Державний університет інтелектуальних технологій і зв'язку

Факультет Інформаційних технологій та кібербезпеки

Кафедра Інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «**Алгоритми та технології побудови рекомендаційних систем**»

на тему:

«РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ПІДБОРУ МОБІЛЬНИХ ТЕЛЕФОНІВ НА ОСНОВІ ПОВЕДІНКОВИХ УПОДОБАНЬ КОРИСТУВАЧІВ»

Студентки \_Маковєєвої Олександри\_

спеціальності 121 ІПЗ

Іванова П. М.

Керівник: Ід.т.н., проф.

Проф. кафедри ІТ Ісмаілова Н.П.

Одеса 2025

Державний університет інтелектуальних технологій і зв'язку

Факультет Інформаційних технологій та кібербезпеки

Кафедра Інженерії програмного забезпечення

|  |  |
| --- | --- |
| Спеціальність | 121 Інженерія програмного забезпечення |

“**ЗАТВЕРДЖУЮ**”

Зав. кафедри ІПЗ

\_\_\_\_\_\_\_\_\_\_\_ Гаджиєв М.М.

"\_\_\_\_" \_\_\_\_\_\_\_\_\_ 2025 року

ЗАВДАННЯ

На курсовий проект з дисципліни «**Алгоритми та технології побудови рекомендаційних систем**» студента:\_Маковєєвої Олександри Олександрівни

(прізвище, ім’я, по батькові)

1. Тема: \_«Розробка рекомендаційної системи підбору мобільних телефонів на основі поведінкових уподобань користувачів»\_.

2. Термін здачі закінченої роботи: \_\_\_\_\_\_05.12.2025\_\_\_\_\_\_.

3.Вихідні дані: тематична область електронної комерції; характеристики мобільних пристроїв (камера, акумулятор, RAM, екран, ціна); поведінкові переваги користувачів (сценарії використання); мова програмування Python; використання Flask, SQLite і модифікованого алгоритму kNN.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити): аналіз предметної області та існуючих підходів до побудови рекомендаційних систем;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

вибір архітектури та технологій для розробки програмного рішення;\_\_\_\_\_\_\_\_

проектування структури бази даних та алгоритму рекомендацій;\_\_\_\_\_\_\_\_\_\_\_\_

реалізація веб-додатки на Flask з урахуванням користувальницьких переваг;\_\_

розробка інтерфейсу користувача та механізму авторизації;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

тестування системи, аналіз коректності роботи алгоритму і сервера;\_\_\_\_\_\_\_\_\_

формування висновків і пропозицій щодо подальшого розвитку системи.\_\_\_\_

5. Перелік графічного матеріалу: Слайд 1 – Титульна сторінка, \_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 2 – Актуальність теми дослідження,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 3 – Мета і завдання роботи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 4 – Аналіз існуючих підходів,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 5 – Архітектура системи,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 6 – Стек технологій\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 7 – Структура бази даних\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 8 – Алгоритм рекомендацій\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Слайд 9 – Інтерфейс користувача\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

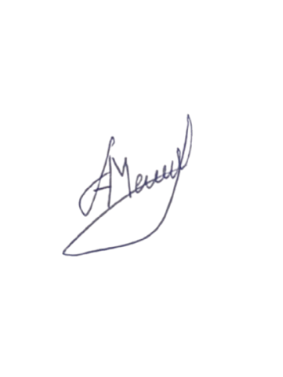
Слайд 10 – Результати та висновки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/пр** | **Назва етапів курсової роботи** | **Термін виконання єтапів роботи** | **Примітки** |
| 1 | Отримання завдання на курсову роботу | 31.10.2025 | ✓ |
| 2 | Аналіз предметної області, дослідження літератури та матеріалів на задану тему | 01.11.2025 – 05.11.2025 | ✓ |
| 3 | Аналіз існуючих аналогів програмного забезпечення | 06.11.2025 – 08.11.2025 | ✓ |
| 4 | Постановка задачі за темою курсової роботи | 09.11.2025 – 10.11.2025 | ✓ |
| 5 | Вибір інструментальних засобів розробки системи | 11.11.2025 – 12.11.2025 | ✓ |
| 6 | Проєктування структури та алгоритму роботи системи, що розробляється | 13.11.2025 – 17.11.2025 | ✓ |
| 7 | Розробка компонентів системи | 18.11.2025 – 24.11.2025 | ✓ |
| 8 | Розгортання та тестування системи, що розробляється | 25.11.2025 – 27.11.2025 | ✓ |
| 9 | Написання розділів пояснювальної записки | 28.11.2025 – 30.11.2025 | ✓ |
| 10 | Представлення завершеної роботи на перевірку | 04.12.2025 | ✓ |
| 11 | Захист курсової роботи | За розкладом кафедри | ✓ |

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наук. ступінь, вчене звання, прізвище та ініціали) (підпис)

****

Завдання прийнято до виконання \_\_Маковєєва О.О.\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали) (підпис)

**АНОТАЦІЯ**

Маковєєва О.О. Розробка рекомендаційної системи підбору мобільних телефонів на основі поведінкових уподобань користувачів. – Курсовий проект за спеціальністю «121 Інженерія програмного забезпечення». – Державний університет інтелектуальних технологій і зв'язку. Факультет Інформаційних технологій та кібербезпеки. Кафедра Інженерії програмного забезпечення. – 33 с. У курсовій роботі розроблено інтелектуальну рекомендаційну систему для підбору мобільних телефонів з урахуванням індивідуальних уподобань користувачів. Досліджено сучасний ринок мобільних пристроїв, проаналізовано наявні методи формування рекомендацій, їх обмеження та вимоги до таких систем в електронній комерції, а також обґрунтовано вибір методів інтелектуального аналізу даних.

В межах проєкту розроблено веб-додаток рекомендаційної системи підбору мобільних пристроїв на основі аналізу користувальницьких переваг і характеристик смартфонів. Для розробки використовувалися мова програмування Python, фреймворк Flask для серверної частини, вбудована база даних SQLite для зберігання каталогу пристроїв і історії взаємодій користувачів, а також власна реалізація алгоритму рекомендацій на основі методу k найближчих сусідів (k-NN).

Алгоритм включає нормалізацію критеріїв, зважування ознак відповідно до вибраних сценаріїв використання смартфона та обчислення відстані між «ідеальним» пристроєм та реальними моделями з каталогу. Додаток надає веб-інтерфейс для вибору уподобань, авторизацію Користувача, Відображення персоналізованих рекомендацій та збереження історії запитів у базі даних.

Пояснювальна записка складається з трьох розділів загальним обсягом 87 сторінок, містить 8 таблиць, 5 рисунків, 1 додаток та 24 використаних джерела.

Ключові слова: рекомендаційні системи, Data Mining, Python, kNN, контентно-орієнтована фільтрація, мобільні телефони.

**ЗМІСТ**

[ВСТУП 1](#_Toc215753004)

[РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ 3](#_Toc215753005)

[1.1. Аналіз ринку мобільних пристроїв та проблеми вибору смартфона користувачем 3](#_Toc215753006)

[1.2. Класифікація та особливості рекомендаційних систем 4](#_Toc215753007)

[1.3. Методи і алгоритми Data Mining, застосовні до задачі підбору 7](#_Toc215753008)

[1.4. Аналіз існуючих рішень та їх обмежень 9](#_Toc215753009)

[1.5. Постановка завдання і визначення вимог до системи 11](#_Toc215753010)

[РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ 14](#_Toc215753011)

[2.1. Вибір інструментальних засобів розробки 14](#_Toc215753012)

[2.2. Аналіз вихідних даних і їх характеристик 16](#_Toc215753013)

[2.3. Попередня обробка, очищення та нормалізація даних 19](#_Toc215753014)

[2.4. Проектування архітектури програмної системи 22](#_Toc215753015)

[2.5. Проектування інтерфейсу користувача 24](#_Toc215753016)

[РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ 27](#_Toc215753017)

[3.1. Загальна архітектура програмної реалізації 27](#_Toc215753018)

[3.2. Реалізація серверної частини на Flask 29](#_Toc215753019)

[3.3. Реалізація бази даних SQLite 31](#_Toc215753020)

[3.4. Реалізація алгоритму рекомендацій 34](#_Toc215753021)

[3.5. Реалізація інтерфейсу користувача та модуля авторизації 37](#_Toc215753022)

[3.6 Тестування рекомендаційної системи 41](#_Toc215753023)

[ВИСНОВКИ 44](#_Toc215753024)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 45](#_Toc215753025)

[Додаток А 48](#_Toc215753026)

[Додоток Б 76](#_Toc215753027)

**ПЕРЕЛІК СКОРОЧЕНЬ УМОВНИХ ПОЗНАК**

5G – п’яте покоління мобільного зв’язку.

Adobe – компанія, що спеціалізується на цифрових продуктах і дослідженнях.

AI / ШІ – штучний інтелект (де застосовано в англоязичному джерелі).

Amazon Personalize – сервіс персоналізованих рекомендацій від Amazon.

BCG (Boston Consulting Group) – Бостонська консалтингова група.

CF (Collaborative Filtering) – колаборативна фільтрація.

CPU (Central Processing Unit) – центральний процесор.

Data Mining – добування даних (інтелектуальний аналіз даних).

Deep Learning – глибинне навчання (Deep Neural Networks).

Deep Neural Networks – глибокі нейронні мережі.

Deloitte – міжнародна аудиторсько-консалтингова компанія.

GPU (Graphics Processing Unit) – графічний процесор.

GSMA (Global System for Mobile Communications Association) – Глобальна асоціація мобільних комунікацій.

GSMArena – веб-сайт для порівняння характеристик мобільних пристроїв, пов’язаний з GSMA.

Google Research – дослідницька група Google, що публікує роботи з персоналізації та аналізу поведінки.

IDC (International Data Corporation) – Міжнародна аналітична компанія.

IPS (In-Plane Switching) – тип матриці дисплея.

Item-Based CF – колаборативна фільтрація на основі об’єктів/товарів.

kNN (k-Nearest Neighbors) – метод найближчих сусідів.

McKinsey – міжнародна консалтингова компанія.

MIT (Massachusetts Institute of Technology) – Массачусетський технологічний інститут.

MP (Megapixels) – мегапікселі (для камер).

Netflix – компанія/сервіс, що використовується як джерело прикладів гібридних рекомендацій.

OS (Operating System) – операційна система.

PCI (Peripheral Component Interconnect) – інтерфейс підключення периферії (якщо зустрічалося у джерелах, можна додати).

ppi (pixels per inch) – щільність пікселів на дюйм.

Python – мова програмування.

RAM (Random Access Memory) – оперативна пам’ять.

User-Based CF – колаборативна фільтрація на основі користувачів.

USD (United States Dollar) – долар США.

# ВСТУП

Кожного дня людство створює величезний обсяг нової інформації: близько 90% усіх даних було згенеровано лише за 2020–2022 роки. Технологічний прогрес значно розширив фізичні можливості людини і підвищив її мобільність у часі та просторі. Щохвилини у світі відбувається 154 200 дзвінків у Skype, надсилається 16 мільйонів текстових повідомлень і 103 447 520 спам-листів електронною поштою [1][4].

На фоні стрімкого зростання даних цифрова економіка переживає глибоку трансформацію: за останні десять років бізнес-середовище стало більш взаємопов’язаним, а Інтернет перетворився на ключову платформу для взаємодії з споживачами. Компанії стикаються з труднощами в обробці величезних масивів даних і пошуку інструментів, які дозволили б ефективно використовувати їх для підвищення якості клієнтського досвіду. Дослідження McKinsey (2020) показують, що персоналізація є одним із ключових факторів зростання конверсії в онлайн-торгівлі, збільшуючи показники взаємодії з користувачами до 20–30% [2]. Крім того, за даними Adobe Digital Trends Report (2021), 61% організацій вважають персоналізовані рекомендації необхідним компонентом сучасної цифрової стратегії [3].

В цих умовах однією з найбільш актуальних задач електронної комерції стає інтелектуальний підбір товарів, максимально відповідних індивідуальним уподобанням користувачів. Рекомендаційні системи дозволяють автоматизувати аналіз поведінки клієнтів, виявляти приховані залежності і на основі цього формувати точні персональні рекомендації. Особливу важливість такі технології набувають на насичених ринках, де кількість представлених моделей вимірюється сотнями, а критерії вибору включають десятки технічних і користувацьких характеристик. Ринок мобільних телефонів якраз належить до таких: висока конкуренція, швидке оновлення модельного ряду та різноманіття користувацьких сценаріїв створюють додаткові складнощі при виборі пристрою.

У цьому контексті розробка рекомендаційної системи підбору мобільних телефонів на основі поведінкових уподобань користувачів є значущим та практично орієнтованим напрямом дослідження.

Об’єкт дослідження – процес формування персональних рекомендацій у системах електронної комерції.

Предмет дослідження – методи та алгоритми інтелектуального аналізу даних, застосовувані для прогнозування користувацьких уподобань при виборі мобільних пристроїв.

Метою даної роботи є створення програмного застосунку, здатного аналізувати поведінку користувачів та характеристики смартфонів, а потім формувати релевантні рекомендації, що відповідають індивідуальним потребам людини. Для досягнення мети застосовуються методи Data Mining, машинного навчання, контентно-орієнтованої фільтрації та алгоритмів пошуку найближчих сусідів, а також методи статистичного аналізу та нормалізації даних.

Наукова новизна роботи полягає в адаптації сучасних алгоритмів рекомендаційних систем до задачі підбору мобільних телефонів із урахуванням багатокритеріальних параметрів та користувацьких поведінкових даних. На відміну від стандартних фільтрів, застосовуваних в інтернет-магазинах, розроблена модель поєднує об’єктивні характеристики пристроїв і індивідуальний профіль користувача.

Практичне значення результатів полягає в можливості інтеграції розробленої рекомендаційної системи у веб-платформи та електронні каталоги мобільних пристроїв, що дозволить підвищити задоволеність користувачів, скоротити час пошуку відповідного смартфона, збільшити конверсію продажів. Розроблена система може бути використана як у комерційних сервісах, так і в навчальних, дослідницьких та демонстраційних цілях.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

## 1.1. Аналіз ринку мобільних пристроїв та проблеми вибору смартфона користувачем

Ринок мобільних пристроїв є одним з найбільш динамічних і висококонкурентних сегментів глобальної електронної індустрії. За останні роки він продовжує демонструвати стійке зростання, що супроводжується швидким оновленням модельного ряду, ускладненням технічних характеристик і диверсифікацією користувальницьких сценаріїв. За даними міжнародної аналітичної компанії IDC, в 2022-2023 роках щорічно поставлялося понад 1,2 млрд смартфонів, незважаючи на тимчасові коливання, викликані глобальними економічними факторами [4]. Додатково організація GSMA зазначає, що до 2025 року кількість унікальних мобільних користувачів досягне 5,9 млрд, що становить близько 70% світового населення, що продовжує посилювати тиск на виробників в частині розширення продуктових лінійок [5].

Зростання асортименту призводить до ускладнення процесу вибору смартфона для кінцевого користувача. Виробники пропонують десятки моделей щорічно, кожна з яких відрізняється десятками характеристик: об'ємом оперативної і вбудованої пам'яті, якістю камери, енергоефективністю, продуктивністю, частотою оновлення екрану, підтримкою мереж 5G, наявністю ШІ-функцій і багатьма іншими параметрами. Дослідження Deloitte Global Consumer Survey підкреслює, що понад 47% покупців зазнають труднощів при виборі смартфона саме через надмірну кількість варіантів і складність їх об'єктивного порівняння [6]. Це ускладнює процес прийняття рішення і підвищує ймовірність вибору пристрою, який не відповідає реальним потребам Користувача.

Додатковим фактором, що впливає на складності вибору, є особливості поведінки користувачів. Згідно з дослідженням Google "Consumer Insights in Mobile Shopping", більше 60% користувачів вивчають технічні характеристики смартфона не менше ніж на трьох різних джерелах перед покупкою, включаючи огляди, соціальні мережі, порівняння пристроїв і рекомендаційні платформи [7]. Такий підхід формує складний і багатопараметричний шлях прийняття рішення, в якому кожен користувач враховує не тільки технічні параметри, але і власні поведінкові переваги: досвід використання попередніх пристроїв, найбільш важливі функції (камера, автономність, ігри), бюджет, брендові переваги, а також враження від відгуків інших користувачів.

Проблемою стає і суб'єктивність оцінки характеристик. Наприклад, один користувач може вважати критично важливим наявність високопродуктивного процесора для ігор, тоді як іншому важливіше автономність і якість камери. Дослідження Boston Consulting Group підтверджують, що споживачі орієнтуються не стільки на об'єктивні параметри, скільки на відповідність пристрою їх індивідуальним сценаріям використання, що робить універсальні фільтри в інтернет-магазинах малоефективними [8].

Підсумовуючи, сучасний ринок мобільних пристроїв створює об'єктивні складності у виборі оптимальної моделі, оскільки поєднує величезний асортимент, багатовимірність технічних характеристик і широкий спектр призначених для користувача очікувань. Все це формує потребу в інтелектуальних інструментах, здатних враховувати поведінкові переваги і автоматично підбирати найбільш релевантні варіанти. У цих умовах рекомендаційні системи стають ключовим механізмом персоналізації вибору і підвищують якість взаємодії користувача з цифровими платформами.

## 1.2. Класифікація та особливості рекомендаційних систем

Система рекомендацій – це тип програмного забезпечення, яке збирає інформацію про ваші вподобання та поведінку, щоб надавати вам персоналізовані пропозиції. Ці системи використовують алгоритми для навчання з різних джерел даних та надають вам припущення, що це може бути найбільш актуальним для вас [9]. З огляду на наведене, не дивує факт, що вони давно стали ключовим елементом сучасних цифрових платформ, що дозволяє персоналізувати взаємодію з користувачами і підвищувати ефективність електронних сервісів.

Згідно з дослідженням MIT Technology Review, понад 35% споживчих покупок на великих онлайн-платформах ініціюються рекомендаційними системами, що підкреслює їх стратегічну роль у сучасній економіці даних [10].

Існує кілька основних типів рекомендаційних систем: контентно-орієнтовані, колаборативні та гібридні. Кожна з них використовує власний підхід до аналізу даних і формування пропозицій, а вибір методу залежить від характеру завдання, доступних даних і вимог до якості прогнозування.

Контентно-орієнтовані рекомендаційні системи (Content-Based Filtering) грунтуються на аналізі властивостей об'єктів і переваг конкретного користувача. Їх основна ідея полягає в тому, що якщо користувачеві сподобався об'єкт з певними характеристиками, то йому будуть цікаві і інші об'єкти зі схожими властивостями. Такі системи широко застосовуються в завданнях, де у товарів є формалізовані атрибути: технічні характеристики смартфонів, жанри фільмів, параметри товарів і т. п. Дослідження Ricci, Rokach та Shapira зазначає, що алгоритми, орієнтовані на вміст, є одними з найбільш стійких до проблеми «холодного старту» для нових користувачів, оскільки вони можуть формувати рекомендації на основі обмеженого профілю [11]. Для ринку мобільних телефонів цей підхід особливо зручний, оскільки кожен смартфон має структурований набір характеристик – обсяг пам'яті, тип процесора, камера, акумулятор, ціна – що дозволяє порівнювати пристрої в багатовимірному просторі ознак.

Колаборативні рекомендаційні системи (Collaborative Filtering), навпаки, спираються на аналіз поведінки безлічі користувачів. В їх основі лежить припущення, що якщо два користувача демонструють схожі переваги, то рекомендації, які підходять одному з них, будуть релевантними і для іншого. Найбільш відомими методами є User-Based CF і Item-Based CF, а також матрична факторизація, активно застосовується в сучасних рекомендаційних платформах. Компанія Netflix у звіті 2020 року підкреслює, що саме методи колаборативної фільтрації забезпечують значну частину персоналізованих рекомендацій, що формуються їх системою [12]. Однак подібні алгоритми сильно залежать від якості даних і схильні до проблеми розрідженості матриці переваг, що робить їх менш ефективними в сферах, де призначені для користувача оцінки збираються рідко або нерегулярно.

Гібридні рекомендаційні системи об'єднують сильні сторони обох підходів і компенсують їх недоліки. Згідно з дослідженням IEEE Access, гібридні моделі демонструють більш високу точність рекомендацій завдяки використанню відразу декількох джерел інформації: характеристик об'єктів, поведінки користувачів, логів взаємодії та контексту [13]. Такий підхід дозволяє значно поліпшити якість рекомендацій, особливо в умовах великого асортименту і високої варіативності призначених для користувача переваг. У разі вибору мобільних телефонів гібридні системи можуть об'єднувати дані про характеристики смартфонів (контентний метод) і статистику поведінки користувачів (колаборативний метод), забезпечуючи більш точну відповідність рекомендацій індивідуальним потребам.

Зважаюче на усе перераховане, то контентно-орієнтовані, колаборативні та гібридні рекомендаційні системи пропонують різні підходи до формування персональних рекомендацій: перші спираються на властивості об'єктів, другі – на схожість поведінки користувачів, а треті об'єднують обидва види даних для підвищення точності. З огляду на специфіку завдання підбору смартфонів, де важливі як об'єктивні характеристики пристроїв, так і індивідуальні поведінкові переваги, в рамках даної роботи в якості основного методу побудови рекомендацій буде доцільним використовувати контентно-орієнтований підхід, об'єднаний з алгоритмом k-Nearest Neighbors (kNN), який застосовується для оцінки ступеня близькості характеристик конкретного пристрою до «ідеального» профілю, що формується на основі переваг користувача. Такий підхід дозволяє враховувати багатокритеріальний набір параметрів смартфона (камера, батарея, Оперативна пам'ять, діагональ екрану, вартість) і зіставляти їх з індивідуальними сценаріями використання пристрою.

## 1.3. Методи і алгоритми Data Mining, застосовні до задачі підбору

Завдання інтелектуального підбору смартфонів вимагає застосування методів Data Mining, здатних обробляти багатовимірні дані, враховувати поведінкові переваги користувачів і виявляти приховані. Люди занадто різні, а їх переваги – ще більш різноманітні. Тому сучасні рекомендаційні системи зазвичай комбінують кілька підходів, і кожен з них по-своєму корисний. Дослідження показують, що саме комбінація алгоритмів класифікації, кластеризації та фільтрування вмісту є найбільш ефективною при аналізі складних наборів даних споживачів [14].

Одним з найбільш інтуїтивних методів є алгоритм найближчих сусідів kNN. Його особливість полягає в тому, що він не будує явної навчальної моделі, а визначає рішення на основі подібності нових об'єктів з уже відомими даними. Незважаючи на свою простоту, kNN все ще залишається надійним базовим інструментом для пошуку подібних об'єктів у багатовимірних даних. Наприклад, дослідження «a Comparative Evaluation of Neighbor-Based Collaborative Filtering» показує, що kNN чудово справляється з ситуаціями, коли Користувач не має довгої історії взаємодій – достатньо кількох уподобань, і алгоритм вже може запропонувати щось подібне[15]. Для смартфонів це особливо актуально: Користувач рідко купує десятки пристроїв, і тому «класична» колаборативна фільтрація працює далеко не ідеально. А ось пошук найближчих сусідів по перевагам-цілком життєздатний варіант.

Але kNN хороший, поки простір ознак не розростається до розмірів Всесвіту. Смартфони ж-істоти багатовимірні: камера, пам'ять, процесор, екран, акумулятор, ціна, операційна система, підтримка 5g, частота оновлення, якість нічної зйомки – і все це одночасно. Тому на допомогу зазвичай приходить кластеризація. Алгоритми на кшталт k-means дозволяють розбити сотні моделей смартфонів на компактні групи, де пристрої дійсно схожі один на одного. Причому дослідження останніх років підтверджують ефективність такого підходу. Наприклад, в роботі " Clustering-Based Recommendations for High-Dimensional Data» підкреслюється, що попередня кластеризація дозволяє підвищити точність рекомендацій, особливо коли особливої ролі відіграє комбінація безлічі характеристик [16]. На практиці це означає, що користувач, якому важлива камера і автономність, з більшою ймовірністю опиниться в кластері «мобільних фотографів», ніж в кластері «геймерів» – і рекомендації відразу стають змістовнішими. Крім того, кластеризація допомагає виявляти приховані сегменти користувачів на основі їх поведінки та уподобань, що особливо важливо для персоналізованого підбору пристроїв.

Однак технічні характеристики – це лише половина історії. Люди вибирають смартфони не тільки по цифрам. Поведінка користувача, його сценарії використання, звички – все це формує куди більш цікаву картину. Саме тут вступає в гру контентно-орієнтована фільтрація. У класичному вигляді вона пропонує об'єкти, схожі на ті, що сподобалися користувачеві по набору ознак. Але в сучасних реалізаціях вона стає куди розумнішою. В огляді «Deep Neural Networks for Content-Based Recommender Systems» описано, як глибинні моделі дозволяють враховувати структурні особливості об'єктів, їх текстові описи, метадані і навіть стилістику користувацьких оглядів [17]. Для вибору смартфона такі методи особливо корисні: вони допомагають системі «читати між рядків» і розуміти, який тип пристрою людина віддає перевагу, навіть якщо він сам не формулює це явно.

Якщо ж спробувати об'єднати все це – переваги, характеристики, поведінка-стає очевидно, що окремі алгоритми дають лише шматочок рішення. На практиці гібридні моделі працюють найкраще. Вони дозволяють з'єднати контентний аналіз з поведінкою користувачів і зменшити недоліки кожного підходу окремо. У дослідженні Netflix про якість гібридних рекомендацій прямо говориться, що комбінація ознак і поведінкових даних робить модель набагато стійкіше до «холодного старту» і покращує релевантність видачі [18].

І ось що цікаво: навіть просте об'єднання цих методів – наприклад, кластеризація для угруповання моделей, kNN для пошуку найближчих варіантів всередині кластера і контентна фільтрація для обліку індивідуальних особливостей – створює вже цілком зрілу рекомендаційну систему. Це той випадок, коли кожен алгоритм вирішує свою частину завдання, а разом вони дають відчуття, що система дійсно «розуміє», що потрібно користувачеві.

Звичайно, можна жартувати, що смартфони підбираються «як друзі» – за спільними інтересами і подібністю характерів, – але саме так сучасні методи Data Mining дозволяють наблизитися до персоналізованого підбору пристрою. І якщо врахувати, наскільки перевантажений ринок і наскільки різні потреби у покупців, стає ясно: без цих алгоритмів Користувач тоне в море характеристик. З алгоритмами ж-починає, нарешті, рухатися по карті.

## 1.4. Аналіз існуючих рішень та їх обмежень

Коли я почала розбиратися, які рекомендаційні механізми вже використовуються на ринку підбору мобільних пристроїв, перше, що кинулося в очі – майже всі великі інтернет-магазини дійсно намагаються вирішити проблему вибору, але роблять це, по суті, досить прямолінійно. Наприклад, Amazon застосовує рекомендації на основі історії переглядів і покупок, використовуючи класичні моделі колаборативної фільтрації і контент-орієнтованого аналізу. Про це прямо йдеться в документації Amazon Personalize, де підкреслюється, що система будує рекомендації переважно на поведінкових патернах користувачів, а не на глибокому розумінні складних технічних характеристик товарів [19]. Це добре працює, коли мова йде про книги або аксесуари, але в світі смартфонів, де десятки параметрів є критичними для вибору, проста кореляція «подібні покупки» часто дає дивні або надто узагальнені результати.

Інший поширений варіант – сервіси порівняння пристроїв на зразок [GSMArena](https://www.gsmarena.com), які надають величезну кількість фактичної інформації, специфікацій та оцінок користувачів. На них можна провести кілька годин, відкриваючи по десять вкладок, дивлячись на таблиці з процесорами, камерами і акумуляторами, і все одно піти з почуттям легкого когнітивного виснаження. Сервіси цього типу прекрасно справляються з агрегуванням даних, але не намагаються зрозуміти, які параметри дійсно важливі для конкретного користувача. Це все ще «ручний вибір», лише з більш зручними таблицями.

Є й дослідні прототипи. У статтях останніх років досить робіт, що пропонують моделі рекомендацій для мобільних пристроїв, в тому числі заснованих на машинному навчанні і глибинних уявленнях даних. Наприклад, в огляді по сучасним рекомендаційним системам «Deep Learning based Recommender System: a Survey» [20] підкреслюється, що більшість промислових додатків все ще обмежені необхідністю великих історичних даних і не здатні враховувати багатий контекст предметної області. Тобто алгоритм може прекрасно помічати, що користувачеві подобаються моделі бренду Samsung, але він абсолютно не розуміє, що цей користувач – мобільний фотограф і йому в десять разів важливіше Світлосила камери і якість нічної зйомки, ніж продуктивність процесора.

Навіть у більш прикладних дослідженнях, наприклад, у роботі «Context-Aware Recommendations for Mobile Device Selection» автори підкреслюють, що більшість існуючих систем не враховують ситуативні фактори: реальний сценарій використання, мобільні звички, вимоги до батареї або навіть частоту зйомки фото і відео. Такі речі формують поведінковий профіль, а поточні рішення сайти-магазинів або ігнорують його, або зводять до двох-трьох поверхневим ознаками [21].

У підсумку виходить цікавий парадокс: інформаційні ресурси стали феноменально багатими, але самі рекомендації – ні. Інтернет-магазини занадто зав'язані на історію покупок, а сервіси порівняння – на сухі таблиці характеристик. Користувач же виявляється десь посередині і намагається поєднати незводиме: особисті очікування, різні типи інформації і занадто великий асортимент. Саме тому ідея впровадити систему, яка враховує поведінкові уподобання конкретного користувача, починає виглядати не просто як академічна вправа, а як реальний спосіб знизити інформаційну перевантаження і наблизити Вибір смартфона до чогось, що нагадує розумний діалог, а не марафон гортання характеристик.

## 1.5. Постановка завдання і визначення вимог до системи

Швидко стало ясно, що сформулювати завдання для власної системи рекомендацій неможливо «в двох словах», тому що вибір смартфона – це вже більше, ніж просто порівняння мегапікселів і гігабайт. Це майже невелике дослідження людини, її звичок, оточення, завдань і навіть циркадних ритмів. З цієї причини при постановці завдань в цій роботі слід враховувати не тільки технічну сторону, але і логіку того, як користувачі приймають рішення в умовах інформаційного перевантаження.

Об'єктом дослідження в моїй роботі виступає процес формування персональних рекомендацій в системах електронної комерції. Це те саме «Чорне дзеркало», в яке кожен користувач дивиться, сподіваючись побачити саме той пристрій, який буде для нього комфортним, зручним і відповідним звичкам. При цьому предметом дослідження є більш вузький аспект – методи інтелектуального аналізу даних, що дозволяють врахувати поведінкові переваги користувачів при виборі мобільних телефонів. Такий фокус виправданий: сучасні рекомендації вже давно вимагають від розробників розуміння не просто продукту, а контексту поведінки людини, що підкреслюють численні дослідження з персоналізації, наприклад робота від Google Research про вплив поведінкових патернів на ефективність рекомендацій [22].

Мета моєї роботи – розробити програмний додаток, який може аналізувати характеристики смартфонів і зіставляти їх з індивідуальними уподобаннями користувача, враховуючи не тільки базові фільтри, але і реальні патерни поведінки: чи любить людина мобільну фотографію, грає він у важкі ігри, як часто використовує навігацію, наскільки важлива автономність і чи варто йому переплачувати за флагманський процесор. З точки зору завдань – тут все одночасно просто і складно. Необхідно провести повний цикл: дослідити предметну область, проаналізувати існуючі рішення, вибрати алгоритми та інструментальні засоби, побудувати архітектуру системи, реалізувати алгоритми рекомендації, протестувати їх і оцінити, наскільки адекватно система "розуміє" користувача.

Вимоги до системи народжуються природним чином з цих завдань. Функціональні вимоги включають можливість завантаження і обробки даних про смартфони, побудова профілю користувача, застосування алгоритмів рекомендацій і подання підсумкового результату в зрозумілій формі. Крім того, Користувачеві має бути легко взаємодіяти з системою – чим менше когнітивного шуму, тим більша ймовірність того, що людина дійсно сприйме рекомендації серйозно.

Технічні вимоги виглядають не менш важливими. Система повинна працювати досить швидко, щоб користувач не встигав пошкодувати про свій вибір сайту; алгоритми повинні коректно обробляти відсутні або суперечливі дані; а структура даних – бути розширюваною, адже ринок смартфонів оновлюється швидше, ніж деякі люди змінюють аватарки.

Крім описанного, система повинна враховувати обмеження реального світу: дані бувають неповними, призначені для користувача переваги – не завжди раціональними, а моделі – вимагають періодичного перегляду. Тому вимоги до гнучкості архітектури стають обов'язковими: важливо, щоб алгоритм можна було замінити або оновити без повного переписування програми.

В результаті завдання в цій роботі виникає як бажання створити систему, яка може не тільки «вибрати правильний смартфон», але і фактично витягти сигнали з поведінки користувача, які роблять рекомендацію персоналізованої. І якщо система зможе запропонувати людині пристрій, який логічно вписується в його спосіб життя, а не просто вписується в таблицю характеристик, то завдання буде виконано правильно, і зусилля по розробці такої розумної моделі будуть повністю виправдані.

# РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

## 2.1. Вибір інструментальних засобів розробки

При проектуванні рекомендаційної системи одним з перших і принципово важливих рішень став вибір технологічного стека. Від коректно підібраних інструментів залежала не тільки швидкість розробки, а й архітектурна гнучкість, масштабованість, а також можливість швидко переходити від експериментальної моделі до робочого веб-додатку. Тому аналіз існуючих платформ і бібліотек проводився з орієнтацією на практичну реалізацію і подальший розвиток проекту.

Основною мовою розробки був обраний Python. Він давно зарекомендував себе як стандарт де-факто в областях аналізу даних, прототипування моделей, побудови алгоритмів рекомендацій і розробки веб-сервісів, і це не просто гарна фраза: згідно з дослідженням IEEE Spectrum, Python вже кілька років впевнено посідає перше місце серед мов для аналізу даних та машинного навчання, випереджаючи як Java, так і C# [23]. Python володіє виразним синтаксисом, багатою екосистемою бібліотек і дозволяє без додаткових шарів Абстракції реалізувати як обробку даних, так і алгоритмічну логіку. Для даної роботи важливою перевагою мови стало те, що алгоритм рекомендацій (модифікований k-Nearest Neighbors) можна ефективно реалізувати засобами чистого Python без залучення важких ml-фреймворків, що робить систему більш прозорою, інтерпретованою і легко розширюваною.

Для серверної частини була обрана Flask-мінімалістична і гнучка веб-фреймворк-платформа, ідеально підходить для проектів з логікою обробки даних всередині Python-коду. На відміну від Django, орієнтованого на великі серверні програми, Flask надає розробнику весь необхідний набір інструментів, залишаючись при цьому легким, модульним і повністю контрольованим. Завдяки цьому вдалося без складної інфраструктури побудувати веб-інтерфейс, інтегрований з алгоритмом рекомендацій і базою даних.

Для зберігання даних був використаний SQLite, вбудований механізм зберігання, доступний через стандартний модуль sqlite3 в Python. Його вибір пояснюється практичністю: для навчального і дослідницького проекту не потрібно розгортати окремий сервер СУБД, а єдиний файл бази даних забезпечує достатню гнучкість, підтримує SQL-запити і легко переноситься між пристроями. У базі зберігаються як характеристики смартфонів, так і історія взаємодій користувачів, що дозволяє формувати персоналізовані добірки і аналізувати поведінкові дані.

Алгоритмічне ядро рекомендаційної системи реалізовано як самостійний модуль на Python. Модель поєднує в собі контентно-орієнтований підхід і модифіковану евклідову метрику kNN, яка обчислює відстань між параметрами смартфонів і «ідеальним» профілем користувача. Таке рішення дозволяє гнучко враховувати користувальницькі переваги, що задаються через інтерфейс, і одночасно зберігати інтерпретованість роботи алгоритму.

Фронтенд-частина системи заснована на стандартних веб-технологіях: HTML, CSS і мінімальному обсязі JavaScript. Відмова від важких клієнтських фреймворків на кшталт React або Vue був свідомим, оскільки завдання проекту полягає не в розробці високонавантаженого веб-клієнта, а в створенні прозорого інтерфейсу, який демонструє роботу рекомендаційного алгоритму і забезпечує зручність взаємодії користувача з системою.

Для наочності обраний стек інструментів зведений в таблицю 2.1.

*Таблиця 2.1.*

**Вибрані інструменти розробки та їх роль у системі**

|  |  |  |
| --- | --- | --- |
| **Інструмент** | **Призначення в системі** | **Причина вибору** |
| Python | Основна мова розробки | Простота, гнучкість, широка екосистема; ідеальний для роботи з даними та алгоритмами |
| Flask | Серверна частина і маршрутизація | Легкість, модульність, швидка інтеграція з алгоритмами Python |
| SQLite | Зберігання даних пристроїв та історії взаємодій | Не потребує окремого сервера, проста у використанні, достатня для навчального проєкту |
| Чиста реалізація kNN | Алгоритм рекомендацій | Прозорість, інтерпретованість, відсутність зайвих залежностей |
| HTML / CSS / JS | Формування користувацького інтерфейсу | Легкий стек, швидка розробка, зручний доступ до функціональності |

Таким чином, обраний стек технологій утворює логічну, збалансовану архітектуру. Python забезпечує реалізацію алгоритмів, Flask-механізм взаємодії з користувачем, SQLite-надійне зберігання даних, а HTML/CSS/JS — доступний і інтуїтивний інтерфейс. Таке поєднання дозволяє не тільки створити працездатну рекомендаційну систему, але і зробити її зручною, розширюваною і готовою до подальшої модернізації.

## 2.2. Аналіз вихідних даних і їх характеристик

Переглядаючи оригінальний датасет зі смартфонами, я відчула: це типовий реальний набір даних, де все змішано, ніби його збирали різні люди у різний час, у різному настрої та на різних пристроях. Саме такі дані зустрічає розробник рекомендаційних систем – сирі, неоднорідні, іноді суперечливі. Ці «шорсткості» потрібно перетворити на акуратну, формалізовану структуру, придатну для алгоритмів машинного навчання.

Для побудови рекомендаційної моделі я орієнтувалася на відкриті специфікації смартфонів, подібні до тих, які публікує GSMArena, а також на структуровані дані з дослідницьких наборів Kaggle, де зустрічаються агреговані характеристики пристроїв, що наведені в таблиці 2.2. Щоб розуміти, які параметри дійсно впливають на рекомендацію, важливо оцінити їх інформативність. Дослідження останніх років підкреслюють, що при виборі смартфона користувачі звертають увагу не стільки на "сухі" технічні цифри, скільки на функціональні характеристики. Більшість покупців виділяють камеру, автономність і продуктивність як ключові параметри. А дослідження Samsung Research, присвячене користувацьким сценаріям мобільної зйомки, показує зв'язок між частотою фото – та відеозйомки та вибором пристрою за характеристиками сенсора та оптики [24].

*Таблиця 2.2*

**Основні характеристики смартфонів**

|  |  |  |  |
| --- | --- | --- | --- |
| **Категорія параметра** | **Конкретні ознаки** | **Тип даних** | **Коментар** |
| Проду-ктивність | CPU model, CPU cores, CPU frequency, RAM, GPU | Рядкові / Числові | Впливають на швидкість роботи, ігри та багатозадачність. |
| Пам’ять | Internal storage, Expandable storage | Числові / Булеві | Важливі для користувачів, які часто знімають фото/відео або зберігають дані локально. |
| Камери | Main camera MP, Ultra-wide MP, Optical stabilization, Video resolution | Числові / Булеві | Ключові параметри для мобільної фото- та відеозйомки. |
| Дисплей | Display size, Resolution, Pixel density (ppi), Panel type (IPS/OLED) | Числові / Категоріальні | Визначають якість зображення та комфорт використання. |
| Акумулятор | Battery capacity, Fast charging, Wireless charging | Числові / Булеві | Критично важливі для сценаріїв тривалого автономного використання. |
| Зв’язок | 5G support, Wi-Fi type, Bluetooth version | Булеві / Категоріальні | Значимі для продуктивності мережі та швидкості обміну даними. |
| Габарити | Weight, Thickness | Числові | Впливають на зручність у повсякденному використанні. |
| Програмна платформа | OS version, Update policy | Рядкові / Категоріальні | Задають можливості системи, безпеку та термін актуальності. |
| Ціна | Price (USD) | Числові | Один із найбільш чутливих параметрів для алгоритму рекомендацій. |

При розробці рекомендаційної системи важливо сформувати такий набір даних, який одночасно буде досить інформативним для алгоритму і в той же час дозволить побудувати працюючий прототип без надмірного ускладнення моделі. Спочатку розглядалися розширені специфікації смартфонів, подібні до тих, які публікуються у відкритих джерелах (GSMArena, Kaggle та ін.). Такі датасети включають десятки характеристик: модель процесора, тактові частоти, типи матриць, параметри оптичної стабілізації, підтримку бездротової зарядки, версії бездротових протоколів, оновлення ОС і багато іншого.

Однак подібні набори, незважаючи на свою повноту, мають два суттєвих недоліки: вони складні в попередній обробці; велика частина ознак не критична для базового контентно-орієнтованого рекомендатора.

Крім того, при проектуванні системи треба урахувати, що для більшості користувальницьких сценаріїв ключовими залишаються всього кілька характеристик. Дослідження користувальницьких переваг підтверджують, що покупці смартфонів перш за все орієнтуються на камеру, час автономної роботи, продуктивність пристрою і діагональ екрану. Ціна залишається одним з найбільш чутливих параметрів, що визначають підсумковий вибір. Ці показники формують основу реального користувацького досвіду, і на них же орієнтується більшість комерційних платформ, що пропонують підбір пристроїв.

З урахуванням цього, в рамках курсового проекту був використаний спрощений, але достатній для побудови робочої моделі набір характеристик смартфонів: камера (Camera), ємність акумулятора (Battery), оперативна пам'ять (RAM), діагональ екрану (Screen), Ціна (Price).

Такий набір параметрів повністю відповідає можливостям обраного алгоритму (kNN-підхід з нормалізацією ознак) і охоплює найбільш значущі фактори вибору смартфона для основної маси користувачів. Ці ознаки легко формалізуються, нормалізуються і дозволяють обчислювати відстань між пристроями в багатовимірному просторі без додаткової обробки категоріальних або текстових даних.

Структуровані дані завантажуються в базу SQLite у вигляді таблиці phones. Вихідний CSV-файл використовується одноразово для ініціалізації бази, після чого всі операції з пристроями виконуються через SQL-запити. Така архітектура дозволяє легко розширювати каталоги пристроїв і доповнювати їх новими характеристиками без модифікації основної логіки рекомендатора.

Важливо відзначити, що наведений набір характеристик не обмежує подальший розвиток системи. Повний спектр параметрів, що включають продуктивність процесора, тип дисплея, підтримку мереж, габарити пристрою, параметри камер і програмної платформи, розглядається як потенційне розширення моделі. У майбутньому такий багатий набір ознак може бути використаний для побудови гібридних рекомендаційних систем, що включають як контентну, так і поведінкову складову.

Таким чином, фінальна структура даних, використана в проекті, являє собою раціонально спрощений, але інформативний набір ознак, який забезпечує високу інтерпретованість алгоритму і дозволяє отримати якісні рекомендації без надмірної складності. Правильна організація даних і вибір ключових характеристик створюють фундамент для алгоритмічного шару системи і дозволяють надалі масштабувати проект до більш просунутих моделей і розширених датасетів.

## 2.3. Попередня обробка, очищення та нормалізація даних

Робота будь-якої рекомендаційної системи починається з підготовки даних, і саме від якості цього етапу залежить, наскільки коректно буде функціонувати алгоритмічне ядро. В рамках проекту використовувався невеликий, але структурований набір характеристик смартфонів, що включає п'ять ключових параметрів: камеру, ємність акумулятора, обсяг оперативної пам'яті, діагональ екрану і ціну. Ці ознаки були обрані заздалегідь, тому на етапі попередньої обробки акцент змістився не на чистку даних, а на забезпечення їх коректного математичного представлення для подальших обчислень.

На відміну від великих реальних датасетів, де пропуски, дублі і аномальні значення зустрічаються регулярно, вихідні дані, використовувані в системі, не містили відсутніх значень або некоректних записів. Завдяки цьому потреба в класичних процедурах обробки пропусків (наприклад, заповненні медіаною), видаленні дублів або застосуванні складних евристик для очищення даних відпала сама собою. Такий заздалегідь сформований і однорідний набір параметрів дозволив зосередитися на головному завданні – підготовці даних до роботи алгоритму рекомендацій.

Основним кроком попередньої обробки стала нормалізація ознак. Параметри смартфонів вимірюються в різних одиницях: камера - в мегапікселях, акумулятор – в міліампер-годинах, ціна – в доларах/гривнях, діагональ екрану – в дюймах. Якщо використовувати їх в початковому вигляді, параметри з великими числовими значеннями будуть домінувати над іншими, і алгоритм буде сприймати їх як більш важливі, що створює зміщення в результатах. Щоб уникнути подібної ситуації, всі числові ознаки були приведені до Єдиного діапазону від 0 до 1 методом min–max нормалізації. Це рішення забезпечує коректність відстаней в багатовимірному просторі ознак і гарантує, що підсумковий рейтинг пристроїв буде визначатися вагами переваг користувача, а не вихідним масштабом даних.

Окремою частиною попередньої обробки став відбір ознак. Оскільки модель будувалася навколо основних факторів, що впливають на Вибір смартфона, категоричні і важко стандартизовані параметри (тип екрану, інтерфейси зв'язку, версія ОС, тип процесора) були виключені з основного набору. Це дозволило спростити алгоритм і зробити модель більш прозорою, зберігши при цьому достатню чутливість до ключових призначених для користувача сценаріями: зйомці фото і відео, ігровий продуктивності, тривалого часу роботи і зручності відображення контенту.

Таким чином, етап попередньої обробки даних в розробленій системі включає два фундаментальних кроки: відбір і формалізація спрощеного набору числових ознак, релевантних для рекомендаційного алгоритму; приведення всіх характеристик до Єдиного масштабу методом min-max нормалізації, що забезпечує коректну роботу процедури обчислення відстані в kNN-подібної моделі.

Ці операції зведені в таблицю 2.3, що відображає реальні етапи підготовки даних.

*Таблиця 2.3*

**Етапи попередньої обробки даних у рекомендаційній системі**

|  |  |  |  |
| --- | --- | --- | --- |
| **Етап обробки** | **Застосовувані методи** | **Приклади даних** | **Мета використання** |
| **Відбір ознак** | Виключення нерелевантних і категоріальних ознак | Тип дисплея, версія ОС, підтримка 5G | Спрощення структури даних, підвищення інтерпретованості алгоритму |
| **Нормалізація даних** | Min–max scaling | Камера, батарея, RAM, екран, ціна | Приведення ознак до єдиного масштабу для коректного обчислення відстані |
| **Перевірка коректності** | Контроль відсутності пропусків та аномалій | Усі числові параметри | Усунення помилок у даних перед обчисленнями |

В результаті попередня обробка в даній системі являє собою компактний і раціонально організований процес, орієнтований на підтримку контентно-орієнтованого алгоритму рекомендацій. Він забезпечує сувору математичну коректність і високу інтерпретованість моделі, що особливо важливо на етапі побудови прототипу і аналізу поведінки рекомендаційної системи.

## 2.4. Проектування архітектури програмної системи

Проектування архітектури рекомендаційної системи стало тим етапом, на якому абстрактні алгоритми почали перетворюватися в реальний програмний продукт. Коли дивишся на завдання цілком-обробка даних про смартфони, облік призначених для користувача переваг, робота алгоритму і відображення результату в веб-інтерфейсі – швидко розумієш, що без чіткої модульної структури система ризикує перетворитися в набір незв'язаних скриптів. Тому першим кроком стало поділ майбутнього додатка на кілька логічно ізольованих блоків.

Центральне місце в архітектурі займає модуль роботи з даними, який відповідає за зберігання і надання інформації про смартфони. В якості сховища використовується SQLite-легка, але повноцінна реляційна база даних. Модуль виконує ініціалізацію таблиць, завантаження смартфонів з вихідного CSV-файлу, а також збереження історії взаємодії користувачів з системою. Завдяки цьому всі дані виявляються структурованими, доступні через SQL-запити і легко розширювані.

Наступний компонент-модуль формування користувацького запиту, який на відміну від складних профілюючих систем спирається не на накопичені поведінкові дані, а на поточні переваги Користувача, явно зазначені ним в інтерфейсі. Такий підхід обраний усвідомлено: він робить систему прозорою, контрольованою і дозволяє точно розуміти, які параметри впливають на підсумкову рекомендацію. У той же час Історія взаємодій, зібрана в базі даних, розглядається як основа для подальшого розвитку моделі, але в поточній версії алгоритму вона не використовується для автоматичного коригування профілю.

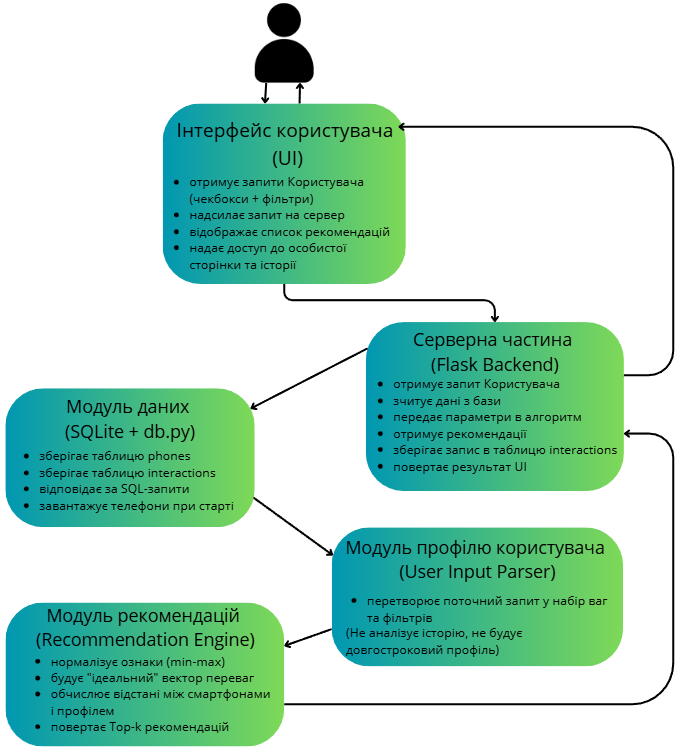
Третім ключовим елементом є алгоритмічний модуль, в якому реалізований контентно-орієнтований підхід до рекомендацій. Він використовує набір нормалізованих характеристик пристрою та обчислює відстань між смартфонами та «ідеальним» профілем користувача за допомогою модифікованої версії методу K-Nearest Neighbors. Алгоритм порівнює пристрої виключно на основі їх об'єктивних технічних параметрів і ваг, визначених обраними Користувачем сценаріями. Такий підхід забезпечує інтерпретованість і стійкість результатів без залучення зовнішніх ml-фреймворків.

За зв'язку всіх компонентів відповідає серверна частина на Flask. Цей модуль отримує запити від користувача, обробляє вхідні параметри, звертається до бази даних, викликає алгоритмічний модуль і формує підсумковий список рекомендацій. Простота Flask дозволяє вибудувати архітектуру так, щоб серверний код залишався компактним, повністю контрольованим і легко модифікується в разі розвитку алгоритму.

Завершує архітектуру користувальницький інтерфейс, створений на основі HTML, CSS і невеликої кількості JavaScript. Він виконує дві функції: надає користувачеві зручний спосіб вказати свої уподобання і відображає рекомендації у вигляді візуально зрозумілих карток смартфонів. Інтерфейс максимально спрощений, але при цьому досить виразний, щоб демонструвати роботу алгоритму і забезпечувати комфортну взаємодію.

Взаємодія всіх компонентів системи підсумовано на малюнку 2.1.

Так сформувалася архітектура системи: не перевантажена, але досить гнучка, щоб розширюватися; компактна, але здатна обробляти реальні дані; і, що для мене особливо важливо, логічна і прозора. Така структура дозволяє поступово покращувати систему, змінювати алгоритми і гнучко підлаштовувати її під реальні призначені для користувача сценарії, не переписуючи все з нуля – мрія будь-якого інженера-програміста.



**Рисунок 2.1** – Спрощена схема архітектури рекомендаційної системи

## 2.5. Проектування інтерфейсу користувача

Створення користувальницького інтерфейсу для рекомендаційної системи виявилося куди більш тонкої завданням, ніж я очікувала спочатку. Чим глибше я занурювалася в проект, тим ясніше розуміла, що інтерфейс – це не просто «обгортка» навколо алгоритму, а повноцінний учасник процесу прийняття рішення. Саме він визначає, наскільки зручним виявиться взаємодія, наскільки зрозумілою буде логіка вибору смартфона і наскільки довіру користувач відчуває до системи.

Основна ідея полягала в тому, щоб інтерфейс залишався легким, чесним і не перевантаженим. Я свідомо орієнтувалася на користувача, який не хоче аналізувати десятки характеристик пристроїв і не зобов'язаний знати відмінності між конкретними моделями процесорів або типами матриць дисплеїв. Такий користувач приходить за рекомендацією, а не за технічною консультацією, і завдання інтерфейсу – максимально спростити комунікацію між людиною і алгоритмом.

В результаті основою форми настройки переваг стали чекбокси і прості Числові поля. Користувач відзначає сценарії використання смартфона-фотографія, ігри, Перегляд відео, активне використання соцмереж, робота або орієнтація на бюджет. Ці варіанти представляють реальну модель поведінки, а не набір складних технічних вимог. Додатково можна вказати верхню межу бюджету, мінімальний обсяг оперативної пам'яті і бажаний діапазон діагоналі екрану – все це задається через звичайні input type="number", які не вимагають додаткових пояснень і не лякають складністю.

Щоб зафіксувати логіку процесу, я намалювала рисунок 2.2 – простий потік взаємодії:

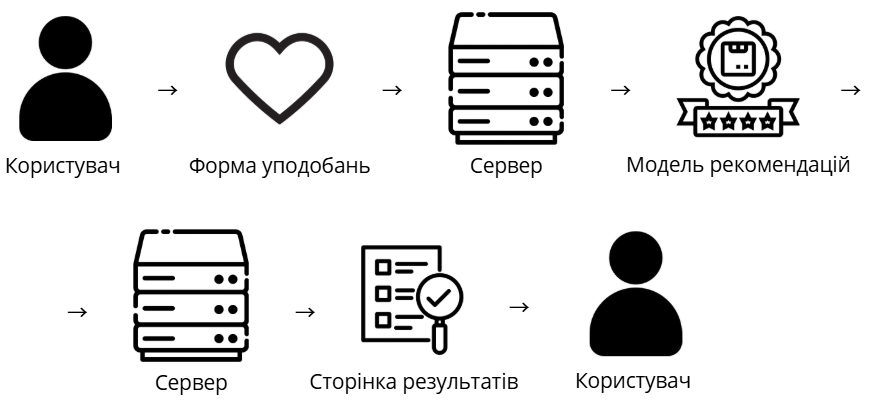


Рисунок 2.2 – Узагальнена схема рекомендальної системи

Такий підхід виявився оптимальним: він дозволяє швидко задати параметри, не перевантажуючи інтерфейс елементами, що вимагають довгого вибору. Я свідомо відмовилася від слайдерів, що випадають списків і багатокрокових анкет. Вони створюють відчуття «контролю», але ускладнюють взаємодію. В рамках навчального проекту акуратність, простота і швидкість були важливішими візуальних експериментів.

Для відображення результатів я вибрала формат карток смартфонів. Кожна картка містить зображення моделі, основні характеристики (камера, ємність батареї, об'єм оперативної пам'яті, діагональ екрану і ціна) і виглядає візуально цілісно. Такий спосіб подання робить результат зрозумілим і інтуїтивним: користувач бачить конкретні пристрої, а не абстрактні рядки таблиці. Це візуально підтримує відчуття того, що система дійсно підібрала кілька реальних, осмислених варіантів.

Відмовилася я і від автоматичного текстового пояснення причин вибору того чи іншого смартфона. Алгоритм використовує зважену метрику відстані до «ідеального» профілю користувача, і спроба перетворити це на фрази цілком могла призвести до неточних або офіційних пояснень. Замість цього акцент зроблений на прозорості механіки вибору: характеристик в картці досить для того, щоб користувач інтуїтивно зрозумів, чому модель запропонована системою.

Підсумковий інтерфейс вийшов саме таким, яким я його собі уявляла: спокійним, ясним, ненав'язливим і чесним. Він не перевантажує користувача параметрами, не створює візуального шуму і не вимагає занурюватися в пристрій алгоритму. Все виглядає так, ніби Користувач веде легкий діалог з системою, а не заповнює складну технічну анкету. І навіть якщо формули, метрики та бази даних працюють за лаштунками, інтерфейс дозволяє людині сприймати це як природний та дружній сервіс.

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

## 3.1. Загальна архітектура програмної реалізації

Без продуманої архітектури проєкт рекомендаційної системи легко ризикує перетворитися на хаотичний набір несинхронізованих скриптів. Завдання полягало не просто в тому, щоб «змусити код працювати», а у створенні повноцінного програмного продукту – з чітким розподілом відповідальності між компонентами, логічною структурою взаємодії модулів і можливістю подальшого розширення функціоналу. Саме тому на початковому етапі основну увагу було зосереджено на проєктуванні архітектури, яка забезпечує прозорість, керованість і стабільність роботи всієї системи.

В кінцевому вигляді додаток вийшло досить компактним, але при цьому добре організованим. Основу серверної логіки становить модуль app.py – саме він відповідає за маршрутизацію, обробку запитів, роботу з сесіями і відображення шаблонів. Його можна сприймати як Центральний «координаційний блок», через який проходять всі призначені для користувача дії.

Другим великим елементом став модуль recommender.py, де зосереджений головний алгоритм – реалізація контентно-орієнтованого пошуку найближчих сусідів. Тут відбувається нормалізація даних, формування ваг під переваги Користувача і обчислення відстані до «ідеального» профілю пристрою. Цей модуль свідомо винесено окремо: так простіше тестувати алгоритм, модернізувати його і при необхідності замінити більш складною моделлю.

Функції зберігання даних і роботи з SQLite я зібрав в модулі db.py. цей блок відповідає за створення таблиць, імпорт каталогів пристроїв, запис історії взаємодій та отримання даних. Таке виділення виявилося зручним: база даних живе своїм самостійним життям, і логіка роботи з нею не перемішується ні з Flask, ні з алгоритмами.

Нарешті, в директорії templates / знаходяться всі HTML-Шаблони, що формують користувальницький інтерфейс, а папка static/ містить зображення смартфонів, CSS-стилі і допоміжні ресурси. Це звична структура для веб-додатків на Flask, але в даному проекті вона ще й підкреслює поділ рівнів: інтерфейс стоїть окремо, алгоритм окремо, серверна логіка окремо.

Архітектура, зображена на рисунку 3.1, побудована навколо дуже простого, але логічного сценарію з чотирьох основних шарів: інтерфейсу, серверної логіки, бази даних і алгоритмічного ядра. У моєму розумінні рекомендаційна система повинна нагадувати діалог: Користувач формулює запит, система його розуміє, обробляє і повертає осмислену відповідь. З цього і народилася ланцюжок взаємодії, яка стала основою архітектури.

Нижче у теблиці 3.1 наведено короткий підсумок основних файлів та каталогів проекту:

*Таблиця 3.1*

**Основні компоненти програмної системи**

|  |  |
| --- | --- |
| **Компонент** | **Призначення** |
| app.py | Точка входу в застосунок, маршрутизація HTTP-запитів, зв’язок інтерфейсу з алгоритмом |
| recommender.py | Алгоритмічне ядро: нормалізація даних, побудова векторів ознак, розрахунок відстаней та формування рекомендацій |
| db.py | Взаємодія з базою даних SQLite: створення таблиць, імпорт даних, запис і читання історії |
| templates/ | HTML-шаблони: головна сторінка, результати, авторизація, профіль користувача |
| static/ | Статичні файли: CSS-стилі, зображення телефонів, допоміжні ресурси |

Така модульна побудова дозволяє сприймати систему не як моноліт, а як конструктор, де кожен компонент чітко виконує свою роль. Якщо виникне необхідність замінити базу даних, змінити алгоритм або переробити інтерфейс – це можна зробити точково, не зачіпаючи всю систему цілком.

## 3.2. Реалізація серверної частини на Flask

«Серверна частина» – це не просто набір функцій, а сполучний шар, який визначає, наскільки природно буде відчуватися робота всієї системи. Для мене Flask став оптимальним інструментом: він легкий, не нав'язує структуру, але при цьому досить потужний, щоб акуратно організувати маршрути, роботу з базою даних і взаємодію з алгоритмом рекомендацій.

Перший крок в реалізації серверної частини-ініціалізація екземпляра Flask. Це той момент, коли додаток як би «прокидається» і отримує своє ім'я, конфігурацію і правила поведінки. Фрагмент нижче відображає основну структуру:

|  |
| --- |
| app = Flask(\_\_name\_\_)  app.secret\_key = 'super\_secret\_key\_for\_demo' |

Вибір секретного ключа пов'язаний з використанням сесій: система зберігає інформацію про поточного користувача в cookie, і Flask повинен вміти підписувати ці дані, щоб захистити їх від підміни. Для навчального проекту досить одного константного ключа, хоча в реальній експлуатації він би, звичайно, зберігався в змінних оточення.

Мій проект використовує SQLite, і це рішення виявилося напрочуд зручним. Підключення до бази проводиться не безпосередньо з Flask, а через окремий модуль db.py, який інкапсулює всі операції: від створення таблиць до імпорту даних з CSV. Це дозволило зберегти серверну частину максимально чистою і сфокусованою.

Після ініціалізації програми залишається зареєструвати маршрути-тобто пояснити Flask, яка функція відповідає за той чи інший URL. Цей крок немов встановлює дорожні покажчики всередині програми: коли приходить запит, сервер точно знає, куди його направити.

Продумуючи маршрути, я намагалася зробити структуру зрозумілою навіть тому, хто вперше бачить код. В результаті серверна частина включає п'ять ключових ендпоінтів, кожен з яких виконує конкретну роль в призначеному для користувача сценарії.

Для зручності вся структура зведена в таблицю 3.2:

*Таблиця 3.2*

**Основні маршрути серверної частини**

|  |  |  |
| --- | --- | --- |
| **Маршрут** | **Призначення** | **Що робить** |
| **/** | Головна сторінка | Відображає форму уподобань і запрошує користувача налаштувати параметри підбору |
| **/recommend** | Обробка уподобань | Приймає POST-дані з форми, викликає алгоритм рекомендацій і відображає картки смартфонів |
| **/login** | Авторизація | Дозволяє користувачу увійти в систему після перевірки логіна і пароля |
| **/logout** | Вихід із системи | Завершує користувацьку сесію |
| **/user** | Профіль користувача | Відображає ім’я користувача та історію його взаємодій із системою |

Кожен маршрут реалізований як функція Python, яка повертає HTML-сторінку. Flask бере на себе те, що зазвичай доставляє багато клопоту – розбір даних форми, управління сесією, перенаправлення, генерацію шаблонів.

Наприклад, маршрут / recommend служить сполучною ланкою між інтерфейсом і алгоритмом. Він отримує переваги Користувача, передає їх в модуль recommender.py, отримує результат і віддає його шаблону result.html. На цій стадії додаток працює майже як конвеєр: дані проходять рівно ті етапи, що описані в архітектурній схемі («Користувач → форма → сервер → модель → результат»).

Маршрут / user, навпаки, працює «у зворотний бік»: спочатку перевіряє, чи ввійшов користувач в систему, потім запитує історію взаємодій в базі даних і акуратно відображає її у вигляді таблиці.

Щоб не дублювати код у всіх маршрутах, я використовувала механізм контекстних процесорів:

|  |
| --- |
| @app.context\_processor  def inject\_current\_user():  return dict(current\_user=session.get('user')) |

Контекстний процесор викликається Flask автоматично перед рендерингом будь-якого шаблону і додає змінну current\_user до всіх сторінок. З точки зору коду – це лише кілька рядків, але з точки зору архітектури – це крок до більш акуратної та масштабованої структури.

Тепер будь-який шаблон може просто перевірити:

|  |
| --- |
| {% if current\_user %}  показати "вийти" та ім'я користувача  {% else %}  показати"Увійти"  {% endif %} |

В результаті серверна частина вийшла компактною, логічною і добре організованою. Flask дав можливість вибудувати мінімалістичне, але гнучке додаток, де всі ключові елементи – маршрути, робота з даними, алгоритм і інтерфейс – взаємодіють природно і прозоро. Це той випадок, коли технологія дійсно допомагає, а не стає перешкодою на шляху від ідеї до працюючої системи.

## 3.3. Реалізація бази даних SQLite

Алгоритм рекомендацій спирається на набір характеристик смартфонів, користувачі залишають після себе історію запитів, а сам інтерфейс повинен виводити результати швидко і без втрат. Все це робить базу даних ключовим компонентом всієї архітектури-чим акуратніше вона спроектована, тим стійкіше буде працювати система.

Я обрала минималистичну, але досить виразну структуру: всього дві таблиці – phones і interactions. Вони відображають два головних джерела даних: каталог пристроїв і історію поведінки користувачів. В рамках навчальної дослідницької системи цього виявилося повністю достатньо.

Перша таблиця, phones, зберігає детальну інформацію про смартфони. Це свого роду» каталог " смартфонів, якому довіряє алгоритм. Кожен рядок описує одну модель і зберігає лише ті характеристики, які насправді беруть участь у розрахунках. У спрощеному вигляді структура зображена в таблиці 3.3.

*Таблиця 3.3*

**Структура таблиці phones**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Призначення** |
| **id** | INTEGER PRIMARY KEY | Унікальний ідентифікатор пристрою |
| **name** | TEXT | Назва моделі смартфона |
| **camera** | INTEGER | Кількість мегапікселів основної камери |
| **battery** | INTEGER | Ємність акумулятора |
| **ram** | INTEGER | Обсяг оперативної памʼяті |
| **screen** | REAL | Діагональ екрана |
| **price** | INTEGER | Ціна в умовних одиницях |
| **image\_name** | TEXT | Назва файлу зображення смартфона |

Друга таблиця-interactions-призначена для фіксування дій Користувача в процесі роботи з системою. Фактично це журнал подій, який зберігає: хто виконував підбір смартфона, які переваги вказав, які моделі отримав у відповідь, і коли саме це сталося. Структура таблиці надана в таблиці 3.4.

*Таблиця 3.4*

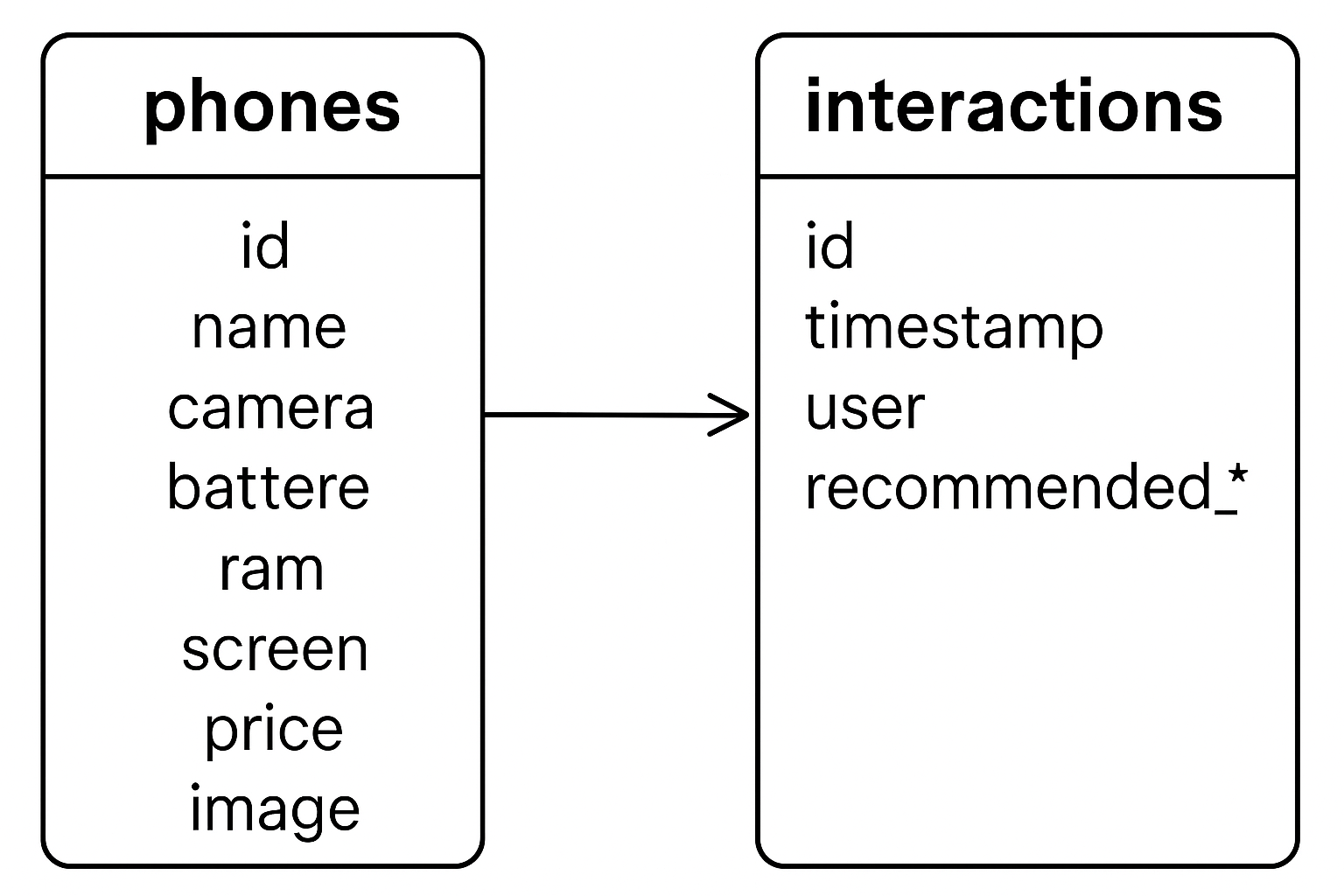
**Структура таблиці interactions**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Призначення** |
| **id** | INTEGER PRIMARY KEY | Унікальний ідентифікатор запису |
| **timestamp** | TEXT | Час виконання запиту |
| **user** | TEXT | Ім’я користувача (або «guest») |
| **preferences** | TEXT | Список уподобань користувача через “;” |
| **recommended\_phones** | TEXT | Назви рекомендованих моделей через “;” |

Особливість цієї таблиці в тому, що вона не використовує зовнішні ключі. На перший погляд це здається спрощенням, але в рамках цього проекту такий підхід виправданий: структура залишається гнучкою, а самі алгоритми ніяк не залежать від строгих обмежень. Крім того, поле recommended\_phones – це простий рядковий список, а не окрема таблиця зв'язків many-to-many. Це вдалий компроміс: логіка залишається зрозумілою, а саму історію легко вивести на сторінку користувача.

Тож маємо, що ці дві таблиці логічно пов'язані між собою не зовнішніми ключами, а смисловою залежністю: interactions містить назви моделей, які реально існують в phones. Такий підхід забезпечує гнучкість, дозволяючи не ускладнювати схему зайвими обмеженнями, але при цьому зберігає достатню структурність для аналітики і подальших поліпшень алгоритму.

Якщо схематично зобразити структуру бази, вона нагадує лаконічну дворівневу модель, представлену на рисунку 3.1.

  
Рисунок 3.1 – Дворівнева модель зберігання даних

Зліва-стабільні дані (каталог пристроїв), праворуч – динамічні (історія поведінки). Разом вони формують основу для «циклу рекомендацій», який живе всередині програми.

Щоб робота з базою була акуратно відокремлена від серверної логіки, я виділила її в окремий модуль db.py. він став таким проміжним» мостом " між сервером Flask і SQLite – всі операції з даними проходять через нього. Такий підхід сильно спрощує підтримку проекту: якщо в майбутньому я захочу перейти, наприклад, на PostgreSQL, досить буде переписати тільки цей модуль, не чіпаючи решті код.

Модуль включає кілька ключових функцій:

– init\_db() – створює файл бази даних та таблиці, якщо вони ще не існують. Це фактично момент «ініціалізації проекту», коли структура даних з'являється на диску.

– seed\_phones\_from\_csv() – одноразовий імпорт смартфонів з вихідного файлу phones.csv. Ця функція наповнює довідник пристроїв і перетворює CSV в зручну SQL-структуру.

– get\_all\_phones() – повертає список всіх смартфонів у вигляді Python-словників. Саме цей метод використовується в алгоритмі рекомендацій.

– log\_interaction() – записує в базу дії користувача: час запиту, його переваги і список запропонованих пристроїв.

– get\_user\_history() – дістає історію взаємодій конкретного користувача для розділу «профіль».

В результаті реалізація бази даних стала фундаментом, на якому тримаються і алгоритм рекомендацій, і персоналізація інтерфейсу, і історія користувача. Мінімалістична структура, продумана логіка модуля db.py і зручність SQLite зробили цю частину системи максимально надійною і при цьому досить гнучкою, щоб проект можна було розвивати далі, не переписуючи його з нуля.

## 3.4. Реалізація алгоритму рекомендацій

Розробка алгоритму рекомендацій виявилася найцікавішим етапом всієї роботи, тому що саме тут абстрактні ідеї перетворюються в конкретні обчислення. Хотілося, щоб модель була не просто набором формул, а логічним і прозорим механізмом, який можна пояснити словами так само легко, як показати в коді. В результаті Архітектура алгоритму склалася навколо контентно-орієнтованого підходу: система аналізує не те, що вибирали інші користувачі, а реальні характеристики смартфонів і переваги конкретної людини в момент запиту. Такий підхід дає повний контроль над моделлю і забезпечує високу інтерпретованість результату.

В основі алгоритму лежить модифікована метрика k-Nearest Neighbors. Однак тут мова йде не про класичний kNN між точками одного простору, а про обчислення відстані між кожним смартфоном і так званим «ідеальним вектором користувача». Цей вектор формується динамічно: алгоритм виділяє ті ознаки, які важливі для обраного сценарію використання, і на їх основі будує точку в багатовимірному просторі характеристик. Ідея здається простою, але в реальності вона дозволяє дуже гнучко підбирати відповідні моделі – від камерофонів до бюджетних варіантів з хорошою автономністю.

Робота алгоритму починається з підготовки даних. Всі відомості про смартфони витягуються з таблиці phones в базі SQLite: камера, батарея, об'єм оперативної пам'яті, діагональ екрану і ціна. Ці параметри досить різноманітні за шкалами вимірювання, тому без нормалізації вони б впливали на результат незбалансовано: наприклад, ціна вимірюється сотнями, а діагональ – всього лише в діапазоні 5-7 дюймів. Щоб привести всі ознаки до єдиного масштабу, застосовується min-max нормалізація, яка переводить кожне значення в діапазон від 0 до 1. На цьому етапі кожен смартфон перетворюється на вектор фіксованої довжини, де координати відображають його об'єктивні характеристики.

Наступний крок-формування профілю користувача. Інтерфейс надає набір чекбоксів, що позначають сценарії використання: фотографія, ігри, перегляд відео, соцмережі, офісні завдання або орієнтація на бюджет. За кожним із цих сценаріїв стоїть набір ваг, який відображає важливість відповідних характеристик. Наприклад, для режиму «ігри» підвищену значимість отримують RAM, екран і батарея, а для «Фото» – камера. Алгоритм підсумовує ваги вибраних сценаріїв і формує єдину матрицю пріоритетів. Після цього будується «ідеальний вектор»: точка з координатами, рівними 1.0 за значущими ознаками і 0.5 – за другорядними. Такий підхід дозволяє висловити очікування користувачів у математичній формі.

Коли і смартфони, і переваги Користувача переведені в загальний простір, починається пошук кращих варіантів. Для кожного пристрою обчислюється зважена евклідова дистанція до ідеального вектора. Вагові коефіцієнти зміщують відстань в сторону тих характеристик, які важливі користувачеві. В результаті телефони, максимально наближені до переваг, виявляються ближче всіх до ідеальної точки. Після сортування по відстані система вибирає три кращі моделі, що є невеликий, але досить інформативний список, який надає користувачеві реальні варіанти для вибору.

Архітектура алгоритму відображена в структурі модуля recommender.py. у таблиці 3.5 нижче наведені ключові функції, що складають основу моделі.

*Таблиця 3.5*

**Основні функції модуля recommender.py**

|  |  |
| --- | --- |
| **Функція** | **Призначення** |
| **\_compute\_feature\_ranges** | Знаходить мінімальні та максимальні значення для кожної ознаки для нормалізації |
| **\_normalize\_value** | Приводить кожне числове значення до діапазону [0, 1] |
| **\_build\_weights** | Формує підсумкові ваги характеристик на основі вибраних уподобань |
| **\_build\_ideal\_vector** | Створює ідеальний вектор користувача для порівняння з пристроями |
| **\_weighted\_euclidean\_distance** | Обчислює відстань між смартфоном та ідеальним вектором з урахуванням ваг |
| **recommend()** | Об’єднує всі етапи алгоритму та повертає top-3 підходящих смартфона |

Кожен із цих методів виконує невелику, але важливу частину загального завдання. Всі разом вони утворюють компактну, але дуже виразну модель, здатну адаптуватися під різні користувацькі запити. Така архітектура зберігає баланс між строгістю математичного підходу і реальною практичною користю. Алгоритм залишається прозорим, легко розширюється і дозволяє в майбутньому додати нові ознаки або поліпшити логіку обчислення відстаней. Саме це робить його хорошою основою для подальших експериментальних і дослідницьких робіт.

## 3.5. Реалізація інтерфейсу користувача та модуля авторизації

Інтерфейс рекомендаційної системи став тим простором, де Технічна логіка зустрічається з людським сприйняттям. Хотілося, щоб користувач не відчував себе гостем в лабораторії машинного навчання, а сприймав систему як спокійний, доброзичливий сервіс, готовий допомогти без зайвих питань. Тому основу візуальної частини я будувала на простоті і мінімалізмі: ніяких важких фільтрів, перевантажених форм і вибору з випадаючих списків довжиною в половину екрану – тільки ключові елементи, які відповідають реальним сценаріям використання смартфона. Користувач думає не термінами «частота CPU» або «тип матриці», а запитами на кшталт «мені потрібно знімати відео» або «хочу, щоб телефон тримав заряд». Саме такі сценарії і лягли в основу структури інтерфейсу.

Центральною точкою входу в систему стала сторінка введення переваг, зображена на рисунку 3.2. Вона представлена у вигляді акуратної форми, де користувач відзначає чекбоксами ті сценарії, які йому дійсно важливі: фотографії, ігри, Перегляд відео, активне використання соцмереж, робочі завдання або орієнтація на бюджетний сегмент. Поруч розташовані невеликі Числові поля-бюджет, мінімальний обсяг оперативної пам'яті, бажана діагональ екрану. Ці поля не перевантажують інтерфейс: користувач просто вводить числа без необхідності рухати повзунки або відкривати складні меню. Все це пов'язано з Flask через стандартну форму POST, так що передача даних відбувається природно і непомітно для користувача.

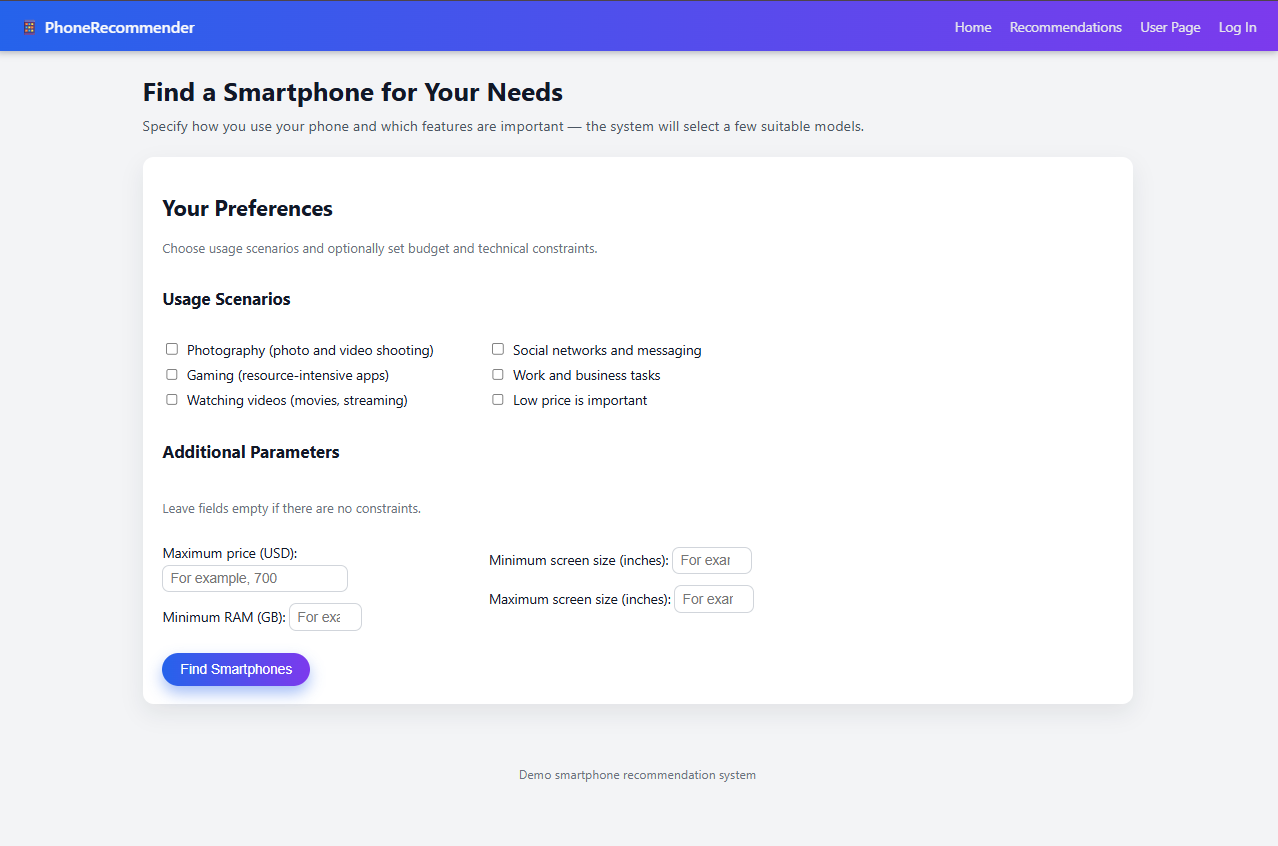


Рисунок 3.2 – Сторінка введення переваг

Результати підбору, надані на рисунку 3.3, представлені в форматі карток – одному з найбільш інтуїтивних і звичних способів візуального сприйняття сучасних рекомендацій. Кожна картка містить зображення моделі, її ключові характеристики (камера, батарея, RAM, екран, ціна) і автоматично підбирається fallback-картинки, якщо для конкретної моделі немає власного фото. Додатково виводяться застосовані фільтри і уточнення, що допомагає користувачеві побачити, які обмеження вплинули на вибір. Такий підхід робить інтерфейс чесним: система не ховає параметри і не видає рекомендації як «магічне» рішення, а показує результати добірки відкрито і структуровано.

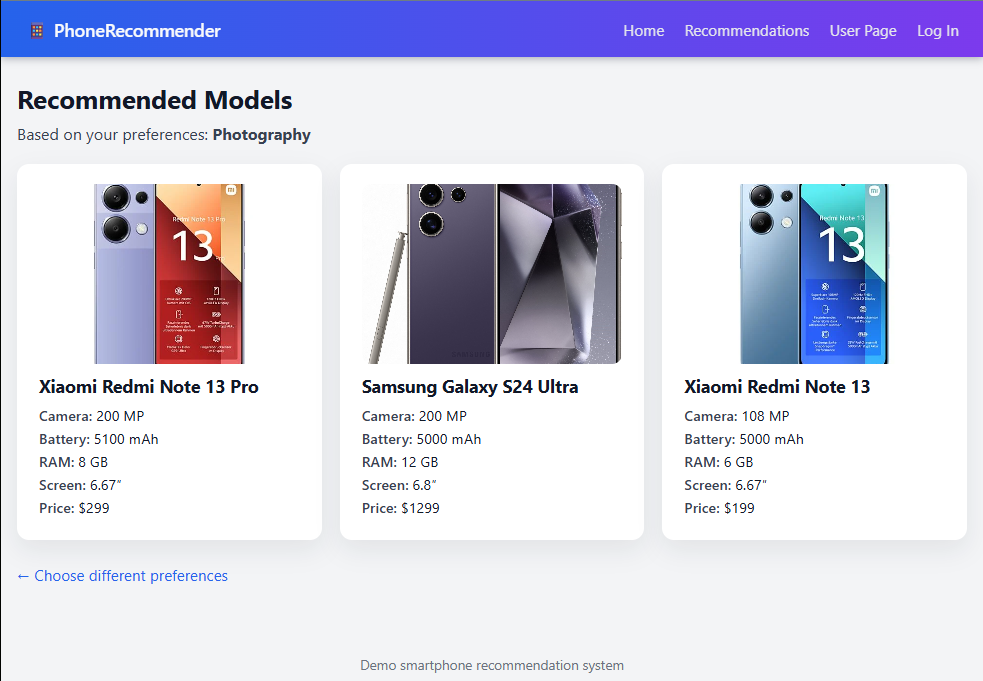


Рисунок 3.3 – Результати підбору

Для тих, хто увійшов в систему під своїм логіном, передбачена окрема сторінка профілю, представлена на рисунку 3.4. Тут відображається ім'я Користувача, а нижче – таблиця історії його взаємодій з рекомендаційною системою. Кожен рядок містить час запиту, вибрані уподобання та список моделей, які були повернуті алгоритмом. Ця сторінка виконує відразу дві функції: по-перше, створює відчуття персоналізації, а по – друге, служить основою для майбутніх розширень-наприклад, для аналізу переваг або формування довгострокового профілю користувача. Вже зараз видно напрямки розвитку: можна додати збережені пристрої, рекомендації на основі історії або індивідуальні налаштування ваг параметрів.

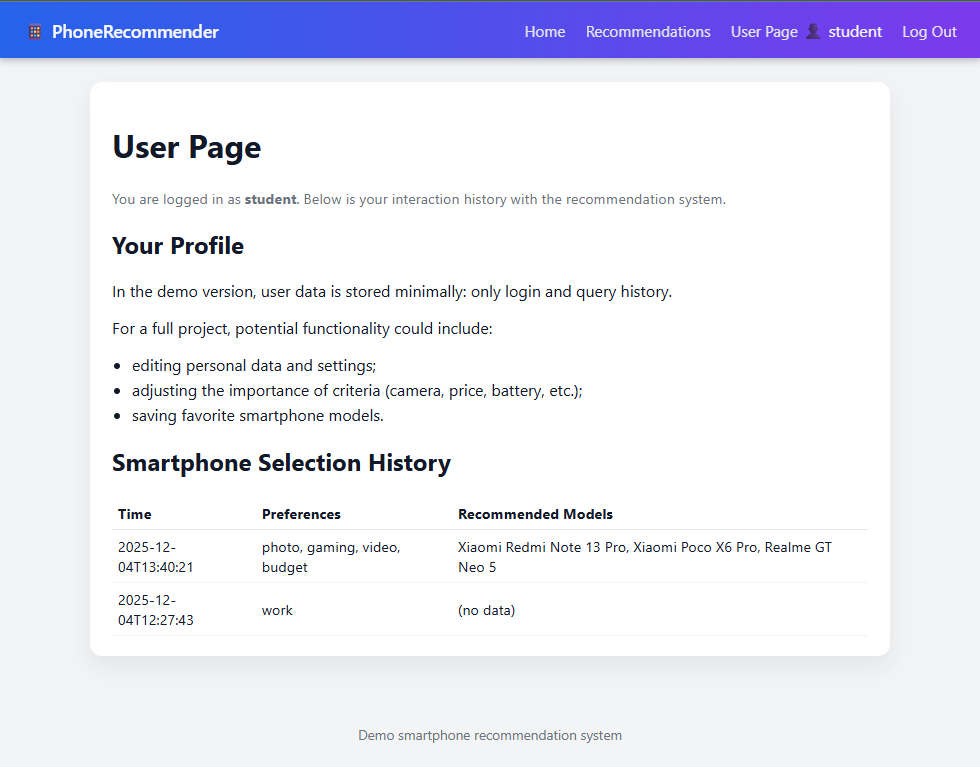


Рисунок 3.4 – Сторінка профілю

Важливу роль в організації інтерфейсу відіграє загальний макет base.html. Саме тут знаходиться Верхня панель навігації, яка об'єднує ключові розділи: «Головна», «рекомендації», «профіль». Навбар реагує на стан авторизації: якщо користувач увійшов, в правій частині відображається його ім'я і посилання «вийти»; якщо немає – посилання «Увійти». Завдяки контекстному процесору Flask змінна current\_user автоматично передається в усі шаблони, так що інтерфейс коректно адаптується під стан сесії без додаткових перевірок в кожному маршруті.

Модуль авторизації теж вийшов гранично легким, але функціональним. Flask використовує cookie-based session, тому дані про поточного користувача зберігаються в session['user'] і доступні на всіх сторінках до виходу. Маршрут /login перевіряє введені дані за словником USERS, а /logout видаляє інформацію з сесії. Все працює швидко, прозоро і не порушує загальний ритм взаємодії користувача з системою. Звичайно, всередині дослідницького проекту логін і пароль зберігаються у відкритому вигляді, але структура вже дозволяє перейти до більш серйозних підходів: реєстрації нових користувачів, зберігання акаунтів в таблиці users і використання хешування паролів через werkzeug.security.

В цілому, реалізація користувальницького інтерфейсу і модуля авторизації виявилася природним продовженням архітектури системи. Інтерфейс залишається легким, не перевантаженим і доброзичливим, а механізми сесій і профілів забезпечують відчуття персоналізації і закріплюють зв'язок між Користувачем і логікою рекомендацій. Саме така комбінація-простота взаємодії і акуратна внутрішня структура – робить систему зручною як для демонстрації, так і для реального використання.

## 3.6 Тестування рекомендаційної системи

Етап тестування став для мене не формальною перевіркою працездатності, а скоріше спостереженням за тим, як себе веде створена система в реальних умовах, в руках «живого» користувача. Після завершення розробки захотілося переконатися, що всі елементи – від маршрутів Flask до алгоритму рекомендацій і інтерфейсу – працюють разом так, як було задумано, без прихованих протиріч і неприємних сюрпризів.

Перевірка почалася з серверної частини. Маршрути Flask – це те, через що проходить будь-який запит, і найменша помилка тут здатна повністю зупинити роботу програми. Я вручну відправляла POST-запити на /recommend, підставляючи як коректні, так і навмисно некоректні дані. Це дозволило переконатися, що додаток стабільно приймає переваги користувача, коректно передає їх в алгоритм і обробляє випадки, коли частина полів залишена порожньою або заповнена некоректно. Окрему увагу приділила маршрутам /login і /logout: важливо було перевірити логіку перенаправлень і переконатися, що механізм сесій не ламається при повторних входах і виходах. Цей міні-тестовий набір показав, що сервер витримує навіть не зовсім «ідеальні» запити, що добре для користувацького досвіду.

Наступним кроком стала перевірка бази даних. Незважаючи на гадану простоту, саме вона зберігає основу всієї функціональності – каталог смартфонів і історію взаємодій. Я перевірила коректність автоматичного створення таблиць phones і interactions, подивилася, як саме відбувається імпорт даних з CSV і чи відображаються вони в SQLite в точності з вихідним набором. Потім протестувала операції запису: кожен запит рекомендацій повинен залишати слід в таблиці interactions, включаючи час, користувача, список переваг і назви виданих моделей. Нарешті, Витяг історії для сторінки профілю також перевірялося вручну-кілька викликів поспіль повинні формувати акуратний журнал дій.

Окрему увагу я приділила тестуванню самого алгоритму рекомендацій. Перевірка почалася з базових речей: чи коректно працює нормалізація ознак, чи однаково обробляються смартфони з близькими характеристиками і чи не виникає ділення на нуль на маленьких вибірках. Далі послідувала серія експериментів з різними комбінаціями переваг: тільки «фото», тільки «ігри», поєднання з трьох і більше сценаріїв. При кожному наборі я перевіряла, чи з'являються у видачі дійсно більш підходящі моделі. Крім того, протестувала фільтри за ціною, RAM і розміром екрану, щоб переконатися, що алгоритм не просто віддає найближчі варіанти, але і поважає обмеження користувача. Про всяк випадок провела тести з порожніми вхідними даними – наприклад, коли людина нічого не вибрав, – щоб система не падала, а повертала розумний набір рекомендацій.

Паралельно з цим тестувався інтерфейс. Спочатку функціонально: чи коректно відправляються форми, чи оновлюються зображення смартфонів, чи правильно відображаються картки пристроїв. Потім-візуально: як поводиться верстка на різних дозволах, чи немає перекриттів, «поїхали» блоки або занадто довгі назви, які ламають сітку. Всі ключові сценарії – від головної сторінки до історії користувача пройшли цю перевірку.

Фінальну крапку в тестуванні поставили сценарії, що імітують реальну поведінку різних типів користувачів. В одному випадку я виступала як новий користувач, у якого немає історії і який просто хоче підібрати смартфон швидко і без деталей. У другому – як зареєстрований користувач, який вже робив запити раніше і тепер хоче побачити історію рекомендацій. І нарешті, тестувала складні кейси: кілька переваг одночасно, фільтри по бюджету і екрану, спроба підібрати телефон на дуже маленький бюджет і т.д. всі три сценарії продемонстрували стабільну роботу системи.

Підводячи підсумок тестування, можна сказати, що рекомендаційна система відповідає тим функціональним вимогам, які я ставила перед собою на початку. Вона дійсно генерує осмислені рекомендації, коректно враховує переваги користувача, не ламається на некоректних даних і залишається стабільною навіть при нетипових комбінаціях фільтрів. Проте в процесі тестування стали помітні і обмеження поточної версії. Наприклад, система поки не використовує колаборативну фільтрацію: історія користувача фіксується, але не впливає на подальші рекомендаці.

Ці спостереження підказують напрямки подальшого розвитку. У наступній версії можна передбачити навчання ваг на основі накопиченої історії запитів, розширити набір характеристик пристроїв і додати більш складні методи машинного навчання. У наступній версії можна передбачити навчання ваг на основі накопиченої історії запитів, розширити набір характеристик пристроїв.

# ВИСНОВКИ

Результатом виконаної роботи стала повноцінна рекомендаційна система підбору мобільних пристроїв, що об'єднує сучасні методи обробки даних, алгоритмічний аналіз і практичну веб-реалізацію. Всі завдання, поставлені у вступі, були послідовно вирішені: проведено аналіз предметної області, спроектована архітектура системи, розроблений алгоритм рекомендацій, створено веб-додаток і виконано його тестування.

У ході дослідження було обрано та впроваджено контентно-орієнтований підхід із використанням модифікованої версії методу k-найближчих сусідів. Такий вибір виправдав себе в контексті розроблюваної задачі: алгоритм виявився досить гнучким, щоб враховувати різні сценарії використання смартфона, і в той же час досить прозорим, щоб зберігати інтерпретованість і простоту подальшого розвитку. Додаткове застосування нормалізації ознак і динамічного формування ваг дозволило домогтися більш точного обліку переваг користувача.

Практична частина роботи завершилася розробкою серверного додатка на Flask з інтуїтивним веб-інтерфейсом і використанням SQLite для зберігання даних. Створена система надає користувачеві зручну форму введення переваг, відображає рекомендації в наочному вигляді і підтримує авторизацію з веденням історії взаємодій.

Тестування підтвердило коректність роботи всіх модулів: серверні маршрути стабільно обробляють запити, база даних коректно зберігає і витягує дані, алгоритм впевнено працює при різних наборах переваг і обмежень, а інтерфейс залишається зрозумілим і стійким на різних дозволах.

З практичної точки зору розроблена система може застосовуватися як в освітніх проектах, так і як прототип для комерційних рекомендаційних сервісів в сферах електронної комерції та онлайн-каталогів. Архітектура програми легко розширювана.

# Список використаних джерел

1. Forbes. How Much Data Do We Create Every Day [Электронный ресурс] / Forbes. – 2018. – Режим доступа: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/>
2. McKinsey & Company. The Future of Personalization / McKinsey & Company. – 2020. – [Электронный ресурс]. – Режим доступа: <https://www.mckinsey.com>
3. Adobe. Digital Trends Report / Adobe. – 2021. – [Электронный ресурс]. – Режим доступа: <https://www.adobe.com>
4. IDC. Data Creation and Replication Report / IDC. – 2021. – [Электронный ресурс]. – Режим доступа: <https://www.idc.com>
5. GSMA. Mobile Economy Report 2022. – 2022. – [Электронный ресурс]. – Режим доступа: <https://www.gsmaintelligence.com/research/the-mobile-economy-2022>
6. Deloitte. Global Mobile Consumer Trends 2021 / Deloitte. – 2021. – [Электронный ресурс]. – Режим доступа: <https://www.deloitte.com/global/en/Industries/tmt/perspectives/gx-global-mobile-consumer-trends.html>
7. Google. Consumer Insights: Mobile Shopping Trends / Google. – 2020. – [Электронный ресурс]. – Режим доступа: <https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-shopping-statistics/>
8. Boston Consulting Group. The Consumer Value Study / Boston Consulting Group. – 2021. – [Электронный ресурс]. – Режим доступа: <https://www.bcg.com/capabilities/customer-insights/insights>
9. Berger+. Що означає «система рекомендацій» [Електронний ресурс] / Berger+. – 04.02.2025. – Режим доступу: <https://www.berger.team/uk/glossar/empfehlungssystem/>
10. Ontañón S., Zhu J. The Personalization Paradox: the Conflict between Accurate User Models and Personalized Adaptive Systems / MIT Technology Review. – 2021. – [Электронный ресурс]. – Режим доступа: <https://doi.org/10.48550/arXiv.2103.01771>
11. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook / Springer. – 2022. – 1060 с. – ISBN 978‑1‑0716‑2196‑7.
12. Netflix Technology Blog. Personalization and Recommendation at Netflix / Netflix Technology Blog. – 2020. – [Электронний ресурс]. – Режим доступа: <https://netflixtechblog.com/>
13. IEEE Access. Hybrid Recommender Systems: A Survey / IEEE Access. –2021. – [Электронний ресурс]. – Режим доступа: <https://ieeexplore.ieee.org/abstract/document/10716623>
14. Zhang S., Yao L., Sun A., Tay Y. Deep Learning Based Recommender System: A Survey and New Perspectives / ACM Computing Surveys. – 2019. – 52, 1: 5‑38. – DOI: 10.1145/3285029.
15. A Comparative Evaluation of Neighbor-Based Collaborative Filtering / … // ACM. – 2020. – [Электронний ресурс]. – DOI: 10.1145/3383313.3412244 – Режим доступу: <https://dl.acm.org/doi/10.1145/3383313.3412244>
16. Clustering-Based Recommendations for High-Dimensional Data / [Электронный ресурс] // Springer, 2021. – Режим доступу: <https://link.springer.com/chapter/10.1007/978-3-030-73689-7_12>
17. Deep Neural Networks for Content-Based Recommender Systems / [Электронный ресурс] // IEEE, 2021. – Режим доступу: <https://ieeexplore.ieee.org/document/9442793>
18. Netflix Tech Blog. Learning a Personalized Embedding for Recommendations / Netflix Tech Blog. – 2022. – [Электронний ресурс]. – Режим доступу: <https://netflixtechblog.com/learning-a-personalized-embedding-for-recommendations-3b3b1e4a8f6f>
19. Amazon Web Services. What is Amazon Personalize? [Електронний ресурс] / Amazon Web Services. – 2023. – Режим доступу: <https://docs.aws.amazon.com/personalize/latest/dg/what-is-personalize.html>
20. Zhang S., Yao L., Sun A., Tay Y. Deep Learning Based Recommender System: A Survey and New Perspectives / ACM Computing Surveys. – 2019. – Vol. 52, No. 1. – Article 5. – DOI: 10.1145/3285029 – [Электронний ресурс]. — Режим доступу: <https://dl.acm.org/doi/10.1145/3285029>
21. Das J., Majumder S., Mali K. Clustering Techniques to Improve Scalability and Accuracy of Recommender Systems / International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. – 2021. – Vol. 29, № 4. – [Электронний ресурс]. – DOI: 10.1142/S0218488521500276 – Режим доступу: <https://doi.org/10.1142/S0218488521500276>
22. Google Research. Understanding User Intent in Recommendation Systems / Google Research. – 2021. – [Электронний ресурс]. – Режим доступу: <https://research.google/pubs/pub51694/>
23. IEEE Spectrum. Top Programming Languages 2023 / IEEE Spectrum. – 2023. – [Электронний ресурс]. – Режим доступу: <https://spectrum.ieee.org/top-programming-languages-2023>
24. Samsung Research. User Scenarios in Mobile Photography / Samsung Research. – 2021. – [Электронний ресурс]. – Режим доступу: <https://research.samsung.com>

# Додаток А

**Повний вихідний код реалізованої системи**

**Лістинг коду app.py**

|  |
| --- |
| from flask import Flask, request, render\_template, redirect, url\_for, session  import recommender  import db  app = Flask(\_\_name\_\_)  app.secret\_key = 'super\_secret\_key\_for\_demo'  # for sessions  # Demo "user database"  USERS = {      'student': 'password123',      'admin': 'admin123'  }  # Initialize DB and import phones at app startup  db.init\_db()  # db.seed\_phones\_from\_csv('data/phones.csv')  def \_parse\_int(value):      if not value:          return None      try:          return int(value)      except ValueError:          return None  def \_parse\_float(value):      if not value:          return None      try:          return float(value)      except ValueError:          return None  @app.context\_processor  def inject\_current\_user():      return dict(current\_user=session.get('user'))  @app.route('/', methods=['GET'])  def index():      return render\_template('index.html', title="Phone Selector")  @app.route('/recommend', methods=['POST'])  def recommend\_view():      selected\_prefs = request.form.getlist('preferences')      price\_max = \_parse\_int(request.form.get('price\_max'))      ram\_min = \_parse\_int(request.form.get('ram\_min'))      screen\_min = \_parse\_float(request.form.get('screen\_min'))      screen\_max = \_parse\_float(request.form.get('screen\_max'))      # Get phones from DB      phones\_data = db.get\_all\_phones()      recommended\_phones = recommender.recommend(          selected\_prefs,          phones\_data,          k=3,          price\_max=price\_max,          ram\_min=ram\_min,          screen\_min=screen\_min,          screen\_max=screen\_max      )      user\_id = session.get('user')      db.log\_interaction(user\_id, selected\_prefs, recommended\_phones)      # Mapping preferences to English names for display      pref\_names = {          'photo': 'Photography',          'gaming': 'Gaming',          'video': 'Video Watching',          'social': 'Social Media & Messaging',          'work': 'Work & Productivity',          'budget': 'Budget Friendly'      }      preferences\_en = [pref\_names[p] for p in selected\_prefs if p in pref\_names]      filters = {          'price\_max': price\_max,          'ram\_min': ram\_min,          'screen\_min': screen\_min,          'screen\_max': screen\_max      }      return render\_template(          'result.html',          title="Recommendations",          recommended=recommended\_phones,          preferences=preferences\_en,          filters=filters      )  @app.route('/login', methods=['GET', 'POST'])  def login():      error = None      if request.method == 'POST':          username = request.form.get('username', '').strip()          password = request.form.get('password', '').strip()          if username in USERS and USERS[username] == password:              session['user'] = username              return redirect(url\_for('user\_page'))          else:              error = "Invalid username or password"      return render\_template('login.html', title="Login", error=error)  @app.route('/logout')  def logout():      session.pop('user', None)      return redirect(url\_for('index'))  @app.route('/user', methods=['GET'])  def user\_page():      user\_id = session.get('user')      history = db.get\_user\_history(user\_id, limit=10) if user\_id else []      return render\_template(          'user.html',          title="User Page",          user=user\_id,          history=history      )  if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True) |

**Лістинг коду recommender.py**

|  |
| --- |
| import csv  import math  import os  from datetime import datetime  # Feature importance weights for each user preference category.  # New preferences:  #  - social: focus on battery, moderate camera and screen  #  - work: focus on battery and RAM  #  - budget: handled separately in \_build\_weights  weight\_map = {      'photo':  {'camera': 3, 'battery': 1, 'ram': 1, 'screen': 1},  # Photography      'gaming': {'camera': 0, 'battery': 2, 'ram': 3, 'screen': 2},  # Gaming      'video':  {'camera': 0, 'battery': 2, 'ram': 1, 'screen': 4},  # Video      'social': {'camera': 1, 'battery': 2, 'ram': 1, 'screen': 1},  # Social networks & messaging      'work':   {'camera': 1, 'battery': 2, 'ram': 2, 'screen': 1},  # Work & business tasks      # 'budget' — handled separately via price  }  def \_compute\_feature\_ranges(phones):      """      Compute min and max values for each numeric feature for normalization.      """      cameras = [p['Camera'] for p in phones]      batteries = [p['Battery'] for p in phones]      rams = [p['RAM'] for p in phones]      screens = [p['Screen'] for p in phones]      prices = [p['Price'] for p in phones]      ranges = {          'Camera': (min(cameras), max(cameras)),          'Battery': (min(batteries), max(batteries)),          'RAM': (min(rams), max(rams)),          'Screen': (min(screens), max(screens)),          'Price': (min(prices), max(prices))      }      return ranges  def \_normalize\_value(value, vmin, vmax):      """      Normalize value to [0,1]. Returns 0.5 if min == max.      """      if vmax == vmin:          return 0.5      return (value - vmin) / (vmax - vmin)  def \_phone\_to\_vector(phone, ranges):      """      Convert phone into a normalized numeric feature vector.      Feature order: [camera, battery, ram, screen, price].      Price is inversely normalized (cheaper = better).      """      cam = \_normalize\_value(phone['Camera'], \*ranges['Camera'])      bat = \_normalize\_value(phone['Battery'], \*ranges['Battery'])      ram = \_normalize\_value(phone['RAM'], \*ranges['RAM'])      scr = \_normalize\_value(phone['Screen'], \*ranges['Screen'])      price\_raw = \_normalize\_value(phone['Price'], \*ranges['Price'])      price = 1.0 - price\_raw  # invert price: cheaper = closer to 1      return [cam, bat, ram, scr, price]  def \_build\_weights(user\_prefs):      """      Build final feature weights based on user's selected preferences.      Returns a dict with weights for camera, battery, ram, screen, price.      """      weights = {'camera': 0.0, 'battery': 0.0, 'ram': 0.0, 'screen': 0.0}      for pref in user\_prefs:          if pref in weight\_map:              for feature, w in weight\_map[pref].items():                  weights[feature] += w      # Base handling for price      if not user\_prefs:          # If user has no preferences – uniform distribution          weights = {'camera': 1.0, 'battery': 1.0, 'ram': 1.0, 'screen': 1.0}          price\_weight = 2.0  # emphasize price more      else:          # If preferences exist, price is considered but not dominant          price\_weight = 1.0      # If user explicitly wants low price — increase price weight      if 'budget' in user\_prefs:          price\_weight += 2.0      weights['price'] = price\_weight      return weights  def \_build\_ideal\_vector(weights):      """      Build the "ideal" phone vector in normalized feature space.      For features with non-zero weight, ideal = 1 (max),      for zero-weight features = 0.5 (neutral).      Price ideal = 1 (cheaper is better).      Returns vector in order [camera, battery, ram, screen, price].      """      def ideal\_for\_feature(name):          return 1.0 if weights[name] > 0 else 0.5      ideal\_cam = ideal\_for\_feature('camera')      ideal\_bat = ideal\_for\_feature('battery')      ideal\_ram = ideal\_for\_feature('ram')      ideal\_scr = ideal\_for\_feature('screen')      ideal\_price = 1.0      return [ideal\_cam, ideal\_bat, ideal\_ram, ideal\_scr, ideal\_price]  def \_weighted\_euclidean\_distance(vec\_phone, vec\_ideal, weights):      """      Weighted Euclidean distance between phone and ideal vector.      This is the k-nearest neighbors core:      we look for k phones closest to the "ideal" device.      """      feature\_order = ['camera', 'battery', 'ram', 'screen', 'price']      diff\_sum = 0.0      for i, feature\_name in enumerate(feature\_order):          w = weights.get(feature\_name, 0.0)          if w <= 0:              continue          diff = vec\_phone[i] - vec\_ideal[i]          diff\_sum += w \* (diff \*\* 2)      return math.sqrt(diff\_sum)  def recommend(user\_prefs, phones, k=3,                price\_max=None, ram\_min=None,                screen\_min=None, screen\_max=None):      """      Return top-k phone models recommended based on user preferences.      Steps:      1. Apply hard filters (price, RAM, screen size).      2. Build "ideal" vector in normalized feature space.      3. Find k nearest neighbors (minimum weighted distance).      """      if not phones:          return []      # 1. Filter by hard constraints      filtered = []      for phone in phones:          if price\_max is not None and phone['Price'] > price\_max:              continue          if ram\_min is not None and phone['RAM'] < ram\_min:              continue          if screen\_min is not None and phone['Screen'] < screen\_min:              continue          if screen\_max is not None and phone['Screen'] > screen\_max:              continue          filtered.append(phone)      # If nothing remains after filtering, use full list      if not filtered:          filtered = phones      ranges = \_compute\_feature\_ranges(filtered)      weights = \_build\_weights(user\_prefs)      ideal\_vec = \_build\_ideal\_vector(weights)      scored = []      for phone in filtered:          phone\_vec = \_phone\_to\_vector(phone, ranges)          dist = \_weighted\_euclidean\_distance(phone\_vec, ideal\_vec, weights)          scored.append((dist, phone))      scored.sort(key=lambda x: x[0])      top\_recommendations = [item[1] for item in scored[:k]]      return top\_recommendations  def log\_user\_interaction(user\_id, user\_prefs, recommended\_phones, log\_path='data/behavior\_log.csv'):      """      Log user behavior:      - user\_id: user identifier (login/name)      - selected preferences      - recommended phones      Data is stored in CSV and can be used for behavior analysis (Data Mining).      """      os.makedirs(os.path.dirname(log\_path), exist\_ok=True)      file\_exists = os.path.isfile(log\_path)      with open(log\_path, 'a', newline='', encoding='utf-8') as f:          writer = csv.writer(f)          if not file\_exists:              writer.writerow(['timestamp', 'user', 'preferences', 'recommended\_phones'])          timestamp = datetime.now().isoformat(timespec='seconds')          prefs\_str = ';'.join(user\_prefs) if user\_prefs else ''          rec\_names = ';'.join([p['Name'] for p in recommended\_phones])          user\_str = user\_id if user\_id else 'guest'          writer.writerow([timestamp, user\_str, prefs\_str, rec\_names])  def get\_user\_history(user\_id, log\_path='data/behavior\_log.csv', limit=10):      """      Return the last `limit` recommendation records for the specified user.      If user\_id == None, returns an empty list.      """      if not user\_id:          return []      if not os.path.isfile(log\_path):          return []      history = []      with open(log\_path, newline='', encoding='utf-8') as f:          reader = csv.DictReader(f)          for row in reader:              if row.get('user') != user\_id:                  continue              history.append({                  'timestamp': row.get('timestamp'),                  'preferences': row.get('preferences', '').split(';') if row.get('preferences') else [],                  'recommended\_phones': row.get('recommended\_phones', '').split(';') if row.get('recommended\_phones') else []              })      # last `limit` records      history = history[-limit:]      # reverse so newest are first      history.reverse()      return history |

**Лістинг коду db.py**

|  |
| --- |
| import sqlite3  import os  import csv  from datetime import datetime  DB\_PATH = "database.db"  def get\_connection():      """      Returns a connection to the SQLite database.      """      conn = sqlite3.connect(DB\_PATH)      return conn  def init\_db():      """      Creates tables if they do not exist:      - phones: smartphone catalog      - interactions: user interaction history      """      conn = get\_connection()      cur = conn.cursor()      # Phones table      cur.execute("""      CREATE TABLE IF NOT EXISTS phones (          id INTEGER PRIMARY KEY AUTOINCREMENT,          name TEXT NOT NULL,          camera INTEGER NOT NULL,          battery INTEGER NOT NULL,          ram INTEGER NOT NULL,          screen REAL NOT NULL,          price INTEGER NOT NULL,          image\_name TEXT      )      """)      # Interaction history table      cur.execute("""      CREATE TABLE IF NOT EXISTS interactions (          id INTEGER PRIMARY KEY AUTOINCREMENT,          timestamp TEXT NOT NULL,          user TEXT NOT NULL,          preferences TEXT,          recommended\_phones TEXT      )      """)      conn.commit()      conn.close()  def seed\_phones\_from\_csv(csv\_path="data/phones.csv"):      """      One-time population of the phones table from a CSV file if the table is empty.      """      if not os.path.exists(csv\_path):          print(f"[seed\_phones\_from\_csv] File {csv\_path} not found, skipping import.")          return      conn = get\_connection()      cur = conn.cursor()      cur.execute("SELECT COUNT(\*) FROM phones")      count = cur.fetchone()[0]      if count > 0:          # Table already has data – do nothing          conn.close()          return      with open(csv\_path, newline="", encoding="utf-8") as f:          reader = csv.DictReader(f)          rows = list(reader)      for row in rows:          name = row["Name"]          camera = int(row["Camera"])          battery = int(row["Battery"])          ram = int(row["RAM"])          screen = float(row["Screen"])          price = int(row["Price"])          # Generate image filename based on model name          # Example: "PhotoMaster 3000" -> "PhotoMaster\_3000.jpg"          image\_name = name.replace(" ", "\_") + ".jpg"          cur.execute(              """              INSERT INTO phones (name, camera, battery, ram, screen, price, image\_name)              VALUES (?, ?, ?, ?, ?, ?, ?)              """,              (name, camera, battery, ram, screen, price, image\_name),          )      conn.commit()      conn.close()      print(f"[seed\_phones\_from\_csv] Imported {len(rows)} phones.")  def get\_all\_phones():      """      Returns a list of phones in the same format as CSV:      list of dictionaries with keys Name, Camera, Battery, RAM, Screen, Price.      """      conn = get\_connection()      cur = conn.cursor()      cur.execute("""          SELECT name, camera, battery, ram, screen, price, image\_name          FROM phones      """)      rows = cur.fetchall()      conn.close()      phones = []      for name, camera, battery, ram, screen, price, image\_name in rows:          phones.append({              "Name": name,              "Camera": camera,              "Battery": battery,              "RAM": ram,              "Screen": screen,              "Price": price,              "Image": image\_name,  # for future use if needed          })      return phones  def log\_interaction(user\_id, user\_prefs, recommended\_phones):      """      Saves a recommendation record into the interactions table.      """      conn = get\_connection()      cur = conn.cursor()      ts = datetime.now().isoformat(timespec="seconds")      prefs\_str = ";".join(user\_prefs) if user\_prefs else ""      rec\_names = ";".join([p["Name"] for p in recommended\_phones]) if recommended\_phones else ""      user\_str = user\_id if user\_id else "guest"      cur.execute(          """          INSERT INTO interactions (timestamp, user, preferences, recommended\_phones)          VALUES (?, ?, ?, ?)          """,          (ts, user\_str, prefs\_str, rec\_names),      )      conn.commit()      conn.close()  def get\_user\_history(user\_id, limit=10):      """      Returns the last `limit` interaction records for the specified user (by name).      Each record format:      {          "timestamp": ...,          "preferences": [...],          "recommended\_phones": [...]      }      """      if not user\_id:          return []      conn = get\_connection()      cur = conn.cursor()      cur.execute(          """          SELECT timestamp, preferences, recommended\_phones          FROM interactions          WHERE user = ?          ORDER BY id DESC          LIMIT ?          """,          (user\_id, limit),      )      rows = cur.fetchall()      conn.close()      history = []      for ts, prefs\_str, rec\_str in rows:          prefs = prefs\_str.split(";") if prefs\_str else []          recs = rec\_str.split(";") if rec\_str else []          history.append({              "timestamp": ts,              "preferences": prefs,              "recommended\_phones": recs,          })      return history |

**Лістинг коду base.html**

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>{{ title or "Recommendation System" }}</title>      <link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">  </head>  <body>  <header class="navbar">      <div class="navbar\_\_logo">          📱 PhoneRecommender      </div>      <nav class="navbar\_\_links">          <a href="{{ url\_for('index') }}">Home</a>          <a href="{{ url\_for('index') }}#form">Recommendations</a>          <a href="{{ url\_for('user\_page') }}">User Page</a>          {% if current\_user %}              <span class="navbar\_\_user">👤 {{ current\_user }}</span>              <a href="{{ url\_for('logout') }}">Log Out</a>          {% else %}              <a href="{{ url\_for('login') }}">Log In</a>          {% endif %}      </nav>  </header>  <main class="page">      {% block content %}{% endblock %}  </main>  <footer class="footer">      <p>Demo smartphone recommendation system</p>  </footer>  </body>  </html> |

**Лістинг коду index.html**

|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <section class="hero">      <div class="hero\_\_content">          <h1>Find a Smartphone for Your Needs</h1>          <p>Specify how you use your phone and which features are important — the system will select a few suitable models.</p>      </div>  </section>  <section class="form-section" id="form">      <div class="card card--form">          <h2>Your Preferences</h2>          <p class="muted">Choose usage scenarios and optionally set budget and technical constraints.</p>          <form action="{{ url\_for('recommend\_view') }}" method="post" class="form-grid">              <div class="form-group form-group--full">                  <h3>Usage Scenarios</h3>              </div>              <div class="form-group">                  <label>                      <input type="checkbox" name="preferences" value="photo">                      Photography (photo and video shooting)                  </label>                  <label>                      <input type="checkbox" name="preferences" value="gaming">                      Gaming (resource-intensive apps)                  </label>                  <label>                      <input type="checkbox" name="preferences" value="video">                      Watching videos (movies, streaming)                  </label>              </div>              <div class="form-group">                  <label>                      <input type="checkbox" name="preferences" value="social">                      Social networks and messaging                  </label>                  <label>                      <input type="checkbox" name="preferences" value="work">                      Work and business tasks                  </label>                  <label>                      <input type="checkbox" name="preferences" value="budget">                      Low price is important                  </label>              </div>              <div class="form-group form-group--full">                  <h3>Additional Parameters</h3>                  <p class="muted">Leave fields empty if there are no constraints.</p>              </div>              <div class="form-group">                  <label>                      Maximum price (USD):                      <input type="number" name="price\_max" min="0" step="50" placeholder="For example, 700">                  </label>                  <label>                      Minimum RAM (GB):                      <input type="number" name="ram\_min" min="1" max="32" step="1" placeholder="For example, 6">                  </label>              </div>              <div class="form-group">                  <label>                      Minimum screen size (inches):                      <input type="number" name="screen\_min" min="4" max="8" step="0.1" placeholder="For example, 5.5">                  </label>                  <label>                      Maximum screen size (inches):                      <input type="number" name="screen\_max" min="4" max="8" step="0.1" placeholder="For example, 6.7">                  </label>              </div>              <div class="form-actions form-group--full">                  <button type="submit" class="btn-primary">Find Smartphones</button>              </div>          </form>      </div>  </section>  {% endblock %} |

**Лістинг коду login.html**

|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <section class="user-section">      <div class="card card--user">          <h1>Login</h1>          <p class="muted">              In the demo version, pre-defined accounts are used for authentication.              For example: <code>student / password123</code>.          </p>          {% if error %}              <p style="color: #b91c1c; font-weight: 500;">{{ error }}</p>          {% endif %}          <form action="{{ url\_for('login') }}" method="post" class="form-grid">              <div class="form-group form-group--full">                  <label>                      Username:                      <input type="text" name="username" required>                  </label>              </div>              <div class="form-group form-group--full">                  <label>                      Password:                      <input type="password" name="password" required>                  </label>              </div>              <div class="form-actions form-group--full">                  <button type="submit" class="btn-primary">Log In</button>              </div>          </form>      </div>  </section>  {% endblock %} |

**Лістинг коду result.html**

|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <section class="results-header">      <h1>Recommended Models</h1>      {% if preferences and preferences|length > 0 %}          <p>Based on your preferences: <strong>{{ preferences|join(', ') }}</strong></p>      {% else %}          <p>General recommendations (no preferences specified)</p>      {% endif %}      {% if filters and (filters.price\_max or filters.ram\_min or filters.screen\_min or filters.screen\_max) %}          <div class="filters-box">              <p>Additional constraints considered:</p>              <ul>                  {% if filters.price\_max %}                      <li>Price up to: ${{ filters.price\_max }}</li>                  {% endif %}                  {% if filters.ram\_min %}                      <li>Minimum RAM: {{ filters.ram\_min }} GB</li>                  {% endif %}                  {% if filters.screen\_min %}                      <li>Screen at least: {{ "%.1f"|format(filters.screen\_min) }}″</li>                  {% endif %}                  {% if filters.screen\_max %}                      <li>Screen at most: {{ "%.1f"|format(filters.screen\_max) }}″</li>                  {% endif %}              </ul>          </div>      {% endif %}  </section>  <section class="cards-grid">      {% for phone in recommended %}      <article class="card card--phone">          <div class="card\_\_image">              <img              src="{{ url\_for('static', filename='images/' ~ phone.Image) }}"              onerror="this.onerror=null; this.src='{{ url\_for('static', filename='images/default\_phone.png') }}';"              alt="{{ phone.Name }}"              class="phone-img"/>          </div>          <div class="card\_\_body">              <h2>{{ phone.Name }}</h2>              <ul class="spec-list">                  <li><span>Camera:</span> {{ phone.Camera }} MP</li>                  <li><span>Battery:</span> {{ phone.Battery }} mAh</li>                  <li><span>RAM:</span> {{ phone.RAM }} GB</li>                  <li><span>Screen:</span> {{ phone.Screen }}″</li>                  <li><span>Price:</span> ${{ phone.Price }}</li>              </ul>          </div>      </article>      {% endfor %}  </section>  <div class="back-link-wrapper">      <a class="back-link" href="{{ url\_for('index') }}#form">← Choose different preferences</a>  </div>  {% endblock %} |

**Лістинг коду user.html**

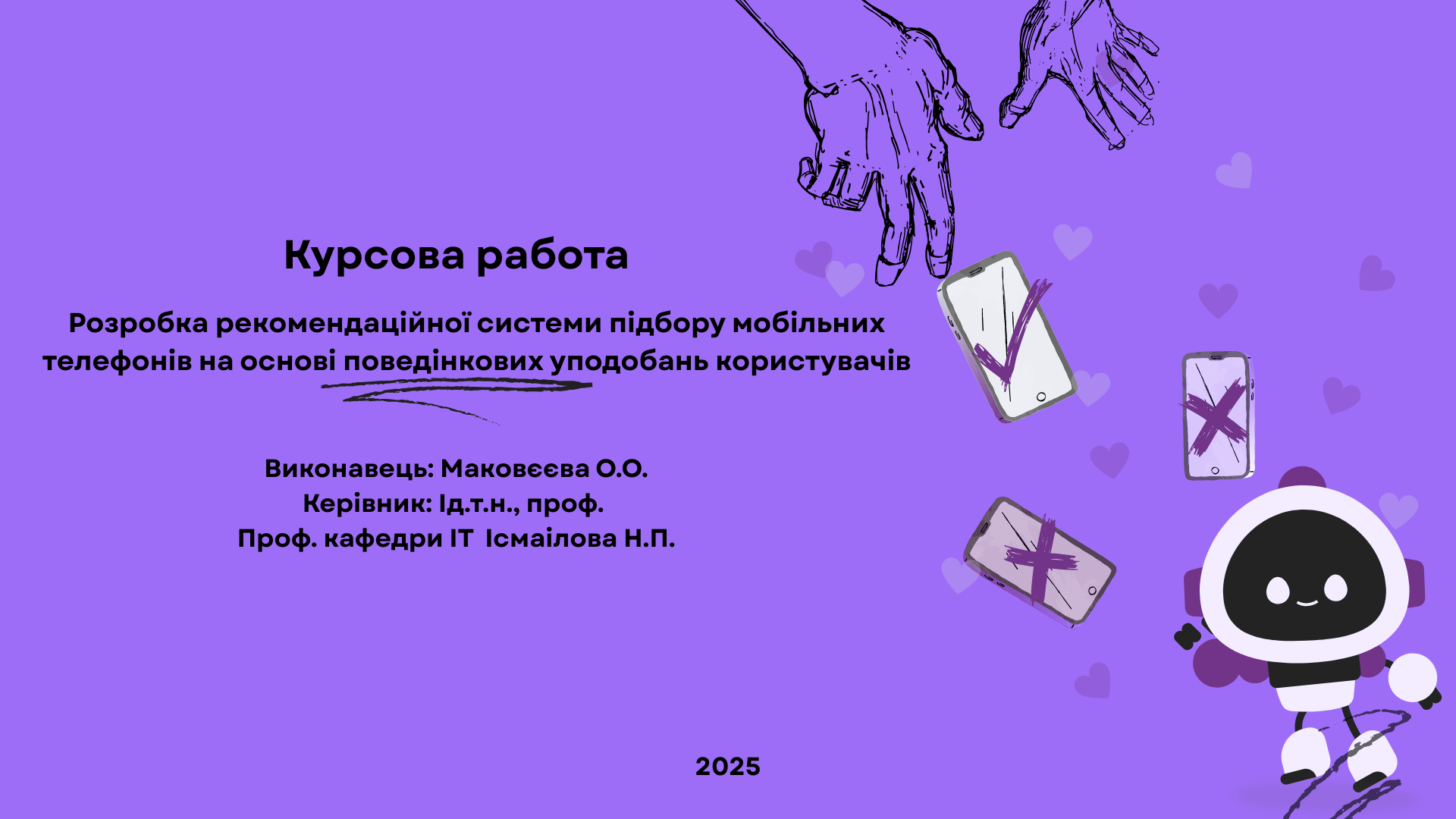
|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <section class="user-section">      <div class="card card--user">          <h1>User Page</h1>          {% if not user %}              <p class="muted">                  You are not logged in. To view your query history and personal data,                  please log in first.              </p>              <p>                  <a href="{{ url\_for('login') }}" class="btn-primary">Log In</a>              </p>          {% else %}              <p class="muted">                  You are logged in as <strong>{{ user }}</strong>.                  Below is your interaction history with the recommendation system.              </p>              <div class="user-info">                  <h2>Your Profile</h2>                  <p>In the demo version, user data is stored minimally: only login and query history.</p>                  <p>For a full project, potential functionality could include:</p>                  <ul>                      <li>editing personal data and settings;</li>                      <li>adjusting the importance of criteria (camera, price, battery, etc.);</li>                      <li>saving favorite smartphone models.</li>                  </ul>              </div>              <div class="user-history" style="margin-top: 20px;">                  <h2>Smartphone Selection History</h2>                  {% if history and history|length > 0 %}                      <table style="width: 100%; border-collapse: collapse; font-size: 0.9rem;">                          <thead>                              <tr>                                  <th style="border-bottom: 1px solid #e5e7eb; text-align: left; padding: 6px;">Time</th>                                  <th style="border-bottom: 1px solid #e5e7eb; text-align: left; padding: 6px;">Preferences</th>                                  <th style="border-bottom: 1px solid #e5e7eb; text-align: left; padding: 6px;">Recommended Models</th>                              </tr>                          </thead>                          <tbody>                              {% for item in history %}                              <tr>                                  <td style="border-bottom: 1px solid #f3f4f6; padding: 6px;">{{ item.timestamp }}</td>                                  <td style="border-bottom: 1px solid #f3f4f6; padding: 6px;">                                      {% if item.preferences %}                                          {{ item.preferences|join(', ') }}                                      {% else %}                                          (not specified)                                      {% endif %}                                  </td>                                  <td style="border-bottom: 1px solid #f3f4f6; padding: 6px;">                                      {% if item.recommended\_phones %}                                          {{ item.recommended\_phones|join(', ') }}                                      {% else %}                                          (no data)                                      {% endif %}                                  </td>                              </tr>                              {% endfor %}                          </tbody>                      </table>                  {% else %}                      <p class="muted">The history is empty. Try selecting a smartphone on the “Recommendations” page.</p>                  {% endif %}              </div>          {% endif %}      </div>  </section>  {% endblock %} |

**Лістинг коду style.css**

|  |
| --- |
| /\* Base typography and background \*/  body {      margin: 0;      font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", Arial, sans-serif;      background: #f3f4f6;      color: #111827;  }  .page {      max-width: 1100px;      margin: 80px auto 40px;      padding: 0 16px;  }  /\* Navbar \*/  .navbar {      position: fixed;      top: 0;      left: 0;      right: 0;      height: 56px;      background: linear-gradient(90deg, #2563eb, #7c3aed);      color: #f9fafb;      display: flex;      align-items: center;      justify-content: space-between;      padding: 0 24px;      box-sizing: border-box;      box-shadow: 0 2px 8px rgba(15, 23, 42, 0.3);      z-index: 100;  }  .navbar\_\_logo {      font-weight: 600;      font-size: 1.1rem;      display: flex;      align-items: center;      gap: 6px;  }  .navbar\_\_links a {      color: #e5e7eb;      text-decoration: none;      margin-left: 16px;      font-size: 0.95rem;  }  .navbar\_\_links a:hover {      color: #ffffff;      text-decoration: underline;  }  /\* Header / Hero \*/  .hero {      margin-top: 16px;      margin-bottom: 24px;  }  .hero\_\_content h1 {      font-size: 1.8rem;      margin-bottom: 8px;  }  .hero\_\_content p {      margin: 0;      color: #4b5563;  }  /\* Cards and general elements \*/  .card {      background: #ffffff;      border-radius: 12px;      padding: 20px 22px;      box-shadow: 0 10px 25px rgba(15, 23, 42, 0.08);      box-sizing: border-box;  }  .card--form {      margin-top: 8px;  }  .card--phone {      display: flex;      flex-direction: column;      transition: transform 0.15s ease, box-shadow 0.15s ease;  }  .card--phone:hover {      transform: translateY(-4px);      box-shadow: 0 16px 35px rgba(15, 23, 42, 0.18);  }  .card--user {      max-width: 800px;      margin: 0 auto;  }  /\* Form \*/  .form-section {      margin-top: 12px;  }  .form-grid {      display: grid;      grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));      gap: 16px 32px;      margin-top: 16px;  }  .form-group {      display: flex;      flex-direction: column;      gap: 8px;  }  .form-group--full {      grid-column: 1 / -1;  }  .form-group label {      font-size: 0.95rem;      color: #111827;  }  .form-group input[type="number"] {      margin-top: 4px;      padding: 6px 8px;      border-radius: 8px;      border: 1px solid #d1d5db;      font-size: 0.95rem;      box-sizing: border-box;  }  .form-group input[type="checkbox"] {      margin-right: 6px;  }  .form-actions {      margin-top: 8px;  }  /\* Button \*/  .btn-primary {      display: inline-block;      padding: 10px 20px;      background: linear-gradient(90deg, #2563eb, #7c3aed);      color: #ffffff;      border: none;      border-radius: 999px;      font-size: 0.95rem;      font-weight: 500;      cursor: pointer;      box-shadow: 0 8px 16px rgba(37, 99, 235, 0.4);  }  .btn-primary:hover {      opacity: 0.95;  }  /\* Results \*/  .results-header h1 {      font-size: 1.6rem;      margin-bottom: 8px;  }  .results-header p {      margin: 4px 0;      color: #374151;  }  .filters-box {      margin-top: 8px;      padding: 10px 12px;      border-radius: 8px;      background: #eef2ff;      border: 1px solid #c7d2fe;      font-size: 0.9rem;  }  .filters-box ul {      margin: 4px 0 0;      padding-left: 18px;  }  /\* Phone cards grid \*/  .cards-grid {      margin-top: 20px;      display: grid;      grid-template-columns: repeat(auto-fit, minmax(230px, 1fr));      gap: 18px;  }  .card\_\_image {      width: 100%;      height: 180px;      border-radius: 10px;      overflow: hidden;      background: #e5e7eb;      display: flex;      align-items: center;      justify-content: center;  }  .card\_\_image img {      width: 100%;      height: 100%;      object-fit: cover;  }  .card\_\_body h2 {      font-size: 1.1rem;      margin: 10px 0 8px;  }  .spec-list {      list-style: none;      padding: 0;      margin: 0;      font-size: 0.9rem;  }  .spec-list li {      margin: 3px 0;  }  .spec-list span {      font-weight: 500;      color: #374151;  }  /\* User page \*/  .user-section {      margin-top: 12px;  }  .user-info ul {      margin-top: 8px;      padding-left: 20px;  }  .user-info li {      margin-bottom: 4px;  }  /\* Footer \*/  .footer {      margin-top: 40px;      padding: 16px;      text-align: center;      font-size: 0.85rem;      color: #6b7280;  }  /\* Helper classes \*/  .muted {      color: #6b7280;      font-size: 0.9rem;  }  .back-link-wrapper {      margin-top: 24px;  }  .back-link {      text-decoration: none;      color: #2563eb;      font-size: 0.95rem;  }  .back-link:hover {      text-decoration: underline;  } |

# Додоток Б

**ПЕРЕЛІК КОПІЙ ДЕМОНСТРАЦІЙНОГО МАТЕРІАЛУ**

****

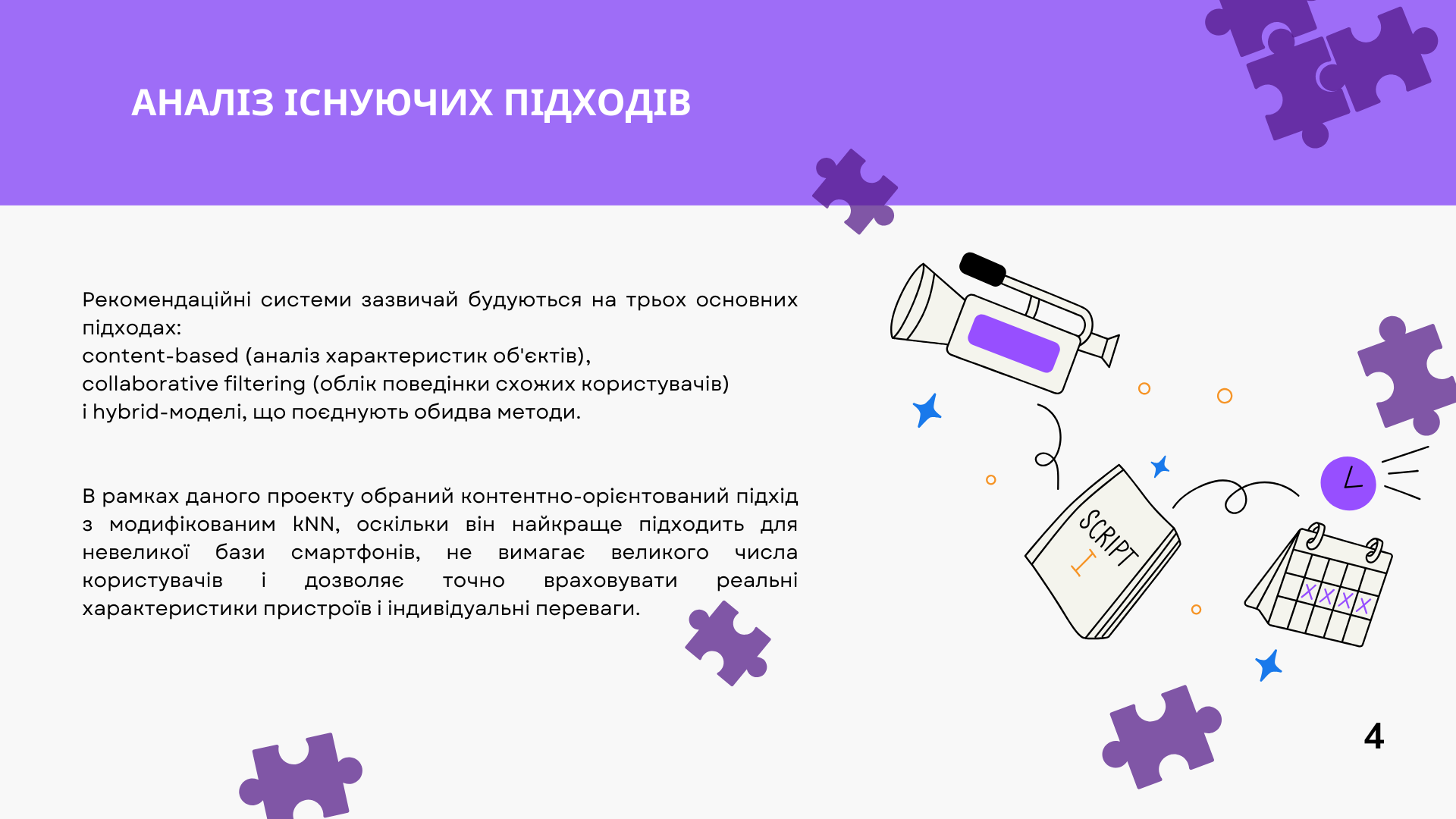
**Слайд 1** – Титульна сторінка

****

**Слайд 2** – Актуальність теми дослідження

****

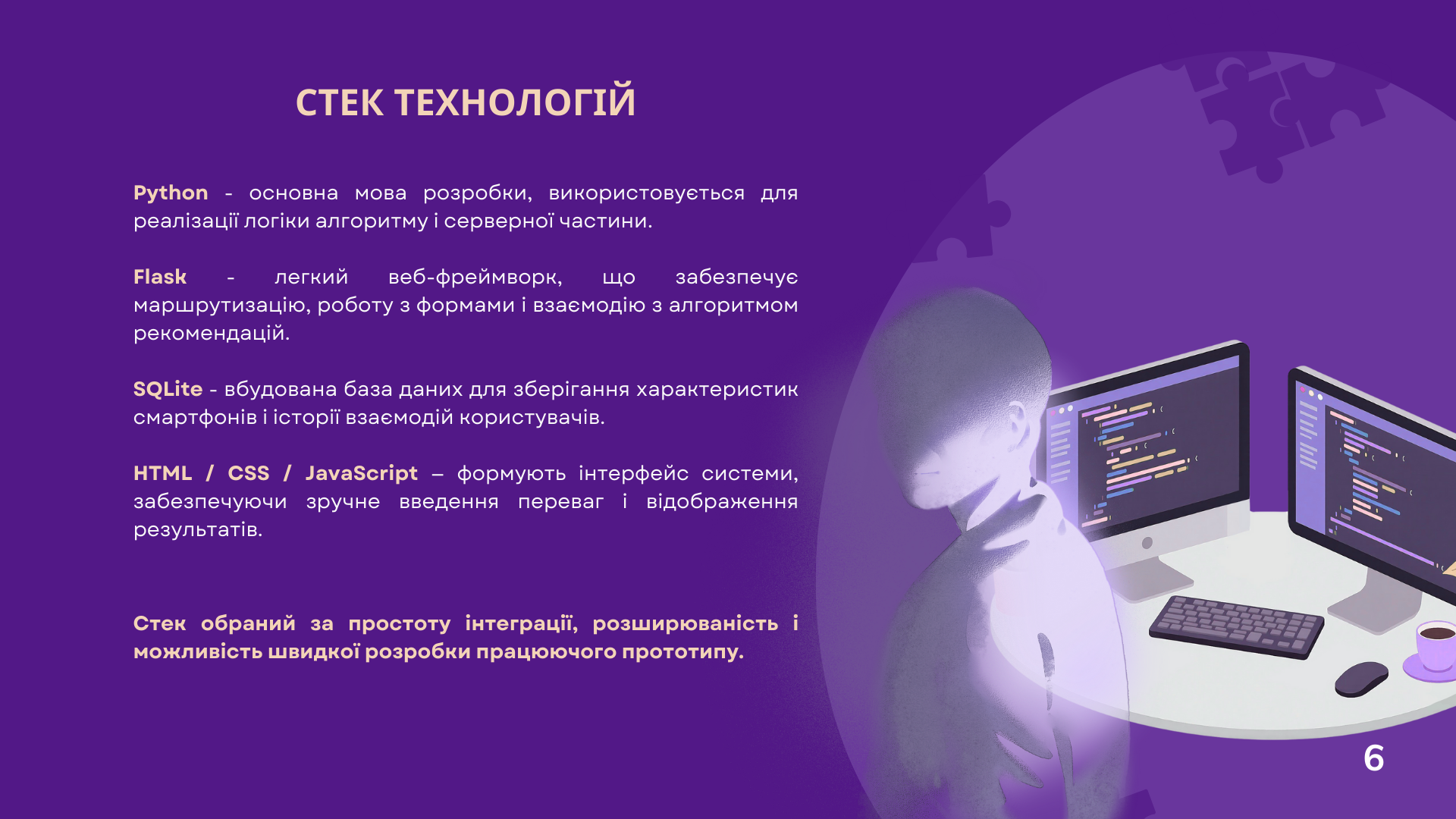
**Слайд 3 –** Мета і завдання роботи

****

**Слайд 4 –** Аналіз існуючих підходів

****

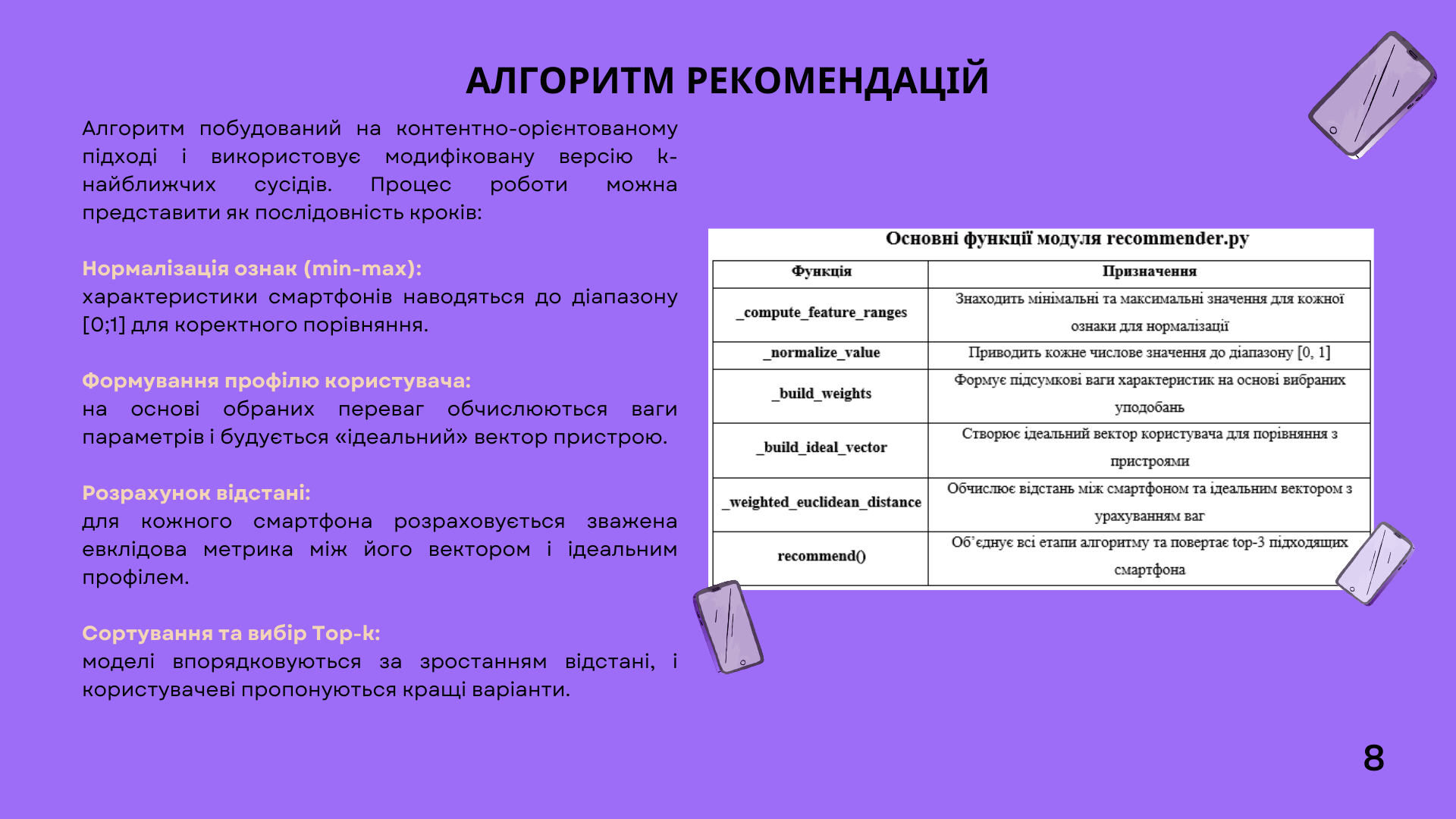
**Слайд 5 –** Архітектура системи

****

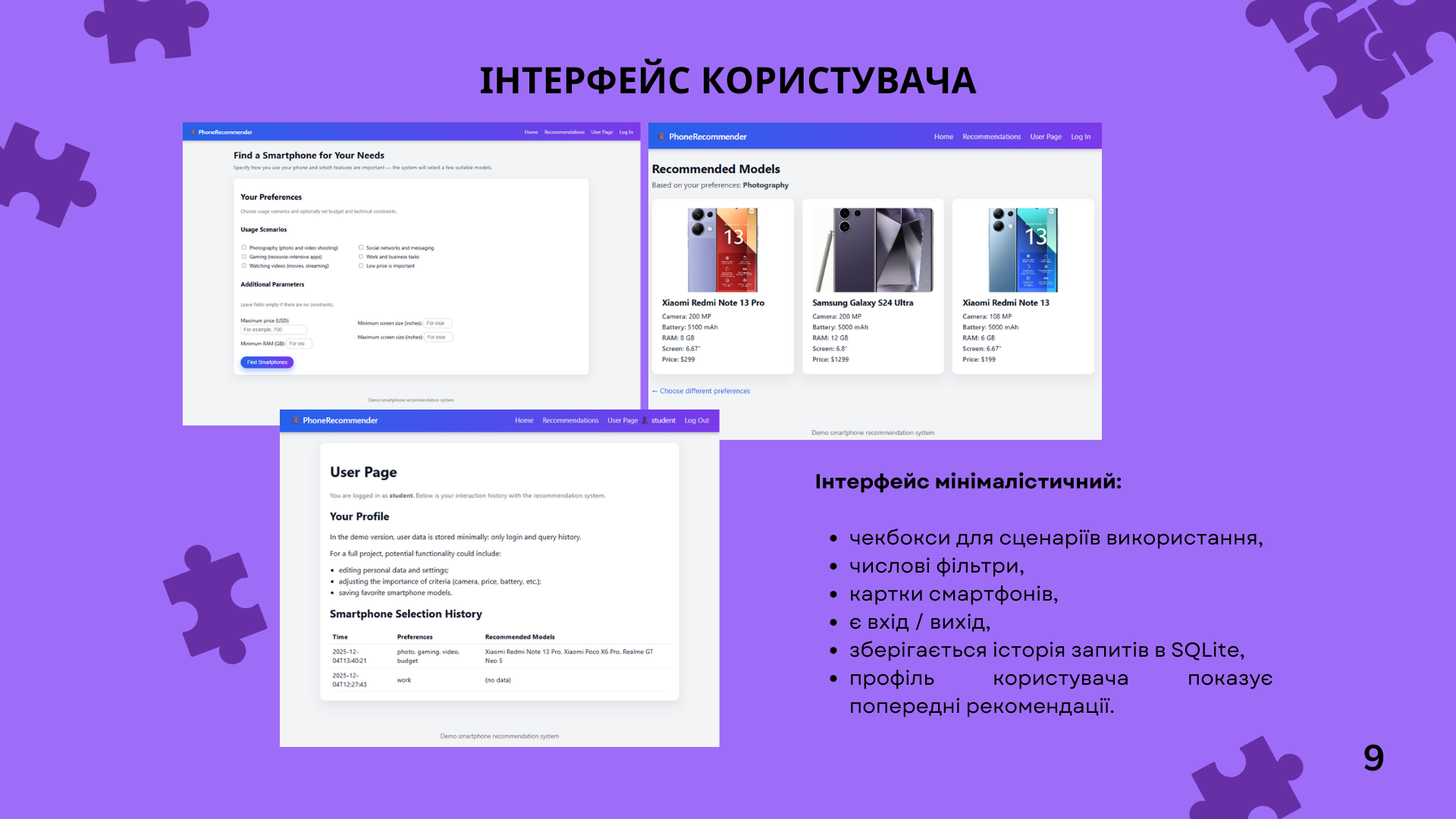
**Слайд 6 –** Стек технологій

****

**Слайд 7 –** Структура бази даних

****

**Слайд 8 –** Алгоритм рекомендацій

****

**Слайд 9 –** Інтерфейс користувача

****

**Слайд 10 –** Результати та висновки