.1 - Виконання модульних тестів (рисунок виконаний самостійно)

5.2 Тестування продуктивності та масштабованості серверної частини

5.2.1 Вибір інструменту навантажувального тестування

Для виконання навантажувального тестування було обрано Apache JMeter — один із найпопулярніших і потужних інструментів з відкритим кодом для

тестування продуктивності веб-додатків, веб-сервісів та інших серверних компонентів.

Причини вибору Apache JMeter:

- підтримка різних типів протоколів: хоча головним чином JMeter орієнтований на HTTP(S), він також підтримує інші протоколи, зокрема FTP, JDBC, SOAP, REST, MQTT, WebSocket, що робить його універсальним;

- інтуїтивно зрозумілий інтерфейс користувача: графічний інтерфейс JMeter дозволяє створювати сценарії тестування без необхідності програмування, що полегшує налаштування для швидкої перевірки;

- масштабованість: JMeter дозволяє симулювати сотні або тисячі одночасних користувачів завдяки використанню Thread Group і можливості розподіленого тестування (distributed testing);

- гнучкість налаштувань: інструмент підтримує параметризацію запитів, сценарії з логікою (цикли, умови, таймери), обробку сесій (cookies, tokens), що дозволяє створювати наближені до реальності сценарії;

- звітність та аналіз результатів: JMeter генерує докладні звіти з ключовими метриками: час відповіді, кількість помилок, пропускна здатність, латентність тощо. Це дозволяє ефективно аналізувати продуктивність системи.

5.2.2 Performance Profiling

Згідно нефункціональних вимог потрібно перевірити, чи здатна система стабільно обробляти навантаження в обсязі не менше 100 HTTP-запитів на хвилину, що відповідає середньому очікуваному трафіку в умовах реальної експлуатації.

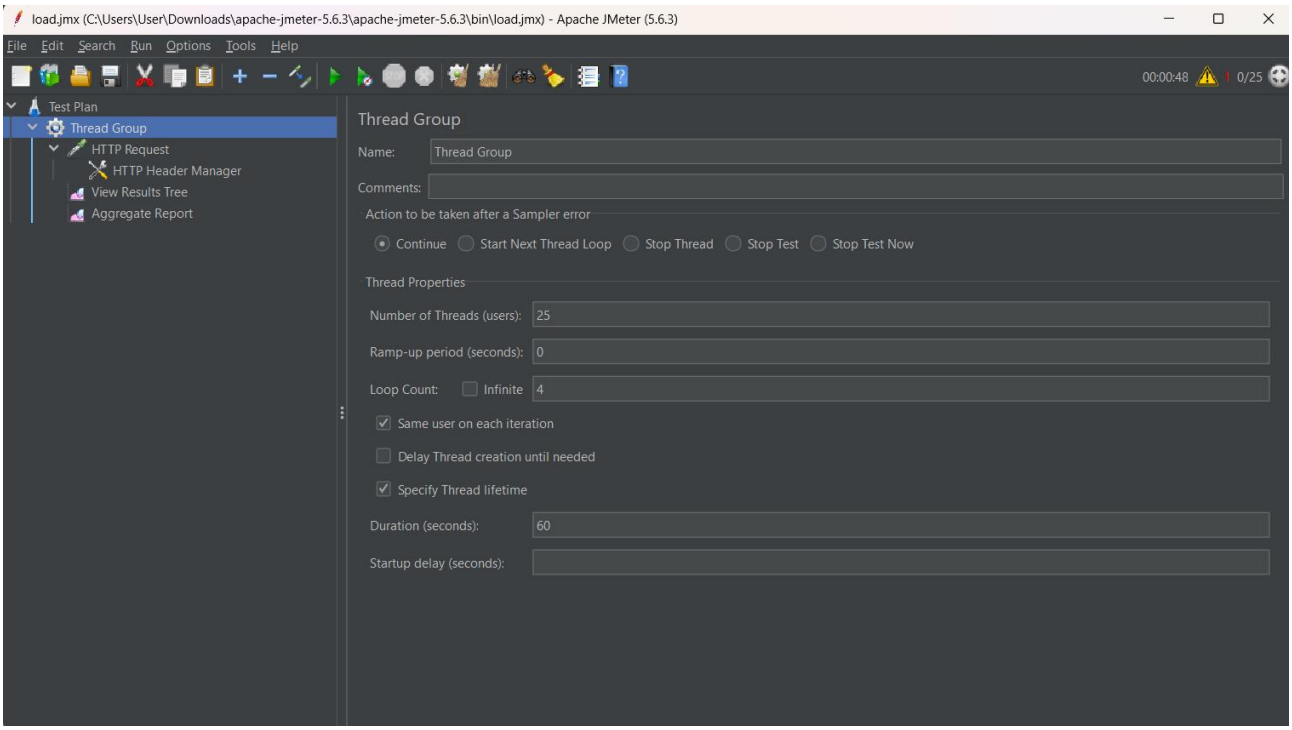


Рисунок 5.2 - Запуск Performance Profiling тесту (рисунок виконаний самостійно)

Після запуску тесту на навантаження, яке має витримувати сервер отримуємо:

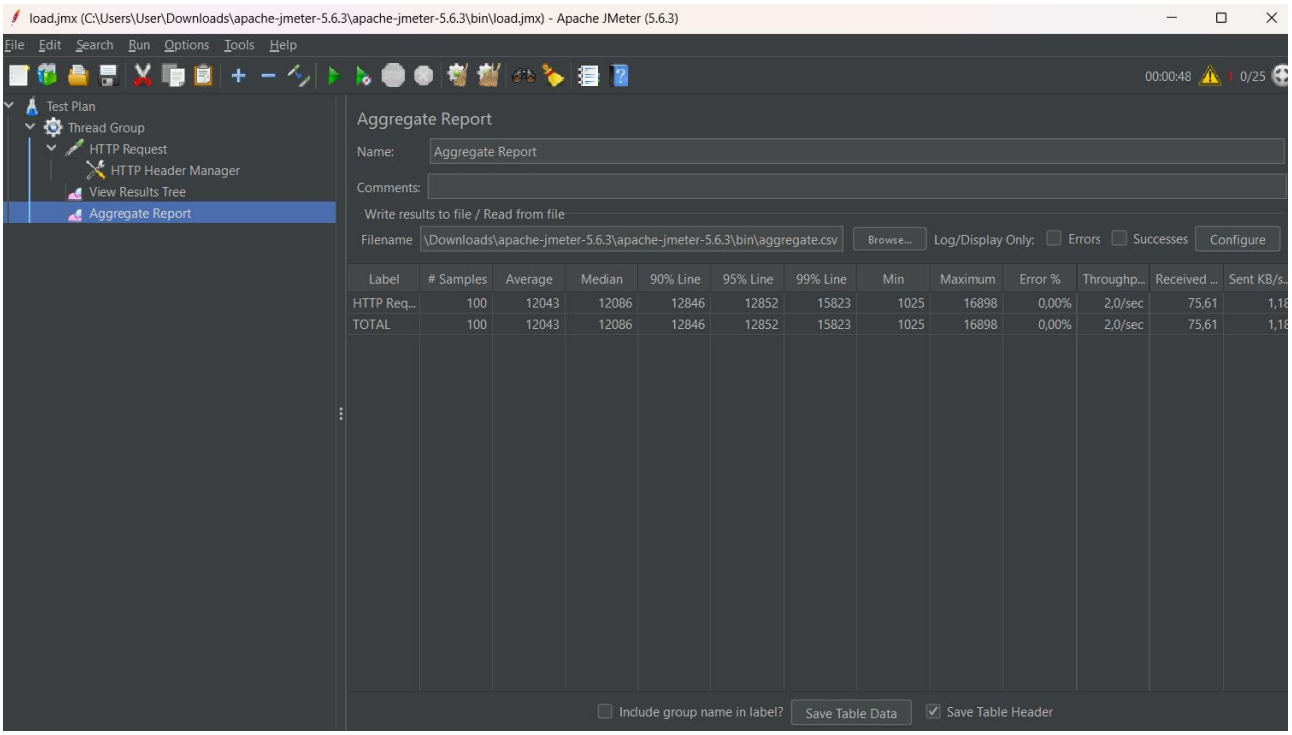


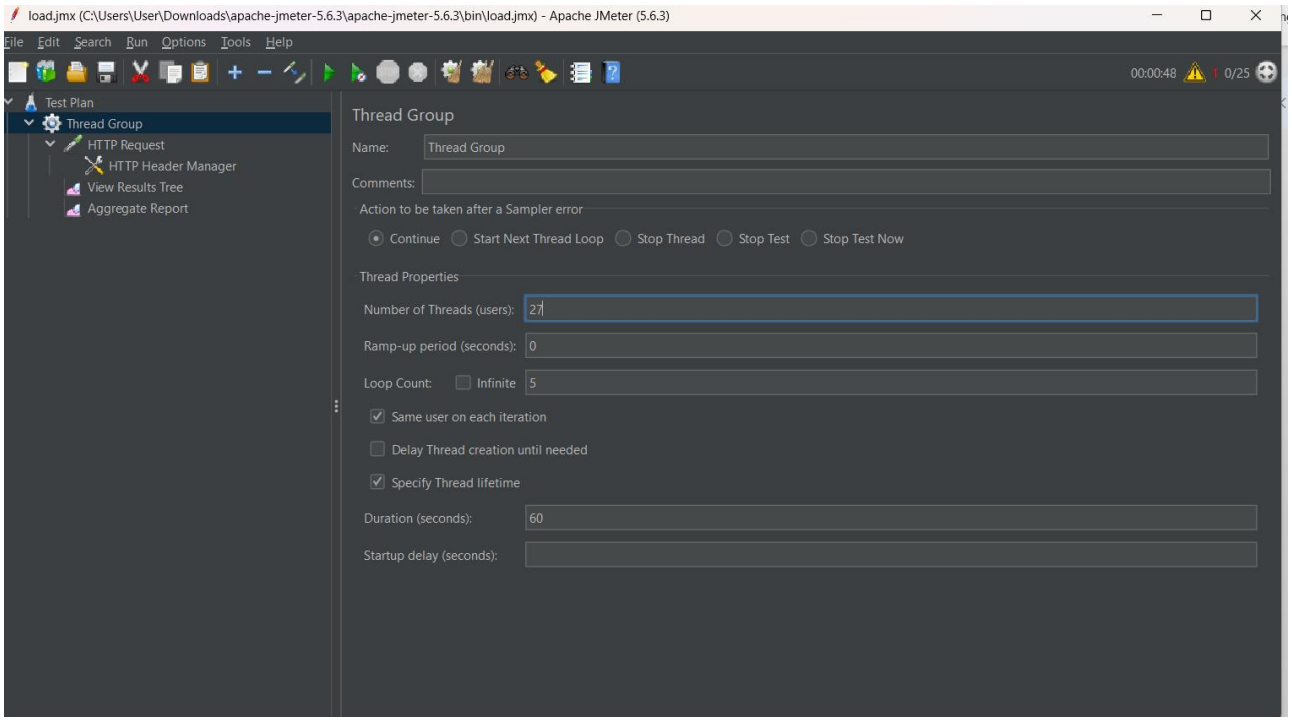
Рисунок 5.3 - Результат Performance Profiling тесту (рисунок виконаний самостійно)

Таблиця 5.1 – Тест-кейс №13 (таблиця виконана самостійно)

Назва тесту:	*Performance Profiling*	
Функція:	**Реєстрація користувача**	
Дія	Очікуваний результат	Результат тесту: - пройдено (P); - провалено (F); - заблоковано(B).
Передумова:		
Запустити сервер	Сервер запущений та приймає запити	Пройдено
Кроки тесту:		
Створити 25 користувачів по 4запити, що відповідає навантаженню системи	Запити усішно пройдені	Пройдено
Постумова:		
Переглянути результати	Має бути 100 успішно виконаних запити	Пройдено
Результати тесту		
Тестувальник: Замковий А.Г.	**Дата прогона теста:** 29.05.2025	**Результат теста (P/F/B): ПРОЙДЕНО (P)**

5.2.3 Load testing

Згідно нефункціональних вимог потрібно перевірити, чи здатна система стабільно обробляти запити при більшому навантаженні.



Після запуску тесту на навантаження, яке більше, ніж те, що має витримувати сервер отримуємо:

Рисунок 5.4 - Запуск Load testing тесту (рисунок виконаний самостійно)

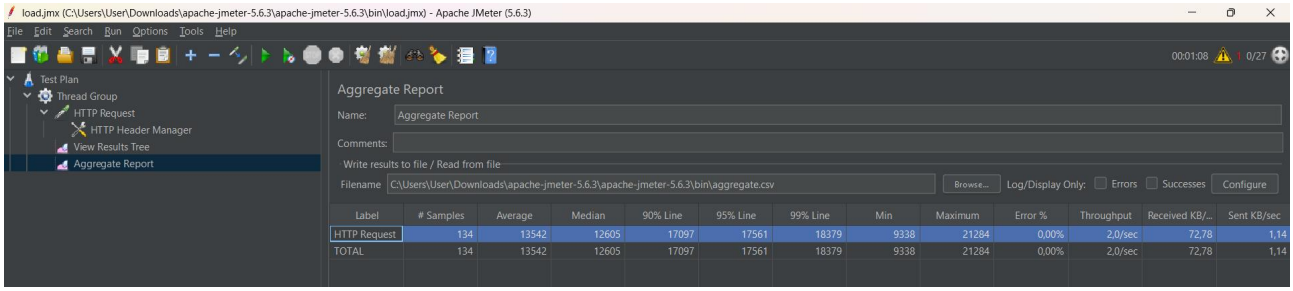


Рисунок 5.5 - Результат Load testing тесту (рисунок виконаний самостійно)

Таблиця 5.2 – Тест-кейс №14 (таблиця виконана самостійно)

Назва тесту:	*Load testing*	
Функція:	**Реєстрація користувача**	
Дія	Очікуваний результат	Результат тесту: - пройдено (P); - провалено (F); -заблоковано(B).
Передумова:		
Запустити сервер	Сервер запущений та приймає запити	Пройдено
Кроки тесту:		

Кінець таблиці 5.2

Створити 27 користувачів по 5 запитів, що є більшим за звичайне навантаження системи	Запити усішно пройдені	Пройдено
Постумова:		
Переглянути результати	Має бути успішно виконані запити	Пройдено
Результати тесту		
Тестувальник: Замковий А.Г.	**Дата прогона теста:** 29.05.2025	**Результат теста (P/F/B): ПРОЙДЕНО (P)**

5.2.4 Stress testing

Необхідно перевірити максимальну навантажувальність серверу.

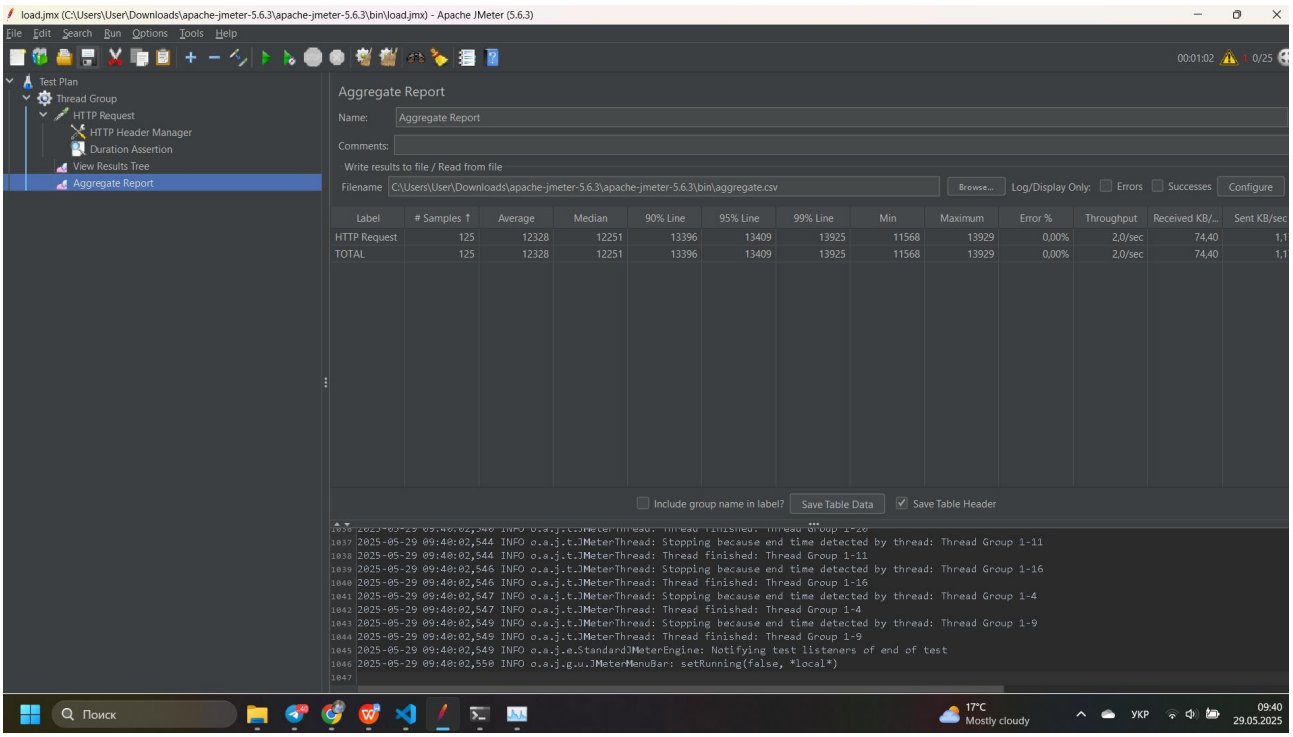


Рисунок 5.6 - Результат Stress Testing тесту (рисунок виконаний самостійно)

Таблиця 5.3 – Тест-кейс №15 (таблиця виконана самостійно)

Назва тесту:	*Stress Testing*	
Функція:	**Контакт-Питання**	
Дія	Очікуваний результат	Результат тесту:
Передумова:		
Запустити сервер	Сервер запущений та приймає запити	Пройдено
Кроки тесту:		
Збільшувати кількість вхідних запитів, щоб визначити максимальну навантажувальність серверу	Перевірити максимально допустиму навантаженість сервера	Пройдено, результат виконання: 125 запитів на хвилину
Постумова:		
Переглянути результати	Має бути записи успішних хапитів	Пройдено
Результати тесту		
Тестувальник: Замковий А.Г.	**Дата прогона теста:** 29.05.2025	**Результат теста (P/F/B): ПРОЙДЕНО (P)**
Примітка:		

ВИСНОВКИ

У результаті виконаної роботи було спроєктовано та реалізовано мобільний додаток керування гардеробом, що дозволяє користувачеві зберігати, організовувати та комбінувати елементи одягу. Основна мета — створення зручного інструменту для ведення цифрового гардероба — була досягнута завдяки впровадженню клієнт-серверної архітектури, яка забезпечує ефективну взаємодію між Android-додатком та серверною частиною.

У процесі розробки було створено серверну частину інформаційної системи керування гардеробом, яка реалізує REST API для взаємодії з клієнтським додатком. Архітектура системи побудована за клієнт-серверною моделлю, що забезпечує масштабованість, розділення відповідальностей та зручність у супроводі й розширенні.

Для реалізації серверної логіки обрано мову програмування Python та фреймворк FastAPI, який забезпечує високу продуктивність, асинхронність та зручне описання API через OpenAPI/Swagger. Було розроблено набір кінцевих точок для роботи з користувачами, елементами одягу, їхніми комбінаціями та синхронізацією даних.

Розроблена серверна частина є універсальною платформою, яку можна легко масштабувати або інтегрувати з іншими сервісами. У майбутньому система може бути доповнена модулями автоматичної класифікації одягу, машинного навчання для персоналізованих рекомендацій, а також хмарним зберіганням для зображень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stylebook Closet App [Електронний ресурс] – URL: https://stylebookapp.com/ (дата звернення: 05.04.2025).

2. ClosetSpace - Style Management [Електронний ресурс] – URL: https://closetspace.ua.aptoide.com/app (дата звернення: 05.04.2025).

3. Smart Closet: Your Personal Stylist [Електронний ресурс] – URL: https://smartcloset.me/ (дата звернення: 05.04.2025).

4. UML Diagram Types | Learn About All 14 Types of UML Diagrams. Creately Blog. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples/ (дата звернення: 02.05.2025).

5. Venv - Creation of virtual environments. Python documentation. URL: https://docs.python.org/3/library/venv.html (дата звернення: 30.04.2025).

6. SQLAlchemy ORM – SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/20/orm/ (дата звернення: 12.04.2025).

7. Welcome to PyJWT – PyJWT 2.10.1 documentation. URL: https://pyjwt.readthedocs.io/en/stable/ (дата звернення: 22.05.2025).

8. What is Color Theory?. The Interaction Design Foundation. URL: https://www.interaction-design.org/literature/topics/color-theory (дата звернення: 17.04.2025).

9. GitHub - danielgatis/rembg: Rembg is a tool to remove images background. GitHub. URL: https://github.com/danielgatis/rembg (дата звернення: 29.04.2025).

10. GeeksforGeeks. Dunder or magic methods in Python. URL: https://www.geeksforgeeks.org/dunder-magic-methods-python/ (дата звернення: 14.04.2025).

11. Strategy. Refactoring and Design Patterns. Fowler M., Beck K. Refactoring. 2018.

12. Pytest documentation. URL: https://docs.pytest.org/en/stable/ (дата звернення: 16.05.2025).

13. GitHub - NureZamkovyiAnatolii/2025_B_PI_PZPI-21-1_Zamkovyi_A_G. URL: https://github.com/NureZamkovyiAnatolii/2025_B_PI_PZPI-21-1_Zamkovyi_A_G (дата звернення: 12.06.2025).